FORTHCOMING FROM MIT Press, 2027

The Principles of Diffusion Models

Chieh-Hsin Lai
Sony AI

Yang Song
OpenAI

Dongjun Kim
Stanford University

Yuki Mitsufuji
Sony Group Corporation, Sony AI

Stefano Ermon
Stanford University

Contents

Acknowledgements **3**

A Introduction to Deep Generative Modeling **15**

1 Deep Generative Modeling **16**
1.1 What is Deep Generative Modeling? 17
1.2 Prominent Deep Generative Models 23
1.3 Taxonomy of Modeling 27
1.4 Closing Remarks 28

B Foundations and Perspectives of Diffusion Models **30**

2 Variational Perspective: From VAEs to DDPMs **37**
2.1 Variational Autoencoder 39
2.2 Variational Perspective: DDPM 49
2.3 Closing Remarks 62

3 Score-Based Perspective: From EBMs to NCSN **63**
3.1 Energy-Based Models 65
3.2 From Energy-Based to Score-Based Generative Models 71
3.3 Denoising Score Matching 75
3.4 Multi-Noise Levels of Denoising Score Matching (NCSN) 86
3.5 Summary: A Comparative View of NCSN and DDPM 91
3.6 Closing Remarks 92

4 Diffusion Models Today: Score SDE Framework **93**
4.1 Score SDE: Its Principles . . . 95
4.2 Matching Marginals in Forward/Reverse-Time SDEs and PF-ODE . . 105
4.3 Score SDE: Its Training and Sampling . . . 113
4.4 Instantiations of SDEs . . . 119
4.5 A Direct View Through Accumulated Perturbation Kernels . . . 123
4.6 (Optional) Fokker–Planck Equation and Reverse-Time SDEs via Marginalization and Bayes' Rule . . . 130
4.7 Closing Remarks . . . 134

5 Flow-Based Perspective: From NFs to Flow Matching **135**
5.1 Flow-Based Models: Normalizing Flows and Neural ODEs . . . 138
5.2 Flow Matching Framework . . . 147
5.3 Constructing Probability Paths and Velocities Between Distributions . 161
5.4 (Optional) Properties of the Canonical Affine Flow . . . 176
5.5 Closing Remarks . . . 182

6 A Unified and Systematic Lens on Diffusion Models **183**
6.1 Conditional Tricks: The Secret Sauce of Diffusion Models . . . 185
6.2 A Roadmap for Elucidating Training Losses in Diffusion Models . . . 190
6.3 Equivalence in Diffusion Models . . . 197
6.4 Beneath It All: The Fokker–Planck Equation . . . 209
6.5 Closing Remarks . . . 213

7 (Optional) Diffusion Models and Optimal Transport **215**
7.1 Prologue of Distribution-to-Distribution Translation . . . 217
7.2 Taxonomy of the Problem Setups . . . 219
7.3 Relationship of Variant Optimal Transport Formulations . . . 230
7.4 Is Diffusion Model's SDE Optimal Solution to SB Problem? . . . 237
7.5 Is Diffusion Model's ODE an Optimal Map to OT Problem? . . . 241
7.6 Closing Remarks . . . 249

C Sampling of Diffusion Models **250**

8 Guidance and Controllable Generation **252**
8.1 Prologue . . . 254
8.2 Classifier Guidance . . . 258
8.3 Classifier-Free Guidance . . . 261
8.4 Training-Free Guidance . . . 264
8.5 From Reinforcement Learning to Direct Preference Optimization for Model Alignment . . . 273

8.6 Closing Remarks . . . 283

9 Numerical Solvers for Fast Diffusion Sampling 284
9.1 Prologue . . . 286
9.2 DDIM . . . 296
9.3 DEIS . . . 313
9.4 DPM-Solver . . . 320
9.5 DPM-Solver++ . . . 333
9.6 (Optional) ParaDiGMs . . . 343
9.7 Closing Remarks . . . 349

D Toward Learning Fast Diffusion-Based Generators 350

10 Distillation-Based Methods for Fast Sampling 351
10.1 Prologue . . . 352
10.2 Distribution-Based Distillation . . . 356
10.3 Progressive Distillation . . . 360
10.4 Closing Remarks . . . 366

11 Learning Fast Generators from Scratch 367
11.1 Prologue . . . 369
11.2 Special Flow Map: Consistency Model in Discrete Time . . . 374
11.3 Special Flow Map: Consistency Model in Continuous Time . . . 381
11.4 General Flow Map: Consistency Trajectory Model . . . 389
11.5 General Flow Map: Mean Flow . . . 399
11.6 Closing Remarks . . . 404

Epilogue and Outlook 410

12 A Unifying Principle and the Road Ahead 410
12.1 The Density-Transport Backbone . . . 412
12.2 Beyond Continuous States: Probability Evolution on Discrete Spaces . . . 416
12.3 Closing Remarks of the Book . . . 435

Appendices 436

A Crash Course on Differential Equations 437
A.1 Foundation of Ordinary Differential Equations . . . 439
A.2 Foundation of Stochastic Differential Equations . . . 450

B Density Evolution: From Change of Variable to Fokker–Planck **454**
B.1 Change-of-Variable Formula: From Deterministic Maps to Stochastic Flows . . . 456
B.2 Intuition of the Continuity Equation . . . 465
B.3 (Optional) Wasserstein Gradient Flows as Distribution-Level Training . 467

C Behind the Scenes of Diffusion Models: Itô's Calculus and Girsanov's Theorem **473**
C.1 Itô's Formula: The Chain Rule for Random Processes . . . 475
C.2 Change-of-Variable For Measures: Girsanov's Theorem in Diffusion Models . . . 485

D Supplementary Materials and Proofs **489**
D.1 Variational Perspective . . . 489
D.2 Score-Based Perspective . . . 492
D.3 Flow-Based Perspective . . . 503
D.4 Theoretical Supplement: A Unified and Systematic View on Diffusion Models . . . 506
D.5 Theoretical Supplement: Learning Fast Diffusion-Based Generators . . 507
D.6 (Optional) Elucidating Diffusion Model (EDM) . . . 511

E Miscellaneous Proofs **515**
E.1 Two Perspectives on the Divergence Theorem . . . 515
E.2 Multivariate Integration by Parts . . . 516
E.3 Jensen's Inequality . . . 517

References **519**

The Principles of Diffusion Models

Chieh-Hsin Lai[1], Yang Song[2], Dongjun Kim[3], Yuki Mitsufuji[4] and Stefano Ermon[5]

[1] *Sony AI; chieh-hsin.lai@sony.com / chiehhsinlai@gmail.com*
[2] *OpenAI*; thusongyang@gmail.com*
[3] *Stanford University; dongjun@stanford.edu*
[4] *Sony Group Corporation, Sony AI; yuhki.mitsufuji@sony.com*
[5] *Stanford University; ermon@cs.stanford.edu*

ABSTRACT

This book focuses on the principles that have shaped the development of diffusion models, tracing their origins and showing how different formulations arise from common mathematical ideas.

Diffusion modeling begins by specifying a *forward corruption process* that gradually turns data into noise. This forward process links the data distribution to a simple noise distribution by defining a continuous family of intermediate distributions. The core objective of a diffusion model is to construct another process that runs in the opposite direction, transforming noise into data while recovering the same intermediate distributions defined by the forward corruption process.

We describe three complementary ways to formalize this idea. The *variational view*, inspired by variational autoencoders, sees diffusion as learning to remove noise step by step, solving small denoising objectives that together teach the model to turn noise back into data. The *score-based view*, rooted in energy-based modeling, learns the gradient of the evolving data distribution, which indicates how to nudge samples toward more likely regions. The *flow-based view*, related to normalizing flows, treats generation as following a smooth path that moves samples from noise to data under a learned velocity field.

These perspectives share a common backbone: a learned time-dependent velocity field whose flow transports a simple prior to the data. With this in hand, sampling amounts to solving a differential equation that evolves noise into data along a continuous generative trajectory. On this foundation, the book discusses *guidance* for controllable generation, *advanced numerical solvers* for efficient sampling, and diffusion-motivated

*Affiliation reflects the institution at the time of the work.

flow-map models that learn direct mappings between arbitrary times along this trajectory.

This book is intended for readers with a basic background in deep learning who seek a clear, conceptual, and mathematically grounded understanding of diffusion models. It develops the core principles underlying the subject, explains the ideas that unify its many formulations, and provides a solid foundation for further study in this rapidly evolving area. As such, it serves both as a principled reference for researchers and as an accessible entry point for students and newcomers. Supplementary materials for the book are available at the book website:

https://the-principles-of-diffusion-models.github.io/

Acknowledgements

We would like to express our sincere gratitude to the many members of the community who have offered valuable feedback and helped us identify errata. In particular, we thank the following individuals, listed in alphabetical order: *Francis Bach*, *Ilia Badanin*, *Rwiddhi Chakraborty*, *Mauricio Delbracio*, *Jacob Lessing*, *Kaiming He*, *Yutong (Kelly) He*, *Durk Kingma*, *Shucheng Li*, *Ramtin Moslemi*, *Rukmangadh Sai Myana*, *Stefano Peluchetti*, *Yuri Plotkin*, *François Rozet*, *Yair Shenfeld*, *Molei Tao*, *Mohamad Ternanni*, and *Baojian Zhou*, as well as anonymous reviewers and other readers who kindly shared feedback.

The authors would also like to express their deep gratitude to Professor *Dohyun Kwon* from the University of Seoul and KIAS for his generous time and effort in engaging with this work. He carefully reviewed parts of Chapter 7, helping to ensure the correctness of statements and proofs, and contributed to several valuable discussions that clarified the presentation. Beyond his technical suggestions, his thoughtful feedback and willingness to share his perspectives have been a source of encouragement throughout the writing of this book. We sincerely appreciate his support and collegial spirit, which have enriched the final version.

Preface and Roadmap

Diffusion models have rapidly become a central paradigm in generative modeling, with a vast body of work spanning machine learning, computer vision, natural language processing, and beyond. This literature is dispersed across communities and highlights different dimensions of progress, including theoretical foundations that concern modeling principles, training objectives, sampler design, and the mathematical ideas behind them; implementation advances that cover engineering practices and architectural choices; practical applications that adapt the models to specific domains or tasks; and system level optimizations that improve efficiency in computation, memory, and deployment.

This book sets out to provide a *principled foundation* of diffusion models, focusing on the following central themes:

- We present the essential concepts and formulations that anchor diffusion model research, giving readers the core understanding needed to navigate the broader literature. We do not survey all variants or domain specific applications; instead we establish a stable conceptual foundation from which such developments can be understood.

- Unlike classical generative models that learn a direct mapping from noise to data, diffusion models view generation as a gradual transformation over time, refining coarse structures into fine details. This central idea has been developed through three main perspectives, i.e., *variational*, *score-based*, and *flow-based* methods, which offer complementary ways to understand and implement diffusion modeling. We focus on the core principles and foundations of these formulations, aiming to trace the origins of their key ideas, clarify the relations among different formulations, and develop a coherent understanding that connects intuitive insight with rigorous mathematical formulation.

- Building on these foundations, we examine how diffusion models can be further developed to generate samples more efficiently, provide greater control over the generative process, and inspire standalone forms of generative modeling grounded in the principles of diffusion.

This book is intended for researchers, graduate students, and practitioners who have a basic understanding of deep learning (for example, what a neural network is and how training works), or more specifically, deep generative modeling, and who wish to deepen their grasp of diffusion models beyond surface-level familiarity. By the end, readers will have a principled understanding of the foundations of diffusion modeling, the ability to interpret different formulations within a coherent framework, and the background needed to both apply existing models with confidence and pursue new research directions.

Roadmap of This Book

This book systematically introduces the foundations of diffusion models, tracing them back to their core underlying principles.

Suggested Reading Path. We recommend reading this book in the presented order to build a comprehensive understanding. Sections marked as *Optional* can be skipped by readers already familiar with the fundamentals. For instance, those comfortable with deep generative models (DGM) may bypass the overview in Chapter 1. Similarly, prior knowledge of Variational Autoencoders (Section 2.1), Energy-Based Models (Section 3.1), or Normalizing Flows (Section 5.1) allows skipping these introductory sections. Other optional parts provide deeper insights into advanced or specialized topics and can be consulted as needed.

The book is organized into four main parts.

Parts A & B: Foundations of Diffusion Models. This section traces the origins of diffusion models by reviewing three foundational perspectives that have shaped the field[1]. Figure 3 provides an overview of this part.

[1] We remark that our organization of the three perspectives on diffusion models is conceptual rather than chronological. For instance, *energy-based approaches* draw on a much longer history. Statistical mechanics connected probability with energy, while studies of Brownian motion and subsequent work described how probability distributions evolve over time (Gibbs, 1902; Langevin, 1908; Fokker, 1914; Planck, 1917; Kolmogoroff, 1931). Modern diffusion probabilistic models later brought related ideas into generative modeling and explicitly drew inspiration from nonequilibrium statistical physics (Sohl-Dickstein *et al.*, 2015). The three perspectives presented here should therefore be understood as modern ways of explaining diffusion models, rather than as a strict historical ordering of where their underlying ideas first arose.

Part A: Introduction to Deep Generative Modeling (DGM). We begin in Chapter 1 with a review of the fundamental goals of deep generative modeling. Starting from a collection of data examples, the aim is to build a model that can produce new examples that appear to come from the same underlying, and generally unknown, data distribution. Many approaches achieve this by learning how the data are distributed, either explicitly through a probability model or implicitly through a learned transformation. We then explain how such models represent the data distribution with neural networks, how they learn from examples, and how they generate new samples. The chapter concludes with a taxonomy of major generative frameworks, highlighting their central ideas and key distinctions.

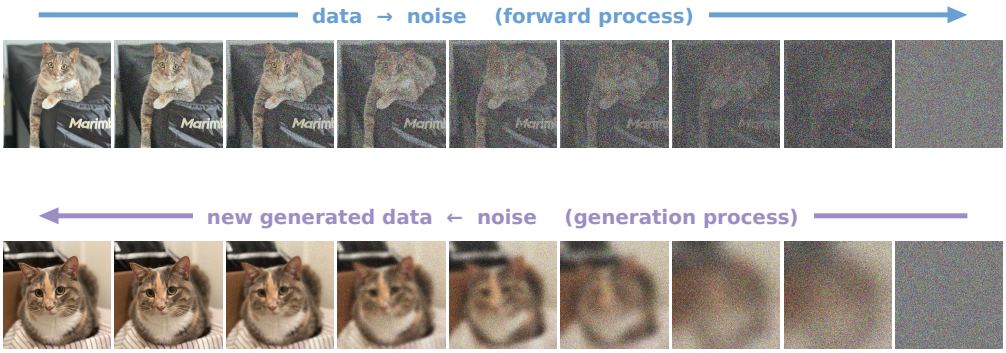


Figure 1: Forward and reverse diffusion trajectories. In the forward process (top), a real data sample is gradually corrupted by noise, passing through a sequence of intermediate noisy stages until it becomes indistinguishable from pure noise. In the reverse process (bottom), generation starts from a fresh noise sample and gradually removes noise to produce a new image. The reverse process is not about reconstructing a specific example seen during training. Instead, it begins from a fresh noise and gradually shapes it into a realistic new sample.

Source: Created by the authors.

Part B: Core Perspectives on Diffusion Models. Having outlined the general goals and mechanisms of deep generative modeling, we now turn to diffusion models, a class of methods that realize generation as a gradual transformation from noise to data (see Figure 1). The core idea is simple: a data sample can be progressively corrupted by adding noise until it becomes indistinguishable from pure randomness. This *forward process* moves different data samples into the same simple noise space, which is easy to sample from and serves as the starting point for generation. Along the way, it also creates a smooth sequence of intermediate noisy stages connecting data to noise. To generate, we draw a fresh noise sample from this space and gradually remove noise through a *reverse-time (generation) process*, eventually producing a new data sample. This reverse process does not recover one particular original example; instead, it starts from fresh noise and gradually turns it into realistic data. We examine three interconnected frameworks,

each built on this same principle: a forward process that gradually adds noise, and a reverse-time process approximated by a sequence of models that gradually denoise:

- **Variational View** (Chapter 2): Originating from Variational Autoencoders (VAEs) (Kingma and Welling, 2013), it frames diffusion as learning a denoising process through a variational objective, giving rise to Denoising Diffusion Probabilistic Models (DDPMs) (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020).

- **Score-Based View** (Chapter 3): Rooted in Energy-Based Models (EBMs) (Ackley *et al.*, 1985) and developed into Noise Conditional Score Networks (NCSN) (Song and Ermon, 2019). It learns the score function, the gradient of the log data density, which guides how to gradually remove noise from samples. In continuous time, Chapter 4 introduces the *Score SDE framework*, which describes this denoising process as a Stochastic Differential Equation (SDE) and its deterministic counterpart as an Ordinary Differential Equation (ODE). This view connects diffusion modeling with classical differential equation theory, providing a clear mathematical basis for analysis and algorithm design.

- **Flow-Based View** (Chapter 5): Building on Normalizing Flows (Rezende and Mohamed, 2015) and generalized by Flow Matching (Lipman *et al.*, 2022), this view models generation as a continuous transformation that transports samples from a simple prior toward the data distribution. The evolution is governed by a velocity field through an ODE, which explicitly defines how probability mass moves over time. This flow-based formulation naturally extends beyond prior-to-data generation to more general *distribution-to-distribution translation* problems, where one seeks to learn a flow connecting any pair of source and target distributions.

Although these perspectives may seem different at first, Chapter 6 shows that they are deeply connected. Each uses a *conditioning strategy* that turns the learning objective into a tractable regression problem. At a deeper level, they all describe the same temporal evolution of probability distributions, from the prior toward the data. This evolution is governed by the *Fokker–Planck equation*, which can be viewed as the continuous-time change of variables for densities, ensuring consistency between the stochastic and deterministic formulations.

Since diffusion models can be viewed as approaches for transporting one distribution to another, Chapter 7 develops their connections to classical optimal transport and the Schrödinger bridge, interpreted as optimal transport with entropy regularization. We review both the static and dynamic formulations and explain their relations to the continuity equation and the Fokker–Planck perspective. This

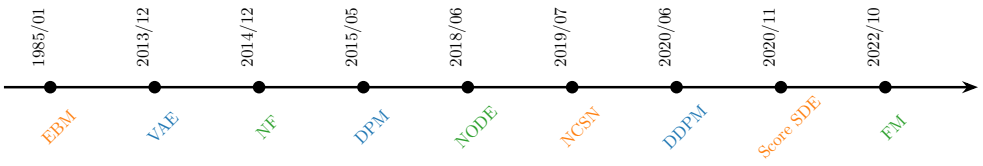


Figure 2: Timeline of modern machine-learning perspectives on diffusion models. Each group shares the same color.
▪ In Chapter 2, Variational Autoencoder (VAE) (Kingma and Welling, 2013) → Diffusion Probabilistic Models (DPM) (Sohl-Dickstein *et al.*, 2015) → DDPM (Ho *et al.*, 2020).
▪ In Chapters 3 and 4, Energy-Based Model (EBM) (Ackley *et al.*, 1985) → Noise Conditional Score Network (NCSN) (Song and Ermon, 2019) → Score SDE (Song *et al.*, 2020c).
▪ In Chapter 5, Normalizing Flow (NF) (Rezende and Mohamed, 2015) → Neural ODE (NODE) (Chen *et al.*, 2018) → Flow Matching (FM) (Lipman *et al.*, 2022).

Source: Created by the authors.

chapter is optional for readers focused on practical aspects, but it provides rigorous mathematical background and pointers to the classical literature for those who wish to study these links in depth.

Part C & D: Controlling and Accelerating the Diffusion Sampling. With the foundational principles unified, we now turn to practical aspects of utilizing diffusion models for efficient generation. Sampling from a diffusion model corresponds to solving a differential equation. However, this procedure is typically computationally expensive. Parts C and D focus on improving generation quality, controllability, and efficiency through enhanced sampling and learned acceleration techniques.

Part C: Sampling from Diffusion Models. The generation process of diffusion models exhibits a distinctive coarse-to-fine refinement: noise is removed step by step, yielding samples with increasingly coherent structure and detail. This property comes with trade-offs. On the positive side, it affords fine-grained control; by adding a guidance term to the learned, time-dependent velocity field, we can steer the ODE flow to reflect user intent and make sampling controllable. On the negative side, the required iterative integration makes sampling slow compared with single-shot generators. This part focuses on improving the generative process at inference time, without retraining.

- **Steering Generation** (Chapter 8)**:** Techniques such as classifier guidance and classifier-free guidance make it possible to condition the generation process on user-defined objectives or attributes. Building on this, we next discuss how the use of a preference dataset can further align diffusion models with such preferences.

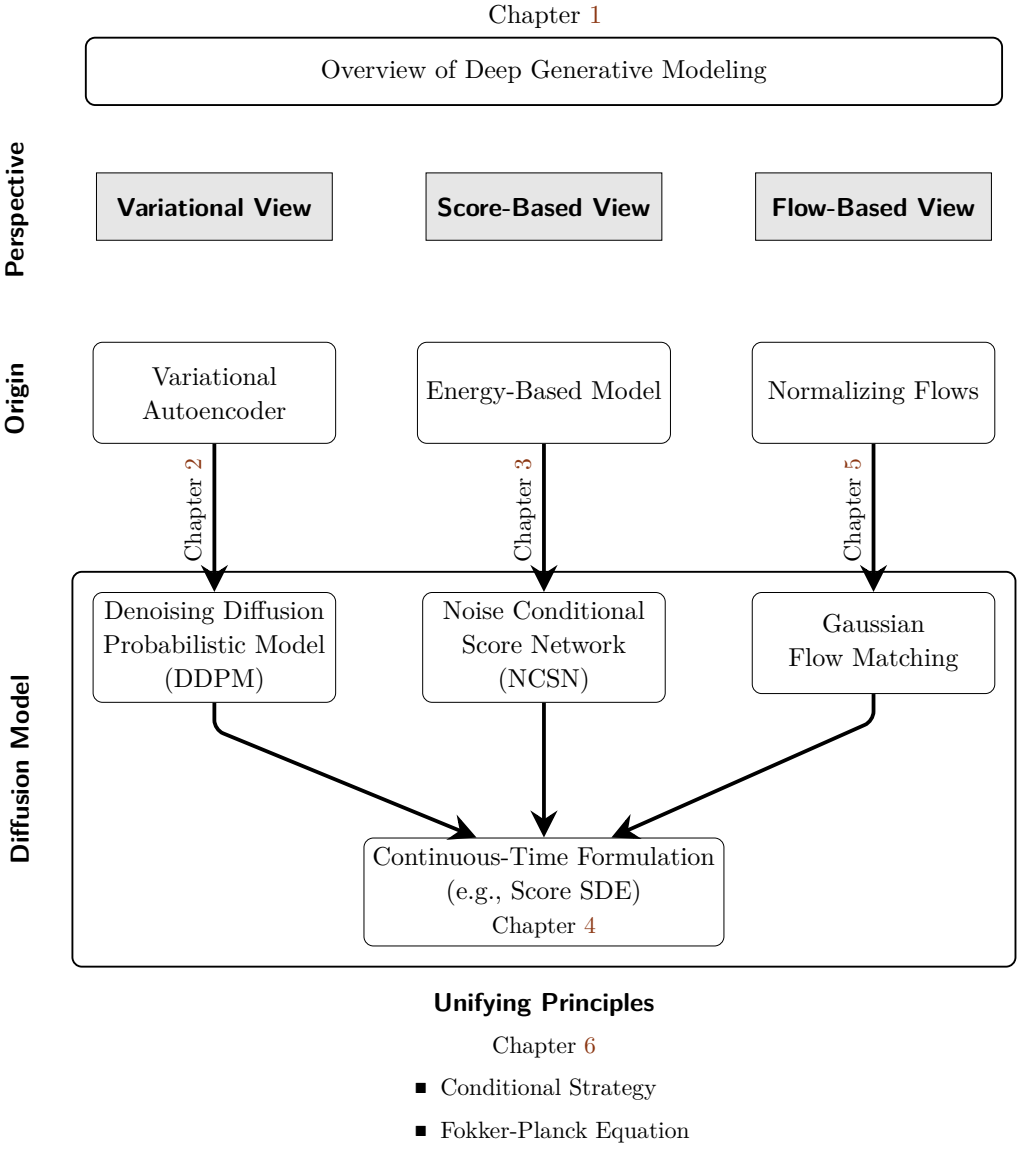


Figure 3: Part B. Unifying and Principled Perspectives on Diffusion Models. This diagram visually connects classical generative modeling approaches—Variational Autoencoders, Energy-Based Models, and Normalizing Flows—with their corresponding diffusion model formulations. Each vertical path illustrates a conceptual lineage, culminating in the continuous-time framework. The three views (Variational, Score-Based, and Flow-Based) offer distinct yet mathematically equivalent interpretations.

Source: Created by the authors.

- **Fast Generation with Numerical Solvers** (Chapter 9)**:** Sampling can be significantly accelerated using advanced numerical solvers that approximate

the reverse process in fewer steps, reducing cost while preserving quality.

Part D: Learning Fast Generative Models. Beyond improving existing sampling algorithms, we investigate how to directly learn fast generators that approximate the diffusion process.

- **Distillation-Based Methods** (Chapter 10): This approach focuses on training a student model to imitate the behavior of a pre-trained, slow diffusion model (the teacher). Instead of reducing the teacher's size, the goal is to reproduce its sampling trajectory or output distribution with far fewer integration steps, often only a few or even one.

- **Learning from Scratch** (Chapter 11)**:** Since sampling in diffusion models can be seen as solving an ODE, this approach learns the solution map (i.e., the flow map) directly from scratch, without relying on a teacher model. The learned map can take noise directly to data, or more generally perform anytime-to-anytime jumps along the solution trajectory.

Epilogue: Beyond Diffusion on Continuous State Spaces. We close in Chapter 12 by making explicit the principle that has guided the entire book. At its heart, diffusion modeling is not tied to one particular noise type, model class, or even to continuous state spaces. Rather, it is the study of how probability mass evolves under a prescribed time-dependent forward corruption process, and how that evolution can be reversed, approximated, or exploited for generation. In the continuous setting, this principle appears through the change-of-variable formula, the continuity equation, and the Fokker–Planck equation. In the discrete setting, the same structural role is played by transition kernels, continuous-time Markov chains, and the master equation.

This final chapter also shows that the three perspectives developed throughout the book, namely the variational, score-based, and flow-based viewpoints, extend beyond continuous data modeling. Even for discrete data such as text, protein sequences, and other token-based objects, one can formulate reverse modeling through variational objectives, score-like ratio or reverse-rate characterizations, and flow-like transport viewpoints. The mathematical objects change, but the underlying principles do not. In this way, the epilogue serves both as a conceptual culmination of the book and as an outlook toward a broader landscape of generative modeling.

Appendices. To ensure our journey is accessible to all, the appendices provide background for foundational concepts. Chapter A offers a crash course on the differential equations that have become the language of diffusion models.

The core insight behind diffusion models, despite their varied perspectives and origins, lies in the *change-of-variables formula*. This foundation naturally extends to deeper concepts such as the *Fokker–Planck equation* and the *continuity equation*, which describe how probability densities transform and evolve under mappings defined by functions (discrete time) or differential equations (continuous time). Chapter B offers a gentle introduction that bridges these foundational ideas to more advanced concepts. In Chapter C, we present two powerful but often overlooked tools underlying diffusion models: *Itô's formula* and *Girsanov's theorem*, which provide rigorous support for the Fokker–Planck equation and the reverse-time sampling process. Finally, Chapter D gathers proofs of selected propositions and theorems discussed in the main chapters.

What This Book Covers and What It Does Not. We aim for durability. From a top-down viewpoint, this book begins with a single principle: construct continuous-time dynamics that transport a simple prior to the data distribution while ensuring that the marginal distribution at each time matches the marginal induced by a prescribed forward process from data to noise. From this principle, we develop the stochastic and deterministic flows that enable sampling, show how to steer the trajectory (guidance), and explain how to accelerate it (numerical solvers). We then study diffusion-motivated fast generators, including distillation methods and flow-map models. With these tools, readers can place new papers within a common template, understand why methods work, and design improved models.

We do not attempt to provide an exhaustive survey of the diffusion model literature, nor do we catalog architectures, training practices, hyperparameters, compare empirical results across methods, cover datasets and leaderboards, describe domain- or modality-specific applications, address system-level deployment, provide recipes for large-scale training, or discuss hardware engineering. These topics evolve rapidly and are better covered by focused surveys, open repositories, and implementation guides.

Notations

Numbers and Arrays

a	A scalar.
$\mathbf{a}$	A column vector (e.g., $\mathbf{a} \in \mathbb{R}^D$).
$\mathbf{A}$	A matrix (e.g., $\mathbf{A} \in \mathbb{R}^{m \times n}$).
$\mathbf{A}^\top$	Transpose of $\mathbf{A}$.
$\mathrm{Tr}(\mathbf{A})$	Trace of $\mathbf{A}$.
$\mathbf{I}_D$	Identity matrix of size $D \times D$.
$\mathbf{I}$	Identity matrix; dimension implied by context.
$\mathrm{diag}(\mathbf{a})$	Diagonal matrix with diagonal entries given by $\mathbf{a}$.
$\boldsymbol{\phi}$, $\boldsymbol{\theta}$	Learnable neural network parameters.
$\boldsymbol{\phi}^\times$, $\boldsymbol{\theta}^\times$	Parameters after training (fixed during inference).
$\boldsymbol{\phi}^*$, $\boldsymbol{\theta}^*$	Optimal parameters of an optimization problem.

Calculus

$\dfrac{\partial \mathbf{y}}{\partial \mathbf{x}}$	Partial derivatives of $\mathbf{y}$ w.r.t. $\mathbf{x}$ (componentwise).
$\dfrac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}$ or $\mathrm{D}\mathbf{y}(\mathbf{x})$	Total (Fréchet) derivative of $\mathbf{y}$ w.r.t. $\mathbf{x}$.
$\nabla_{\mathbf{x}} y$	Gradient of scalar $y : \mathbb{R}^D \to \mathbb{R}$; a column in $\mathbb{R}^D$.
$\dfrac{\partial \mathbf{F}}{\partial \mathbf{x}}$ or $\nabla_{\mathbf{x}} \mathbf{F}$	Jacobian of $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$; shape $m \times n$.
$\nabla \cdot \mathbf{y}$	Divergence of a vector field $\mathbf{y} : \mathbb{R}^D \to \mathbb{R}^D$; a scalar.
$\nabla^2_{\mathbf{x}} f(\mathbf{x})$ or $\mathbf{H}(f)(\mathbf{x})$	Hessian of $f : \mathbb{R}^D \to \mathbb{R}$; shape $D \times D$.
$\displaystyle\int f(\mathbf{x})\,\mathrm{d}\mathbf{x}$	Integral of f over the domain of $\mathbf{x}$.

Probability and Information Theory

$p(\mathbf{x})$	Density/distribution over a continuous vector $\mathbf{x}$.
$\Pr(\mathbf{x} = \mathbf{a})$, or informally $\Pr(\mathbf{x})$	Point probability in the discrete setting.
$\Pr(\mathbf{x} = \mathbf{a} \mid \mathbf{y} = \mathbf{b})$, or informally $\Pr(\mathbf{x} \mid \mathbf{y})$	Conditional point probability in the discrete setting.
p_{data}	Data distribution.
p_{prior}	Prior distribution (e.g., standard normal).
p_{src}	Source distribution.
p_{tgt}	Target distribution.
$\mathbf{a} \sim p$	Random vector $\mathbf{a}$ is distributed as p.
$\mathbb{E}_{\mathbf{x}\sim p}[\mathbf{f}(\mathbf{x})]$	Expectation of $\mathbf{f}(\mathbf{x})$ under $p(\mathbf{x})$.
$\mathbb{E}[\mathbf{f}(\mathbf{x}) \mid \mathbf{z}]$, or $\mathbb{E}_{\mathbf{x}\sim p(\cdot\mid\mathbf{z})}[\mathbf{f}(\mathbf{x})]$	Conditional expectation of $\mathbf{f}(\mathbf{x})$ given $\mathbf{z}$, with $\mathbf{x}$ distributed as $p(\cdot\mid\mathbf{z})$.
$\mathrm{Var}(\mathbf{f}(\mathbf{x}))$	Variance under $p(\mathbf{x})$.
$\mathrm{Cov}(\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x}))$	Covariance under $p(\mathbf{x})$.
$\mathcal{D}_{\mathrm{KL}}(p \,\|\, q)$	Kullback–Leibler divergence from q to p.
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$	Standard normal sample.
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian over $\mathbf{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

Notational Conventions. We collect several conventions used throughout this book.

I. Random Variables and Their Realizations. We use the same symbol for a random vector and its realized value. This convention, common in deep learning and generative modeling, keeps notation compact. The intended meaning is determined by context.

For densities, $p(\mathbf{x})$ denotes the functional form of the distribution rather than evaluation at a particular sample. When evaluation at a specific point is intended, we state so explicitly. Conditional densities are read by context: in $p(\mathbf{x}|\mathbf{y})$, fixing $\mathbf{y}$ gives a density in $\mathbf{x}$, while fixing $\mathbf{x}$ gives a function of $\mathbf{y}$.

II. Conditional Expectations. The expression $\mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{z}]$ denotes a function of $\mathbf{z}$, giving the expected value of $\mathbf{f}(\mathbf{x})$ conditional on $\mathbf{z}$. When conditioning on a specific realized value, we write $\mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{Z} = \mathbf{z}]$. Equivalently, this can be expressed as an integral with respect to the conditional distribution:

$$\mathbb{E}_{\mathbf{x}\sim p(\cdot|\mathbf{z})}[\mathbf{f}(\mathbf{x})] = \int \mathbf{f}(\mathbf{x})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{x}.$$

This distinction clarifies whether $\mathbf{z}$ is treated as a variable defining a function $\mathbf{z} \mapsto \mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{z}]$, or as a fixed value at which that function is evaluated.

III. Implicit Time Indices. Since many objects in this book are indexed by time, writing every time variable explicitly can make formulas heavy. When the relevant time indices are clear from context, we leave them implicit. For example,

$$\mathbb{E}[\mathbf{x}_s|\mathbf{x}_t] \quad \text{is shorthand for} \quad \mathbb{E}[\mathbf{x}_s|\mathbf{x}_t,\, s,\, t],$$

meaning the average of $\mathbf{x}_s$ at time s given $\mathbf{x}_t$ at time t, with s and t understood from the surrounding discussion. Similarly, $p(\mathbf{x}_s|\mathbf{x}_t)$ denotes the conditional distribution of $\mathbf{x}_s$ given $\mathbf{x}_t$, with both time indices left implicit.

IV. Gaussian Perturbation and Equality in Distribution. For a fixed index t, we frequently use the Gaussian perturbation kernel

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_t;\, \alpha_t\mathbf{x}_0,\, \sigma_t^2\mathbf{I}\big),$$

where $\mathbf{x}_0 \sim p_{\text{data}}$, α_t and σ_t are deterministic scalars. We equivalently write this as the identity *in distribution*

$$\mathbf{x}_t \overset{d}{=} \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where $\boldsymbol{\epsilon}$ is independent of $\mathbf{x}_0$. The symbol "$\overset{d}{=}$" means the two sides have the same probability law, so that

$$\mathbb{E}[\phi(\mathbf{x}_t)] = \mathbb{E}[\phi(\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon})]$$

for any test function ϕ. For brevity, we will simply write $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}$, understood either as an equality in distribution or as a sample realization depending on context; this shorthand is used throughout.

Part A

Introduction to Deep Generative Modeling

1

Deep Generative Modeling

What I cannot create, I do not understand.

Richard P. Feynman

Deep generative models (DGMs) are neural networks that learn a probability distribution over high-dimensional data (e.g., images, text, audio) so they can generate new examples that resemble the dataset. We denote the model distribution by $p_{\boldsymbol{\phi}}$ and the data distribution by p_{data}. Given a finite dataset, we fit $\boldsymbol{\phi}$ by minimizing a loss that measures how far $p_{\boldsymbol{\phi}}$ is from p_{data}. After training, generation amounts to running the model's sampling procedure to draw $\mathbf{x} \sim p_{\boldsymbol{\phi}}$ (the density $p_{\boldsymbol{\phi}}(\mathbf{x})$ may or may not be directly computable, depending on the model class). Model quality is judged by how well generated samples and their summary statistics match those of p_{data}, together with task-specific or perceptual metrics.

This chapter builds the mathematical and conceptual foundations behind these ideas. We formalize the problem in Section 1.1, present representative model classes in Section 1.2, and summarize a practical taxonomy in Section 1.3.

After this chapter, you will be able to...

- explain, in plain language, what a generative model is trying to learn from a dataset and what it means to model the data distribution;
- understand the core idea behind autoregressive models, VAEs, energy-based models, normalizing flows, GANs, and diffusion models, and see how they differ in the way they learn and generate new samples;

Reader route. If you are already familiar with the basic ideas behind modern generative models, you may skim this chapter and use it later as a quick reference.

1.1 What is Deep Generative Modeling?

DGMs take as input a large collection of real-world examples (e.g., images, text) drawn from an unknown and complex distribution p_{data} and output a trained neural network that parameterizes an approximate distribution $p_{\boldsymbol{\phi}}$. Their goals are twofold:

1. **Realistic Generation:** To generate novel, realistic samples indistinguishable from real data.
2. **Controllable Generation:** To enable fine-grained and interpretable control over the generative process.

This section presents the fundamental concepts and motivations behind DGMs, preparing for a detailed exploration of their mathematical framework and practical applications.

1.1.1 Mathematical Setup

Suppose we are given a finite collection of examples $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, such as images, audio, or text. We imagine that these examples were drawn from some underlying data distribution p_{data}, but we do not know this distribution explicitly. In particular, we only observe the available training examples; we do not have a formula for p_{data} or a direct procedure for drawing arbitrary new samples from it[1].

Goal of DGM. The goal of deep generative modeling is therefore to learn a model that can produce *new* samples whose statistical characteristics resemble those of the observed data. We denote the distribution represented by a model with parameters $\boldsymbol{\phi}$ by $p_{\boldsymbol{\phi}}$. After training, we denote the learned parameters by $\boldsymbol{\phi}^{\times}$ and hope that, in the aspects relevant to generation,

$$p_{\boldsymbol{\phi}^{\times}} \approx p_{\text{data}}.$$

This expression should be understood conceptually: with finite data, finite model capacity, and imperfect optimization, we do not expect the two distributions to be exactly identical. Rather, the learned model should capture enough of the structure of the data distribution to generate realistic and diverse new examples.

Once the model is trained, generation simply means drawing a new sample

$$\mathbf{x} \sim p_{\boldsymbol{\phi}^{\times}}.$$

[1] This is a common assumption in machine learning. For simplicity, we use the symbol p to represent either a probability distribution or its probability density/mass function, depending on the context.

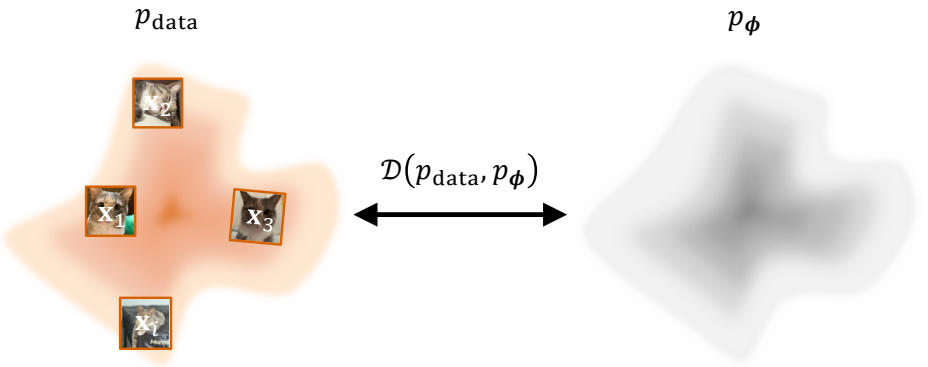


Figure 1.1: Illustration of the target in DGM. Training a DGM is essentially minimizing the discrepancy between the model distribution $p_{\boldsymbol{\phi}}$ and the unknown data distribution p_{data}. Since p_{data} is not directly accessible, this discrepancy must be estimated efficiently using a finite set of independent and identically distributed (i.i.d.) samples, $\mathbf{x}_i$, drawn from it.

Source: Created by the authors.

Different families of generative models provide different mechanisms for doing this. Some models additionally allow us to evaluate how much density they assign to an observed sample, while others are primarily defined through their sampling procedure. This distinction will become useful later when we discuss how different generative models are trained; for now, the central objective is simply to learn a distribution from data from which we can generate new samples.

Training of DGM. We learn parameters $\boldsymbol{\phi}$ of a model family $\{p_{\boldsymbol{\phi}}\}$ by minimizing a discrepancy $\mathcal{D}(p_{\text{data}}, p_{\boldsymbol{\phi}})$:

$$\boldsymbol{\phi}^* \in \arg\min_{\boldsymbol{\phi}} \ \mathcal{D}(p_{\text{data}}, p_{\boldsymbol{\phi}}). \tag{1.1.1}$$

Because p_{data} is unknown, a practical choice of $\mathcal{D}$ must admit efficient estimation from i.i.d. samples from p_{data}. With sufficient capacity, $p_{\boldsymbol{\phi}^*}$ can closely approximate p_{data}.

Forward KL and Maximum Likelihood Estimation (MLE). How should we choose the model parameters so that $p_{\boldsymbol{\phi}}$ resembles p_{data}? When the model density $p_{\boldsymbol{\phi}}(\mathbf{x})$ can be evaluated, a natural idea is to make the observed data receive high density under the model. This leads to *maximum likelihood estimation* (MLE): we choose $\boldsymbol{\phi}$ so that the training examples are collectively assigned high model density[2].

[2]For a fixed observed sample $\mathbf{x}$, the quantity $p_{\boldsymbol{\phi}}(\mathbf{x})$, viewed as a function of the model parameters $\boldsymbol{\phi}$, is called the *likelihood*. For continuous data, $p_{\boldsymbol{\phi}}(\mathbf{x})$ is a probability density rather than the probability of observing exactly $\mathbf{x}$.

A standard way to formalize this idea is through the (forward) Kullback–Leibler divergence[3]

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\boldsymbol{\phi}}) &:= \int p_{\mathrm{data}}(\mathbf{x}) \log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})}\, \mathrm{d}\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\big[\log p_{\mathrm{data}}(\mathbf{x}) - \log p_{\boldsymbol{\phi}}(\mathbf{x})\big].\end{aligned}$$

The KL divergence is asymmetric, i.e.,

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\boldsymbol{\phi}}) \neq \mathcal{D}_{\mathrm{KL}}(p_{\boldsymbol{\phi}}\|p_{\mathrm{data}}).$$

At first, the forward KL may appear unusable because the data density $p_{\mathrm{data}}(\mathbf{x})$ is unknown. However, it can be decomposed as

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\boldsymbol{\phi}}) = \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[\log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})}\right] = -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\big[\log p_{\boldsymbol{\phi}}(\mathbf{x})\big] - \mathcal{H}(p_{\mathrm{data}}),$$

where

$$\mathcal{H}(p_{\mathrm{data}}) := -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\big[\log p_{\mathrm{data}}(\mathbf{x})\big]$$

is the entropy of the data distribution. Crucially, this term does not depend on $\boldsymbol{\phi}$. Therefore, although we cannot evaluate $p_{\mathrm{data}}(\mathbf{x})$ itself, we do not need to know it in order to optimize the model parameters. This gives the following equivalence:

Lemma 1.1.1: Minimizing KL $\Leftrightarrow$ MLE

$$\min_{\boldsymbol{\phi}} \mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_{\boldsymbol{\phi}}) \iff \max_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\big[\log p_{\boldsymbol{\phi}}(\mathbf{x})\big]. \tag{1.1.2}$$

In words, minimizing the forward KL divergence is equivalent to making real data samples receive high log-likelihood under the model. This is exactly maximum likelihood estimation.

In practice, the expectation over p_{data} is replaced by the available training samples $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, assumed to be drawn i.i.d. from p_{data}. This gives the empirical negative log-likelihood objective

$$\hat{\mathcal{L}}_{\mathrm{MLE}}(\boldsymbol{\phi}) := -\frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{\phi}}(\mathbf{x}^{(i)}),$$

which can be optimized using stochastic gradients over minibatches. Thus, MLE requires only samples from p_{data}; it does not require evaluating the unknown data density itself.

[3] All integrals are in the Lebesgue sense and reduce to sums under counting measures.

The direction of the KL divergence matters because it determines *where the discrepancy is averaged*. For the forward KL,

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \| p_{\boldsymbol{\phi}}) = \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[\log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})}\right],$$

the expectation is taken over the data distribution. Thus, regions that occur under p_{data} must be accounted for by the model. By contrast, the reverse KL, $\mathcal{D}_{\mathrm{KL}}(p_{\boldsymbol{\phi}} \| p_{\mathrm{data}})$, averages instead over samples from the model distribution.

This difference has an important consequence. Suppose there exists a region A with positive data probability, $p_{\mathrm{data}}(A) > 0$, but the model assigns zero density there, $p_{\boldsymbol{\phi}}(\mathbf{x}) = 0$ for $\mathbf{x} \in A$. Then the forward KL becomes infinite, since

$$\log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})} = +\infty$$

on that region. Therefore, minimizing the forward KL strongly penalizes a model for completely missing regions where the data distribution places probability mass. This behavior is commonly referred to as *mode covering*.

Fisher Divergence. The Fisher divergence is another important concept for (score-based) diffusion modeling (see Chapter 3). For two distributions p and q, it is defined as

$$\mathcal{D}_{\mathrm{F}}(p\|q) := \mathbb{E}_{\mathbf{x}\sim p}\left[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \nabla_{\mathbf{x}} \log q(\mathbf{x})\|_2^2\right]. \tag{1.1.3}$$

It measures the discrepancy between the *score functions* $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ and $\nabla_{\mathbf{x}} \log q(\mathbf{x})$, which are vector fields pointing toward regions of higher probability. In short, $\mathcal{D}_{\mathrm{F}}(p\|q) \geq 0$ with equality if and only if $p = q$ almost everywhere. It is invariant to normalization constants, since scores depend only on gradients of log-densities, and it forms the basis of *score matching* (Equations (3.1.3) and (3.2.1)): a method that learns the gradient of the log-density for generation (score-based models). In this setting, the data distribution $p = p_{\mathrm{data}}$ serves as the target, while the model $q = p_{\boldsymbol{\phi}}$ is trained to align its score field with that of the data.

Beyond KL. Although the KL divergence is the most widely used measure of difference between probability distributions, it is not the only one. Different divergences capture different geometric or statistical notions of discrepancy, which in turn affect the optimization dynamics of learning algorithms. A broad family is the *f-divergences* (Csiszár, 1963):

$$\mathcal{D}_f(p\|q) = \int q(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) \mathrm{d}\mathbf{x}, \qquad f(1) = 0, \tag{1.1.4}$$

where $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex function. By changing f, we obtain many well-known divergences:

$$\begin{aligned} f(u) &= u \log u && \Rightarrow \quad \mathcal{D}_f = \mathcal{D}_{\mathrm{KL}}(p\|q) \quad \text{(forward KL)}, \\ f(u) &= \tfrac{1}{2}\left[u \log u - (u+1)\log \tfrac{1+u}{2}\right] && \Rightarrow \quad \mathcal{D}_f = \mathcal{D}_{\mathrm{JS}}(p\|q) \quad \text{(Jensen–Shannon)}, \\ f(u) &= \tfrac{1}{2}|u-1| && \Rightarrow \quad \mathcal{D}_f = \mathcal{D}_{\mathrm{TV}}(p,q) \quad \text{(total variation)}. \end{aligned}$$

For clarity, the explicit forms are

$$\mathcal{D}_{\mathrm{JS}}(p\|q) = \tfrac{1}{2}\mathcal{D}_{\mathrm{KL}}\big(p\|\tfrac{1}{2}(p+q)\big) + \tfrac{1}{2}\mathcal{D}_{\mathrm{KL}}\big(q\|\tfrac{1}{2}(p+q)\big),$$

and

$$\mathcal{D}_{\mathrm{TV}}(p,q) = \tfrac{1}{2}\int_{\mathbb{R}^D} |p-q|\,\mathrm{d}\mathbf{x} = \sup_{A\subset\mathbb{R}^D} |p(A) - q(A)|.$$

Intuitively, the JS divergence provides a smooth and symmetric measure that balances both distributions and avoids the unbounded penalties of KL (we will later see that it helps interpret the Generative Adversarial Network (GAN) framework), while the total variation distance captures the largest possible probability difference between the two.

A different viewpoint comes from *optimal transport* (see Chapter 7), whose representative is the Wasserstein distance (see Equation (7.2.2)). It measures the minimal cost of moving probability mass from one distribution to another. Unlike f-divergences, which compare density ratios, Wasserstein distances depend on the geometry of the sample space and remain meaningful even when the supports of p and q do not overlap.

Each divergence embodies a different notion of closeness between distributions and thus induces distinct learning behavior. We will revisit these divergences when they arise naturally in the context of generative modeling throughout this monograph.

1.1.2 Challenges in Modeling Distributions

To model a complex data distribution, we can parameterize the probability density function p_{data} using a neural network with parameters $\boldsymbol{\phi}$, creating a model we denote as $p_{\boldsymbol{\phi}}$. For $p_{\boldsymbol{\phi}}$ to be a valid probability density function, it must satisfy two fundamental properties:

(i) **Non-Negativity:** $p_{\boldsymbol{\phi}}(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ in the domain.

(ii) **Normalization:** The integral over the entire domain must equal one, i.e., $\int p_{\boldsymbol{\phi}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1$.

A network can naturally produce a real scalar $E_{\boldsymbol{\phi}}(\mathbf{x}) \in \mathbb{R}$ for input $\mathbf{x}$. To interpret this output as a valid density, it must be transformed to satisfy conditions (i) and (ii). A practical alternative is to view $E_{\boldsymbol{\phi}} \colon \mathbb{R}^D \to \mathbb{R}$ as defining an *unnormalized* density and then enforce these properties explicitly.

Step 1: Ensuring Non-Negativity. We can guarantee that our model's output is always non-negative by applying a positive function to the raw output of the neural network $E_{\boldsymbol{\phi}}(\mathbf{x})$, such as $|E_{\boldsymbol{\phi}}(\mathbf{x})|$, $E^2_{\boldsymbol{\phi}}(\mathbf{x})$. A standard and convenient choice is the exponential function. This gives us an unnormalized density, $\tilde{p}_{\boldsymbol{\phi}}(\mathbf{x})$, that is guaranteed to be positive:

$$\tilde{p}_{\boldsymbol{\phi}}(\mathbf{x}) = \exp(E_{\boldsymbol{\phi}}(\mathbf{x})).$$

Step 2: Enforcing Normalization. The function $\tilde{p}_{\boldsymbol{\phi}}(\mathbf{x})$ is positive but does not integrate to one. To create a valid probability density, we must divide it by its integral over the entire space. This leads to the final form of our model:

$$p_{\boldsymbol{\phi}}(\mathbf{x}) = \frac{\tilde{p}_{\boldsymbol{\phi}}(\mathbf{x})}{\int \tilde{p}_{\boldsymbol{\phi}}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'} = \frac{\exp(E_{\boldsymbol{\phi}}(\mathbf{x}))}{\int \exp(E_{\boldsymbol{\phi}}(\mathbf{x}'))\,\mathrm{d}\mathbf{x}'}.$$

The denominator in this expression is known as the *normalizing constant* or *partition function*, denoted by $Z(\boldsymbol{\phi})$:

$$Z(\boldsymbol{\phi}) := \int \exp(E_{\boldsymbol{\phi}}(\mathbf{x}'))\,\mathrm{d}\mathbf{x}'.$$

While this procedure provides a valid construction for $p_{\boldsymbol{\phi}}(\mathbf{x})$, it introduces a major computational challenge. For most high-dimensional problems, the integral required to compute the normalizing constant $Z(\boldsymbol{\phi})$ is intractable. This intractability is a central problem that motivates the development of many different families of deep generative models.

In the following sections, we introduce several prominent approaches of DGM. Each is designed to circumvent or reduce the computational cost of evaluating this normalizing constant.

1.2 Prominent Deep Generative Models

A central challenge in generative modeling is to learn expressive probabilistic models that can capture the rich and complex structure of high-dimensional data. Over the years, various modeling strategies have been developed, each making different trade-offs between tractability, expressiveness, and training efficiency. In this section, we explore some of the most influential strategies that have shaped the field, accompanied by a comparison of their computation graphs in Figure 1.2.

Energy-Based Models (EBMs). EBMs (Ackley *et al.*, 1985; LeCun *et al.*, 2006) define a probability distribution through an energy function $E_{\boldsymbol{\phi}}(\mathbf{x})$ that assigns lower energy to more probable data points. The probability of a data point is defined as:

$$p_{\boldsymbol{\phi}}(\mathbf{x}) := \frac{1}{Z(\boldsymbol{\phi})} \exp(-E_{\boldsymbol{\phi}}(\mathbf{x})),$$

where

$$Z(\boldsymbol{\phi}) = \int \exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))\, \mathrm{d}\mathbf{x}$$

is the partition function. Training EBMs typically involves maximizing the log-likelihood of the data. However, this requires techniques to address the computational challenges arising from the intractability of the partition function. In the following chapter, we will explore how Diffusion Models offer an alternative by generating data from *the gradient of the log density*, which does not depend on the normalizing constant, thereby circumventing the need for partition function computation.

Autoregressive Models. Deep autoregressive (AR) models (Frey *et al.*, 1995; Larochelle and Murray, 2011; Uria *et al.*, 2016) build on a simple but powerful observation: any joint distribution can be decomposed into a sequence of conditional distributions using the *chain rule of probability*. For a chosen ordering of the variables $\mathbf{x} = (x_1, \ldots, x_D)$, the data distribution satisfies

$$p_{\text{data}}(\mathbf{x}) = \prod_{i=1}^{D} p_{\text{data}}(x_i|\mathbf{x}_{<i}), \qquad \mathbf{x}_{<i} = (x_1, \ldots, x_{i-1}).$$

Importantly, this factorization is an exact identity rather than a modeling assumption: any joint distribution admits such a decomposition for any chosen ordering of its variables.

The challenge is that the true conditional distributions $p_{\text{data}}(x_i|\mathbf{x}_{<i})$ are unknown. AR models therefore parameterize a model with the same factorized form,

$$p_{\boldsymbol{\phi}}(\mathbf{x}) = \prod_{i=1}^{D} p_{\boldsymbol{\phi}}(x_i|\mathbf{x}_{<i}),$$

and use a neural network, such as a Transformer, to approximate these conditional distributions from data. For example, for discrete data the network may output a categorical distribution through a softmax, whereas for continuous data it may output the parameters of a tractable conditional density. Because every conditional distribution is normalized, their product automatically defines a normalized joint distribution.

This construction also makes likelihood-based training particularly simple. For a data sample $\mathbf{x}$, the log-likelihood decomposes as

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) = \sum_{i=1}^{D} \log p_{\boldsymbol{\phi}}(x_i|\mathbf{x}_{<i}),$$

so the model can be trained directly by maximum likelihood, or equivalently by minimizing the negative log-likelihood. During training, the observed preceding variables $\mathbf{x}_{<i}$ are available, so many of these conditional predictions can typically be evaluated in parallel.

Generation, however, remains inherently sequential:

$$x_1 \sim p_{\boldsymbol{\phi}}(x_1), \qquad x_2 \sim p_{\boldsymbol{\phi}}(x_2|x_1), \qquad \ldots, \qquad x_D \sim p_{\boldsymbol{\phi}}(x_D|\mathbf{x}_{<D}).$$

Thus, AR models provide tractable exact likelihoods and flexible density modeling, but sampling requires generating variables one after another and can therefore become computationally expensive for long sequences. AR models remain a foundational class of likelihood-based generative models and underlie many modern sequence and language models.

Variational Autoencoders (VAEs). VAEs (Kingma and Welling, 2013) extend classical autoencoders by introducing latent variables $\mathbf{z}$ that capture hidden structure in the data $\mathbf{x}$. Instead of directly learning a mapping between $\mathbf{x}$ and $\mathbf{z}$, VAEs adopt a probabilistic view: they learn both an *encoder*, $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which approximates the unknown distribution of latent variables given the data, and a *decoder*, $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$, which reconstructs data from these latent variables. To make training feasible, VAEs maximize a tractable surrogate to the true log-likelihood, called the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})\right] - \mathcal{D}_{\mathrm{KL}} \left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \,\|\, p_{\mathrm{prior}}(\mathbf{z})\right).$$

Here, the first term encourages accurate reconstruction of the data, while the second regularizes the latent variables by keeping them close to a simple prior distribution $p_{\mathrm{prior}}(\mathbf{z})$ (often Gaussian).

VAEs provide a principled way to combine neural networks with latent-variable models and remain one of the most widely used likelihood-based approaches. However, they also face practical challenges, such as limited sample sharpness and

training pathologies (e.g., the tendency of the decoder to ignore latent variables). Despite these limitations, VAEs laid important foundations for later advances, including diffusion models.

Normalizing Flows. Classic flow-based generative models include Normalizing Flows (NFs) (Rezende and Mohamed, 2015) and their continuous-time counterparts based on Neural Ordinary Differential Equations (NODEs) (Chen *et al.*, 2018), aim to learn a bijective mapping $\mathbf{f}_{\boldsymbol{\phi}}$ between a simple latent distribution $\mathbf{z}$ and a complex data distribution $\mathbf{x}$ via an invertible operator. This is achieved either through a sequence of bijective transformations (in NFs) or by modeling the transformation as an Ordinary Differential Equation (in NODEs). These models leverage the "change-of-variable formula for densities", enabling MLE training:

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) = \log p(\mathbf{z}) + \log \left| \det \frac{\partial \mathbf{f}_{\boldsymbol{\phi}}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|,$$

where $\mathbf{f}_{\boldsymbol{\phi}}$ represents the invertible transformation mapping $\mathbf{z}$ to $\mathbf{x}$. NFs explicitly model normalized densities using invertible transformations with tractable Jacobian determinants. The normalization constant is absorbed analytically via the change-of-variables formula, making likelihood computation exact and tractable.

Despite their conceptual elegance, classic flow-based models often face practical limitations. For instance, NFs typically impose restrictive architectural constraints to ensure bijectivity, while NODEs may encounter training inefficiencies due to the computational overhead of solving ODEs. Both approaches face challenges when scaling to high-dimensional data. In later chapters, we will explore how Diffusion Models relate to and build upon these classic flow-based methods.

Generative Adversarial Networks (GANs). GANs (Goodfellow *et al.*, 2014) consist of two neural networks, a generator $\mathbf{G}_{\boldsymbol{\phi}}$ and a discriminator $D_{\boldsymbol{\zeta}}$, that compete against each other. The generator aims to create realistic samples $\mathbf{G}_{\boldsymbol{\phi}}(\mathbf{z})$ from random noise $\mathbf{z} \sim p_{\text{prior}}$, while the discriminator attempts to distinguish between real samples $\mathbf{x}$ and generated samples $\mathbf{G}_{\boldsymbol{\phi}}(\mathbf{z})$. The objective function for GANs can be formulated as:

$$\min_{\mathbf{G}_{\boldsymbol{\phi}}} \max_{D_{\boldsymbol{\zeta}}} \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D_{\boldsymbol{\zeta}}(\mathbf{x})]}_{\text{real}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\text{prior}}(\mathbf{z})} \left[\log(1 - D_{\boldsymbol{\zeta}}\left(\mathbf{G}_{\boldsymbol{\phi}}(\mathbf{z})\right))\right]}_{\text{fake}}.$$

GANs do not define an explicit density function and therefore bypass likelihood estimation entirely. Instead of computing a normalization constant, they focus on generating samples that closely mimic the data distribution.

From a divergence perspective, the discriminator implicitly measures the discrepancy between the true data distribution p_{data} and the generator distribution

$p_{\mathbf{G}_\phi}$, where $p_{\mathbf{G}_\phi}$ denotes the distribution of generated samples $\mathbf{G}_\phi(\mathbf{z})$ obtained from noise $\mathbf{z} \sim p_{\text{prior}}$. With an optimal discriminator for a fixed generator $\mathbf{G}_\phi$ computed as

$$\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\mathbf{G}_\phi}(\mathbf{x})},$$

the generator's minimization reduces to

$$\min_{\mathbf{G}_\phi} \; 2\,\mathcal{D}_{\text{JS}}\left(p_{\text{data}} \,\|\, p_{\mathbf{G}_\phi}\right) - \log 4.$$

Here, $\mathcal{D}_{\text{JS}}$ denotes the Jensen–Shannon divergence, defined as

$$\mathcal{D}_{\text{JS}}(p \,\|\, q) := \tfrac{1}{2}\mathcal{D}_{\text{KL}}\left(p \,\middle\|\, \tfrac{p+q}{2}\right) + \tfrac{1}{2}\mathcal{D}_{\text{KL}}\left(q \,\middle\|\, \tfrac{p+q}{2}\right).$$

This shows that GANs implicitly minimize $\mathcal{D}_{\text{JS}}(p_{\text{data}} \,\|\, p_{\mathbf{G}_\phi})$. More broadly, extensions such as f-GANs (Nowozin *et al.*, 2016) generalize this view by demonstrating that adversarial training can minimize a family of f-divergences, placing GANs within the same divergence-minimization framework as other generative models.

Although GANs are capable of generating high-quality data, their min-max training process is notoriously unstable, often requiring carefully designed architectures and engineering techniques to achieve satisfactory performance. However, GANs have since been revived as an auxiliary component to enhance other generative models, particularly Diffusion Models.

1.3 Taxonomy of Modeling

As we have seen, DGMs span a wide spectrum of modeling strategies. A fundamental distinction lies in how these models *parameterize* the underlying data distribution, that is, whether they specify $p_{\phi}(\mathbf{x})$ *explicitly* or only *implicitly*, irrespective of the training objective.

- **Explicit Models:** These models directly parameterize a probability distribution $p_{\phi}(\mathbf{x})$ via a tractable or approximately tractable density or mass function. Examples include ARs, NFs, VAEs, and DMs, all of which define $p_{\phi}(\mathbf{x})$ either exactly or through a tractable bound.
- **Implicit Models:** These models specify a distribution only through a sampling procedure, typically of the form $\mathbf{x} = \mathbf{G}_{\phi}(\mathbf{z})$ for some noise variable $\mathbf{z} \sim p_{\text{prior}}$. In this case, $p_{\phi}(\mathbf{x})$ is not available in closed form and may not be defined at all.

The table in Table 1.1 offers a concise summary of these contrasting approaches.

Table 1.1: Comparison of Explicit and Implicit Generative Models

	Explicit		**Implicit**
	Exact Likelihood	**Approx. Likelihood**	
Likelihood	Tractable	Bound/Approx.	Not Directly Modeled/ Intractable
Objective	MLE	ELBO	Adversarial
Examples	NFs, ARs	VAEs, DMs	GANs

Connection to Diffusion Models. Taken together, these classical families of DGMs illustrate complementary strategies for modeling complex distributions. Beyond their standalone importance, they also provide guiding principles for understanding diffusion models. Diffusion methods inherit ideas from several of these perspectives: they connect to VAEs through variational training objectives, to EBMs through score-matching approaches that learn gradients of the log-density (closely tied to energy functions), and to NFs through continuous-time transformations.

To lay the groundwork for the diffusion methods discussed in later chapters, we will focus on three central paradigms: VAEs (Section 2.1), EBMs (Section 3.1), and NFs (Section 5.1). This exploration provides a foundation for the core principles that underlie modern diffusion-based generative modeling, which will be developed further in the chapters that follow.

1.4 Closing Remarks

This chapter has established the foundational concepts of deep generative modeling. We begin by defining the primary objective: to learn a tractable model distribution p_{model} (parametrized by $\boldsymbol{\phi}$) that approximates an unknown, complex data distribution p_{data}. A central challenge is the computational intractability of the normalizing constant, or partition function $Z(\boldsymbol{\phi})$, which is required to define a valid probability density.

To circumvent this problem, various families of deep generative models have been developed, each employing a distinct strategy. We surveyed several prominent approaches, including Energy-Based Models (EBMs), Autoregressive Models (ARs), Variational Autoencoders (VAEs), Normalizing Flows (NFs), and Generative Adversarial Networks (GANs). These models can be broadly categorized into explicit models, which define a tractable density, and implicit models, which define a distribution only through a sampling procedure.

While each of these classical frameworks is significant, three in particular serve as the conceptual origins for the diffusion models that are the focus of this monograph: VAEs, EBMs, and NFs. In the chapters that follow, we will trace the evolution of diffusion models from these three foundational paradigms:

1. Part B will begin by exploring the variational perspective (Chapter 2), showing how (the hierarchical latent variable structure of) VAEs leads naturally to the formulation of Denoising Diffusion Probabilistic Models (DDPMs).

2. Next, we will examine the score-based perspective (Chapter 3), which originates from EBMs and score matching, and develops into Noise Conditional Score Networks (NCSN) and the more general Score SDE framework (Chapter 4).

3. Finally, we will investigate the flow-based perspective (Chapter 5), which builds upon the principles of Normalizing Flows to frame generation as a continuous transformation, generalized by the concept of Flow Matching.

By understanding these origins, we will build a coherent framework for interpreting the diverse formulations of diffusion models and uncovering the deep principles that unify them.

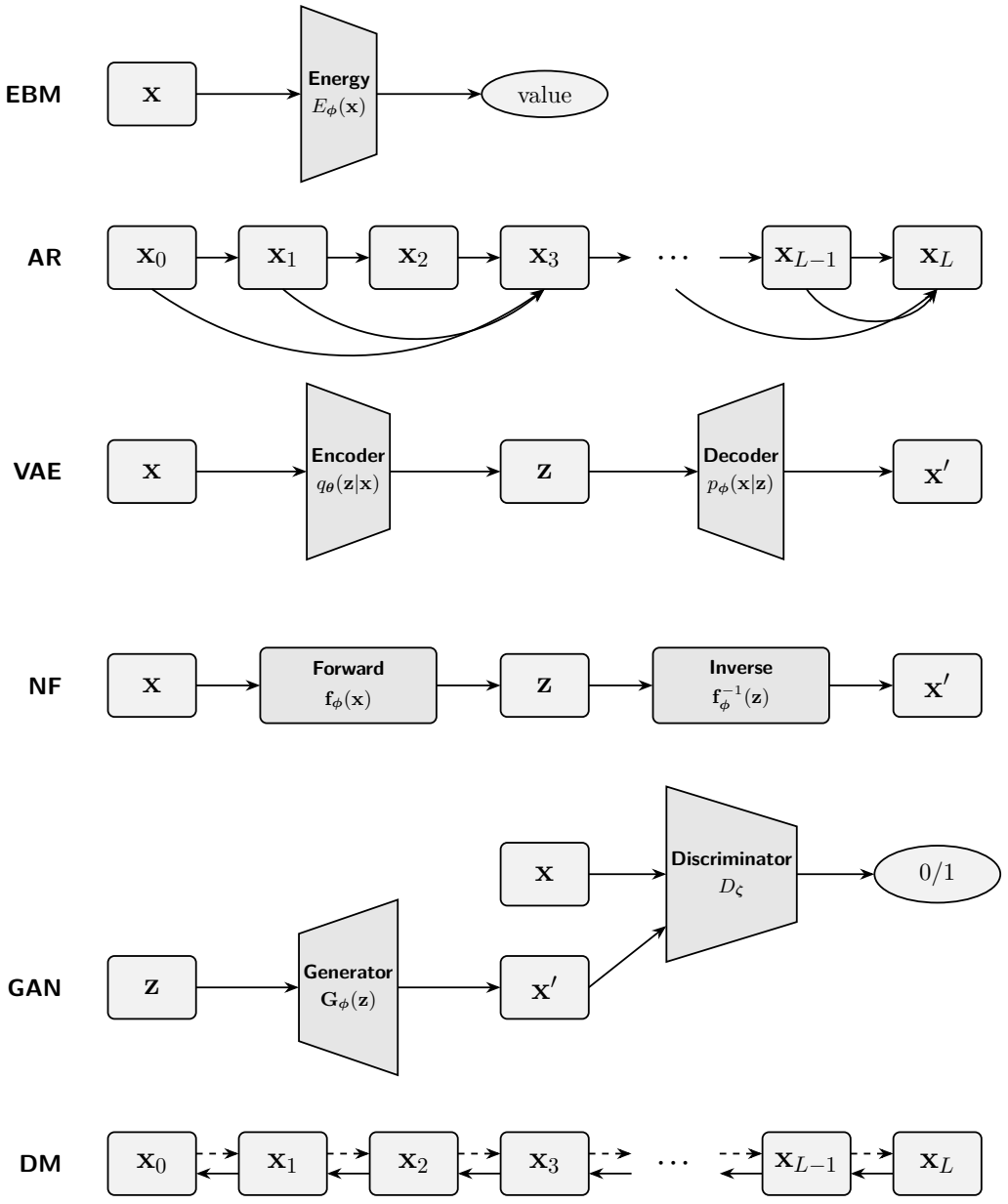
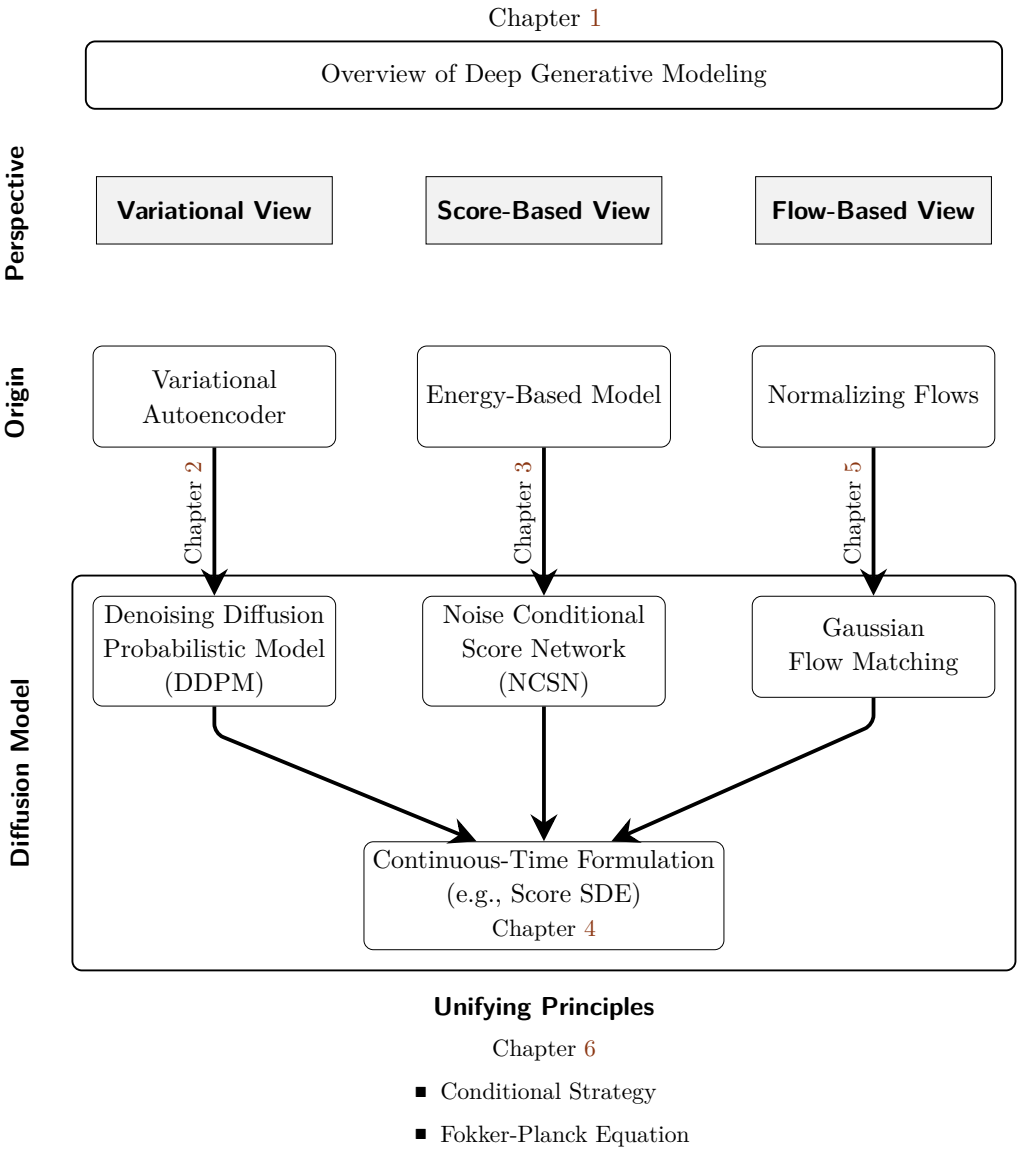


Figure 1.2: Computation graphs of prominent deep generative models. Top to bottom: **EBM** maps an input $\mathbf{x}$ to a scalar energy; **AR** generates a sequence $\{\mathbf{x}_\ell\}_{\ell=0}^{L}$ left to right with causal dependencies; **VAE** encodes $\mathbf{x}$ to a latent $\mathbf{z}$ and decodes to a reconstruction $\mathbf{x}'$; **NF** applies an invertible map $\mathbf{f}_{\boldsymbol{\phi}}$ between $\mathbf{x}$ and $\mathbf{z}$ and uses $\mathbf{f}_{\boldsymbol{\phi}}^{-1}$ to produce $\mathbf{x}'$; **GAN** transforms noise $\mathbf{z}$ to a sample $\mathbf{x}'$ that is judged against real $\mathbf{x}$ by a discriminator $D_{\boldsymbol{\varsigma}}$; **DM** iteratively refines a noisy sample through a multi-step denoising chain $\{\mathbf{x}_\ell\}_{\ell=0}^{L}$. Boxes denote variables, trapezoids are learnable networks, ovals are scalars; arrows indicate computation flow. The trapezoid was intended to indicate a dimension-changing transformation, while rectangles denote variables or mappings that preserve dimensionality.

Source: Created by the authors.

Part B

Foundations and Perspectives of Diffusion Models


Chapter 1
Overview of Deep Generative Modeling
Perspective
Variational View
Score-Based View
Flow-Based View
Origin
Variational Autoencoder
Energy-Based Model
Normalizing Flows
Chapter 2
Chapter 3
Chapter 5
Diffusion Model
Denoising Diffusion Probabilistic Model (DDPM)
Noise Conditional Score Network (NCSN)
Gaussian Flow Matching
Continuous-Time Formulation (e.g., Score SDE)
Chapter 4
Unifying Principles
Chapter 6
Conditional Strategy
Fokker-Planck Equation

A First Operational View of Diffusion Model Training and Sampling

Before diving into the mathematics of diffusion models, let us first develop a simple operational view of how they are trained and how they generate samples.

The basic construction has two parts. First, we define a known corruption rule that can turn clean data into noisy observations at any chosen noise level. Second, we train a neural network to infer the clean signal from such a noisy observation. Once this prediction rule has been learned across many noise levels, it can also be used for generation. We start from a fresh noise sample and repeatedly apply the network while moving from high noise levels to lower ones. The result is a new sample that gradually acquires the structure of the data distribution.

Message

Create noisy data at many noise levels, learn to predict the clean signal, and generate by repeatedly applying this prediction from high noise to low.

Creating Noisy Data. Let us make precise what a noisy version of the data actually means. Take a a clean data sample

$$\mathbf{x}_0 \sim p_{\text{data}},$$

such as an image or any other kind of data we care about. We corrupt it at noise level t by mixing in independent Gaussian noise[4]:

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Think of α_t and σ_t as two knobs, fixed in advance, that set how much clean signal and how much noise enter the mix at noise level t.

The simplest possible choice, $\alpha_t = 1$ and $\sigma_t = t$, gives

$$\mathbf{x}_t = \mathbf{x}_0 + t\boldsymbol{\epsilon}.$$

Now t is literally a noise dial: turn it toward zero and $\mathbf{x}_t$ is nearly the clean sample; turn it up and the noise term $t\boldsymbol{\epsilon}$ increasingly takes over.

More generally, every noise level t defines a distribution p_t over noisy samples. It is useful to think of p_t as a *snapshot*: if we generated many noisy samples using the same value of t, their distribution would be p_t.

[4] The formula $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ gives the accumulated corruption at a chosen noise level t directly. In DDPMs and Score SDEs, the same marginal distribution at that level can also be obtained by adding noise through many smaller sequential steps; see Section 2.2 and Chapter 4. In the discrete DDPM notation introduced later, this accumulated corruption is written as

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2}\, \boldsymbol{\epsilon}.$$

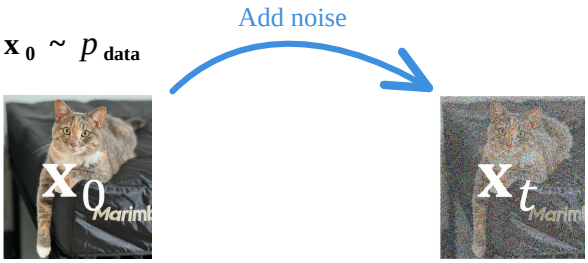


Figure 1.3: Forward diffusion. A clean sample $\mathbf{x}_0$ is corrupted to produce a noisy observation $\mathbf{x}_t$ at noise level t.

Source: Created by the authors.

The schedule (α_t, σ_t) is chosen so that these snapshots move from clean data toward a simple noise distribution:

$$p_0 = p_{\text{data}} \qquad \longrightarrow \qquad p_t \qquad \longrightarrow \qquad p_T \approx p_{\text{prior}},$$

where p_{prior} is typically a Gaussian distribution that is easy to sample from. Thus, the family

$$\{p_t\}_{t\in[0,T]}$$

describes how the distribution gradually changes as the noise level increases.

Training: Learn to Predict the Clean Signal. With a way to corrupt data now in hand, training becomes almost mechanical, and it hinges on one key trick: because we are the ones corrupting the data, we always know the right answer. To build a single training example, we sample a clean datum $\mathbf{x}_0$, choose a noise level t, draw noise $\boldsymbol{\epsilon}$, and form

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}.$$

We manufactured $\mathbf{x}_t$ ourselves, so we already know exactly which clean $\mathbf{x}_0$ produced it.

The network, however, never sees $\mathbf{x}_0$ or $\boldsymbol{\epsilon}$ directly. It sees only the pair $(\mathbf{x}_t, t)$, the corrupted observation and how corrupted it is. Its task is to guess the clean data behind it:

$$\mathbf{x}_0^{\text{pred}} := \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t).$$

Because we know the true target, we can score this guess and improve it using an ordinary squared-error loss:

$$\left\| \mathbf{x}_0^{\text{pred}} - \mathbf{x}_0 \right\|_2^2.$$

Message

In a nutshell: create a noisy observation with a known clean target, hide that target from the network, and train it to predict the target back.

Repeat this many times, over different data samples, noise draws, and noise levels, and a single network learns to make clean-data predictions across the entire range of corruption, from barely touched to nearly pure noise.

It is worth pausing on what this loss teaches the network to predict. Under squared error, the best possible prediction is

$$\mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t, t].$$

A noisy observation generally does not correspond to one unique clean sample. Several different clean samples could have produced a similar $\mathbf{x}_t$. This ambiguity exists at any nonzero noise level, but it becomes much stronger when the noise level is high. The network therefore learns to average over the clean samples that are plausible given the current noisy observation [5].

Sampling: Start from Fresh Noise. Training leaves us with a network that is good at guessing clean data from noisy data. The next question is how to put that skill to use for generating something new.

Once training is finished, we freeze the network's parameters and call them $\boldsymbol{\phi}^\times$. Generation now runs in the direction opposite to the one we have been describing: rather than starting at data and adding noise, it starts from noise alone,

$$\tilde{\mathbf{x}}_T \sim p_{\text{prior}}.$$

We use a tilde to mark these generated states, $\tilde{\mathbf{x}}_t$, and keep them visually distinct from the training variables $\mathbf{x}_t$ produced by forward corruption.

This distinction matters more than it might first appear. The state $\tilde{\mathbf{x}}_T$ was not produced by secretly corrupting some real, hidden sample: there is no true $\mathbf{x}_0$ behind it waiting to be recovered. Generation is therefore not "rewinding a recording" of some specific corruption, because no such recording exists. Instead, at every noise level we simply ask the trained network what it currently believes the clean signal looks like, and use that belief to take one step back toward data.

Concretely, suppose we currently hold the state $\tilde{\mathbf{x}}_t$. Feeding it to the network gives a clean-data guess,

$$\mathbf{x}_0^{\text{pred}} := \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_t, t).$$

[5] When the noise level is clear from context, we will often shorten this to $\mathbb{E}[\mathbf{x}_0|\mathbf{x}_t]$.

For $t > 0$, we can also compute an implied noise residual, defined as whatever is left over once the predicted signal is removed from the current state:

$$\boldsymbol{\epsilon}_t^{\text{pred}} := \frac{\tilde{\mathbf{x}}_t - \alpha_t \mathbf{x}_0^{\text{pred}}}{\sigma_t}.$$

These two quantities are not independent guesses: by construction, they exactly reassemble the current state,

$$\tilde{\mathbf{x}}_t = \alpha_t \mathbf{x}_0^{\text{pred}} + \sigma_t \boldsymbol{\epsilon}_t^{\text{pred}}.$$

Here is the idea behind a single sampling step: keep the predicted signal and predicted noise as they are, but re-mix them using the coefficients for a slightly lower noise level $t - \Delta t$ in place of t:

$$\tilde{\mathbf{x}}_{t-\Delta t} = \alpha_{t-\Delta t} \mathbf{x}_0^{\text{pred}} + \sigma_{t-\Delta t} \boldsymbol{\epsilon}_t^{\text{pred}}.$$

Equivalently, this is a small update on top of the current state,

$$\tilde{\mathbf{x}}_{t-\Delta t} = \tilde{\mathbf{x}}_t + (\alpha_{t-\Delta t} - \alpha_t)\mathbf{x}_0^{\text{pred}} + (\sigma_{t-\Delta t} - \sigma_t)\boldsymbol{\epsilon}_t^{\text{pred}}. \tag{1.4.1}$$

That is one step. We now feed the new state $\tilde{\mathbf{x}}_{t-\Delta t}$ back into the network, obtain an updated, usually different, clean-data prediction, and take another step down in noise. Sampling is exactly this loop, repeated from $t = T$ down to $t = 0$.

Message

In a nutshell: start from fresh noise, predict the clean signal, step to a lower noise level, and predict again.

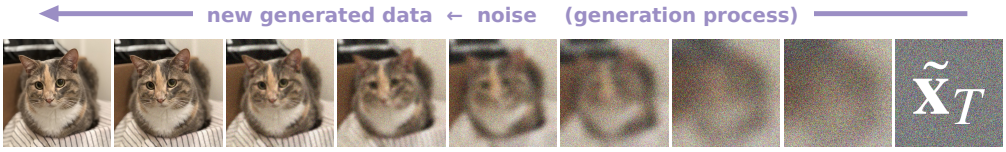


Figure 1.4: Diffusion generation. Starting from fresh noise $\tilde{\mathbf{x}}_T \sim p_{\text{prior}}$, repeated model predictions construct a new sample as the noise level decreases.

Source: Created by the authors.

A natural question at this point: why not simply output that very first prediction $\mathbf{x}_0^{\text{pred}}$ and stop? Because at high noise levels the current state carries very little information about clean structure, so the prediction is little more than a rough average over many plausible samples. Moving to a slightly lower noise level gives the network a slightly less ambiguous state to work from, so its next prediction can sharpen. Repeating this many times is what lets a genuinely detailed sample gradually emerge[6].

[6] We will later connect this deterministic update to DDIM sampling and its Euler-type

A Simple Example. Let us make this tangible using our earlier running example where $\alpha_t = 1$ and $\sigma_t = t$. Substituting into the update above gives

$$\tilde{\mathbf{x}}_{t-\Delta t} = \mathbf{x}_0^{\text{pred}} + (t - \Delta t)\frac{\tilde{\mathbf{x}}_t - \mathbf{x}_0^{\text{pred}}}{t},$$

which rearranges into a simple weighted average,

$$\tilde{\mathbf{x}}_{t-\Delta t} = \frac{t - \Delta t}{t}\tilde{\mathbf{x}}_t + \frac{\Delta t}{t}\mathbf{x}_0^{\text{pred}}.$$

In words, for small Δt the next state is simply a blend of where we currently stand and where the network thinks we should end up.

So each step nudges the sample toward whatever the network currently predicts to be clean. The word *currently* is doing real work here: once we move, the network sees a new, less noisy state and typically changes its prediction. The final sample is therefore not the output of a single clean-data guess. It is built from an entire sequence of guesses, each one revised in light of the last.

How this Connects to the Next Chapters? The clean-data prediction template developed here is only a first operational view. The basic pattern will remain the same throughout the book: during training, we create noisy observations from clean data and learn how to predict useful information from them; during generation, we start from fresh noise and use these learned predictions to construct a new sample over decreasing noise levels.

The following chapters explain this same process from different mathematical perspectives. Chapter 2 connects DDPMs to latent-variable models, where the fixed noising process plays the role of an encoder and the learned reverse transitions play the role of a decoder. Chapter 3 describes denoising through the score, which provides a direction for updating noisy samples. Chapter 4 moves from discrete noise levels to continuous time and describes generation through reverse-time stochastic dynamics. Chapter 5 instead views generation as transporting samples along a continuous path from a simple prior toward the data distribution. Finally, Chapter 6 explains how these viewpoints are connected.

The main takeaway is simple: diffusion models learn from noisy versions of data at many noise levels, and generation uses these learned predictions repeatedly to turn fresh noise into a new sample.

---

interpretation (Song *et al.*, 2020a) in Section 9.2; the stochastic DDPM sampler is introduced in Section 2.2.

2

Variational Perspective: From VAEs to DDPMs

In this chapter we view diffusion models through a variational lens. We begin with the Variational Autoencoders (VAEs), which represent data with latent variables and are trained by maximizing a tractable lower bound on the log likelihood. In this setting a learned encoder maps observations to latents, and a learned decoder maps latents back to observations, closing the modeling loop.

Building on this pattern, hierarchical variants (Hierarchical VAEs) stack several latent layers to capture structure at multiple scales. With this setup, Denoising Diffusion Probabilistic Models (DDPM) conceptually follow the similar template: instead of jointly training both the encoder and decoder, the encoder is fixed as a forward noising process that gradually maps data to noise, and training learns a decoder that reverses this path in successive denoising steps. In this view, VAEs, hierarchical VAEs, and diffusion models all optimize a likelihood surrogate defined by a variational bound, providing a common foundation for the methods introduced here.

After this chapter, you will be able to...

- understand why VAEs optimize the ELBO when exact maximum-likelihood training is difficult;
- see how DDPM extends the latent-variable view of VAEs into many noisy layers, with a fixed noising process and a learned reverse process;
- understand the conditioning trick that turns the difficult reverse-denoising problem into a tractable training target;
- compare mean, clean-data, and noise prediction, and see how they represent the same underlying denoising information;
- connect the practical DDPM denoising loss back to its variational founda-

tion through the ELBO.

Reader route. Readers already familiar with VAEs and ELBO training may skim Section 2.1 and begin with Section 2.2.

2.1 Variational Autoencoder

How can a neural network learn to generate realistic data? A natural starting point is the *autoencoder*, which consists of two networks: a deterministic *encoder* that compresses an input to a low-dimensional latent code, and a deterministic *decoder* that reconstructs the input from this code. Training minimizes the reconstruction error between the original input and its reconstruction. While this setup enables accurate reconstruction, the latent space is unstructured: randomly sampling latent codes usually produces meaningless outputs, limiting the model's use for generation.

The *Variational Autoencoder (VAE)* (Kingma and Welling, 2013) solves this by imposing a probabilistic structure on the latent space. This transforms the model from a simple reconstruction tool into a true generative model, capable of producing novel and realistic data.

2.1.1 Probabilistic Encoder and Decoder

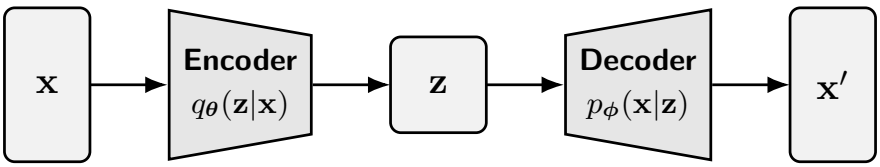


Figure 2.1: Illustration of a VAE. It consists of a stochastic encoder $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ that maps data $\mathbf{x}$ to a latent variable $\mathbf{z}$, and a decoder $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$ that reconstructs data from the latent.

Source: Created by the authors.

Construction of Decoder (Generator). In VAEs, we distinguish between two types of variables: *observed variables* $\mathbf{x}$, which correspond to the data we see (e.g., an image), and *latent variables* $\mathbf{z}$, which capture the hidden factors of variation (e.g., object shape, color, or style). The model assumes that each observation $\mathbf{x}$ is generated from a latent variable sampled from a simple *prior distribution*, typically a standard Gaussian, $\mathbf{z} \sim p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$.

To map $\mathbf{z}$ back to data space, we define a *decoder (generator)* distribution $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$. In practice, this decoder is kept simple, often a factorized Gaussian (see Section 2.1.3) or similar distribution, so that learning focuses on extracting useful latent features rather than memorizing data. Intuitively, directly generating pixels one by one is extremely hard; instead, the latent variable provides a compact representation, from which decoding the exact pixel arrangement becomes much easier. New samples are drawn by first sampling $\mathbf{z} \sim p_{\text{prior}}$ and then decoding via $\mathbf{x} \sim p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$.

The VAE thereby defines a latent-variable generative model through the marginal likelihood:

$$p_{\boldsymbol{\phi}}(\mathbf{x}) = \int p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z}.$$

Ideally, the decoder parameters $\boldsymbol{\phi}$ are learned by maximizing this marginal likelihood, as in maximum likelihood estimation (see Equation (1.1.2)). However, because the integral over $\mathbf{z}$ is intractable for expressive, non-linear decoders, direct MLE is computationally infeasible, motivating the variational approach used in VAEs.

Construction of Encoder (Inference Network). To connect our intractable generator to real data, consider the reverse question: given an observation $\mathbf{x}$, what latent codes $\mathbf{z}$ could have produced it? By Bayes' rule, the posterior distribution is

$$p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \;=\; \frac{p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_{\boldsymbol{\phi}}(\mathbf{x})}.$$

The difficulty is that the denominator involves the marginal likelihood $p_{\boldsymbol{\phi}}(\mathbf{x})$, which requires integrating over all latent variables and is intractable for nonlinear decoders. Thus, exact inference of $\mathbf{z}$ from $\mathbf{x}$ is computationally prohibitive.

The "variational" step in VAEs addresses this by replacing the intractable posterior with a tractable approximation. We introduce an *encoder* (or inference network) $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, parameterized by a neural network, whose role is to serve as a learnable proxy:

$$q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \;\approx\; p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}).$$

In practice, the encoder maps each observed data point $\mathbf{x}$ to a distribution over latent codes, providing a feasible and trainable pathway from $\mathbf{x}$ back to $\mathbf{z}$ that enables learning.

2.1.2 Training via the Evidence Lower Bound (ELBO)

We now define a computable training objective. While we cannot directly optimize $\log p_{\boldsymbol{\phi}}(\mathbf{x})$, we can maximize a lower bound on it—the *Evidence Lower Bound (ELBO)*:

Theorem 2.1.1: Evidence Lower Bound (ELBO)

For any data point $\mathbf{x}$, the log-likelihood satisfies:

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) \geq \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}),$$

where the ELBO is given by:

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})\right]}_{\text{Reconstruction Term}} - \underbrace{\mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right)}_{\text{Latent Regularization}}. \tag{2.1.1}$$

Proof for Theorem.

The ELBO arises from Jensen's inequality (see Section E.3):

$$\begin{aligned}\log p_{\boldsymbol{\phi}}(\mathbf{x}) &= \log \int p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z} = \log \int q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\ &= \log \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right] \geq \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right].\end{aligned}$$

The ELBO objective naturally decomposes into two parts:

- **Reconstruction:** Encourages accurate recovery of $\mathbf{x}$ from its latent code $\mathbf{z}$. With Gaussian encoder and decoder assumptions, this term reduces exactly to the familiar reconstruction loss of an autoencoder (cf. Section 2.1.3). However, as in autoencoders, optimizing this term alone risks memorizing the training data, motivating an additional regularization.
- **Latent KL:** Encourages the encoder distribution $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ to stay close to a simple Gaussian prior $p_{\text{prior}}(\mathbf{z})$. This regularization shapes the latent space into a smooth and continuous structure, enabling meaningful generation by ensuring that samples drawn from the prior can be reliably decoded.

This trade-off ensures both faithful reconstructions and coherent sampling.

Information-Theoretic View: ELBO as a Divergence Bound. The ELBO objective has a natural information-theoretic interpretation. Recall that maximum likelihood training amounts to minimizing the KL divergence

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x}) \| p_{\boldsymbol{\phi}}(\mathbf{x})),$$

which measures how well the model distribution approximates the data distribution. Since this term is intractable in general, the variational framework introduces a joint comparison.

Specifically, consider two joint distributions:

- The **generative joint**, $p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$, which describes how the model generates data;
- The **inference joint**, $q_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which couples real data with its inferred latent.

Comparing these distributions yields the inequality

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x})\|p_{\boldsymbol{\phi}}(\mathbf{x})) \leq \mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\|p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})), \tag{2.1.2}$$

sometimes referred to as the chain rule for KL divergence. Intuitively, comparing only marginals ($\mathbf{x}$) can hide mismatches that are revealed when the full latent–data joint is considered.

Formally, one can expand the joint KL as

$$\begin{aligned}
&\underbrace{\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\|p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}))}_{\text{Total Error Bound}} \\
=&\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}\left[\log \frac{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right] \\
=&\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})} + \mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right)\right] \\
=& \underbrace{\mathcal{D}_{\text{KL}}(p_{\text{data}}\|p_{\boldsymbol{\phi}})}_{\text{True Modeling Error}} + \underbrace{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))]}_{\text{Inference Error}},
\end{aligned}$$

where the first term is the true modeling error and the second is the inference error, i.e., the gap between the approximate and true posteriors. The latter is always non-negative, which explains Equation (2.1.2).

Finally, note that since $p_{\boldsymbol{\phi}}(\mathbf{x})$ does not depend on $\mathbf{z}$,

$$\begin{aligned}
\log p_{\boldsymbol{\phi}}(\mathbf{x}) &= \log\left[p_{\boldsymbol{\phi}}(\mathbf{x})\int q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\,\mathrm{d}\mathbf{z}\right] = \log\int q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\,p_{\boldsymbol{\phi}}(\mathbf{x})\,\mathrm{d}\mathbf{z} \\
&= \int q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\log p_{\boldsymbol{\phi}}(\mathbf{x})\,\mathrm{d}\mathbf{z} = \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x})] = \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\cdot\frac{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right] + \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) + \mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\,\|\,p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})),
\end{aligned}$$

i.e.,

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) - \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right).$$

Thus the inference error is exactly the gap between the log-likelihood and the ELBO. Maximizing the ELBO therefore corresponds to directly reducing inference error, ensuring that training minimizes a meaningful part of the overall bound.
Corollary. Optimizing the ELBO w.r.t. the encoder parameters $\boldsymbol{\theta}$ corresponds to minimizing the inference error (the gap to the log-likelihood, which itself does not depend on $\boldsymbol{\theta}$); optimizing w.r.t. the decoder parameters $\boldsymbol{\phi}$ corresponds to minimizing the inference error while maximizing the true log-likelihood.

2.1.3 Gaussian VAE

A standard formulation of the VAE employs Gaussian distributions for both the encoder and decoder.

Encoder Part. The encoder $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is typically modeled as a Gaussian distribution as:

$$q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) := \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\mathbf{x}))\right),$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^d$ and $\boldsymbol{\sigma}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^d_+$ are deterministic outputs of the encoder network.

Decoder Part. The decoder is typically modeled as a Gaussian distribution with fixed variance:

$$p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}) := \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{z}), \sigma^2\mathbf{I}),$$

where $\boldsymbol{\mu}_{\boldsymbol{\phi}} : \mathbb{R}^d \to \mathbb{R}^D$ is a neural network, and $\sigma > 0$ is a small constant controlling the variance.

Under this assumption, the reconstruction term in the ELBO simplifies as

$$\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})\right] = -\frac{1}{2\sigma^2}\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x} - \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{z})\|^2\right] + C,$$

where C is a constant independent of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. The ELBO objective thus reduces to:

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\frac{1}{2\sigma^2}\|\mathbf{x} - \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{z})\|^2\right] + \mathcal{D}_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p_{\mathrm{prior}}(\mathbf{z})),$$

where the KL term admits a closed-form solution due to the Gaussian assumption. Training the VAE therefore reduces to minimizing a regularized reconstruction loss.

2.1.4 Drawbacks of Standard VAE

Despite the theoretical appeal of the VAE framework, it suffers from a critical drawback: it often produces blurry outputs[1].

[1] The limitation we study here is specific to the common VAE setup introduced above: a simple Gaussian decoder trained through a squared reconstruction error. In this setting, the population-

Blurry Generations in VAEs. To understand this phenomenon, consider a fixed Gaussian encoder $q_{\text{enc}}(\mathbf{z}|\mathbf{x})$, and a decoder of the form

$$p_{\text{dec}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}(\mathbf{z}), \sigma^2\mathbf{I}),$$

where $\boldsymbol{\mu}(\mathbf{z})$ denotes the decoder network. With an arbitrary encoder, optimizing the ELBO reduces (up to an additive constant) to minimizing the expected reconstruction error:

$$\begin{aligned}
&\arg\min_{\boldsymbol{\mu}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\text{enc}}(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x} - \boldsymbol{\mu}(\mathbf{z})\|^2\right] \\
&\quad = \arg\min_{\boldsymbol{\mu}} \mathbb{E}_{q_{\text{enc}}(\mathbf{z})}\mathbb{E}_{q_{\text{enc}}(\mathbf{x}|\mathbf{z})}\left[\|\mathbf{x} - \boldsymbol{\mu}(\mathbf{z})\|^2\right] \\
&\quad = \mathbb{E}_{q_{\text{enc}}(\mathbf{z})}\Big[\arg\min_{\boldsymbol{\mu}(\mathbf{z})} \mathbb{E}_{q_{\text{enc}}(\mathbf{x}|\mathbf{z})}\left[\|\mathbf{x} - \boldsymbol{\mu}(\mathbf{z})\|^2\right]\Big].
\end{aligned}$$

This is a standard least squares problem in $\boldsymbol{\mu}(\mathbf{z})$, and its solution is given in closed form by the conditional mean:

$$\boldsymbol{\mu}^*(\mathbf{z}) = \mathbb{E}_{q_{\text{enc}}(\mathbf{x}|\mathbf{z})}[\mathbf{x}],$$

where $q_{\text{enc}}(\mathbf{x}|\mathbf{z})$ is the encoder-induced posterior on inputs given latents, defined via Bayes' rule:

$$q_{\text{enc}}(\mathbf{x}|\mathbf{z}) = \frac{q_{\text{enc}}(\mathbf{z}|\mathbf{x})\, p_{\text{data}}(\mathbf{x})}{q_{\text{enc}}(\mathbf{z})}, \qquad q_{\text{enc}}(\mathbf{z}) := \int q_{\text{enc}}(\mathbf{z}|\mathbf{x})\, p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

An equivalent form of the optimal generator via Bayes' rule is:

$$\begin{aligned}
\boldsymbol{\mu}^*(\mathbf{z}) &= \mathbb{E}_{q_{\text{enc}}(\mathbf{x}|\mathbf{z})}[\mathbf{x}] = \int \mathbf{x}\, \frac{q_{\text{enc}}(\mathbf{z}|\mathbf{x})\, p_{\text{data}}(\mathbf{x})}{q_{\text{enc}}(\mathbf{z})}\, \mathrm{d}\mathbf{x} \\
&= \frac{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\text{enc}}(\mathbf{z}|\mathbf{x}) \cdot \mathbf{x}]}{q_{\text{enc}}(\mathbf{z})} = \frac{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\text{enc}}(\mathbf{z}|\mathbf{x}) \cdot \mathbf{x}]}{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\text{enc}}(\mathbf{z}|\mathbf{x})]}.
\end{aligned}$$

Now suppose that two distinct inputs $\mathbf{x} \neq \mathbf{x}'$ are mapped to overlapping regions in latent space, i.e., the supports of $q_{\text{enc}}(\cdot|\mathbf{x})$ and $q_{\text{enc}}(\cdot|\mathbf{x}')$ intersect. Then $\boldsymbol{\mu}^*(\mathbf{z})$ averages over multiple, potentially unrelated inputs, which leads to blurry, non-distinct outputs. This averaging effect over conflicting modes is a fundamental reason for the characteristic blurriness in VAE-generated samples.

optimal decoder returns a conditional mean. If several visually different inputs correspond to overlapping latent regions, this averaging can appear as blur. This should not be read as a universal statement about every VAE architecture or observation model.

2.1.5 (Optional) From Standard VAE to Hierarchical VAEs

To model complex data, Hierarchical Variational Autoencoders (HVAEs) (Vahdat and Kautz, 2020) enhance VAEs by introducing a hierarchy of latent variables. This deep, layered structure allows the model to capture data features at multiple levels of abstraction, significantly boosting expressive power and mirroring the compositional nature of real-world data.

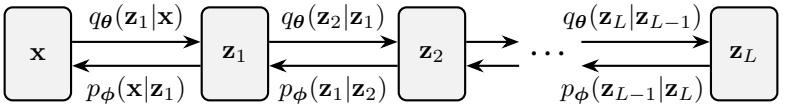


Figure 2.2: Computation graph of the HVAE. It has a hierarchical structure with stacked, trainable encoders and decoders across multiple latent layers. (Note the Markov conditioning structure.)

Source: Created by the authors.

HVAE's Modeling. Unlike standard VAEs that use a single latent code $\mathbf{z}$, hierarchical VAEs (HVAEs) introduce multiple layers of latent variables arranged in a top-down hierarchy. Each latent layer conditions the one below it, forming a chain of conditional priors that captures structure at progressively finer levels of abstraction. This leads to the following top-down factorization of the joint distribution:

$$p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}_1) \prod_{i=2}^{L} p_{\boldsymbol{\phi}}(\mathbf{z}_{i-1}|\mathbf{z}_i) p(\mathbf{z}_L).$$

This structure defines the marginal data distribution,

$$p_{\text{HVAE}}(\mathbf{x}) := \int p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L}) \, \mathrm{d}\mathbf{z}_{1:L}.$$

Generation proceeds progressively: starting from the top latent variable $\mathbf{z}_L$, each latent is decoded sequentially down to $\mathbf{z}_1$, followed by generating the final observation $\mathbf{x}$.

For encoding part, HVAEs utilize a structured, learnable variational encoder $q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})$ that mirrors the generative hierarchy. A common choice is a bottom-up Markov factorization:

$$q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x}) = q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x}) \prod_{i=2}^{L} q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1}).$$

HVAE's ELBO. Similar to Equation (2.1.1), ELBO is derived via Jensen's inequality:

$$\begin{aligned}\log p_{\text{HVAE}}(\mathbf{x}) &= \log \int p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L})\, \mathrm{d}\mathbf{z}_{1:L} \\ &= \log \int \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})} q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})\, \mathrm{d}\mathbf{z}_{1:L} \\ &= \log \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}\right] \\ &\geq \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}\right] \\ &=: \mathcal{L}_{\text{ELBO}}(\boldsymbol{\phi}).\end{aligned} \tag{2.1.3}$$

Substituting the factorized forms yields:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}\left[\log \frac{p(\mathbf{z}_L)\prod_{i=2}^{L} p_{\boldsymbol{\phi}}(\mathbf{z}_{i-1}|\mathbf{z}_i) p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}_1)}{q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})\prod_{i=2}^{L} q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})}\right].$$

This hierarchical ELBO decomposes into interpretable terms, including a reconstruction term, adjacent-layer matching terms, and a top-level KL regularizer to the prior.

The leap from shallow to deep networks revolutionized machine learning, and a similar idea transformed generative models. HVAEs showed the power of using deep, stacked layers to build data. This concept of a layered hierarchy is a cornerstone of modern generative modeling, appearing again in score-based methods (Section 3.4) and normalizing flows (Section 5.1). The core insight is simple yet powerful:

Observation 2.1.1:

Stacking layers allows the model to generate data progressively, starting with coarse details and adding finer ones at each step. This process makes it far easier to capture the complex structure of high-dimensional data.

Why Deeper Networks in a Flat VAE are Not Enough. There are two fundamental limitations of a standard flat VAE that are not resolved by simply making the encoder and decoder deeper.

The first limitation is the variational family (unimodal). In a standard VAE,

$$q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z};\, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}), \mathrm{diag}(\sigma_{\theta}^2(\mathbf{x}))),$$

so for each fixed $\mathbf{x}$ the encoder posterior is a single Gaussian with diagonal covariance. Greater network depth improves the accuracy of $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\sigma_{\boldsymbol{\theta}}$ but does not expand the family; even a full covariance remains one unimodal ellipsoid. When $p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is multi-peaked, this family cannot match it, which loosens the ELBO and

weakens inference. Addressing this requires a richer posterior class, not merely deeper networks.

Second, if the decoder is too expressive, the model may suffer from posterior collapse. To see why, let us recall that the objective of the VAE is

$$\begin{aligned}
&\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathcal{L}_{\text{ELBO}}(\mathbf{x})] \\
&\quad = \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))\right] \\
&\quad = \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})] - \int q(\mathbf{x},\mathbf{z}) \log \frac{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{z} \\
&\quad = \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})] - \int q(\mathbf{x},\mathbf{z}) \log \frac{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\theta}}(\mathbf{z})}\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{z} \\
&\qquad\qquad - \int q(\mathbf{x},\mathbf{z}) \log \frac{q_{\boldsymbol{\theta}}(\mathbf{z})}{p(\mathbf{z})}\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{z} \\
&\quad = \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})] - \mathcal{I}_q(\mathbf{x};\mathbf{z}) - \mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z})\|p(\mathbf{z})),
\end{aligned}$$

where $\mathcal{I}_q(\mathbf{x};\mathbf{z})$ is the mutual information defined by

$$\mathcal{I}_q(\mathbf{x};\mathbf{z}) = \mathbb{E}_{q(\mathbf{x},\mathbf{z})}\left[\log \tfrac{q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|q(\mathbf{z}))\right],$$

and the aggregated posterior is $q_{\boldsymbol{\theta}}(\mathbf{z}) = \int p_{\text{data}}(\mathbf{x}) q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\, \mathrm{d}\mathbf{x}$.

If the decoder class can model the data well without using $\mathbf{z}$ (i.e., it contains some $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}) = r(\mathbf{x})$ close to p_{data}), then a maximizer of the ELBO sets $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$, so $\mathcal{I}_q(\mathbf{x};\mathbf{z}) = 0$ and $q_{\boldsymbol{\theta}}(\mathbf{z}) = p(\mathbf{z})$. This "ignore $\mathbf{z}$" solution does not disappear by making the networks deeper: (1) the learned code becomes independent of $\mathbf{x}$ (so it carries no data-dependent structure useful for downstream tasks), and (2) conditioning or moving in $\mathbf{z}$ has no effect on generated samples, so controllable generation fails.

What Hierarchy Changes? An HVAE introduces multiple latent levels,

$$p_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}_1) \prod_{i=2}^{L} p_{\boldsymbol{\phi}}(\mathbf{z}_{i-1}|\mathbf{z}_i) p(\mathbf{z}_L),$$

with ELBO

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\mathbf{x}) = &\ \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}_1)] \\
&- \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:2}|\mathbf{x})}\left[\log q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x}) - \log p_{\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{z}_2)\right] \\
&- \sum_{i=2}^{L-1} \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{1:i+1}|\mathbf{x})}\left[\log q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1}) - \log p_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{z}_{i+1})\right] \\
&- \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}_{L-1}|\mathbf{x})}\left[\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z}_L|\mathbf{z}_{L-1})\|p(\mathbf{z}_L))\right].
\end{aligned}$$

Here, we denote $\mathbb{E}_q := \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})}$. Each latent level interacts only with its neighboring levels in the hierarchy: the encoder passes information upward through $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$, while the decoder passes information downward through $p_{\boldsymbol{\phi}}(\mathbf{z}_{i-1}|\mathbf{z}_i)$. Accordingly, the ELBO decomposes into a reconstruction term, adjacent-layer matching terms, and a top-level KL regularizer to the prior. These properties stem from the hierarchical latent graph, not from simply deepening networks in a flat VAE.

What Will be Ahead? While HVAEs extend the VAE framework with multiple latent layers for expressiveness, their training poses unique challenges. Because the encoder and decoder must be optimized jointly, learning becomes unstable: lower layers and the decoder can already reconstruct $\mathbf{x}$, leaving higher-level latents with little effective signal. Moreover, gradient information reaching deeper variables is often indirect and weak, making it difficult for them to contribute meaningfully. An additional difficulty lies in balancing model capacity, since overly expressive conditionals can dominate the reconstruction task and suppress the utility of higher latents.

Interestingly, the core idea of a deep, layered hierarchy finds a more powerful incarnation in variational diffusion models, a topic we explore in Section 2.2. Diffusion models inherit the progressive structure of HVAEs but elegantly sidestep their central weakness. By fixing the encoding process and focusing solely on learning the generative reversal, they unlock newfound stability and modeling flexibility, leading to a significant leap in the quality of generated outputs.

For notational simplicity, we deviate from the common VAE convention that uses q for the encoder and p for the generator. To avoid ambiguity, we denote distributions as p and will always specify their roles through appropriate subscripts or superscripts, clarifying them in context.

2.2 Variational Perspective: DDPM

Denoising Diffusion Probabilistic Models (DDPMs) (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020) represent a cornerstone of diffusion modeling. Conceptually, they operate within a variational framework, much like VAEs and HVAEs. However, DDPMs introduce a clever twist that tackles some of the challenges faced by their predecessors.

At their core, DDPMs involve two distinct stochastic processes:

- **The Forward Pass (Fixed Encoder):** This process gradually corrupts data by injecting Gaussian noise over multiple steps via a transition kernel $p(\mathbf{x}_i|\mathbf{x}_{i-1})$. The data evolves into an isotropic Gaussian distribution, effectively becoming pure noise. This means the encoder is fixed and not learned.
- **The Reverse Denoising Process (Learnable Decoder):** Here, a neural network learns to reverse the noise corruption through a parameterized distribution $p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$. Starting from pure noise, this process iteratively denoises to generate realistic samples. Crucially, each individual denoising step is a more manageable task than generating a complete sample from scratch, as VAEs often attempt to do.

By fixing the encoder and concentrating learning on the gradual generative trajectory, DDPMs achieve remarkable stability and expressive power.

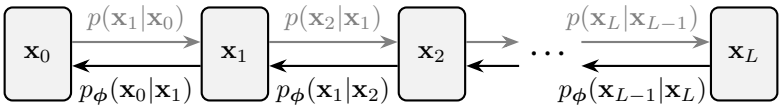


Figure 2.3: Illustration of DDPM. It consists of a fixed forward process (in gray) that gradually adds Gaussian noise to the data, and a learned reverse process that denoises step-by-step to generate new samples.

Source: Created by the authors.

In this section, we focus on DDPMs, postponing the broader discussion to Section 4.5, where we present a more general and flexible framework.

2.2.1 Forward Process (Fixed Encoder)

In DDPMs, the forward process is a fixed, non-trainable operation that serves as an encoder. It progressively corrupts the original data by adding noise over multiple steps, eventually transforming it into a simple prior distribution $p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$. This transformation is depicted as the forward chain in Figure 2.3 or as illustrated in Figure 2.4.

Let us formalize this step-by-step degradation:

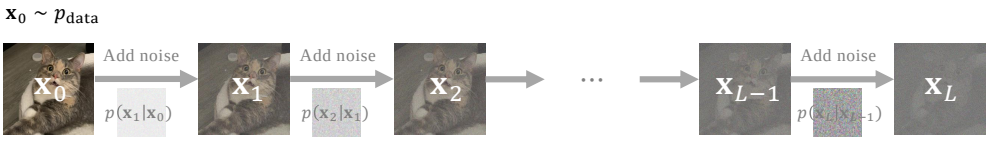


Figure 2.4: Illustration of the DDPM forward process, wherein Gaussian noise is incrementally added to corrupt a data sample into pure noise.

Source: Created by the authors.

Fixed Gaussian Transitions. Each step in the forward process is governed by a fixed Gaussian transition kernel[2]:

$$p(\mathbf{x}_i|\mathbf{x}_{i-1}) := \mathcal{N}(\mathbf{x}_i; \sqrt{1-\beta_i^2}\mathbf{x}_{i-1}, \beta_i^2\mathbf{I}).$$

Here, the process begins with $\mathbf{x}_0$, representing a sample drawn from the real data distribution p_{data}. The sequence $\{\beta_i\}_{i=1}^L$ denotes a pre-determined, monotonically increasing noise schedule, where each $\beta_i \in (0,1)$ controls the variance of the Gaussian noise injected at step i. For convenience, we define $\alpha_i := \sqrt{1-\beta_i^2}$. Geometrically, α_i traces the quarter circle $y = \sqrt{1-x^2}$: as β_i grows toward 1, α_i shrinks to 0. This mathematical definition is precisely equivalent to the following intuitive iterative update:

$$\mathbf{x}_i = \alpha_i \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i,$$

where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are independently and identically distributed. This means at each step i, we scale down the previous state $\mathbf{x}_{i-1}$ by α_i and add a controlled amount of Gaussian noise scaled by β_i. Why are the two forms equivalent? Conditioned on $\mathbf{x}_{i-1}$: $\mathbb{E}[\mathbf{x}_i|\mathbf{x}_{i-1}] = \alpha_i\mathbf{x}_{i-1} + \beta_i\,\mathbb{E}[\boldsymbol{\epsilon}_i] = \alpha_i\mathbf{x}_{i-1}$ and $\text{Var}[\mathbf{x}_i|\mathbf{x}_{i-1}] = \beta_i^2\,\text{Var}[\boldsymbol{\epsilon}_i] = \beta_i^2\mathbf{I}$, matching the kernel.

Perturbation Kernel and Prior Distribution. By recursively applying the transition kernels, we obtain a closed-form expression for the distribution of noisy samples at step i given the original data $\mathbf{x}_0$:

$$p_i(\mathbf{x}_i|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_i; \bar{\alpha}_i\mathbf{x}_0, (1-\bar{\alpha}_i^2)\mathbf{I}\right),$$

where

$$\bar{\alpha}_i := \prod_{k=1}^{i}\sqrt{1-\beta_k^2} = \prod_{k=1}^{i}\alpha_k.$$

[2]This formulation, while potentially appearing different, is mathematically equivalent to the original DDPM transition kernel.

Proof by induction. The base case ($i = 1$) is the transition kernel itself. For the inductive step, assume $p_i(\mathbf{x}_i|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_i; \bar{\alpha}_i\mathbf{x}_0, (1-\bar{\alpha}_i^2)\mathbf{I})$. By the Markov chain, reparameterizing $p(\mathbf{x}_{i+1}|\mathbf{x}_i)$ and $p_i(\mathbf{x}_i|\mathbf{x}_0)$,

$$\begin{aligned}
\mathbf{x}_{i+1} &= \alpha_{i+1}(\bar{\alpha}_i\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\,\boldsymbol{\epsilon}_i) + \beta_{i+1}\boldsymbol{\epsilon}_{i+1} && (\boldsymbol{\epsilon}_i \perp \boldsymbol{\epsilon}_{i+1}) \\
&= \bar{\alpha}_{i+1}\mathbf{x}_0 + \sqrt{\alpha_{i+1}^2(1-\bar{\alpha}_i^2) + \beta_{i+1}^2}\,\boldsymbol{\epsilon} && (\text{sum of independent Gaussians}) \\
&= \bar{\alpha}_{i+1}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{i+1}^2}\,\boldsymbol{\epsilon} && (\alpha_{i+1}^2 - \bar{\alpha}_{i+1}^2 + \beta_{i+1}^2 = 1 - \bar{\alpha}_{i+1}^2),
\end{aligned}$$

i.e., $p_{i+1}(\mathbf{x}_{i+1}|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{i+1}; \bar{\alpha}_{i+1}\mathbf{x}_0, (1-\bar{\alpha}_{i+1}^2)\mathbf{I})$. ■

This means we can sample $\mathbf{x}_i$ directly from $\mathbf{x}$ as

$$\mathbf{x}_i = \bar{\alpha}_i\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.2.1}$$

Let the noise schedule $\{\beta_i\}_{i=1}^L$ be an increasing sequence, then the marginal distribution of the forward process converges as

$$p_L(\mathbf{x}_L|\mathbf{x}_0) \longrightarrow \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \text{as } L \to \infty,$$

which motivates the choice of the prior distribution as

$$p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$$

with no reliance on data $\mathbf{x}_0$.

2.2.2 Reverse Denoising Process (Learnable Decoder)

At its core, the essence of DDPMs lies in their ability to *reverse* the controlled degradation imposed by the forward diffusion process. Starting from pure, unstructured noise, $\mathbf{x}_L \sim p_{\text{prior}}$, the objective is to progressively *denoise* this randomness, step by step, until a coherent and meaningful data sample emerges. This reverse generation proceeds through a Markov chain, illustrated by Figure 2.5.

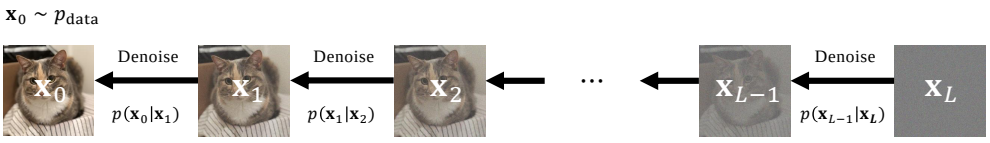


Figure 2.5: Illustration of DDPM reverse (denoising) process. Starting from noise $\mathbf{x}_L \sim p_{\text{prior}}$, the model sequentially samples $\mathbf{x}_{i-1} \sim p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ for $i = L, \ldots, 1$ to obtain a newly generated data $\mathbf{x}_0$. The oracle transition $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is unknown; thus, we aim to approximate it.

Source: Created by the authors.

The fundamental challenge, and the central question guiding DDPM development, then becomes:

Question 2.2.1

Can we precisely compute, or at least effectively approximate, these reverse transition kernels $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$, especially when considering the complex distribution of $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$?

Rather than diving immediately into the mathematically intricate derivation of the Evidence Lower Bound (ELBO), as the original DDPM paper does (for which a detailed discussion awaits in Section 2.2.5), we will instead approach the training objective from a more intuitive perspective: by leveraging conditional probabilities to achieve a tractable formulation.

Overview: Modeling and Training Objective. To enable the generative process, our goal is to approximate the unknown true reverse transition kernel, $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$. We achieve this by introducing a learnable parametric model, $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$, and training it to minimize the expected KL divergence:

$$\mathbb{E}_{p_i(\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))\right]. \tag{2.2.2}$$

However, a direct computation of the target distribution $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is challenging. By Bayes' theorem, we would need to evaluate:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i) = p(\mathbf{x}_i|\mathbf{x}_{i-1}) \underbrace{\frac{p_{i-1}(\mathbf{x}_{i-1})}{p_i(\mathbf{x}_i)}}_{\text{intractable}}.$$

The marginals $p_i(\mathbf{x}_i)$ and $p_{i-1}(\mathbf{x}_{i-1})$ are expectations over the unknown data distribution p_{data}, given by:

$$p_i(\mathbf{x}_i) = \int p_i(\mathbf{x}_i|\mathbf{x}_0)p_{\mathrm{data}}(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0,$$

and analogously for $p_{i-1}(\mathbf{x}_{i-1})$. Since p_{data} is unknown, these integrals have no closed-form evaluation; at best they can be approximated from samples, so the exact densities are not available in practice.

Overcoming Intractability with Conditioning. A central insight in DDPMs resolves this intractability: we condition the reverse transition on a clean data sample $\mathbf{x}_0$. This subtle yet powerful step transforms the intractable kernel into one that is mathematically tractable:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) = p(\mathbf{x}_i|\mathbf{x}_{i-1})\frac{p(\mathbf{x}_{i-1}|\mathbf{x}_0)}{p_i(\mathbf{x}_i|\mathbf{x}_0)}.$$

This tractability arises from two key properties of the forward process: its Markov property, meaning $p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{x}_0) = p(\mathbf{x}_i|\mathbf{x}_{i-1})$, and the Gaussian nature of all

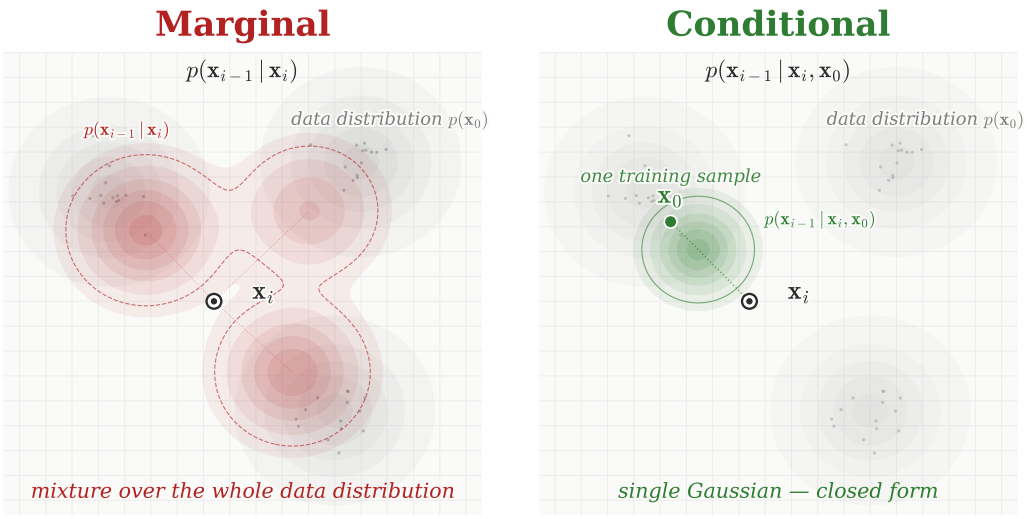


Figure 2.6: The conditioning trick in DDPM. (Left) For a fixed noisy sample $\mathbf{x}_i$, the reverse distribution $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ averages over all possible clean samples $\mathbf{x}_0$ that could have produced $\mathbf{x}_i$. It can therefore be a complicated mixture. **(Right)** During training, however, the clean sample $\mathbf{x}_0$ used to generate $\mathbf{x}_i$ is known. Conditioning on this sample gives $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)$, a Gaussian with closed-form mean and variance. DDPM can therefore use this tractable conditional distribution as a simple regression target. Although each training example conditions on only one clean sample, averaging the conditional training objective over all samples gives the same optimal model as directly learning the marginal reverse distribution; see Theorem 2.2.1. Thus, DDPM can learn the same optimal reverse prediction without directly regressing onto the complicated, generally intractable target induced by the marginal mixture.

Source: Created by the authors with AI-assisted coding.

involved distributions. As a result, $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)$ itself is Gaussian and admits a closed-form expression (which we will see in Equation (2.2.4)). We visualize it in Figure 2.6. Crucially, this elegant conditioning strategy allows us to derive a tractable objective that is functionally equivalent to the seemingly intractable marginal KL divergence in Equation (2.2.2).

Theorem 2.2.1: Equivalence Between Marginal and Conditional KL Minimization

The following equality holds:

$$\begin{aligned} &\mathbb{E}_{p_i(\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i)\|p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i))\right] \\ &= \mathbb{E}_{p_{\mathrm{data}}(\mathbf{x}_0)}\mathbb{E}_{p_i(\mathbf{x}_i|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\|p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i))\right] + C. \end{aligned} \tag{2.2.3}$$

where C is a constant independent of $\boldsymbol{\phi}$. Moreover, the minimizer of Equation (2.2.3) satisfies

$$p^*(\mathbf{x}_{i-1}|\mathbf{x}_i) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\right] = p(\mathbf{x}_{i-1}|\mathbf{x}_i), \quad \mathbf{x}_i \sim p_i.$$

Proof for Theorem.

The proof rewrites a KL-divergence expectation by expanding definitions, applying the chain rule of probability, and using a logarithmic identity to decompose it into the sum of an expected conditional KL divergence and a marginal KL divergence. A complete derivation is in Section D.1.1.

This alternative viewpoint: conditioning to obtain a tractable objective, forms the foundation of DDPMs and reveals a profound commonality with other influential diffusion models, as we will explore in Chapter 3 and Chapter 5.

It reveals a powerful equivalence: minimizing the KL divergence between marginal distributions is mathematically identical to minimizing the KL divergence between specific conditional distributions. This latter formulation is exceptionally useful because the crucial conditional distribution, $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$, possesses a convenient closed-form expression:

Lemma 2.2.2: Reverse Conditional Transition Kernel

$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$ is Gaussian with the closed-form expression:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}) = \mathcal{N}\left(\mathbf{x}_{i-1}; \boldsymbol{\mu}\left(\mathbf{x}_i, \mathbf{x}, i\right), \sigma^2(i)\mathbf{I}\right),$$

where

$$\boldsymbol{\mu}\left(\mathbf{x}_i, \mathbf{x}, i\right) := \frac{\bar{\alpha}_{i-1}\beta_i^2}{1-\bar{\alpha}_i^2}\mathbf{x} + \frac{(1-\bar{\alpha}_{i-1}^2)\alpha_i}{1-\bar{\alpha}_i^2}\mathbf{x}_i, \quad \sigma^2(i) := \frac{1-\bar{\alpha}_{i-1}^2}{1-\bar{\alpha}_i^2}\beta_i^2. \quad (2.2.4)$$

Later in Lemma 4.5.2, we present a more general formula that extends beyond the DDPM noising process described in Equation (2.2.1).

2.2.3 Modeling of Reverse Transition Kernel $p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i)$

Leveraging the gradient-level equivalence as in Theorem 2.2.1 and the Gaussian form of the reverse conditional $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$ as in Lemma 2.2.2, DDPM assumes that each reverse transition $p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is Gaussian, parameterized as

$$p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i) := \mathcal{N}(\mathbf{x}_{i-1}; \boldsymbol{\mu}_{\phi}(\mathbf{x}_i, i), \sigma^2(i)\mathbf{I}), \quad (2.2.5)$$

where $\boldsymbol{\mu}_{\phi}(\cdot, i)\colon \mathbb{R}^D \to \mathbb{R}^D$ is a learnable mean function, and $\sigma^2(i) > 0$ is fixed as defined in Equation (2.2.4).

We denote the KL divergence, averaged over time steps i and conditioned on data $\mathbf{x}_0 \sim p_{\text{data}}$, to match all layers of distributions as:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \boldsymbol{\phi}) := \sum_{i=2}^{L} \mathbb{E}_{p_i(\mathbf{x}_i|\mathbf{x}_0)}\left[\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\|p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i))\right]. \quad (2.2.6)$$

Thanks to the Gaussian forms of both distributions and the parameterization defined in Equation (2.2.5), the objective admits a closed-form expression and can be simplified as:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0;\boldsymbol{\phi}) = \sum_{i=2}^{L} \frac{1}{2\sigma^2(i)} \mathbb{E}_{p_i(\mathbf{x}_i|\mathbf{x}_0)} \left\|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\right\|_2^2 + C, \tag{2.2.7}$$

where C is a constant independent of $\boldsymbol{\phi}$. Averaging over the data distribution and omitting the constant C (which does not affect the optimization), the final DDPM training objective is

$$\mathcal{L}_{\text{DDPM}}(\boldsymbol{\phi}) := \sum_{i=2}^{L} \frac{1}{2\sigma^2(i)} \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p_i(\mathbf{x}_i|\mathbf{x}_0)} \left[\left\|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\right\|_2^2\right], \tag{2.2.8}$$

where $\mathbf{x}_0 \sim p_{\text{data}}$.

We remark that the forward perturbation kernel $p_i(\mathbf{x}_i|\mathbf{x}_0)$ plays an important computational role in the training objective. It gives the distribution of the noisy state $\mathbf{x}_i$ obtained from a clean sample $\mathbf{x}_0$ after i forward steps. Because each forward transition is linear and Gaussian, their composition also has a closed-form Gaussian distribution; see the discussion around Equation (2.2.1). Consequently, after selecting a time step i, we can sample $\mathbf{x}_i \sim p_i(\mathbf{x}_i|\mathbf{x}_0)$ directly from $\mathbf{x}_0$, without first generating the intermediate states $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$. Thus, each loss term can be estimated using only a clean sample, a selected time step, and a directly corrupted sample, making stochastic training computationally efficient.

2.2.4 Practical Choices of Predictions and Loss

$\boldsymbol{\epsilon}$-Prediction. In typical DDPM implementations, training is not conducted directly using the original loss based on the *mean prediction* parameterization from Equation (2.2.8). Instead, an equivalent reparameterization, known as the *$\boldsymbol{\epsilon}$-prediction* (noise prediction) formulation, is commonly adopted.

Recall that in the DDPM forward process, a noisy sample $\mathbf{x}_i \sim p(\mathbf{x}_i|\mathbf{x}_0)$ at noise level i is generated by

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\boldsymbol{\epsilon}, \quad \mathbf{x}_0 \sim p_{\text{data}}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.2.9}$$

Using this expression, the reverse mean $\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)$ from Equation (2.2.4) can be rewritten as:

$$\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i) = \frac{1}{\alpha_i}\left(\mathbf{x}_i - \frac{1-\alpha_i^2}{\sqrt{1-\bar{\alpha}_i^2}}\boldsymbol{\epsilon}\right).$$

(To see this, solve Equation (2.2.9) for $\mathbf{x}_0$, substitute into Equation (2.2.4), and simplify.)

This motivates a parameterization of the model mean $\boldsymbol{\mu}_{\boldsymbol{\phi}}$ using a neural network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ that directly predicts the noise:

$$\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) = \frac{1}{\alpha_i}\left(\mathbf{x}_i - \frac{1-\alpha_i^2}{\sqrt{1-\bar{\alpha}_i^2}}\underbrace{\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)}_{\boldsymbol{\epsilon}\text{-prediction}}\right).$$

Substituting this into the original loss leads to a squared ℓ_2 error between predicted and true noise:

$$\|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 \propto \|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2,$$

up to a weighting factor depending on i. Intuitively, the model acts as a "noise detective", estimating the random noise added at each step of the forward process. Subtracting this estimate from the corrupted sample moves it closer to the clean original, and repeating this step-by-step reconstructs the data from pure noise.

Simplified Loss with $\boldsymbol{\epsilon}$-Prediction. In practice, this expression is further simplified by omitting the weighting term, yielding the widely used DDPM training loss:

$$\mathcal{L}_{\text{simple}}(\boldsymbol{\phi}) := \mathbb{E}_i \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2\right], \tag{2.2.10}$$

where $\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\boldsymbol{\epsilon}$ with $\mathbf{x}_0 \sim p_{\text{data}}$. Since the target noise has unit variance at every timestep t, the ℓ_2 loss in Equation (2.2.10) maintains a consistent scale across all t. This prevents vanishing or exploding targets and eliminates the need for explicit loss weighting.

Importantly, both $\mathcal{L}_{\text{DDPM}}$ and $\mathcal{L}_{\text{simple}}$ share the same optimal solution $\boldsymbol{\epsilon}^*$, this is because Equation (2.2.10) essentially reduces to a least-squares problem (as shown similarly in Proposition 4.3.1 or Proposition 6.3.1)[3]:

$$\boldsymbol{\epsilon}^*(\mathbf{x}_i, i) = \mathbb{E}\left[\boldsymbol{\epsilon}|\mathbf{x}_i\right], \quad \mathbf{x}_i \sim p_i.$$

Crucially, this is only possible because the network is conditioned on the noise level i. The weights in $\mathcal{L}_{\text{DDPM}}$ rescale the loss at each level i, but since $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ is free to make a separate prediction for every i, the minimizer at each level is the same conditional expectation $\mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_i]$ no matter how that level is weighted. Intuitively, the $\boldsymbol{\epsilon}$-prediction network $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ estimates the noise added by the forward process to produce $\mathbf{x}_i$. At optimality, this estimate coincides with the conditional expectation of the true noise, even though $\mathbf{x}_i$ does not uniquely determine the original clean sample.

[3] We remark that removing the ELBO-derived weights in $\mathcal{L}_{\text{DDPM}}$ changes the training objective, but not its population-optimal prediction: $\boldsymbol{\epsilon}^*(\mathbf{x}_i, i) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_i]$. Thus, the weighted and simplified DDPM losses have the same ideal predictor, but they emphasize different noise levels during training. In practice, this weighting can still affect optimization and performance.

Another Equivalent Parametrization: x-Prediction. Equation (2.2.4) motivates an alternative yet equivalent parameterization, known as **x**-*prediction* (clean prediction), in which a neural network $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ is trained to predict a clean (denoised) sample from a given noisy input $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$ at noise level i. Replacing the ground-truth clean sample $\mathbf{x}_0$ in the reverse mean expression with $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ leads to the following model:

$$\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) = \frac{\bar{\alpha}_{i-1}\beta_i^2}{1-\bar{\alpha}_i^2}\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) + \frac{(1-\bar{\alpha}_{i-1}^2)\alpha_i}{1-\bar{\alpha}_i^2}\mathbf{x}_i.$$

Analogous to the $\boldsymbol{\epsilon}$-prediction formulation, the training objective can be expressed as

$$\|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 \propto \|\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \mathbf{x}_0\|_2^2, \quad \mathbf{x}_0 \sim p_{\text{data}},$$

where the model is trained to predict the original data sample $\mathbf{x}_0$ from its noisy version $\mathbf{x}_i$. This equivalence reduces the mean-matching loss in Equation (2.2.8) to

$$\mathbb{E}_i \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\omega_i \left\| \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \mathbf{x}_0 \right\|_2^2 \right],$$

for some weighting function ω_i. Since this is a least-squares problem, the optimal solution is given by (see Proposition 4.3.1 or Proposition 6.3.1)

$$\mathbf{x}^*(\mathbf{x}_i, i) = \mathbb{E}\left[\mathbf{x}_0 | \mathbf{x}_i\right], \quad \mathbf{x}_i \sim p_i, \tag{2.2.11}$$

that is, the model should predict the expected clean data given a noisy observation $\mathbf{x}_i$ at timestep i.

The **x**-prediction and $\boldsymbol{\epsilon}$-prediction parameterizations are mathematically equivalent and connected via the forward process:

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) + \sqrt{1-\bar{\alpha}_i^2}\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i). \tag{2.2.12}$$

That is, one may either predict the clean sample $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$ or the noise $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i)$, such that their combination reproduces $\mathbf{x}_i$ under the forward noising process.

2.2.5 DDPM's ELBO

With the reverse transitions defined as in Equation (2.2.5), this leads to the definition of the joint generative distribution in DDPM as:

$$p_{\boldsymbol{\phi}}(\mathbf{x}_0, \mathbf{x}_{1:L}) := p_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_1) p_{\boldsymbol{\phi}}(\mathbf{x}_1|\mathbf{x}_2) \cdots p_{\boldsymbol{\phi}}(\mathbf{x}_{L-1}|\mathbf{x}_L) p_{\text{prior}}(\mathbf{x}_L),$$

and the marginal generative model for data is given by:

$$p_{\boldsymbol{\phi}}(\mathbf{x}_0) := \int p_{\boldsymbol{\phi}}(\mathbf{x}_0, \mathbf{x}_{1:L}) \, \mathrm{d}\mathbf{x}_{1:L}.$$

Indeed, DDPM training via Equation (2.2.6) can be rigorously grounded in maximum likelihood estimation (Equation (1.1.2)). Specifically, its objective forms

an ELBO, similar to those in VAEs and HVAEs from Section 2.1, which serves as a lower bound on the log-density:

Theorem 2.2.3: DDPM's ELBO

$$\begin{aligned} -\log p_{\boldsymbol{\phi}}(\mathbf{x}_0) &\leq -\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0;\boldsymbol{\phi}) \\ &:= \mathcal{L}_{\text{prior}}(\mathbf{x}_0) + \mathcal{L}_{\text{recon.}}(\mathbf{x}_0;\boldsymbol{\phi}) + \mathcal{L}_{\text{diffusion}}(\mathbf{x}_0;\boldsymbol{\phi}) \end{aligned} \tag{2.2.13}$$

Here, each component of losses are defined as:

$$\begin{aligned} \mathcal{L}_{\text{prior}}(\mathbf{x}_0) &:= \mathcal{D}_{\text{KL}}\Big(p(\mathbf{x}_L|\mathbf{x}_0)\|p_{\text{prior}}(\mathbf{x}_L)\Big) \\ \mathcal{L}_{\text{recon.}}(\mathbf{x}_0;\boldsymbol{\phi}) &:= \mathbb{E}_{p(\mathbf{x}_1|\mathbf{x}_0)}\left[-\log p_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_1)\right] \\ \mathcal{L}_{\text{diffusion}}(\mathbf{x}_0;\boldsymbol{\phi}) &= \sum_{i=2}^{L} \mathbb{E}_{p_i(\mathbf{x}_i|\mathbf{x}_0)}\left[\mathcal{D}_{\text{KL}}\Big(p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\|p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)\Big)\right]. \end{aligned}$$

Proof for Theorem.

The derivation applies Jensen's inequality, as in the HVAE/VAE ELBO (Equation (2.1.3)), with further simplifications. The detailed proof is deferred to Section D.1.2.

The ELBO $\mathcal{L}_{\text{ELBO}}$ consists of three terms:

- $\mathcal{L}_{\text{prior}}$ can be made negligible by choosing the noise schedule $\{\beta_i\}$ such that $p(\cdot|\mathbf{x}_0) \approx p_{\text{prior}}(\cdot)$.
- For $\mathcal{L}_{\text{recon.}}$, this can be approximated and optimized using a Monte Carlo estimate; see (Ho *et al.*, 2020; Kingma *et al.*, 2021) for practical implementations.
- $\mathcal{L}_{\text{diffusion}}$ (cf. Equation (2.2.6)) matches the reverse conditionals $p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ to $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ at all steps i.

The ELBO objective $\mathcal{L}_{\text{ELBO}}$ can also be interpreted through the lens of the Data Processing Inequality with latents $\mathbf{z} = \mathbf{x}_{1:L}$, as illustrated in Equation (2.1.2):

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x}_0)\|p_{\boldsymbol{\phi}}(\mathbf{x}_0)) \leq \mathcal{D}_{\text{KL}}\left(p(\mathbf{x}_0,\mathbf{x}_{1:L})\|p_{\boldsymbol{\phi}}(\mathbf{x}_0,\mathbf{x}_{1:L})\right),$$

where $p(\mathbf{x}_0,\mathbf{x}_{1:L}) := p_{\text{data}}(\mathbf{x}_0)p(\mathbf{x}_1|\mathbf{x}_0)p(\mathbf{x}_2|\mathbf{x}_1)\cdots p(\mathbf{x}_L|\mathbf{x}_{L-1})$ denotes the joint distribution along the forward process.

Remark.

Diffusion's variational view fits the hierarchical latent-variable picture in a particularly convenient way: the "encoder" is the fixed forward noising chain, so there is no learned inference network. The ELBO nevertheless does decompose

into per-step KL terms, together with the terminal-prior and reconstruction terms in Equation (2.2.13). What makes DDPM training especially tractable is that, under the linear–Gaussian forward process, these per-step KL terms can be rewritten as simple denoising regression problems with closed-form conditional targets. Thus the key simplification is not the absence of per-level KL terms, but the fact that the forward process is fixed and turns them into supervised denoising objectives that can be sampled efficiently.

2.2.6 Sampling

After training the $\boldsymbol{\epsilon}$-prediction model, $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)$[4], sampling is performed sequentially as illustrated in Figure 2.5, using the parametrized transition $p_{\boldsymbol{\phi}^\times}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ instead.

More specifically, starting from a random seed $\mathbf{x}_L \sim p_{\text{prior}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$, we recursively sample from $p_{\boldsymbol{\phi}^\times}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ following the update rule below for $i = L, L-1, \ldots, 1$:

$$\mathbf{x}_{i-1} \leftarrow \underbrace{\frac{1}{\alpha_i}\left(\mathbf{x}_i - \frac{1-\alpha_i^2}{\sqrt{1-\bar{\alpha}_i^2}}\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)\right)}_{\boldsymbol{\mu}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)} + \sigma(i)\boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.2.14}$$

This "denoising" process continues until $\mathbf{x}_0$ is obtained as the final clean generated sample.

Another Interpretation of DDPM Sampling. A useful way to interpret DDPM sampling is to rewrite the update rule so that the noise-level structure becomes explicit. Recall from Equation (2.2.12) that the predicted noise can be expressed in terms of the current sample $\mathbf{x}_i$ and the predicted clean sample $\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)$:

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i) = \frac{\mathbf{x}_i - \bar{\alpha}_i\, \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)}{\sqrt{1-\bar{\alpha}_i^2}}.$$

Substituting this into the DDPM sampling rule (Equation (2.2.14)) and rearranging, we obtain

$$\mathbf{x}_{i-1} \leftarrow \bar{\alpha}_{i-1}\, \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i) + \underbrace{\sqrt{1-\bar{\alpha}_{i-1}^2 - \tilde{\beta}_i^2}\, \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i) + \tilde{\beta}_i\, \boldsymbol{\epsilon}_i}_{\text{combined std.} = \sqrt{1-\bar{\alpha}_{i-1}^2}}, \tag{2.2.15}$$

[4] We use the symbol "×" to indicate that the model has been trained and is now frozen.

where $\tilde{\beta}_i^2 = \frac{(1-\bar{\alpha}_{i-1}^2)(1-\alpha_i^2)}{1-\bar{\alpha}_i^2}$ is the DDPM posterior variance[5]. Moreover, the residual coefficients satisfy

$$(1 - \bar{\alpha}_{i-1}^2 - \tilde{\beta}_i^2) + \tilde{\beta}_i^2 = 1 - \bar{\alpha}_{i-1}^2,$$

so the update has the same noise-level structure as the forward marginal at level $i-1$. This returns to the first operational sampling template introduced in Equation (1.4.1): predict the clean part and rewrite the sample at a lower noise level. DDPM carries out this rewrite stochastically, combining the noise direction estimated from $\mathbf{x}_i$ with a fresh Gaussian noise sample. A deterministic sampler follows the same basic idea, but removes the fresh randomness and adjusts the remaining noise coefficient accordingly. We will revisit this construction when studying diffusion samplers more systematically in Section 9.2.

Compare Equation (2.2.15) with the forward marginal $\mathbf{x}_{i-1} = \bar{\alpha}_{i-1}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{i-1}^2}\,\bar{\boldsymbol{\epsilon}}$. The structure is identical: signal coefficient $\bar{\alpha}_{i-1}$ and noise standard deviation $\sqrt{1-\bar{\alpha}_{i-1}^2}$, except that the unknown $\mathbf{x}_0$ is replaced by the network prediction $\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)$. In other words, each DDPM step *predicts the clean data and then re-noises it to exactly noise level $i-1$*. One can therefore view DDPM sampling as iterating two operations:

1. **Denoise.** Estimate the clean data $\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i, i)$ from the current noisy input $\mathbf{x}_i$.
2. **Re-noise.** Add back noise at the reduced level $i-1$ via Equation (2.2.15), producing a sample $\mathbf{x}_{i-1}$ whose noise level matches the forward process at step $i-1$.

However, even if $\mathbf{x}_{\boldsymbol{\phi}^\times}$ is trained as the optimal denoiser (i.e., the conditional expectation minimizer; see Equation (2.2.11)), it can only predict the *average* clean sample given $\mathbf{x}_i$. This limitation leads to blurry predictions, particularly at high noise levels, where recovering detailed structure from severely corrupted inputs becomes difficult. From this viewpoint, diffusion sampling moves from high to low noise and progressively refines an estimate of the clean signal. Early steps set the global structure, later steps add fine detail, and the sample becomes more realistic as the noise is removed.

Slow Sampling Speed of DDPM. Sampling in DDPMs (i.e., diffusion models) is inherently slow. In the original formulation and implementation, generating a sample typically requires around 1,000 denoising steps. This is because sampling proceeds through a long sequence of small refinements: at each step, the model updates the current noisy sample slightly toward a cleaner one, and the next

[5] The coefficient $\sqrt{1-\bar{\alpha}_{i-1}^2-\tilde{\beta}_i^2}$ simplifies to $\frac{\alpha_i(1-\bar{\alpha}_{i-1}^2)}{\sqrt{1-\bar{\alpha}_i^2}}$, which is always non-negative, so the square root is well-defined.

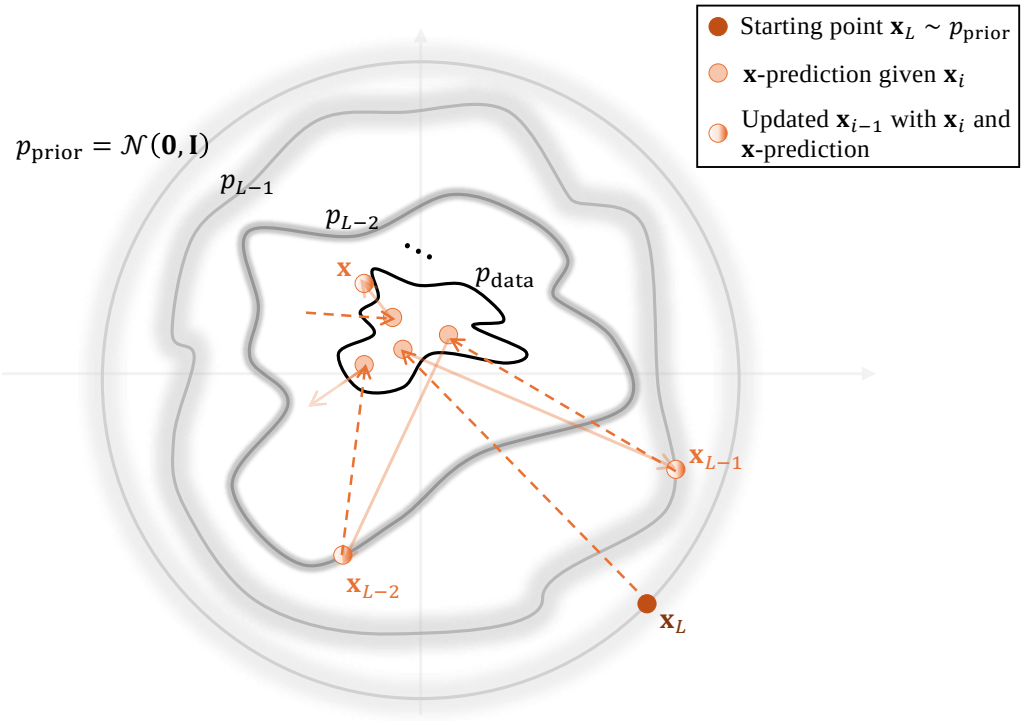


Figure 2.7: Illustration of the denoise-then-re-noise view of DDPM sampling. From the current noisy sample $\mathbf{x}_i$, the model first estimates the underlying clean signal $\mathbf{x}_{\phi^\times}(\mathbf{x}_i, i)$, and then samples a less noisy point $\mathbf{x}_{i-1}$ by re-noising this estimate to the $(i-1)$-th noise level.

Source: Created by the authors.

update must start from the result of the previous step. As a result, standard DDPM sampling is time-stepping and strongly sequential, and good sample quality usually requires many such steps to closely track the reverse diffusion path.

Theorem 2.2.1 shows that an expressive $p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ can theoretically match the true reverse distribution $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$. However, in practice, $p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is typically modeled as a Gaussian to approximate $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$, limiting its expressiveness.

For small forward noise scales β_i, the true reverse distribution is approximately Gaussian, enabling accurate approximation. Conversely, large β_i induce multi-modality or strong non-Gaussianity that a single Gaussian cannot capture. To maintain accuracy, DDPM employs many small β_i steps, forming a sequential chain where each step depends on the previous and requires a neural network evaluation $\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_i, i)$. This results in $\mathcal{O}(L)$ sequential passes, preventing parallelization and slowing generation.

Later in Chapter 4 we show a more principled interpretation of this inherent sampling bottleneck as a differential-equation problem, which motivates continuous-time numerical strategies for accelerating generation.

2.3 Closing Remarks

In this chapter, we have traced the origins of diffusion models through the variational lens. We began with the Variational Autoencoder (VAE), a foundational generative model that learns a probabilistic mapping between data and a structured latent space via the Evidence Lower Bound (ELBO). We saw how Hierarchical VAEs (HVAEs) extended this idea by stacking latent layers, introducing the powerful concept of progressive, coarse-to-fine generation. However, these models face challenges with training stability and sample quality.

We then framed Denoising Diffusion Probabilistic Models (DDPMs) as a pivotal evolution within this variational framework. By fixing the encoder to a gradual noising process and learning only the reverse denoising steps, DDPMs elegantly sidestep the training instabilities of HVAEs. Crucially, we demonstrated that DDPMs are also trained by maximizing a variational bound on the log-likelihood, with a training objective that decomposes into a series of simple denoising tasks. This tractability is enabled by a powerful conditioning strategy that transforms an intractable marginal objective into a tractable conditional one, a recurring theme in diffusion models.

While this variational framework provides a complete and powerful foundation for DDPMs, it is not the only way to understand them. An alternative and equally fundamental perspective emerges from the principles of energy-based modeling. In the next chapter, we will explore this score-based perspective:

1. We will shift our focus from learning the denoising transition probabilities $p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i)$ to directly learning the gradient of the data's log-density, i.e., the score function.

2. We will see how this approach, originating from EBMs, gives rise to Noise Conditional Score Networks (NCSN) and reveals a deep, mathematical equivalence between the noise prediction ($\boldsymbol{\epsilon}$-prediction) learned in DDPMs and the score function itself.

This alternative viewpoint will not only offer new insights but also serve as another cornerstone for the unified, continuous-time framework of diffusion models to be developed later.

3

Score-Based Perspective: From EBMs to NCSN

In the previous chapters we traced diffusion models to their variational roots and showed how they arise within the framework of VAEs. We now turn to a second, equally fundamental viewpoint: *Energy-Based Models* (EBMs) (Ackley *et al.*, 1985; LeCun *et al.*, 2006). An EBM represents a distribution by an energy landscape that is low on data and high elsewhere. Sampling typically relies on Langevin dynamics, which moves samples toward high density regions by following the gradient of this landscape. This gradient field, known as the *score*, points toward directions of higher probability.

The central observation is that knowing the score is enough for generation: it moves samples toward likely regions without computing the intractable normalization constant. *Score-based* diffusion models build directly on this idea. Instead of focusing only on the clean data distribution, they consider a sequence of Gaussian noise–perturbed distributions whose scores are easier to approximate. Learning these scores yields a family of vector fields that guide noisy samples step by step back to data, turning generation into progressive denoising.

After this chapter, you will be able to...

- understand the core idea of score-based modeling: what a score function is, and how it can be used with Langevin dynamics for generation;
- understand how denoising score matching learns the score at a single noise level through conditioning trick, and how Tweedie's formula connects score prediction to denoising;
- understand how Noise Conditional Score Networks (NCSNs) extend denoising score matching to multiple noise levels, why multiple noise levels are useful, and how NCSN score prediction relates to DDPM noise prediction.

Reader route. Readers already familiar with energy-based models may skim Section 3.1 and begin with Section 3.2.1.

3.1 Energy-Based Models

For readers already familiar with EBMs, this section is meant as a concise refresher and a bridge to the score-based view of diffusion.

3.1.1 Modeling Probability Distributions Using Energy Functions

Let $\mathbf{x} \in \mathbb{R}^D$ denote a data point. EBMs define a probability density via an energy function $E_{\boldsymbol{\phi}}(\mathbf{x})$, parameterized by $\boldsymbol{\phi}$, which assigns lower energy to more likely configurations. The resulting distribution is given by

$$p_{\boldsymbol{\phi}}(\mathbf{x}) := \frac{\exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))}{Z_{\boldsymbol{\phi}}}, \quad Z_{\boldsymbol{\phi}} := \int_{\mathbb{R}^D} \exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))\, \mathrm{d}\mathbf{x},$$

where $Z_{\boldsymbol{\phi}}$ is called the *partition function* ensuring normalization:

$$\int_{\mathbb{R}^D} p_{\boldsymbol{\phi}}(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1.$$

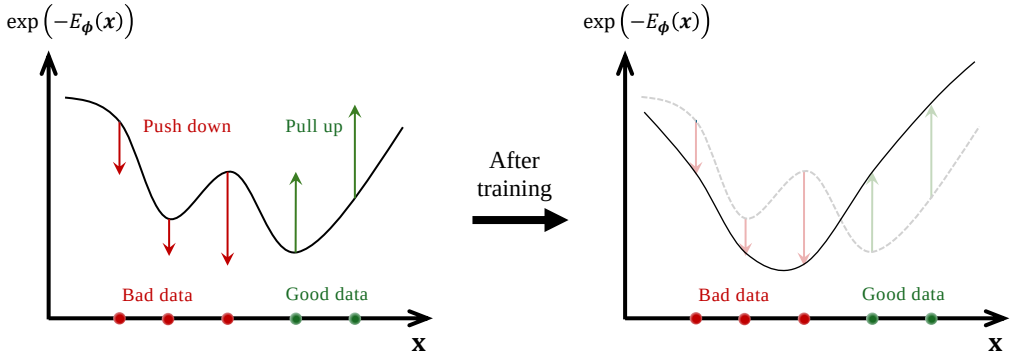


Figure 3.1: Illustration of EBM training. The model lowers density (raises energy) at "bad" data points (red arrows), and raises density (lowers energy) at "good" data points (green arrows).

Source: Created by the authors.

In this view, points with lower energy correspond to higher probability, much like a ball rolling down into a valley. The partition function $Z_{\boldsymbol{\phi}}$ ensures that all probabilities add up to one, and as a result only the *relative* values of energy matter. For instance, adding a constant to all energies multiplies both numerator and denominator by the same factor, leaving the distribution unchanged.

Moreover, because the partition function $Z_{\boldsymbol{\phi}}$ enforces that probabilities sum to one, it follows mathematically that decreasing the energy within a region increases its probability, while the probability of its complement decreases accordingly. Thus, EBMs obey a strict global trade-off: making one valley deeper inevitably makes others shallower, and probability mass is redistributed across the entire space rather than assigned independently to each region.

Challenges of Maximum Likelihood Training in EBMs. In principle, EBMs can be trained by maximum likelihood, which naturally balances fitting the data with global regularization (see Equation (1.1.2)):

$$\begin{aligned}\mathcal{L}_{\mathrm{MLE}}(\boldsymbol{\phi}) &= \mathbb{E}_{p_{\mathrm{data}}(\mathbf{x})}\left[\log \frac{\exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))}{Z_{\boldsymbol{\phi}}}\right] \\ &= -\underbrace{\mathbb{E}_{p_{\mathrm{data}}}[E_{\boldsymbol{\phi}}(\mathbf{x})]}_{\text{lowers energy of data}} \underbrace{-\log \int \exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x}}_{\text{global regularization}},\end{aligned} \tag{3.1.1}$$

with $Z_{\boldsymbol{\phi}} = \int \exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x}$. The first term lowers the energy of real data, while the second enforces normalization via the partition function.

However, in high dimensions computing $\log Z_{\boldsymbol{\phi}}$ and its gradient is intractable, as it requires expectations under the model distribution. This motivates alternative objectives that either approximate the term, such as contrastive divergence (Hinton, 2002), or avoid it altogether through *score matching*.

In what follows, we first introduce the notion of the score function in Section 3.1.2 and present score matching as a tractable training objective that bypasses the partition function in Section 3.1.3, and then discuss Langevin dynamics as a practical sampling method with score functions in Section 3.1.4.

3.1.2 Motivation: What Is the Score?

For a density $p(\mathbf{x})$ on $\mathbb{R}^D$, the *score function* is the gradient of the log-density:

$$\mathbf{s}(\mathbf{x}) := \nabla_{\mathbf{x}} \log p(\mathbf{x}), \qquad \mathbf{s}\colon \mathbb{R}^D \to \mathbb{R}^D.$$

Intuitively, the score forms a vector field that points toward regions of higher probability, providing a local guide to where the data is most likely to occur (see Figure 3.2).

Why Model Scores Instead of Densities? Modeling the score offers both theoretical and practical benefits:

1. Freedom from Normalization Constants. Many distributions are defined only up to an unnormalized density $\tilde{p}(\mathbf{x})$, e.g., $\exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))$ in EBMs:

$$p(\mathbf{x}) = \frac{\tilde{p}(\mathbf{x})}{Z}, \qquad Z = \int \tilde{p}(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

While computing Z is intractable, the score depends only on $\tilde{p}$:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z}_{=0} = \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}), \tag{3.1.2}$$

since Z is constant in $\mathbf{x}$. This bypasses the partition function entirely.

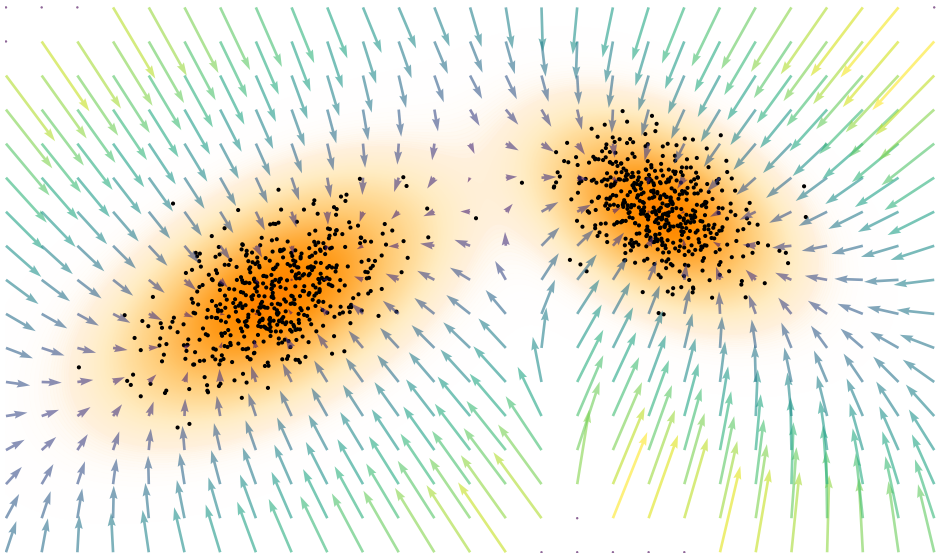


Figure 3.2: Illustration of score vector fields. Score vector fields $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ indicate directions of increasing density.

Source: Created by the authors.

2. A Complete Representation. The score function fully characterizes the underlying distribution. Since it is the gradient of the log-density, the density can be recovered (up to a constant) via

$$\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^1 \mathbf{s}(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0))^\top (\mathbf{x} - \mathbf{x}_0)\, \mathrm{d}t,$$

where $\mathbf{x}_0$ is a reference point and $\log p(\mathbf{x}_0)$ is fixed by normalization. Recall the path-integral definition $\int_\gamma \mathbf{f} \cdot \mathrm{d}\boldsymbol{\gamma} = \int_{t_0}^{t_1} \mathbf{f}(\boldsymbol{\gamma}(t))^\top \boldsymbol{\gamma}'(t)\, \mathrm{d}t$; here $\boldsymbol{\gamma}(t) = \mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)$, so $\boldsymbol{\gamma}'(t) = \mathbf{x} - \mathbf{x}_0$. Thus, modeling the score is as expressive as modeling $p(\mathbf{x})$ itself, while often more tractable for generative modeling.

3.1.3 Training EBMs via Score Matching

In EBMs, the density is defined as $p_{\boldsymbol{\phi}}(\mathbf{x}) = \frac{\exp(-E_{\boldsymbol{\phi}}(\mathbf{x}))}{Z_{\boldsymbol{\phi}}}$. Maximum likelihood training requires computing $Z_{\boldsymbol{\phi}}$, which is generally intractable. A key observation is that the model score of $p_{\boldsymbol{\phi}}$ simplifies to: $-\nabla_{\mathbf{x}} E_{\boldsymbol{\phi}}(\mathbf{x})$, independent of $Z_{\boldsymbol{\phi}}$ (see Equation (3.1.2)).

Score matching (Hyvärinen and Dayan, 2005) leverages the fact that scores depend only on the energy function. Instead of fitting normalized probabilities, it trains EBMs by aligning the model score with the (unknown) data score:

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) = \tfrac{1}{2} \mathbb{E}_{p_{\mathrm{data}}(\mathbf{x})} \left\| \nabla_{\mathbf{x}} \log p_{\boldsymbol{\phi}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}) \right\|_2^2. \tag{3.1.3}$$

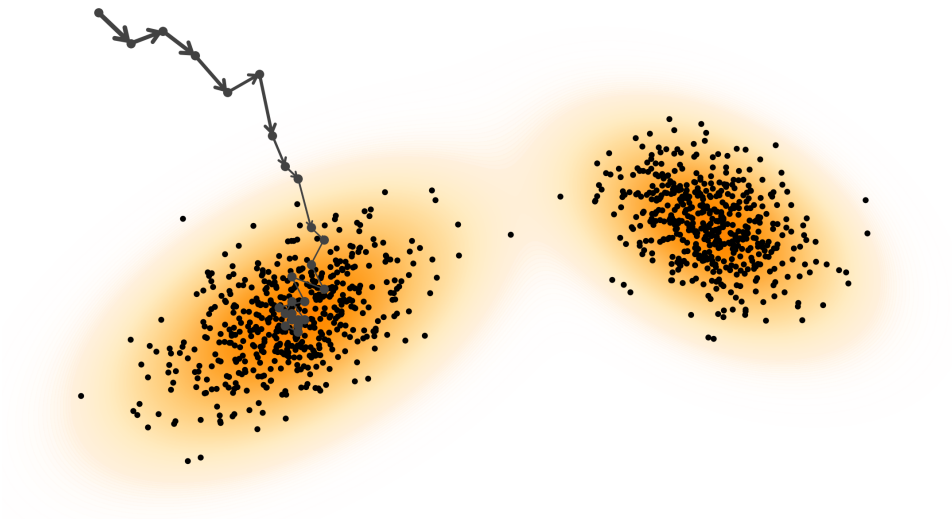


Figure 3.3: Illustration of Langevin sampling. Langevin sampling using the score function $\nabla_{\mathbf{x}} \log p_{\boldsymbol{\phi}}(\mathbf{x})$ to guide trajectories toward high-density regions via the update in Equation (3.1.5) (indicating by arrows).

Source: Created by the authors.

Although the data score is inaccessible, integration by parts yields an equivalent expression involving only the energy and its derivatives (see Proposition 3.2.1 for more details):

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) = \mathbb{E}_{p_{\mathrm{data}}(\mathbf{x})}\left[-\operatorname{Tr}\left(\nabla_{\mathbf{x}}^2 E_{\boldsymbol{\phi}}(\mathbf{x})\right) + \tfrac{1}{2}\|\nabla_{\mathbf{x}} E_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2\right] + C,$$

where $\nabla_{\mathbf{x}}^2 E_{\boldsymbol{\phi}}(\mathbf{x})$ is the Hessian of $E_{\boldsymbol{\phi}}$ and C is a constant independent of $\boldsymbol{\phi}$.

This formulation is attractive because it eliminates the partition function and avoids sampling from the model during training. Its main drawback is the need for second-order derivatives: although one only needs the trace of the Hessian ($\operatorname{Tr}\left(\nabla_{\mathbf{x}}^2 E_{\boldsymbol{\phi}}\right)$), rather than the full Hessian matrix, exact computation of this term generally requires aggregating second-derivative information across input coordinates, so its cost typically increases with the input dimension D. We will revisit approaches to addressing this limitation later in the chapter.

3.1.4 Langevin Sampling with Score Functions

Sampling from EBMs, defined by the energy function $E_{\boldsymbol{\phi}}(\mathbf{x})$, can be performed using *Langevin dynamics*. We first present the discrete-time Langevin update and then its continuous-time limit as a stochastic differential equation (SDE). Finally, we discuss the physical intuition behind how Langevin dynamics enables efficient exploration of complex energy landscapes.

Discrete-Time Langevin Dynamics. The discrete-time Langevin update is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla_{\mathbf{x}} E_{\phi}(\mathbf{x}_n) + \sqrt{2\eta}\boldsymbol{\epsilon}_n, \quad n = 0, 1, 2, \ldots, \tag{3.1.4}$$

where $\mathbf{x}_0$ is initialized from some distribution (often Gaussian), $\eta > 0$ is the step size, and $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is Gaussian noise. The noise enables exploration beyond local minima by adding stochasticity.

Since the score function can be computed as

$$\nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\phi}(\mathbf{x}).$$

the update can equivalently be written as

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta \nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}_n) + \sqrt{2\eta}\boldsymbol{\epsilon}_n, \tag{3.1.5}$$

where the score function guides the samples toward high-density regions. This formulation is central to diffusion models, as will be detailed later.

Continuous-Time Langevin Dynamics. As the step size η approaches zero, the discrete Langevin updates naturally converge to a continuous-time process described by the *Langevin Stochastic Differential Equation (SDE)*[1] :

$$\mathrm{d}\mathbf{x}(t) = \nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}(t))\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}\mathbf{w}(t), \tag{3.1.6}$$

where $\mathbf{w}(t)$ denotes a standard Brownian motion (also known as a Wiener process[2]). It is important to understand that the discrete update rule in Equation (3.1.4) serves as the Euler–Maruyama discretization of this continuous SDE.

Under standard regularity assumptions (e.g., $p_{\phi} \propto e^{-E_{\phi}}$ with a confining, sufficiently smooth E_{ϕ}), the distribution of $\mathbf{x}(t)$ converges (exponentially fast) to p_{ϕ} as $t \to \infty$; thus we can sample by simulating (solving) the SDE Equation (3.1.6).

Why Langevin Sampling? A natural way to understand Langevin sampling is through the lens of physics, where the energy function $E_{\phi}(\mathbf{x})$ defines a potential landscape that shapes the behavior of particles. According to Newtonian dynamics,

[1] With the factor $\sqrt{2}$, the Langevin dynamics leave p_{ϕ} unchanged in time. Namely, p_{ϕ} is stationary: if $\mathbf{x}(0) \sim p_{\phi}$ then $\mathbf{x}(t) \sim p_{\phi}$ for all $t \geq 0$. Equivalently, p_{ϕ} is the stationary solution of the Fokker–Planck equation (see Chapter B):

$$\partial_t \rho = -\nabla \cdot (\rho \nabla \log p_{\phi}) + \tfrac{\sigma^2}{2} \Delta \rho.$$

Setting $\rho = p_{\phi}$ gives $(\frac{\sigma^2}{2} - 1)\Delta p_{\phi} = 0$, which holds only if $\sigma = \sqrt{2}$.

[2] Brownian increments satisfy $\mathbf{w}(t+\eta) - \mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, \eta\mathbf{I})$. Euler–Maruyama therefore uses a step noise $\sqrt{2}[\mathbf{w}(t+\eta) - \mathbf{w}(t)] = \sqrt{2\eta}\boldsymbol{\epsilon}_n$ with $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which explains the $\sqrt{\eta}$ factor.; this is the source of the square-root scaling. For a detailed introduction to Brownian motion and SDEs, please refer to Chapter A.

the motion of a particle under the force field derived from this energy is described by the ordinary differential equation (ODE)

$$\mathrm{d}\mathbf{x}(t) = -\nabla_{\mathbf{x}} E_{\boldsymbol{\phi}}(\mathbf{x}(t))\,\mathrm{d}t,$$

which deterministically drives the particle downhill toward a local minimum of the energy function. However, such deterministic dynamics can become trapped in local minima, preventing exploration of the full data distribution.

To overcome this limitation, Langevin dynamics introduces stochastic perturbations, resulting in the SDE

$$\mathrm{d}\mathbf{x}(t) = -\nabla_{\mathbf{x}} E_{\boldsymbol{\phi}}(\mathbf{x}(t))\,\mathrm{d}t + \underbrace{\sqrt{2}\,\mathrm{d}\mathbf{w}(t)}_{\text{injected noise}},$$

where $\mathbf{w}(t)$ is a standard Brownian motion. The noise term allows the particle to escape local minima by crossing energy barriers, making the trajectory a stochastic process whose stationary distribution converges to the Boltzmann distribution

$$p_{\boldsymbol{\phi}}(\mathbf{x}) \propto e^{-E_{\boldsymbol{\phi}}(\mathbf{x})}.$$

From this perspective, EBMs can be viewed as learning a force field that pushes samples toward regions of high probability. Langevin sampling is particularly useful for EBMs because it provides a practical method to generate samples from the model distribution $p_{\boldsymbol{\phi}}(\mathbf{x})$ without explicitly computing the partition function. By iteratively applying the Langevin update, one obtains samples that approximate the target distribution.

Inherent Challenges of Langevin Sampling. Langevin dynamics, a widely used MCMC-based sampler, faces serious limitations in high-dimensional spaces. Its efficiency is highly sensitive to the choice of step size η, noise scale, and the number of iterations required to approximate the target distribution accurately.

At the heart of this inefficiency lies the issue of poor "mixing time": In complex data distributions with many isolated modes, Langevin sampling often requires an extremely long time to transition between regions of high probability. This problem becomes significantly worse as dimensionality increases, leading to prohibitively slow convergence.

One can think of sampling as exploring a vast and rugged landscape with many distant valleys, each corresponding to a different data mode. Langevin dynamics, relying on local stochastic updates, struggles to traverse between these valleys efficiently. As a result, it often fails to capture the full diversity of the distribution.

This inefficiency hints the need for more structured and guided sampling methods that can navigate complex data manifolds more effectively than purely random exploration.

3.2 From Energy-Based to Score-Based Generative Models

EBMs show that generation depends only on the score, which points toward regions of higher probability, rather than on the full normalized density. While score matching avoids the partition function, training through the energy still requires expensive second derivatives. The key idea is that since sampling with Langevin dynamics needs only the score, we can learn it directly with a neural network. This shift, from modeling energies to modeling scores, forms the foundation of score-based generative models.

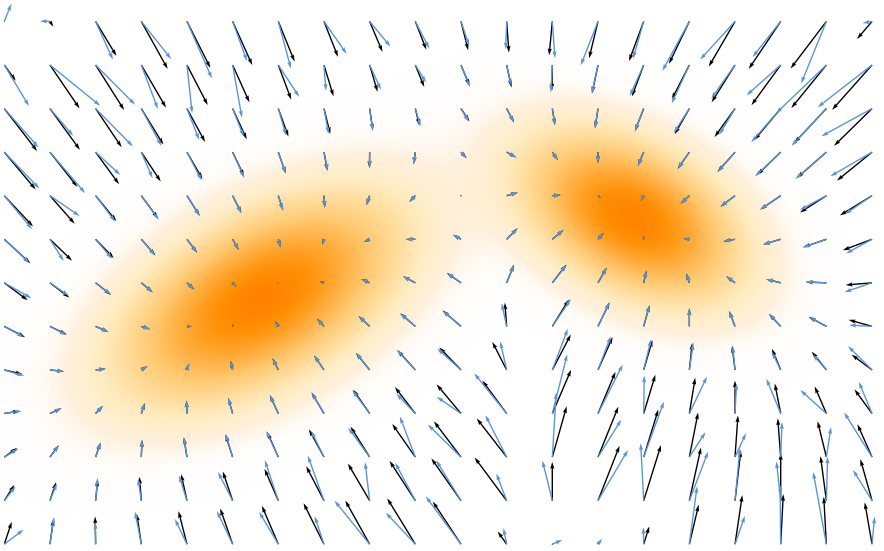


Figure 3.4: Illustration of Score Matching. The neural network score $\mathbf{s}_\phi(\mathbf{x})$ is trained to match the ground truth score $\mathbf{s}(\mathbf{x})$ using a MSE loss. Both are represented as vector fields.

Source: Created by the authors.

3.2.1 Training with Score Matching

Score Matching. To approximate the score function $\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ from samples of p_{data}, we approximate it directly as a vector field parameterized by a neural network $\mathbf{s}_\phi(\mathbf{x})$ (see Figure 3.4):

$$\mathbf{s}_\phi(\mathbf{x}) \approx \mathbf{s}(\mathbf{x}).$$

Score matching fits this vector field by minimizing the mean squared error (MSE) between the true and estimated scores:

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) := \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}) - \mathbf{s}(\mathbf{x})\|_2^2\right]. \tag{3.2.1}$$

Tractable Score Matching. At first glance, this objective seems infeasible because the true score $\mathbf{s}(\mathbf{x})$, which serves as the regression target, is unknown. Fortunately, Hyvärinen and Dayan (2005) showed that integration by parts yields an equivalent objective that depends only on the model $\mathbf{s}_{\boldsymbol{\phi}}$ and the data samples, without requiring access to the true score. We state this key result in the following proposition:

Proposition 3.2.1: Hyvärinen's Tractable Form of SM

We can express the following equation as:

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) = \widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi}) + C.$$

where

$$\widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi}) := \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\mathrm{Tr}\left(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\right) + \frac{1}{2}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\right\|_2^2\right]. \tag{3.2.2}$$

and C is a constant that does not depend on $\boldsymbol{\phi}$. The minimizer $\mathbf{s}^*$ is obtained as: $\mathbf{s}^*(\cdot) = \nabla_{\mathbf{x}} \log p(\cdot)$.

Proof for Proposition.

The result follows by expanding the MSE in $\mathcal{L}_{\mathrm{SM}}$ and applying integration by parts. The proof is given in Section D.2.1.

Using the equivalent objective in Equation (3.2.2), we train the score model $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})$ solely from observed samples of p_{data}, eliminating the need for the true score function.

Intuition of Equation (3.2.2). The alternative score matching objective $\widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi})$ can be understood from its two terms. The norm term $\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|^2$ penalizes large score values, especially in regions where p_{data} is large. The divergence term $\mathrm{Tr}(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))$ appears with a positive sign in the loss, so minimizing the objective also favors locally contracting behavior. Together, these two terms yield the population-optimal score field $\mathbf{s}^*(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x})$. We explain the geometric intuition in more detail below.

Effect of the Magnitude Term. Since the expectation in $\widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi})$ is taken under p_{data}, regions where $p_{\mathrm{data}}(\mathbf{x})$ is large (high data density) contribute more

strongly to the loss. The magnitude term $\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|^2$ therefore places a stronger penalty on large score values in these regions. By itself, however, this term does not imply that $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}) = \mathbf{0}$ at every high-density point; at the population optimum,

$$\mathbf{s}^*(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}),$$

which vanishes only at stationary points of the log-density.

Concavity When the Field is (Approximately) a Gradient. Because the divergence term $\text{Tr}(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))$ enters the loss with a positive sign, minimizing the objective favors negative divergence in regions emphasized by the data distribution. Negative divergence indicates net local contraction, although by itself it does not guarantee that a stationary point is a sink. To make this precise, assume $\mathbf{s}_{\boldsymbol{\phi}} = \nabla_{\mathbf{x}} u$ for a scalar function $u : \mathbb{R}^D \to \mathbb{R}$, as is natural when matching a log density. Then $\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}} = \nabla^2_{\mathbf{x}} u$ (the Hessian) and $\nabla \cdot \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}) = \text{Tr}(\nabla^2_{\mathbf{x}} u(\mathbf{x}))$ (the divergence).
Recall that for a vector field $\mathbf{F} = (F_1, \ldots, F_D)$,

$$\text{Div}\,\mathbf{F} := \nabla \cdot \mathbf{F} = \begin{pmatrix} \partial/\partial x_1 \\ \vdots \\ \partial/\partial x_D \end{pmatrix} \cdot \begin{pmatrix} F_1(\mathbf{x}) \\ \vdots \\ F_D(\mathbf{x}) \end{pmatrix} = \text{Tr}\left(\nabla_{\mathbf{x}}\mathbf{F}\right).$$

Intuition: if, as you take a step, $\mathbf{F}$ increases in that same direction, then the point is likely a source.

At a stationary point $\mathbf{x}_\star$, where $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_\star) = \nabla_{\mathbf{x}} u(\mathbf{x}_\star) = \mathbf{0}$, a second order Taylor expansion gives

$$u(\mathbf{x}) = u(\mathbf{x}_\star) + \tfrac{1}{2}(\mathbf{x} - \mathbf{x}_\star)^\top \nabla^2_{\mathbf{x}} u(\mathbf{x}_\star)(\mathbf{x} - \mathbf{x}_\star) + o(\|\mathbf{x} - \mathbf{x}_\star\|^2).$$

If the Hessian $\nabla^2_{\mathbf{x}} u(\mathbf{x}_\star)$ is negative definite, then u is locally concave at $\mathbf{x}_\star$ and the log density attains a strict local maximum[3] there. Because all eigenvalues of the Hessian are negative, the trace is also negative: $\text{Tr}(\nabla^2_{\mathbf{x}} u(\mathbf{x}_\star)) < 0$. Thus, in this case, the stationary point is locally attracting under the score field: small perturbations are pulled back toward $\mathbf{x}_\star$.

3.2.2 Sampling with Langevin Dynamics

Once trained by minimizing Equation (3.2.2), the score model $\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x})$ can replace the oracle score in Langevin dynamics for sampling:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta \mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_n) + \sqrt{2\eta}\boldsymbol{\epsilon}_n, \quad \boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{3.2.3}$$

[3] We remark that strict concavity (and thus a strict local maximum of the log density) requires the entire Hessian $\nabla^2_{\mathbf{x}} u$ to be negative definite, not merely to have negative trace. A negative trace guarantees that the sum of eigenvalues is negative, but some eigenvalues could still be positive, leading to a saddle point rather than a maximum.

for $n = 0, 1, 2, \ldots$, initialized at $\mathbf{x}_0$. As in the EBM case Equation (3.1.6), this recursion is precisely the Euler–Maruyama discretization of the continuous-time Langevin SDE:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{s}_{\phi^\times}(\mathbf{x}(t))\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}\mathbf{w}(t),$$

with initialization $\mathbf{x}(0)$. Hence, in the limit of small step size, the discrete and continuous formulations coincide. In practice, one can either run the discrete sampler or directly simulate the SDE.

3.2.3 Prologue: Score-Based Generative Models

In the remainder of this chapter, we examine the foundational role of the score function in modern diffusion models. Initially introduced to enable efficient training of EBMs, the score function has evolved into a central component of a new generation of generative models. Building on this foundation, we explore how the score function informs the theoretical formulation and practical implementation of *score-based diffusion models*, offering a principled framework for data generation via stochastic processes.

3.3 Denoising Score Matching

3.3.1 Motivation

Although the alternative objective in Equation (3.2.2)

$$\widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi}) = \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[\mathrm{Tr}(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2\right]$$

is more tractable, it still requires computing the trace of the Jacobian, $\mathrm{Tr}(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))$. Although this does not require forming the full Jacobian matrix, exact evaluation generally becomes more expensive as the input dimension D increases, since it requires aggregating derivative information across input coordinates. This can therefore be computationally expensive in high dimensions.

To address this, sliced score matching (Song *et al.*, 2020b) replaces the trace term with a stochastic estimate based on random projections. We briefly outline the idea below.

Sliced Score Matching and Hutchinson's Estimator. Sliced score matching replaces the trace in score matching by averaging directional derivatives along random "slices". Let $\mathbf{u} \in \mathbb{R}^D$ be an *isotropic* random vector (e.g., Rademacher or standard Gaussian) with $\mathbb{E}[\mathbf{u}] = 0$ and $\mathbb{E}[\mathbf{u}\mathbf{u}^\top] = \mathbf{I}$. By Hutchinson's identity

$$\mathrm{Tr}(\mathbf{A}) = \mathbb{E}_{\mathbf{u}}[\mathbf{u}^\top\mathbf{A}\mathbf{u}], \quad \text{and} \quad \mathbb{E}_{\mathbf{u}}[(\mathbf{u}^\top\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))^2] = \|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2,$$

we obtain the exact form

$$\widetilde{\mathcal{L}}_{\mathrm{SM}}(\boldsymbol{\phi}) = \mathbb{E}_{\mathbf{x},\mathbf{u}}\Big[\mathbf{u}^\top(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))\mathbf{u} + \tfrac{1}{2}(\mathbf{u}^\top\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}))^2\Big].$$

This objective can be evaluated efficiently with automatic differentiation, using Jacobian- and vector-Jacobian-product operations (JVP/VJP) instead of explicitly computing large Jacobian or Hessian matrices. Averaging over K random probes yields an unbiased estimator with variance $\mathcal{O}(1/K)$, and the directional term $\mathbf{u}^\top(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}})\mathbf{u}$ can be computed efficiently using JVP/VJP routines without explicit Jacobians. This works since $\mathbf{u}^\top(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}})\mathbf{u} = \mathbf{u}^\top\nabla_{\mathbf{x}}(\mathbf{s}_{\boldsymbol{\phi}}^\top\mathbf{u})$, the gradient of a scalar. Intuitively, this means we only check the model's behavior along random directions: the projected score is nudged to align with regions of higher data density, so data points become stationary in expectation.

From Sliced to Denoising Score Matching. Sliced score matching sidesteps Jacobians but still relies on the raw data distribution. This makes it fragile: for image data lying on low-dimensional manifolds, the score $\nabla_{\mathbf{x}}\log p_{\mathrm{data}}(\mathbf{x})$ may be undefined or unstable, and the method only constrains the vector field at observed points, providing weak control in their neighborhoods. It further suffers from probe-induced variance and repeated JVP/VJP costs.

A more robust alternative, which we focus on here, is *Denoising Score Matching* (DSM) (Vincent, 2011), which offers a principled and scalable solution.

3.3.2 Training

Let us revisit the SM loss in Equation (3.2.1):

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x})\|_2^2\right],$$

where the issue arises from the intractable term $\nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x})$.

Vincent (2011)'s Solution by Conditioning. To overcome the intractability of $\nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x})$, Vincent (2011) proposed injecting noise into the data $\mathbf{x} \sim p_{\mathrm{data}}$ via a known conditional distribution $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ with scale σ. The neural network $\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, \sigma)$ is trained to approximate the score of the marginal perturbed distribution

$$p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\mathrm{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

by minimizing the loss

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}; \sigma) := \frac{1}{2}\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, \sigma) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2\right]. \tag{3.3.1}$$

Even though $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ is generally intractable, Vincent (2011) showed that *conditioning* on $\mathbf{x} \sim p_{\mathrm{data}}$ yields an equivalent, tractable objective—the *Denoising Score Matching* (DSM) loss:

$$\mathcal{L}_{\mathrm{DSM}}(\boldsymbol{\phi}; \sigma) := \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}, \tilde{\mathbf{x}}\sim p_\sigma(\cdot|\mathbf{x})}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, \sigma) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2\right]. \tag{3.3.2}$$

The optimal minimizer $\mathbf{s}^*$ of Equation (3.3.2) satisfies

$$\mathbf{s}^*(\tilde{\mathbf{x}}, \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}),$$

which is also optimal for Equation (3.3.1).

For example, when $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is Gaussian noise with variance σ^2,

$$p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2\mathbf{I}),$$

the gradient $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ has a closed form (see Equation (3.3.4)), making the regression target explicit and computationally tractable.

Moreover, as $\sigma \approx 0$, $p_\sigma(\tilde{\mathbf{x}}) \approx p_{\mathrm{data}}(\mathbf{x})$ and

$$\mathbf{s}^*(\tilde{\mathbf{x}}, \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \approx \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}),$$

indicating the learned score approximates the original data score, enabling its use in generation.

We formalize this discussion on the gradient equivalence between $\mathcal{L}_{\text{SM}}$ and $\mathcal{L}_{\text{DSM}}$ in the following theorem:

Theorem 3.3.1: Equivalence of $\mathcal{L}_{\text{SM}}$ and $\mathcal{L}_{\text{DSM}}$

For any fixed noise scale $\sigma > 0$, the following holds:

$$\mathcal{L}_{\text{SM}}(\boldsymbol{\phi}; \sigma) = \mathcal{L}_{\text{DSM}}(\boldsymbol{\phi}; \sigma) + C, \tag{3.3.3}$$

where C is a constant independent of the parameter $\boldsymbol{\phi}$. Furthermore, the minimizer $\mathbf{s}^*(\cdot, \sigma)$ of both losses satisfies

$$\mathbf{s}^*(\tilde{\mathbf{x}}, \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \quad \text{for almost every } \tilde{\mathbf{x}}.$$

Proof for Theorem.

The equivalence follows from a direct computation: by expanding the MSE in $\mathcal{L}_{\text{SM}}$ and $\mathcal{L}_{\text{DSM}}$, all $\boldsymbol{\phi}$-dependent terms cancel, leaving only a constant difference independent of $\boldsymbol{\phi}$.
The derivation of the minimizer follows the same argument as in Proposition 4.3.1. We defer the detailed derivation to Section D.2.2. ∎

This theorem, like Theorem 2.2.1 in DDPM, illustrates a key shared principle:

Insight 3.3.1: Conditioning Technique

The conditioning technique also appears in the variational view of diffusion models in DDPM (see Theorem 2.2.1), where conditioning on a data point $\mathbf{x}$ turns an intractable loss into a tractable one for Monte Carlo estimation. A similar idea arises in the flow-based perspective (e.g., Flow Matching (Lipman *et al.*, 2022)), as we will see in Section 5.2.

Special Case: Additive Gaussian Noise. We now consider the common case where Gaussian noise $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ with variance σ^2 is added to each data point $\mathbf{x} \sim p_{\text{data}}$:

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

so that the corrupted data $\tilde{\mathbf{x}}$ follows

$$p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2\mathbf{I}).$$

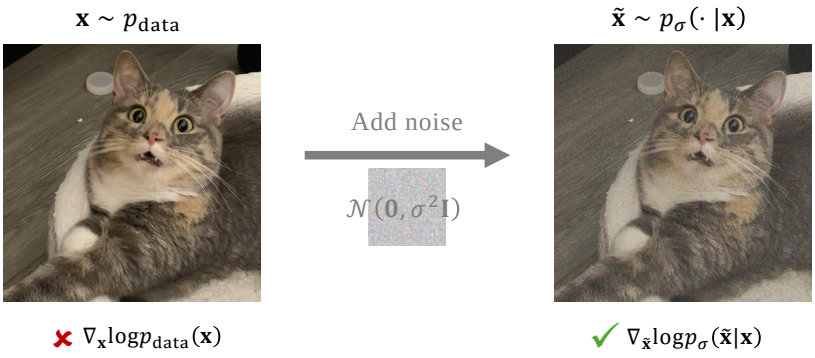


Figure 3.5: Illustration of DSM via the conditioning technique. By perturbing the data distribution p_{data} with small additive Gaussian noise $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, the resulting conditional distribution $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2\mathbf{I})$ admits a closed-form score function.

Source: Created by the authors.

In this setting, the conditional score is analytically given by

$$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2}.$$

Hence, the DSM loss simplifies to:

$$\begin{aligned} \mathcal{L}_{\text{DSM}}(\boldsymbol{\phi}; \sigma) &= \frac{1}{2}\mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}}|\mathbf{x}}\left[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, \sigma) - \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2}\right\|_2^2\right] \\ &= \frac{1}{2}\mathbb{E}_{\mathbf{x},\boldsymbol{\epsilon}}\left[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x} + \sigma\boldsymbol{\epsilon}, \sigma) + \frac{\boldsymbol{\epsilon}}{\sigma}\right\|_2^2\right], \end{aligned} \tag{3.3.4}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This objective forms the core of the (score-based) Diffusion Model.

At a fixed noise level $\sigma > 0$, DSM learns the score of the Gaussian-smoothed distribution

$$p_\sigma = p_{\text{data}} * \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}),$$

rather than the score of p_{data} itself. When σ is small, the corruption is mild, so p_σ still closely preserves the local structure of the data distribution. Its score therefore provides a useful direction for moving noisy samples back toward likely data regions. As σ becomes larger, the Gaussian smoothing progressively removes fine details and merges nearby modes. The corresponding score then captures mainly coarser, more global structure. This behavior is useful rather than undesirable: the multi-noise construction in Section 3.4 deliberately combines many noise levels, using large noise for coarse exploration and small noise for fine refinement.

To better see why the objective naturally corresponds to a "denoising" process, we expand on the discussion in Sections 3.3.4 and 3.3.5.

3.3.3 Sampling

Once we have a trained score model $\mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}, \sigma)$ at noise level σ, we generate samples using Langevin dynamics by replacing the true score with the learned model. The update rule is:

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + \eta \underbrace{\mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma)}_{\approx \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}_n)} + \sqrt{2\eta}\boldsymbol{\epsilon}_n, \quad \boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{3.3.5}$$

for $n = 0, 1, 2, \ldots$, starting from an initial value $\tilde{\mathbf{x}}_0$. If σ is sufficiently small, then after enough iterations, $\tilde{\mathbf{x}}_n$ approximates samples from p_{data}.

Advantages of Noise Injection. We additionally remark that, compared to vanilla score matching in Equation (3.2.1), injecting Gaussian noise to form p_σ (e.g., Equation (3.3.4)) provides two key advantages (Song and Ermon, 2019):

- **Well-Defined Gradients.** The noise perturbs data away from its lower-dimensional manifold, resulting in a distribution p_σ with full support in $\mathbb{R}^D$. Consequently, the score function $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ is well-defined everywhere.
- **Improved Coverage.** The noise smooths out sparse regions between modes, enhancing training signal quality and facilitating Langevin dynamics to traverse low-density regions more effectively.

3.3.4 Why DSM is Denoising: Tweedie's Formula

We begin with *Tweedie's formula* (Efron, 2011), which provides a principled basis for denoising from noisy observations alone. Concretely, it states that: given a single Gaussian–corrupted observation $\tilde{\mathbf{x}} \sim \mathcal{N}(\,\cdot\,; \alpha\mathbf{x}, \sigma^2\mathrm{I})$ from an unknown $\mathbf{x} \sim p_{\text{data}}$, a denoised estimate (the average over all plausible clean signals given $\tilde{\mathbf{x}}$) is obtained by nudging $\tilde{\mathbf{x}}$ a step of size σ^2 in the direction of the score $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ of its noisy marginal defined as:

$$p_\sigma(\tilde{\mathbf{x}}) := \int \mathcal{N}(\tilde{\mathbf{x}}; \alpha\mathbf{x}, \sigma^2\mathrm{I}) p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

We present the proposition formally below.

Lemma 3.3.2: Tweedie's Formula

Assume $\mathbf{x} \sim p_{\text{data}}$ and, conditionally on $\mathbf{x}$, $\tilde{\mathbf{x}} \sim \mathcal{N}(\,\cdot\,; \alpha\mathbf{x}, \sigma^2\mathrm{I})$ with $\alpha \neq 0$. Then Tweedie's formula states

$$\alpha \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \tag{3.3.6}$$

where the expectation is taken over the posterior distribution $p(\mathbf{x}|\tilde{\mathbf{x}})$ of $\mathbf{x}$ given $\tilde{\mathbf{x}}$.

Proof for Lemma.

The proof proceeds by computing the score of the marginal

$$p(\tilde{\mathbf{x}}) = \int p(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

Differentiating under the integral and using the Gaussian form of the conditional density leads directly to an expression that rearranges into the desired identity linking the score with the posterior mean. See Section D.2.3 for details. ∎

Tweedie's formula plays a central role in diffusion models, where multiple layers of noise are introduced as in DDPM. It enables the estimation of clean samples from noisy observations via the score function, thereby establishing a fundamental link between score prediction and denoiser:

$$\underbrace{\mathbb{E}\left[\mathbf{x}|\tilde{\mathbf{x}}\right]}_{\substack{\text{denoiser} \\ \text{estimated from } \tilde{\mathbf{x}}}} = \frac{1}{\alpha}\left(\tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\right).$$

Especially, a single gradient-ascent step on the noisy log-likelihood with the particular step size σ^2 is the denoised estimate (the conditional average clean signal). This makes DSM training and denoising tightly related: if $\mathbf{s}_\phi(\tilde{\mathbf{x}}) \approx \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ trained from DSM, then

$$\frac{1}{\alpha}\left(\tilde{\mathbf{x}} + \sigma^2 \mathbf{s}_\phi(\tilde{\mathbf{x}})\right)$$

is the denoiser.

Proof. In DDPM, $\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\,\boldsymbol{\epsilon}$, so taking $\mathbb{E}[\,\cdot\,|\mathbf{x}_i]$ of both sides,

$$\mathbf{x}_i = \mathbb{E}[\bar{\alpha}_i \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\,\boldsymbol{\epsilon}\,|\,\mathbf{x}_i] \iff \bar{\alpha}_i\,\mathbb{E}[\mathbf{x}_0|\mathbf{x}_i] = \mathbf{x}_i - \sqrt{1-\bar{\alpha}_i^2}\,\mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_i].$$

Subtracting Tweedie's formula, $\bar{\alpha}_i\,\mathbb{E}[\mathbf{x}_0|\mathbf{x}_i] = \mathbf{x}_i + (1-\bar{\alpha}_i^2)\,\nabla_{\mathbf{x}_i} \log p_i(\mathbf{x}_i)$, from both sides,

$$\mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_i] = -\sqrt{1-\bar{\alpha}_i^2}\,\nabla_{\mathbf{x}_i} \log p_i(\mathbf{x}_i). \qquad \blacksquare$$

(Optional) Higher Order Tweedie's Formula. The classical Tweedie's formula expresses the posterior mean $\mathbb{E}[\mathbf{x}_0|\tilde{\mathbf{x}}]$ through the gradient $\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}})$. Higher order extensions (Meng *et al.*, 2021a) express the posterior covariance and higher cumulants through higher derivatives of $\log p(\tilde{\mathbf{x}})$.

Exponential Family Setup with the Log-Normalizer $\lambda(\tilde{\mathbf{x}})$. Assume the conditional law of $\tilde{\mathbf{x}}$ given a latent natural parameter $\boldsymbol{\eta} \in \mathbb{R}^D$ belongs to a natural exponential family written as

$$q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta}))\, q_0(\tilde{\mathbf{x}}).$$

Here $\psi(\boldsymbol{\eta}) := \log Z(\boldsymbol{\eta})$ with $Z(\boldsymbol{\eta}) := \int \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}})\, q_0(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}}$, the partition and log-partition functions. Indeed,

$$\int q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta})\, \mathrm{d}\tilde{\mathbf{x}} = \int \exp\left(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta})\right) q_0(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}} = \frac{\int \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}})\, q_0(\tilde{\mathbf{x}})\, \mathrm{d}\tilde{\mathbf{x}}}{Z(\boldsymbol{\eta})} = 1.$$

Here $q_0(\tilde{\mathbf{x}})$ is the *base measure*, namely the part that does not depend on $\boldsymbol{\eta}$; for additive Gaussian noise with variance $\sigma^2\mathbf{I}$ it equals $(2\pi\sigma^2)^{-D/2} \exp(-\|\tilde{\mathbf{x}}\|^2/2\sigma^2)$. Let $p(\boldsymbol{\eta})$ be the pre-defined distribution of the latent natural parameter, which can be viewed as the reparameterized clean-data distribution (for Gaussian location, $\boldsymbol{\eta} = \mathbf{x}/\sigma^2$). The observed noisy marginal is

$$p_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta})\, p(\boldsymbol{\eta})\, \mathrm{d}\boldsymbol{\eta}.$$

Define the *log-partition* (log-normalizer) in $\tilde{\mathbf{x}}$ by

$$\lambda(\tilde{\mathbf{x}}) := \log p_\sigma(\tilde{\mathbf{x}}) - \log q_0(\tilde{\mathbf{x}}).$$

Then the posterior of $\boldsymbol{\eta}$ given $\tilde{\mathbf{x}}$ is

$$p(\boldsymbol{\eta}|\tilde{\mathbf{x}}) \;\propto\; \exp\!\big(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta}) - \lambda(\tilde{\mathbf{x}})\big)\, p(\boldsymbol{\eta}),$$

which shows that, as a function of $\tilde{\mathbf{x}}$, the posterior has exponential-family form with natural parameter $\tilde{\mathbf{x}}$, sufficient statistic $\boldsymbol{\eta}$, and log-partition $\lambda(\tilde{\mathbf{x}})$.

Proof. The posterior is $p(\boldsymbol{\eta}|\tilde{\mathbf{x}}) \propto q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta})\, p(\boldsymbol{\eta}) = \exp\left(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta})\right) q_0(\tilde{\mathbf{x}})\, p(\boldsymbol{\eta})$, whose normalizer over $\boldsymbol{\eta}$ is

$$\int \exp\left(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta})\right) p(\boldsymbol{\eta})\, \mathrm{d}\boldsymbol{\eta} = \frac{1}{q_0(\tilde{\mathbf{x}})} \int q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta})\, p(\boldsymbol{\eta})\, \mathrm{d}\boldsymbol{\eta} = \frac{p_\sigma(\tilde{\mathbf{x}})}{q_0(\tilde{\mathbf{x}})},$$

so its logarithm is $\log p_\sigma(\tilde{\mathbf{x}}) - \log q_0(\tilde{\mathbf{x}}) = \lambda(\tilde{\mathbf{x}})$: the log-partition of the latent posterior is exactly $\lambda(\tilde{\mathbf{x}})$. ∎

Derivatives of λ Produce Posterior Cumulants. Two simple rules are at play. First, normalization: for every $\tilde{\mathbf{x}}$,

$$\int \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta}) - \lambda(\tilde{\mathbf{x}}))\, p(\boldsymbol{\eta})\, \mathrm{d}\boldsymbol{\eta} = 1.$$

Differentiating this identity with respect to $\tilde{\mathbf{x}}$ brings down powers of $\boldsymbol{\eta}$ from the exponential and derivatives of $\lambda(\tilde{\mathbf{x}})$; setting the result to zero yields equalities between derivatives of λ and posterior moments of $\boldsymbol{\eta}$. Second, a standard property of exponential families: the log-partition is the cumulant generating function of the sufficient statistic. Therefore

$$\nabla_{\tilde{\mathbf{x}}} \lambda(\tilde{\mathbf{x}}) = \mathbb{E}[\boldsymbol{\eta}|\tilde{\mathbf{x}}], \quad \nabla^2_{\tilde{\mathbf{x}}} \lambda(\tilde{\mathbf{x}}) = \mathrm{Cov}[\boldsymbol{\eta}|\tilde{\mathbf{x}}], \quad \nabla^{(k)}_{\tilde{\mathbf{x}}} \lambda(\tilde{\mathbf{x}}) = \kappa_k(\boldsymbol{\eta}|\tilde{\mathbf{x}}) \quad (k \geq 3),$$

where κ_k are the *conditional cumulants* of order k of the random vector $\boldsymbol{\eta}$ given $\tilde{\mathbf{x}}$, obtained via the standard moment–cumulant relations.

These are the higher order Tweedie's formulas. Specializing to the Gaussian location model with $\boldsymbol{\eta} = \mathbf{x}/\sigma^2$ yields the familiar forms in terms of derivatives of $\log p_\sigma(\tilde{\mathbf{x}})$:

$$\mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \qquad \mathrm{Cov}[\mathbf{x}|\tilde{\mathbf{x}}] = \sigma^2 \mathrm{I} + \sigma^4 \nabla^2_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}),$$

and higher cumulants scale with higher derivatives of $\log p_\sigma(\tilde{\mathbf{x}})$.

Several studies have explored training neural networks to estimate higher order scores (Meng *et al.*, 2021a; Lu *et al.*, 2022a; Lai *et al.*, 2023a). In contrast, our aim is to clarify their relationship with statistical quantities, and we refer the reader to these works for methodological details.

3.3.5 (Optional) Why DSM is Denoising: SURE

SURE (Stein's Unbiased Risk Estimator). At a high level, Stein's Unbiased Risk Estimator (SURE) is a technique that allows one to estimate the mean squared error (MSE) of a denoiser $\mathbf{D}$ *without knowing the clean signal*. In other words, SURE provides a way to select or train denoisers when only noisy data are available. Main takeaway: integration by parts lets you estimate, under expectation, an MSE involving an unknown variable with an observable surrogate objective.

For clarity, consider the additive Gaussian noise setting:

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where $\mathbf{x} \in \mathbb{R}^D$ is the unknown clean signal and $\tilde{\mathbf{x}}$ is the observed noisy version. A denoiser is any (weakly differentiable) mapping $\mathbf{D} : \mathbb{R}^D \to \mathbb{R}^D$ that produces an estimate $\mathbf{D}(\tilde{\mathbf{x}})$ of $\mathbf{x}$.

The natural quality measure is the conditional MSE

$$R(\mathbf{D}; \mathbf{x}) := \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}} \left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|_2^2 \,|\, \mathbf{x} \right].$$

This quantity depends on the unknown ground truth $\mathbf{x}$, and therefore cannot be computed directly. Stein's identity (see Section D.2.4), however, yields the following *observable* surrogate:

$$\mathrm{SURE}(\mathbf{D}; \tilde{\mathbf{x}}) = \|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|_2^2 + 2\sigma^2 \, \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2, \tag{3.3.7}$$

where $\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}})$ denotes the divergence of $\mathbf{D}$. We emphasize that $\mathrm{SURE}(\mathbf{D}; \tilde{\mathbf{x}})$ requires only the noisy observation $\tilde{\mathbf{x}}$, not the clean $\mathbf{x}$.

Intuitively, SURE consists of two parts that complement each other. The term $\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2$ measures how far the denoiser's output is from the noisy input; by itself this underestimates the true error since $\tilde{\mathbf{x}}$ is already corrupted. The divergence term acts as a correction: it captures how sensitive the denoiser is to small

perturbations in its input, effectively accounting for the variance introduced by the noise.

Importantly, for any fixed but unknown $\mathbf{x}$,

$$\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}}\left[\mathrm{SURE}(\mathbf{D};\mathbf{x}+\sigma\boldsymbol{\epsilon})\,\middle|\,\mathbf{x}\right] = R(\mathbf{D};\mathbf{x}),$$

where the expectation is over the Gaussian noise $\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$. Thus, minimizing SURE (in expectation or empirically) is equivalent to minimizing the true MSE, while relying only on noisy data. In practice, averaging SURE over both $\mathbf{x}\sim p_{\text{data}}$ and the corruption noise $\boldsymbol{\epsilon}$ yields an unbiased estimate of the global MSE risk.

Link to Tweedie's Formula and Bayes Optimality. Let $p_\sigma(\tilde{\mathbf{x}}) = \big(p_{\text{data}} * \mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})\big)(\tilde{\mathbf{x}})$ denote the noisy marginal considered in this section.

SURE is an unbiased estimator of the mean squared error with respect to the noise, conditional on $\mathbf{x}$:

$$\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}}[\mathrm{SURE}(\mathbf{D};\tilde{\mathbf{x}})] = \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}}[\|\mathbf{D}(\tilde{\mathbf{x}})-\mathbf{x}\|^2].$$

Hence minimizing the expected SURE equals minimizing the Bayes risk $\mathbb{E}_{(\mathbf{x},\tilde{\mathbf{x}})}\left[\|\mathbf{D}(\tilde{\mathbf{x}})-\mathbf{x}\|^2\right] = \mathbb{E}_{\tilde{\mathbf{x}}}[\mathbb{E}_{\mathbf{x}|\tilde{\mathbf{x}}}[\|\mathbf{D}(\tilde{\mathbf{x}})-\mathbf{x}\|^2]]$ by the law of total expectation (tower property). This decomposition yields a pointwise optimization: for almost every $\tilde{\mathbf{x}}$,

$$\mathbf{D}^*(\tilde{\mathbf{x}}) = \arg\min_{\mathbf{z}}\ \mathbb{E}_{\mathbf{x}|\tilde{\mathbf{x}}}[\|\mathbf{z}-\mathbf{x}\|^2] = \mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}].$$

Therefore the SURE-optimal denoiser coincides with the Bayes estimator in Section 3.3.4, and by Tweedie's identity:

$$\mathbf{D}^*(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}). \tag{3.3.8}$$

Relationship of SURE and Score Matching. The identity in Equation (3.3.8) motivates parameterizing the denoiser $\mathbf{D}$ via a score field:

$$\mathbf{D}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}},\sigma),$$

with $\mathbf{s}_{\boldsymbol{\phi}}(\cdot,\sigma)$ meant to approximate the noisy score $\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\cdot)$. Plugging $\mathbf{D}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}},\sigma)$ in Equation (3.3.7) gives

$$\frac{1}{2\sigma^4}\,\mathrm{SURE}(\mathbf{D};\tilde{\mathbf{x}}) = \mathrm{Tr}\big(\nabla_{\tilde{\mathbf{x}}}s_{\boldsymbol{\phi}}(\tilde{\mathbf{x}},\sigma)\big) + \tfrac{1}{2}\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}},\sigma)\|_2^2 + \mathrm{const}(\sigma).$$

Therefore, taking expectation with respect to $\tilde{\mathbf{x}}\sim p_\sigma$, minimizing SURE is equivalent (up to an additive constant) to minimizing Hyvärinen's alternative score matching objective at noise level σ, with the expectation taken under p_σ (see Equation (3.2.2)). Consequently, both objectives share the same minimizer, namely the denoiser in Equation (3.3.8).

3.3.6 (Optional) Generalized Score Matching

Motivation. Classical score matching, denoising score matching, and higher order variants all target

$$\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}, \quad \text{for some density } p$$

with a linear operator $\mathcal{L}$ acting on the density. In the classical case $\mathcal{L} = \nabla_{\mathbf{x}}$, this gives

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})}$$

The $\frac{\mathcal{L}p}{p}$ structure allows integration by parts to remove normalizing constants, yielding a tractable objective that depends only on samples from p and the learned field $\mathbf{s}_{\boldsymbol{\phi}}$. See Section D.2.1. This viewpoint motivates the generalized score matching framework.

Generalized Fisher Divergence. Let p be the data distribution and q any model distribution. For a linear operator $\mathcal{L}$ on scalar functions of $\mathbf{x}$, define the generalized Fisher divergence

$$\mathcal{D}_{\mathcal{L}}(p \,\|\, q) := \int p(\mathbf{x}) \left\| \frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} - \frac{\mathcal{L}q(\mathbf{x})}{q(\mathbf{x})} \right\|_2^2 \mathrm{d}\mathbf{x}.$$

If $\mathcal{L}$ is *complete*, i.e.,

$$\frac{\mathcal{L}p_1}{p_1} = \frac{\mathcal{L}p_2}{p_2} \text{ a.e.} \quad \text{implies} \quad p_1 = p_2 \text{ a.e.},$$

then $\mathcal{D}_{\mathcal{L}}(p \,\|\, q) = 0$ identifies $q = p$. For $\mathcal{L} = \nabla_{\tilde{\mathbf{x}}}$ this recovers the classical Fisher divergence (see Equation (1.1.3)).

Score Parameterization. In practice we do not model a normalized density q. Instead, we directly parameterize a vector field $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})$ to approximate the generalized score $\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}$. Consider

$$\mathcal{D}_{\mathcal{L}}(p \,\|\, \mathbf{s}_{\boldsymbol{\phi}}) := \mathbb{E}_{\mathbf{x}\sim p}\left[\left\| \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}) - \frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} \right\|_2^2 \right].$$

Although $\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}$ is unknown, "integration by parts" makes the loss depend only on $\mathbf{s}_{\boldsymbol{\phi}}$. Let $\mathcal{L}^\dagger$ be the adjoint of $\mathcal{L}$, defined by

$$\int (\mathcal{L}f)^\top g = \int f\, (\mathcal{L}^\dagger g) \quad \text{for all test functions } f, g,$$

which formally "moves" $\mathcal{L}$ across the integral when boundary terms vanish. Expanding the square and applying this identity yields the tractable objective

$$\mathcal{L}_{\mathrm{GSM}}(\boldsymbol{\phi}) = \mathbb{E}_{\mathbf{x}\sim p}\left[\frac{1}{2} \|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2 - (\mathcal{L}^\dagger \mathbf{s}_{\boldsymbol{\phi}})(\mathbf{x}) \right] + \text{const},$$

where the constant does not depend on $\boldsymbol{\phi}$. We use p only through expectations, so the generalized score matching loss admits an empirical estimator from training data, exactly as in classical score matching.

For $\mathcal{L} = \nabla$ we have $\mathcal{L}^\dagger = -\nabla\cdot$, which recovers Hyvärinen's score matching objective $\mathbb{E}_p\big[\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\phi}}\|_2^2 + \nabla\cdot\mathbf{s}_{\boldsymbol{\phi}}\big]$ in Equation (3.2.2).

Examples of Operators.

- **Classical Score Matching.** Consider $\mathcal{L} = \nabla_{\mathbf{x}}$. Then the generalized score reduces to the classical score function
$$\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} = \nabla_{\mathbf{x}} \log p(\mathbf{x}).$$
- **Denoising Score Matching.** For additive Gaussian noise, define the operator
$$(\mathcal{L}f)(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}\, f(\tilde{\mathbf{x}}) + \sigma^2 \nabla_{\tilde{\mathbf{x}}} f(\tilde{\mathbf{x}}).$$
Then
$$\frac{\mathcal{L}p_\sigma(\tilde{\mathbf{x}})}{p_\sigma(\tilde{\mathbf{x}})} = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{x}_0|\tilde{\mathbf{x}}],$$
with
$$p_\sigma(\tilde{\mathbf{x}}) := \int \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}_0, \sigma^2\mathbf{I})\, p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0, \qquad \tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}.$$
This is exactly Tweedie's identity for additive Gaussian noise. Minimizing $\mathcal{L}_{\text{GSM}}$ with this operator therefore trains $\mathbf{s}_{\boldsymbol{\phi}}$ to approximate the posterior-mean denoiser. If one instead uses a scaled Gaussian location model $\tilde{\mathbf{x}} = \alpha\mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$, the corresponding identity contains the factor α as in Equation (3.3.6).
- **Higher Order Targets.** Stacking derivatives inside $\mathcal{L}$ exposes $\nabla^2 \log p$ and higher derivatives, which align with posterior covariance and higher order cumulants.

Extensions and Use Cases. Generalized score matching extends beyond continuous variables to discrete settings, including language modeling (Meng *et al.*, 2022; Lou *et al.*, 2024). It also motivates score inspired training that yields denoising style objectives. This operator view unifies a range of objectives, admits empirical estimation from data, and offers a general principle for designing loss functions through suitable choices of $\mathcal{L}$.

3.4 Multi-Noise Levels of Denoising Score Matching (NCSN)

3.4.1 Motivation

Adding Gaussian noise with a single fixed variance to the data distribution smooths it to a certain extent, but training a score-based model at only one noise level introduces key limitations. At low levels of injected noise, Langevin dynamics struggles to traverse modes in multi-modal distributions due to vanishing gradients in low-density regions. In contrast, at high noise levels, sampling becomes easier, but the model captures only coarse structures, resulting in blurry samples that lack fine detail. Furthermore, Langevin dynamics can be slow to converge or even fail in high-dimensional spaces. Since it depends on the gradient of the log-density for guidance, poor initialization, particularly in plateau regions or near saddle points, can impede exploration or cause the sampler to get trapped in a single mode.

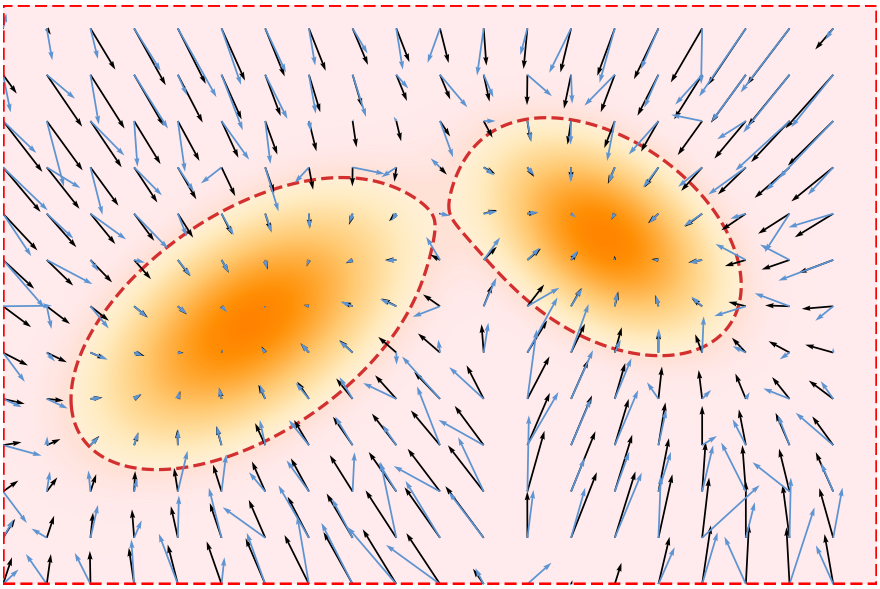


Figure 3.6: Illustration of SM inaccuracy (revisiting Figure 3.4). the red region indicates low-density areas with potentially inaccurate score estimates due to limited sample coverage, while high-density regions tend to yield more accurate estimates.

Source: Created by the authors.

To address these challenges, Song and Ermon (2019) propose injecting Gaussian noise at multiple levels into the data distribution and jointly training a noise-conditional score network (NCSN) to estimate score functions across a range of noise scales. During generation, Langevin dynamics is applied in a noise-annealed fashion: beginning with high-noise levels to enable coarse exploration, and gradually refining toward low-noise levels to recover fine details.

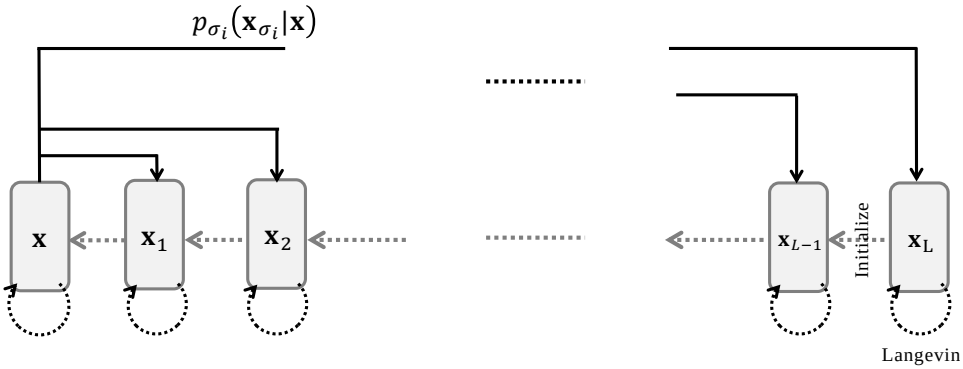


Figure 3.7: Illustration of NCSN. The forward process perturbs the data with multiple levels of additive Gaussian noise $p_\sigma(\mathbf{x}_\sigma|\mathbf{x})$. Generation proceeds (in dash lines) via Langevin sampling at each noise level, using the result from the current level to initialize sampling at the next lower variance.

Source: Created by the authors.

3.4.2 Training

To overcome the limitations of score-based models trained at a single noise level, Song and Ermon (2019) propose adding Gaussian noise at multiple levels to the data distribution. Specifically, a sequence of L noise levels $\{\sigma_i\}_{i=1}^L$ is chosen such that

$$0 < \sigma_1 < \sigma_2 < \cdots < \sigma_L,$$

where σ_1 is small enough to preserve most of the data's fine details, and σ_L is large enough to sufficiently smooth the distribution, facilitating easier training.

Each noisy sample is constructed by perturbing a clean data point $\mathbf{x} \sim p_{\text{data}}$ as $\mathbf{x}_\sigma = \mathbf{x} + \sigma\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This defines the

Perturbation Kernel:

$$p_\sigma(\mathbf{x}_\sigma|\mathbf{x}) := \mathcal{N}(\mathbf{x}_\sigma; \mathbf{x}, \sigma^2\mathbf{I}),$$

which induces the

Marginal Distribution:

$$p_\sigma(\mathbf{x}_\sigma) = \int p_\sigma(\mathbf{x}_\sigma|\mathbf{x})p_{\text{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

at each noise level σ. It presents the Gaussian smoothed data distribution.

Training Objective of NCSN. The goal is to train a noise-conditional score network $\mathbf{s}_\phi(\mathbf{x}, \sigma)$ to estimate the score function $\nabla_\mathbf{x} \log p_\sigma(\mathbf{x})$ for all $\sigma \in \{\sigma_i\}_{i=1}^L$.

This is achieved by minimizing the DSM objective across all noise levels:

$$\mathcal{L}_{\text{NCSN}}(\boldsymbol{\phi}) := \sum_{i=1}^{L} \lambda(\sigma_i)\mathcal{L}_{\text{DSM}}(\boldsymbol{\phi};\sigma_i), \tag{3.4.1}$$

where

$$\mathcal{L}_{\text{DSM}}(\boldsymbol{\phi};\sigma) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x}),\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left[\left\|\mathbf{s}_\phi(\tilde{\mathbf{x}},\sigma) - \left(\frac{\mathbf{x}-\tilde{\mathbf{x}}}{\sigma^2}\right)\right\|_2^2\right],$$

and $\lambda(\sigma_i) > 0$ is a weighting function for each scale.

Minimizing this objective yields the score model $\mathbf{s}^*(\mathbf{x},\sigma)$ that recovers the true score at each noise level:

$$\mathbf{s}^*(\cdot,\sigma) = \nabla_{\mathbf{x}} \log p_\sigma(\cdot), \quad \text{for all } \sigma \in \{\sigma_i\}_{i=1}^L,$$

as it is essentially DSM minimization (see Theorem 3.3.1).

Relationship with DDPM Loss. Let $\mathbf{x}_\sigma = \mathbf{x} + \sigma\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and let p_σ denote the marginal distribution. By Tweedie's formula,

$$\nabla_{\mathbf{x}_\sigma} \log p_\sigma(\mathbf{x}_\sigma) = -\frac{1}{\sigma}\mathbb{E}\left[\boldsymbol{\epsilon}|\mathbf{x}_\sigma\right].$$

Thus the NCSN optimum is the true score $\mathbf{s}^*(\mathbf{x}_\sigma,\sigma) = \nabla_{\mathbf{x}_\sigma} \log p_\sigma(\mathbf{x}_\sigma)$, while the Bayes optimal noise predictor under the DDPM loss Equation (2.2.10) is $\boldsymbol{\epsilon}^*(\mathbf{x}_\sigma,\sigma) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_\sigma]$. They are exactly equivalent via

$$\mathbf{s}^*(\mathbf{x}_\sigma,\sigma) = -\frac{1}{\sigma}\boldsymbol{\epsilon}^*(\mathbf{x}_\sigma,\sigma), \qquad \boldsymbol{\epsilon}^*(\mathbf{x}_\sigma,\sigma) = -\sigma\mathbf{s}^*(\mathbf{x}_\sigma,\sigma).$$

In the DDPM setting, the noisy sample at step i can be written as

$$\sigma_i := \sqrt{1-\bar{\alpha}_i^2}, \qquad \mathbf{x}_i = \bar{\alpha}_i\mathbf{x}_0 + \sigma_i\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I}).$$

Therefore, the score and the optimal noise prediction contain the same information:

$$\mathbf{s}^*(\mathbf{x}_i,i) = -\frac{1}{\sigma_i}\mathbb{E}\left[\boldsymbol{\epsilon}|\mathbf{x}_i\right].$$

In other words, predicting the noise in DDPM is equivalent, up to the known scale factor $1/\sigma_i$, to predicting the score at noise level i.

We will systematically compare and summarize this equivalence of parameterizations in Chapter 6.

3.4.3 Sampling

With trained score networks available at multiple noise levels

$$\mathbf{s}_{\boldsymbol{\phi}^\times}(\cdot,\sigma_1), \quad \mathbf{s}_{\boldsymbol{\phi}^\times}(\cdot,\sigma_2), \quad \cdots, \quad \mathbf{s}_{\boldsymbol{\phi}^\times}(\cdot,\sigma_{L-1}), \quad \mathbf{s}_{\boldsymbol{\phi}^\times}(\cdot,\sigma_L),$$

Algorithm 1 Annealed Langevin Dynamics

Input: Trained score $\mathbf{s}_{\phi^\times}(\cdot, \sigma_\ell)$, step sizes η_ℓ, and Langevin iteration budgets N_ℓ for each noise level $\ell = L, \ldots, 1$; initialization distribution p_{init}

$\mathbf{x} \sim p_{\text{init}}$

▷ Initialize from unstructured noise at the largest noise level

for $\ell = L, L-1, \ldots, 1$ **do**

  $\tilde{\mathbf{x}}_0 \leftarrow \mathbf{x}$

  ▷ Initialize this noise level from the previous level's output

  for $n = 0$ **to** $N_\ell - 1$ **do**

    $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

    $\tilde{\mathbf{x}}_{n+1} \leftarrow \tilde{\mathbf{x}}_n + \eta_\ell \mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_\ell) + \sqrt{2\eta_\ell}\, \boldsymbol{\epsilon}_n$

  end for

  $\mathbf{x} \leftarrow \tilde{\mathbf{x}}_{N_\ell}$

  ▷ Pass the final state to the next smaller noise level, if any

end for

Output: $\mathbf{x}$

the sampling procedure known as *annealed Langevin dynamics* (Song and Ermon, 2019) generates data by progressively denoising from a high noise level σ_L down to a low noise level $\sigma_1 \approx 0$.

With trained scores at noise levels $0 < \sigma_1 < \cdots < \sigma_L$, annealed Langevin dynamics generates samples by moving gradually from high noise to low noise. Starting from a simple initialization $\mathbf{x} \sim p_{\text{init}}$, it first applies Langevin dynamics at the largest noise level σ_L, whose score $\mathbf{s}_{\phi^\times}(\cdot, \sigma_L)$ guides the samples toward the corresponding smoothed distribution p_{σ_L}. The resulting samples are then used to initialize Langevin dynamics at the next smaller noise level. Repeating this procedure progressively refines the samples until the smallest noise level σ_1 is reached.

At each noise level σ_ℓ, Langevin dynamics performs the update

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + \eta_\ell \mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_\ell) + \sqrt{2\eta_\ell}\, \boldsymbol{\epsilon}_n, \qquad \boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where the output from the previous noise level provides the initialization $\tilde{\mathbf{x}}_0$. A common choice is to scale the step size with the noise level as

$$\eta_\ell = \delta \frac{\sigma_\ell^2}{\sigma_1^2}, \qquad \delta > 0.$$

The algorithm performs Langevin updates at every noise level, including σ_1, so the final sample is obtained after refinement at the smallest noise level. Algorithm 1 summarizes the procedure.

The initial distribution p_{init} simply provides a starting point for the Markov chain. At the largest noise level, Langevin dynamics moves this initial distribution

toward p_{σ_L}, after which the successive noise levels gradually guide the samples toward increasingly less smoothed versions of the data distribution.

Slow Sampling Speed of NCSN. NCSN generates samples using annealed MCMC (commonly Langevin dynamics) across noise scales $\{\sigma_i\}_{i=1}^{L}$. For each scale σ_i, it performs K iterative updates of the form "update $\tilde{\mathbf{x}}_n$ using the score $\mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_i)$ plus a small random perturbation", each requiring a forward pass through the score network. Two factors necessitate large $L \times K$:

(i) **Local Accuracy and Stability:** the learned score is reliable only for small perturbations, requiring small step sizes and many iterations per noise level to avoid bias or instability;

(ii) **Slow Mixing in High Dimensions:** local MCMC moves explore multimodal, high-dimensional targets inefficiently, demanding many iterations to reach typical data regions.

Because updates are strictly sequential (each iteration depends on the previous one) and each requires an expensive network evaluation, the overall cost is $\mathcal{O}(LK)$ sequential network passes, making sampling computationally slow.

3.5 Summary: A Comparative View of NCSN and DDPM

We begin by comparing the graphical models of NCSN and DDPM in Figure 3.7, with key differences and similarities summarized in Table 3.1.

Table 3.1: Comparisons of NCSN and DDPM

	NCSN	**DDPM**
$\mathbf{x}_{i+1}\|\mathbf{x}_i$	Derive as $\mathbf{x}_{i+1} = \mathbf{x}_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\boldsymbol{\epsilon}$	Define as $\mathbf{x}_{i+1} = \sqrt{1-\beta_i}\mathbf{x}_i + \sqrt{\beta_i}\boldsymbol{\epsilon}$
$\mathbf{x}_i\|\mathbf{x}$	Define as $\mathbf{x}_i = \mathbf{x} + \sigma_i\boldsymbol{\epsilon}$	Derive as $\mathbf{x}_i = \bar{\alpha}_i\mathbf{x} + \sqrt{1-\bar{\alpha}_i^2}\boldsymbol{\epsilon}$
p_{prior}	$\mathcal{N}(\mathbf{0}, \sigma_L^2\mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$
Loss	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_i, \sigma_i) + \frac{\boldsymbol{\epsilon}}{\sigma_i}\right\|_2^2\right]$	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2\right]$
Sampling	Apply Langevin per layer; use output to initialize the next	Traversing the Markovian chain with $p_{\boldsymbol{\phi}^\times}(\mathbf{x}_{i-1}\|\mathbf{x}_i)$

A Shared Bottleneck. Despite their different formulations, both NCSN and DDPM rely on dense time discretization. This leads to a critical limitation: sampling often requires hundreds or even thousands of iterations, making generation slow and computationally intensive.

Question 3.5.1

How can we accelerate sampling in diffusion models?

We return to this challenge in Chapter 9, which develops more efficient samplers; Chapter 10, which compresses pretrained diffusion models requiring hundreds of sampling steps into fewer-step student generators; and Chapter 11, which studies how to learn fast few-step generators directly from scratch under diffusion-inspired principles.

3.6 Closing Remarks

This chapter has charted a second major path to diffusion models, beginning from the score-based perspective rooted in Energy-Based Models (EBMs). We started by identifying the core challenge of EBMs—the intractable partition function—and introduced the score function, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, as a powerful tool that circumvents this issue entirely.

Our journey led us from classic score matching to its more scalable and robust variant, Denoising Score Matching (DSM). Through DSM, we saw how perturbing data with noise enables a tractable training objective, once again leveraging a conditioning strategy to create a simple regression target. Furthermore, we established a profound connection between score estimation and the act of denoising via Tweedie's formula, which showed that the score provides the precise direction needed to estimate a clean signal from its noisy observation.

This principle was then extended from a single noise level to a *ladder* of noise levels with Noise Conditional Score Networks (NCSN), which learn a single score model conditioned on multiple noise scales and generate samples via annealed Langevin dynamics. The next chapter takes the additional step from this discrete ladder to a continuum of noise levels through the Score SDE framework.

This convergence is no coincidence; it hints at a deeper, unified mathematical structure. The limitations of these discrete-time models motivate the need for a more general framework. In the next chapter, we will take this crucial step:

1. We will move into a continuous-time perspective, showing that both DDPMs and NCSNs can be elegantly unified as different discretizations of a single, powerful process described by a Stochastic Differential Equation (SDE).

2. This Score SDE framework will formally connect the variational and score-based views, recasting the problem of generation as one of solving a differential equation.

This unifying lens will not only provide profound theoretical clarity but also unlock a new class of advanced numerical methods designed to tackle the fundamental challenge of slow sampling.

4

Diffusion Models Today: Score SDE Framework

> *There is only one precise way of presenting the laws, and that is by means of differential equations. They have the advantage of being fundamental and, so far as we know, precise.*
>
> Richard P. Feynman

So far, we have studied diffusion models from two perspectives: the variational view and the score-based view, the latter naturally emerging from the EBM formulation. We now take the next step and move to the *continuous-time framework*. At its core lies the *Score SDE* (Song *et al.*, 2020c), which provides a continuous-time view that brings DDPM and NCSN under the same framework.

Instead of working with only a finite sequence of noise levels, we imagine inserting more and more intermediate levels between data and noise. In the continuous-time limit, the noise level varies smoothly with time, giving the basic picture

$$p_{\text{data}} \xrightarrow{\text{forward SDE}} p_T \approx p_{\text{prior}}, \qquad p_{\text{prior}} \xrightarrow{\text{reverse dynamics}} p_{\text{data}}.$$

The forward SDE describes how data is gradually corrupted into noise, playing the continuous-time role of the noise schedule in DDPM or the sequence of noise levels in NCSN. Once this forward process is specified, the main challenge is generation in the opposite direction: starting from a simple noise distribution, how can we recover the data distribution?

The key quantity is the time-dependent score

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}),$$

which describes the local geometry of the noisy distribution at every time t. Once this score is known, or approximated by a neural network, it determines the reverse-time dynamics used for generation, transforming noise into data. This

leads to two complementary descriptions: a stochastic *reverse-time SDE* and a deterministic *probability flow ODE* (PF-ODE). Although their sample trajectories are different, they can follow the same evolution of marginal distributions.

This continuous-time view turns generation into the problem of solving differential equations, opening the door to a broad set of tools from stochastic processes, differential equations, and numerical analysis. In particular, once the generative dynamics are written as an SDE or ODE, standard numerical methods can be used to simulate them, while more advanced solvers can improve sampling efficiency and accuracy. This perspective therefore provides not only a unified mathematical description of diffusion models, but also a practical foundation for understanding and designing faster sampling methods developed later in this book.

After this chapter, you will be able to...

- understand how the discrete noise levels of DDPM and NCSN lead to a continuous-time forward SDE that gradually transforms data into noise;
- understand how generation from noise back to data can be described either by a stochastic reverse-time SDE or by a deterministic probability flow ODE (PF-ODE);
- explain how the reverse-time SDE and PF-ODE can follow the same sequence of marginal distributions even though their individual sample trajectories are different, and understand how the Fokker–Planck equation describes the continuous-time evolution of probability densities;
- understand how the time-dependent score is learned by denoising score matching and then used for sampling through the reverse-time SDE or PF-ODE;
- understand how the PF-ODE can also be run forward for inversion and track changes in probability density through the continuous change-of-variables formula, enabling likelihood evaluation;
- recognize VE and VP SDEs as two important instances of the same continuous-time diffusion framework, connecting back to NCSN and DDPM.

Reader route. Readers new to stochastic differential equations may consult the differential-equation appendices in Chapter A as needed. For a first pass, focus on the roles of the forward SDE, reverse-time SDE, and PF-ODE, together with their training and sampling procedures. The Gaussian example and optional derivations can be revisited after the main picture is clear.

4.1 Score SDE: Its Principles

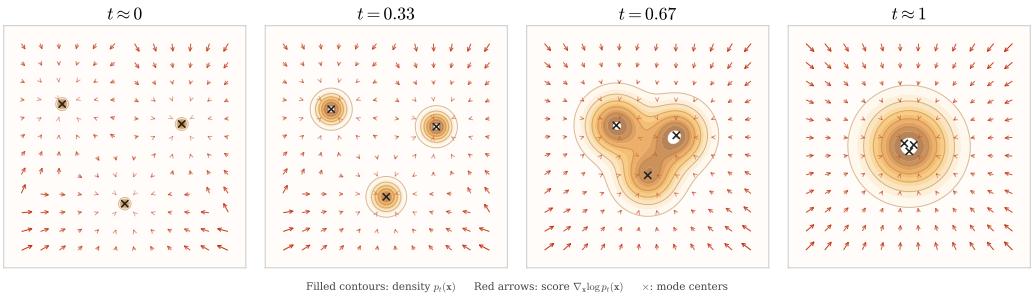


Figure 4.1: Illustration of the time-dependent score landscape. As the noise level increases, the perturbed density $p_t(\mathbf{x})$ evolves from a multimodal distribution (data) concentrated near the data modes to a broad nearly unimodal distribution (prior). Accordingly, the score field $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ changes from a sharply local vector field that pulls samples toward nearby modes to a smooth global field that points toward the overall center of mass. This time-dependent score field is the key object learned in score-based diffusion models (i.e., Score SDE).

Source: Created by the authors with AI-assisted coding.

The use of multiple noise scales has been a crucial ingredient in the success of NCSN and DDPM frameworks. In this section, we introduce the foundation of the *Score SDE* (Song *et al.*, 2020c), which elevates this idea by considering a continuum of noise levels. A continuous-time limit of forward and reverse diffusion processes had already been noted by Sohl-Dickstein *et al.* (2015), but Song *et al.* (2020c) make this perspective central by formulating the data evolution as a stochastic/ordinary differential equation, where the noise level increases smoothly over time. Central to this formulation is the *time-dependent score function* $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$: at each noise level t, it points toward regions of higher density under the current noisy distribution p_t, and it is precisely this quantity that determines how to reverse the diffusion. As Figure 4.1 illustrates, the score field evolves from sharply converging toward individual data modes at small t to pointing gently toward the global center at large t. This continuous-time formulation not only unifies prior discrete-time models but also provides a principled and flexible foundation for generative modeling: learning this family of score functions across all noise levels reduces generation to the problem of solving differential equations.

4.1.1 Motivation: From Discrete to Continuous-Time Processes

We revisit the forward noise injection schemes of NCSN and DDPM. NCSN uses a sequence of increasing noise levels $\{\sigma_i\}_{i=1}^{L}$. Each clean sample $\mathbf{x}_0 \sim p_{\text{data}}$ is perturbed as

$$\mathbf{x}_{\sigma_i} = \mathbf{x}_0 + \sigma_i \boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

DDPM instead injects noise incrementally with a variance schedule $\{\beta_i\}_{i=1}^{L}$[1]:

$$\mathbf{x}_i = \sqrt{1-\beta_i}\,\mathbf{x}_{i-1} + \sqrt{\beta_i}\,\boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

We view them together on a discrete time grid, where the sequential update from $\mathbf{x}_t$ to $\mathbf{x}_{t+\Delta t}$ takes the form[2]:

$$\textbf{NCSN: } \mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \sqrt{\sigma_{t+\Delta t}^2 - \sigma_t^2}\boldsymbol{\epsilon}_t \qquad \approx \mathbf{x}_t + \sqrt{\frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t}\Delta t}\boldsymbol{\epsilon}_t$$

$$\textbf{DDPM: } \mathbf{x}_{t+\Delta t} = \sqrt{1-\beta(t)\Delta t}\mathbf{x}_t + \sqrt{\beta(t)\Delta t}\boldsymbol{\epsilon}_t \quad \approx \mathbf{x}_t - \frac{1}{2}\beta(t)\mathbf{x}_t\Delta t + \sqrt{\beta(t)\Delta t}\boldsymbol{\epsilon}_t,$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The DDPM approximation uses the Taylor expansion of $h(x) = \sqrt{1-x}$ about 0: $h(x) = h(0) + h'(0)x + \cdots \approx 1 - \frac{1}{2}x$, with $x = \beta(t)\Delta t$. Interestingly, both noise injection processes follow a common structural pattern:

$$\mathbf{x}_{t+\Delta t} \approx \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t + g(t)\sqrt{\Delta t}\,\boldsymbol{\epsilon}_t, \tag{4.1.1}$$

with $\mathbf{f} : \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$ and $g : \mathbb{R} \to \mathbb{R}$ given by:

$$\textbf{NCSN:} \quad \mathbf{f}(\mathbf{x}, t) = 0, \qquad g(t) = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}$$

$$\textbf{DDPM:} \quad \mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\,\mathbf{x}, \qquad g(t) = \sqrt{\beta(t)}.$$

This formulation corresponds to the following Gaussian transition:

$$p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t) := \mathcal{N}\Big(\mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t,\, g^2(t)\Delta t\,\mathbf{I}\Big), \tag{4.1.2}$$

where, by a slight abuse of notation, we treat $\mathbf{x}_t$ as a fixed sample and $\mathbf{x}_{t+\Delta t}$ as a random variable.

As $\Delta t \to 0$ (which can be conceptually understood as preparing *infinitely many* layers of noises), the discrete-time process converges to a continuous time SDE evolving forward in time[3]:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t),$$

[1] In Section 2.2, the DDPM forward step is written as $\mathbf{x}_i = \sqrt{1-\beta_i^2}\,\mathbf{x}_{i-1} + \beta_i\,\boldsymbol{\epsilon}_i$, where β_i denotes the standard deviation of the injected noise and $\alpha_i := \sqrt{1-\beta_i^2}$. Here and throughout this chapter, we follow the convention of Song *et al.* (2020c), where β_i instead denotes the variance of the injected noise, so that $\mathbf{x}_i = \sqrt{1-\beta_i}\,\mathbf{x}_{i-1} + \sqrt{\beta_i}\,\boldsymbol{\epsilon}_i$. The two conventions are related by $\beta_i^{(\text{here})} = \big(\beta_i^{(\text{Ch. 2})}\big)^2$ and yield identical dynamics.

[2] For convenience, we use $\mathbf{x}(t)$ and $\mathbf{x}_t$ interchangeably (and similarly for other time-dependent variables) to denote samples at time t.

[3] The forward kernel in Equation (4.1.2) converges, as $\Delta t \to 0$, to the solution of the corresponding Itô SDE. A fully rigorous proof relies on advanced results which we defer to the literature.

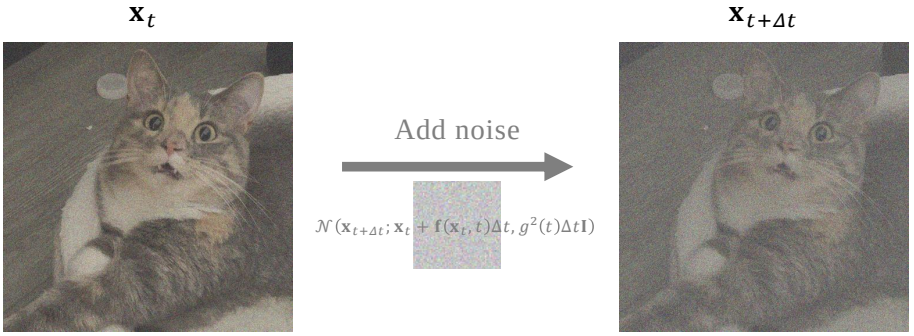


Figure 4.2: Illustration of the discrete-time noise-adding step. It adds noise from t to $t + \Delta t$ with mean drift $\mathbf{f}(\mathbf{x}_t, t)$ and diffusion coefficient $g(t)$.

Source: Created by the authors.

where $\mathbf{w}(t)$ is a standard Wiener process (or Brownian motion).

Remark.

> While a full formal definition is not necessary here, a *Wiener process* is a continuous-time stochastic process $\mathbf{w}(t)$ that starts at zero, has independent increments, and satisfies that for any $s < t$, the increment $\mathbf{w}(t) - \mathbf{w}(s)$ is normally distributed with mean zero and variance $t - s$. It represents the accumulation of independent Gaussian fluctuations over time, and although it is almost surely continuous, it is nowhere differentiable.
>
> Over an infinitesimal time interval $[t, t + \mathrm{d}t]$, the increment of a Wiener process is defined as
>
> $$\mathrm{d}\mathbf{w}(t) := \mathbf{w}(t + \mathrm{d}t) - \mathbf{w}(t),$$
>
> which is modeled as a Gaussian random variable with zero mean and variance $\mathrm{d}t$:
>
> $$\mathrm{d}\mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, \mathrm{d}t\mathbf{I}).$$

A brief introduction to the foundations of SDEs is provided in Section A.2, with a more advanced discussion in Chapter C. However, we can conceptually understand the connection between the discrete and continuous formulations as follows:

- $\mathbf{x}(t + \Delta t) - \mathbf{x}(t) \approx \mathrm{d}\mathbf{x}(t)$,
- $\Delta t \approx \mathrm{d}t$,
- $\sqrt{\Delta t}\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \Delta t\mathbf{I}) \approx \mathrm{d}\mathbf{w}(t)$.

Brownian motion is the continuous analogy to a discrete Gaussian walk: if $p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_{t-1}, \mathbf{I})$, then $p(\mathbf{x}_{t+n}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t+n}; \mathbf{x}_t, n\mathbf{I})$. Once the drift $\mathbf{f}(\mathbf{x}, t)$ and diffusion $g(t)$ are specified, the forward time SDE automatically induces a reverse time SDE that transports the terminal noise distribution back to the data distribution. The reverse dynamics involve only a single unknown term, surprisingly the *score function at each continuous-time level*. This identifies score matching as the training objective; once the score is learned, sampling amounts to numerically integrating the reverse time SDE with the learned score.

While Section 4.3 presents practical implementations, we first examine the theoretical foundations of the forward and reverse processes in Section 4.1.2 and Section 4.1.3.

4.1.2 Forward-Time SDEs: From Data to Noise

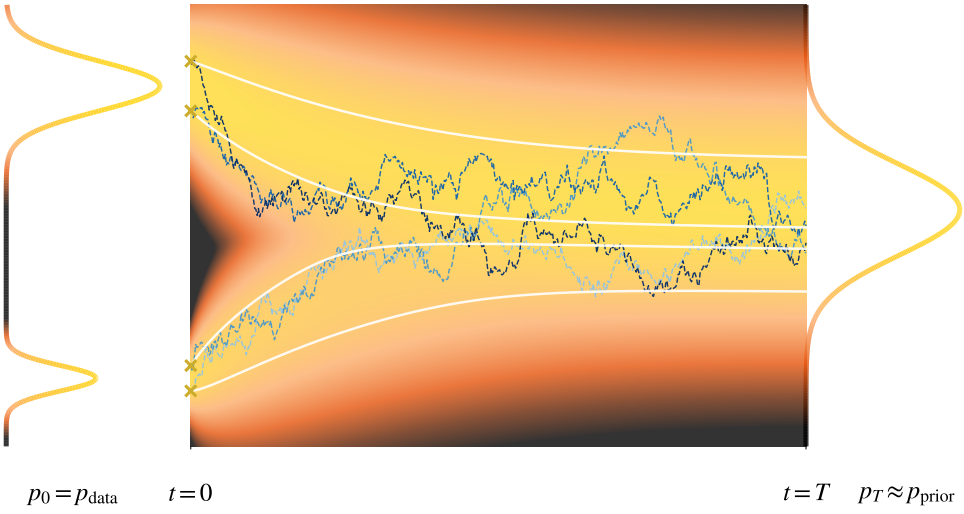


Figure 4.3: (1D) Visualization of the forward process in a diffusion model. The process starts from initial points sampled (denoted as "×") from a complex bimodal data distribution ($p_0 = p_{\text{data}}$) and evolves toward a simple, unimodal Gaussian prior ($p_T \approx p_{\text{prior}}$). The background heatmap illustrates the evolving marginal probability density, p_t, which smooths over time. Sample trajectories are shown evolving from $t = 0$ to $t = T$, comparing the stochastic forward SDE process (blue paths) with its deterministic counterpart, the PF-ODE (white paths). Note that the PF-ODE is a deterministic transport map for densities, not generally the mean of sample paths started from a single point.

Source: Created by the authors.

With this formulation, earlier methods based on discrete time, such as NCSN (Song and Ermon, 2019) and DDPM (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020), can be unified under the *continuous-time* framework through a stochastic process $\mathbf{x}(t)$ governed by a forward SDE defined on the interval $[0, T]$:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t), \quad \mathbf{x}(0) \sim p_{\text{data}}. \tag{4.1.3}$$

Here, $\mathbf{f}(\cdot, t)\colon \mathbb{R}^D \to \mathbb{R}^D$ is the drift, $g(t) \in \mathbb{R}$ is the scalar diffusion coefficient, and $\mathbf{w}(t)$ denotes a standard Wiener process. We refer to this as the *forward SDE*, which describes how clean data is gradually perturbed into noise over time.

Once the drift $\mathbf{f}$ and diffusion coefficient g are specified, the forward process is fully determined, describing how the data variable is progressively corrupted through the injection of Gaussian noise. In particular, two families of time-dependent densities are induced:

Perturbation Kernels. The conditional law

$$p_t(\mathbf{x}_t|\mathbf{x}_0)$$

describes how a clean data sample $\mathbf{x}_0 \sim p_{\text{data}}$ evolves into its noisy counterpart $\mathbf{x}_t$ at time t. In general, the drift term $\mathbf{f}(\mathbf{x}, t)$ in Equation (4.1.3) can be an arbitrary function of $\mathbf{x}$, but a common and analytically convenient choice is to assume it is affine:

$$\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x}, \tag{4.1.4}$$

where $f(t)$ is a scalar function of t, typically taken to be non-positive. Under this structure, the conditional process remains Gaussian given $\mathbf{x}_0$, and the perturbation kernel admits a closed-form solution obtained by solving the associated mean–variance ODEs (Särkkä and Solin, 2019) see also Section 4.4.3). In particular,

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{m}(t), P(t)\mathbf{I}_D),$$

with

$$\mathbf{m}(t) = \exp\Big(\int_0^t f(u)\,\mathrm{d}u\Big)\mathbf{x}_0, \quad P(t) = \int_0^t \exp\Big(2\int_s^t f(u)\,\mathrm{d}u\Big)g^2(s)\,\mathrm{d}s,$$

and initial conditions $\mathbf{m}(0) = \mathbf{x}_0$, $P(0) = 0$.

This explicit form allows one to sample $\mathbf{x}_t$ given $\mathbf{x}_0$ directly, without numerically integrating the SDE, hence the term *simulation-free*. Both NCSN and DDPM fall into this affine-drift setting.

In the remainder, we develop the general theory for arbitrary drifts $\mathbf{f}(\mathbf{x}, t)$, but will return to the affine drift when closed-form analysis is useful.

Marginal Densities. The time-marginal density $p_t(\mathbf{x}_t)$ is obtained by integrating over the perturbation kernel:

$$p_t(\mathbf{x}_t) := \int p_t(\mathbf{x}_t|\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0, \quad \text{with } p_0 = p_{\text{data}}. \tag{4.1.5}$$

By choosing the coefficients $f(t)$ and $g(t)$ appropriately, the forward process gradually adds noise until the influence of the initial state is effectively forgotten. As T becomes large, the conditional distribution $p_T(\mathbf{x}_T|\mathbf{x}_0)$ no longer depends on $\mathbf{x}_0$, because its mean evolves as

$$\mathbf{m}(T) = \exp\Big(\int_0^T f(u)\,\mathrm{d}u\Big)\mathbf{x}_0 \longrightarrow \mathbf{0}, \quad \text{as } T \to \infty,$$

provided $f(u)$ is non-positive so that the exponential factor decays. At the same time, the variance grows and stabilizes to match a chosen prior distribution. Consequently, the marginal

$$p_T(\mathbf{x}_T) = \int p_T(\mathbf{x}_T|\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0,$$

which initially represents a complicated mixture over data samples, converges to a simple prior p_{prior}, typically Gaussian. In this limit,

$$p_T(\mathbf{x}_T) \approx p_{\text{prior}}(\mathbf{x}_T) \quad \text{and} \quad p_T(\mathbf{x}_T|\mathbf{x}_0) \approx p_{\text{prior}}(\mathbf{x}_T),$$

so the forward process maps any data distribution into a tractable prior, providing a convenient starting point for reversal and generation.

4.1.3 Reverse-Time Stochastic Process for Generation

Intuitively, data generation from noise can be achieved by "reversing" the forward process: starting from a random point sampled from the prior distribution and evolving it backward in time to obtain a generated sample. For deterministic systems (that is, ODEs), this idea works naturally. Since no randomness is involved, reversing time simply means tracing the trajectory of a point in the opposite direction along the same path as in the forward process[4]. In contrast, SDEs incorporate stochasticity at every time step, meaning that a single point can evolve along many plausible random trajectories. As a result, reversing such processes is more subtle[5].

While individual stochastic trajectories are not reversible, the remarkable insight is that the distribution over these trajectories can be reversed. This is formalized by a foundational result from Anderson (1982), which shows that the time-reversed process $\{\bar{\mathbf{x}}(t)\}_{t\in[0,T]}$[6] of the forward process in Equation (4.1.3) is itself governed by a well-defined SDE. This reverse-time process evolves from T to 0, and its dynamics are given by:

[4] Technically, this corresponds to solving the ODE with a time-flipping substitution $t \leftrightarrow T - t$.

[5] Naively flipping time does not yield the correct reverse process.

[6] We use the "bar" notation to distinguish the reverse process $\{\bar{\mathbf{x}}(t)\}_{t\in[0,T]}$ from the forward process $\{\mathbf{x}(t)\}_{t\in[0,T]}$, defined by the forward-time SDE.

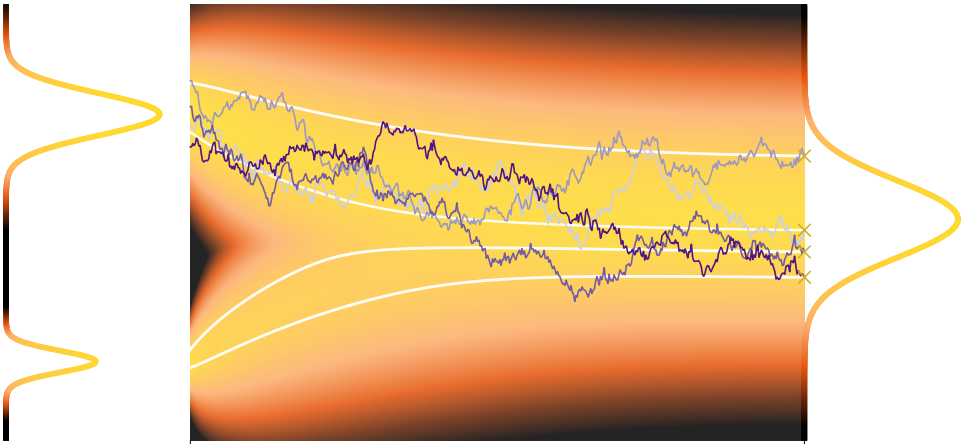


Figure 4.4: Visualization of the reverse-time stochastic process for data generation. It begins from samples drawn from a simple prior distribution (p_{prior}) at $t = T$ (denoted as "$\times$"), which are evolved backward in time using a reverse-SDE. The resulting trajectories terminate at $t = 0$ and collectively form the target bimodal data distribution ($p_0 = p_{\text{data}}$). The background heatmap illustrates how the probability density is gradually transformed from a simple Gaussian into the complex target distribution.

Source: Created by the authors.

$$\mathrm{d}\bar{\mathbf{x}}(t) = \Big[\mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))\Big]\,\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}(t), \qquad (4.1.6)$$
$$\bar{\mathbf{x}}(T) \sim p_{\text{prior}} \approx p_T.$$

Here, $\bar{\mathbf{w}}(t)$ denotes a standard Wiener process in reverse time, defined as $\bar{\mathbf{w}}(t) := \mathbf{w}(T - t) - \mathbf{w}(T)$.

Remark.

> Throughout this chapter, t denotes the original diffusion clock: $t = 0$ corresponds to data and $t = T$ to noise. Accordingly, the reverse-time SDE in Equation (4.1.6) is integrated from $t = T$ down to $t = 0$, so $\mathrm{d}t < 0$ along the reverse process. Equivalently, one may introduce a forward-running reverse clock $s := T - t$; unless stated otherwise, we keep the original clock t.

To build intuition for Equation (4.1.6), we present a concrete example in Section 4.2.2 with a Gaussian data distribution and linear–Gaussian dynamics. This setting is analytically tractable: one can derive the time-reversal formula directly using basic calculus and linear algebra, without invoking the full general theory of Anderson (1982).

Note that the presence of stochasticity ($g \neq 0$) introduces an additional correction term, $-g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))$, which accounts for the effect of diffusion and ensures that the reversed dynamics correctly reproduce the evolution of marginal distributions induced by the forward SDE (see Section 4.2).

Conceptually, Why Does the Reverse Process Work? Section 4.6.2 presents an intuitive derivation of the reverse-time SDE by connecting it to the DDPM variational framework (optional but insightful). Here, we provide complementary intuition for how the reverse-time dynamics recover structured data from noise.

At first glance, the presence of Brownian noise in the reverse time process may seem paradoxical. If the forward diffusion spreads data into increasingly noisy configurations, it is unclear how reversing this process, particularly one that introduces additional randomness through $\bar{\mathbf{w}}(t)$, can produce clean, structured samples concentrated near the data manifold. The key point is that the reverse time SDE does not inject arbitrary randomness. The diffusion term $g(t)\,\mathrm{d}\bar{\mathbf{w}}(t)$ is always coupled with the score–driven drift $-g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))$. Together, these terms balance one another: the score guides trajectories toward regions of higher density, while the noise introduces controlled stochasticity that allows exploration without overwhelming the dynamics.

To see this more clearly, return to the Langevin intuition in Equation (3.1.6). When $f(t) \equiv 0$, Equation (4.1.6) reads

$$\mathrm{d}\bar{\mathbf{x}}(t) = -g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))\,\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}(t).$$

Reparameterize time forward via $s := T - t$ (so $\mathrm{d}t = -\mathrm{d}s$), and rename the Brownian motion in law so that $\mathrm{d}\bar{\mathbf{w}}(t) = -\,\mathrm{d}\mathbf{w}_s$. Writing $\bar{\mathbf{x}}_s := \bar{\mathbf{x}}(T-s)$ and $\pi_s := p_{T-s}$ then gives

$$\begin{aligned}\mathrm{d}\bar{\mathbf{x}}_s &= g^2(T-s)\nabla \log \pi_s(\bar{\mathbf{x}}_s)\,\mathrm{d}s + g(T-s)\,\mathrm{d}\mathbf{w}_s \\ &= 2\tau(s)\nabla \log \pi_s(\bar{\mathbf{x}}_s)\,\mathrm{d}s + \sqrt{2\tau(s)}\,\mathrm{d}\mathbf{w}_s, \quad \tau(s) := \tfrac{1}{2}g^2(T-s).\end{aligned}$$

This has the Langevin form with a time-varying temperature $\tau(s)$, targeting the evolving density π_s. By Tweedie's formula (Equation (3.3.6)), the score direction $\nabla \log \pi_s$ points toward the conditional clean signal at each time slice, so the drift continually "pulls back" denoised structure.

Crucially, $g(t)$ is *annealing* along the reverse trajectory. Early on ($s \approx 0$, i.e., $t \approx T$), $g(T-s)$ is typically larger, so the injected noise is stronger and the process explores broadly. As s increases, $g(T-s)$ decreases, the stochastic term weakens, and the score term dominates, pulling samples into high-density regions of π_s; by $s = T$ (i.e., $t = 0$), trajectories concentrate near the data manifold.

Overview of Reverse-Time SDE Capabilities. It is fascinating how the time-dependent score function

$$\mathbf{s}(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$$

naturally appears in Equation (4.1.6). Once the forward coefficients $f(t)$ and $g(t)$ are specified, the score is the only remaining unknown in the reverse dynamics. This highlights its central role: with the score in hand, the reverse process is determined, and sampling amounts to numerically integrating Equation (4.1.6) with the learned score.

Since the oracle score generally lacks a closed-form expression, we adopt the approach of Chapter 3 and train a neural network $\mathbf{s}_{\phi}(\mathbf{x}, t)$ to approximate it via score matching; see Section 4.3.1 for details. Substituting $\mathbf{s}(\mathbf{x}, t)$ with $\mathbf{s}_{\phi}(\mathbf{x}, t)$ in Equation (4.1.6) then specifies the reverse dynamics completely.

Generation corresponds to solving the reverse-time SDE reversely from $t = T$, starting with $\mathbf{x}_T \sim p_{\text{prior}}$, to $t = 0$. Importantly, Anderson (1982) proves that the marginal densities of the forward and reverse processes coincide, ensuring that samples at $t = 0$ approximately follow p_{data} when $p_{\text{prior}} \approx p_T$. We will explore this further in Section 4.3.2.

4.1.4 Deterministic Process (Probability Flow ODE) for Generation

Although the SDE in Equation (4.1.6) introduces stochasticity and potentially increases the diversity of generated samples, a question arises:

Question 4.1.1

Is it necessary to sample using the SDE in Equation (4.1.6)?

Inspired by Maoutsa *et al.* (2020), Song *et al.* (2020c) also introduced a deterministic process, an ODE, that evolves samples with the same marginal distributions as the forward SDE. This process $\{\tilde{\mathbf{x}}(t)\}_{t\in[0,T]}$[7], called the *Probability Flow ODE (PF-ODE)*, is given by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\tilde{\mathbf{x}}(t) = \mathbf{f}(\tilde{\mathbf{x}}(t), t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\tilde{\mathbf{x}}(t)). \tag{4.1.7}$$

Analogous to the SDE case, one can replace the true score with a learned approximation and integrate the reverse-time ODE from $t = T$ to $t = 0$ to generate samples. Concretely, the generated sample (solution of PF-ODE at time $t = 0$) takes the form

$$\tilde{\mathbf{x}}(T) + \int_T^0 \left[\mathbf{f}(\tilde{\mathbf{x}}(\tau), \tau) - \frac{1}{2} g^2(\tau) \nabla_{\mathbf{x}} \log p_\tau(\tilde{\mathbf{x}}(\tau))\right] \mathrm{d}\tau,$$

[7] We use a tilde to distinguish processes associated with the forward and reverse-time SDEs. Going forward, we omit this notational distinction for simplicity.

where the initial condition $\tilde{\mathbf{x}}(T) \sim p_{\text{prior}}$. Since this integral is intractable in closed form, practical generation relies on numerical solvers (e.g., Euler method, see Equation (4.3.4)).

Compared to the reverse-time SDE, the PF-ODE offers two key advantages:

- The ODE can be integrated in either direction, from $t = 0$ to $t = T$ or from $t = T$ to $t = 0$, using the same formulation of the equation, provided the corresponding initial condition is specified at the chosen endpoint. This bidirectionality contrasts with SDEs, which generally admit only forward time integration.
- It benefits from a wide range of well-established, off-the-shelf numerical solvers developed for ODEs.

We emphasize that the PF-ODE is not obtained by simply removing the diffusion term in Equation (4.1.6); notably, the factor of $\frac{1}{2}$ in its drift term has a principled origin. At a high level, Equation (4.1.7) arises by choosing the drift of an ODE such that its evolution preserves the same marginal densities as the forward SDE in Equation (4.1.3). The underlying principle (i.e., Fokker-Planck Equation (Øksendal, 2003)) ensuring this alignment of marginals will be detailed in the next section.

4.2 Matching Marginals in Forward/Reverse-Time SDEs and PF-ODE

In this section, we present the *Fokker-Planck equation* as the central principle governing diffusion models. It describes how probability densities evolve under noisy dynamics and serves as the natural analogue of a change-of-variables formula when randomness is present (see Chapter B). Starting from the forward diffusion process, which defines the marginals p_t, we explain how both the reverse-time SDE and the PF-ODE are designed to reproduce the same marginal family. We then make this principle concrete through a Gaussian example, where the reverse-time SDE and the PF-ODE can be computed directly and explicitly.

4.2.1 Fokker-Planck Equation to Ensure Alignment of Marginal Densities

A central concept in diffusion models is that *different processes can lead to the same sequence of marginal distributions* (as we will illustrate later in this subsection). The objective is to construct a process that transforms p_{prior} into p_{data} by aligning the marginals across time, and in particular at $t = 0$.

For example, take Process 1: sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ once and set $\mathbf{x}_0 = \mathbf{z}, \ldots, \mathbf{x}_t = \mathbf{z}$; and Process 2: $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \ldots, \mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ all independent. Processes 1 and 2 differ (Process 1 is constant over time), but they have the same marginal distribution at every point in time: $\mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The exact form of the process is secondary, provided it is tractable and admits efficient sampling. This naturally leads to a fundamental question:

Question 4.2.1

How can we ensure that different processes yield identical marginal distributions?

Returning to our setup, once the forward SDE is specified, it defines the evolution of marginal densities from p_{data} to p_{prior}. The reverse-time SDE and PF-ODE are then constructed so that their trajectories yield marginal distributions that exactly match those of the forward process. The key to this correspondence lies in the Fokker–Planck equation, which governs how marginal densities evolve under diffusion processes. An illustration is provided in Figure 4.5. The following theorem (Anderson, 1982; Song *et al.*, 2020c) establishes the foundation for this connection:

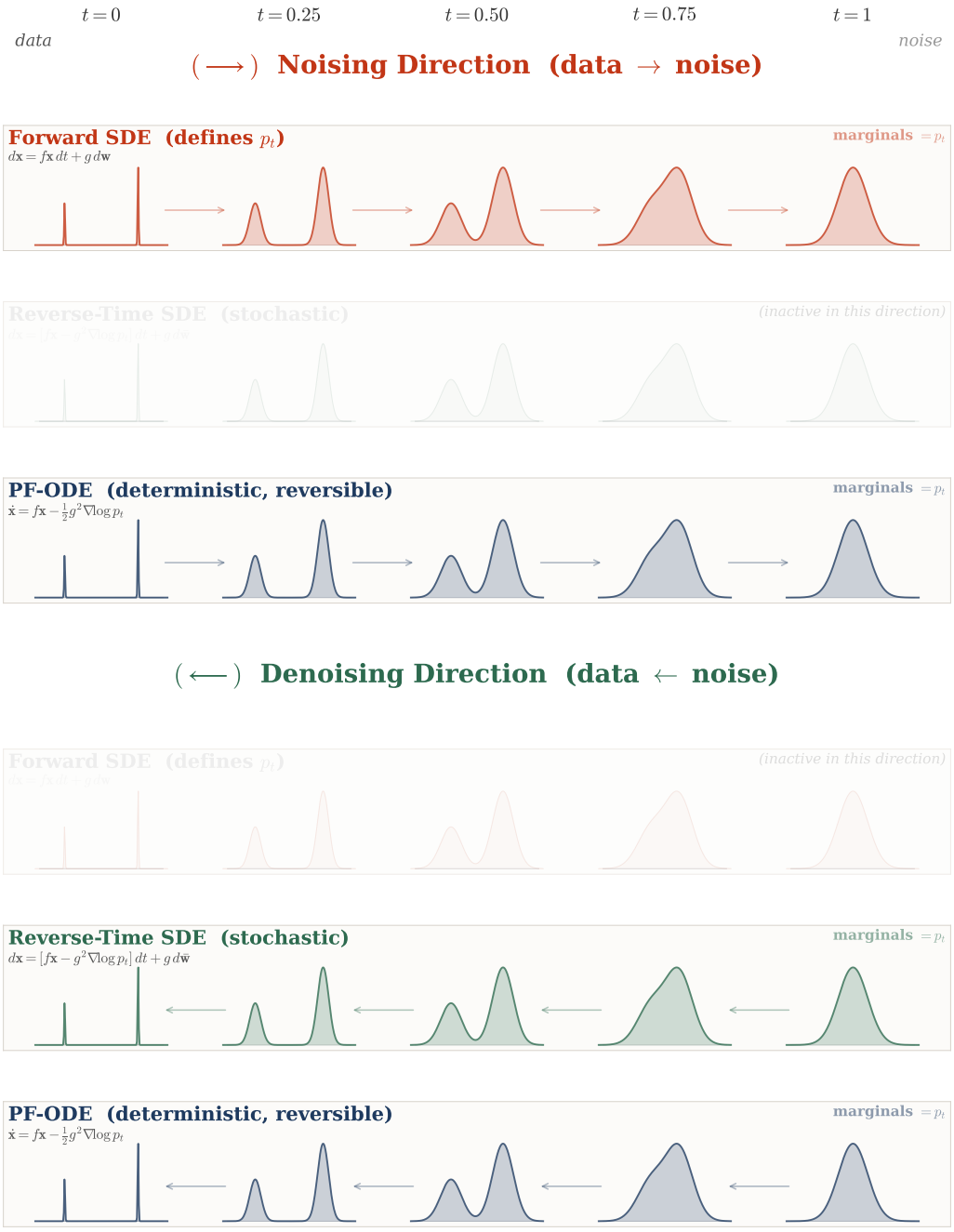


Figure 4.5: Three dynamics sharing the same marginals p_t. Upper panel: noising direction. The forward SDE, which serves as an anchor reference, defines the marginals p_t by diffusing data toward a Gaussian prior. The PF-ODE follows the same marginal path deterministically when *run forward in time*. **Lower panel: denoising direction.** The reverse-time SDE and the PF-ODE *run backward in time* yield the same marginals in reverse, as guaranteed by the Fokker–Planck equation. The reverse-time SDE gives a stochastic denoising process, whereas the PF-ODE gives a deterministic one. Thus, the repeated density snapshots are intentional: the processes agree at the level of one-time marginals, but differ in their sample-level dynamics. The PF-ODE appears in both panels because it is a deterministic flow that can be integrated in either time direction.

Source: Created by the authors with AI-assisted coding.

Theorem 4.2.1: Marginal Alignment via the Fokker–Planck Equation

Consider the forward SDE

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t), \qquad \mathbf{x}(0) \sim p_0 = p_{\text{data}}.$$

Under standard regularity conditions, its marginal density p_t evolves according to the Fokker–Planck equation

$$\begin{aligned} \partial_t p_t(\mathbf{x}) &= -\nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})\right] + \tfrac{1}{2} g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}) \\ &= -\nabla_{\mathbf{x}} \cdot \left[\mathbf{v}(\mathbf{x}, t) p_t(\mathbf{x})\right], \end{aligned} \tag{4.2.1}$$

where $\Delta_{\mathbf{x}}$ denotes the Laplacian operator and

$$\mathbf{v}(\mathbf{x}, t) := \mathbf{f}(\mathbf{x}, t) - \tfrac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Importantly, the same family of marginal distributions $\{p_t\}_{t\in[0,T]}$ can be realized by two different dynamics:

(i) The *PF-ODE*

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{v}(\tilde{\mathbf{x}}(t), t).$$

Starting from $\tilde{\mathbf{x}}(0) \sim p_0$ and solving forward from 0 to T gives $\tilde{\mathbf{x}}(t) \sim p_t$. Equivalently, starting from $\tilde{\mathbf{x}}(T) \sim p_T$ and solving the same ODE backward from T to 0 follows the same marginals in reverse.

(ii) The *reverse-time SDE*

$$\mathrm{d}\bar{\mathbf{x}}(t) = \left[\mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))\right] \mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}(t).$$

Starting from $\bar{\mathbf{x}}(T) \sim p_T$, this SDE is solved backward in time from $t = T$ to $t = 0$. Along this reverse evolution, its marginal distribution at time t is

$$\bar{\mathbf{x}}(t) \sim p_t.$$

Thus, the reverse-time SDE and PF-ODE can have the same marginal distribution at every noise level even though one is stochastic and the other is deterministic.

Proof for Theorem.

The proof is provided in Section D.2.5, while Section 4.6.1 offers further intuition behind the Fokker–Planck equation using the marginalization technique of probability.

Multiple Conditional Distributions for a Fixed Marginal. To understand how the PF-ODE transports p_{data} forward in time (or equivalently p_{prior} in reverse), consider the *flow map* $\boldsymbol{\Psi}_{s\to t} : \mathbb{R}^D \to \mathbb{R}^D$, where $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$ denotes the PF-ODE solution at time t initialized from $\mathbf{x}_s$ at time s, for any time $s, t \in [0, T]$. In other words, this map takes an initial state $\mathbf{x}_s$ and directly jumps to its state at t:

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) \coloneqq \mathbf{x}_s + \int_s^t \mathbf{v}(\mathbf{x}_\tau, \tau)\mathrm{d}\tau, \tag{4.2.2}$$

with velocity field

$$\mathbf{v}(\mathbf{x}, \tau) := \mathbf{f}(\mathbf{x}, \tau) - \frac{1}{2} g^2(\tau)\nabla_{\mathbf{x}} \log p_\tau(\mathbf{x}).$$

Here, the integral captures the net displacement accumulated along the PF-ODE trajectory. The term *flow map* is indeed classical in differential equations, dynamical systems, and continuum mechanics: it denotes the map that carries a state from its position at time s to its position at time t under the given dynamics. The velocity field $\mathbf{v}(\mathbf{x}, t)$ describes the instantaneous motion, whereas the flow map $\boldsymbol{\Psi}_{s\to t}$ records the accumulated motion over a finite time interval (Arnold, 1988; Chorin and Marsden, 1993)[8]. We use the term in this standard mathematical and physical sense throughout the book, particularly when discussing fast diffusion-based generators in Chapters 10 and 11. Under mild smoothness assumptions on $\mathbf{v}$, the flow map $\boldsymbol{\Psi}_{s\to t} : \mathbb{R}^D \to \mathbb{R}^D$ is a smooth bijection[9].

For any $t \in [0, T]$, the *pushforward density* is defined as

$$p_t^{\text{fwd}}(\mathbf{x}_t) := \int \delta(\mathbf{x}_t - \boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0))\, p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0,$$

denoted $\boldsymbol{\Psi}_{0\to t}\sharp p_{\text{data}}$, representing the distribution at time t under $\boldsymbol{\Psi}_{0\to t}$. Theorem 4.2.1 ensures $p_t^{\text{fwd}} = p_t$, where p_t is the marginal density of the forward SDE, equating the deterministic PF-ODE and stochastic kernel:

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t|\mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0 = \int \delta(\mathbf{x}_t - \boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0))\, p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0.$$

This implies infinitely many conditionals $Q_t(\mathbf{x}_t|\mathbf{x}_0)$ yield the same $p_t(\mathbf{x}_t)$, for instance:

- **Stochastic (Simulation-Free):** $Q_t(\mathbf{x}_t|\mathbf{x}_0) = p_t(\mathbf{x}_t|\mathbf{x}_0)$,

[8] The physical idea underlying this map is much older. Eighteenth-century fluid mechanics already studied motion through velocity fields and by following the evolution of material particles, notably in the foundational works of Euler and Lagrange (Euler, 1757; Lagrange, 1783).

[9] Spoiler: the PF-ODE flow map $\boldsymbol{\Psi}_{s\to t}$ is exactly the Normalizing Flow (NF) bijection carrying p_s to p_t (to be detailed in Section 5.1). The difference is that, once the forward SDE and its marginal densities are fixed, the PF-ODE formula uniquely specifies its associated velocity field, $\mathbf{v}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, whereas a normalizing flow, or continuous-time normalizing flow, parameterizes and learns a velocity field directly. Both rely on the same continuous change-of-variables principle.

- **Deterministic (Requires ODE Solving):** $Q_t(\mathbf{x}_t|\mathbf{x}_0) = \delta(\mathbf{x}_t - \mathbf{\Psi}_{0\to t}(\mathbf{x}_0))$,
- **Mixture:** $Q_t(\mathbf{x}_t|\mathbf{x}_0) = \lambda p_t(\mathbf{x}_t|\mathbf{x}_0) + (1-\lambda)\delta(\mathbf{x}_t - \mathbf{\Psi}_{0\to t}(\mathbf{x}_0))$, $\lambda \in [0,1]$.

This nonuniqueness of $Q_t(\mathbf{x}_t|\mathbf{x}_0)$ arises from the fact that the marginal constraint does not uniquely determine the conditional distribution. This concept reappears in Section 5.2.2 and Section 9.2.3. In particular, there exists an entire family of reverse-time SDEs that are consistent with the same marginal p_t.

Observation 4.2.1: Matching Prescribed Marginal Densities

Multiple processes can give rise to the same sequence of marginal densities; what truly matters is satisfying the Fokker–Planck equation. This fundamental insight affords us remarkable flexibility in designing generative processes that transition from p_{prior} to p_{data}, or vice versa.

While marginal densities do not fully characterize the path distribution, our objective in generative modeling only requires modeling the marginals at $t = 0$ and $t = T$. The Fokker–Planck equation lies at the heart of diffusion models and is rooted in the fundamental *change-of-variable formula* for probability densities (see Chapter B for a systematic treatment). Far from being a minor technical detail, this principle recurs throughout our development, most notably in Section 5.2.

4.2.2 A Computable Example: Evolutions of Gaussian Dynamics

When p_{data} is a normal distribution (or a mixture of Gaussians), the score function admits a closed-form expression. This makes it an ideal setting for building intuition about diffusion processes: we can explicitly derive the reverse-time SDE and the PF-ODE using only basic calculus, without resorting to advanced mathematical tools. In this subsection, we illustrate how these equations behave in such a tractable case.

Exact Computation of the Reverse-Time SDE with a Gaussian. When p_{data} is Gaussian, the formula in Equation (4.1.6) can be derived directly, without relying on the general theory and proofs of Anderson (1982). To illustrate the core idea, we consider the one-dimensional case; the extension to higher dimensions follows in the same way.

Start from the forward SDE

$$\mathrm{d}x(t) = f(t)x(t)\,\mathrm{d}t + g(t)\,\mathrm{d}w_t,$$

and take one small Euler step of size $\Delta t > 0$:

$$x_{t+\Delta t} = ax_t + r\epsilon,$$

where $a := 1 + f(t)\Delta t$, $r := g(t)\sqrt{\Delta t}$, and $\epsilon \sim \mathcal{N}(0, 1)$. Equivalently, the forward one-step transition kernel is Gaussian:

$$x_{t+\Delta t}|x_t \sim \mathcal{N}(\cdot; ax_t, r^2).$$

Since p_{data} is assumed to be Gaussian, the current marginal at time t is also Gaussian, which takes the following form:

$$x_t \sim \mathcal{N}(\cdot; m_t, s_t^2),$$

for some scalar m_t and s_t. So conditioning will amount to multiplying two Gaussians and renormalizing. This keeps the algebra elementary.

By Bayes' rule the conditional density is, up to a constant, the product of the prior and the transition kernel:

$$\begin{aligned} p(x_t|x_{t+\Delta t}) &\propto p(x_{t+\Delta t}|x_t)p_t(\mathbf{x}_t) \\ &\propto \exp\Big(-\frac{(x_t - m_t)^2}{2s_t^2}\Big)\exp\Big(-\frac{(x_{t+\Delta t} - ax_t)^2}{2r^2}\Big). \end{aligned}$$

The exponent is a quadratic in x_t. Expanding both squares and grouping terms shows exactly which coefficients matter:

$$-2\log p(x_t|x_{t+\Delta t}) = Ax_t^2 - 2Bx_t + \text{const},$$

with

$$A := \frac{1}{s_t^2} + \frac{a^2}{r^2}, \quad B := \frac{m_t}{s_t^2} + \frac{ax_{t+\Delta t}}{r^2}.$$

Here A is the sum of precisions (prior precision plus the transition-kernel precision transported through a), while B is the corresponding precision-weighted sum of targets. With these in hand, completing the square gives the posterior in one line:

$$Ax_t^2 - 2Bx_t = A\Big(x_t - \frac{B}{A}\Big)^2 - \frac{B^2}{A},$$

so the conditional distribution is Gaussian with variance $1/A$ and mean B/A:

$$\text{Var}(x_t|x_{t+\Delta t}) = \frac{1}{\frac{1}{s_t^2} + \frac{a^2}{r^2}}, \qquad \mathbb{E}[x_t|x_{t+\Delta t}] = \frac{\frac{m_t}{s_t^2} + \frac{ax_{t+\Delta t}}{r^2}}{\frac{1}{s_t^2} + \frac{a^2}{r^2}}.$$

These closed forms already describe the reverse transition for any small Δt. To read off a reverse-time SDE, we now expand them for small Δt.

Use $a = 1 + f(t)\Delta t$ and $r^2 = g^2(t)\Delta t$. As $\Delta t \to 0$, the contribution $\frac{a^2}{r^2} \sim \frac{1}{g^2(t)\Delta t}$ dominates the precision, so the variance becomes

$$\text{Var}(x_t|x_{t+\Delta t}) = \left(\frac{1}{s_t^2} + \frac{a^2}{r^2}\right)^{-1} = g^2(t)\Delta t + \mathcal{O}(\Delta t^2),$$

which tells us the reverse step has the same diffusion scale $g(t)$ as the forward step. For the mean, expand the ratio B/A to first order:

$$\mathbb{E}[x_t|x_{t+\Delta t}] = x_{t+\Delta t} + \Delta t \left[-\left(f(t) + \frac{g^2(t)}{s_t^2} \right) x_{t+\Delta t} + \frac{g^2(t)}{s_t^2} m_t \right] + \mathcal{O}(\Delta t^2).$$

Putting the mean and variance together yields the one-step reverse transition kernel

$$x_t|x_{t+\Delta t} \sim \mathcal{N}\left(x_{t+\Delta t} + \Delta t \left[-\left(f + \frac{g^2}{s_t^2} \right) x_{t+\Delta t} + \frac{g^2}{s_t^2} m_t \right], g^2 \Delta t \right) + \mathcal{O}(\Delta t^2).$$

This is recognized as the Euler–Maruyama update, run backward from $t + \Delta t$ to t:

$$x_t - x_{t+\Delta t} = \Delta t \left[-\left(f + \frac{g^2}{s_t^2} \right) x_{t+\Delta t} + \frac{g^2}{s_t^2} m_t \right] + g\sqrt{\Delta t}\, \epsilon.$$

This backward drift can be written with the score because for a Gaussian marginal $p_t = \mathcal{N}(m_t, s_t^2)$,

$$\partial_x \log p_t(x) = -\frac{x - m_t}{s_t^2} \Longrightarrow -\left(f + \frac{g^2}{s_t^2} \right) x + \frac{g^2}{s_t^2} m_t = -\left[fx - g^2 \partial_x \log p_t(x) \right].$$

Therefore, letting $\Delta t \to 0$ gives the reverse-time SDE on the original diffusion clock,

$$\mathrm{d}\bar{x}(t) = \left[f(t)\bar{x}(t) - g^2(t) \partial_x \log p_t(\bar{x}(t)) \right] \mathrm{d}t + g(t)\, \mathrm{d}\bar{w}(t), \qquad t : T \to 0.$$

This exactly matches the general reverse-time SDE in Equation (4.1.6). In vector form, the scalar derivative $\partial_x \log p_t$ is replaced by $\nabla_{\mathbf{x}} \log p_t$.

Exact Computation of PF–ODE with a Gaussian. When the data distribution is assumed to be Gaussian, we can also directly derive the PF-ODE formula, avoiding heavy machinery such as the Fokker–Planck equation. In the end, we will see that the marginal densities of the PF-ODE coincide with those of both the forward SDE and the reverse-time SDE, providing a constructive verification of the Fokker–Planck theory to be discussed in Section 4.2.

Assume $x_t \sim \mathcal{N}(\cdot; m_t, s_t^2)$ at time t. A small deterministic step of size Δt can be written as a smooth map

$$x_{t+\Delta t} = \Phi_{t,\Delta t}(x_t) = x_t + \Delta t v_t(x_t) + \mathcal{O}(\Delta t^2),$$

which is simply the first–order Taylor expansion in Δt. Our goal is to construct a velocity field whose deterministic flow reproduces the same Gaussian marginal evolution. A particularly simple choice is an affine velocity field

$$v_t(x) = a_t x + b_t.$$

Since affine dynamics preserve Gaussianity, we can determine a_t and b_t by matching the evolution of the mean and variance to those of the forward SDE. Plugging this back into the step gives

$$x_{t+\Delta t} = (1 + \alpha_t \Delta t)x_t + \beta_t \Delta t + \mathcal{O}(\Delta t^2), \qquad \alpha_t := a_t, \quad \beta_t := b_t.$$

We now push $x_t \sim \mathcal{N}(\cdot; m_t, s_t^2)$ through this map and track mean and variance to first order:

$$\begin{aligned} \mathbb{E}[x_{t+\Delta t}] &= m_t + \Delta t(\alpha_t m_t + \beta_t) + \mathcal{O}(\Delta t^2), \\ \mathrm{Var}(x_{t+\Delta t}) &= s_t^2 + \Delta t(2\alpha_t s_t^2) + \mathcal{O}(\Delta t^2). \end{aligned}$$

On the other hand, the forward SDE $\mathrm{d}x = f(t)x\,\mathrm{d}t + g(t)\,\mathrm{d}w_t$ has the elementary moment formulas (see Equation (4.4.3)):

$$m_t' = f(t)m_t, \qquad (s_t^2)' = 2f(t)s_t^2 + g^2(t).$$

Matching the coefficients of Δt gives

$$\alpha_t = f(t) + \frac{g^2(t)}{2s_t^2}, \qquad \beta_t = -\frac{g^2(t)}{2s_t^2} m_t.$$

With these choices, the step becomes

$$x_{t+\Delta t} = x_t + \Delta t \left[\Big(f(t) + \frac{g^2(t)}{2s_t^2} \Big) x_t - \frac{g^2(t)}{2s_t^2} m_t \right] + \mathcal{O}(\Delta t^2).$$

Since for a Gaussian $p_t = \mathcal{N}(m_t, s_t^2)$ we have $\partial_x \log p_t(x) = -(x - m_t)/s_t^2$, we can rewrite the bracket as $f(t)x_t - \frac{1}{2}g^2(t)\partial_x \log p_t(x_t)$. Therefore,

$$x_{t+\Delta t} = x_t + \Delta t \Big[f(t)x_t - \tfrac{1}{2}g^2(t)\partial_x \log p_t(x_t) \Big] + \mathcal{O}(\Delta t^2).$$

Finally, dividing by Δt and letting $\Delta t \to 0$ yields the PF-ODE

$$x'(t) = f(t)x(t) - \tfrac{1}{2}g^2(t)\partial_x \log p_t(x(t)).$$

To see why this ODE has the same marginals as the forward SDE (and the reverse–time SDE), observe that the drift above is linear plus a shift. Thus $x(t)$ depends affinely on $x(0)$, and affine maps send Gaussians to Gaussians. Moreover, the mean m_t and variance s_t^2 along this ODE satisfy exactly the same two scalar ODEs as the forward SDE (by our matching), with the same initial values. Hence $p_t = \mathcal{N}(m_t, s_t^2)$ is identical for both evolutions at every time t.

4.3 Score SDE: Its Training and Sampling

4.3.1 Training

Building on the philosophy as in Chapter 3, we approximate the oracle score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ using a time-conditioned neural network

$$\mathbf{s}_{\boldsymbol{\phi}} = \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}, t)$$

across all $t \in [0, T]$, by minimizing a score-matching objective as in Equation (3.2.1):

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}; \omega(\cdot)) := \frac{1}{2}\mathbb{E}_{t\sim p_{\mathrm{time}}}\mathbb{E}_{\mathbf{x}_t\sim p_t}\left[\omega(t)\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\right\|_2^2\right],$$

where p_{time} is some time distribution (e.g., uniform on $[0, T]$), $\omega(\cdot)$ is a time-weighting function.

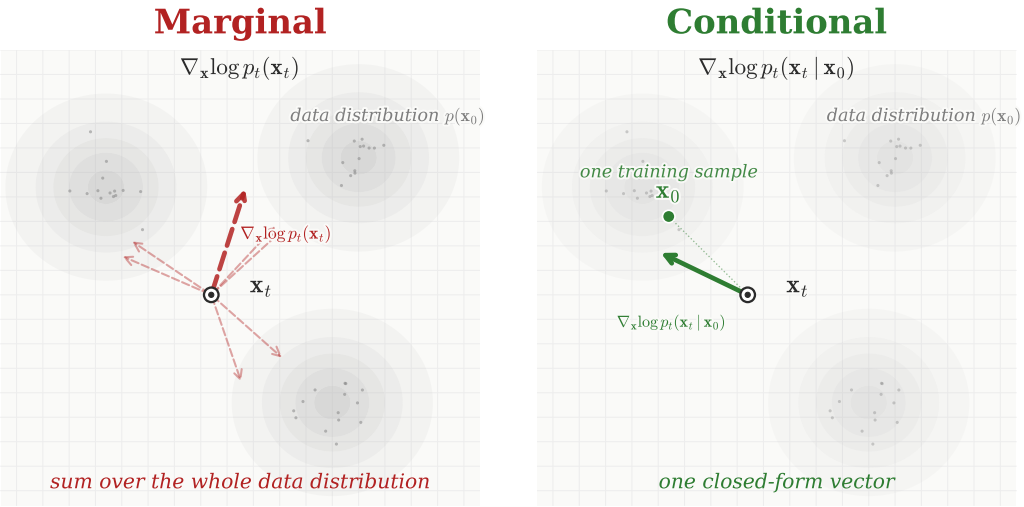


Figure 4.6: The denoising score matching trick. (Left) For a fixed noisy sample $\mathbf{x}_t$, the marginal score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ points toward regions where noisy samples are more likely. Computing it directly requires combining the contributions from all possible clean samples $\mathbf{x}_0$ that could have produced $\mathbf{x}_t$, which is generally intractable. **(Right)** During training, however, the clean sample $\mathbf{x}_0$ used to generate $\mathbf{x}_t$ is known. For Gaussian corruption, the corresponding conditional score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)$ is available in closed form, providing one simple vector as the regression target. Across training examples, these conditional score vectors average to the marginal score. Thus, denoising score matching learns the desired marginal score without directly regressing on the generally intractable score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, whose evaluation requires averaging over all possible samples from the data distribution.

Source: Created by the authors with AI-assisted coding.

To avoid relying on the intractable oracle score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, the DSM loss in Equation (3.3.2) is employed. Conditioned on a data point $\mathbf{x}_0$, this approach allows the use of the analytically tractable score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)$ via Equation (D.2.4), with concrete examples given in Section 4.4. Specifically, we exploit the following loss function:

$$\mathcal{L}_{\mathrm{DSM}}(\boldsymbol{\phi};\omega(\cdot)) := \frac{1}{2}\mathbb{E}_t\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\omega(t)\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t,t)-\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2\right], \tag{4.3.1}$$

where $\mathbf{x}_0 \sim p_{\mathrm{data}}$. Equation (4.3.1) can be interpreted as the continuous-time counterpart of Equation (3.4.1), with the summation in the discrete case replaced by integration.

Similar to the result in Theorem 3.3.1, the minimizer of Equation (4.3.1) is uniquely determined as follows:

Theorem 3.3.1 already implies this minimizer. The proposition below instead minimizes $\mathcal{L}_{\mathrm{DSM}}$ directly, without comparing losses, because its intermediate step is the one reused for flow-based models in Section 5.2.1.

Proposition 4.3.1: Minimizer of DSM

The minimizer $\mathbf{s}^*$ satisfies

$$\mathbf{s}^*(\mathbf{x}_t,t) = \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0|\mathbf{x}_t)}\left[\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{x}_0)\right] = \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t), \tag{4.3.2}$$

for almost every $\mathbf{x}_t \sim p_t$ and $t \in [0,T]$.

Proof for Proposition.

DSM objective can be understood as a least-squares error problem. Specifically, at each time t, the optimal score function is given by the conditional expectation of the gradient of the log conditional density, which, under Bayes' rule, is equivalent to the gradient of the log marginal density. For a detailed proof, see Appendix D.2.6. ∎

4.3.2 Sampling and Inference

After learning

$$\mathbf{s}_{\boldsymbol{\phi}^\times} := \mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x},t) \approx \nabla_{\mathbf{x}}\log p_t(\mathbf{x}),$$

we replace the intractable oracle score $\nabla_{\mathbf{x}}\log p_t(\mathbf{x})$ in the reverse-time SDE (Equation (4.1.6)) and PF-ODE (Equation (4.1.7)) with the learned proxy $\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x},t)$. This substitution enables tractable inference via either the SDE or the ODE. For clarity, we distinguish the resulting processes as $\mathbf{x}^{\mathrm{SDE}}_{\boldsymbol{\phi}^\times}(t)$ and $\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(t)$, respectively, but will omit this distinction in later sections.[10]

[10] This is to simplify notation after this subsection.

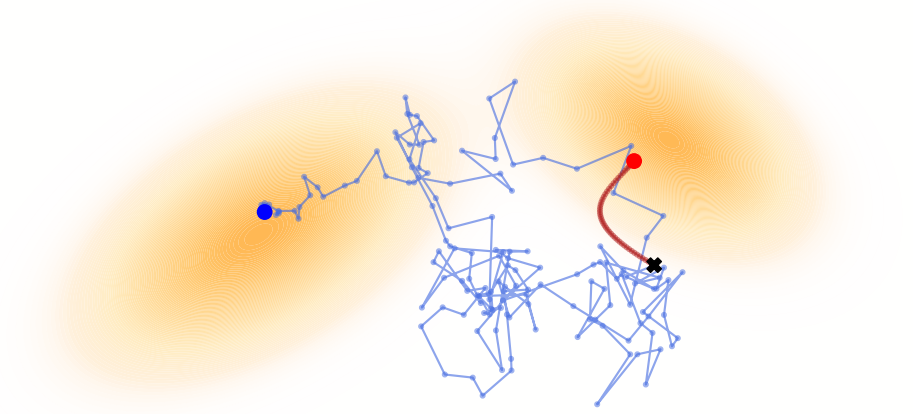


Figure 4.7: (2D) Illustration of sampling from the Score SDE. The forward SDE is designed as $\mathrm{d}\mathbf{x}(t) = \sqrt{2t}\,\mathrm{d}\mathbf{w}(t)$, with drift $\mathbf{f} \equiv \mathbf{0}$ and diffusion coefficient $g(t) = \sqrt{2t}$ on $[0, T]$, evolves a two-mode Gaussian mixture $p_0 = p_{\text{data}}$ into $p_T \approx p_{\text{prior}} := \mathcal{N}(\mathbf{0}, T^2\mathbf{I})$. Shown are sampling trajectories of the reverse-time SDE (blue; solved by Equation (4.3.4)) and the PF-ODE (red; solved by Equation (4.3.6)). Starting from a random point $\mathbf{x}_T \sim p_{\text{prior}}$ (dark "×"), both trajectories move toward the support of p_{data} as $t \to 0$.

Source: Created by the authors.

Empirical Reverse-Time SDE. By substituting the trained score model $\mathbf{s}_{\phi^\times}$ for the true score in Equation (4.1.6), we obtain the parameterized reverse-time SDE used for generation:

$$\mathrm{d}\mathbf{x}_{\phi^\times}^{\text{SDE}}(t) = \left[\mathbf{f}\left(\mathbf{x}_{\phi^\times}^{\text{SDE}}(t), t\right) - g^2(t)\mathbf{s}_{\phi^\times}\left(\mathbf{x}_{\phi^\times}^{\text{SDE}}(t), t\right)\right]\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}(t). \tag{4.3.3}$$

To generate a sample, we first draw an initial value $\mathbf{x}_T$ from the prior distribution p_{prior} and then numerically solve Equation (4.3.3) backward in time from $t = T$ to $t = 0$. A standard numerical solver for this is the Euler–Maruyama method, which provides the discrete update rule:

$$\mathbf{x}_{t-\Delta t} \leftarrow \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)\right]\Delta t + g(t)\sqrt{\Delta t}\cdot\boldsymbol{\epsilon}, \tag{4.3.4}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\Delta t > 0$ is the step size. Instead of integrating $g(t)\,\mathrm{d}\bar{\mathbf{w}}(t)$ backwards, one can let $s = T - t$; then $\mathrm{d}\bar{\mathbf{w}}(t) = \mathrm{d}\bar{\mathbf{w}}(T - s) = \mathrm{d}\mathbf{w}(s)$, as in Section D.2.5.

Iterating this update rule yields a final sample $\mathbf{x}_{\phi^\times}^{\text{SDE}}(0)$. If the score model is accurate, the distribution of these generated samples, denoted $p_{\phi^\times}^{\text{SDE}}(\cdot; 0)$, provides

a close approximation to the true data distribution[11]:

$$p^{\mathrm{SDE}}_{\boldsymbol{\phi}^\times}(\cdot;0) \approx p_{\mathrm{data}}(\cdot).$$

Indeed, for the VP construction, the DDPM sampling scheme in Equation (2.2.14) has the same continuous-time reverse-SDE limit as the Euler–Maruyama discretization above; see Section 4.4.

Empirical PF-ODE. The PF-ODE defines a continuous flow connecting p_{prior} and p_{data}, enabling sampling, encoding, and exact likelihood evaluation. The following section provides further details on each of these operations.

I. Sampling with PF-ODE. Replacing the oracle score in Equation (4.1.7) with $\mathbf{s}_{\boldsymbol{\phi}^\times}$ yields the empirical PF-ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(t) = \mathbf{f}\left(\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(t), t\right) - \frac{1}{2}g^2(t)\mathbf{s}_{\boldsymbol{\phi}^\times}\left(\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(t), t\right). \tag{4.3.5}$$

To generate samples, we begin by drawing an initial sample $\mathbf{x}_T$ from the prior distribution, p_{prior}. We then numerically solve the PF-ODE from Equation (4.3.5) backward in time from $t = T$ to $t = 0$. This process is equivalent to approximating the integral:

$$\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(0) = \mathbf{x}_T + \int_T^0 \left[\mathbf{f}\left(\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\tau), \tau\right) - \frac{1}{2}g^2(\tau)\mathbf{s}_{\boldsymbol{\phi}^\times}\left(\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\tau), \tau\right)\right] \mathrm{d}\tau.$$

Solving this integral yields a final sample, $\mathbf{x}^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(0)$. The distribution of samples generated via this deterministic process, denoted $p^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\cdot;0)$, provides an approximation to the data distribution, such that $p^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\cdot;0) \approx p_{\mathrm{data}}$.

Let $\Delta t > 0$ denote a discretization step size. A standard numerical integration approach is the Euler method, which estimates

$$\mathbf{f}(\mathbf{x}_\tau, \tau) - \frac{1}{2}g^2(\tau)\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t), \quad \tau \in [t - \Delta t, t],$$

leading to the following update rule:

$$\mathbf{x}_{t-\Delta t} \leftarrow \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t)\right]\Delta t. \tag{4.3.6}$$

This connection allows us to reframe the process of generation with the following core insight:

[11] Theoretically, estimation accuracy depends on the discrepancy between p_T and p_{prior} (typically negligible), model training error, and numerical discretization error (De Bortoli, 2022; Wang *et al.*, 2024). We do not pursue formal bounds here.

Insight 4.3.1: Generation ⇔ ODE/SDE Solving

Sampling from diffusion models is fundamentally equivalent to solving a corresponding probability flow ODE or a reverse-time SDE.

This equivalence provides a clear explanation for the slow sampling speeds of diffusion models, as raised in Question 3.5.1. Generation is computationally intensive because numerical solvers for these differential equations are inherently iterative, often requiring many steps to accurately approximate a solution trajectory[12]. However, the PF-ODE formulation is also advantageous, as it allows us to leverage the extensive literature on accelerated numerical solvers. Exploring these techniques to speed up diffusion model sampling is the primary focus of Chapter 9.

II. Inversion with PF-ODE. As discussed, unlike in the case of SDEs, we can solve the same Equation (4.3.5) both forward (from 0 to T) and reverse (from T to 0) in time. When solving it forward, the ODE flow maps data to its (noisy) latent representations across all $t \in [0, T]$, which plays a role of an encoder. This concept enables powerful applications for controllable generation, such as image translation and editing and beyond (Mokady *et al.*, 2023; Su *et al.*, 2022).

III. Log-Likelihood Evaluation via PF-ODE. The PF-ODE also allows us to evaluate the likelihood of a data sample. Recall from Equation (5.1.9) that, for an ODE, the change in log-density along a trajectory is determined by the divergence of its velocity field.

For the learned PF-ODE, define

$$\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t) := \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t).$$

If $\mathbf{x}(t)$ evolves according to

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t),$$

then its log-density satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t); t) = -\nabla_{\mathbf{x}} \cdot \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t).$$

To evaluate the likelihood of an observed sample $\mathbf{x}_0$, we run the PF-ODE forward from $t = 0$ to $t = T$, mapping $\mathbf{x}_0$ to a terminal point where the model density is the tractable prior. We can simultaneously track the log-density change by defining

$$\delta(t) := \log p^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t); t) - \log p^{\mathrm{ODE}}_{\boldsymbol{\phi}^\times}(\mathbf{x}_0; 0).$$

[12] For example, DDPM and Score SDE typically use $1,000$ function evaluations for generation.

Thus, $\delta(0) = 0$, and the state and log-density change can be computed together through

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{x}(t)\\ \delta(t)\end{bmatrix} = \begin{bmatrix}\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t),t)\\ -\nabla_{\mathbf{x}}\cdot\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t),t)\end{bmatrix}, \qquad \begin{bmatrix}\mathbf{x}(0)\\ \delta(0)\end{bmatrix} = \begin{bmatrix}\mathbf{x}_0\\ 0\end{bmatrix}. \tag{4.3.7}$$

After integrating to $t = T$, the terminal density is $p_{\text{prior}}(\mathbf{x}(T))$. Therefore,

$$\log p^{\text{ODE}}_{\boldsymbol{\phi}^\times}(\mathbf{x}_0; 0) = \log p_{\text{prior}}(\mathbf{x}(T)) - \delta(T).$$

Equivalently,

$$\log p^{\text{ODE}}_{\boldsymbol{\phi}^\times}(\mathbf{x}_0; 0) = \log p_{\text{prior}}(\mathbf{x}(T)) + \int_0^T \nabla_{\mathbf{x}}\cdot\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t),t)\ \mathrm{d}t.$$

Thus, the same PF-ODE used for generation and inversion also provides a continuous change-of-variables formula for likelihood evaluation.

4.4 Instantiations of SDEs

Song *et al.* (2020c) categorize the drift term $\mathbf{f}(\mathbf{x}, t)$ and the diffusion term $g(t)$ in the forward SDE into three types based on the behavior of the variance during evolution. Here, we focus on two commonly used types: the *Variance Explosion* (VE) SDE and *Variance Preserving* (VP) SDE. While it is possible to design custom noise schedulers, their design can substantially influence empirical performance. Table 4.1 summarizes these two SDE instantiations.

Table 4.1: Summary of the forward SDEs

	VE SDE	**VP SDE**
$\mathbf{f}(\mathbf{x}, t)$	$\mathbf{0}$	$-\frac{1}{2}\beta(t)\mathbf{x}$
$g(t)$	$\sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}$	$\sqrt{\beta(t)}$
SDE	$\mathrm{d}\mathbf{x}(t) = g(t)\,\mathrm{d}\mathbf{w}(t)$	$\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}(t)$
$p_t(\mathbf{x}_t \vert \mathbf{x}_0)$	$\mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, \left(\sigma^2(t) - \sigma^2(0)\right)\mathbf{I}\right)$	$\mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0 e^{-\frac{1}{2}\int_0^t \beta(\tau)\,\mathrm{d}\tau}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(\tau)\,\mathrm{d}\tau}\right)$
p_{prior}	$\mathcal{N}(\mathbf{0}, \sigma^2(T)\mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$

4.4.1 VE SDE

VE SDE has the following components:

- **Drift Term:** A zero drift term $\mathbf{f} = \mathbf{0}$.
- **Diffusion Term:** $g(t) = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}$ for some function $\sigma(t)$.

The forward SDE then takes the following form:

$$\mathrm{d}\mathbf{x}(t) = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w}(t). \tag{4.4.1}$$

Similarly, the results from Section 4.4.3 imply the perturbation kernel for the VE SDE and suggest selecting an appropriate prior distribution:

- **Perturbation Kernel:**

$$p_t(\mathbf{x}_t \vert \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, \left(\sigma^2(t) - \sigma^2(0)\right)\mathbf{I}\right)$$

- **Prior Distribution:** Assume that $\sigma(t)$ is an increasing function for $t \in [0, T]$ and that $\sigma^2(T) \gg \sigma^2(0)$. The prior distribution is given by:

$$p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \sigma^2(T)\mathbf{I}).$$

A typical instance of a VE SDE is NCSN with the following design:

$$\sigma(t) := \sigma_{\min} \left(\frac{\sigma_{\max}}{\sigma_{\min}}\right)^t, \quad \text{for } t \in (0, 1],$$

where $\sigma_{\min}$ and $\sigma_{\max}$ are pre-specified constants. Namely, the sequence of variances is designed as a geometric sequence. With this, NCSN is viewed as a discretized version of VE SDE, as discussed in Section 4.1.1.

4.4.2 VP SDE

Let $\beta\colon [0, T] \to \mathbb{R}_{\geq 0}$ be a non-negative function of t. A VP SDE is defined with the following components:

- **Drift Term:** A linear drift given by $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$.
- **Diffusion Term:** $g(t) = \sqrt{\beta(t)}$.

Thus, the forward SDE is expressed as:

$$\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}(t). \tag{4.4.2}$$

Using the results from Section 4.4.3, we can derive the perturbation kernel for the VP SDE and select an appropriate prior distribution:

- **Perturbation Kernel:**

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0 e^{-\frac{1}{2}\int_0^t \beta(\tau)\,\mathrm{d}\tau}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(\tau)\,\mathrm{d}\tau}\right).$$

- **Prior Distribution:** $p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$.

We remark that since the perturbation kernel is a Gaussian with a known mean and covariance, we can apply Equation (D.2.5) to compute its score function.

A classic example of a VP SDE is the DDPM, where the noise schedule $\beta(t)$ is defined as:

$$\beta(t) := \beta_{\min} + t(\beta_{\max} - \beta_{\min}), \quad \text{for all } t \in [0, 1].$$

Here, $\beta_{\min}$ and $\beta_{\max}$ are pre-defined constants. With this, DDPM can be interpreted as a discretization of the VP SDE, as discussed in Section 4.1.1.

4.4.3 (Optional) How Is the Perturbation Kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ Derived?

If the drift term in the forward SDE Equation (4.1.3) is linear in $\mathbf{x}$, taking the form

$$\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x},$$

for some scalar-valued, time-dependent function $f(t) \in \mathbb{R}$, then Equation (4.1.3) becomes a linear SDE:

$$\mathrm{d}\mathbf{x}(t) = f(t)\mathbf{x}(t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t).$$

Even if the initial distribution p_{data} is non-Gaussian, the linearity of the drift ensures that the conditional process remains Gaussian. In particular, for $t > 0$, the transition kernel admits the form:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{m}(t), P(t)\mathbf{I}_D\right),$$

where $\mathbf{x}_0 \sim p_{\text{data}}$, and $\mathbf{m}(t) \in \mathbb{R}^D$, $P(t) \in \mathbb{R}_{\geq 0}$ denote the conditional mean and (scalar) variance given $\mathbf{x}_0$, defined as:

$$\mathbf{m}(t) = \mathbb{E}\left[\mathbf{x}_t|\mathbf{x}(0) = \mathbf{x}_0\right], \qquad P(t)\mathbf{I}_D = \mathrm{Cov}\left[\mathbf{x}_t|\mathbf{x}(0) = \mathbf{x}_0\right].$$

These first and second moments evolve according to the following ODEs (Särkkä and Solin, 2019):

$$\begin{aligned} \frac{\mathrm{d}\mathbf{m}(t)}{\mathrm{d}t} &= f(t)\mathbf{m}(t), \\ \frac{\mathrm{d}P(t)}{\mathrm{d}t} &= 2f(t)P(t) + g^2(t), \end{aligned} \tag{4.4.3}$$

provided that the initial mean $\mathbf{m}(0)$ and variance $P(0)$ are finite.

Since both ODEs are linear, they admit closed-form solutions via the integrating factor method. Given the initial condition $\mathbf{x}_0$, the mean and variance evolve as

$$\mathbf{m}(t) = \mathcal{E}(0 \to t)\mathbf{x}_0, \qquad P(t) = \int_0^t \mathcal{E}^2(s \to t)g(s)^2\,\mathrm{d}s, \tag{4.4.4}$$

with $\mathbf{m}(0) = \mathbf{x}_0$ and $P(0) = 0$. Here $\mathcal{E}(s \to t)$ denotes the exponential integrating factor

$$\mathcal{E}(s \to t) := \exp\left(\int_s^t f(u)\,\mathrm{d}u\right),$$

which captures the accumulated effect of the drift from time s to t. Consequently, the transition kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ also admits a closed-form expression.

We defer the justification that the conditional covariance of $p_t(\mathbf{x}_t|\mathbf{x}_0)$ is isotropic, that is $\mathrm{Cov}[\mathbf{x}_t|\mathbf{x}_0] = P(t)\mathbf{I}_D$ under a D-dimensional Wiener process with independent coordinates and diffusion $g(t)\mathbf{I}_D$, as well as the derivation of Equation (4.4.3), to Section C.1.5, which relies on Itô calculus.

Example: VE SDE's Transition Kernel

> In the special case of VE SDE: $\mathbf{f} \equiv 0$ and $g(t) = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}$, the mean and covariance of the solution to the SDE evolve as follows.

Mean.

$$\frac{\mathrm{d}\mathbf{m}(t)}{\mathrm{d}t} = 0, \quad \text{with} \quad \mathbf{m}(0) = \mathbf{x}_0 \Longrightarrow \mathbf{m}(t) = \mathbf{x}_0.$$

Variance.

$$\frac{\mathrm{d}P(t)}{\mathrm{d}t} = \frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}, \quad \text{with} \quad P(0) = 0 \Longrightarrow P(t) = \sigma^2(t) - \sigma^2(0).$$

Therefore

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, (\sigma^2(t) - \sigma^2(0))\mathbf{I}_D\right).$$

Example: VP SDE's Transition Kernel

In the VP SDE case with drift $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ and diffusion $g(t) = \sqrt{\beta(t)}$:

Mean $\mathbf{m}(t)$.

$$\frac{\mathrm{d}\mathbf{m}}{\mathrm{d}t} = -\frac{1}{2}\beta(t)\mathbf{m}(t), \quad B(t) \coloneqq \int_0^t \beta(s)\,\mathrm{d}s, \quad \mathbf{m}(t) = e^{-\frac{1}{2}B(t)}\mathbf{x}_0.$$

Variance $P(t)$. The variance satisfies

$$\frac{\mathrm{d}P}{\mathrm{d}t} = -\beta(t)P(t) + \beta(t).$$

Applying the integrating factor $e^{B(t)}$ with $B(t) = \int_0^t \beta(s)\,\mathrm{d}s$, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[P(t)e^{B(t)}\right] = \beta(t)e^{B(t)}.$$

Integrating both sides gives

$$P(t)e^{B(t)} - P(0)e^{B(0)} = \int_0^t \beta(s)\,e^{B(s)}\,\mathrm{d}s = \int_0^t \frac{\mathrm{d}}{\mathrm{d}s}e^{B(s)}\,\mathrm{d}s = e^{B(t)} - 1,$$

since $\frac{\mathrm{d}}{\mathrm{d}t}e^{B(t)} = e^{B(t)}\frac{\mathrm{d}}{\mathrm{d}t}B(t) = \beta(t)e^{B(t)}$; with $P(0) = 0$ and $B(0) = 0$,

$$P(t) = 1 - e^{-B(t)}.$$

Hence the covariance is isotropic with

$$\mathbf{P}(t) = P(t)\mathbf{I}_D = (1 - e^{-B(t)})\mathbf{I}_D.$$

Final Closed-Form Transition Kernel.

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \underbrace{e^{-\frac{1}{2}B(t)}\mathbf{x}_0}_{\mathbf{m}(t)}, \underbrace{(1 - e^{-B(t)})\mathbf{I}_D}_{P(t)\mathbf{I}_D}\right), \quad B(t) = \int_0^t \beta(s)\,\mathrm{d}s.$$

Also, note that the squared mean coefficient plus the variance is $e^{-B(t)} + (1 - e^{-B(t)}) = 1$, just like DDPM.

4.5 A Direct View Through Accumulated Perturbation Kernels

DDPM and Score SDE are commonly introduced by specifying how noise is added locally in time: through discrete forward transitions $p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t})$ in DDPM, or through a continuous-time SDE in Score SDE. For training, however, a particularly useful object is the *accumulated perturbation kernel*

$$p_t(\mathbf{x}_t|\mathbf{x}_0),$$

which describes the distribution at time t directly conditioned on the original data sample. This is precisely the form that appears naturally in the DDPM and continuous-time denoising objectives Equations (2.2.8) and (4.3.1).

This suggests a complementary viewpoint: rather than beginning with the local noise injection and then deriving $p_t(\mathbf{x}_t|\mathbf{x}_0)$, we can specify the accumulated perturbation kernel directly. For the linear-Gaussian processes considered below, an admissible accumulated kernel determines the corresponding linear SDE, and vice versa. This viewpoint makes the amount of signal and noise at each time explicit, connects directly to practical training objectives, and will also provide a useful bridge to the probability-path viewpoint of Flow Matching in Chapter 5.

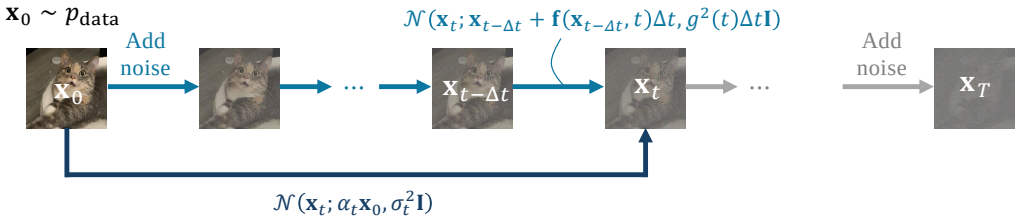


Figure 4.8: Two views of the same affine forward process. For an admissible affine diffusion schedule, incremental noise injection through a continuous-time linear SDE and direct specification of the accumulated perturbation kernel in Equation (4.5.1) describe the same conditional distributions $p_t(\mathbf{x}_t|\mathbf{x}_0)$.

Source: Created by the authors.

A General Affine Forward Process $p_t(\mathbf{x}_t|\mathbf{x}_0)$. We begin by defining the accumulated forward perturbation kernel

$$p_t(\mathbf{x}_t|\mathbf{x}_0) := \mathcal{N}(\mathbf{x}_t; \alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I}), \tag{4.5.1}$$

where $\mathbf{x}_0 \sim p_{\text{data}}$, and α_t and σ_t are nonnegative scalar functions of $t \in [0, T]$. In the standard forward diffusion setup, we take

$$\alpha_0 = 1, \qquad \sigma_0 = 0,$$

so that the process starts exactly from the data, and assume $\alpha_t > 0$ and $\sigma_t > 0$ for $t \in (0, T]$ whenever the corresponding ratios are used.

Equivalently, a sample from Equation (4.5.1) can be written as

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

This simple form includes several familiar probability paths:

- **VE (NCSN):** $\alpha_t \equiv 1$, with the noise scale σ_t increasing over time;
- **VP (DDPM):** $\alpha_t^2 + \sigma_t^2 = 1$;
- **Linear Gaussian interpolation:** $\alpha_t = 1 - t$ and $\sigma_t = t$ on $t \in [0, 1]$, corresponding to the familiar linear path used in Flow Matching and Rectified Flow when written in the data-to-noise time convention.

4.5.1 Connection to Score SDE

For Score SDE, it is often more intuitive to specify the accumulated perturbation kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ first and then recover the corresponding SDE coefficients, rather than starting from the drift and diffusion terms and solving ODEs for the moments (see Section 4.4.3).

For an admissible affine schedule, the corresponding forward SDE takes the linear-in-$\mathbf{x}$ form in Equation (4.4.2):

$$\mathrm{d}\mathbf{x}(t) = \underbrace{f(t)\mathbf{x}(t)}_{\mathbf{f}(\mathbf{x}(t),t)}\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t),$$

where $f, g\colon [0, T] \to \mathbb{R}$ are real-valued functions of time.

The coefficients $f(t)$ and $g(t)$ can be recovered analytically from α_t and σ_t. Importantly, not every choice of α_t and σ_t defines a valid diffusion: the implied diffusion variance $g^2(t)$ must remain nonnegative. Equivalently, in this linear-Gaussian setting, the signal-to-noise ratio α_t^2/σ_t^2 cannot increase with time. The following lemma makes this relationship precise.

Lemma 4.5.1: Forward Perturbation Kernel ⇔ Linear SDE

Assume $\alpha_0 = 1$, $\sigma_0 = 0$, and that $\alpha_t > 0$ and $\sigma_t > 0$ are differentiable for $t \in (0, T]$. Define $\lambda_t := \log \frac{\alpha_t}{\sigma_t}$.
Given the forward perturbation kernel

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

there exists a corresponding linear SDE

$$\mathrm{d}\mathbf{x}(t) = f(t)\mathbf{x}(t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t)$$

with the same conditional kernel $p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ if and only if the coefficients

$$\begin{aligned} f(t) &= \frac{\mathrm{d}}{\mathrm{d}t} \log \alpha_t, \\ g^2(t) &= \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}}{\mathrm{d}t} \log \alpha_t\, \sigma_t^2 = -2\sigma_t^2 \frac{\mathrm{d}}{\mathrm{d}t} \lambda_t \end{aligned} \tag{4.5.2}$$

satisfy $g^2(t) \geq 0$ for $t \in (0, T]$. Equivalently, λ_t must be non-increasing.
Conversely, if a linear SDE has conditional kernels $\mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ with $\alpha_t > 0$ and $\sigma_t > 0$, then its coefficients necessarily satisfy Equation (4.5.2).

Proof for Lemma.

From Section 4.4.3, the result follows by matching the mean and covariance ODEs $\mathbf{m}'(t) = f(t)\mathbf{m}(t)$ and $\mathbf{P}'(t) = 2f(t)\mathbf{P}(t) + g^2(t)\mathbf{I}$ with $\mathbf{m}(t) = \alpha_t \mathbf{x}_0$ and $\mathbf{P}(t) = \sigma_t^2 \mathbf{I}$.

Proof. For the mean, $\mathbf{m}'(t) = f(t)\mathbf{m}(t) = (\frac{\mathrm{d}}{\mathrm{d}t} \log \alpha_t)\mathbf{m}(t)$, so

$$\mathbf{m}(t) = \exp\Big(\int_0^t \tfrac{\mathrm{d}}{\mathrm{d}s} \log \alpha_s \, \mathrm{d}s\Big)\mathbf{x}_0 = \exp(\log \alpha_t)\, \mathbf{x}_0 = \alpha_t \mathbf{x}_0,$$

using $\alpha_0 = 1$. For the covariance, $\mathbf{P}'(t) = 2f(t)\mathbf{P}(t) + g^2(t)\mathbf{I} = 2(\frac{\mathrm{d}}{\mathrm{d}t} \log \alpha_t)\mathbf{P}(t) - 2\sigma_t^2(\frac{\mathrm{d}}{\mathrm{d}t}\lambda_t)\mathbf{I}$, and the integrating factor α_t^{-2} gives

$$\mathbf{P}(t) = -2\alpha_t^2 \int_0^t \alpha_s^{-2}\sigma_s^2 \frac{\mathrm{d}\lambda_s}{\mathrm{d}s}\, \mathrm{d}s\, \mathbf{I} = -2\alpha_t^2\Big(-\frac{\sigma_t^2}{2\alpha_t^2}\Big)\mathbf{I} = \sigma_t^2\, \mathbf{I},$$

since $\frac{\mathrm{d}}{\mathrm{d}s}(-\frac{\sigma_s^2}{2\alpha_s^2}) = \alpha_s^{-2}\sigma_s^2 \frac{\mathrm{d}\lambda_s}{\mathrm{d}s}$, which can be verified by expanding both sides, and $\sigma_0 = 0$. ∎

Remark.

To make the terminal conditional distribution independent of $\mathbf{x}_0$ and equal to a prescribed Gaussian prior, we need $\alpha_T = 0$ and σ_T^2 equal to the prior variance.

Since

$$\alpha_t = \exp\Big(\int_0^t f(u)\,\mathrm{d}u\Big),$$

achieving $\alpha_T = 0$ at a finite terminal time requires

$$\int_0^T f(u)\,\mathrm{d}u = -\infty.$$

Thus, the contraction induced by the drift must become singular near the terminal time; in particular, f cannot remain bounded on $[0, T]$.

The diffusion coefficient is related to the prescribed variance through

$$g^2(t) = (\sigma_t^2)' - 2\frac{\alpha_t'}{\alpha_t}\sigma_t^2.$$

For the usual regular schedules in which σ_t^2 approaches a finite nonzero terminal variance, $(\sigma_t^2)'$ remains bounded, and $\alpha_t'/\alpha_t \to -\infty$, this also gives

$$g^2(t) \to \infty \qquad \text{as } t \to T.$$

More generally, the precise behavior of $g(t)$ depends on the chosen schedule.

If f and g remain bounded on a finite interval $[0, T]$, then $\alpha_T > 0$, so exact forgetting cannot occur at finite T. In practice, one therefore chooses a terminal time for which the residual dependence on $\mathbf{x}_0$ is negligible and $p_T \approx p_{\text{prior}}$, or considers an infinite-time formulation in which $\alpha_t \to 0$ asymptotically.

From the above lemma, specifying the incremental noise injection via a linear SDE with coefficients $f(t)$ and $g(t)$ is mathematically equivalent to defining the perturbation kernel with parameters α_t and σ_t. In the diffusion model literature, these two viewpoints are used interchangeably. Therefore, we conclude:

Observation 4.5.1:

For a valid affine diffusion schedule, the two descriptions are equivalent: we may either specify the accumulated perturbation kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ through α_t and σ_t, or specify the corresponding linear SDE through $f(t)$ and $g(t)$.

4.5.2 Connection to Variational-Based Diffusion Model

We revisit a core identity from DDPM, derived via Bayes' rule:

$$p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}) = p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t}) \cdot \frac{p_{t-\Delta t}(\mathbf{x}_{t-\Delta t}|\mathbf{x})}{p_t(\mathbf{x}_t|\mathbf{x})}, \tag{4.5.3}$$

for any $\mathbf{x}$ (usually $\mathbf{x} \sim p_{\text{data}}$). This reverse conditional $p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x})$ is central to modeling, enabling both tractable training targets and efficient sampling.

Although DDPM typically defines the incremental kernel $p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t})$ first, the accumulated transition $p_t(\mathbf{x}_t|\mathbf{x}_0)$ often provides a more interpretable and practical formulation, especially for the prior and loss design.

Deriving Transition Kernels. We now extend this to the continuous-time setting. Let $0 \le t < s \le T$ be two (continuous) time points. Given the perturbation kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$, we can compute the reverse conditional $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x})$ for any $\mathbf{x}$ by applying Equation (4.5.3)[13], using the forward kernel $p(\mathbf{x}_s|\mathbf{x}_t)$ as an intermediate. The following lemma summarizes this derivation, extending Lemma 2.2.2 without assuming $\alpha_t^2 + \sigma_t^2 = 1$.

Lemma 4.5.2: Reverse Conditional Transition Kernels

Let $0 \le t < s \le T$. The reverse *conditional* transition kernel is:

$$p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}) = \mathcal{N}\left(\mathbf{x}_t; \boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}; s, t), \sigma^2(s,t)\mathbf{I}\right),$$

where

$$\boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}; s, t) := \frac{\alpha_{s|t}\sigma_t^2}{\sigma_s^2}\mathbf{x}_s + \frac{\alpha_t \sigma_{s|t}^2}{\sigma_s^2}\mathbf{x}, \quad \sigma^2(s,t) := \sigma_{s|t}^2 \frac{\sigma_t^2}{\sigma_s^2}. \tag{4.5.4}$$

Here, $\alpha_{s|t}$ and $\sigma_{s|t}$ are defined as:

$$\alpha_{s|t} := \frac{\alpha_s}{\alpha_t}, \qquad \sigma_{s|t}^2 := \sigma_s^2 - \alpha_{s|t}^2 \sigma_t^2.$$

Proof for Lemma.

We first compute the forward transition kernel:

$$p(\mathbf{x}_s|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_s; \alpha_{s|t}\mathbf{x}_t, \sigma_{s|t}^2\mathbf{I}\right). \tag{4.5.5}$$

The reverse kernel then follows from Bayes' rule, and since all involved distributions are Gaussian, the result can be derived by direct computation. For further details, see Appendix A of (Kingma *et al.*, 2021). ∎

Proof of Equation (4.5.5). Assume the forward process is Gaussian and Markovian, so $p(\mathbf{x}_s|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_s; \alpha\mathbf{x}_t, \sigma^2\mathbf{I})$ for some α, σ. Then, since $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}_1$,

$$\mathbf{x}_s = \alpha\mathbf{x}_t + \sigma\boldsymbol{\epsilon}_2 = \alpha\alpha_t\mathbf{x}_0 + \alpha\sigma_t\boldsymbol{\epsilon}_1 + \sigma\boldsymbol{\epsilon}_2 = \alpha\alpha_t\mathbf{x}_0 + \sqrt{\alpha^2\sigma_t^2 + \sigma^2}\,\boldsymbol{\epsilon}.$$

Matching with $p(\mathbf{x}_s|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_s; \alpha_s\mathbf{x}_0, \sigma_s^2\mathbf{I})$, i.e., $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$, gives

$$\alpha = \frac{\alpha_s}{\alpha_t} = \alpha_{s|t}, \qquad \sigma^2 = \sigma_s^2 - \alpha_{s|t}^2\sigma_t^2 = \sigma_{s|t}^2. \qquad \blacksquare$$

[13] This identity extends naturally to continuous time by treating s as a general earlier time.

Although $p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)$ and $p_t(\mathbf{x}_t|\mathbf{x}_0)$ are theoretically equivalent, $p_t(\mathbf{x}_t|\mathbf{x}_0)$ often takes a more central role. The step-wise transition in Equation (4.5.5) mainly serves to obtain a closed-form reverse kernel. Recent works (Kingma *et al.*, 2021) thus favor directly specifying $p_t(\mathbf{x}_t|\mathbf{x}_0)$ for its clarity and interpretability.

Reverse Process Modeling, Training, and Sampling. The training objective (ELBO in Equation (2.2.13)) and the modeling framework introduced in Section 2.2 remain applicable under our generalized setting. For clarity, we adopt the $\mathbf{x}$-prediction formulation, denoted by $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s)$, following Kingma *et al.* (2021). However, the equivalent $\boldsymbol{\epsilon}$-prediction perspective, represented by $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_s, s)$, is also valid due to the relationship (as in Equation (2.2.12))

$$\mathbf{x}_s = \alpha_s \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s) + \sigma_s \boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_s, s), \quad \text{for any given } \mathbf{x}_s \sim q_s.$$

Modeling and Diffusion Loss $\mathcal{L}_{\text{diffusion}}$. Similar to DDPM, the conditional distribution $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x})$ in Equation (4.5.4) motivates replacing the clean signal $\mathbf{x}$ with a learnable predictor $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s)$, yielding a parameterized reverse model of the form:

$$p_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_s) := \mathcal{N}\left(\mathbf{x}_t; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_s, s, t), \sigma^2(s,t)\mathbf{I}\right), \tag{4.5.6}$$

with the mean parametrized as:

$$\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_s, s, t) = \frac{\alpha_{s|t}\sigma_t^2}{\sigma_s^2}\mathbf{x}_s + \frac{\alpha_t \sigma_{s|t}^2}{\sigma_s^2}\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s).$$

Given the forward kernel in Equation (4.5.1), the KL divergence in $\mathcal{L}_{\text{diffusion}}(\mathbf{x}; \boldsymbol{\phi})$ reduces to a weighted regression loss:

$$\begin{aligned} \mathcal{D}_{\text{KL}}\big(p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) \| p_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_s)\big) &= \frac{1}{2\sigma^2(s,t)} \left\| \boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}_0; s, t) - \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}_s, s, t) \right\|_2^2 \\ &= \frac{1}{2}\big(\text{SNR}(t) - \text{SNR}(s)\big) \left\| \mathbf{x}_0 - \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s) \right\|_2^2, \end{aligned} \tag{4.5.7}$$

where $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$, with $\mathbf{x}_0 \sim p_{\text{data}}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\text{SNR}(s) := \alpha_s^2/\sigma_s^2$ denotes the signal-to-noise ratio at time s.

Remark.

> In (Kingma *et al.*, 2021), the authors study the continuous-time limit of Equation (4.5.7) as $t \to s$, yielding:
>
> $$\mathcal{L}_{\text{VDM}}^{\infty}(\mathbf{x}_0) = -\frac{1}{2}\mathbb{E}_{s, \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})} \text{SNR}'(s) \left\| \mathbf{x}_0 - \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_s, s) \right\|_2^2.$$

> This setup also introduces a learnable noise schedule, and while it generalizes beyond continuous data, such extensions fall outside the scope of our current discussion.

Sampling. Sampling proceeds similarly to DDPM using the parameterized kernel from Equation (4.5.6):

$$\mathbf{x}_t = \underbrace{\frac{\alpha_{s|t}\sigma_t^2}{\sigma_s^2}\mathbf{x}_s + \frac{\alpha_t \sigma_{s|t}^2}{\sigma_s^2}\mathbf{x}_{\phi^\times}(\mathbf{x}_s, s)}_{\boldsymbol{\mu}_{\phi^\times}(\mathbf{x}_s,t,s)} + \sigma_{s|t}\frac{\sigma_t}{\sigma_s}\boldsymbol{\epsilon}_s, \quad \boldsymbol{\epsilon}_s \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{4.5.8}$$

4.6 (Optional) Fokker–Planck Equation and Reverse-Time SDEs via Marginalization and Bayes' Rule

In this section, we offer a probabilistic perspective on the structure of the Fokker–Planck equation and the reverse-time SDE. By leveraging fundamental tools such as the marginalization trick and Bayes' rule, we illuminate the connection between the statistical formulation of stochastic processes and their corresponding differential equations.

We emphasize that the "derivation" presented here is not mathematically rigorous proofs, but rather heuristic arguments intended to convey the underlying connections.

4.6.1 Fokker-Planck Equation from the Marginalization of Transition Kernels

Given the forward transition probability as in Equation (4.1.2)

$$p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t, g^2(t)\Delta t\mathbf{I}\right),$$

and the marginal distributions

$$p_t(\mathbf{x}_t), \quad p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}),$$

we aim to derive the Fokker-Planck equation that governs the time evolution of the marginal distribution p_t.

Change of Variables. By the Markov property, the marginal distribution at time $t + \Delta t$ can be expressed as an integral over the previous state $\mathbf{x}_t$ (i.e., Chapman-Kolmogorov equation):

$$p_{t+\Delta t}(\mathbf{x}) = \int \mathcal{N}(\mathbf{x}; \mathbf{y} + \mathbf{f}(\mathbf{y}, t)\Delta t, g^2(t)\Delta t\mathbf{I}) p_t(\mathbf{y})\, \mathrm{d}\mathbf{y}.$$

We introduce a new variable

$$\mathbf{u} := \mathbf{y} + \mathbf{f}(\mathbf{y}, t)\Delta t,$$

so the Gaussian is centered at $\mathbf{u}$. For small Δt, this map is invertible with

$$\mathbf{y} = \mathbf{u} - \mathbf{f}(\mathbf{u}, t)\Delta t + \mathcal{O}(\Delta t^2), \quad \left|\det \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\right| = 1 - (\nabla_{\mathbf{u}} \cdot \mathbf{f})(\mathbf{u}, t)\Delta t + \mathcal{O}(\Delta t^2).$$

Hence, change-of-variable formula leads us to:

$$\begin{aligned} p_{t+\Delta t}(\mathbf{x}) = &\int \mathcal{N}\left(\mathbf{x}; \mathbf{u}, g^2(t)\Delta t\mathbf{I}\right) \cdot \\ &\Big[p_t(\mathbf{u}) - \Delta t \mathbf{f}(\mathbf{u}, t) \cdot \nabla_{\mathbf{u}} p_t(\mathbf{u}) - \Delta t (\nabla_{\mathbf{u}} \cdot \mathbf{f})(\mathbf{u}, t) p_t(\mathbf{u})\Big]\, \mathrm{d}\mathbf{u} + \mathcal{O}(\Delta t^2), \end{aligned}$$

Taylor Expansion. For any smooth function $\phi : \mathbb{R}^D \to \mathbb{R}$ and scale $\sigma > 0$, if $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the following approximation holds (known as the *Taylor–Gaussian smoothing formula*):

$$\int \mathcal{N}(\mathbf{x}; \mathbf{u}, \sigma^2 \mathbf{I}) \phi(\mathbf{u}) \, \mathrm{d}\mathbf{u} = \mathbb{E}\left[\phi(\mathbf{x} + \sigma \mathbf{z})\right] = \phi(\mathbf{x}) + \frac{\sigma^2}{2} \Delta_{\mathbf{x}} \phi(\mathbf{x}) + \mathcal{O}(\sigma^4).$$

This is because Taylor expansion for:

$$\phi(\mathbf{x} + \sigma \mathbf{z}) = \phi(\mathbf{x}) + \sigma \nabla_{\mathbf{x}} \phi(\mathbf{x}) \cdot \mathbf{z} + \frac{\sigma^2}{2} \mathbf{z}^\top \nabla_{\mathbf{x}}^2 \phi(\mathbf{x}) \mathbf{z} + \mathcal{O}(\sigma^3)$$

and $\mathbb{E}[\mathbf{z}] = \mathbf{0}$, $\mathbb{E}[\mathbf{z}\mathbf{z}^\top] = \mathbf{I}$.

Apply this with $\phi = p_t$, $\phi = \mathbf{f} \cdot \nabla_{\mathbf{u}} p_t$, and $\phi = (\nabla_{\mathbf{u}} \cdot \mathbf{f}) p_t$, and use $\sigma^2 = g^2(t) \Delta t$, we can obtain

$$\begin{aligned}
& p_{t+\Delta t}(\mathbf{x}) - p_t(\mathbf{x}) \\
= & -\Delta t \mathbf{f}(\mathbf{x}, t) \cdot \nabla_{\mathbf{x}} p_t(\mathbf{x}) - \Delta t (\nabla_{\mathbf{x}} \cdot \mathbf{f})(\mathbf{x}, t) p_t(\mathbf{x}) + \frac{g^2(t)}{2} \Delta t \Delta_{\mathbf{x}} p_t(\mathbf{x}) + \mathcal{O}(\Delta t^2) \\
= & -\Delta t \nabla_{\mathbf{x}} \cdot \left(\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})\right) + \frac{g^2(t)}{2} \Delta t \Delta_{\mathbf{x}} p_t(\mathbf{x}) + \mathcal{O}(\Delta t^2).
\end{aligned}$$

Divide by Δt and let $\Delta t \to 0$ to obtain the Fokker–Planck equation.

In Section C.1.4, we present the Itô–based derivation to complement the discrete–time view above.

4.6.2 Why Does Reverse-Time SDE Take The Form?

The rigorous derivation of the reverse-time SDE is technical and requires delving into the properties of the Fokker-Planck equation. However, the form of the reverse-time SDE can be understood intuitively through Bayes' theorem. Here, we present a heuristic derivation to provide insight into why Equation (4.1.6) takes its form, with the appearance of score functions[14].

Using Bayes' Rule for Inversion. Our goal is to determine the reverse-time transition kernel by first considering the discrete-time case:

$$p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}),$$

and then taking $\Delta t \to 0$ to obtain the continuous-time formulation. Using Bayes' theorem, we express:

$$\begin{aligned}
p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}) &= p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) \frac{p_t(\mathbf{x}_t)}{p_{t+\Delta t}(\mathbf{x}_{t+\Delta t})} \\
&= p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) \exp\left(\log p_t(\mathbf{x}_t) - \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t})\right). \qquad (4.6.1)
\end{aligned}$$

[14] This derivation is inspired by the approach in this post.

The forward transition kernel is assumed to be as in Equation (4.1.2):

$$p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t, g^2(t)\Delta t\mathbf{I}\right)$$

Taylor Expansion. To handle the exponential term, we apply a first-order Taylor expansion. The key insight is to expand around the point $(\mathbf{x}_t, t)$ in both space and time:

$$\begin{aligned}\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = \log p_t(\mathbf{x}_t) + \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ + \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t}\Delta t + \mathcal{O}(\|\mathbf{h}\|_2^2)\end{aligned}$$

where $\mathbf{h} := (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t, \Delta t)$. Therefore:

$$\begin{aligned}\log p_t(\mathbf{x}_t) - \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = -\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ - \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t}\Delta t + \mathcal{O}(\|\mathbf{h}\|_2^2)\end{aligned} \tag{4.6.2}$$

For a diffusion process, $\mathbf{x}_{t+\Delta t} - \mathbf{x}_t = \mathcal{O}_p(\sqrt{\Delta t})$, so the second-order spatial Taylor terms are generally of order $\mathcal{O}_p(\Delta t)$. Accordingly, the following calculation should be understood as a leading-order heuristic.

Substituting into the Reverse Transition. Substituting equations Equation (4.1.2) and Equation (4.6.2) into Equation (4.6.1):

$$\begin{aligned}&p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) \\ =&\frac{1}{(2\pi g^2(t)\Delta t)^{D/2}} \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t)\Delta t\|_2^2}{2g^2(t)\Delta t}\right) \\ &\cdot \exp\left(-\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) - \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t}\Delta t + \mathcal{O}(\Delta t)\right).\end{aligned}$$

Algebraic Manipulation. The key step is to complete the square in the exponent. We have:

$$\begin{aligned}&-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t)\Delta t\|_2^2}{2g^2(t)\Delta t} - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ =&-\frac{[\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t)\Delta t\|_2^2 + 2g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t)]}{2g^2(t)\Delta t}\end{aligned}$$

Let $\boldsymbol{\delta} := \mathbf{x}_{t+\Delta t} - \mathbf{x}_t$ and $\boldsymbol{\mu} := \mathbf{f}(\mathbf{x}_t, t)\Delta t$. Then:

$$\begin{aligned}&\|\boldsymbol{\delta} - \boldsymbol{\mu}\|_2^2 + 2g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot \boldsymbol{\delta} \\ =&\|\boldsymbol{\delta}\|_2^2 - 2\boldsymbol{\delta} \cdot \boldsymbol{\mu} + \|\boldsymbol{\mu}\|_2^2 + 2g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot \boldsymbol{\delta} \\ =&\|\boldsymbol{\delta}\|_2^2 - 2\boldsymbol{\delta} \cdot [\boldsymbol{\mu} - g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)] + \|\boldsymbol{\mu}\|_2^2 \\ =&\|\boldsymbol{\delta} - [\boldsymbol{\mu} - g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\|_2^2 - \|g^2(t)\Delta t\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2\end{aligned}$$

Substituting back:

$$\begin{aligned}&\|\boldsymbol{\delta} - [\mathbf{f}(\mathbf{x}_t, t)\Delta t - g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\|_2^2 \\ =&\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t\|_2^2.\end{aligned}$$

Therefore,

$$\begin{aligned}p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) = &\frac{1}{(2\pi g^2(t)\Delta t)^{D/2}} \\ &\cdot \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t\|_2^2}{2g^2(t)\Delta t}\right) \\ &\cdot \exp(\mathcal{O}(\Delta t)) \\ =&\, \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t, g^2(t)\Delta t \mathbf{I}\right) \\ &\cdot (1 + \mathcal{O}(\Delta t)).\end{aligned}$$

The additional term $\|g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2$ from completing the square is $\mathcal{O}((\Delta t)^2)$ and can be absorbed into the error term. Similarly, the time derivative term $\frac{\partial \log p_t(\mathbf{x}_t)}{\partial t}\Delta t$ is $\mathcal{O}(\Delta t)$ and will vanish in the continuous limit. It is harmless since it appears only as the multiplicative factor $\exp(\mathcal{O}(\Delta t)) = 1 + \mathcal{O}(\Delta t)$.

Taking $\Delta t \to 0$ Limit. As $\Delta t \approx 0$, under smoothness assumptions, the following approximations hold:

$$\begin{aligned}\mathbf{f}(\mathbf{x}_t, t) &\approx \mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t), \\ g(t) &\approx g(t + \Delta t), \\ \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) &\approx \nabla_{\mathbf{x}} \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = \mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t).\end{aligned}$$

Using these approximations and some rearrangements, we obtain:

$$\begin{aligned}&p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) \\ \approx&\frac{1}{(2\pi g^2(t)\Delta t)^{D/2}} \exp\Bigg(\\ &-\frac{\left\|\mathbf{x}_t - \left(\mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t) - g^2(t + \Delta t)\mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t)]\Delta t\right)\right\|_2^2}{2g^2(t + \Delta t)\Delta t} \\ &\Bigg).\end{aligned}$$

This implies that $p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t})$ is roughly a normal distribution with:

$$\begin{aligned}\textbf{Mean:}\quad &\mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t) - g^2(t + \Delta t)\mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t)]\Delta t, \\ \textbf{Covariance:}\quad &g^2(t + \Delta t)\Delta t \mathbf{I}.\end{aligned}$$

Taking the limit as $\Delta t \to 0$, we "derive" the reverse-time continuous SDE given in Equation (4.1.6).

4.7 Closing Remarks

This chapter marked a pivotal moment in our journey, unifying the discrete-time diffusion processes from the variational and score-based perspectives into a single, elegant continuous-time framework. We demonstrated that both DDPM and NCSN can be understood as discretizations of Stochastic Differential Equations (SDEs) with different drift/volatility coefficients.

The cornerstone of this framework is the existence of a corresponding reverse-time SDE, which formally defines a generative process that reverses the noise corruption. Crucially, the drift of this reverse process depends on a single unknown quantity: the score function, $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, of the marginal data distributions at every point in time. This insight solidifies the score function's central role in generative modeling.

Furthermore, we introduced a purely deterministic counterpart, the Probability Flow Ordinary Differential Equation (PF-ODE), whose solution trajectories evolve along the same marginal densities $\{p_t\}$ as the SDEs. This remarkable consistency is guaranteed by the underlying Fokker-Planck equation. The profound implication is that the complex task of generation is fundamentally equivalent to solving a differential equation. Training reduces to learning the score function that defines the equation's vector field, while sampling becomes a problem of numerical integration.

The introduction of the PF-ODE, a purely deterministic flow, provides a powerful bridge to the third and final perspective on diffusion models. This concept of learning a deterministic transformation governed by a velocity field is the central principle of recent major family of generative models. In the next chapter, we will:

1. Explore this flow-based perspective, starting from its origins in Normalizing Flows and Neural ODEs.

2. Show how this viewpoint leads to the modern framework of Flow Matching, which directly learns a velocity field to transport samples between distributions.

Ultimately, we will see how the deterministic PF-ODE, which we derived from stochastic principles, can be constructed and generalized from this entirely different, flow-based origin, completing our unified picture of diffusion modeling.

5

Flow-Based Perspective: From NFs to Flow Matching

Everything flows.

Heraclitus

The *change-of-variables formula*, a cornerstone of probability theory (Tabak and Vanden-Eijnden, 2010; Turner, 2013), takes on new life in modern generative modeling. In Chapter 4, we saw that the Fokker–Planck equation describes how probability densities evolve under a diffusion process, and that the same marginal density evolution can also be realized deterministically through the associated PF-ODE. This naturally suggests studying generative modeling directly from the viewpoint of deterministic probability transport.

Change-of-Variables Formula of Densities. Given an invertible transformation $\mathbf{f}$, the density of $\mathbf{x} = \mathbf{f}(\mathbf{z})$, where $\mathbf{z} \sim p_{\text{prior}}$, is

$$p(\mathbf{x}) = p_{\text{prior}}(\mathbf{z}) \left| \det \frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|, \qquad \text{where } \mathbf{z} = \mathbf{f}^{-1}(\mathbf{x}). \tag{5.0.1}$$

This deceptively simple formula enables bidirectional transport of samples and densities when $\mathbf{f}$ is tractable, forming the foundation of Normalizing Flows, which we introduce in Section 5.1. But what happens when this transformation is viewed continuously in time?

In this chapter, we follow this idea from Normalizing Flows to Neural ODEs and then to Flow Matching. The key connection to Chapter 4 is especially important. In Score SDEs, the learned score is used to construct the velocity field of the PF-ODE. Flow Matching instead learns a velocity field directly. For the same diffusion

probability path, these provide two views of the same deterministic probability flow.

A useful mental picture for this chapter is therefore, where both rely on the same conditioning principle:

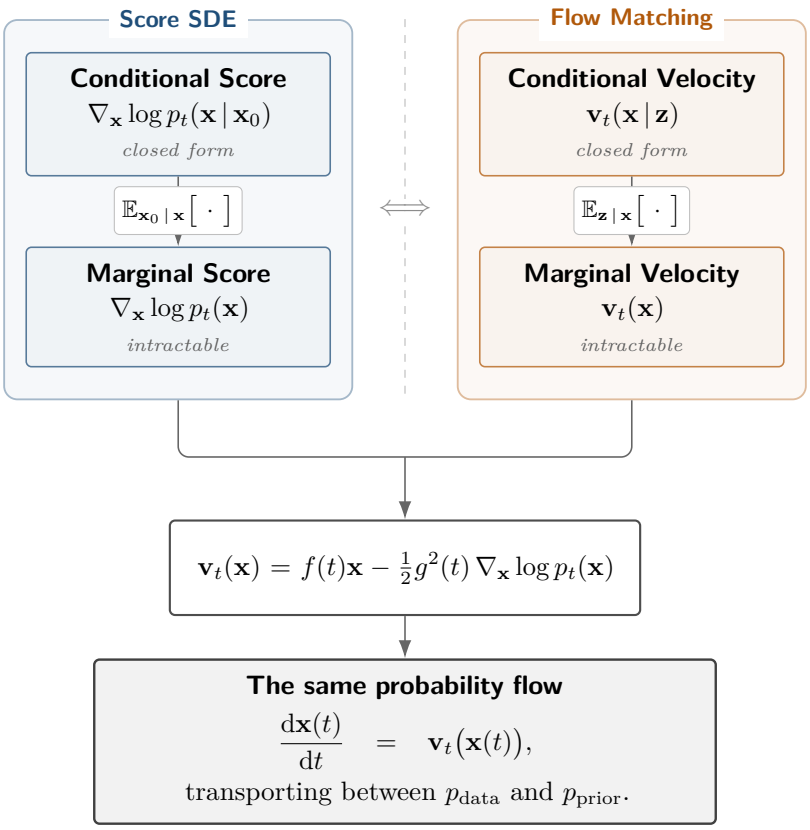


Figure 5.1: Score SDEs and Flow Matching. Both frameworks train on the closed-form *conditional* quantity in the top row. Averaging it over the conditioning variable (vertical arrows) gives the intractable *marginal* quantity below, which is what the trained model recovers. The columns differ only in which quantity that is. On a diffusion path the score determines the velocity through the conversion in the middle, so both routes end at the same ODE; Flow Matching learns that velocity directly and needs neither endpoint to be Gaussian.

Source: Created by the authors.

Flow Matching carries this principle beyond the usual Gaussian diffusion setting. It can construct continuous transport between more general source and target distributions from which samples are available, provided that the chosen conditional probability paths can be sampled efficiently and their conditional velocities can be computed tractably.

To support a solid understanding of this chapter, we provide in Chapter B an intuitive, self-contained overview of the different forms of the change-of-variables

principle, progressing from the ordinary change-of-variables formula to the continuity equation and, with stochastic dynamics, to the Fokker–Planck equation.

After this chapter, you will be able to...

- connect Normalizing Flows, Neural ODEs, and Flow Matching through the common idea of transporting probability distributions, from discrete invertible transformations to continuous-time velocity fields;
- recognize the key analogy with the Score SDE framework in Chapter 4: for the same diffusion probability path, Score SDEs learn a score that determines the PF-ODE velocity, whereas Flow Matching can learn that velocity field directly; in both cases, an intractable marginal target can be learned through tractable conditional targets;
- understand the continuity equation as the density-level rule that connects an ODE velocity field to the evolution of its probability distribution;
- understand how Flow Matching extends this conditional strategy beyond Gaussian diffusion paths to transport between general source and target distributions from which samples are available, while retaining simulation-free training through tractable conditional paths and velocities;
- distinguish the prescribed probability path $\{p_t\}_{t\in[0,1]}$ from the particular velocity field, coupling, and particle trajectories used to realize it, and understand why different flows can share the same marginal distributions;
- understand the two complementary ways of constructing tractable conditional flows: starting from a conditional probability path and deriving its velocity, or starting from a conditional flow map and deriving its velocity along trajectories.

Reader route. Readers already comfortable with Normalizing Flows and Neural ODEs may skim Section 5.1 and begin with Section 5.2.1. For a first pass, focus on the connection between the Score SDE and Flow Matching views, the continuity equation, and the conditional-to-marginal velocity construction. The detailed path constructions and optional theoretical properties can be revisited afterward.

5.1 Flow-Based Models: Normalizing Flows and Neural ODEs

In this section, we will introduce Flow-Based Models, including Normalizing Flows (NFs) (Rezende and Mohamed, 2015) and Neural Ordinary Differential Equations (NODEs) (Chen *et al.*, 2018).

NFs enable flexible and tractable probability density estimation by applying a series of invertible transformations to a simple base distribution. NODEs extend this framework to continuous time, where the transformation is governed by an ODE. By treating the transformations as continuous-time dynamics, NODEs provide a smooth, scalable extension to the NF paradigm.

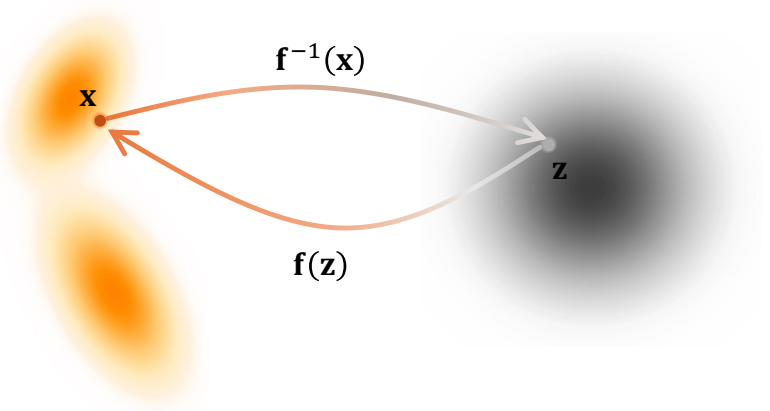


Figure 5.2: Illustration of sample movement of NF under an invertible map. It consists of a sequence of invertible functions $\mathbf{f} : \mathbf{z} \mapsto \mathbf{x}$ that transform latent variable $\mathbf{z}$ into a data $\mathbf{x}$, together with the inverse mapping $\mathbf{f}^{-1} : \mathbf{x} \mapsto \mathbf{z}$ that reconstructs the data. An NF resembles an encoder–decoder structure, but with the encoder realized as a smooth invertible map and the decoder given exactly by its inverse. The corresponding change in density can be computed via the change-of-variables formula, as given in Equation (5.0.1).

Source: Created by the authors.

5.1.1 Normalizing Flows

NFs (Rezende and Mohamed, 2015) model a complex data distribution $p_{\text{data}}(\mathbf{x})$ by transforming a simple prior $p_{\text{prior}}(\mathbf{z})$ (e.g., standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$) via an invertible mapping

$$\mathbf{f}_\phi : \mathbb{R}^D \to \mathbb{R}^D,$$

with $\mathbf{x} = \mathbf{f}_\phi(\mathbf{z})$ and $\mathbf{z} \sim p_{\text{prior}}$. Here, $\mathbf{x}$ and $\mathbf{z}$ share the same dimension. Using the

change-of-variables formula in Equation (5.0.1), the model likelihood is[1]

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) = \log p_{\text{prior}}(\mathbf{z}) + \log \left| \det \frac{\partial \mathbf{f}_{\boldsymbol{\phi}}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|. \tag{5.1.1}$$

Training Objective. Parameters $\boldsymbol{\phi}$ are learned by maximizing the likelihood over data:

$$\mathcal{L}_{\text{NF}}(\boldsymbol{\phi}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log p_{\boldsymbol{\phi}}(\mathbf{x}) \right]. \tag{5.1.2}$$

Computing the Jacobian determinant in Equation (5.1.1) can be costly, scaling as $\mathcal{O}(D^3)$ in general.

Constructing Invertible Transformations. A single complex invertible network can be expensive due to its Jacobian determinant. Conversely, simple transforms (e.g., linear) are efficient but lack expressivity.

To balance this, NFs employ a sequence of L trainable invertible mappings $\{\mathbf{f}_k\}_{k=0}^{L-1}$, each with efficiently computable Jacobians:

$$\mathbf{f}_{\boldsymbol{\phi}} = \mathbf{f}_{L-1} \circ \mathbf{f}_{L-2} \circ \cdots \circ \mathbf{f}_0.$$

Each $\mathbf{f}_k$ is parameterized by a neural network, though we omit the explicit dependence on $\boldsymbol{\phi}$ for notational simplicity.

Samples transform via

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k), \quad k = 0, \ldots, L-1, \tag{5.1.3}$$

with $\mathbf{z} = \mathbf{x}_0 \sim p_{\text{prior}}$ and $\mathbf{x} = \mathbf{x}_L$, corresponding to data. The resulting (log-)density is derived as

$$\begin{aligned} p_{\boldsymbol{\phi}}(\mathbf{x}) &= p_{\text{prior}}(\mathbf{x}_0) \prod_{k=0}^{L-1} \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|^{-1}, \text{ or equivalently,} \\ \log p_{\boldsymbol{\phi}}(\mathbf{x}) &= \log p_{\text{prior}}(\mathbf{x}_0) + \sum_{k=0}^{L-1} \log \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|^{-1}. \end{aligned} \tag{5.1.4}$$

Examples of Invertible Flows. Extensive literature has focused on designing single-layer flow constructions that enable efficient computation of the Jacobian. Below, we introduce two representative types: Planar Flows (Rezende and Mohamed, 2015) and Residual Flows (Chen *et al.*, 2019; Behrmann *et al.*, 2019), with the latter motivating the developments in Section 5.1.2.

[1] If the map is further constrained to be the gradient of a convex potential, $\mathbf{f}_{\boldsymbol{\phi}} = \nabla \psi_{\boldsymbol{\phi}}$ with $\psi_{\boldsymbol{\phi}}$ convex, then Equation (5.1.1) reduces to the Monge–Ampère relation in Equation (7.2.4). This PDE characterizes the optimal transformation of one distribution into another under the quadratic cost. See Chapter 7 and (Huang *et al.*, 2021) for further details.

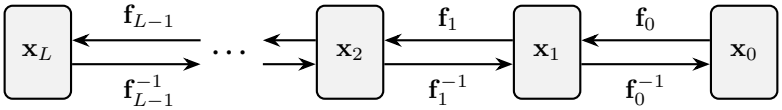


Figure 5.3: Illustration of a NF. An NF consists of a stack of invertible maps $\mathbf{f}_\phi = \mathbf{f}_{L-1} \circ \mathbf{f}_{L-2} \circ \cdots \circ \mathbf{f}_0$. The transformation maps latent samples $\mathbf{x}_0 \sim p_{\text{prior}}$ to data samples $\mathbf{x}_L \sim p_{\text{data}}$.

Source: Created by the authors.

Planar Flows: Planar flows (Rezende and Mohamed, 2015) apply a simple transformation

$$\mathbf{f}(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b),$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$, and $h(\cdot)$ is an activation function. The parameters are appropriately constrained or reparameterized to ensure that the transformation is invertible. Its Jacobian determinant has the simple form

$$\left| \det \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right| = \left| 1 + h'(\mathbf{w}^\top \mathbf{z} + b)\, \mathbf{w}^\top \mathbf{u} \right|,$$

making the change-of-variables computation efficient.

Lemma. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$, $\det(\mathbf{I} + \mathbf{x}\mathbf{y}^\top) = 1 + \mathbf{y}^\top \mathbf{x}$.

Proof. Multiplying out the block matrices,

$$\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{y}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} + \mathbf{x}\mathbf{y}^\top & \mathbf{x} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{y}^\top & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{y}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{x} \\ -\mathbf{y}^\top & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{x} \\ \mathbf{0} & \mathbf{y}^\top \mathbf{x} + 1 \end{pmatrix}.$$

The two outer factors are unit triangular with determinant 1, so taking determinants of both sides,

$$\det(\mathbf{I} + \mathbf{x}\mathbf{y}^\top) = \det \begin{pmatrix} \mathbf{I} & \mathbf{x} \\ \mathbf{0} & \mathbf{y}^\top \mathbf{x} + 1 \end{pmatrix} = 1 + \mathbf{y}^\top \mathbf{x}. \qquad \blacksquare$$

Proof of the planar-flow determinant. By the chain rule,

$$\frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \mathbf{I} + \mathbf{u}\, h'(\mathbf{w}^\top \mathbf{z} + b)\, \frac{\partial}{\partial \mathbf{z}} (\mathbf{w}^\top \mathbf{z}) = \mathbf{I} + h'(\mathbf{w}^\top \mathbf{z} + b)\, \mathbf{u}\mathbf{w}^\top,$$

so by the lemma with $\mathbf{x} = h'(\mathbf{w}^\top \mathbf{z} + b)\, \mathbf{u}$ and $\mathbf{y} = \mathbf{w}$,

$$\det \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \det \left(\mathbf{I} + h'(\mathbf{w}^\top \mathbf{z} + b)\, \mathbf{u}\mathbf{w}^\top \right) = 1 + h'(\mathbf{w}^\top \mathbf{z} + b)\, \mathbf{w}^\top \mathbf{u}. \qquad \blacksquare$$

Residual Flows: Define the transform $\mathbf{f}$ as

$$\mathbf{f}(\mathbf{z}) = \mathbf{z} + \mathbf{v}(\mathbf{z}), \tag{5.1.5}$$

with $\mathbf{v}$ contractive (Lipschitz constant < 1). This ensures invertibility via the Banach fixed-point theorem.

Proof. Fix $\mathbf{y}$ and set $T_{\mathbf{y}}(\mathbf{z}) := \mathbf{y} - \mathbf{v}(\mathbf{z})$, so that $\mathbf{f}(\mathbf{z}) = \mathbf{y}$ if and only if $T_{\mathbf{y}}(\mathbf{z}) = \mathbf{z}$. For any $\mathbf{z}_1, \mathbf{z}_2$,

$$\|T_{\mathbf{y}}(\mathbf{z}_1) - T_{\mathbf{y}}(\mathbf{z}_2)\| = \|(\mathbf{y} - \mathbf{v}(\mathbf{z}_1)) - (\mathbf{y} - \mathbf{v}(\mathbf{z}_2))\| = \|\mathbf{v}(\mathbf{z}_1) - \mathbf{v}(\mathbf{z}_2)\| \leq L\,\|\mathbf{z}_1 - \mathbf{z}_2\|,$$

with $L < 1$ since $\mathbf{v}$ is contractive. By the Banach fixed-point theorem $T_{\mathbf{y}}$ has a unique fixed point, so $\mathbf{f}(\mathbf{z}) = \mathbf{y}$ has a unique solution $\mathbf{z}$ for every $\mathbf{y}$, i.e., $\mathbf{f}$ is invertible. ∎

The log-determinant of the Jacobian reduces to a trace expansion:

$$\begin{aligned}
\log \left| \det \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right| &= \log \left(\det \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \\
&= \operatorname{Tr} \left(\log \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right) \\
&= \operatorname{Tr} \left(\log \left(\mathbf{I} + \frac{\partial \mathbf{v}(\mathbf{z})}{\partial \mathbf{z}} \right) \right) \\
&= \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \operatorname{Tr} \left(\left(\frac{\partial \mathbf{v}(\mathbf{z})}{\partial \mathbf{z}} \right)^k \right), \qquad (5.1.6)
\end{aligned}$$

making evaluation efficient via trace estimators (Hutchinson, 1989).

Lemma 1. For any convergent matrix power series $f(\mathbf{A}) = \sum_{n \geq 0} c_n \mathbf{A}^n$ and invertible $\mathbf{Q}$, $f(\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q}) = \mathbf{Q}^{-1} f(\mathbf{A}) \mathbf{Q}$.

Proof. If $(\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q})^{n-1} = \mathbf{Q}^{-1}\mathbf{A}^{n-1}\mathbf{Q}$, then $(\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q})^n = \mathbf{Q}^{-1}\mathbf{A}^{n-1}\mathbf{Q}\,\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q} = \mathbf{Q}^{-1}\mathbf{A}^n\mathbf{Q}$, so by induction $(\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q})^n = \mathbf{Q}^{-1}\mathbf{A}^n\mathbf{Q}$ for all n. Hence

$$f(\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q}) = \sum_{n \geq 0} c_n (\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q})^n = \sum_{n \geq 0} c_n \mathbf{Q}^{-1}\mathbf{A}^n\mathbf{Q} = \mathbf{Q}^{-1} f(\mathbf{A}) \mathbf{Q}. \qquad ∎$$

Lemma 2 (the trace is cyclic). For $n \geq 2$, $\operatorname{Tr}(\mathbf{X}_1 \cdots \mathbf{X}_n) = \operatorname{Tr}(\mathbf{X}_n \mathbf{X}_1 \cdots \mathbf{X}_{n-1}) = \operatorname{Tr}(\mathbf{X}_2 \cdots \mathbf{X}_n \mathbf{X}_1)$.

Proof for $n = 3$.

$$\begin{aligned}
\operatorname{Tr}(\mathbf{X}_1\mathbf{X}_2\mathbf{X}_3) &= \sum_i (\mathbf{X}_1\mathbf{X}_2\mathbf{X}_3)_{ii} = \sum_i \sum_j \sum_k (\mathbf{X}_1)_{ij} (\mathbf{X}_2)_{jk} (\mathbf{X}_3)_{ki} \\
&= \sum_k \sum_i \sum_j (\mathbf{X}_3)_{ki} (\mathbf{X}_1)_{ij} (\mathbf{X}_2)_{jk} = \operatorname{Tr}(\mathbf{X}_3\mathbf{X}_1\mathbf{X}_2) \\
&= \sum_j \sum_k \sum_i (\mathbf{X}_2)_{jk} (\mathbf{X}_3)_{ki} (\mathbf{X}_1)_{ij} = \operatorname{Tr}(\mathbf{X}_2\mathbf{X}_3\mathbf{X}_1). \qquad ∎
\end{aligned}$$

Proof of $\log\det \mathbf{J} = \operatorname{Tr}(\log \mathbf{J})$ *in Equation* (5.1.6). Let $\mathbf{J} := \partial \mathbf{f} / \partial \mathbf{z}$. By the Schur decomposition, $\mathbf{J} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*$ with $\mathbf{T}$ upper triangular and $\mathbf{Q}$ unitary. Then

$$\operatorname{Tr}(\log \mathbf{J}) = \operatorname{Tr}\left(\log(\mathbf{Q}\mathbf{T}\mathbf{Q}^*)\right) \overset{\text{L1}}{=} \operatorname{Tr}(\mathbf{Q} \log \mathbf{T}\, \mathbf{Q}^*) \overset{\text{L2}}{=} \operatorname{Tr}(\log \mathbf{T}\, \mathbf{Q}^*\mathbf{Q}) = \operatorname{Tr}(\log \mathbf{T}),$$

and likewise $\log\det \mathbf{J} = \log\det(\mathbf{Q}\mathbf{T}\mathbf{Q}^*) = \log\det \mathbf{T}$. Let $\lambda_1, \ldots, \lambda_D$ be the diagonal entries of $\mathbf{T}$. Since $\mathbf{T}$ is upper triangular, so are $\mathbf{T} - \mathbf{I}$ and every power $(\mathbf{T} - \mathbf{I})^k$, hence so is

$$\log \mathbf{T} = \log\left(\mathbf{I} + (\mathbf{T} - \mathbf{I})\right) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} (\mathbf{T} - \mathbf{I})^k,$$

whose diagonal entries are $(\log \mathbf{T})_{ii} = \sum_{k\geq 1} \frac{(-1)^{k+1}}{k} (\lambda_i - 1)^k = \log \lambda_i$. Therefore

$$\mathrm{Tr}(\log \mathbf{T}) = \sum_{i=1}^{D} \log \lambda_i = \log \prod_{i=1}^{D} \lambda_i = \log\det \mathbf{T},$$

and so $\log\det \mathbf{J} = \mathrm{Tr}(\log \mathbf{J})$. The series converges because $\|\mathbf{T} - \mathbf{I}\|_2 = \|\mathbf{Q}^*(\mathbf{J} - \mathbf{I})\mathbf{Q}\|_2 = \|\partial \mathbf{v}/\partial \mathbf{z}\|_2 < 1$ by contractivity of $\mathbf{v}$, using that $\mathbf{Q}$ is unitary. ■

Sampling and Inference. Sampling from NFs is straightforward: draw $\mathbf{x}_0 \sim p_{\text{prior}}$ and compute $\mathbf{x} = \mathbf{f}_{\boldsymbol{\phi}}(\mathbf{x}_0)$. Exact likelihoods are obtained from Equation (5.1.4).

5.1.2 Neural ODEs

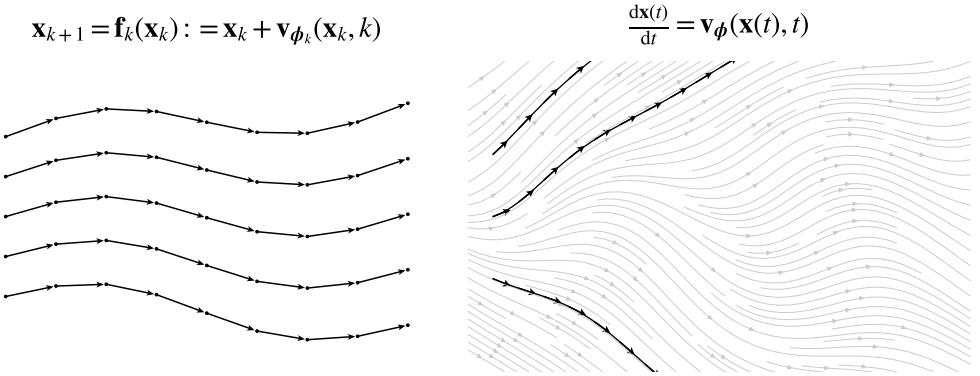


Figure 5.4: Discrete- vs. continuous-time normalizing flows. (Left) A discrete NF transports samples by a finite sequence of invertible maps $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k)$, yielding stepwise, non-crossing trajectories (dots with arrows). (Right) A continuous NF (Neural ODE) evolves states along integral curves of $\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t)$, where black paths with tangent arrows are shown over the gray vector field.

Source: Created by the authors.

From Discrete-Time NFs to Continuous-Time NFs (Neural ODEs). A discrete NF transports samples through a composition of L invertible maps. To obtain a meaningful continuous-depth limit, we write each layer as a small residual update:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t\, \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_k, t_k), \qquad t_k = k\Delta t, \qquad \Delta t = \frac{T}{L}.$$

Here, $\mathbf{v}_\phi$ specifies the direction in which each point is moved, while Δt controls the size of one layer.

The geometric meaning of this limit is illustrated in Figure 5.5. As L increases, the same source-to-target transport is divided into more near-identity bijections. The sample trajectories are resolved more finely, and the deformation of the surrounding reference grid becomes progressively smoother.

As $L \to \infty$ and $\Delta t \to 0$, the discrete layer index becomes continuous time, yielding the ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_\phi(\mathbf{x}(t), t).$$

This continuous-depth model is known as a Neural ODE or Continuous Normalizing Flow.

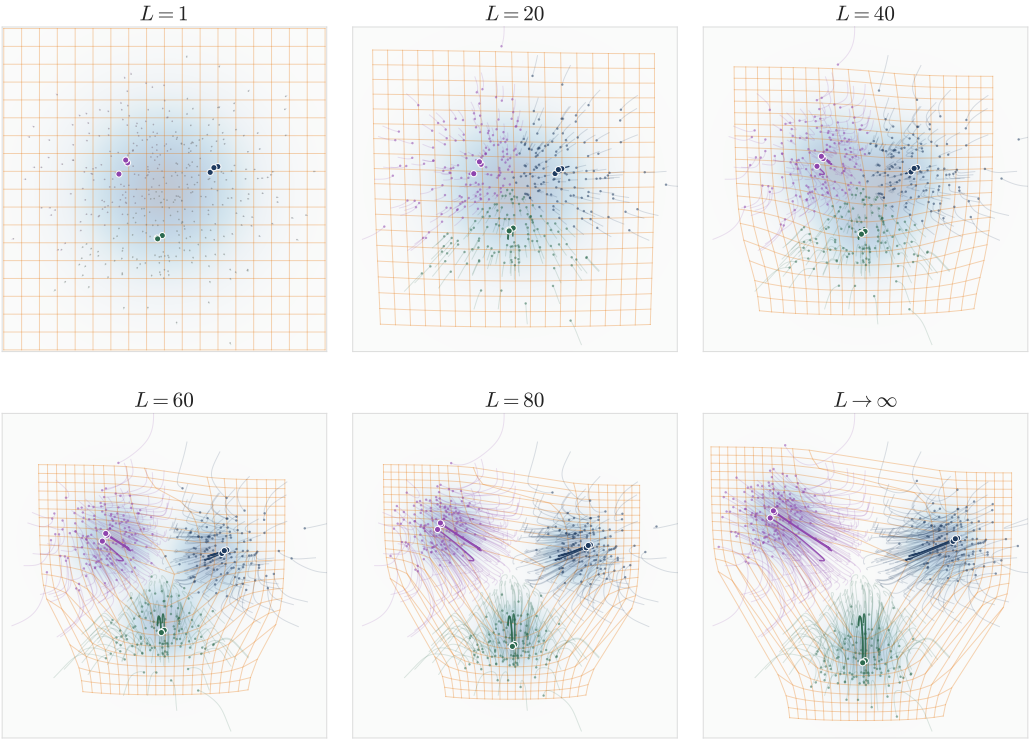


Figure 5.5: From discrete bijections to a continuous transport flow. Each panel shows a transport built by composing L invertible layers. Starting from a source distribution (Gaussian), the layers gradually move the colored samples toward a target distribution with three modes. The orange reference grid shows how the surrounding space is deformed during this transport. With only a few layers, each layer makes a relatively large and discrete change. As L increases, the transport is divided into more, smaller updates, so the sample trajectories and grid deformation become smoother. If the total transport time T is fixed and each layer has step size $\Delta t = T/L$, then, as $L \to \infty$, the stacked bijections approach a continuous time-dependent flow. The transported density evolves according to the continuity equation in Equation (5.2.8).

Source: Created by the authors with AI-assisted coding.

Formal Setup of Neural ODEs. A Neural ODE defines a continuous transformation through:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t), \quad t \in [0, T] \tag{5.1.7}$$

where:

- $\mathbf{x}(t) \in \mathbb{R}^D$ is the state at time t; we sometimes write $\mathbf{x}_t$ for brevity;
- $\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t)$ is a neural network parameterized by $\boldsymbol{\phi}$.

Goal of NODE. Starting from the initial condition $\mathbf{x}(0) \sim p_{\text{prior}}$, the ODE evolves the state continuously over time, inducing a family of marginal distributions $p_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$ (similar to PF-ODEs!)[2].

The goal is to learn the neural vector field $\mathbf{v}_{\boldsymbol{\phi}}$, which intuitively represents a velocity that transports points along continuous trajectories in data space. By learning this velocity, the terminal distribution at $t = T$ matches the target distribution $p_{\text{data}}(\cdot)$. This continuous transformation unifies discrete normalizing flows and neural ODEs within a single framework.

Continuous-Time Change-of-Variables Formula. Analogous to Equation (5.0.1) or Equation (5.1.4), Chen *et al.* (2018) derived a continuous-time analogue of the change-of-variables formula. For the time-dependent density $p_{\boldsymbol{\phi}}(\mathbf{x}(t), t)$ of a process $\mathbf{x}(t)$ evolving under Equation (5.1.7), the so-called *Instantaneous Change-of-Variables Formula* is

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p_{\boldsymbol{\phi}}(\mathbf{x}(t), t) = -\nabla_{\mathbf{x}} \cdot \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t).$$

Here, the derivative is taken along the ODE trajectory $\mathbf{x}(t)$.

Therefore, if $\mathbf{x}(0) \sim p_{\text{prior}}$ and the ODE transports the prior from $t = 0$ to $t = T$, integrating this identity gives

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}(T), T) = \log p_{\text{prior}}(\mathbf{x}(0)) - \int_0^T \nabla_{\mathbf{x}} \cdot \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t)\, \mathrm{d}t. \tag{5.1.8}$$

This expression provides likelihood evaluation through numerical integration, which in turn enables maximum likelihood training of the model.

Although this formula may appear unfamiliar at first, it is closely connected to the Fokker–Planck equation. In particular, it can be understood as the along-trajectory form of the *continuity equation*, which is the deterministic, zero-diffusion case of the Fokker–Planck equation (see Chapter B). It can also be interpreted as the

[2] We adopt a flipped time convention, with $t = 0$ denoting the prior (source) and $t = 1$ the data (target) distribution. The prior is interchangeably written as $p_{\boldsymbol{\phi}}(\mathbf{x}(0), 0)$, $p_{\text{prior}}(\mathbf{x}(0), 0)$, or simply $p_{\text{prior}}(\mathbf{z})$.

continuous-time limit of the discrete change-of-variables formula in Equation (5.1.4). We summarize the result below.

Lemma 5.1.1: Instantaneous Change of Variables

Let $\mathbf{z}(t)$ be a continuous random process with time-dependent density $p(\mathbf{z}(t), t)$, and suppose it evolves according to the ODE

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{F}(\mathbf{z}(t), t).$$

Assuming sufficient regularity so that the ODE flow and the evolving density are well defined, the log-density along the trajectory satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p(\mathbf{z}(t), t) = -\nabla_{\mathbf{z}} \cdot \mathbf{F}(\mathbf{z}(t), t). \tag{5.1.9}$$

Proof for Lemma.

This follows from the continuity equation together with the chain rule. We present detailed alternative derivations in Section D.3.

Connection to Discrete-Time Formula. The NODE likelihood in Equation (5.1.8),

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}(T), T) = \log p_{\text{prior}}(\mathbf{x}(0), 0) - \int_0^T \nabla_{\mathbf{x}} \cdot \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}(t), t)\, \mathrm{d}t,$$

can be seen as the continuous-time analogue of the discrete normalizing flow formulation in Equation (5.1.4):

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}_L) = \log p_{\text{prior}}(\mathbf{x}_0) - \sum_{k=0}^{L-1} \log \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|.$$

The integral mirrors the summation, and the trace operator replaces the log-determinant, as discussed in Equation (5.1.6). These parallels are further explored in the proof of the lemma.

Training NODEs. Based on Equation (5.1.8), NODEs learn a parameterized velocity field $\mathbf{v}_{\boldsymbol{\phi}}$ such that the terminal distribution

$$p_{\boldsymbol{\phi}}(\cdot, T) \approx p_{\text{data}},$$

where trajectories evolve from latent variables $\mathbf{x}(0) \sim p_{\text{prior}}$ via the ODE flow. Training follows the MLE framework from Equation (1.1.2):

$$\mathcal{L}_{\text{NODE}}(\boldsymbol{\phi}) := \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \big[\log p_{\boldsymbol{\phi}}(\mathbf{x}, T) \big].$$

Exact Log-Likelihood Computation. To compute $\log p_{\phi}(\mathbf{x}, T)$ for data point $\mathbf{x}$, we integrate the change-of-variables formula equation 5.1.8:

$$\log p_{\phi}(\mathbf{x}, T) = \log p_{\text{prior}}(\mathbf{z}(0)) - \int_0^T \nabla_{\mathbf{z}} \cdot \mathbf{v}_{\phi}(\mathbf{z}(t), t)\, \mathrm{d}t. \tag{5.1.10}$$

Here, $\mathbf{z}(t)$ solves the ODE reversely from $t = T$ to $t = 0$:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{v}_{\phi}(\mathbf{z}(t), t)$$

with $\mathbf{z}(T) = \mathbf{x}$. The prior term $\log p_{\text{prior}}(\mathbf{z}(0))$ is tractable for standard distributions. This enables exact likelihood-based training and evaluation in neural ODEs.

Gradient-Based Optimization. Maximizing $\mathcal{L}_{\text{NODE}}$ requires backpropagation through the ODE solver. The adjoint sensitivity method (Pontryagin, 2018; Chen *et al.*, 2018) computes gradients via an auxiliary ODE with $\mathcal{O}(1)$ memory complexity, but NODE training remains expensive due to numerical integration at each step.

Inference with NODEs. Sampling with a trained model $\mathbf{v}_{\phi^\times}$ proceeds by drawing $\mathbf{x}(0) \sim p_{\text{prior}}$ and integrating forward (by numerical solvers):

$$\mathbf{x}(T) = \mathbf{x}(0) + \int_0^T \mathbf{v}_{\phi^\times}(\mathbf{x}(t), t)\, \mathrm{d}t.$$

The terminal state $\mathbf{x}(T)$ approximates a sample from p_{data}.

Moreover, we note that for any vector field $\mathbf{F}$, the following identity holds:

$$\operatorname{Tr}\left(\frac{\partial \mathbf{F}}{\partial \mathbf{z}(t)}\right) = \nabla_{\mathbf{z}} \cdot \mathbf{F}.$$

Hence, the divergence can be estimated efficiently using stochastic trace estimators, such as Hutchinson's estimator (Hutchinson, 1989), without explicitly constructing the full Jacobian. This makes likelihood evaluation much more practical in high-dimensional settings. However, because the trace is then estimated from random samples, the resulting numerical log-likelihood estimate is stochastic rather than an exact evaluation of the divergence.

Having seen in Figure 5.5 how many small bijections approach a continuous transport flow, we now ask a practical question:

Question 5.1.1

How can we learn its velocity field without repeatedly solving an ODE during training?

This question leads naturally to Flow Matching, which learns the velocity field directly through a simulation-free regression objective and provides the flow-based perspective on diffusion models.

5.2 Flow Matching Framework

Both Score SDEs (Chapter 4) and NODEs (Section 5.1) describe generative modeling as a continuous transport process. Score SDEs formulate this transport through stochastic dynamics together with an associated PF-ODE, whereas NODEs directly parameterize a deterministic ODE. In the standard generative setting, both aim to transport a simple Gaussian prior p_{prior} into the data distribution p_{data}.

A practical limitation of NODEs is that likelihood-based training requires repeatedly solving an ODE. The *Flow Matching* (FM) framework (Lipman *et al.*, 2022; Lipman *et al.*, 2024; Tong *et al.*, 2024) avoids this cost by training a neural network to predict the velocity field directly through a regression objective. ODE integration is therefore needed for generation, but not inside the training loop, making FM training simulation-free.

More generally, FM is not restricted to Gaussian-to-data transport. It considers a source distribution p_{src} and a target distribution p_{tgt}, both assumed to be easy to sample from. Standard generative modeling is recovered as the special case

$$p_{\text{src}} = p_{\text{prior}}, \qquad p_{\text{tgt}} = p_{\text{data}},$$

where p_{prior} is typically Gaussian.

In this section, we adopt the FM viewpoint[3] Its goal is to learn a time-dependent velocity field $\mathbf{v}_t(\mathbf{x}_t)$ whose associated ODE transports samples along a prescribed probability path $\{p_t\}_{t\in[0,1]}$ satisfying

$$p_0 = p_{\text{src}}, \qquad p_1 = p_{\text{tgt}}.$$

When p_{src} is Gaussian, we refer to this setting as *Gaussian Flow Matching*. Thus, FM provides both a simulation-free training method for generative modeling and a general framework for learning continuous transport between arbitrary endpoint distributions.

5.2.1 Lesson from Score-Based Methods

We revisit the Score SDE framework (Chapter 4) using a slightly different but equivalent formulation to extract key insights that motivate the FM approach. This analysis reveals how diffusion models implicitly learn probability flows and motivates a more direct formulation.

[3]Several related approaches share the core idea of transporting between endpoint distributions using a continuous-time flow, although their formulations differ slightly. These include Flow Matching (FM) (Lipman *et al.*, 2022; Neklyudov *et al.*, 2023), Rectified Flow (RF) (Liu, 2022; Heitz *et al.*, 2023), and Stochastic Interpolants (Albergo *et al.*, 2023; Albergo and Vanden-Eijnden, 2023; Ma *et al.*, 2024). Here, we use FM as a unifying terminology.

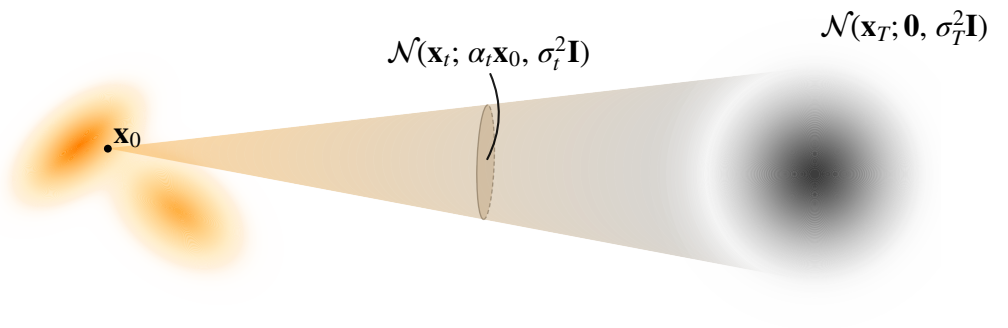


Figure 5.6: Illustration of the conditional transition distribution. $p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I})$, defines a (Gaussian) conditional probability path from a data sample $\mathbf{x}_0 \sim p_{\text{data}}$ (left) towards the Gaussian prior p_{prior} (right).

Source: Created by the authors.

Step 1: Defining a Conditional Path and Its Marginal Densities. A diffusion model specifies a continuous-time family of densities $\{p_t\}_{t\in[0,1]}$ that transports a simple prior p_{prior} (e.g., Gaussian) at $t = 1$, used as the source, to a target data distribution p_{data} at $t = 0$:

$$p_1(\mathbf{x}_1) = p_{\text{prior}}(\mathbf{x}_1), \quad p_0(\mathbf{x}_0) = p_{\text{data}}(\mathbf{x}_0).$$

This path is implicitly defined via the forward conditional distribution

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I}), \quad \mathbf{x}_0 \sim p_{\text{data}} \tag{5.2.1}$$

which induces the marginal density

$$p_t(\mathbf{x}_t) := \int p_t(\mathbf{x}_t|\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_0)\mathrm{d}\mathbf{x}_0.$$

The increasing variance σ_t^2 of the conditional Gaussian drives the evolution of p_t toward the Gaussian prior.

Step 2: Velocity Field. The time evolution of the marginal density p_t is governed by a velocity field $\mathbf{v}_t : \mathbb{R}^D \to \mathbb{R}^D$, derived from the Fokker-Planck equation:

$$\mathbf{v}_t(\mathbf{x}) := f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}), \tag{5.2.2}$$

which defines a deterministic particle flow through the PF-ODE:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underbrace{f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t\big(\mathbf{x}(t)\big)}_{\mathbf{v}_t(\mathbf{x}(t))}.$$

This ODE transports an initial random variable $\mathbf{x}(0) \sim p_{\text{data}}$ forward in time or $\mathbf{x}(1) \sim p_{\text{prior}}$ backward in time, such that the evolving marginal density of $\mathbf{x}(t)$ matches p_t at every $t \in [0, 1]$ (see "Underlying Rule" below).

The scalar functions $f(t)$ and $g(t)$ are determined by the coefficients of the associated forward SDE, or equivalently the Gaussian kernel parameters α_t and σ_t defined in the conditional path (see Lemma 4.5.1).

Step 3: Learning via the Conditional Strategy. The PF-ODE velocity in Equation (5.2.2) depends on the marginal score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, which is generally unknown. Score-based methods therefore learn this score using a neural network $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$ through the ideal score-matching objective

$$\mathcal{L}_{\text{SM}}(\boldsymbol{\phi}) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\, \mathbf{x}_t\sim p_t} \left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\|^2 \right].$$

Once the score is learned, the PF-ODE velocity is obtained from it as

$$\mathbf{v}_t(\mathbf{x}_t) = f(t)\mathbf{x}_t - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t),$$

with $\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x})$

The difficulty is that the marginal score itself is inaccessible when computing the regression target. As in Chapter 3, we overcome this using the conditioning trick. The conditional distribution $p_t(\mathbf{x}_t|\mathbf{x}_0)$ is tractable, and its conditional score satisfies

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0\sim p(\cdot|\mathbf{x}_t)} \left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0) \right]. \tag{5.2.3}$$

This motivates the denoising score matching objective

$$\mathcal{L}_{\text{DSM}}(\boldsymbol{\phi}) := \mathbb{E}_{t,\, \mathbf{x}_0\sim p_{\text{data}},\, \mathbf{x}_t\sim p_t(\cdot|\mathbf{x}_0)} \left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|^2 \right].$$

As shown in Theorem 3.3.1, $\mathcal{L}_{\text{SM}}$ and $\mathcal{L}_{\text{DSM}}$ differ only by a constant independent of $\boldsymbol{\phi}$ and therefore have the same population minimizer:

$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{\mathbf{x}_0\sim p(\cdot|\mathbf{x}_t)} \left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0) \right] = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

Importantly, the same conditioning principle can also be expressed directly at the level of the PF-ODE velocity. Define the conditional velocity

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0) := f(t)\mathbf{x}_t - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0).$$

Using Equation (5.2.3), we immediately obtain

$$\mathbf{v}_t(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0\sim p(\cdot|\mathbf{x}_t)} \left[\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0) \right].$$

Thus, score-based diffusion learns the score through tractable conditional targets and then uses it to construct the PF-ODE velocity. Flow Matching will apply the same conditional-to-marginal principle directly to the velocity field.

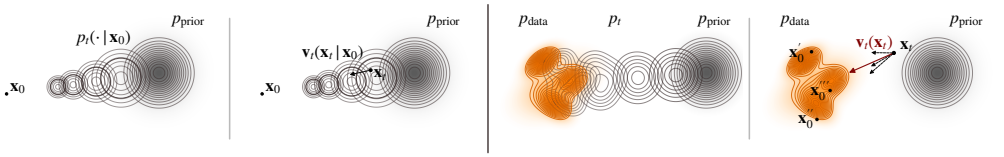


Figure 5.7: Illustration of conditional versus marginal perspectives in diffusion. (This figure is motivated by Lipman *et al.* (2024).) (1) Conditional Gaussian path $p_t(\cdot|\mathbf{x}_0)$, showing expanding densities from a fixed $\mathbf{x}_0$ toward the prior. (2) Conditional velocity $\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0) = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{x}_0)$. (3) Marginal density p_t, transporting the data distribution p_{data} (orange) into the prior p_{prior} (gray). (4) Marginal velocity $\mathbf{v}_t(\mathbf{x}_t) = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)$, obtained by averaging conditional directions from $\mathbf{x}_t$ to multiple plausible origins (dashed), yielding the red arrow. In the FM framework with one-sided conditioning $\mathbf{z} = \mathbf{x}_0$, the same illustration applies to $\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0)$ and $\mathbf{v}_t(\mathbf{x}_t)$, without requiring them to be written explicitly in terms of the scores $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{x}_0)$ or $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)$.

Source: Created by the authors.

Underlying Rule: The Fokker–Planck Equation. The marginal density p_t evolves according to the Fokker–Planck equation:

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot \left(\underbrace{\left(f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right)}_{\mathbf{v}_t(\mathbf{x})} p_t(\mathbf{x}) \right) = 0.$$

This PDE ensures that the density given by the PF-ODE matches the marginal distribution of the forward SDE. To see this, recall the *flow map* $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$ of the PF-ODE as defined in Equation (4.2.2), which carries an initial state $\mathbf{x}_s$ at time s directly to its state at t. Running the PF-ODE backward from $t = 1$ to $t = 0$, starting with $\mathbf{x}_1 \sim p_{\text{prior}}$, we obtain time-dependent densities through the pushforward formula:

$$p_t^{\text{rev}}(\mathbf{x}) = \int \delta\left(\mathbf{x} - \boldsymbol{\Psi}_{1\to t}(\mathbf{x}_1)\right) p_{\text{prior}}(\mathbf{x}_1)\mathrm{d}\mathbf{x}_1. \tag{5.2.4}$$

The Fokker–Planck equation ensures that the induced density path coincides with the same evolving density:

$$p_t^{\text{rev}} = p_t. \tag{5.2.5}$$

In particular, this implies $p_0^{\text{rev}} = p_0 = p_{\text{data}}$, thereby recovering the data distribution at time $t = 0$. Since the ODE solution map is bidirectional, we can similarly consider initializing at $\mathbf{x}_0 \sim p_{\text{data}}$ and solving the ODE forward in time, enabling a parallel analysis.

5.2.2 Flow Matching Framework

The analysis in Section 5.2.1 reveals that diffusion models succeed by learning a velocity field, specifically, the score, that transports between distributions while satisfying boundary conditions. The design of the Gaussian conditional path in Equation (5.2.1), with increasing variance σ_t^2, implicitly anchors one endpoint to a Gaussian prior while allowing the conditional density to be defined over the entire space, enabling score-based gradient computation.

In this subsection, we introduce the FM framework, which builds on this insight (the same illustration in Figure 5.7 also applies to the FM framework) and extends it to learning continuous flows that transport samples between two arbitrary distributions, p_{src} and p_{tgt}.

Step 1: Defining a Conditional Path and Its Marginal Densities. Consider arbitrary source and target probability distributions p_{src} and p_{tgt} on $\mathbb{R}^D$. We set[4]

$$p_0(\mathbf{x}) = p_{\text{src}}(\mathbf{x}), \quad p_1(\mathbf{x}) = p_{\text{tgt}}(\mathbf{x}). \tag{5.2.6}$$

FM implicitly defines a continuous family of intermediate densities $\{p_t\}_{t\in[0,1]}$ interpolating between these endpoints. Each marginal p_t is expressed via a latent variable $\mathbf{z}$ drawn from a known distribution $\pi(\mathbf{z})$ and a conditional distribution $p_t(\mathbf{x}_t|\mathbf{z})$:

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}, \tag{5.2.7}$$

with $(\pi(\mathbf{z}), \{p_t(\cdot|\mathbf{z})\})$ chosen to satisfy the boundary conditions in Equation (5.2.6).

We remark that, in general, the marginal densities p_t are not tractable, since they require integrating over $\pi(\mathbf{z})$, and both $\pi(\mathbf{z})$ and the conditional distributions $p_t(\mathbf{x}_t|\mathbf{z})$ can be complex. Nonetheless, conditioning on the latent $\mathbf{z}$ grants FM the flexibility to model a broad class of interpolation paths beyond those discussed in Section 5.2.1. Common choices for $\mathbf{z}$ include:

- **Two-sided conditioning:** $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1) \sim p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$, where π couples source and target distributions. This allows FM to define transport between arbitrary distributions.
- **One-sided conditioning:** $\mathbf{z} = \mathbf{x}_0$ or $\mathbf{z} = \mathbf{x}_1$. It especially recovers diffusion-like setups when the source distribution is chosen to be Gaussian.

In all cases, the conditional path $p_t(\mathbf{x}_t|\mathbf{z})$ should be efficiently sampleable, and the corresponding conditional velocity should be tractable. In the probability-path-first constructions below, we further use conditional distributions with tractable closed-form expressions. We present specific constructions in Section 5.3.2, with illustrations in Figure 5.9.

[4] To align with the standard notation in FM literature, we reverse the time axis compared to earlier sections: $t = 0$ corresponds to the source distribution and $t = 1$ to the target.

Step 2: Velocity Field. A probability path $\{p_t\}_{t\in[0,1]}$ specifies how the distribution should evolve over time, but it does not generally specify how individual samples should move. In other words, the same sequence of marginal densities can be realized by different velocity fields[5].

The goal of FM is therefore to find a velocity field $\mathbf{v}_t(\mathbf{x})$ whose induced ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_t(\mathbf{x}(t)), \qquad t \in [0,1],$$

produces marginal distributions that match the prescribed p_t at every time t. When this holds, the ODE can transport samples forward from $\mathbf{x}(0) \sim p_{\text{src}}$ to p_{tgt}, or backward from $\mathbf{x}(1) \sim p_{\text{tgt}}$ to p_{src} (see Section 5.2.4 for a more formal discussion).

This requirement is captured by the *continuity equation*[6]:

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot (\mathbf{v}_t(\mathbf{x}) p_t(\mathbf{x})) = 0. \tag{5.2.8}$$

Any velocity field $\mathbf{v}_t$ satisfying Equation (5.2.8), together with the initial condition $p_0 = p_{\text{src}}$, generates an ODE whose evolving marginals follow the prescribed path $\{p_t\}$, under suitable regularity conditions. Thus, solving the ODE provides a sample-wise transport between p_{src} and p_{tgt} while matching the intermediate distributions.

Importantly, the continuity equation constrains the evolution of probability density, but generally does not uniquely determine the velocity field. Different sample trajectories can therefore produce exactly the same marginal density evolution. For example, on regions where $p_t > 0$, if $\mathbf{v}_t$ satisfies Equation (5.2.8), then

$$\mathbf{v}_t + \frac{1}{p_t}\tilde{\mathbf{v}}_t$$

also satisfies it for any sufficiently regular divergence-free field $\tilde{\mathbf{v}}_t$ with $\nabla \cdot \tilde{\mathbf{v}}_t = 0$.

FM therefore seeks *a* velocity field that realizes the prescribed probability path. For general source and target distributions, neither the marginal density p_t nor such a velocity field is usually available in closed form. This motivates the conditional construction introduced next, where tractable conditional paths and velocities are used to learn a valid marginal velocity field.

[5] The key distinction is that a probability path specifies the evolution of the *distribution*, but generally not the particle-level transport that realizes it. A particular velocity becomes determined only after additional transport structure is specified. For example, a sufficiently regular and invertible flow map $\mathbf{\Psi}_{0\to t}$ determines its Eulerian velocity through $\mathbf{v}_t = \partial_t \mathbf{\Psi}_{0\to t} \circ \mathbf{\Psi}_{0\to t}^{-1}$. Likewise, once a forward diffusion SDE is specified, its associated PF-ODE selects the particular velocity $\mathbf{v}_t(\mathbf{x}) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. Thus, both cases add particle-level transport structure beyond the marginal path $\{p_t\}$ itself.

[6] The deterministic analogue of the Fokker–Planck equation, without the diffusion term.

As a concrete illustration, in Section 5.3.1 we consider a Gaussian-to-Gaussian path together with a particular affine flow map, for which the corresponding velocity field can be computed explicitly.

Step 3: Learning via the Conditional Strategy. Once a probability path and a corresponding marginal velocity field $\mathbf{v}_t$ have been chosen, FM learns this velocity field directly using a neural network $\mathbf{v}_{\boldsymbol{\phi}}$. The ideal FM objective is

$$\mathcal{L}_{\mathrm{FM}}(\boldsymbol{\phi}) = \mathbb{E}_{t,\,\mathbf{x}_t\sim p_t}\left[\left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t,t) - \mathbf{v}_t(\mathbf{x}_t)\right\|^2\right].$$

We refer to this neural network parameterization as $\mathbf{v}$-*prediction* (velocity prediction), since the network directly predicts the drift of the ODE. This differs from the score-based construction in Section 5.2.1, where the score is learned first and then used to construct the PF-ODE velocity.

The difficulty is that the marginal velocity $\mathbf{v}_t(\mathbf{x}_t)$ is generally intractable. As in denoising score matching, we overcome this using a conditioning strategy. We introduce a latent variable $\mathbf{z}\sim\pi(\mathbf{z})$ and construct a tractable conditional probability path $p_t(\mathbf{x}_t|\mathbf{z})$ together with a corresponding conditional velocity field $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$.

If each conditional velocity field generates its corresponding conditional probability path, then marginalizing over $\mathbf{z}$ gives a valid marginal velocity field:

$$\mathbf{v}_t(\mathbf{x}_t) := \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x}_t)}\left[\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\right]. \tag{5.2.9}$$

The posterior $p(\mathbf{z}|\mathbf{x}_t)$ depends on t. This is implied by the notation $p(\mathbf{z}\,|\,\mathbf{X}_t=\mathbf{x}_t)$, and could be written explicitly as $p_t(\mathbf{z}|\mathbf{x}_t)$. Thus, the generally intractable marginal velocity is the conditional average of tractable conditional velocities. This is directly analogous to the conditional-score identity in Equation (5.2.3).

This observation leads to the *conditional flow matching* (CFM) objective:

$$\mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) := \mathbb{E}_{t,\,\mathbf{z}\sim\pi(\mathbf{z}),\,\mathbf{x}_t\sim p_t(\cdot|\mathbf{z})}\left[\left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t,t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\right\|^2\right]. \tag{5.2.10}$$

Here, $\mathbf{z}$ is used to construct the training sample $\mathbf{x}_t$ and the conditional regression target $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$; it is not an additional input to the neural network $\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t,t)$. Importantly, the two objectives differ only by a constant $C\le 0$ independent of $\boldsymbol{\phi}$:

$$\mathcal{L}_{\mathrm{FM}}(\boldsymbol{\phi}) = \mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) + C.$$

They therefore have the same population minimizer,

$$\mathbf{v}^*(\mathbf{x}_t,t) = \mathbf{v}_t(\mathbf{x}_t). \tag{5.2.11}$$

In other words, although each training example uses a conditional velocity $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ as its target, averaging over these examples recovers the desired marginal velocity field.

For CFM to provide tractable, simulation-free training, two practical conditions are needed:

(i) we can easily sample $\mathbf{x}_t \sim p_t(\cdot|\mathbf{z})$ without solving an ODE;

(ii) the conditional velocity $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ has a tractable expression that can be used as the regression target.

We provide concrete constructions satisfying these conditions in Section 5.3.2. Thus, just as denoising score matching learns an intractable marginal score through tractable conditional scores, Flow Matching learns an intractable marginal velocity through tractable conditional-velocity targets.

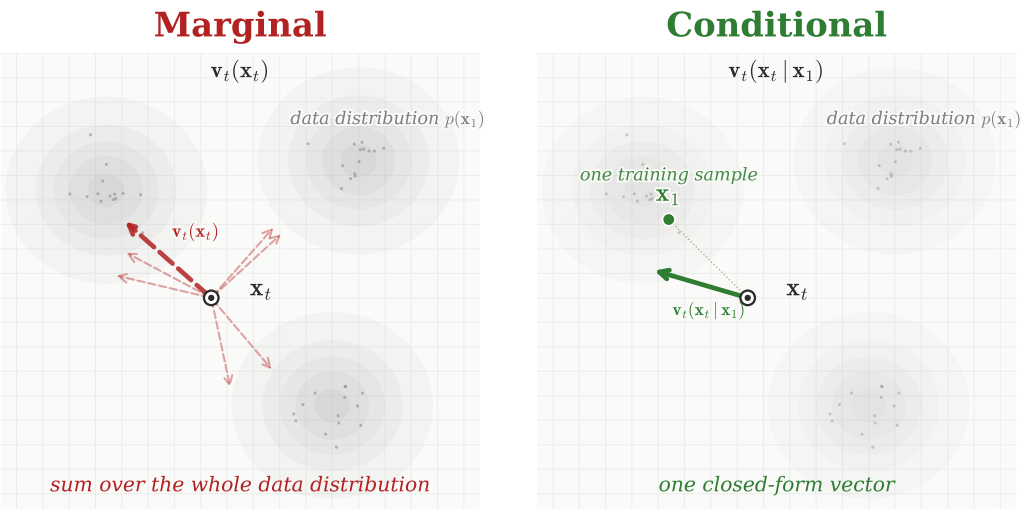


Figure 5.8: The conditional trick in flow matching. Here, $\mathbf{x}_1 \sim p_{\text{tgt}}$ denotes a target data sample. **(Left)** For a fixed intermediate sample $\mathbf{x}_t$, the marginal velocity $\mathbf{v}_t(\mathbf{x}_t)$ averages the conditional velocities associated with all possible target samples $\mathbf{x}_1$ that could have produced $\mathbf{x}_t$. The contribution of each $\mathbf{x}_1$ is weighted by how compatible it is with the observed $\mathbf{x}_t$. Computing this average directly requires integrating over the whole target distribution and is generally intractable. **(Right)** During training, however, the particular target sample $\mathbf{x}_1$ used to construct $\mathbf{x}_t$ is known. Conditioning on this sample gives the closed-form velocity $\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_1) = b_t'\mathbf{x}_1 + \frac{a_t'}{a_t}\big(\mathbf{x}_t - b_t\mathbf{x}_1\big)$, which provides one simple vector as the regression target (Proposition 5.3.1). Although each training example conditions on only one target sample, averaging the conditional training objective over the data gives the same optimal marginal velocity field; see Theorem 5.2.1. Thus, conditional flow matching learns $\mathbf{v}_t(\mathbf{x}_t)$ without directly regressing on the generally intractable marginal velocity. This is the same conditioning principle used in the variational perspective (Theorem 2.2.1) and in denoising score matching (Theorem 3.3.1).

Source: Created by the authors with AI-assisted coding.

We summarize the above discussion as follows:

Theorem 5.2.1: Equivalence of $\mathcal{L}_{\mathrm{FM}}$ and $\mathcal{L}_{\mathrm{CFM}}$

The following holds:

$$\mathcal{L}_{\mathrm{FM}}(\boldsymbol{\phi}) = \mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) + C,$$

where C is a constant independent of the parameter $\boldsymbol{\phi}$. Furthermore, the minimizer $\mathbf{v}^*$ of both losses satisfies

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbf{v}_t(\mathbf{x}_t), \quad \text{for almost every } \mathbf{x}_t \sim p_t,$$

where $\mathbf{v}_t(\mathbf{x}_t)$ is defined in Equation (5.2.9).

Proof for Theorem.

The argument and derivation of the minimizer follows exactly the same reasoning as in the score matching case of Proposition 4.3.1.

Proof. Write $\mathbf{v}_{\boldsymbol{\phi}} = \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$, and abbreviate the marginal and conditional expectations by

$$\mathbb{E}_{\mathrm{m}} := \mathbb{E}_{t,\, \mathbf{x}_t \sim p_t}, \qquad \mathbb{E}_{\mathrm{c}} := \mathbb{E}_{t,\, \mathbf{z}\sim\pi,\, \mathbf{x}_t \sim p_t(\cdot|\mathbf{z})}.$$

Expanding the squared norms,

$$\begin{aligned}
\mathcal{L}_{\mathrm{FM}}(\boldsymbol{\phi}) &= \mathbb{E}_{\mathrm{m}}\big[\|\mathbf{v}_{\boldsymbol{\phi}}\|^2\big] - 2\,\mathbb{E}_{\mathrm{m}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t)\big] + \mathbb{E}_{\mathrm{m}}\big[\|\mathbf{v}_t(\mathbf{x}_t)\|^2\big], \\
\mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) &= \mathbb{E}_{\mathrm{c}}\big[\|\mathbf{v}_{\boldsymbol{\phi}}\|^2\big] - 2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + \mathbb{E}_{\mathrm{c}}\big[\|\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|^2\big].
\end{aligned}$$

The first terms are equal, because for any function h of $(\mathbf{x}_t, t)$ alone,

$$\begin{aligned}
\mathbb{E}_{\mathrm{c}}[h] &= \mathbb{E}_t\Big[\iint p_t(\mathbf{x}_t|\mathbf{z})\,\pi(\mathbf{z})\,h(\mathbf{x}_t,t)\;\mathrm{d}\mathbf{z}\;\mathrm{d}\mathbf{x}_t\Big] \\
&\overset{(5.2.7)}{=} \mathbb{E}_t\Big[\int p_t(\mathbf{x}_t)\,h(\mathbf{x}_t,t)\;\mathrm{d}\mathbf{x}_t\Big] = \mathbb{E}_{\mathrm{m}}[h].
\end{aligned}$$

The third terms do not involve $\boldsymbol{\phi}$, so $C := \mathbb{E}_{\mathrm{m}}[\|\mathbf{v}_t(\mathbf{x}_t)\|^2] - \mathbb{E}_{\mathrm{c}}[\|\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|^2]$ is a constant. Hence

$$\begin{aligned}
\mathcal{L}_{\mathrm{FM}}(\boldsymbol{\phi}) - \mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) &= \cancel{\mathbb{E}_{\mathrm{m}}\big[\|\mathbf{v}_{\boldsymbol{\phi}}\|^2\big]} - 2\,\mathbb{E}_{\mathrm{m}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t)\big] \\
&\quad - \cancel{\mathbb{E}_{\mathrm{c}}\big[\|\mathbf{v}_{\boldsymbol{\phi}}\|^2\big]} + 2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + C \\
&\overset{(5.2.9)}{=} -2\,\mathbb{E}_t\Big[\int p_t(\mathbf{x}_t)\,\mathbf{v}_{\boldsymbol{\phi}}^\top \int p(\mathbf{z}|\mathbf{x}_t)\,\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\;\mathrm{d}\mathbf{z}\;\mathrm{d}\mathbf{x}_t\Big] \\
&\quad + 2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + C \\
&\overset{\text{Bayes}}{=} -2\,\mathbb{E}_t\Big[\iint p_t(\mathbf{x}_t|\mathbf{z})\,\pi(\mathbf{z})\,\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\;\mathrm{d}\mathbf{z}\;\mathrm{d}\mathbf{x}_t\Big] \\
&\quad + 2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + C \\
&= -2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + 2\,\mathbb{E}_{\mathrm{c}}\big[\mathbf{v}_{\boldsymbol{\phi}}^\top \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] + C = C,
\end{aligned}$$

where Bayes' rule reads $p_t(\mathbf{x}_t)\, p(\mathbf{z}|\mathbf{x}_t) = p_t(\mathbf{x}_t|\mathbf{z})\, \pi(\mathbf{z})$.
Minimizer. Rewriting the CFM weighting with the same Bayes' rule,

$$\begin{aligned}\mathcal{L}_{\mathrm{CFM}}(\boldsymbol{\phi}) &= \mathbb{E}_t\Big[\iint \pi(\mathbf{z})\, p_t(\mathbf{x}_t|\mathbf{z})\, \|\mathbf{v}_{\boldsymbol{\phi}} - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|^2 \, \mathrm{d}\mathbf{z}\, \mathrm{d}\mathbf{x}_t\Big] \\ &\overset{\text{Bayes}}{=} \mathbb{E}_t\Big[\int p_t(\mathbf{x}_t) \int p(\mathbf{z}|\mathbf{x}_t)\, \|\mathbf{v}_{\boldsymbol{\phi}} - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|^2 \, \mathrm{d}\mathbf{z}\, \mathrm{d}\mathbf{x}_t\Big] \\ &= \mathbb{E}_{\mathrm{m}}\Big[\mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x}_t)} \big\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big\|^2\Big].\end{aligned}$$

For each fixed $(\mathbf{x}_t, t)$ the inner term is a mean-squared error in $\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$, minimized by the conditional mean of the target. Hence

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x}_t)}\big[\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\big] \overset{(5.2.9)}{=} \mathbf{v}_t(\mathbf{x}_t);$$

the outer expectation over $(t, \mathbf{x}_t)$ drops because we condition on them. ■

This marks the third instance where the conditioning trick yields a tractable training objective, as illustrated in Figure 5.8. Notably, the variational, score based, and flow based approaches all reflect the same underlying principle.

Remark.

Taking $\pi = p_{\mathrm{data}}$, we can apply Bayes' rule:

$$p(\mathbf{x}_0|\mathbf{x}_t) = \frac{p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\mathrm{data}}(\mathbf{x}_0)}{p_t(\mathbf{x}_t)},$$

a similar decomposition of Equation (5.2.9) appears in score-based models:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) &= \mathbb{E}_{\mathbf{x}_0\sim p(\cdot|\mathbf{x}_t)}\left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\right] \\ &= \mathbb{E}_{\mathbf{x}_0\sim p_{\mathrm{data}}}\left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0) \cdot \frac{p_t(\mathbf{x}_t|\mathbf{x}_0)}{p_t(\mathbf{x}_t)}\right],\end{aligned}$$

which mirrors the marginalization strategy in Equation (5.2.9).

As in Section 5.2.1, where the conditional density $p_t(\mathbf{x}_t|\mathbf{z})$ and conditional velocity field $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ must be explicitly specified, with $p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ and $\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0) = f(t)\mathbf{x}_t - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)$, the general conditional flow matching framework also requires these two components. However, we have not yet construct the conditional density $p_t(\mathbf{x}_t|\mathbf{z})$ or the conditional velocity field $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ in this general case. In Section 5.3, we introduce several common instantiations of these components.

5.2.3 Comparison of Diffusion Models, General Flow Matching, and NODEs

Comparison of Diffusion Models and General Flow Matching. The insight from Section 5.2.1 leads to an extended FM framework that retains the same underlying principles. To highlight their similarities, we summarize them in Table 5.1.

Table 5.1: Comparison between diffusion models (or Gaussian FM) and the general FM framework. Here, the general FM framework refers to the setting with two-sided conditioning, where $\mathbf{x}_0 \sim p_{\text{src}}$ and $\mathbf{x}_1 \sim p_{\text{tgt}}$ are sampled independently.

Aspect	(Score-Based) Diffusion Model	General FM
Source dist. p_{src}	Gaussian prior	Any
Target dist. p_{tgt}	Data distribution	Any
Latent dist. $\pi(\mathbf{z})$	p_{data}	See Section 5.3.2
Cond. dist. $p_t(\mathbf{x}_t\|\mathbf{z})$	$\mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$	See Section 5.3.2
Marginal dist. $p_t(\mathbf{x}_t)$	$\int p_t(\mathbf{x}_t\|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0$	$\int p_t(\mathbf{x}_t\|\mathbf{z})\pi(\mathbf{z})\, \mathrm{d}\mathbf{z}$
Cond. velocity $\mathbf{v}_t(\mathbf{x}\|\mathbf{z})$	$f(t)\mathbf{x} - \frac{1}{2} g^2(t) \nabla \log p_t(\mathbf{x}\|\mathbf{x}_0)$	See Section 5.3.2
Marginal velocity $\mathbf{v}_t(\mathbf{x})$	$f(t)\mathbf{x} - \frac{1}{2} g^2(t) \nabla \log p_t(\mathbf{x})$	See Equation (5.2.9)
Learning objective	$\mathcal{L}_{\text{SM}} = \mathcal{L}_{\text{DSM}} + C$	$\mathcal{L}_{\text{FM}} = \mathcal{L}_{\text{CFM}} + C$
Underlying Rule	Fokker-Planck / Continuity Equation	

We remark that, assuming the Gaussian schedule satisfies the diffusion-SDE compatibility condition in Lemma 4.5.1, Gaussian FM is essentially equivalent to the corresponding diffusion formulation (see more in Chapter 6); we therefore do not differentiate between them unless explicitly stated.

Connection to NODEs. FM can be viewed as a simulation-free alternative to NODEs, introduced in Section 5.1.2. While CNFs require solving ODEs during maximum likelihood training, which is computationally intensive, FM bypasses this by directly regressing a prescribed velocity field through a simple regression loss. The key insight is that when the marginal density path connecting the source and target distributions is fixed, exact simulation during training becomes unnecessary.

5.2.4 (Optional) Underlying Rules

Continuity Equation: Mass Conservation Criterion. Similar to the PF-ODE and Fokker–Planck analysis in Section 5.2.1, we now present a criterion for verifying whether the density path induced by an ODE flow aligns with a prescribed path $\{p_t\}_{t\in[0,1]}$.

Consider the ODE describing the flow of particles under a time-dependent velocity field $\mathbf{v}_t$:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_t\left(\mathbf{x}(t)\right).$$

As in Equation (5.2.4), this ODE defines a flow map $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_0)$ for any $s, t \in [0,1]$, which in particular transports an initial point $\mathbf{x}_0 \sim p_{\text{src}}$ at time 0 to its state at time t. The induced distribution at time t is given by the pushforward

$$p_t^{\text{fwd}}(\mathbf{x}) = \int \delta\left(\mathbf{x} - \boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0)\right) p_{\text{src}}(\mathbf{x}_0) \mathrm{d}\mathbf{x}_0 =: \boldsymbol{\Psi}_{0\to t} \# p_{\text{src}}, \tag{5.2.12}$$

so that $\mathbf{\Psi}_{0\to t}(\mathbf{x}_0) \sim p_t^{\text{fwd}}$ whenever $\mathbf{x}_0 \sim p_{\text{src}}$. Similarly, one can transport backward from $\mathbf{x}_1 \sim p_{\text{tgt}}$ to p_{src} via $\mathbf{\Psi}_{1\to 0}(\mathbf{x}_1)$.

Suppose we are given a prescribed density path $\{p_t\}_{t\in[0,1]}$, and we construct a velocity field $\{\mathbf{v}_t\}_{t\in[0,1]}$ to define a particle flow. This naturally raises the question:

Question 5.2.1

Under what conditions does the flow-induced density p_t^{fwd} exactly match the target density p_t for all $t \in [0,1]$?

Once the two density evolutions align, we can leverage the ODE flow to flexibly transport samples between p_{src} and p_{tgt} by solving the ODE.

As in Equation (5.2.5), a principled way to verify this alignment is via the *continuity equation*, which captures the conservation of mass in time-evolving densities:

Theorem 5.2.2: Mass Conservation Criterion

The flow-induced density p_t^{fwd} equals the prescribed path p_t for all $t \in [0,1]$; i.e.,

$$p_t^{\text{fwd}} = p_t, \quad \text{for all } t \in [0,1],$$

if and only if the pair $(p_t, \mathbf{v}_t)$ satisfies the continuity equation:

$$\partial_t p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x})\mathbf{v}_t(\mathbf{x})) = 0,$$

for all $t \in [0,1]$ and $\mathbf{x}$.

Proof for Theorem.

A conceptual derivation is provided in Section D.3.2, while a more rigorous treatment can be found in (Villani *et al.*, 2008) (see "Mass Conservation Formula"). The derivations of the continuity equation in Chapter B prove only the sufficient direction, flow ⇒ continuity equation; the converse is proved in Section D.3.2. ∎

From Conditional to Marginal Paths. As seen in Section 5.2.2, we begin by defining a conditional probability path $p_t(\cdot|\mathbf{z})$ and a corresponding conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$. We then construct the marginal velocity field via:

$$\mathbf{v}_t(\mathbf{x}) = \int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) \frac{p_t(\mathbf{x}|\mathbf{z})\pi(\mathbf{z})}{p_t(\mathbf{x})} \, \mathrm{d}\mathbf{z},$$

as in Equation (5.2.9). However, we still need to ensure that the resulting marginal velocity $\mathbf{v}_t$ induces an ODE flow whose density path aligns with the prescribed p_t. Fortunately, this verification can be done entirely at the conditional level: if each conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$ induces the conditional density path $p_t(\cdot|\mathbf{z})$, then

the resulting marginal velocity $\mathbf{v}_t$ also induces the correct marginal path. Formally, we state the result below and provide the complete derivation in one place to show explicitly how the marginal velocity field takes this form:

Proposition 5.2.3: Marginal VF Generates Given Marginal Density

If the conditional velocity fields $\mathbf{v}_t(\cdot|\mathbf{z})$ induce conditional density paths that match $p_t(\cdot|\mathbf{z})$ (starting from $p_0(\cdot|\mathbf{z})$), then the marginal velocity field $\mathbf{v}_t(\cdot)$ defined in Equation (5.2.9) induces a marginal density path that aligns with $p_t(\cdot)$, starting from $p_0(\cdot)$.

Proof for Proposition.

This result follows by verifying that the pair $(p_t, \mathbf{v}_t)$ satisfies the Continuity Equation. We present the argument in a converse manner to provide intuition for why the marginalized velocity field takes the form in Equation (5.2.9). Since the conditional velocity fields $\mathbf{v}_t(\cdot|\mathbf{z})$ induce density paths matching the conditional densities $p_t(\cdot|\mathbf{z})$ for $\mathbf{z} \sim \pi$, the continuity equation holds for each conditional pair:

$$\frac{\partial}{\partial t} p_t(\mathbf{x}|\mathbf{z}) = -\nabla_{\mathbf{x}} \cdot \big(\mathbf{v}_t(\mathbf{x}|\mathbf{z}) p_t(\mathbf{x}|\mathbf{z})\big). \tag{5.2.13}$$

We aim to find a velocity field $\mathbf{v}_t(\cdot)$ whose induced densities align with the marginal density p_t, i.e., satisfy

$$\frac{\partial}{\partial t} p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \big(\mathbf{v}_t(\mathbf{x}) p_t(\mathbf{x})\big). \tag{5.2.14}$$

Starting from the definition of p_t in Equation (5.2.7),

$$\begin{aligned}
\frac{\partial}{\partial t} p_t(\mathbf{x}) &= \int \frac{\partial}{\partial t} p_t(\mathbf{x}|\mathbf{z}) \pi(\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&= -\int \nabla_{\mathbf{x}} \cdot \Big(\mathbf{v}_t(\mathbf{x}|\mathbf{z}) p_t(\mathbf{x}|\mathbf{z})\Big) \pi(\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&= -\nabla_{\mathbf{x}} \cdot \Big(\int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) p_t(\mathbf{x}|\mathbf{z}) \pi(\mathbf{z})\, \mathrm{d}\mathbf{z} \Big),
\end{aligned}$$

where the second equality follows by applying Equation (5.2.13). Comparing this with the right-hand side of Equation (5.2.14) shows that, up to a divergence-free term,

$$\mathbf{v}_t(\mathbf{x}) p_t(\mathbf{x}) = \int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) p_t(\mathbf{x}|\mathbf{z}) \pi(\mathbf{z})\, \mathrm{d}\mathbf{z}.$$

Therefore, we can define

$$\mathbf{v}_t(\mathbf{x}) := \int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) \frac{p_t(\mathbf{x}|\mathbf{z})}{p_t(\mathbf{x})} \pi(\mathbf{z})\, \mathrm{d}\mathbf{z},$$

which is precisely the form in Equation (5.2.9). The proof of this theorem essentially follows the reverse of this argument.

This connection allows us to reduce the construction of the potentially intractable marginal velocity field to defining simpler conditional fields $\mathbf{v}_t(\cdot|\mathbf{z})$, which are easier to work with by construction.

5.3 Constructing Probability Paths and Velocities Between Distributions

The essence of flow matching lies in the gradual transformation of a source distribution into a target. To direct this transformation, two key elements are needed: the *probability path* p_t, which provides a snapshot of the evolving distribution at each time t, and the *velocity field* $\mathbf{v}_t$, which describes how individual particles move along the path. These two objects are not independent; they are linked through the continuity equation, which ensures that particle dynamics are consistent with the evolution of the distribution. Thus, the learning task reduces to finding a velocity field $\mathbf{v}_t$ that faithfully drives the process. The difficulty, however, is that for general and complex distributions, the true marginal velocity $\mathbf{v}_t$ is unknown, leaving us with an intractable target that cannot be accessed directly.

The core idea of Conditional Flow Matching is to address the intractability of the true marginal velocity by constructing an artificial but tractable process. To do this, we introduce a conditioning variable $\mathbf{z}$ and design either a conditional velocity $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ and/or a conditional path $p_t(\mathbf{x}_t|\mathbf{z})$, which are deliberately chosen to be simple.

Because these conditional objects are known in closed form, they serve as surrogate targets that the model can regress against. This leads to a valid training loss $\mathcal{L}_{\text{CFM}}$, provided two practical requirements are met: (i) we can sample efficiently from $p_t(\cdot|\mathbf{z})$, and (ii) the corresponding velocity $\mathbf{v}_t(\cdot|\mathbf{z})$ admits a closed-form expression.

How should we design a well behaved conditional process? For inspiration, we turn to the one case that is fully understood: the Gaussian to Gaussian bridge (Section 5.3.1). This example highlights two natural design strategies: adopt a Gaussian probability path at each time t, or prescribe an affine velocity field, both of which are analytically tractable.

Guided by this insight, we extend to general endpoint distributions using two complementary construction views (see also Section B.1.2). These correspond to the classical Eulerian and Lagrangian descriptions of transport:

- **Conditional Probability-Path First (Eulerian View).** We begin with the conditional density path $p_t(\cdot|\mathbf{z})$ and seek a conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$ satisfying the continuity equation. Since a density path alone generally does not determine a unique velocity, this construction also requires choosing a particular compatible velocity field.

- **Conditional Flow First (Lagrangian View).** We instead specify how individual particles move through a conditional flow map $\mathbf{\Psi}_{0\to t}(\cdot|\mathbf{z})$. The induced density is obtained by pushforward, and, when the flow map is invertible, the corresponding Eulerian velocity field is

$$\mathbf{v}_t = \partial_t \mathbf{\Psi}_{0\to t} \circ \mathbf{\Psi}_{0\to t}^{-1}.$$

In Section 5.3.2, we detail the first approach, which shows its close analogy to diffusion model construction discussed in Section 5.2.1, while in Section 5.3.3 we present the second. Together, these perspectives provide a practical framework for defining $p_t(\mathbf{x}_t|\mathbf{z})$ and $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$, enabling simulation-free training and the construction of flows between arbitrary source and target distributions.

5.3.1 A Key Special Case: Marginal $p_t(\mathrm{x}_t)$ and Velocity $\mathrm{v}_t(\mathrm{x}_t)$ in the Gaussian-to-Gaussian Bridge

We begin with the Gaussian–endpoint case, where we can compute the marginal density $p_t(\mathbf{x}_t)$ and velocity field $\mathbf{v}_t(\mathbf{x}_t)$ analytically. This serves as a template for the general construction of the conditional density $p_t(\mathbf{x}_t|\mathbf{z}_t)$ and velocity field $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}_t)$.

When the source and target distributions, p_{src} and p_{tgt}, are both Gaussian, the velocity field $\mathbf{v}_t(\cdot)$ admits a closed-form expression. We consider the interpolated marginal density path:

$$p_t(\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_t; \boldsymbol{\mu}(t), \sigma^2(t)\mathbf{I}\right), \tag{5.3.1}$$

with time-varying mean $\boldsymbol{\mu}(t)$ and variance $\sigma^2(t) > 0$. The two endpoints are given by

$$p_{\text{src}} = p_0 = \mathcal{N}\left(\mathbf{x}; \boldsymbol{\mu}(0), \sigma^2(0)\mathbf{I}\right), \quad p_{\text{tgt}} = p_1 = \mathcal{N}\left(\mathbf{x}; \boldsymbol{\mu}(1), \sigma^2(1)\mathbf{I}\right),$$

so that the path $\{p_t\}_{t\in[0,1]}$ connects these distributions.

With the given path $\{p_t\}_{t\in[0,1]}$, there are indeed many velocity fields that induce an ODE flow $\boldsymbol{\Psi}_{0\to t}(\mathbf{x})$ such that $\mathbf{x} \sim p_0$ implies $\boldsymbol{\Psi}_{0\to t}(\mathbf{x}) \sim p_t$. For this Gaussian path, a particularly simple realization is given by[7]:

$$\boldsymbol{\Psi}_{0\to t}(\mathbf{x}) := \boldsymbol{\mu}(t) + \sigma(t)\left(\frac{\mathbf{x} - \boldsymbol{\mu}(0)}{\sigma(0)}\right). \tag{5.3.2}$$

Proof that Equation (5.3.2) realizes Equation (5.3.1). Since $\boldsymbol{\Psi}_{0\to t}$ is affine, it maps the Gaussian p_0 to a Gaussian, so it suffices to match the mean and covariance.

[7] In (Lipman *et al.*, 2022), the authors consider $\boldsymbol{\Psi}_{0\to t}(\mathbf{x}) = \boldsymbol{\mu}(t) + \sigma(t)\mathbf{x}$, which requires constraints on $\boldsymbol{\mu}(t)$ and $\sigma(t)$ to ensure boundary conditions. We adopt an equivalent normalized formulation that avoids such constraints.

For $\mathbf{x} \sim p_0$,

$$\begin{aligned}\mathbb{E}\left[\boldsymbol{\Psi}_{0\to t}(\mathbf{x})\right] &= \mathbb{E}\Big[\boldsymbol{\mu}(t) + \sigma(t)\,\frac{\mathbf{x}-\boldsymbol{\mu}(0)}{\sigma(0)}\Big] = \boldsymbol{\mu}(t) + \frac{\sigma(t)}{\sigma(0)}\left(\mathbb{E}[\mathbf{x}] - \boldsymbol{\mu}(0)\right)\\ &= \boldsymbol{\mu}(t) + \frac{\sigma(t)}{\sigma(0)}\left(\boldsymbol{\mu}(0) - \boldsymbol{\mu}(0)\right) = \boldsymbol{\mu}(t),\\ \mathrm{Cov}\left[\boldsymbol{\Psi}_{0\to t}(\mathbf{x})\right] &= \mathrm{Cov}\left[\boldsymbol{\mu}(t) + \frac{\sigma(t)}{\sigma(0)}(\mathbf{x}-\boldsymbol{\mu}(0))\right] = \frac{\sigma^2(t)}{\sigma^2(0)}\,\mathrm{Cov}[\mathbf{x}]\\ &= \frac{\sigma^2(t)}{\sigma^2(0)}\,\sigma^2(0)\mathbf{I} = \sigma^2(t)\mathbf{I},\end{aligned}$$

so $\boldsymbol{\Psi}_{0\to t}(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}(t), \sigma^2(t)\mathbf{I}) = p_t$. ■

For the defined Gaussian path p_t (Gaussian for all t), the velocity field $\mathbf{v}_t(\cdot)$ inducing the ODE flow in Equation (5.3.2) is uniquely and analytically characterized as follows (Lipman *et al.*, 2022):

Proposition 5.3.1: Closed-Form Velocity Field for Gaussian Density Path

Let p_t be the Gaussian path in Equation (5.3.1). Then the velocity field $\mathbf{v}_t(\cdot)$ that generates the ODE flow Equation (5.3.2) is unique for the defined $\boldsymbol{\Psi}_{0\to t}$ and has the closed-form expression:

$$\mathbf{v}_t(\mathbf{x}) = \frac{\sigma'(t)}{\sigma(t)}\left(\mathbf{x} - \boldsymbol{\mu}(t)\right) + \boldsymbol{\mu}'(t).$$

Proof for Proposition.

Write $\mathbf{x}_0$ for the initial point and $\mathbf{x}_t := \boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0)$ for the current point of the trajectory. By definition of the flow,

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0) = \mathbf{v}_t\left(\boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0)\right) = \mathbf{v}_t(\mathbf{x}_t).$$

Differentiating Equation (5.3.2) with respect to t gives

$$\boldsymbol{\Psi}'_{0\to t}(\mathbf{x}_0) = \boldsymbol{\mu}'(t) + \sigma'(t)\,\frac{\mathbf{x}_0 - \boldsymbol{\mu}(0)}{\sigma(0)}.$$

This is expressed in terms of the initial point $\mathbf{x}_0$, whereas a velocity field must be a function of the current point $\mathbf{x}_t$. Since $\boldsymbol{\Psi}_{0\to t}$ is invertible (as $\sigma(t) > 0$), we can trade one for the other through $\mathbf{x}_0 = \boldsymbol{\Psi}^{-1}_{0\to t}(\mathbf{x}_t) = \boldsymbol{\Psi}_{t\to 0}(\mathbf{x}_t)$, so that

$$\mathbf{v}_t(\mathbf{x}_t) = \boldsymbol{\Psi}'_{0\to t}\left(\boldsymbol{\Psi}_{t\to 0}(\mathbf{x}_t)\right).$$

Solving Equation (5.3.2) for $\mathbf{x}_0 = \boldsymbol{\Psi}_{t\to 0}(\mathbf{x}_t)$ yields

$$\mathbf{x}_0 = \boldsymbol{\mu}(0) + \sigma(0)\,\frac{\mathbf{x}_t - \boldsymbol{\mu}(t)}{\sigma(t)},$$

and substituting this into $\boldsymbol{\Psi}'_{0\to t}(\mathbf{x}_0)$ gives

$$\mathbf{v}_t(\mathbf{x}_t) = \boldsymbol{\mu}'(t) + \sigma'(t)\,\frac{\mathbf{x}_t - \boldsymbol{\mu}(t)}{\sigma(t)} = \frac{\sigma'(t)}{\sigma(t)}\,(\mathbf{x}_t - \boldsymbol{\mu}(t)) + \boldsymbol{\mu}'(t),$$

as claimed.

We note that for a fixed flow map $\boldsymbol{\Psi}_{0\to t}$ (flow-first view), the velocity is uniquely determined by

$$\mathbf{v}_t = \partial_t \boldsymbol{\Psi}_{0\to t} \circ \boldsymbol{\Psi}^{-1}_{0\to t}.$$

Under this construction, the pair $(p_t, \mathbf{v}_t)$ automatically satisfies the continuity equation. By contrast, for a given density path $t \mapsto p_t$ without fixing $\boldsymbol{\Psi}_{0\to t}$ (probability-path-first view), the velocity field is not unique.

This distinction precisely characterizes the difference between the flow-first and probability-path-first perspectives.

This closed-form characterization remains valid when conditioning on a latent variable $\mathbf{z}$. In the following, we extend this insight to construct a conditional Gaussian path $p_t(\cdot|\mathbf{z})$ and derive the corresponding conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$ for the general marginal setting.

5.3.2 Conditional Probability-Path-First Construction of $\mathbf{v}_t(\cdot|\mathbf{z})$ and $p_t(\cdot|\mathbf{z})$

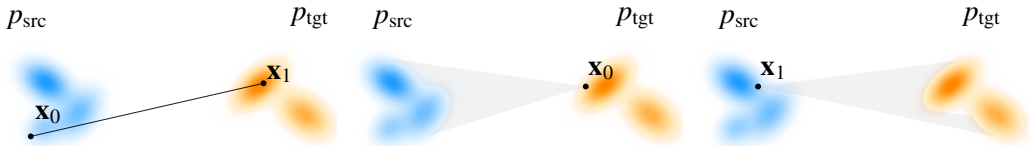


Figure 5.9: Illustrations of two common types of conditioning probability paths. It includes: (i) *two-sided*, conditioned on $\mathbf{x}_0 \sim p_{\text{tgt}}$ and $\mathbf{x}_1 \sim p_{\text{src}}$ with general endpoint distributions; (ii) *one-sided*, conditioned at either $\mathbf{x}_0 \sim p_{\text{tgt}}$ or $\mathbf{x}_1 \sim p_{\text{src}}$.

Source: Created by the authors.

We aim to construct a conditional density path $p_t(\cdot|\mathbf{z})$ first and then derive its corresponding conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$ (via Proposition 5.3.1), under conditioning with respect to $\pi(\mathbf{z})$. Depending on how $\mathbf{z}$ is chosen, there are two natural scenarios: (i) two-sided conditioning with $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$, or (ii) one-sided conditioning with $\mathbf{z} = \mathbf{x}_0$ or $\mathbf{x}_1$. In either case, the construction must match the boundary distributions:

$$p_{\text{src}}(\mathbf{x}_0) = \int p_0(\mathbf{x}_0|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}, \quad p_{\text{tgt}}(\mathbf{x}_1) = \int p_1(\mathbf{x}_1|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}.$$

Since verifying these constraints is straightforward once a concrete construction is specified, we do not emphasize the verification step here.

I. Two-Sided $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$ — "Beam-Like" Path.

Choice of $\pi(\mathbf{z})$. Consider general distributions p_{src} and p_{tgt} over $\mathbb{R}^D$. Let $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$ with $\mathbf{x}_0 \sim p_{\text{src}}$ and $\mathbf{x}_1 \sim p_{\text{tgt}}$ independently, i.e.,

$$\pi(\mathbf{z}) = p_{\text{src}}(\mathbf{x}_0) p_{\text{tgt}}(\mathbf{x}_1).$$

Choice of Conditional Path $p_t(\cdot|\mathbf{z})$. Define the conditional path by linear interpolation with a fixed and small variance $\sigma > 0$[8]:

$$p_t(\mathbf{x}_t|\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)) = \mathcal{N}(\mathbf{x}_t; a_t\mathbf{x}_0 + b_t\mathbf{x}_1, \sigma^2\mathbf{I}),$$

where a_t and b_t are time-dependent functions satisfying $a_0 = 1$, $b_0 = 0$ and $a_1 = 0$, $b_1 = 1$. A choice suggested by (Lipman *et al.*, 2022; Liu, 2022) is $a_t = 1 - t$, $b_t = t$. In the deterministic case $\sigma = 0$, we obtain

$$p_t(\mathbf{x}_t|\mathbf{z}) = \delta(\mathbf{x}_t - [a_t\mathbf{x}_0 + b_t\mathbf{x}_1]),$$

which describes a deterministic interpolating path from $\mathbf{x}_0$ to $\mathbf{x}_1$.

Derived Conditional Velocity $\mathbf{v}_t(\cdot|\mathbf{z})$. By Proposition 5.3.1, the conditional velocity is

$$\begin{aligned}\mathbf{v}_t(\mathbf{x}|\mathbf{z}) &= \frac{\sigma'}{\sigma}(\mathbf{x} - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_t' \\ &= \boldsymbol{\mu}_t' = a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1,\end{aligned}$$

where $\boldsymbol{\mu}_t := a_t\mathbf{x}_0 + b_t\mathbf{x}_1$ is the conditional mean and $\sigma' = 0$ because the conditional variance is constant in time.

CFM Loss. When $\sigma = 0$ so that $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$, the CFM loss reduces to

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,\mathbf{x}_0\sim p_{\text{src}},\mathbf{x}_1\sim p_{\text{tgt}}} \left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - (a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1)\right\|^2.$$

From Equations (5.2.9) and (5.2.11), the optimal velocity field is

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E}\left[\mathbf{x}_t'|\mathbf{x}_t\right] = \mathbb{E}\left[a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1|\mathbf{x}_t\right].$$

Here, the expectation is taken over $p(\mathbf{x}_0, \mathbf{x}_1|\mathbf{x}_t)$, the conditional distribution over source-target pairs $(\mathbf{x}_0, \mathbf{x}_1)$ that could have produced the observed interpolation $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$ at time t.

[8] One may instead add a fixed conditional variance $\sigma^2\mathbf{I}$. The conditional velocity remains $a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1$ because the variance is constant in time, but the endpoint marginals then become Gaussian smoothings of p_{src} and p_{tgt}, approaching the desired endpoints as $\sigma \to 0$.

II. One-Sided $\mathbf{z} = \mathbf{x}_0$ or $\mathbf{x}_1$ — "Spotlight-Like" Path. We illustrate the conditional probability–path–first construction in the one-sided setting, considering the standard generative setup with $p_{\text{src}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $p_{\text{tgt}} = p_{\text{data}}$. Crucially, this Gaussian source is not an additional assumption but a direct consequence of the conditional path defined below. A more general treatment of arbitrary endpoints will be given in Section 5.3.3.

Choice of $\pi(\mathbf{z})$. We take $\mathbf{z} = \mathbf{x}_1$ with $\pi(\mathbf{z}) = p_{\text{data}}(\mathbf{x}_1)$, or $\mathbf{z} = \mathbf{x}_0$ with $\pi(\mathbf{z}) = p_{\text{prior}}(\mathbf{x}_0)$, and derive both cases below.

Choice of Conditional Path $p_t(\cdot|\mathbf{z})$ for $\mathbf{z} = \mathbf{x}_1$. For fixed $\mathbf{x}_1 \sim p_{\text{data}}$, define

$$p_t(\mathbf{x}_t|\mathbf{z} = \mathbf{x}_1) = \mathcal{N}(\mathbf{x}_t; b_t\mathbf{x}_1, a_t^2\mathbf{I}),$$

with $a_0 = 1, b_0 = 0, a_1 = 0, b_1 = 1$ (usually interpreted as the limit). At the boundaries,

$$p_0(\cdot|\mathbf{z} = \mathbf{x}_1) = \mathcal{N}(\cdot; \mathbf{0}, \mathbf{I}), \qquad p_1(\cdot|\mathbf{z} = \mathbf{x}_1) = \delta(\cdot - \mathbf{x}_1).$$

Marginalizing over $\mathbf{x}_1$ yields $\{p_t\}_{t\in[0,1]}$ with $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ (independent of p_{data}) and $p_1 = p_{\text{data}}$.

Derived Conditional Velocity $\mathbf{v}_t(\cdot|\mathbf{z})$ for $\mathbf{z} = \mathbf{x}_1$. For $t \in (0, 1)$ with $a_t > 0$, applying Proposition 5.3.1 to the conditional Gaussian path gives

$$\mathbf{v}_t(\mathbf{x}|\mathbf{x}_1) = b_t'\mathbf{x}_1 + \frac{a_t'}{a_t}(\mathbf{x} - b_t\mathbf{x}_1).$$

One-Sided CFM Objective for $\mathbf{z} = \mathbf{x}_1$. With $t \sim \mathcal{U}(0, 1)$ (or any fixed sampling distribution) and $\mathbf{x}_1 \sim p_{\text{data}}$, the CFM loss becomes

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,\mathbf{x}_1}\mathbb{E}_{\mathbf{x}_t\sim p_t(\cdot|\mathbf{x}_1)}\left\|\mathbf{v}_\phi(\mathbf{x}_t, t) - \left[b_t'\mathbf{x}_1 + \tfrac{a_t'}{a_t}(\mathbf{x}_t - b_t\mathbf{x}_1)\right]\right\|_2^2. \tag{5.3.3}$$

Equivalence to Two-Sided Target for $\mathbf{z} = \mathbf{x}_1$. For paired samples $(\mathbf{x}_0, \mathbf{x}_1)$ with $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$,

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_1) = b_t'\mathbf{x}_1 + \tfrac{a_t'}{a_t}(\mathbf{x}_t - b_t\mathbf{x}_1) = a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1.$$

Here $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$ is the reparameterization of $\mathbf{x}_t \sim \mathcal{N}(b_t\mathbf{x}_1, a_t^2\mathbf{I})$: write $\mathbf{x}_t = b_t\mathbf{x}_1 + a_t\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and identify $\boldsymbol{\epsilon}$ with $\mathbf{x}_0 \sim p_{\text{src}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Conditional Path $p_t(\cdot|\mathbf{z})$ for $\mathbf{z} = \mathbf{x}_0$. Here $\pi(\mathbf{z}) = p_{\text{prior}}(\mathbf{x}_0)$, and the path is induced by the interpolation $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$ with $\mathbf{x}_1 \sim p_{\text{data}}$ independent of $\mathbf{x}_0$. Fix $\mathbf{x}_0$ and integrate over $\mathbf{x}_1$. For $b_t > 0$,

$$\begin{aligned} p_t(\mathbf{x}|\mathbf{x}_0) &= \int \delta(\mathbf{x} - a_t\mathbf{x}_0 - b_t\mathbf{x}_1)\, p_{\text{data}}(\mathbf{x}_1)\, \mathrm{d}\mathbf{x}_1 \\ &\overset{\delta \text{ scaling}}{=} \frac{1}{b_t^D} \int \delta\Big(\mathbf{x}_1 - \frac{\mathbf{x} - a_t\mathbf{x}_0}{b_t}\Big)\, p_{\text{data}}(\mathbf{x}_1)\, \mathrm{d}\mathbf{x}_1 \\ &\overset{\text{sifting}}{=} \boxed{\frac{1}{b_t^D}\, p_{\text{data}}\Big(\frac{\mathbf{x} - a_t\mathbf{x}_0}{b_t}\Big)}. \end{aligned}$$

Here $\mathbf{x}_0$ is the only conditioning variable; $\mathbf{x}_1$ has been integrated out. The conditional distribution is a scaled and shifted copy of the data distribution. At the endpoints,

$$p_0(\mathbf{x}|\mathbf{x}_0) = \delta(\mathbf{x} - \mathbf{x}_0), \qquad p_1(\mathbf{x}|\mathbf{x}_0) = p_{\text{data}}(\mathbf{x}),$$

where the first is read off directly from $\mathbf{x}_t = \mathbf{x}_0$ at $t = 0$, since the formula above requires $b_t > 0$. Marginalizing over $\mathbf{x}_0 \sim p_{\text{prior}}$ therefore gives $p_0 = p_{\text{prior}}$ and $p_1 = p_{\text{data}}$. Unlike the case $\mathbf{z} = \mathbf{x}_1$, the source here is the chosen π rather than a consequence of the path, so p_{prior} need not be Gaussian.

Derived Conditional Velocity $\mathbf{v}_t(\cdot|\mathbf{z})$ for $\mathbf{z} = \mathbf{x}_0$. Differentiate the interpolation while keeping the sampled endpoints fixed:

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1.$$

Substitute $\mathbf{x}_1 = (\mathbf{x}_t - a_t\mathbf{x}_0)/b_t$:

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = a_t'\mathbf{x}_0 + \frac{b_t'}{b_t}(\mathbf{x}_t - a_t\mathbf{x}_0).$$

Thus the conditional velocity field is

$$\boxed{\mathbf{v}_t(\mathbf{x}|\mathbf{x}_0) = a_t'\mathbf{x}_0 + \frac{b_t'}{b_t}(\mathbf{x} - a_t\mathbf{x}_0).}$$

At a sampled point $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$,

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0) = a_t'\mathbf{x}_0 + \frac{b_t'}{b_t}(\mathbf{x}_t - a_t\mathbf{x}_0) = a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1.$$

Shared Two-Sided Target. For either choice of $\mathbf{z}$, the conditional velocity at a sampled point equals $a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1$, so with $t \sim \mathcal{U}(0, 1)$ the objective becomes

$$\boxed{\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1} \big\| \mathbf{v}_\phi(\mathbf{x}_t, t) - (a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1) \big\|_2^2.}$$

Thus the one-sided loss regresses to the *conditional expectation* of the two-sided target given $\mathbf{x}_t$:

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}\big[a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1 | \mathbf{x}_t = \mathbf{x}\big],$$

so the one-sided and two-sided CFM objectives share the same minimizer.

Gaussian FM = Diffusion Model. We use the FM convention where $t = 0$ denotes the source/prior and $t = 1$ denotes the target/data:

$$p_{\text{src}} = p_{\text{prior}}, \qquad p_{\text{tgt}} = p_{\text{data}}.$$

By contrast, diffusion models typically index time from data to noise (i.e., $t = 0$ is data and $t = 1$ is prior). Here, we adopt the FM convention indexing in the main discussion. For comparison, however, the VE and VP entries in Table 5.2 are summarized in their usual diffusion-time indexing, where $t = 0$ corresponds to data and $t = 1$ to prior/noise. If further $p_{\text{src}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$, then for fixed condition $\mathbf{x}_1 \sim p_{\text{data}}$, the conditional path $p_t(\cdot|\mathbf{x}_1)$ is naturally chosen to be Gaussian, while the target distribution p_{tgt} itself need not be Gaussian. We refer to this Gaussian conditional-path setting as *Gaussian FM*. Diffusion probability paths form an important subclass of such Gaussian paths. A generic Gaussian schedule, however, need not correspond to a valid forward diffusion SDE; the schedule must additionally satisfy the SDE compatibility condition in Lemma 4.5.1.

Choosing $a_t = 1 - t$ and $b_t = t$ (equivalently, $\alpha_t = t$ and $\sigma_t = 1 - t$ under the relabeling $a_t := \sigma_t$, $b_t := \alpha_t$ in diffusion model) recovers the familiar FM/RF schedule (Lipman *et al.*, 2022; Liu, 2022).

Table 5.2: Summary of different interpolants written as $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$, where $\mathbf{x}_0 \sim p_{\text{src}} = p_{\text{prior}}$ and $\mathbf{x}_1 \sim p_{\text{tgt}} = p_{\text{data}}$. The FM/RF and trigonometric columns use the FM convention ($t = 0$ source, $t = 1$ target), whereas the VE/VP columns retain the usual diffusion-time convention ($t = 0$ data, $t = 1$ prior/noise), with $a_t := \sigma_t$ and $b_t := \alpha_t$. For VE at a finite terminal noise level, the exact terminal marginal is $p_{\text{data}} * \mathcal{N}(\mathbf{0}, a_1^2\mathbf{I})$; the Gaussian prior shown in the table is the usual approximation when the terminal noise dominates the scale of the data.

	VE	**VP**	**FM/RF**	**Trig.** (Albergo *et al.*, 2023)
a_t (prior coeff.)	a_t	$\sqrt{1 - b_t^2}$	$1 - t$	$\cos\left(\frac{\pi}{2}t\right)$
b_t (data coeff.)	1	b_t	t	$\sin\left(\frac{\pi}{2}t\right)$
a_0	0	0	1	1
b_0	1	1	0	0
a_1	a_1	1	0	0
b_1	1	0	1	1
p_{prior}	$p_{\text{data}} * \mathcal{N}(\mathbf{0}, a_1^2\mathbf{I}) \approx \mathcal{N}(\mathbf{0}, a_1^2\mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$

In the Gaussian FM setting, both the beam-like and spotlight-like conditional paths lead to training objectives that are similar to the standard diffusion losses. As we will elaborate in Chapter 6, Gaussian FM can in fact be equivalently interpreted as a diffusion model trained to predict the *velocity*, under the linear schedule

$a_t = 1 - t$ and $b_t = t$. This perspective highlights that flow matching and diffusion are not fundamentally different, but rather two equivalent formulations that can be transformed into one another. The Gaussian FM objective is particularly appealing in practice: its loss function ($\mathbb{E}_{t,\mathbf{x}_t}\left[\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|_2^2\right]$) is simple, and it has been shown to achieve competitive performance at scale (Esser *et al.*, 2024).

5.3.3 Conditional Flow-First Construction of $\mathbf{v}_t(\cdot|\mathbf{z})$ and $p_t(\cdot|\mathbf{z})$

We treat the general case where the endpoints p_{src} (at $t = 0$) and p_{tgt} (at $t = 1$) are arbitrary. Our goal is to design, directly in trajectory space, a conditional flow that transports samples from p_{src} to p_{tgt} and yields a closed-form $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ usable as a regression target.

Motivation. Instead of first designing conditional density path, we may directly specify a conditional flow map $\boldsymbol{\Psi}_{0\to t}(\cdot;\mathbf{z})$ that moves samples along trajectories. This has two practical advantages: (i) it immediately yields a regression target for training via a time derivative along trajectories; (ii) on geometry-structured spaces (Riemannian manifolds, Lie groups, or constrained submanifolds), it is often natural to construct the conditional flow map $\boldsymbol{\Psi}_{0\to t}$ directly from the geometry (e.g., geodesics, exponential maps, or premetrics) (Lipman *et al.*, 2024) which yields analytic, simulation-free target velocities for training.

Conditional Affine Flow (Link to Proposition 5.3.1). We fix a conditioning variable $\mathbf{z} \sim \pi$ (e.g., $\mathbf{z} = \mathbf{x}_1 \sim p_{\text{tgt}}$ in one-sided "spotlight" training) and push forward $\mathbf{x}_0 \sim p_{\text{src}}$ through the time-varying *conditional affine flow*

$$\boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0;\mathbf{z}) := \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0, \qquad t \in [0,1],$$

where $\boldsymbol{\mu}_t(\mathbf{z}) \in \mathbb{R}^D$ and $\mathbf{A}_t(\mathbf{z}) \in \mathbb{R}^{D\times D}$ is invertible for $t \in (0,1)$. The boundary $\mathbf{A}_0(\mathbf{z}) = \mathbf{I}$, $\boldsymbol{\mu}_0(\mathbf{z}) = \mathbf{0}$ recovers p_{src} at $t = 0$. It is standard to interpret boundary when $t \to 1$ as a limit (the terminal map may concentrate mass on a lower-dimensional set or a point)[9].

Induced Conditional Path $p_t(\cdot|\mathbf{z})$. The construction defines

$$p_t(\cdot|\mathbf{z}) = \left(\boldsymbol{\Psi}_{0\to t}(\cdot;\mathbf{z})\right)_{\#} p_{\text{src}}, \qquad p_t(\cdot) = \int p_t(\cdot|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}.$$

What ultimately matters in $\mathcal{L}_{\text{CFM}}$ is how to sample from it: first draw $\mathbf{z} \sim \pi$, then draw $\mathbf{x}_0 \sim p_{\text{src}}$, and finally set

$$\mathbf{x}_t = \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0.$$

[9] Allowing $\mathbf{A}_1(\mathbf{z})$ to be singular (e.g., $\mathbf{0}$) is compatible with invertibility on $(0,1)$ and causes the path to contract onto the prescribed endpoint at $t = 1$.

We remark that when $\mathbf{\Psi}_{0\to t}$ is affine in $\mathbf{x}_0$, then $p_t(\cdot|\mathbf{z})$ is Gaussian if and only if p_{src} is Gaussian. In particular, for arbitrary (non-Gaussian) p_{src}, an affine flow yields a generally non-Gaussian $p_t(\cdot|\mathbf{z})$.

Derived Conditional Velocity $\mathbf{v}_t(\cdot|\mathbf{z})$. The conditional velocity $\mathbf{v}_t(\cdot|\mathbf{z})$ is obtained by t-differentiating the conditional flow map $\mathbf{\Psi}_{0\to t}$. Following the derivation in Proposition 5.3.1, consider the *conditional* ODE defined by the flow map $\mathbf{\Psi}_{0\to t}(\mathbf{y};\mathbf{z})$ with initial condition $\mathbf{y}$, where the goal is to identify the corresponding conditional velocity field $\mathbf{v}_t(\cdot|\mathbf{z})$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{\Psi}_{0\to t}(\mathbf{y};\mathbf{z}) = \mathbf{v}_t\Big(\underbrace{\mathbf{\Psi}_{0\to t}(\mathbf{y};\mathbf{z})}_{\mathbf{x}}\Big|\mathbf{z}\Big).$$

Since $\mathbf{\Psi}_{0\to t}(\cdot;\mathbf{z})$ is invertible for $t \in (0,1)$, we may express the initial point $\mathbf{y}$ corresponding to the current state $\mathbf{x} = \mathbf{\Psi}_{0\to t}(\mathbf{y};\mathbf{z})$ as

$$\mathbf{y} = \mathbf{\Psi}^{-1}_{0\to t}(\mathbf{x};\mathbf{z}) = \mathbf{\Psi}_{t\to 0}(\mathbf{x};\mathbf{z}).$$

The corresponding Eulerian velocity field is therefore

$$\mathbf{v}_t(\mathbf{x}|\mathbf{z}) := \frac{\partial}{\partial t}\mathbf{\Psi}_{0\to t}(\mathbf{y};\mathbf{z})\Big|_{\mathbf{y}=\mathbf{\Psi}^{-1}_{0\to t}(\mathbf{x};\mathbf{z})}.$$

That is, we first differentiate the forward flow map with respect to t while holding its initial point $\mathbf{y}$ fixed, and then substitute the initial point whose trajectory reaches $\mathbf{x}$ at time t.

Since $\mathbf{x}_t = \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0$ and $\mathbf{A}_t(\mathbf{z})$ is invertible on $(0,1)$, we have $\mathbf{x}_0 = \mathbf{A}_t(\mathbf{z})^{-1}(\mathbf{x} - \boldsymbol{\mu}_t(\mathbf{z}))$, giving

$$\mathbf{v}_t(\mathbf{x}|\mathbf{z}) = \boldsymbol{\mu}_t'(\mathbf{z}) + \mathbf{A}_t'(\mathbf{z})\mathbf{A}_t(\mathbf{z})^{-1}(\mathbf{x} - \boldsymbol{\mu}_t(\mathbf{z})).$$

One-Sided Conditioning ($\mathbf{z} = \mathbf{x}_1$). Choosing $\boldsymbol{\mu}_t(\mathbf{z}) = b_t\mathbf{z}$ and $\mathbf{A}_t(\mathbf{z}) = a_t\mathbf{I}$ with $a_0 = 1, a_1 = 0$ and $b_0 = 0, b_1 = 1$ (with $a_t > 0$ for $t \in (0,1)$) yields

$$\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1, \qquad \mathbf{v}_t(\mathbf{x}|\mathbf{x}_1) = b_t'\mathbf{x}_1 + \frac{a_t'}{a_t}(\mathbf{x} - b_t\mathbf{x}_1).$$

On paired samples $(\mathbf{x}_0, \mathbf{x}_1)$ (with $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$), this simplifies to the usual CFM target:

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_1) = a_t'\mathbf{x}_0 + b_t'\mathbf{x}_1.$$

Two-Sided Conditioning ($\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$). The same template with $\boldsymbol{\mu}_t(\mathbf{x}_0, \mathbf{x}_1) = b_t\mathbf{x}_1$ and $\mathbf{A}_t(\mathbf{x}_0, \mathbf{x}_1) = a_t\mathbf{I}$ makes the conditional path *deterministic*:

$$\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1, \qquad p_t(\cdot|\mathbf{x}_0, \mathbf{x}_1) = \delta\left(\cdot - (a_t\mathbf{x}_0 + b_t\mathbf{x}_1)\right),$$

and the conditional velocity is

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1) = a'_t\mathbf{x}_0 + b'_t\mathbf{x}_1,$$

i.e., the standard two-sided CFM target.

Unconditional Gaussian Path as a Special Case. If $\boldsymbol{\mu}_t$ is independent of $\mathbf{z}$ (denoted $\boldsymbol{\mu}(t)$) and $\mathbf{A}_t = \frac{\sigma(t)}{\sigma(0)}\mathbf{I}$, then

$$\boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0) = \boldsymbol{\mu}(t) + \sigma(t)\frac{\mathbf{x}_0 - \boldsymbol{\mu}(0)}{\sigma(0)}, \quad \mathbf{v}_t(\mathbf{x}) = \boldsymbol{\mu}'(t) + \frac{\sigma'(t)}{\sigma(t)}(\mathbf{x} - \boldsymbol{\mu}(t)),$$

which recovers the Gaussian density path and the closed-form velocity in Proposition 5.3.1.

5.3.4 Probability-Path-First vs. Flow-First Construction

The two constructions correspond to complementary Eulerian and Lagrangian descriptions of transport. The *probability-path-first* (Eulerian) view begins by specifying a conditional density path $p_t(\cdot|\mathbf{z})$ and then selects a compatible velocity field through the continuity equation. Because a density path alone generally does not determine the velocity uniquely, additional transport structure may be needed to select one.

The *flow-first* (Lagrangian) view instead specifies a conditional flow map $\boldsymbol{\Psi}_{0\to t}(\cdot|\mathbf{z})$, which directly describes particle trajectories. The corresponding density path follows by pushforward, and when the map is invertible its Eulerian velocity is uniquely recovered as

$$\mathbf{v}_t = \partial_t\boldsymbol{\Psi}_{0\to t} \circ \boldsymbol{\Psi}^{-1}_{0\to t}.$$

Both constructions therefore describe the same kinds of transport objects from different starting points, but a prescribed density path need not select the same velocity as a separately prescribed flow map.

The following table summarizes these contrasts. The takeaway: path-first is natural when conditional paths admit closed-form velocities; flow-first is natural when you have strong structural priors on trajectories.

5.3.5 Conditional Straightness Does Not Imply Marginal-Flow Straightness

The canonical affine interpolation

$$\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\mathbf{x}_1$$

has an appealing geometric property: conditioned on a particular endpoint pair $(\mathbf{x}_0, \mathbf{x}_1)$, the interpolation is a straight line with constant velocity $\mathbf{x}_1 - \mathbf{x}_0$. This

Table 5.3: Comparison of conditional probability-path-first and conditional flow-first constructions.

Axis	Conditional Probability-Path-First	Conditional Flow-First
Given	Conditional density path $p_t(\cdot\|\mathbf{z})$.	Conditional flow map $\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z})$ (trajectories, for each fixed $\mathbf{z}$).
Get Velocity	For each $\mathbf{z}$, find $\mathbf{v}_t(\cdot\|\mathbf{z})$ s.t. $\partial_t p_t(\cdot\|\mathbf{z}) + \nabla\cdot(p_t(\cdot\|\mathbf{z})\mathbf{v}_t(\cdot\|\mathbf{z})) = 0$; Non-unique: if $\nabla\cdot(p_t\mathbf{w}_t) = 0$ then $\mathbf{v}_t + \mathbf{w}_t$ yields the same p_t.	Along paths (for each $\mathbf{z}$): $\mathbf{v}_t\left(\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z}) \mid \mathbf{z}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z})$. When $\mathbf{\Psi}_{0\to t}$ is invertible, one can solve $\mathbf{v}_t(\mathbf{x}\|\mathbf{z}) = \partial_t\mathbf{\Psi}_{0\to t}(\mathbf{y}\|\mathbf{z})\big\|_{\mathbf{y}=\mathbf{\Psi}_{0\to t}^{-1}(\mathbf{x}\|\mathbf{z})}$.
Closed Form of $\mathbf{v}_t(\cdot\|\mathbf{z})$	Convenient when $p_t(\cdot\|\mathbf{z})$ is Gaussian / exponential-family; otherwise obtaining $\mathbf{v}_t(\cdot\|\mathbf{z})$ is nontrivial.	Convenient when $\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z})$ has structure (affine/low-rank); avoids density evaluation.
Uniqueness of $\mathbf{v}_t(\cdot\|\mathbf{z})$	For each $\mathbf{z}$, $\mathbf{v}_t(\cdot\|\mathbf{z})$ is underdetermined unless a selection rule (e.g., potential flow / min. kinetic energy) is imposed.	Given $\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z})$, both $p_t(\cdot\|\mathbf{z}) = (\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z}))_{\#}p_0$ and the particle trajectories are determined. If the flow map is invertible, the corresponding Eulerian velocity field is uniquely determined; otherwise, a unique global Eulerian field need not exist.
Realizability	Must verify the constructed $\mathbf{v}_t(\cdot\|\mathbf{z})$ solving the continuity equation on the intended support.	Holds by construction: $p_t(\cdot\|\mathbf{z}) = (\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z}))_{\#}\,p_0(\cdot\|\mathbf{z})$.
Match $(p_{\mathrm{src}}, p_{\mathrm{tgt}})$	Mix conditionals: $p_{\mathrm{src}} = \int p_0(\cdot\|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}$, $p_{\mathrm{tgt}} = \int p_1(\cdot\|\mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z}$. Under Gaussian–affine conditional paths with $\mathbf{z}$-independent coefficients, p_{src} can be forced to be Gaussian. For a general fixed endpoint p_{src} (possibly non-Gaussian), the choice of $p_t(\cdot\|\mathbf{z})$ does not generally pin p_{src}.	Set $\mathbf{\Psi}_{0\to 0} = \mathrm{Id}$ and choose boundary condition to hit any p_{tgt}.
Preferred Scenarios	Diffusion-style constructions; analytic targets via conditional Gaussians $p_t(\cdot\|\mathbf{z})$.	Strong structural priors via maps $\mathbf{\Psi}_{0\to t}(\cdot\|\mathbf{z})$; easy boundary control; accommodates singular/low-dimensional endpoints; natural for map-based regularization/transport costs.

conditional notion of straightness is used extensively in Flow Matching and Rectified Flow.

It is important, however, to distinguish the straightness of these *conditional interpolation paths* from the straightness of the *marginal ODE trajectories* used at generation time. Under a coupling $\pi(\mathbf{x}_0, \mathbf{x}_1)$, the population-optimal marginal velocity is

$$\mathbf{v}^\star(\mathbf{x}, t) = \mathbb{E}\left[\mathbf{x}_1 - \mathbf{x}_0 \,\middle|\, \mathbf{x}_t = \mathbf{x}\right].$$

The posterior over endpoint pairs generally changes with $(\mathbf{x}, t)$. Consequently, the velocity encountered along a marginal ODE trajectory need not remain constant, even though every conditional interpolation used for training is a straight line.

Thus, choosing the canonical affine schedule alone does not imply that the learned marginal ODE is exactly straight or that one Euler step is exact. Rectified

Flow addresses precisely this distinction: its rectification or `Reflow` procedure changes the endpoint coupling so that the induced causal flow can become progressively closer to a straight flow (Liu, Gong, *et al.*, 2022).

The following example makes the distinction explicit.

Example: A Simple 1D Counterexample

Consider the one-dimensional setting with

$$x_0 \sim \mathcal{N}(0, \sigma_0^2), \qquad x_1 \sim \mathcal{N}(0, \sigma_1^2),$$

independent coupling, and the canonical affine interpolation $x_t = (1-t)x_0 + tx_1$. Each conditional path $(1-t)x_0 + tx_1$ is a straight line in time. We now compute the induced marginal velocity field.

Since $(x_1 - x_0,\, x_t)$ is jointly Gaussian, the conditional expectation is linear in x, so

$$v^*(x,t) = \mathbb{E}[x_1 - x_0 | x_t = x] = \frac{\mathrm{Cov}(x_1 - x_0,\, x_t)}{\mathrm{Var}(x_t)}\, x.$$

Proof that the Gaussian conditional expectation is linear. Let (A, B) be jointly Gaussian with $\mathbb{E}[A] = \mathbb{E}[B] = 0$ and $\mathrm{Var}(B) > 0$.
Step 1: split off a leftover. Define

$$c := \frac{\mathrm{Cov}(A,B)}{\mathrm{Var}(B)}, \qquad R := A - cB,$$

so that $A = cB + R$ as an identity; R contains whatever cB does not capture.
Step 2: the leftover is uncorrelated with B.

$$\mathrm{Cov}(R,B) = \mathrm{Cov}(A,B) - c\,\mathrm{Var}(B) = \mathrm{Cov}(A,B) - \frac{\mathrm{Cov}(A,B)}{\mathrm{Var}(B)}\,\mathrm{Var}(B) = 0.$$

Step 3: the pair (R,B) is jointly Gaussian. A random vector Z is jointly Gaussian if every linear combination $u^\top Z$ is Gaussian. For any matrix M, $u^\top(MZ) = (M^\top u)^\top Z$ is Gaussian by that definition, so MZ is jointly Gaussian. Apply this to

$$\begin{pmatrix} R \\ B \end{pmatrix} = \begin{pmatrix} 1 & -c \\ 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix}.$$

Step 4: independence. Jointly Gaussian variables with zero covariance are independent, so R and B are independent. Also $\mathbb{E}[R] = \mathbb{E}[A] - c\,\mathbb{E}[B] = 0$, hence independence gives $\mathbb{E}[R \mid B = x] = 0$.
Step 5: take the conditional expectation.

$$\mathbb{E}[A \mid B = x] = c\,\mathbb{E}[B \mid B = x] + \mathbb{E}[R \mid B = x] = cx + 0 = \frac{\mathrm{Cov}(A,B)}{\mathrm{Var}(B)}\, x,$$

where $\mathbb{E}[B \mid B = x] = x$ because we condition on B taking that value.

For the flow example, substitute $A = x_1 - x_0$ and $B = x_t$:

$$v^*(x,t) = \mathbb{E}[x_1 - x_0 \mid x_t = x] = \frac{\mathrm{Cov}(x_1 - x_0, x_t)}{\mathrm{Var}(x_t)}\, x.$$

The coefficient depends on t but not on x, so the conditional expectation is linear in x. ■

A direct calculation gives

$$\mathrm{Var}(x_t) = (1-t)^2\sigma_0^2 + t^2\sigma_1^2 =: V(t),$$

and

$$\mathrm{Cov}(x_1 - x_0,\, x_t) = t\sigma_1^2 - (1-t)\sigma_0^2 = \tfrac{1}{2}V'(t).$$

Hence

$$v^*(x,t) = \frac{V'(t)}{2V(t)}\, x.$$

The marginal ODE

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = v^*(z_t, t)$$

therefore has the exact solution

$$z_t = z_0\sqrt{\frac{V(t)}{V(0)}} = z_0\sqrt{(1-t)^2 + \frac{\sigma_1^2}{\sigma_0^2}\, t^2}.$$

Now, a straight-line interpolation in time would require z_t to be affine in t, namely $z_t = (1-t)z_0 + t\, z_1$ for some endpoint z_1. But

$$\sqrt{(1-t)^2 + \frac{\sigma_1^2}{\sigma_0^2}\, t^2}$$

is not affine in t for any positive σ_0, σ_1. Thus the ODE trajectory is generally not a straight-line interpolation in time. We illustrate this phenomenon in Section 5.3.5 with the example $\sigma_0 = 2$ and $\sigma_1 = 3$.

For example, when $\sigma_0 = \sigma_1$, we have $z_1 = z_0$, so a straight-line interpolation would simply be the constant path $z_t = z_0$. In contrast, the actual ODE trajectory satisfies

$$z_{1/2} = \frac{z_0}{\sqrt{2}} \neq z_0.$$

Therefore, even though every conditional interpolation path is straight, the induced marginal ODE trajectory need not be. ■

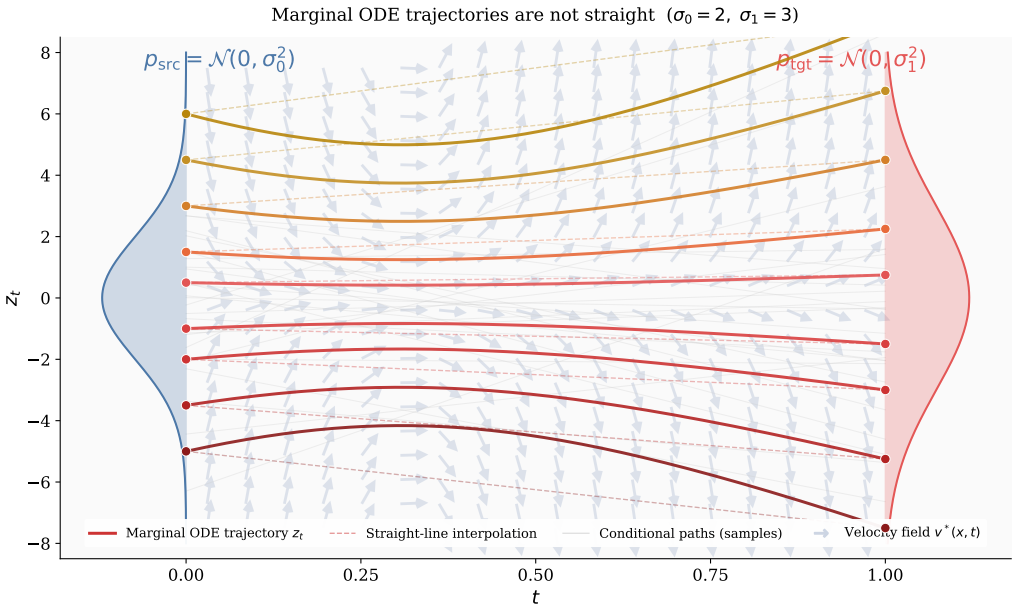


Figure 5.10: Marginal ODE trajectories are not straight under the canonical affine interpolation. We take $x_0 \sim \mathcal{N}(0, \sigma_0^2)$ and $x_1 \sim \mathcal{N}(0, \sigma_1^2)$ with $\sigma_0 = 2$, $\sigma_1 = 3$, and independent coupling. Gray lines show sampled conditional paths $(1-t)x_0 + tx_1$, which are straight by construction. Solid curves show the exact marginal ODE trajectories $z_t = z_0\sqrt{(1-t)^2 + (\sigma_1/\sigma_0)^2 t^2}$, and dashed lines show the corresponding straight-line interpolations from z_0 to z_1. The visible gap between solid and dashed curves confirms that the marginal ODE trajectories are not affine in t, even though every conditional path is a straight line. The marginal densities p_{src} and p_{tgt} are shown on the left and right margins, respectively.

Source: Created by the authors with AI-assisted coding.

5.4 (Optional) Properties of the Canonical Affine Flow

Given two endpoint distributions $p_0 = p_{\text{src}}$ and $p_1 = p_{\text{tgt}}$, a natural and widely used choice for defining the conditional path in flow matching (FM) (Lipman *et al.*, 2022) and rectified flow (RF) (Liu, Gong, *et al.*, 2022) is the linear interpolation

$$a_t = 1 - t, \qquad b_t = t,$$

which yields the interpolant

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1, \qquad \mathbf{x}_0 \sim p_{\text{src}},\ \mathbf{x}_1 \sim p_{\text{tgt}}.$$

Under this choice, the training objective simplifies to

$$\mathbb{E}_{t\sim\mathcal{U}[0,1]}\mathbb{E}_{\mathbf{x}_0,\mathbf{x}_1}\left[\left\|\mathbf{v}_\phi(\mathbf{x}_t,t) - (\mathbf{x}_1 - \mathbf{x}_0)\right\|_2^2\right].$$

This linear flow enjoys several appealing properties. In particular, it admits an iterative refinement scheme, known as ***Reflow***, which progressively straightens the path between distributions while preserving the marginals.

5.4.1 Rectifying Flows: From Noisy Guesses to Structured Pairings

Why Change the Coupling? Under the independent coupling

$$\pi(\mathbf{x}_0, \mathbf{x}_1) = p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1),$$

each conditional interpolation $(1-t)\mathbf{x}_0 + t\mathbf{x}_1$ is a straight line. As emphasized in Section 5.4, however, the corresponding marginal velocity averages directions from many unrelated endpoint pairs, so the induced marginal ODE trajectories need not be straight. Such curved marginal dynamics can be harder to integrate accurately with only a few numerical steps.

Rectification addresses this issue by changing the endpoint coupling. A learned ODE transports each source sample to a model-generated endpoint, producing a dependent coupling that is subsequently used to define a new conditional interpolation.

Rectify the Flow via Dependent Coupling. Rather than relying on arbitrary pairings, we use a pre-trained diffusion model $\mathbf{v}_{\phi^\times}(\cdot, t)$ as the drift in a PF-ODE to *deterministically* transport each source point. Starting from $\mathbf{z}(0) = \mathbf{z}_0 \sim p_{\text{src}}$, we integrate

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{v}_{\phi^\times}(\mathbf{z}(t), t), \qquad t \in [0,1],$$

to obtain $\hat{\mathbf{z}}_1 := \mathbf{z}(1)$ positioned near the data space learned from the pre-trained model. The resulting pair $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$ forms a *dependent coupling*: it follows a structured, model-guided path rather than an arbitrary interpolation. This idea extends

naturally to affine reference paths of the form $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$, where $\mathbf{x}_0 \sim p_{\text{src}}$ and $\mathbf{x}_1 \sim p_{\text{tgt}}$.

Algorithm 2 `Rectify` Operation

Input: Reference path $\{\mathbf{x}_t\}_{t\in[0,1]}$ (e.g. $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1$)

1: **Pre-Train Diffusion.** Fit $\mathbf{v}_{\phi^\times}$ on the chosen path by minimizing

$$\phi^\times \in \arg\min_{\phi} \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\left\|\mathbf{v}_\phi(\mathbf{x}_t,t) - \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t}\right\|_2^2\right].$$

2: **Rectify.** Sample $\mathbf{z}_0 \sim p_{\text{src}}$ and integrate

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{v}_{\phi^\times}(\mathbf{z}(t),t), \qquad \mathbf{z}(0) = \mathbf{z}_0, \quad t \in [0,1],$$

to obtain $\hat{\mathbf{z}}_1 = \mathbf{z}(1)$ and the trajectory $\{\mathbf{z}(t)\}_{t\in[0,1]}$.

Output: Dependent (coherent) pair $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$ or the full trajectory.

Why It Works: Marginal-Preserving Structure. Let $\mathbf{\Phi}_{0\to t}$ denote the flow map generated by the above ODE defined by the pre-trained diffusion $\mathbf{v}_{\phi^\times}$; then $\mathbf{z}(t) = \mathbf{\Phi}_{0\to t}(\mathbf{z}_0)$ and $\hat{\mathbf{z}}_1 = \mathbf{\Phi}_{0\to 1}(\mathbf{z}_0)$. The `Rectify` procedure pairs each source point with its flow endpoint, giving the deterministic joint

$$\pi_{\text{Rectify}}(\mathbf{z}_0,\mathbf{z}_1) = p_{\text{src}}(\mathbf{z}_0)\delta(\mathbf{z}_1 - \mathbf{\Phi}_{0\to 1}(\mathbf{z}_0)).$$

We have two immediate consequences:

- **Source Marginal is Preserved:** $\int \pi_{\text{Rectify}}(\mathbf{z}_0,\mathbf{z}_1)\,\mathrm{d}\mathbf{z}_1 = p_{\text{src}}(\mathbf{z}_0)$.
- **Pushforward Along the Flow:** $(\mathbf{\Phi}_{0\to t})_{\#}p_{\text{src}} = \text{Law}(\mathbf{z}(t))$, i.e., the time–$t$ distribution is the pushforward of p_{src} by $\mathbf{\Phi}_{0\to t}$.

If $\mathbf{v}_{\phi^\times}$ matches the oracle drift of a given reference path $\mathbf{x}_t$, then all intermediate marginals coincide:

$$\text{Law}(\mathbf{z}(t)) = \text{Law}(\mathbf{x}_t), \quad \text{for all } t \in [0,1], \quad \text{and} \quad (\mathbf{\Phi}_{0\to 1})_{\#}p_{\text{src}} = p_{\text{tgt}}.$$

Summary. Rectification replaces the independent endpoint coupling with a coupling induced by the learned flow. Under repeated `Reflow`, this can produce marginal transport that is progressively easier to integrate with few numerical steps.

For the canonical path, repeatedly applying `Rectify` (“*`Reflow`*”) further straightens trajectories without increasing transport cost, making training still easier.

5.4.2 `Reflow`: Iteratively Straightening Flows

Why `Reflow`? Independent pairings often induce irregular and meandering ODE trajectories between p_{src} and p_{tgt}, which increase discretization error and variance during simulation. This raises a natural question:

Question 5.4.1

Can we learn couplings that induce transport paths that are closer to straight lines between the two distributions, while still preserving the correct marginals?

This motivates ***`Reflow`***: repeatedly apply `Rectify` to update the coupling so that successive flows become easier to integrate.

Core Idea: Recursive Straightening via `Rectify`. Start from the canonical interpolation on the product coupling $\pi^{(0)} := p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$,

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1.$$

Applying `Rectify` replaces the independent pairing with a dependent one $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$ induced by the learned ODE. Repeating this procedure, known as ***`Reflow`***, tends to produce straighter trajectories that are easier to integrate accurately with a small number of numerical steps. The precise straightening guarantee is discussed later in Section 5.4.3.

Why `Reflow` straightens trajectories, and why curvature can remain.

- *ODE trajectories cannot collide at the same time.* Flow matching learns $\mathbf{v}_\phi(\mathbf{x}, t)$, defining $\dot{\mathbf{z}}_t = \mathbf{v}_\phi(\mathbf{z}_t, t)$. Under uniqueness of ODE solutions,

  $$\mathbf{z}_t^{(a)} = \mathbf{z}_t^{(b)} \implies \mathbf{z}_1^{(a)} = \mathbf{z}_1^{(b)}.$$

  Trajectories with different endpoints therefore cannot meet at the same $(\mathbf{x}, t)$. They may pass through the same spatial point at different times.

- ***`Reflow`*** *encourages straightness.* Use the current ODE's endpoint pairs $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$, then train on

  $$\mathbf{z}_t = (1-t)\mathbf{z}_0 + t\hat{\mathbf{z}}_1, \qquad \dot{\mathbf{z}}_t = \hat{\mathbf{z}}_1 - \mathbf{z}_0.$$

  This replaces curved journeys with constant-velocity, straight shortcuts as training targets.

- *Curvature can remain.* Non-intersecting curved routes can have straight shortcuts that intersect at the same time. Conflicting targets are averaged:

  $$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}\big[\hat{\mathbf{z}}_1 - \mathbf{z}_0 \,\big|\, \mathbf{z}_t = \mathbf{x}\big].$$

  This average can change along a generated trajectory, bending it. Straight training paths do not guarantee straight ODE trajectories.

The `Reflow` Procedure. Each iteration performs two steps:

- **Re-Fit Flow:** Train a new velocity field from samples of the current coupling:

$$\boldsymbol{\phi}_{k+1} = \arg\min_{\boldsymbol{\phi}} \mathcal{L}\left(\boldsymbol{\phi} \middle| \pi^{(k)}\right), \quad \text{where}$$
$$\mathcal{L}\left(\boldsymbol{\phi} \middle| \pi^{(k)}\right) := \mathbb{E}_{t,(\mathbf{z}_0^{(k)},\hat{\mathbf{z}}_1^{(k)})\sim\pi^{(k)}}\left[\left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{z}_t,t) - (\hat{\mathbf{z}}_1^{(k)} - \mathbf{z}_0^{(k)})\right\|^2\right] \tag{5.4.1}$$

  with $\mathbf{z}_t = (1-t)\mathbf{z}_0^{(k)} + t\hat{\mathbf{z}}_1^{(k)}$.

- **Generate New Coupling:** Solve the learned ODE starting from new source samples $\mathbf{z}_0^{(k+1)} \sim p_{\text{src}}$:

$$\hat{\mathbf{z}}_1^{(k+1)} \leftarrow \mathbf{z}_0^{(k+1)} + \int_0^1 \mathbf{v}_{\boldsymbol{\phi}_{k+1}}(\mathbf{z}(t),t)\mathrm{d}t,$$

  and define the updated coupling:

$$\pi^{(k+1)}(\mathbf{z}_0,\mathbf{z}_1) := p_{\text{src}}(\mathbf{z}_0)\delta\left(\mathbf{z}_1 - \hat{\mathbf{z}}_1^{(k+1)}\right).$$

In other words, `Reflow` can be viewed as repeatedly applying the `Rectify` operator, producing a sequence of progressively refined couplings:

$$\pi^{(k+1)} = \texttt{Rectify}\left(\pi^{(k)}\right) \tag{5.4.2}$$

so that both the flow and the coupling evolve together, yielding progressively more stable transport paths.

5.4.3 Properties of `Reflow`

Two key theoretical properties drive the usefulness of `Reflow`: it reduces transport cost and it straightens the trajectories.

I. `Reflow` Never Increases Transport Cost. Let $c(\mathbf{y})$ be a convex cost function (e.g., $\|\mathbf{y}\|_2^p$ with $p \geq 1$). Each `Rectify` step forms a new coupling $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$ whose cost is no worse than the original:

Proposition 5.4.1: `Rectify` May Reduce Transport Costs

Assuming an ideal velocity field $\mathbf{v}^* = \mathbf{v}_{\boldsymbol{\phi}^\times}$, we have:

$$\mathbb{E}\left[c\left(\hat{\mathbf{z}}_1 - \mathbf{z}_0\right)\right] \leq \mathbb{E}\left[c\left(\mathbf{x}_1 - \mathbf{x}_0\right)\right].$$

Proof for Proposition.

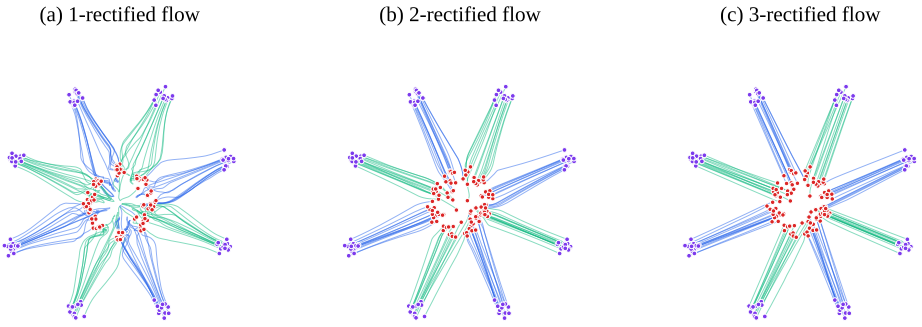


Figure 5.11: Illustration of `Reflow`. Paths become progressively straighter with `Rectify` procedure. *Source: Adapted from Liu, Gong,* et al. *(2022).*

Follows from Jensen's inequality. See Liu, Gong, *et al.* (2022) for a full derivation.

Applying this result recursively shows that the `Reflow` process does not increase the transport cost.

II. `Reflow` Straightens the Path. Recursive application of `Reflow` promotes increasingly straight trajectories, although the straightness measure is not guaranteed to improve monotonically at every iteration. To measure this, define the *straightness functional* of a path $\mathbf{Y} = \{\mathbf{y}_t\}_{t\in[0,1]}$ as

$$\mathcal{S}(\mathbf{Y}) := \int_0^1 \mathbb{E}\left[\left\|\mathbf{y}_1 - \mathbf{y}_0 - \frac{\mathrm{d}\mathbf{y}_t}{\mathrm{d}t}\right\|_2^2\right] \mathrm{d}t.$$

If $\mathcal{S}(\mathbf{Y}) = 0$, then $\mathbf{Y}$ is exactly a straight line.

Proposition 5.4.2: `Reflow` Straightens the Stochastic Path

For rectified paths $\mathbf{Z}^{(k)}$, we have:

$$\min_{k\in\{0,\ldots,K\}} \mathcal{S}(\mathbf{Z}^{(k)}) \leq \frac{\mathbb{E}\left[\|\mathbf{x}_1 - \mathbf{x}_0\|^2\right]}{K}.$$

Proof for Proposition.

See Theorem 3.7 of Liu, Gong, *et al.* (2022).

The FM or RF formulation with linear interpolation kernels, together with the `Reflow` procedure, provides a simpler training objective and a practical method for refining stochastic couplings. For theoretical details, we refer readers to (Liu, Gong, *et al.*, 2022; Liu, 2022).

III. Connection to Optimal Transport. Lastly, we note that straight-line couplings are not necessarily optimal in the sense of optimal transport (OT). This involves some terminology that will be introduced in Section 7.2; we therefore refer readers who are not familiar with OT to that section.

A hallmark of quadratic-cost optimal transport is that particles travel along straight lines: a particle at $\mathbf{x}_0$ moves to $\mathbf{T}(\mathbf{x}_0)$ via $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{T}(\mathbf{x}_0)$, where $\mathbf{T}$ is the optimal transport map. However, not every map $\mathbf{S}$ generating such straight-line paths, i.e., $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{S}(\mathbf{x}_0)$, is optimal. The map $\mathbf{S}$ yields the optimal flow only if it minimizes the Monge cost $\mathbb{E}[\|\mathbf{x}_0 - \mathbf{S}(\mathbf{x}_0)\|^2]$. Thus, while straight-line paths are necessary, they are not sufficient; optimality also depends on the correct endpoint map $\mathbf{T}$.

Example: Straight Couplings Need Not Be Optimal

Let $p_{\text{src}} = p_{\text{tgt}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$. For the cost $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^p$ with $p > 0$, the c-optimal coupling is the identity coupling π^*, where π^* is the law of $(\mathbf{x}, \mathbf{x})$ with $\mathbf{x} \sim p_{\text{src}}$.

Now consider the coupling $\pi_{\mathbf{A}}$ defined as the law of $(\mathbf{x}, \mathbf{A}\mathbf{x})$, where $\mathbf{x} \sim p_{\text{src}}$ and $\mathbf{A}$ is a rotation matrix satisfying $\mathbf{A}^\top \mathbf{A} = \mathbf{I}$, $\det(\mathbf{A}) = 1$, $\mathbf{A} \neq \mathbf{I}$, and -1 is not an eigenvalue. Then $\pi_{\mathbf{A}}$ is a valid coupling of p_{src} and p_{tgt}, and corresponds to straight-line paths between $\mathbf{x}$ and $\mathbf{A}\mathbf{x}$, but it is not c-optimal for any twice-differentiable strictly convex cost c with invertible Hessian. The suboptimality arises from the rotational transformation. As discussed in Equation (7.5.2), even removing the rotation may not lead to an optimal coupling. ■

We will continue exploring the connection to OT in Section 7.5.2.

5.5 Closing Remarks

This chapter has illuminated the third and final foundational perspective on diffusion models, one rooted in the principles of deterministic flows. Our exploration began with Normalizing Flows (NFs), which leverage the change-of-variables formula to learn an exact, invertible mapping between a simple prior and the data distribution. We then saw this concept evolve into a continuous-time process with Neural ODEs, where a learned velocity field governs the transformation. However, this approach comes with the significant drawback of requiring costly ODE simulations within the training loop.

The modern framework of Flow Matching (FM) was presented as an elegant and efficient solution to this challenge. By pre-defining a probability path $\{p_t\}_t$ and a corresponding velocity field that satisfies the continuity equation, FM establishes a clear target for the ODE flow. Crucially, just as we saw in the variational and score-based views, FM employs a powerful conditioning trick. This transforms the intractable problem of matching the marginal velocity field into a simple and tractable regression against a known conditional velocity, making training entirely simulation-free. This perspective recasts diffusion models themselves as a special case of learning a deterministic flow to transport a Gaussian prior to the data distribution.

With the introduction of the flow-based view, our survey of the three conceptual pillars of diffusion modeling is now complete. Throughout this journey, a remarkable pattern has emerged: each framework, despite its unique origins in VAEs, EBMs, or NFs, has converged on a continuous-time generative process and has relied on a conditioning strategy to enable tractable learning.

In the next chapter, we will finally synthesize these parallel threads into a single, unified framework. We will:

1. Formally demonstrate that the variational, score-based, and flow-based perspectives are not merely analogous but are mathematically equivalent at a fundamental level.

2. Show how the Fokker-Planck equation serves as the universal law governing density evolution across all three views, revealing that they are simply different lenses for describing the same core generative principle.

This unified lens will provide a complete and systematic understanding of the modern diffusion paradigm.

6

A Unified and Systematic Lens on Diffusion Models

Mathematics is the art of giving the same name to different things.

Henri Poincaré

This chapter presents a systematic viewpoint that connects the variational, score-based, and flow-based perspectives within a coherent picture. Although motivated by different intuitions, these approaches converge on the same core mechanism underlying modern diffusion models: define a forward corruption process that traces a path of marginal distributions, then learn the local quantities needed to construct generative dynamics that follow this path in reverse. Building on Chapters 2 to 5, we observe a common recipe: define a forward corruption process that traces a path of marginals, then learn a time varying vector field that transports a simple prior to the data distribution along this path.

A key ingredient across all perspectives is the conditioning trick introduced in Section 6.1, which transforms an intractable marginal objective into a tractable conditional one, leading to stable and efficient training.

In Section 6.2 we analyze the training objective in a systematic way, identifying its essential components and clarifying how loss functions are formulated in the variational, score-based, and flow-based viewpoints.

Section 6.3 shows that any affine forward noise injection of the form $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ can be equivalently transformed into the standard linear schedule $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon}$. Moreover, common parameterizations such as noise prediction, clean data prediction, score prediction, and velocity prediction are interchangeable at the level of gradients. Thus, the choices of noise schedulers and parameterizations both adhere to the same modeling principle.

Finally, Section 6.4 brings the discussion together and identifies the governing rule: the Fokker–Planck equation. Whether viewed as a variational scheme (discrete time denoising), a score-based method (SDE formulation), or a flow-based

method (ODE formulation), each constructs a generator whose marginals follow the same density evolution. The Fokker–Planck equation thus serves as the universal constraint respected by all three viewpoints, with differences arising only in parameterization and training objectives.

After this chapter, you will be able to...

- recognize the common conditioning principle behind variational, score-based, and flow-based training: an intractable marginal target can be learned through tractable conditional targets;
- understand the four common prediction types—clean data, noise, score, and velocity—and convert between their oracle predictions when the forward schedule is nondegenerate;
- disentangle the main design choices in diffusion training: the forward noise schedule, prediction target, time-sampling distribution, and time weighting, and understand which changes are mathematical reparameterizations and which can still affect practical optimization;
- distinguish different notions of "equivalence": representing the same oracle information, having equivalent population objectives up to weighting or additive constants, tracing the same marginal density path, and behaving identically as a practical training or sampling algorithm;
- understand how different affine probability paths are related through time reparameterization and spatial rescaling, while keeping track of what these transformations do and do not preserve;
- understand how the continuity and Fokker–Planck equations provide the density-level principle connecting variational, deterministic ODE, and stochastic SDE descriptions of diffusion models.

Reader route. This chapter is the conceptual keystone of the book. For a first pass, focus on the conditioning principle, the relationships among the four prediction types, and the continuity/Fokker–Planck view that connects the three perspectives. The central lesson is not to memorize every conversion formula, but to ask what notion of equivalence is being used and what that equivalence actually preserves. The detailed analysis of weighting, schedules, and parameterization choices can then be revisited to understand why mathematically equivalent formulations may still behave differently in practice.

6.1 Conditional Tricks: The Secret Sauce of Diffusion Models

Until now, we have explored diffusion models from three seemingly distinct origins: variational, score-based, and flow based perspectives. Each was originally motivated by different goals and led to its own training objectives (with a fixed t):

- **Variational View:** Learn a parametrized density $p_{\boldsymbol{\phi}}(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t)$ to approximate the oracle reverse transition $p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t)$ by minimizing:

$$\mathcal{J}_{\mathrm{KL}}(\boldsymbol{\phi}) := \mathbb{E}_{p_t(\mathbf{x}_t)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t)\|p_{\boldsymbol{\phi}}(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t))\right];$$

- **Score-Based View:** Learn a score model $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$ to approximate the marginal score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$ via:

$$\mathcal{J}_{\mathrm{SM}}(\boldsymbol{\phi}) := \mathbb{E}_{p_t(\mathbf{x}_t)}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2\right];$$

- **Flow-Based View:** Learn a velocity model $\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$ to match the oracle velocity $\mathbf{v}_t(\mathbf{x}_t)$ (e.g., defined by Equation (5.2.9)) by minimizing:

$$\mathcal{J}_{\mathrm{FM}}(\boldsymbol{\phi}) := \mathbb{E}_{p_t(\mathbf{x}_t)}\left[\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t)\|_2^2\right].$$

At first glance, these objectives seem hopelessly intractable, since they all require access to oracle quantities that are fundamentally unknowable in general. But here comes the exciting twist: each method independently arrives at the same elegant solution to this problem: *conditioning on the data* $\mathbf{x}_0$. This technique transforms each intractable training target into a tractable one.

This elegant "conditioning technique" rewrites the objectives as expectations over the known Gaussian conditionals $p_t(\mathbf{x}_t|\mathbf{x}_0)$, yielding tractable conditional targets and training objectives that are gradient-equivalent to their marginal counterparts:

- **Variational View** (Equation (2.2.3)):

$$\mathcal{J}_{\mathrm{KL}}(\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0)\|p_{\boldsymbol{\phi}}(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t))\right]}_{\mathcal{J}_{\mathrm{CKL}}(\boldsymbol{\phi})} + C;$$

- **Score-Based View** (Equation (3.3.3)):

$$\mathcal{J}_{\mathrm{SM}}(\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2\right]}_{\mathcal{J}_{\mathrm{DSM}}(\boldsymbol{\phi})} + C;$$

- **Flow-Based View** (Equation (5.2.10)):

$$\mathcal{J}_{\mathrm{FM}}(\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0)\|^2\right]}_{\mathcal{J}_{\mathrm{CFM}}(\boldsymbol{\phi})} + C.$$

To build a unified view, we next revisit the conditional KL, score, and velocity objectives in a systematic manner. Crucially, these objectives are not only tractable but also equivalent to their original forms up to a constant vertical shift. The conditional versions ($\mathcal{J}_{\mathrm{CKL}}$, $\mathcal{J}_{\mathrm{DSM}}$, $\mathcal{J}_{\mathrm{CFM}}$) differ from the originals ($\mathcal{J}_{\mathrm{KL}}$, $\mathcal{J}_{\mathrm{SM}}$, $\mathcal{J}_{\mathrm{FM}}$) only by a constant that is independent of the model parameters. Consequently, each conditional objective has the same minimizers as its original counterpart within the same model class.

To see what these minimizers ideally recover, suppose the model is expressive enough to represent the exact oracle. Despite their different objectives, all three perspectives then share the same conceptual pattern:

$$\underbrace{\text{marginal oracle}}_{\text{generally intractable}} = \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)} \left[\underbrace{\text{conditional oracle}}_{\text{tractable given } \mathbf{x}_0} \right].$$

That is, conditioning on the clean sample $\mathbf{x}_0$ gives a tractable target, and averaging these conditional targets over the possible clean samples consistent with $\mathbf{x}_t$ recovers the desired marginal oracle. Concretely,

$$\begin{aligned} p^*(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)}\left[p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0)\right] &&= p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t), \\ \mathbf{s}^*(\mathbf{x}_t, t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)}\left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\right] &&= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \\ \mathbf{v}^*(\mathbf{x}_t, t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)}\left[\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0)\right] &&= \mathbf{v}_t(\mathbf{x}_t). \end{aligned} \tag{6.1.1}$$

For the variational view, the first identity follows from marginalizing the conditional reverse transition over $\mathbf{x}_0$, and the KL objective is minimized by the resulting marginal reverse transition. For score and velocity matching, the squared-error objectives are least-squares regressions whose population minimizers are the corresponding conditional expectations. These identities describe the unrestricted population oracles; a restricted neural model need not represent them exactly. Below is a concrete example of a scenario with a finite number of data points.

Example: Closed-Form Oracle Targets on a Finite Dataset

To make Equation (6.1.1) more concrete, suppose the data distribution is the empirical measure

$$\hat{p}_0 = \frac{1}{N} \sum_{i=1}^{N} \delta_{\mathbf{x}_0^{(i)}},$$

where $\{\mathbf{x}_0^{(i)}\}_{i=1}^N$ denotes the finite training set.

In Equation (6.1.1), the notation

$$\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)$$

means that we average over the posterior distribution of the clean sample given

the noisy observation $\mathbf{x}_t$. Equivalently, for any quantity $\mathbf{h}(\mathbf{x}_0)$,

$$\mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)}[\mathbf{h}(\mathbf{x}_0)] = \int \mathbf{h}(\mathbf{x}_0)\, p(\mathbf{x}_0|\mathbf{x}_t)\, \mathrm{d}\mathbf{x}_0.$$

By Bayes' rule,

$$p(\mathbf{x}_0|\mathbf{x}_t) = \frac{p_t(\mathbf{x}_t|\mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)}{\int p_t(\mathbf{x}_t|\tilde{\mathbf{x}}_0)\, p_{\text{data}}(\tilde{\mathbf{x}}_0)\, \mathrm{d}\tilde{\mathbf{x}}_0}.$$

Under the standard Gaussian diffusion forward process,

$$p_t(\mathbf{x}_t|\mathbf{x}_0^{(i)}) = \mathcal{N}\left(\mathbf{x}_t; \alpha_t \mathbf{x}_0^{(i)}, \sigma_t^2 \mathbf{I}\right),$$

so the conditional likelihood is explicit. However, at the population level, the posterior $p(\mathbf{x}_0|\mathbf{x}_t)$ still requires averaging over the full data distribution p_{data}.

If we now replace p_{data} by the empirical measure

$$\hat{p}_0 = \frac{1}{N} \sum_{i=1}^{N} \delta_{\mathbf{x}_0^{(i)}},$$

then the posterior $p(\mathbf{x}_0|\mathbf{x}_t)$ is supported only on the training examples, and its mass at $\mathbf{x}_0^{(i)}$ is

$$p(\mathbf{x}_0 = \mathbf{x}_0^{(i)}|\mathbf{x}_t) = \frac{p_t(\mathbf{x}_t|\mathbf{x}_0^{(i)})}{\sum_{j=1}^{N} p_t(\mathbf{x}_t|\mathbf{x}_0^{(j)})} =: w_i(\mathbf{x}_t, t), \qquad \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) = 1.$$

Hence the posterior expectation above reduces to the finite sum

$$\mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)}[\mathbf{h}(\mathbf{x}_0)] = \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) \mathbf{h}(\mathbf{x}_0^{(i)}).$$

Thus, in the finite-dataset case, the abstract posterior averaging in Equation (6.1.1) becomes a concrete weighted average over the training set, with weights $w_i(\mathbf{x}_t, t)$. We now make these three cases explicit.

Variational View. The oracle reverse transition becomes

$$p^*(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t) = \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0^{(i)}).$$

Here each conditional reverse kernel

$$p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0^{(i)})$$

is itself available in closed form under Gaussian diffusion; in fact, it is Gaussian as shown in Lemma 2.2.2. So the true reverse kernel is simply a posterior-weighted mixture of explicit Gaussian bridges.

Score-Based View. Likewise, the oracle score becomes

$$\mathbf{s}^*(\mathbf{x}_t, t) = \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0^{(i)}).$$

The conditional score is explicit:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0^{(i)}) = -\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0^{(i)}}{\sigma_t^2}.$$

Therefore,

$$\mathbf{s}^*(\mathbf{x}_t, t) = -\frac{1}{\sigma_t^2} \left(\mathbf{x}_t - \alpha_t \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) \mathbf{x}_0^{(i)} \right).$$

So the marginal score points from the noisy sample toward a posterior-weighted average of training examples.

Flow-Based View. The same phenomenon appears for flow matching:

$$\mathbf{v}^*(\mathbf{x}_t, t) = \sum_{i=1}^{N} w_i(\mathbf{x}_t, t) \mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0^{(i)}).$$

Again, the building blocks are explicit. The corresponding conditional velocity field has closed form:

$$\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0^{(i)}) = \alpha_t' \mathbf{x}_0^{(i)} + \frac{\sigma_t'}{\sigma_t} (\mathbf{x}_t - \alpha_t \mathbf{x}_0^{(i)}).$$

Hence the oracle velocity is also an explicit posterior-weighted average over the dataset.

Taken together, these formulas show that, on a finite dataset, all three oracle targets reduce to explicit posterior-weighted averages of closed-form conditional quantities.

Remark on Memorization. The formulas above characterize the exact oracle targets induced by the empirical distribution $\hat{p}_0$, rather than by the unknown population distribution p_{data}. If the model learned these empirical targets perfectly at every noise level, and if the reverse process were computed without error, then generation would exactly reproduce the empirical distribution $\hat{p}_0$. Since $\hat{p}_0$ assigns probability only to the training examples, the generated samples would be drawn from the training set itself. In this idealized sense, perfect fitting of the empirical oracle amounts to memorization.

However, a practically trained diffusion model, need not realize this oracle exactly: it is learned with finite model capacity and finite optimization, and its generated distribution is further affected by model approximation and

> numerical sampling errors. Therefore, the finite-dataset calculation establishes memorization as an exact empirical solution, but does not imply that every practically trained model necessarily memorizes its training set.

The common conditional reformulation is no coincidence: by making training tractable, it reveals a profound unification. Variational diffusion, score-based SDEs, and flow matching are simply different facets of the same principle. Three perspectives, one insight, elegantly connected. We will continue to explore their equivalence throughout the rest of this chapter.

6.2 A Roadmap for Elucidating Training Losses in Diffusion Models

This section builds a systematic view of training losses in diffusion models. In Section 6.2.1, we extend the standard three objectives to a broader set of four parameterizations, showing how they arise from different modeling perspectives. In Section 6.2.2, we then distill these results into a general framework that disentangles the structure of diffusion objectives, laying the groundwork for the equivalence results in Section 6.3.

6.2.1 Four Common Parameterizations in Diffusion Models

Throughout this section, we consider the forward perturbation kernel

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}\right),$$

where $\mathbf{x}_0 \sim p_{\text{data}}$, as defined in Equation (4.5.1), unless stated otherwise.

Let $\omega : [0, T] \to \mathbb{R}_{>0}$ denote a positive time-weighting function. The four standard parameterizations (noise $\boldsymbol{\epsilon}_{\boldsymbol{\phi}}$, clean $\mathbf{x}_{\boldsymbol{\phi}}$, score $\mathbf{s}_{\boldsymbol{\phi}}$, and velocity $\mathbf{v}_{\boldsymbol{\phi}}$), together with their respective minimizers $\boldsymbol{\epsilon}^*$, $\mathbf{x}^*$, $\mathbf{s}^*$, and $\mathbf{v}^*$, are summarized below for clarity and to facilitate further discussion.

Variational View. Based on the KL divergence in DDPMs (see Sections 2.2.4 and 4.5.2), this approach reduces to predicting either the expected noise that produces $\mathbf{x}_t$ or the expected clean signal that $\mathbf{x}_t$ was perturbed from.

1. **$\boldsymbol{\epsilon}$-Prediction (Noise Prediction)** (Ho *et al.*, 2020):

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t] = \boldsymbol{\epsilon}^*(\mathbf{x}_t, t) \tag{6.2.1}$$

   with training objective

$$\mathcal{L}_{\text{noise}}(\boldsymbol{\phi}) := \mathbb{E}_t \left[\omega(t)\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}} \left\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}\right\|_2^2\right].$$

   Here, $\boldsymbol{\epsilon}^*$ means the average noise that was injected to obtain the given $\mathbf{x}_t$.

2. **$\mathbf{x}$-Prediction (Clean Prediction)** (Kingma *et al.*, 2021; Karras *et al.*, 2022; Song *et al.*, 2023):

$$\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t] = \mathbf{x}^*(\mathbf{x}_t, t) \tag{6.2.2}$$

   with training objective

$$\mathcal{L}_{\text{clean}}(\boldsymbol{\phi}) := \mathbb{E}_t \left[\omega(t)\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}} \left\|\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{x}_0\right\|_2^2\right].$$

   Here, $\mathbf{x}^*$ means the average of all plausible clean guesses, given the noisy observation $\mathbf{x}_t$.

Score-Based View. Predicts the score function at noise level t, which points in the average direction to denoise $\mathbf{x}_t$ back toward all possible clean samples that could have generated it:

3. **Score Prediction** (Song and Ermon, 2019; Song *et al.*, 2020c):

$$\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \mathbb{E}\left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)|\mathbf{x}_t\right] = \mathbf{s}^*(\mathbf{x}_t, t) \tag{6.2.3}$$

with training objective

$$\mathcal{L}_{\text{score}}(\boldsymbol{\phi}) := \mathbb{E}_t\left[\omega(t)\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2\right],$$

where the conditional score satisfies $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{1}{\sigma_t}\boldsymbol{\epsilon}$.

Flow-Based View. Predicts the instantaneous average velocity of the data as it evolves through $\mathbf{x}_t$:

4. **v-Prediction (Velocity Prediction)** (Lipman *et al.*, 2022; Liu, 2022; Salimans and Ho, 2021; Albergo *et al.*, 2023):

$$\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \mathbb{E}\left[\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t}\middle|\mathbf{x}_t\right] = \mathbf{v}^*(\mathbf{x}_t, t) \tag{6.2.4}$$

with training objective

$$\mathcal{L}_{\text{velocity}}(\boldsymbol{\phi}) := \mathbb{E}_t\left[\omega(t)\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0, \boldsymbol{\epsilon})\right\|_2^2\right],$$

where the conditional velocity is $\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0, \boldsymbol{\epsilon}) = \alpha_t'\mathbf{x}_0 + \sigma_t'\boldsymbol{\epsilon}$.

Here, $\mathbf{v}^*$ indicates the average velocity vector passing through the observation point $\mathbf{x}_t$.

Building on the insight from Equation (6.1.1), all four prediction types ultimately aim to approximate a conditional expectation in the form of the average noise, clean data, score, or velocity given an observed $\mathbf{x}_t$.

6.2.2 Disentangling the Training Objective of Diffusion Models

As shown in Section 6.2.1, the objective functions for the four prediction types commonly share the following template form for diffusion model training:

$$\mathcal{L}(\boldsymbol{\phi}) := \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}} \underbrace{\mathbb{E}_{p_{\text{time}}(t)}}_{\substack{\text{time}\\ \text{distribution}}} \Big[\underbrace{\omega(t)}_{\substack{\text{time}\\ \text{weighting}}} \underbrace{\left\| \text{NN}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - (A_t\mathbf{x}_0 + B_t\boldsymbol{\epsilon}) \right\|_2^2}_{\text{MSE part}} \Big]. \tag{6.2.5}$$

Here, to enhance training efficiency and optimize the diffusion model learning pipeline, several key design choices are crucial (Karras *et al.*, 2022; Lu and Song, 2024):

(A) Noise schedule in the forward process of $\mathbf{x}_t$ via α_t and σ_t;

(B) Prediction types of $\mathrm{NN}_{\boldsymbol{\phi}}$ and their associated regression targets $(A_t\mathbf{x}_0 + B_t\boldsymbol{\epsilon})$;

(C) Time-weighting function $\omega(\cdot) : [0, T] \to \mathbb{R}_{\geq 0}$;

(D) Time distribution p_{time}.

We elaborate on these four components here to serve as a roadmap for the discussions in the following sections.

(A) Noise Schedule α_t and σ_t. Users have the flexibility to choose schedules tailored to their applications, with common examples summarized in Table 5.2. Importantly, as we will demonstrate in Equations (6.3.3) and (6.3.5), all affine flows of the form $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}$ are mathematically equivalent. Specifically, any such interpolation can be converted to the canonical linear schedule ($\alpha_t = 1 - t$, $\sigma_t = t$) or to a trigonometric schedule ($\alpha_t = \cos t$, $\sigma_t = \sin t$) by appropriate time reparametrization and spatial rescaling.

Table 6.1: Summary of the Relationships Between Different Parameterizations. All four parameterizations are mathematically equivalent and can be converted into one another through straightforward algebraic transformations.

Regression Target =	$A_t\mathbf{x}_0 + B_t\boldsymbol{\epsilon}$	
	A_t	B_t
Clean	1	0
Noise	0	1
Conditional Score	0	$-\frac{1}{\sigma_t}$
Conditional Velocity	α_t'	σ_t'

(B) Parameterization $\mathrm{NN}_{\boldsymbol{\phi}}$ and Training Target $\boldsymbol{A_t\mathbf{x}_0 + B_t\epsilon}$. There are two choices that are often coupled in diffusion training: *what quantity the network directly predicts* and *which quantity is compared in the training loss*. Clean data, noise, score, and velocity can be used for either purpose.

Matched Prediction and Loss. The common choice, used in Equation (6.2.5), is to train the network directly in the same prediction type that it outputs. Each prediction type then has a corresponding conditional training target of the form

$$\text{Regression Target} = A_t\mathbf{x}_0 + B_t\boldsymbol{\epsilon},$$

where A_t and B_t depend on the prediction type and the forward schedule (α_t, σ_t). These targets are summarized in Table 6.1.

For example, an $\mathbf{x}$-prediction network is compared with the clean target $\mathbf{x}_0$, whereas a $\mathbf{v}$-prediction network is compared with the conditional velocity

$$\mathbf{v}_t = \alpha_t' \mathbf{x}_0 + \sigma_t' \boldsymbol{\epsilon}.$$

As shown later in Equation (6.3.1), the four prediction types encode the same oracle information and can be converted into one another.

Predicting One Quantity and Evaluating the Loss in Another. The network output and the quantity used in the loss can also be chosen separately. The network prediction is first converted using the relations in Equation (6.3.1), and the loss is then computed against the corresponding target in the converted prediction type.

For example, under the canonical schedule

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon},$$

suppose the network directly predicts clean data $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$. Converting this prediction to velocity gives

$$\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) = \frac{\mathbf{x}_t - \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)}{t},$$

while the conditional velocity target satisfies

$$\mathbf{v}_t = \frac{\mathbf{x}_t - \mathbf{x}_0}{t}.$$

Therefore,

$$\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t\|_2^2 = \frac{1}{t^2} \|\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{x}_0\|_2^2.$$

Thus, with the network output fixed as clean prediction, evaluating a velocity loss is exactly equivalent to applying an additional time-dependent weight to the clean-prediction loss. The corresponding relations for general affine schedules and other prediction types are summarized later in Equation (6.3.6).

Changing the quantity predicted directly by the network has an additional effect. Although the oracle predictions are algebraically convertible, a finite neural network is then trained to represent a different function, with different output scales and parameter gradients. The choice of network prediction can therefore interact with model architecture and optimization, while the choice of loss also controls the relative weighting of prediction errors across noise levels.

For clarity, Equation (6.2.5) uses the matched prediction-and-loss case unless stated otherwise.

(C) Time Distribution $p_{\text{time}}(t)$. We next consider how time t, and hence the noise level, is sampled during training. For convenience, define the fixed-time regression loss

$$\mathcal{L}_t(\boldsymbol{\phi}) := \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \mathrm{NN}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - (A_t \mathbf{x}_0 + B_t \boldsymbol{\epsilon}) \right\|_2^2 \right], \qquad \mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}.$$

Then Equation (6.2.5) can be written compactly as

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{t\sim p_{\text{time}}}\left[\omega(t)\mathcal{L}_t(\boldsymbol{\phi})\right].$$

Thus, if no correction is applied, sampling some times more frequently also gives them more weight in the objective. At the level of the exact expected loss, only the product $p_{\text{time}}(t)\omega(t)$ matters.

A different use of nonuniform sampling is *importance sampling*. Suppose the objective above is fixed, but we instead draw times from a proposal $q_{\text{time}}(t)$. Then

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{t\sim q_{\text{time}}}\left[\frac{p_{\text{time}}(t)}{q_{\text{time}}(t)}\omega(t)\mathcal{L}_t(\boldsymbol{\phi})\right],$$

provided $q_{\text{time}}(t) > 0$ wherever $p_{\text{time}}(t)\omega(t) > 0$. With this correction, changing q_{time} leaves the objective unchanged and changes only its Monte Carlo estimator, for example its variance.

The key distinction is therefore:

$$\begin{aligned} \text{change } p_{\text{time}} \text{ in the objective} &\implies \text{different weighting, different objective} \\ \text{change } q_{\text{time}} + \text{importance correction} &\implies \text{same objective, different estimator} \end{aligned}$$

We therefore keep time sampling and loss weighting conceptually separate. Uniform time sampling makes this distinction less visible because its density contributes only a constant factor, whereas with nonuniform sampling one must specify whether the sampling distribution defines the objective or merely serves as an importance-sampling proposal.

Both uses appear in diffusion-model training. ScoreFlow explicitly uses importance sampling to reduce the variance of a fixed likelihood-weighted objective (Song *et al.*, 2021). VDM and subsequent analyses similarly show that, when the objective is fixed in noise-level coordinates, the noise schedule can act as an importance-sampling distribution (Kingma *et al.*, 2021; Kingma and Gao, 2023). EDM (Karras *et al.*, 2022), by contrast, specifies both a nonuniform training distribution over noise levels and a separate loss weighting, whose combination determines the effective training weight.

(D) Time-Weighting Function $\omega(t)$. For a fixed p_{time}, the weighting function $\omega(t)$ determines the relative contribution of different times to the training objective. Together with the previous paragraph, the effective weighting over time is therefore $p_{\text{time}}(t)\omega(t)$.

A simple baseline is uniform time sampling with $\omega(t) \equiv 1$, as in the standard simplified $\boldsymbol{\epsilon}$-prediction and FM/RF objectives (Ho *et al.*, 2020; Lipman *et al.*, 2022; Liu, 2022). Other weightings emphasize different noise regimes and can also carry likelihood or variational interpretations. For example, for the score-matching objective, the likelihood weighting $\omega(t) = g^2(t)$ yields an upper bound on the

negative log-likelihood under suitable regularity conditions (Song *et al.*, 2021). More generally, SNR-dependent weightings provide a common way to redistribute training emphasis across noise levels (Kingma *et al.*, 2021; Kingma and Gao, 2023).

Monotonic Weighting and the ELBO-with-Augmentation View. A useful way to describe the corruption level is through the *log signal-to-noise ratio* (log-SNR)

$$\Lambda_t := \log \frac{\alpha_t^2}{\sigma_t^2}.$$

We use Λ for the full log-SNR, distinguishing it from the half-log-SNR coordinate $\lambda_t = \log(\alpha_t/\sigma_t) = \frac{1}{2}\Lambda_t$ used later in Chapter 9. Since Λ_t decreases as the forward process becomes noisier, after expressing the loss in a common prediction parameterization, let $w(\Lambda)$ denote the effective weight assigned to each log-SNR level.

A useful result of Kingma and Gao (2023) is that, if $w(\Lambda)$ is non-increasing in Λ (equivalently, its weight does not decrease as the forward process becomes noisier), then the weighted diffusion objective can be rewritten, up to terms independent of the model parameters, as an average negative ELBO over Gaussian-perturbed data[1].

Thus, monotonic weighting has a concrete probabilistic interpretation: rather than merely emphasizing some noise levels more than others, it can be viewed as averaging variational training over Gaussian-corrupted versions of the data at different corruption strengths.

[1] To see why monotonicity matters, define the remaining variational cost from noise level t onward as

$$L(t;\mathbf{x}_0) := \mathcal{D}_{\mathrm{KL}}\left(p(\mathbf{x}_{[t,T]}|\mathbf{x}_0) \,\|\, p_{\boldsymbol{\phi}}(\mathbf{x}_{[t,T]})\right),$$

where $\mathbf{x}_{[t,T]} := \{\mathbf{x}_s\}_{s\in[t,T]}$ denotes the remaining continuous diffusion path. This is the continuous-time counterpart of the negative ELBO in Equation (2.2.13), starting from corruption level t; more precisely, it equals the expected negative ELBO of the perturbed sample $\mathbf{x}_t$, up to terms independent of $\boldsymbol{\phi}$.

For $\boldsymbol{\epsilon}$-prediction, Kingma and Gao (2023) show that

$$\frac{\mathrm{d}}{\mathrm{d}t}L(t;\mathbf{x}_0) = \frac{1}{2}\frac{\mathrm{d}\Lambda_t}{\mathrm{d}t}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_t,t)-\boldsymbol{\epsilon}\|_2^2\right].$$

Hence the weighted diffusion loss is

$$-\int_0^T w(\Lambda_t)\frac{\mathrm{d}}{\mathrm{d}t}L(t;\mathbf{x}_0)\,\mathrm{d}t = w(\Lambda_0)L(0;\mathbf{x}_0) + \int_0^T \frac{\mathrm{d}}{\mathrm{d}t}w(\Lambda_t)L(t;\mathbf{x}_0)\,\mathrm{d}t + C,$$

where C is independent of $\boldsymbol{\phi}$. If $w(\Lambda)$ is non-increasing in Λ, then $\frac{\mathrm{d}}{\mathrm{d}t}w(\Lambda_t) \geq 0$ because Λ_t decreases with t. Thus, after normalization, the terms above form a nonnegative weighting over ELBOs at different Gaussian corruption levels, which gives the ELBO-with-Gaussian-augmentation interpretation.

Conclusion. For our purposes, the main lesson is that many apparent design choices in diffusion training can be understood through how they reshape the effective weighting over noise levels in the objective. This weighting in turn influences both the optimization landscape in practice and, in special cases such as monotonic weighting, the variational interpretation of the loss.

6.3 Equivalence in Diffusion Models

Many formulations of diffusion models look different because they use different prediction types, training objectives, noise schedules, or continuous-time dynamics. In this section, we clarify in what precise sense these formulations can be equivalent at the exact theoretical level, and how their behavior can still differ for learned models in practice.

At the exact or oracle level, several notions of equivalence will appear:

- **Predictions.** Given the noisy observation $\mathbf{x}_t$ and the forward schedule, clean-data, noise, score, and velocity oracles contain the same information and can be converted into one another algebraically.
- **Training Objectives.** Squared-error objectives written in different prediction types can differ by positive time-dependent factors while sharing the same pointwise oracle minimizer. These factors determine how strongly different noise levels are weighted during training.
- **Affine Forward Process.** At the particle level, affine forward processes

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$$

  can be related to canonical affine forms through state rescaling and a corresponding canonical coordinate. When this coordinate evolves monotonically with time, the relation also defines a time reparameterization.
- **Marginal Density Paths.** At the density level, different deterministic and stochastic dynamics can produce the same marginal path $\{p_t\}_{t\in[0,T]}$. Their sample trajectories may differ, while the evolution of p_t is governed by the corresponding continuity or Fokker-Planck equation.

We develop these connections from prediction to sampling. Section 6.3.1 first relates the four common oracle prediction types and their training losses. Section 6.3.2 then shows how these predictions lead to different expressions of the same oracle PF–ODE. Next, Section 6.3.3 relates affine probability paths to canonical linear and trigonometric forms. Finally, Section 6.3.4 develops additional intuition for the canonical linear schedule and velocity prediction, including their implications for training and numerical integration.

These exact relationships provide a common mathematical foundation, while the practical choices remain meaningful. Finite neural networks may approximate equivalent oracle quantities differently, and loss weighting, optimization, model architecture, and numerical integration can lead to different training and sampling behavior.

6.3.1 Four Prediction Types Are Equivalent

We begin by analyzing the design choices for component (B) in Equation (6.2.5).

We have seen that the four prediction types are not independent choices but different views of the same underlying quantity. For example, noise and clean predictions are directly related (Section 2.2.4), as are score and noise predictions (Section 3.4). This recurring pattern points to a deeper principle: all four parameterizations are algebraically equivalent and can be converted into one another through simple transformations. To make this connection precise, we state the following proposition, illustrated in Figure 6.1, following (Kingma *et al.*, 2021).

Proposition 6.3.1: Equivalence of Parametrizations

Let the optimal predictions minimizing their respective objectives be

$$\boldsymbol{\epsilon}^*(\mathbf{x}_t, t), \quad \mathbf{x}^*(\mathbf{x}_t, t), \quad \mathbf{s}^*(\mathbf{x}_t, t), \quad \mathbf{v}^*(\mathbf{x}_t, t),$$

corresponding to noise, clean, score, and velocity parameterizations. These satisfy the following equivalences:

$$\begin{aligned} \boldsymbol{\epsilon}^*(\mathbf{x}_t, t) &= -\sigma_t \mathbf{s}^*(\mathbf{x}_t, t), \\ \mathbf{x}^*(\mathbf{x}_t, t) &= \frac{1}{\alpha_t}\mathbf{x}_t + \frac{\sigma_t^2}{\alpha_t}\mathbf{s}^*(\mathbf{x}_t, t), \\ \mathbf{v}^*(\mathbf{x}_t, t) &= \alpha_t' \mathbf{x}^* + \sigma_t' \boldsymbol{\epsilon}^* = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}_t, t). \end{aligned} \tag{6.3.1}$$

Here, $f(t)$ and $g(t)$ are related to α_t and σ_t via Lemma 4.5.1. Moreover, these minimizers satisfy the identities given in Equations (6.2.1) to (6.2.4).

Proof for Proposition.

The proof is similar to that of Theorem 4.3.1, which analyzes the global optimum of various matching losses under the DSM objective. See Section D.4 for details.

The proposition is most naturally understood as an equivalence of *oracle information*. At any noise level, knowing one oracle prediction, together with the noisy observation $\mathbf{x}_t$ and the forward schedule, determines the others through the identities in Equation (6.3.1). For example, the score determines the posterior-mean clean prediction, and either of these determines the PF-ODE velocity. Thus, at the oracle level, these parameterizations contain the same information about the underlying diffusion process.

For a learned model, however, one must distinguish this oracle equivalence from the quantity predicted directly by the neural network. One may choose any

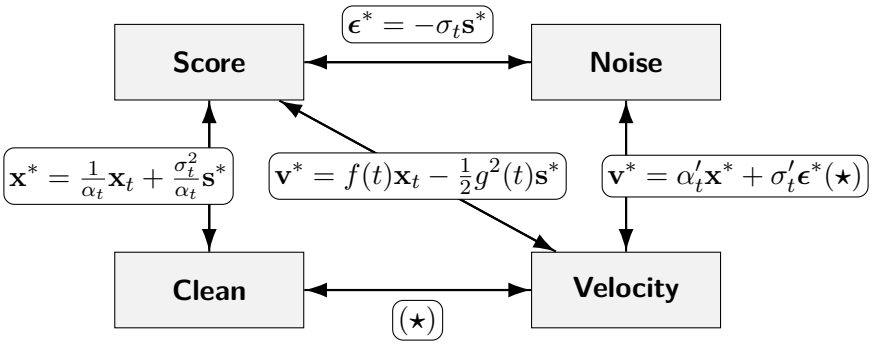


Figure 6.1: Equivalent relations among the four oracle parameterizations. Each oracle prediction can be converted into the others using the forward schedule and the observed state $\mathbf{x}_t$. For example, $\mathbf{v}^* = \alpha_t'\mathbf{x}^* + \sigma_t'\boldsymbol{\epsilon}^*$, while $\mathbf{x}$- and $\boldsymbol{\epsilon}$-predictions satisfy $\mathbf{x}_t = \alpha_t\mathbf{x}^* + \sigma_t\boldsymbol{\epsilon}^*$.

Source: Created by the authors.

of

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_t, t), \quad \mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t), \quad \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t), \quad \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$$

as the network output. Once one prediction is taken as primary, the others can be defined through the same algebraic conversions as in Equation (6.3.1). In particular, for a trained model with parameters $\boldsymbol{\phi}^\times$, these converted predictions are exactly consistent by construction; for example,

$$\mathbf{x} = \alpha_t \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t) + \sigma_t \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t).$$

These conversions are useful both during and after training. For instance, a noise prediction can be converted into a score for reverse-SDE sampling or into a velocity for PF-ODE sampling.

The network prediction parameterization should also be distinguished from the *training loss*. As discussed in component (B) of Equation (6.2.5), the quantity output by the network need not be the quantity on which the loss is evaluated: the prediction may first be converted into another parameterization before computing the objective. Such a conversion generally changes a squared-error objective by a time-dependent weighting, as shown later in Equation (6.3.6).

These distinctions matter for finite neural networks. Although the oracle targets are information-equivalent, directly predicting different quantities leads the network to represent and optimize different functions, and the corresponding objectives may weight noise levels differently. Consequently, models trained independently under different prediction parameterizations need not produce identical empirical predictions or identical PF-ODEs. The equivalence in this proposition therefore concerns the information carried by the ideal oracle predictions; the network output parameterization and the training objective remain separate and meaningful design choices.

6.3.2 PF-ODE in Different Parameterizations

The PF-ODE admits several equivalent parameterizations (score, noise, denoised, and velocity). Although interchangeable in principle, the choice has practical consequences: it changes the stiffness of the vector field, the behavior of discretization error, and the ease of optimization. For fast sampling with advanced ODE solvers (see Chapter 9), practitioners often work with $\boldsymbol{\epsilon}$ or $\mathbf{x}$ prediction because they align well with solver inputs and reduce error accumulation. For training generators that use only a few function evaluations (see Chapter 11), $\mathbf{x}$ or $\mathbf{v}$ prediction often yields smoother objectives and better step to step consistency.

We write the PF-ODE under each parameterization and make the conversions explicit using Equation (6.3.1). The results are collected in the following proposition.

Proposition 6.3.2: PF-ODE in Different Parameterizations

Let α_t and σ_t be the forward perturbation schedules, and denote time derivatives by $\alpha_t' := \frac{\mathrm{d}\alpha_t}{\mathrm{d}t}$ and $\sigma_t' := \frac{\mathrm{d}\sigma_t}{\mathrm{d}t}$. Then the empirical PF-ODE admits the equivalent forms

$$\begin{aligned}
\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} &= \frac{\alpha_t'}{\alpha_t}\mathbf{x}(t) - \sigma_t\left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t}\right)\boldsymbol{\epsilon}^*(\mathbf{x}(t), t) \\
&= \frac{\sigma_t'}{\sigma_t}\mathbf{x}(t) + \alpha_t\left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t}\right)\mathbf{x}^*(\mathbf{x}(t), t) \\
&= \frac{\alpha_t'}{\alpha_t}\mathbf{x}(t) + \sigma_t^2\left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t}\right)\mathbf{s}^*(\mathbf{x}(t), t) \\
&= \alpha_t'\mathbf{x}^*(\mathbf{x}(t), t) + \sigma_t'\boldsymbol{\epsilon}^*(\mathbf{x}(t), t) \\
&= \mathbf{v}^*(\mathbf{x}(t), t).
\end{aligned} \tag{6.3.2}$$

To see the Score SDE notation, we recall Lemma 4.5.1. If we set

$$f(t) = \frac{\alpha_t'}{\alpha_t}, \quad g^2(t) = \frac{\mathrm{d}}{\mathrm{d}t}(\sigma_t^2) - 2\frac{\alpha_t'}{\alpha_t}\sigma_t^2 = 2\sigma_t\sigma_t' - 2\frac{\alpha_t'}{\alpha_t}\sigma_t^2,$$

then the PF-ODE can be written in the familiar Score SDE form:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}(t), t).$$

To give a concrete sense of how the PF-ODE is discretized for sampling, we will present in Section 9.2 the update rule of a widely used diffusion-based ODE sampler, the DDIM scheme. This example will show how an Euler discretization naturally connects with the PF-ODE.

6.3.3 All Affine Flows Are Equivalent

We next analyze the design choices for component (A) in Equation (6.2.5).

State-Level Equivalence. A convenient canonical interpolation used in FM (Lipman *et al.*, 2022) and RF (Liu, 2022) is

$$\mathbf{x}_t^{\mathrm{FM}} = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon} = \mathbf{x}_0 + t(\boldsymbol{\epsilon} - \mathbf{x}_0),$$

whose velocity is the constant vector $\boldsymbol{\epsilon} - \mathbf{x}_0$. The key point of this subsection is that the apparent simplicity of this choice is not essential: any affine interpolation

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$$

can be written as a time–reparameterized and rescaled version of the canonical path. Define

$$c(t) := \alpha_t + \sigma_t, \qquad \tau(t) := \frac{\sigma_t}{\alpha_t + \sigma_t} \quad (c(t) \neq 0).$$

A direct algebraic rewrite yields

$$\begin{aligned}
\mathbf{x}_t &= \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon} \\
&= (\alpha_t + \sigma_t)\left(\frac{\alpha_t}{\alpha_t + \sigma_t}\mathbf{x}_0 + \frac{\sigma_t}{\alpha_t + \sigma_t}\boldsymbol{\epsilon}\right) \\
&= c(t)\left((1-\tau(t))\mathbf{x}_0 + \tau(t)\boldsymbol{\epsilon}\right) = c(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}}.
\end{aligned}$$

Hence, whenever $c(t) \neq 0$, every affine path can be represented pointwise as a scalar rescaling of the canonical path. If, in addition, $\tau(t)$ is continuous and strictly monotone over the time interval of interest, then $t \mapsto \tau(t)$ is also a genuine change of time variable, and the two paths are related by time reparameterization together with the rescaling $c(t)$. For the usual forward diffusion schedules, monotonic decay of the signal-to-noise ratio implies monotonic evolution of $\tau(t)$.

For the associated velocities, apply the chain rule to $\mathbf{x}_t = c(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}}$:

$$\begin{aligned}
\mathbf{v}(\mathbf{x}_t, t) &:= \frac{\mathrm{d}}{\mathrm{d}t}\left(\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}t}\left(c(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}}\right) \\
&= c'(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}} + c(t)\tau'(t)\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{x}_s^{\mathrm{FM}}\Big|_{s=\tau(t)} \\
&= c'(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}} + c(t)\tau'(t)\mathbf{v}^{\mathrm{FM}}\left(\mathbf{x}_{\tau(t)}^{\mathrm{FM}}, \tau(t)\right),
\end{aligned}$$

since $\mathbf{v}^{\mathrm{FM}}(\mathbf{x}_\tau^{\mathrm{FM}}, \tau) = -\mathbf{x}_0 + \boldsymbol{\epsilon}$ along the canonical path.

We summarize the above derivation as a formal statement in the following proposition.

Proposition 6.3.3: Equivalence of Affine Flows

Let $\mathbf{x}_t^{\mathrm{FM}} = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon}$ and $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$, with $c(t) := \alpha_t + \sigma_t \neq 0$ and $\tau(t) := \sigma_t/(\alpha_t + \sigma_t)$. Then

$$\begin{aligned} \mathbf{x}_t &= c(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}}, \\ \mathbf{v}(\mathbf{x}_t, t) &= c'(t)\mathbf{x}_{\tau(t)}^{\mathrm{FM}} + c(t)\tau'(t)\mathbf{v}^{\mathrm{FM}}\left(\mathbf{x}_{\tau(t)}^{\mathrm{FM}}, \tau(t)\right). \end{aligned} \tag{6.3.3}$$

Thus, every affine interpolation with $c(t) \neq 0$ admits this canonical state-level representation. When $\tau(t)$ is additionally continuous and strictly monotone, it can also be interpreted as a time reparameterization together with the scalar rescaling $c(t)$.

Equivalence with Trigonometric Flow. Another widely used affine flow is the trigonometric interpolation (Salimans and Ho, 2021; Albergo *et al.*, 2023; Lu and Song, 2024). As a concrete example, we also show that *any* affine flow can be expressed in this form. The trigonometric path is defined by

$$\mathbf{x}_u^{\mathrm{Trig}} := \cos(u)\mathbf{x}_0 + \sin(u)\boldsymbol{\epsilon}. \tag{6.3.4}$$

Let $R_t := \sqrt{\alpha_t^2 + \sigma_t^2}$ and assume $R_t > 0$. Choose an angle τ_t so that

$$\cos\tau_t = \frac{\alpha_t}{R_t}, \qquad \sin\tau_t = \frac{\sigma_t}{R_t}.$$

Then every affine interpolation $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ admits a pointwise rescaled and re-timed trigonometric path:

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon} = R_t\left(\frac{\alpha_t}{R_t}\mathbf{x}_0 + \frac{\sigma_t}{R_t}\boldsymbol{\epsilon}\right) = R_t \mathbf{x}_{\tau_t}^{\mathrm{Trig}}. \tag{6.3.5}$$

The pair (α_t, σ_t) is a point in the plane. Normalizing by R_t places it on the unit circle, which fixes the angle τ_t and hence the state $\mathbf{x}_{\tau_t}^{\mathrm{Trig}}$; the radius R_t gives the overall scale. If τ_t can additionally be chosen as a continuous and strictly monotone function of t on the interval of interest, then this state-level representation also defines a genuine time reparameterization.

Differentiating $\mathbf{x}_u^{\mathrm{Trig}}$ with respect to u gives its velocity,

$$\mathbf{v}_u^{\mathrm{Trig}} = -\sin(u)\mathbf{x}_0 + \cos(u)\boldsymbol{\epsilon}.$$

Through the same change of variables as in Equation (6.3.5), this relation provides closed-form conversions for the velocity (and analogously for other parameterizations).

Summarizing the above discussion, we arrive at the following conclusion:

Conclusion 6.3.1:

Regardless of the schedule (α_t, σ_t), including VE, VP (such as trigonometric), FM, or RF, affine interpolations are mutually convertible by a suitable change of time variable and a scalar rescaling.

Training Objectives of Four Parameterizations. Let $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ with $\sigma_t > 0$ and differentiable (α_t, σ_t) such that $\alpha_t' \sigma_t - \alpha_t \sigma_t' \neq 0$. Consider the oracle targets

$$\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t], \quad \mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t], \quad \mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E}[\alpha_t' \mathbf{x}_0 + \sigma_t' \boldsymbol{\epsilon}|\mathbf{x}_t].$$

From Proposition 6.3.1, they satisfy

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \frac{\alpha_t}{\sigma_t^2} \left(\mathbf{x}_0^*(\mathbf{x}_t, t) - \frac{\mathbf{x}_t}{\alpha_t} \right), \quad \mathbf{v}^* = \alpha_t' \mathbf{x}_0^* + \sigma_t' \boldsymbol{\epsilon}^*.$$

Under the head conversions

$$\mathbf{s}_{\boldsymbol{\phi}} \equiv -\frac{1}{\sigma_t} \boldsymbol{\epsilon}_{\boldsymbol{\phi}} \equiv \frac{\alpha_t}{\sigma_t^2} \left(\mathbf{x}_{\boldsymbol{\phi}} - \frac{\mathbf{x}_t}{\alpha_t} \right),$$

and the velocity-to-score conversion is

$$\mathbf{s}_{\boldsymbol{\phi}} = \frac{\alpha_t}{\sigma_t(\alpha_t' \sigma_t - \alpha_t \sigma_t')} \mathbf{v}_{\boldsymbol{\phi}} - \frac{\alpha_t'}{\sigma_t(\alpha_t' \sigma_t - \alpha_t \sigma_t')} \mathbf{x}_t,$$

the per–sample squared losses match up to time–dependent weights:

$$\begin{aligned} \left\| \mathbf{s}_{\boldsymbol{\phi}} - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \right\|_2^2 &= \frac{1}{\sigma_t^2} \left\| \boldsymbol{\epsilon}_{\boldsymbol{\phi}} - \boldsymbol{\epsilon}^* \right\|_2^2 \\ &= \frac{\alpha_t^2}{\sigma_t^4} \left\| \mathbf{x}_{\boldsymbol{\phi}} - \mathbf{x}_0^* \right\|_2^2 \\ &= \left(\frac{\alpha_t}{\sigma_t(\alpha_t' \sigma_t - \alpha_t \sigma_t')} \right)^2 \left\| \mathbf{v}_{\boldsymbol{\phi}} - \mathbf{v}^* \right\|_2^2. \end{aligned} \tag{6.3.6}$$

By Proposition 6.3.3, any affine flow $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ is transferable to the canonical FM path via $\mathbf{x}_t = c(t) \mathbf{x}_{\tau(t)}^{\mathrm{FM}}$ with $c(t) = \alpha_t + \sigma_t$ and $\tau(t) = \sigma_t/(\alpha_t + \sigma_t)$. Differentiating gives

$$\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) = c'(t) \mathbf{x}_{\tau(t)}^{\mathrm{FM}} + c(t) \tau'(t) \mathbf{v}_{\boldsymbol{\phi}}^{\mathrm{FM}} \left(\mathbf{x}_{\tau(t)}^{\mathrm{FM}}, \tau(t) \right), \qquad \mathbf{x}_{\tau(t)}^{\mathrm{FM}} = \frac{\mathbf{x}_t}{c(t)},$$

and the same relation holds for $\mathbf{v}^*$. Hence the velocity loss transforms by

$$\left\| \mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}^*(\mathbf{x}_t, t) \right\|_2^2 = \left(c(t) \tau'(t) \right)^2 \left\| \mathbf{v}_{\boldsymbol{\phi}}^{\mathrm{FM}} \left(\frac{\mathbf{x}_t}{c(t)}, \tau(t) \right) - \left(\mathbf{v}^{\mathrm{FM}} \right)^* \left(\frac{\mathbf{x}_t}{c(t)}, \tau(t) \right) \right\|_2^2.$$

With the above observation, we arrive at the following conclusion:

Conclusion 6.3.2:

Score, noise, clean, and velocity training objectives are theoretically equivalent up to time–dependent weights (and, for velocity, an affine head conversion involving $\mathbf{x}_t$) determined by (α_t, σ_t).

6.3.4 (Optional) Analysis of Parametrizations and the Canonical Flow

Even though we have shown in the previous sections that all four parameterizations are mathematically equivalent and can be transformed into one another, and that the forward affine noise-injection flow is equivalent to the canonical form

$$\mathbf{x}_t^{\mathrm{FM}} = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon},$$

in this subsection we provide further intuition and analyze the potential advantages of using the $\mathbf{v}$-prediction parameterization together with this canonical affine flow.

This subsection asks a simple question: *how do different parameterizations and schedules shape what the model learns and how we sample?* We proceed in two conceptual perspectives:

- **Regression Targets and Schedules.** We focus on why combining $\mathbf{v}$-prediction with the canonical linear schedule $(\alpha_t, \sigma_t) = (1-t, t)$ is natural: it keeps the conditional velocity target scale constant in time and removes the explicit schedule-curvature terms from the velocity decomposition.
- **Solver Implications.** We examine how this parameterization interacts with numerical integration, while deferring the detailed solver analysis to Chapter 9.

Before proceeding, we distinguish between two types of velocity fields to avoid ambiguity. The *conditional velocity*, which serves as a tractable training target, is defined as

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}) = \mathbf{x}_t' = \alpha_t'\mathbf{x}_0 + \sigma_t'\boldsymbol{\epsilon}, \quad \text{where } \mathbf{z} = (\mathbf{x}_0, \boldsymbol{\epsilon}),$$

while the *oracle (marginalized) velocity*, used to move samples during inference of PF-ODE solving, is given by

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}\big[\mathbf{v}_t(\cdot|\mathbf{z})\big|\mathbf{x}_t = \mathbf{x}\big].$$

Perspective 1: Why $(\alpha_t, \sigma_t) = (1-t, t)$ is a Natural Schedule. Writing $\sigma_t := \zeta(t)$ and $\alpha_t := 1 - \zeta(t)$ for a time-varying $\zeta(t)$, the conditional velocity becomes

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}) = \zeta'(t)(\boldsymbol{\epsilon} - \mathbf{x}_0), \quad \text{where } \mathbf{z} = (\mathbf{x}_0, \boldsymbol{\epsilon}).$$

Unit-Scale Regression Targets. For the canonical schedule $\zeta(t) = t$, the conditional velocity $\mathbf{v}_t(\cdot|\mathbf{z})$ satisfies

$$\mathbb{E}\left[\|\mathbf{v}_t(\cdot|\mathbf{z})\|_2^2\right] = \mathbb{E}_{\boldsymbol{\epsilon}}\|\boldsymbol{\epsilon}\|_2^2 + \mathbb{E}_{\mathbf{x}_0}\|\mathbf{x}_0\|_2^2 = D + \underbrace{\operatorname{Tr}\operatorname{Cov}[\mathbf{x}_0]}_{\text{total variance}} + \underbrace{\|\mathbb{E}\mathbf{x}_0\|_2^2}_{\text{mean}}. \tag{6.3.7}$$

Thus the expected target magnitude is constant in t. After standardizing the data to zero mean and identity covariance (i.e., $\operatorname{Cov}[\mathbf{x}_0] = \mathbf{I}$), the two components $\alpha_t'\mathbf{x}_0$ and $\sigma_t'\boldsymbol{\epsilon}$ contribute comparably for all t, avoiding gradient explosion/vanishing near the endpoints. To see this, we consider the diffusion's training objective:

$$\mathcal{L}_{\text{velocity}}(\boldsymbol{\phi}) = \mathbb{E}_t\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\right\|_2^2\right].$$

By applying the chain rule, the gradient of this loss with respect to the model parameters $\boldsymbol{\phi}$ is

$$\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\text{velocity}}(\boldsymbol{\phi}) = 2\mathbb{E}_t\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\partial_{\boldsymbol{\phi}}\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)^\top\left(\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\right)\right].$$

Thus the scale of the target $\|\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|_2$ influences gradient stability: if it collapses to 0 (or blows up) at some t, gradients tend to vanish (or explode), all else equal. With the canonical choice $\zeta(t) = t$, Equation (6.3.7) gives a t-independent target magnitude, so there is no endpoint ($t = 0$ or $t = 1$) collapse or blow-up arising from the regression signal (assuming $\mathbb{E}\|\partial_{\boldsymbol{\phi}}\mathbf{v}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)\|^2$ and any time-weights are controlled).

Interplay of the Canonical Schedule and v-Prediction. Under the affine path $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}$, the oracle velocity decomposes as

$$\mathbf{v}^*(\mathbf{x}, t) = \alpha_t'\mathbf{x}^*(\mathbf{x}, t) + \sigma_t'\boldsymbol{\epsilon}^*(\mathbf{x}, t),$$

with $\mathbf{x}^* = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t = \mathbf{x}]$ and $\boldsymbol{\epsilon}^* = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t = \mathbf{x}]$. Differentiating at fixed $\mathbf{x}$ gives

$$\partial_t\mathbf{v}^* = \underbrace{\alpha_t''\mathbf{x}^* + \sigma_t''\boldsymbol{\epsilon}^*}_{\text{schedule curvature}} + \alpha_t'\partial_t\mathbf{x}^* + \sigma_t'\partial_t\boldsymbol{\epsilon}^*.$$

With the linear schedule $\alpha_t = 1 - t,\ \sigma_t = t$, the curvature terms vanish ($\alpha_t'' = \sigma_t'' = 0$), so the time-variation of $\mathbf{v}^*$ primarily reflects the posterior evolution ($\partial_t\mathbf{x}^*, \partial_t\boldsymbol{\epsilon}^*$) rather than the schedule. This effect is especially clean for $\mathbf{v}$-prediction: the coefficients α_t', σ_t' are constants (-1 and $+1$), avoiding extra t-dependent rescaling in the drift. By contrast, score-, $\mathbf{x}_0$, or $\boldsymbol{\epsilon}$-parameterizations often introduce ratios such as σ_t'/σ_t or α_t'/α_t that can vary sharply near the endpoints, even under a linear schedule. Hence, while not exclusive in principle, the linear $(1 - t, t)$ schedule combined with v-prediction offers a particularly stable and transparent time dependence for the oracle velocity.

Minimizing the Conditional Energy. We next adopt a more theoretical perspective of optimal transport (see Chapter 7). Here, the *conditional kinetic energy* quantifies the total expected motion of the conditional velocity along the forward path, that is, the amount of instantaneous movement (or kinetic effort) required to traverse from $\mathbf{x}_0$ to $\boldsymbol{\epsilon}$:

$$\mathcal{K}[\zeta] := \int_0^1 \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}[\|\mathbf{v}_t(\cdot|\mathbf{z})\|_2^2]\,\mathrm{d}t = \left(D + \operatorname{Tr}\operatorname{Cov}[\mathbf{x}_0] + \|\mathbb{E}\mathbf{x}_0\|_2^2\right)\int_0^1 \left(\zeta'(t)\right)^2\,\mathrm{d}t.$$

Minimizing $\mathcal{K}[\zeta]$ therefore corresponds to finding the smoothest, least-energy path in expectation within the affine interpolation family

$$\mathbf{x}_t = (1-\zeta(t))\mathbf{x}_0 + \zeta(t)\boldsymbol{\epsilon}.$$

With the boundary conditions $\zeta(0) = 0$ and $\zeta(1) = 1$, the Euler–Lagrange equation $\zeta''(t) = 0$ gives the minimizer $\zeta(t) = t$, corresponding to a straight conditional path. This means that, among all smooth schedules ζ in this family, the canonical flow $\zeta(t) = t$ is the most energy-efficient way to move from $\mathbf{x}_0$ to $\boldsymbol{\epsilon}$. We will revisit this point in Proposition 7.5.1 for a more detailed treatment.

Remark on the Energy of the Oracle Velocity. The linear schedule $\zeta(t) = t$ always minimizes the *conditional* velocity energy derived above, but it need not minimize the energy of the marginalized oracle velocity $\mathbf{v}^*$. The reason is that the magnitude of $\mathbf{v}^*$ can vary along the probability path, so the energy-minimizing schedule may traverse different parts of the path at different speeds.

To make this dependence explicit, let

$$\mathbf{y}_u = (1-u)\mathbf{x}_0 + u\boldsymbol{\epsilon}, \qquad u \in [0,1],$$

where u denotes position along the affine probability path, and define

$$\mathbf{b}_u(\mathbf{y}) := \mathbb{E}[\boldsymbol{\epsilon} - \mathbf{x}_0|\mathbf{y}_u = \mathbf{y}], \qquad \kappa(u) := \mathbb{E}_{\mathbf{y}_u}\left[\|\mathbf{b}_u(\mathbf{y}_u)\|_2^2\right].$$

Here, $\kappa(u)$ measures the expected squared magnitude of the oracle velocity per unit movement along the path.

A schedule $u = \zeta(t)$ specifies how quickly this path is traversed. Since $\mathbf{x}_t = \mathbf{y}_{\zeta(t)}$, the oracle velocity becomes

$$\mathbf{v}^*(\mathbf{x}, t) = \zeta'(t)\mathbf{b}_{\zeta(t)}(\mathbf{x}),$$

with kinetic energy

$$\mathcal{K}_{\text{marg}}[\zeta] = \int_0^1 \kappa(\zeta(t))\left(\zeta'(t)\right)^2\,\mathrm{d}t.$$

Assuming $\kappa(u) > 0$ and sufficient regularity, minimizing this energy gives

$$\kappa(\zeta(t))\left(\zeta'(t)\right)^2 = \text{constant}, \qquad \text{or equivalently} \qquad \zeta'(t) \propto \frac{1}{\sqrt{\kappa(\zeta(t))}}.$$

Thus, the energy-minimizing schedule moves more slowly where the oracle velocity has larger expected magnitude and faster where it has smaller magnitude.

Consequently, $\zeta(t) = t$ minimizes the marginal-velocity energy only when $\kappa(u)$ is constant along the path. This highlights the distinction from the conditional-velocity result above, where the linear schedule is always the constant-speed minimum-energy choice within the affine family.

Perspective 2: Why Velocity Prediction Can Be Considered Natural for Sampling. Sampling with the PF-ODE ultimately requires its velocity field. This makes $\mathbf{v}$-prediction particularly direct: the network predicts the quantity that appears on the right-hand side of the ODE itself. By comparison, $\mathbf{x}$-, $\boldsymbol{\epsilon}$-, and $\mathbf{s}$-predictions encode the same oracle velocity through a combination of the current state and the network prediction.

At the oracle level, these are exactly convertible forms of the same $\mathbf{v}^*(\mathbf{x}, t)$ by Equation (6.3.2); hence, the underlying oracle PF-ODE is unchanged by the choice of parameterization. Here, we focus on how the learned PF-ODE is expressed during sampling; the loss used to train the corresponding prediction can be chosen separately.

PF-ODE under $\mathbf{x}$-, $\boldsymbol{\epsilon}$-, and $\mathbf{s}$-Predictions. The first three identities in Equation (6.3.2) share the form

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underbrace{L(t)\mathbf{x}(t)}_{\text{known linear term}} + \underbrace{\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t)}_{\text{network-dependent term}},$$

where $L(t)$ denotes the corresponding known scalar coefficient determined by the noise schedule, and $\mathbf{N}_{\boldsymbol{\phi}^\times}$ contains the appropriate time-dependent coefficient multiplying the clean, noise, or score prediction. For example, under $\boldsymbol{\epsilon}$-prediction,

$$L(t) = \frac{\alpha_t'}{\alpha_t}, \qquad \mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t) = -\sigma_t \left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t} \right) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t).$$

This form is useful for numerical integration because the known linear term can be handled analytically while the network-dependent term is approximated numerically. Exponential-integrator-based diffusion solvers such as DPM-Solver (Lu *et al.*, 2022b) exploit precisely this structure. We develop this solver viewpoint in detail in Chapter 9.

PF-ODE under $\mathbf{v}$-Prediction. With $\mathbf{v}$-prediction, the PF-ODE takes the direct and simple form

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t) \approx \mathbf{v}^*(\mathbf{x}(t), t).$$

Thus, a single network evaluation directly provides the quantity required by a standard ODE solver, without first combining the prediction with a separate linear term.

The canonical linear schedule makes this connection especially transparent. For

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon},$$

a velocity prediction determines the corresponding endpoint predictions

$$\widehat{\mathbf{x}}_0 = \mathbf{x}_t - t\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t), \qquad \widehat{\boldsymbol{\epsilon}} = \mathbf{x}_t + (1-t)\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t).$$

Using this single prediction to move from t to s gives

$$(1-s)\widehat{\mathbf{x}}_0 + s\widehat{\boldsymbol{\epsilon}} = \mathbf{x}_t + (s-t)\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t),$$

which is exactly the plain Euler update for the learned velocity field under the canonical linear schedule. Thus, the combination of $\mathbf{v}$-prediction and the linear schedule gives a particularly direct connection between the model prediction and numerical integration of the PF-ODE. We defer a detailed discussion of numerical solvers and more general schedules to Chapter 9.

Taken together, $\mathbf{v}$-prediction directly provides the velocity required by the PF-ODE, whereas the other prediction types represent the same oracle dynamics through schedule-dependent transformations. This direct correspondence is the main sense in which velocity prediction can be considered natural for sampling.

Conclusion. While $\mathbf{v}$-prediction combined with the canonical linear schedule offers certain theoretical advantages, such as a constant target magnitude and the absence of schedule curvature, these properties do not necessarily make it universally superior. In practice, model performance depends on a range of interacting factors, including network architecture, normalization schemes, loss weighting over time, the choice of sampler and discretization steps, guidance strength, regularization strategy, data scaling, and overall training budget. Different datasets and objectives may favor alternative parameterizations or schedules, and the optimal configuration is ultimately an empirical question that should be resolved through validation and ablation studies.

6.4 Beneath It All: The Fokker–Planck Equation

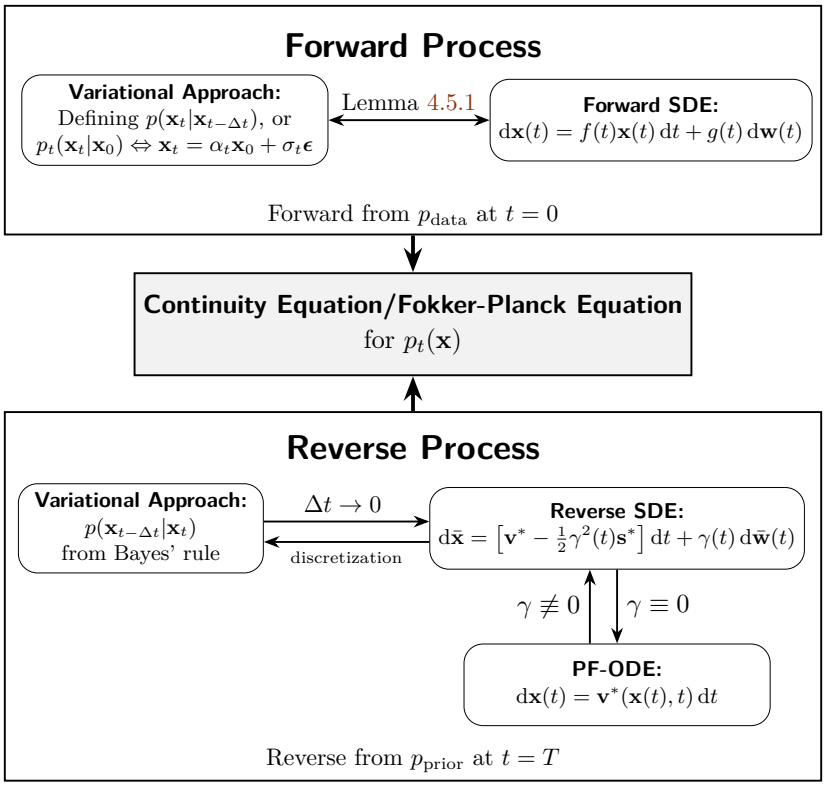


Figure 6.2: Unified perspective connecting variational, SDE, and ODE formulations through the continuity equation, where all $p_t(\mathbf{x})$ evolve under a shared dynamic. The velocity field $\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t)$ is governed by the score function $\mathbf{s}^*(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. Coefficients $f(t)$, $g(t)$, σ_t, and α_t are pre-defined time-dependent functions, and $\gamma(t)$ is a tunable time-varying hyperparameter.

Source: Created by the authors.

In this section, we show that the three main perspectives of diffusion models—variational, score-based, and flow-based—are not separate constructions but arise from a single unifying principle: the continuity (Fokker–Planck) equation that governs density evolution under a chosen forward process.

First, we recall that the analysis in Section 4.6 unifies the variational perspective, based on discrete kernels and Bayes' rule, with the score-based SDE perspective of continuous dynamics. We establish this connection by showing that variational models act as consistent discretizations of the underlying forward and reverse SDEs. Specifically, the marginal densities calculated step-by-step via the discrete kernels evolve in a manner consistent with the Fokker–Planck equation that governs the continuous-time dynamics. This confirms that the two perspectives

are fundamentally equivalent.

We then connect the flow-based and score-based views. In Section 6.4.1, we show that an ODE flow determines a density path whose marginals can always be realized by a family of stochastic processes. This places deterministic flows and stochastic SDEs within the same family.

Together, these results unify the three perspectives under one framework (see Figure 6.2). At last, we conclude this chapter in Section 6.5.

6.4.1 Connection of Flow-Based Approach and Score SDE

A remarkable aspect of diffusion model lies in how different dynamic systems, deterministic or stochastic, can trace out the same evolution of probability distributions. In this section, we reveal a natural and elegant connection between ODE-based flows of Section 5.2 and Score SDEs. Specifically, we show that the velocity field defining a generative ODE can be transformed into a stochastic counterpart that follows the same Fokker-Planck dynamics, providing a principled bridge between deterministic interpolation and stochastic sampling. This offers us a continuous family of models, ranging from ODEs to SDEs, that generate the same data distribution path.

We consider the continuous-time setup where the perturbation kernel is given by

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}),$$

where $\mathbf{x}_0 \sim p_{\text{data}}$. This conditional distribution induces a marginal density path $p_t(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}}[p(\mathbf{x}_t|\mathbf{x}_0)]$ as usual, with $p_T \approx p_{\text{prior}}$.

To match this density path, consider the ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_t(\mathbf{x}(t)), \quad t \in [0, T], \tag{6.4.1}$$

where $\mathbf{v}_t(\mathbf{x}) = \mathbb{E}\left[\alpha_t' \mathbf{x}_0 + \sigma_t' \boldsymbol{\epsilon} | \mathbf{x}\right]$ is the oracle velocity as shown in Equation (5.2.9) (noting that time is flipped to follow the diffusion model convention). Integrating equation 6.4.1 backward from $\mathbf{x}(T) \sim p_{\text{prior}}$ yields samples from p_0.

Although this ODE suffices for generating high-quality samples, incorporating stochasticity may improve sample diversity. This motivates the following question:

Question 6.4.1

Is there an SDE whose dynamics, starting from p_{prior}, yield the same marginal densities as the ODE in Equation (6.4.1)?

This statement affirms that there exists a family of reverse-time SDEs that induce the same marginal density path as the corresponding PF-ODE. The densities induced by these SDEs satisfy the same Fokker–Planck equation for this path,

and therefore their one time marginals coincide with the prescribed interpolation $\{p_t\}_{t\in[0,T]}$[2].

Proposition 6.4.1: Reverse-Time SDEs Generate the Same Marginals as Interpolations

Let $\gamma(t) \geq 0$ be an arbitrary time-dependent coefficient. Consider the reverse-time SDE, written in the original time variable t and integrated backward from T to 0,

$$\mathrm{d}\bar{\mathbf{x}}(t) = \left[\mathbf{v}^*(\bar{\mathbf{x}}(t), t) - \tfrac{1}{2}\gamma^2(t)\mathbf{s}^*(\bar{\mathbf{x}}(t), t)\right]\mathrm{d}t + \gamma(t)\,\mathrm{d}\bar{\mathbf{w}}(t), \qquad t : T \to 0. \tag{6.4.2}$$

with terminal condition $\bar{\mathbf{x}}(T) \sim p_T$. Then this process $\{\bar{\mathbf{x}}(t)\}_{t\in[0,T]}$ matches the prescribed marginals $\{p_t\}_{t\in[0,T]}$ induced by the ODE's density path. Here, $\mathbf{s}^*(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is the score function, and it is related to the velocity field $\mathbf{v}^*(\mathbf{x}, t)$ by

$$\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t), \quad \mathbf{s}^*(\mathbf{x}, t) = \frac{1}{\sigma_t}\frac{\alpha_t\mathbf{v}^*(\mathbf{x}, t) - \alpha_t'\mathbf{x}}{\alpha_t'\sigma_t - \alpha_t\sigma_t'}. \tag{6.4.3}$$

Proof for Proposition.

The reverse-time Fokker–Planck equation, written with respect to the original time variable t while the process is integrated backward, is

$$\partial_t p_t = -\nabla \cdot \left([\mathbf{v}^* - \tfrac{1}{2}\gamma^2\mathbf{s}^*]p_t\right) - \tfrac{1}{2}\gamma^2\Delta p_t.$$

Using the identity $\nabla \cdot (\mathbf{s}^* p_t) = \Delta p_t$ since $\mathbf{s}^* = \nabla \log p_t$, the second-order terms cancel, yielding

$$\partial_t p_t = -\nabla \cdot (\mathbf{v}^* p_t),$$

i.e., the first-order continuity equation associated with the PF-ODE. Hence the reverse-time SDE and the ODE induce the same marginal density path $\{p_t\}$. See Appendix A.2–A.3 of Ma *et al.* (2024). ∎

At the oracle level of Proposition 6.4.1, $\gamma(t)$ can be chosen arbitrarily without changing the prescribed marginal density path: the γ-dependent drift and diffusion terms cancel exactly in the Fokker–Planck equation. Some representative choices are:

[2]For completeness, a forward SDE representation (not needed here) is

$$\mathrm{d}\mathbf{x}(t) = f(t)\mathbf{x}(t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t),$$

with $f(t)$ and $g(t)$ related to (α_t, σ_t) via Equation (4.5.2).

- Setting $\gamma(t) = 0$ recovers the ODE in Equation (6.4.1).
- When $\gamma(t) = g(t)$, Equation (6.4.2) becomes the reverse-time SDE in Equation (4.1.6), since the oracle velocity $\mathbf{v}(\mathbf{x}, t)$ satisfies (see Proposition 6.3.1):

$$\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \tfrac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t).$$

- Other choices for $\gamma(t)$ have been explored; e.g., Ma *et al.* (2024) select $\gamma(t)$ to minimize the KL gap between p_{data} and the $t = 0$ density obtained by solving Equation (6.4.2) from $t = T$.

Following Score SDE, the trained velocity field $\mathbf{v}_{\phi^\times}(\mathbf{x}, t)$ can be converted into a parameterized score function $\mathbf{s}_{\phi^\times}(\mathbf{x}, t)$ via Equation (6.4.3). Plugging this into Equation (6.4.2) defines an *empirical reverse-time SDE*, which can be sampled by numerically integrating from $t = T$ with $\bar{\mathbf{x}}(T) \sim p_{\text{prior}}$.

This proposition highlights a remarkable flexibility of diffusion models: once a *marginal density path* $\{p_t\}_{t\in[0,T]}$ is induced by the forward process, it can be realized by an entire family of reverse-time dynamics, including the PF-ODE and the reverse-time SDEs

$$\mathrm{d}\bar{\mathbf{x}}(t) = \left[\mathbf{v}^*(\bar{\mathbf{x}}, t) - \tfrac{1}{2}\gamma^2(t)\mathbf{s}^*(\bar{\mathbf{x}}, t)\right]\mathrm{d}t + \gamma(t)\,\mathrm{d}\bar{\mathbf{w}}(t), \quad \gamma(t) \geq 0,$$

from $t = T$ to $t = 0$. Although $\gamma(t)$ changes the stochastic dynamics of individual trajectories, the corresponding Fokker–Planck equations all reduce, for the prescribed density path, to the same governing density equation:

$$\partial_t p_t = -\nabla_{\mathbf{x}} \cdot \left(\mathbf{v}^*(\mathbf{x}, t)p_t(\mathbf{x})\right).$$

Indeed, using $\mathbf{s}^*(\mathbf{x}, t) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, the additional γ-dependent drift and diffusion terms cancel at the density level. Thus, $\gamma(t)$ controls the stochasticity of sample trajectories while preserving the same oracle one-time marginals, connecting the deterministic PF-ODE with a family of stochastic SDE counterparts, as illustrated in Figure 6.2.

6.5 Closing Remarks

This chapter has served as the keystone of our theoretical exploration, synthesizing the variational, score-based, and flow-based perspectives into a single, cohesive framework. We have shown that these three seemingly distinct approaches are not merely parallel but are deeply and fundamentally interconnected.

Our unification rests on two core insights. First, we identified the secret sauce common to all frameworks: a conditioning trick that transforms an intractable marginal training objective into a tractable conditional one, enabling stable and efficient learning. Second, we established that the Fokker-Planck equation is the universal law governing the evolution of probability densities. All three perspectives, in their own way, construct a generative process that respects this fundamental dynamic.

Furthermore, we showed that the oracle noise, clean-data, score, and velocity predictions contain the same information and can be converted into one another algebraically. This oracle equivalence does not make the prediction parameterization merely cosmetic: finite-capacity networks may approximate these targets differently, their training objectives can weight errors across noise levels differently, and finite-step numerical solvers can behave differently depending on the representation in which the dynamics are approximated. The important lesson is therefore to distinguish equivalence of the underlying continuous theory from equivalence of a practical training or sampling algorithm. At the density level, however, the common principle remains: modern diffusion methods construct dynamics whose marginals transport a simple prior toward the data distribution along a prescribed probability path.

With this unified and principled foundation firmly established, we are now equipped to move from the foundational theory to the practical application and acceleration of diffusion models. The central insight that generation is equivalent to solving a differential equation provides a powerful platform for control and optimization. The subsequent parts of this monograph will leverage this unified understanding to address key practical challenges:

1. Part C will focus on improving the sampling process at inference time. We will explore how to steer the generative trajectory for controllable generation (Chapter 8) and investigate advanced numerical solvers to dramatically accelerate the slow, iterative sampling process (Chapter 9).

2. Part D will then look beyond iterative solvers to learn fast generators directly. We will examine methods that can produce high-quality samples in just one or a few steps, either through distillation from a teacher model (Chapter 10) or by training from scratch (Chapter 11).

Having unified the *what* and *why* of diffusion models, we now turn our attention to the exciting and practical frontiers of *how*.

7

(Optional) Diffusion Models and Optimal Transport

Mapping one distribution to another (with generation as a special case) is a central challenge. Flow matching addresses this by learning a time-dependent velocity field that transports mass from source to target. This naturally connects to transport theory: classical optimal transport seeks the minimal cost path between distributions, while its entropy regularized form, the Schrödinger bridge, selects the most likely controlled diffusion relative to a reference such as Brownian motion.

In this chapter we review the foundations of optimal transport, entropic optimal transport, and Schrödinger bridges as formulations of the distribution–to–distribution problem. This leads to a central question: to what extent do diffusion models realize such optimal transports? They admit two views: as *stochastic processes*, defined through forward and reverse SDEs, and as *deterministic processes*, given by PF-ODEs. The stochastic view aligns directly with entropic optimal transport, while the PF-ODE does not generally correspond to any known transport objective. This gap leaves an open question: under what conditions can diffusion models be regarded as solving an optimal transport problem?

After this chapter, you will be able to...

- distinguish a transport coupling, a deterministic transport map, a dynamic transport flow, and a stochastic Schrödinger bridge, and connect these notions conceptually to deep generative modeling;
- understand what classical OT, entropic OT, and the Schrödinger bridge problem optimize, and how these formulations are related;
- explain why a standard diffusion forward/reverse SDE does not, in general, solve a prescribed two-endpoint Schrödinger bridge problem;
- explain why a diffusion PF-ODE defines a deterministic transport map but

does not, in general, coincide with the quadratic-cost optimal transport map.

Reader route. This chapter is optional for the main diffusion-model story. Read it for a precise answer to the question "in what sense, if any, is diffusion optimal transport?"

7.1 Prologue of Distribution-to-Distribution Translation

Diffusion models fix the terminal distribution to a standard Gaussian, p_{prior}. However, many applications require *distribution-to-distribution* translation: transforming a source distribution p_{src} into a different target p_{tgt}. Examples include converting sketches to photorealistic images or translating between artistic styles.

Modern diffusion methods provide practical ways to achieve this. *One-endpoint* methods such as SDEdit (Meng *et al.*, 2021b) begin with a source image at $t = 0$, diffuse it to an intermediate step t, and then use a pre-trained diffusion model for the target domain to reverse the process. This produces an output that matches the style and content of the target distribution.

Two-endpoint methods, like Dual Diffusion Bridge (Su *et al.*, 2022), instead connect the two domains through a shared latent distribution, typically a Gaussian at $t = 1$. A forward probability–flow ODE transports samples from p_{src} into this latent space, while a reverse ODE trained on the target domain maps them back to p_{tgt}. Beyond such sampling-time approaches, the Flow Matching framework described in Section 5.2 offers a training-based alternative: it directly learns an ODE flow that continuously moves mass from p_{src} to p_{tgt}.

Crucially, transforming between distributions requires more than two separately trained models. It demands a principled mapping that aligns the dynamics at *both* endpoints and does so in the "cheapest" (cost-efficient) way.

In this section, rather than surveying the many diffusion-based translation applications, we shift our focus to the mathematical foundations of this classic distribution-to-distribution problem. In particular, we highlight optimal transport (OT) and its entropic variant, the Schrödinger bridge (SB), which have long been studied in the theoretical community as canonical formulations of cost-efficient (in mathematical sense) distributional transformation.

At its core, the fundamental question is:

Question 7.1.1

Given two probability distributions, what is the most efficient way to transform one into the other while minimizing the total cost*?*

Here, the cost, $c(\mathbf{x}, \mathbf{y})$, is a non-negative function that assigns a penalty for moving a unit of mass from a point $\mathbf{x}$ to a point $\mathbf{y}$. A common choice is the squared distance, $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$.

This section provides a brief overview to clarify how diffusion-based approaches, including flow matching, connect to classical and regularized optimal transport. The central question we aim to explore is:

Question 7.1.2

Is a diffusion model a form of optimal transport connecting p_{data} and p_{prior},

and in what sense?

To address this question, we first clarify what "optimality" means in Section 7.2. We review classical *optimal transport (OT)* in the static Monge–Kantorovich form (Equation (7.2.1)) and its dynamic Benamou–Brenier formulation (Equation (7.2.3); minimizing kinetic energy subject to the continuity equation), as well as the entropy regularized variant (*entropic OT*) in Equation (7.2.5), which is equivalent to the *Schrödinger Bridge Problem* (Equation (7.2.8)). In the dynamic view, OT induces a deterministic flow satisfying the continuity equation, whereas SB induces a controlled diffusion whose marginals evolve by the Fokker-Planck equation. We provide a high level map between these formulations in Section 7.3.

We then split the discussion into two parts. First, in Section 7.4, we explain that the fixed forward noising SDE used in standard diffusion models is not, by itself, a Schrödinger bridge between arbitrary p_{src} and p_{tgt}: the forward process is a chosen reference diffusion and both forward-in-time or reverse-time SDE do not, in general, enforce exact endpoint matching to a prescribed target. Hence it is not entropic OT optimal unless one explicitly solves the SB problem with those endpoints; while it is an optimal solution to the half-bridge problem as it is anchored with one starting point.

Second, in Section 7.5, we return to the generative setting with $p_{\text{src}} = p_{\text{prior}}$ (Gaussian) and $p_{\text{tgt}} = p_{\text{data}}$. The PF-ODE defines a deterministic map that transports p_{prior} to p_{data} by construction. However, this flow is generally not an OT map for a prescribed transport cost (e.g., quadratic W_2): it realizes one admissible deterministic coupling among many and does not minimize the Benamou–Brenier action. What follows is we discuss if the "rectify flow" procedure (Section 5.4.1) can lead to a OT map; however, in general, there is no such theoretical guarantee. Therefore, the exact characterization between diffusion model's PF-ODE map and OT remains a challenging and unsolved problem.

7.2 Taxonomy of the Problem Setups

In this section, we introduce notions of what constitutes the most "efficient" or "optimal" way to transport mass from p_{src} to p_{tgt}. These include classical optimal transport (OT) and its entropy-regularized variant, which admits an equivalent formulation known as the Schrödinger Bridge. This taxonomy provides background that will later clarify connections with diffusion models.

7.2.1 Optimal Transport (OT)

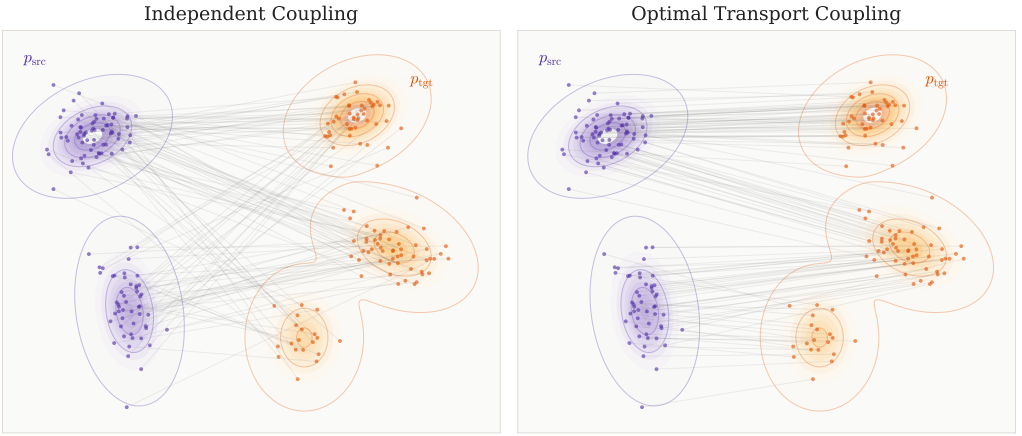


Figure 7.1: Two couplings between the same p_{src} and p_{tgt} in 2D. Left: independent (random) pairing produces crossing paths with high cost. Right: the optimal transport coupling pairs nearby mass, avoiding crossings.

Source: Created by the authors with AI-assisted coding.

Monge–Kantorovich (Static) Formulation of OT Problem. We fix a cost function $c : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ that specifies the expense of sending probability mass from $\mathbf{x}$ to $\mathbf{y}$. The goal is to transform the source distribution p_{src} into the target distribution p_{tgt} as cheaply as possible.

To even define a cost, we must know which pairs $(\mathbf{x}, \mathbf{y})$ are matched. This role is played by a *coupling*: a joint distribution γ on $\mathbb{R}^D \times \mathbb{R}^D$ whose marginals are p_{src} and p_{tgt}. In other words, sampling $(\mathbf{x}, \mathbf{y}) \sim \gamma$ means we match $\mathbf{x}$ from the source with $\mathbf{y}$ from the target. If γ admits a density $\gamma(\mathbf{x}, \mathbf{y})$ with respect to Lebesgue measure, the marginal constraints read

$$\int_{\mathbb{R}^D} \gamma(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{y} = p_{\text{src}}(\mathbf{x}), \qquad \int_{\mathbb{R}^D} \gamma(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{x} = p_{\text{tgt}}(\mathbf{y}).$$

That is, integrating out $\mathbf{y}$ recovers the source density in $\mathbf{x}$, while integrating out $\mathbf{x}$ recovers the target density in $\mathbf{y}$.

We give two standard examples for illustration:

1. **Discrete Supports.** If p_{src} and p_{tgt} are supported on finitely many points, a coupling is represented by a nonnegative matrix (γ_{ij}) whose row sums equal $p_{\text{src}}(i)$ and column sums equal $p_{\text{tgt}}(j)$. Each entry γ_{ij} is the amount of mass sent from i to j.

2. **Deterministic Map.** If there exists a measurable map $\mathbf{T}$ with $\mathbf{T}_{\#}p_{\text{src}} = p_{\text{tgt}}$, then $\gamma = (\mathbf{I}, \mathbf{T})_{\#}p_{\text{src}}$ is a deterministic coupling that moves each point $\mathbf{x}$ directly to $\mathbf{T}(\mathbf{x})$.

Once a coupling γ is fixed, the transport cost is simply the average unit cost under this plan:

$$\int c(\mathbf{x}, \mathbf{y})\ \mathrm{d}\gamma(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}[c(\mathbf{x}, \mathbf{y})].$$

In the discrete case, this reduces to $\sum_{i,j} c_{ij}\gamma_{ij}$, whereas in the continuous setting it becomes a double integral. In what follows, we will focus only on the continuous case.

The optimal transport problem is then to choose, among all admissible couplings, the one that minimizes this expected cost.

$$\mathrm{OT}(p_{\text{src}}, p_{\text{tgt}}) := \inf_{\gamma\in\Gamma(p_{\text{src}},p_{\text{tgt}})} \int c(\mathbf{x}, \mathbf{y})\ \mathrm{d}\gamma(\mathbf{x}, \mathbf{y}), \tag{7.2.1}$$

where the feasible set simply enforces the marginal, or mass-conservation, constraints:

$$\Gamma(p_{\text{src}}, p_{\text{tgt}}) = \Big\{\gamma \in \mathcal{P}(\mathbb{R}^D \times \mathbb{R}^D) : \int \gamma(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{y} = p_{\text{src}}(\mathbf{x}),\ \int \gamma(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{x} = p_{\text{tgt}}(\mathbf{y})\Big\},$$

where $\mathcal{P}(\mathbb{R}^D \times \mathbb{R}^D)$ denotes the set of all probability measures on $\mathbb{R}^D \times \mathbb{R}^D$.

A Special Case: Wasserstein-2 Distance. The Wasserstein-2 distance is a special case of the Monge–Kantorovich problem with the quadratic cost $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^2$. It measures the distance between two probability distributions as follows:

$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) := \inf_{\gamma\in\Gamma(p_{\text{src}},p_{\text{tgt}})} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}\left[\|\mathbf{x} - \mathbf{y}\|^2\right].$$

Under suitable assumptions on p_{src} and p_{tgt}, Brenier's theorem (see Theorem 7.1)[1] guarantees that the optimal coupling γ for the quadratic cost is concentrated on the graph of a deterministic map $\mathbf{T} : \mathbb{R}^D \to \mathbb{R}^D$. Consequently, the

[1] Brenier's theorem is about the existence and structure of the optimal transport map for quadratic cost. In particular, if p_{src} does not give mass to sets of dimension at most $D-1$, then an optimal transport map $\mathbf{T}^*$ uniquely exists.

Wasserstein-2 distance can be equivalently expressed as[2]:

$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) = \inf_{\substack{\mathbf{T}:\mathbb{R}^D \to \mathbb{R}^D, \\ \text{s.t. } \mathbf{T}\#p_{\text{src}} = p_{\text{tgt}}}} \mathbb{E}_{\mathbf{x}\sim p_{\text{src}}} \left[\|\mathbf{T}(\mathbf{x}) - \mathbf{x}\|^2 \right]. \tag{7.2.2}$$

Here, $\mathbf{T}\#p_{\text{src}} = p_{\text{tgt}}$ means that $\mathbf{T}$ pushes p_{src} forward to p_{tgt}, i.e., $\mathbf{T}(\mathbf{x}) \sim p_{\text{tgt}}$ for $\mathbf{x} \sim p_{\text{src}}$.

Thus, the Wasserstein-2 distance represents the minimal expected squared transport cost among all couplings or transport maps that match the given marginals. The optimal transport map denoted by $\mathbf{T}^*(\mathbf{x})$, known as the *Monge map*, yields the most efficient way to transform p_{src} into p_{tgt}.

Benamou–Brenier (Dynamic) Formulation of OT. Instead of mapping distributions directly in a static manner, as in the Monge–Kantorovich formulation, transport can also be modeled as a continuous-time flow:

$$p_0 := p_{\text{src}} \to p_t \to p_1 := p_{\text{tgt}}, \quad t \in [0, 1].$$

This dynamic formulation of optimal transport, introduced by Benamou and Brenier (2000), seeks a smooth velocity field $\mathbf{v}_t(\mathbf{x})$ that describes how mass in $p_t(\mathbf{x})$ evolves over time.

The Benamou–Brenier formulation[3] shows that, for the quadratic cost $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$ (i.e., the $\mathcal{W}_2$ distance), the optimal value of the static OT problem in Equation (7.2.1) is equal to the optimal value of the kinetic energy minimization problem:

$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) = \min_{\substack{(p_t, \mathbf{v}_t) \text{ s.t. } \partial_t p_t + \nabla\cdot(p_t \mathbf{v}_t) = 0, \\ p_0 = p_{\text{src}},\ p_1 = p_{\text{tgt}}}} \int_0^1 \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2 p_t(\mathbf{x})\, \mathrm{d}\mathbf{x}\, \mathrm{d}t \tag{7.2.3}$$

where p_t is a probability distribution on $\mathbb{R}^D$ for each $t \in [0, 1]$. In particular, The optimal transport flow $p_t(\mathbf{x})$ follows *McCann's displacement interpolation*:

$$\mathbf{T}_t^*(\mathbf{x}) = (1 - t)\mathbf{x} + t\mathbf{T}^*(\mathbf{x}),$$

where $\mathbf{T}^*(\mathbf{x})$ is the OT map that transports p_{src} to p_{tgt}. This linear interpolation moves mass along straight lines with constant velocity: $p_t = \mathbf{T}_t^*\#p_{\text{src}}$ for each $t \in [0, 1]$.

[2] There are three commonly used formulations of the $\mathcal{W}_2$ distance: the Monge formulation (based on an optimal transport map), the Kantorovich formulation (based on couplings), and the Benamou–Brenier dynamic formulation (see Equation (7.2.3)). These are equivalent under appropriate regularity conditions.

[3] Benamou–Brenier formulation describes how to compute the $\mathcal{W}_2$ distance by minimizing kinetic energy over continuous paths of measures and velocities.

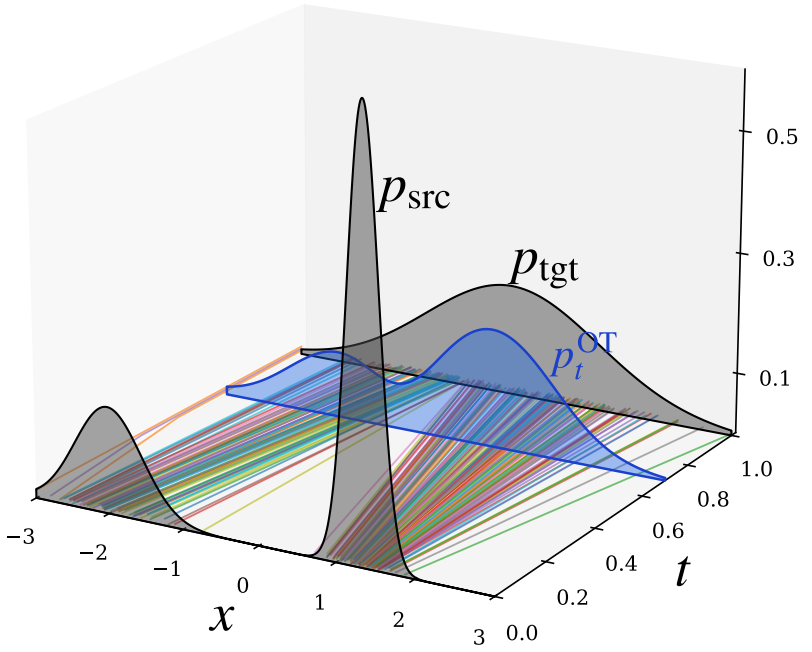


Figure 7.2: Illustration of dynamic view of OT. The interpolation p_t^{OT} evolves continuously in time, providing the least-cost transport plan that deterministically maps p_{src} to p_{tgt} (the McCann's displacement interpolation).

Source: Created by the authors.

The optimal transport map $\mathbf{T}^*$ satisfies the *Monge–Ampère equation*:

$$p_{\text{tgt}}\left(\nabla\psi(\mathbf{x})\right)\det\left(\nabla^2\psi(\mathbf{x})\right) = p_{\text{src}}(\mathbf{x}), \tag{7.2.4}$$

where $\mathbf{T}^*(\mathbf{x}) = \nabla\psi(\mathbf{x})$ for some convex function $\boldsymbol{\psi}$ by Brenier's theorem. However, this nonlinear PDE is typically intractable for explicit solutions. Note that this is precisely the change-of-variables relation used by normalizing flows (c.f., Equation (5.0.1)): flows parametrize an invertible transport map with a tractable Jacobian determinant, but do not in general impose the gradient-of-potential structure $\mathbf{T}^* = \nabla\psi$; consequently, a trained flow can differ substantially from the Brenier/OT map.

7.2.2 Entropy-Regularized Optimal Transport (EOT)

To motivate EOT concretely, consider empirical distributions built from samples. Suppose p_{src} is supported on points $\{\mathbf{x}^{(i)}\}_{i=1}^n \subset \mathbb{R}^D$ with weights a_i, and p_{tgt} on

$\{\mathbf{y}^{(j)}\}_{j=1}^{n} \subset \mathbb{R}^D$ with weights b_j. A coupling is then an $n \times n$ nonnegative matrix $\gamma = (\gamma_{ij})$ whose row sums match a and column sums match b. Each entry γ_{ij} represents the amount of mass transported from $\mathbf{x}^{(i)}$ to $\mathbf{y}^{(j)}$[4].

Why Regularize OT? Classical OT in this discrete setting (obtained by taking counting measures in the continuous formulation Equation (7.2.1)) reduces to minimizing

$$\min_{\gamma=(\gamma_{ij})} \sum_{i,j} C_{ij}\, \gamma_{ij},$$

over all feasible couplings $\gamma = (\gamma_{ij})$, where $C_{ij} = c(\mathbf{x}^{(i)}, \mathbf{y}^{(j)})$ is the cost of moving one unit of mass from source point $\mathbf{x}^{(i)}$ to target point $\mathbf{y}^{(j)}$, for a prescribed ground cost $c : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_{\geq 0}$ (e.g., $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$).

Two main issues arise:

1. **Non-Uniqueness and Instability:** The minimizer γ^* need not be unique. For example, if two transport plans achieve the same minimum cost, the solver may select either one. Consequently, small changes in the inputs (a, b, C) (such as moving a sample, adjusting weights, or slightly perturbing costs) can cause abrupt jumps in the solution.

2. **High Computational Cost:** The problem is a linear program with n^2 variables and $2n$ constraints. Practical solvers (e.g., Hungarian algorithm (Kuhn, 1955; Munkres, 1957), network simplex (Peyré, Cuturi, *et al.*, 2019)) typically scale as $\mathcal{O}(n^3)$, which is infeasible for large n.

To overcome these bottlenecks, EOT objective function introduces a regularization term to the classical OT problem, controlled by a parameter $\varepsilon > 0$:

$$\mathrm{EOT}_\varepsilon(p_{\mathrm{src}}, p_{\mathrm{tgt}}) := \min_{\gamma \in \Gamma(p_{\mathrm{src}}, p_{\mathrm{tgt}})} \int c(\mathbf{x}, \mathbf{y})\, \mathrm{d}\gamma(\mathbf{x}, \mathbf{y}) + \varepsilon \mathcal{D}_{\mathrm{KL}}\left(\gamma \| M\right). \tag{7.2.5}$$

The reference measure M is typically chosen as the product of the marginals, $p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}$. The KL divergence term is directly related to the Shannon entropy of the transport plan γ:

$$\mathcal{D}_{\mathrm{KL}}(\gamma \,\|\, p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}) = -\mathcal{H}(\gamma) + \text{Constant},$$

where $\mathcal{H}(\gamma) := -\int \gamma(\mathbf{x}, \mathbf{y}) \log \gamma(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{y}$.

The addition of this regularization term yields several theoretical and practical advantages, which we briefly outline below:

[4] Empirical (discrete) measures provide a principled proxy for continuous distributions. When the ground cost is $c(x, y) = d(x, y)^p$ (so the OT value equals W_p^p) and the measures have finite pth moments, the empirical measures converge to the population in W_p with quantitative rates; see Fournier and Guillin (2015) and the overview in Peyré, Cuturi, *et al.* (2019).

Why Entropy Regularizer Helps?

1. **Mass Spreading.** Since $t \mapsto t \log t$ is convex and grows rapidly for large t, minimizing $\int \gamma \log \gamma$ penalizes *peaky* couplings (some $\gamma(\mathbf{x}, \mathbf{y})$ very large, most near zero). For fixed total mass $\int \gamma = 1$, it favors plans where $\gamma(\mathbf{x}, \mathbf{y})$ is more evenly distributed over $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^D \times \mathbb{R}^D$. Equivalently, maximizing Shannon entropy promotes higher "uncertainty" (diffuseness).

2. **Strict Convexity and Uniqueness.** Because $\mathcal{H}$ is strictly concave, the objective in Equation (7.2.5) is strictly convex in γ, yielding a *unique* minimizer γ_ε^* that depends continuously on $(p_{\text{src}}, p_{\text{tgt}}, c)$.

3. **Sinkhorn Form and Positivity.** Under mild conditions[5], the optimizer has the *Schrödinger/Sinkhorn form*

$$\gamma_\varepsilon^*(\mathbf{x}, \mathbf{y}) = u(\mathbf{x}) \exp\left(-\tfrac{c(\mathbf{x},\mathbf{y})}{\varepsilon}\right) v(\mathbf{y}) p_{\text{src}}(\mathbf{x}) p_{\text{tgt}}(\mathbf{y}),$$

   for positive scaling functions u, v (unique up to a global factor). In practice, the continuous formulation is approximated with finite samples, reducing EOT to a finite (sampled) Sinkhorn iteration. The entropic objective is strictly convex, and the scaling (Sinkhorn/IPFP) algorithm solves it efficiently (Sinkhorn, 1964; Cuturi, 2013). For a dense problem with n support points per marginal (an $n \times n$ kernel), each Sinkhorn iteration costs $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ memory, making the method more scalable and practical (Altschuler *et al.*, 2017).

4. **Limits in ε.** As $\varepsilon \to 0$, the optimal plan γ_ε^* becomes increasingly concentrated, approaching a (possibly singular) classical OT coupling (we will revisit this connection in Section 7.3.2). As ε increases, γ_ε^* gradually spreads out and approaches the independent coupling $p_{\text{src}} \otimes p_{\text{tgt}}$.

7.2.3 Schrödinger Bridge (SB)

KL Formulation of SB. The Schrödinger Bridge (SB) problem, introduced by Erwin Schrödinger in the 1930s, asks the following question. Suppose particles move according to some simple reference dynamics, such as Brownian motion. Now imagine that we observe the particles at two times: at $t = 0$ their distribution is p_{src}, and at $t = 1$ it is p_{tgt}. Among all possible stochastic processes that connect these two distributions, which one deviates the least from the reference dynamics? Here "deviation" is measured by the KL divergence, so the solution to the SB

[5] We assume that $c < \infty$ holds $p_{\text{src}} \otimes p_{\text{tgt}}$-almost everywhere, and that the marginal kernel integrals are finite and positive. For simplicity, we focus on the case where γ_ε^*, p_{src}, and p_{tgt} admit densities with respect to the Lebesgue measure.

problem is the most likely way to deform Brownian motion into a process that satisfies the prescribed boundary conditions.

To make this precise, let $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t\in[0,T]}$ denote a complete trajectory of the process. We write P for the *law of trajectories*, that is, the probability distribution over entire sample paths. The time-t marginal of P is denoted by p_t (or P_t), which describes the distribution of the state $\mathbf{x}_t$ at a single time. Formally, for a measurable set $A \subseteq \mathbb{R}^D$,

$$p_t(A) = P(\mathbf{x}_t \in A).$$

In other words, p_t can be viewed as the empirical distribution obtained by sampling many full trajectories from P and then collecting the states at time t—for instance, as a histogram if the state is one-dimensional.

Consider a reference diffusion $\{\mathbf{x}_t\}_{t\in[0,T]}$ governed by the SDE

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\ \mathrm{d}t + g(t)\ \mathrm{d}\mathbf{w}_t, \tag{7.2.6}$$

where $\mathbf{f}\colon \mathbb{R}^D \times [0,T] \to \mathbb{R}^D$, $g\colon [0,T] \to \mathbb{R}$, and $\{\mathbf{w}_t\}_{t\in[0,T]}$ is a standard Brownian motion. Let R denote the *path law* (joint distribution) of the full trajectory $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t\in[0,T]}$, and let R_0 denote the initial marginal of the reference process; this R will serve as the *reference* trajectory distribution.

With this notation, the SB problem seeks a trajectory law P that is closest to R in KL divergence while matching the prescribed endpoint marginals:

$$\mathrm{SB}(p_{\mathrm{src}}, p_{\mathrm{tgt}}) := \min_P \mathcal{D}_{\mathrm{KL}}(P\|R) \quad \text{s.t.} \quad P_0 = p_{\mathrm{src}},\ \ P_T = p_{\mathrm{tgt}}. \tag{7.2.7}$$

The optimizer P^* depends on the chosen reference process R.

Stochastic Control View of SB. Rather than optimizing over arbitrary path distributions P in Equation (7.2.7), a more tractable approach is to take the reference dynamics as an anchor and allow it to drift. This is done by introducing a time-dependent drift $\mathbf{v}_t(\mathbf{x}_t)$, which perturbs the reference process and generates a family of candidate trajectory distributions. The resulting dynamics take the form of a *controlled diffusion*:

$$\mathrm{d}\mathbf{x}_t = \big[\mathbf{f}(\mathbf{x}_t, t) + \mathbf{v}_t(\mathbf{x}_t)\big]\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t,$$

where $\mathbf{v}_t : \mathbb{R}^D \to \mathbb{R}^D$ is the drift to be optimized later (Equation (7.2.8)). Under standard integrability conditions (e.g., Novikov's condition) and by Girsanov's theorem (see Section C.2.1), the KL divergence between the controlled law P and the reference R admits the dynamic (kinetic) form

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(P\|R) &= \mathcal{D}_{\mathrm{KL}}(p_{\mathrm{src}}\|R_0) + \mathbb{E}_P\left[\frac{1}{2}\int_0^T \frac{\|\mathbf{v}_t(\mathbf{x}_t)\|^2}{g^2(t)}\,\mathrm{d}t\right] \\ &= \mathcal{D}_{\mathrm{KL}}(p_{\mathrm{src}}\|R_0) + \frac{1}{2}\int_0^T \int_{\mathbb{R}^D} \frac{\|\mathbf{v}_t(\mathbf{x})\|^2}{g^2(t)} p_t(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t.\end{aligned}$$

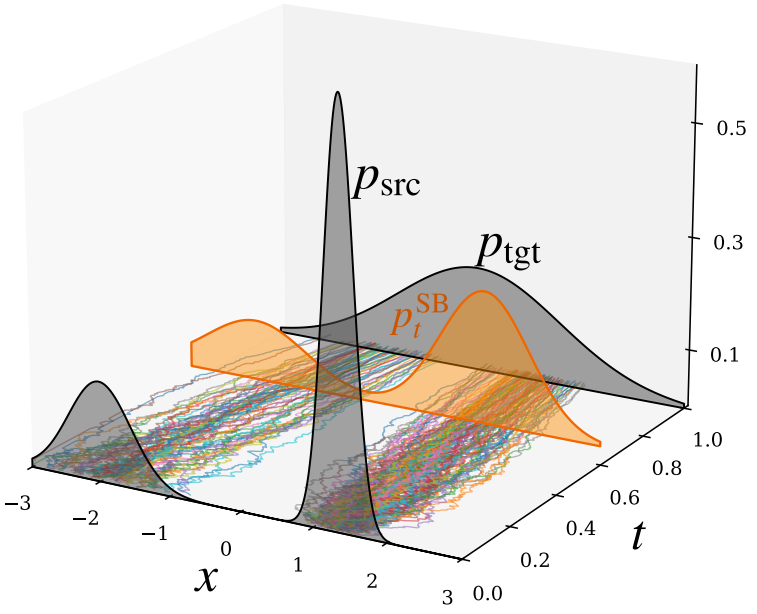


Figure 7.3: Illustration of stochastic control view of SB. The bridge seeks the stochastic path that deviates least from the reference while connecting p_{src} and p_{tgt}.

Source: Created by the authors.

where p_t is the time–t marginal of $\mathbf{x}_t$ under the controlled process. The second equality follows from the law of total expectation. Since $P_0 = p_{\text{src}}$ is fixed in the SB problem, the first term is constant with respect to the control and may therefore be omitted from the optimization. In particular, it vanishes when $R_0 = p_{\text{src}}$.

Hence, the SB problem can be reformulated as minimizing the expected control energy over all admissible drifts $\mathbf{v}_t$ that steer the process from p_{src} at $t = 0$ to p_{tgt} at $t = T$ (Dai Pra, 1991; Pra and Pavon, 1990; Pavon and Wakolbinger, 1991; Chen *et al.*, 2016). Since the initial marginal is fixed to p_{src}, we may take the reference process to have $R_0 = p_{\text{src}}$ without changing the optimizer; then $\mathcal{D}_{\text{KL}}(p_{\text{src}} \| R_0) = 0$. This leads to the *stochastic control formulation*:

$$\text{SB}(p_{\text{src}}, p_{\text{tgt}}) = \min_{\substack{\mathbf{v}_t \text{ s.t. } \mathrm{d}\mathbf{x}_t = [\mathbf{f}(\mathbf{x}_t,t)+\mathbf{v}_t(\mathbf{x}_t)]\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t, \\ \mathbf{x}_0 \sim p_{\text{src}},\, \mathbf{x}_T \sim p_{\text{tgt}}}} \frac{1}{2} \int_0^T \int_{\mathbb{R}^D} \frac{\|\mathbf{v}_t(\mathbf{x})\|^2}{g^2(t)} p_t(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t. \tag{7.2.8}$$

Importantly, the endpoint distributions p_{src} and p_{tgt} are arbitrary; the control $\mathbf{v}_t$ is chosen precisely to "bridge" the reference dynamics between these marginals while staying as close as possible (in KL divergence) to the reference process R.

A Special Brownian Reference. Equation (7.2.8) resembles the Benamou–Brenier formulation of OT in Equation (7.2.3), especially when the reference process R^ε (with $\varepsilon > 0$) is chosen to be a Brownian motion[6]:

$$\mathrm{d}\mathbf{x}_t = \sqrt{\varepsilon}\, \mathrm{d}\mathbf{w}_t,$$

so that $\mathbf{f} \equiv \mathbf{0}$ and $g(t) \equiv \sqrt{\varepsilon}$.

In this setting, the SB problem seeks a path distribution P that stays closest (in KL divergence) to the Brownian reference R^ε, while matching the endpoint marginals:

$$\mathrm{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) := \min_P \mathcal{D}_{\mathrm{KL}}(P \| R^\varepsilon) \quad \text{s.t.} \quad P_0 = p_{\text{src}}, \quad P_T = p_{\text{tgt}}. \tag{7.2.9}$$

The equivalent stochastic control formulation then becomes

$$\mathrm{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) = \min_{\substack{\mathbf{v}_t \text{ s.t. } \mathrm{d}\mathbf{x}_t = \mathbf{v}_t(\mathbf{x}_t)\, \mathrm{d}t + \sqrt{\varepsilon}\, \mathrm{d}\mathbf{w}_t, \\ \mathbf{x}_0 \sim p_{\text{src}},\ \mathbf{x}_T \sim p_{\text{tgt}}}} \frac{1}{2\varepsilon} \int_0^T \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2\, p_t(\mathbf{x})\, \mathrm{d}\mathbf{x}\, \mathrm{d}t, \tag{7.2.10}$$

where p_t denotes the time–t marginal of diffusion driven by the drift $\mathbf{v}_t$:

$$\mathrm{d}\mathbf{x}_t = \mathbf{v}_t(\mathbf{x}_t)\, \mathrm{d}t + \sqrt{\varepsilon}\, \mathrm{d}\mathbf{w}_t.$$

Why We Need to Specify a Reference Distribution? Classical OT does not require a reference path distribution because the prescribed ground cost already tells us how expensive it is to pair a source point with a target point. The optimization therefore compares admissible couplings directly through this cost. In the SB problem, however, we compare entire stochastic trajectories rather than only their endpoints. We therefore need a reference path measure R that specifies what the uncontrolled dynamics would normally look like. The quantity $\mathcal{D}_{\mathrm{KL}}(P\|R)$ then measures how much an admissible process P must deviate from these reference dynamics in order to satisfy the prescribed endpoint distributions.

[6] This resemblance can be made precise. In an equivalent time-symmetric fluid-dynamic formulation, if $\tilde{\mathbf{v}}_t$ denotes the current velocity satisfying the continuity equation $\partial_t \rho_t + \nabla\cdot(\rho_t \tilde{\mathbf{v}}_t) = 0$, then the Brownian-reference SB objective (Equation (7.2.10)) can be written as a Benamou–Brenier-type kinetic term, up to a conventional constant factor, plus a Fisher information term:

$$\int_0^T \int \left[\frac{1}{2} \|\tilde{\mathbf{v}}_t(\mathbf{x})\|^2 + \frac{\varepsilon^2}{8} \|\nabla_{\mathbf{x}} \log \rho_t(\mathbf{x})\|^2 \right] \rho_t(\mathbf{x})\, \mathrm{d}\mathbf{x}\, \mathrm{d}t.$$

Thus, in the Brownian-reference / quadratic-cost case, entropic OT can be viewed as a Fisher-information-regularized version of dynamic OT; see Problem 4.6 in Chen *et al.* (2021).

Coupled PDE Characterization. A convenient way to describe the SB solution is through two space–time potentials $\Psi(x,t)$ and $\widehat{\Psi}(x,t)$. Let p_t^{SB} denote the marginal at time $t \in [0,T]$ of the optimal trajectory law P^* in Equation (7.2.7). Then one has the symmetric factorization (Dai Pra, 1991)

$$p_t^{\mathrm{SB}}(x) = \Psi(x,t)\widehat{\Psi}(x,t), \tag{7.2.11}$$

where Ψ and $\widehat{\Psi}$ solve the (linear) *Schrödinger system* (Caluya and Halder, 2021; Chen *et al.*, 2021; Chen *et al.*, 2022):

$$\begin{aligned} \frac{\partial \Psi}{\partial t}(\mathbf{x},t) &= -\nabla_{\mathbf{x}}\Psi(\mathbf{x},t)\cdot\mathbf{f}(\mathbf{x},t) - \frac{g^2(t)}{2}\,\Delta_{\mathbf{x}}\Psi(\mathbf{x},t), \\ \frac{\partial \widehat{\Psi}}{\partial t}(\mathbf{x},t) &= -\nabla_{\mathbf{x}}\cdot\big(\widehat{\Psi}(\mathbf{x},t)\,\mathbf{f}(\mathbf{x},t)\big) + \frac{g^2(t)}{2}\,\Delta_{\mathbf{x}}\widehat{\Psi}(\mathbf{x},t) \\ \text{subject to}\quad \Psi(\mathbf{x},0)\,\widehat{\Psi}(\mathbf{x},0) &= p_{\mathrm{src}}(\mathbf{x}), \quad \Psi(\mathbf{x},T)\,\widehat{\Psi}(\mathbf{x},T) = p_{\mathrm{tgt}}(\mathbf{x}). \end{aligned} \tag{7.2.12}$$

Forward-Time Schrödinger Bridge SDE. Once Ψ is known, the optimal dynamics is the reference diffusion tilted by the space–time factor Ψ:

$$\mathrm{d}\mathbf{x}_t = \big[\mathbf{f}(\mathbf{x}_t,t) + g^2(t)\,\nabla_{\mathbf{x}}\log\Psi(\mathbf{x}_t,t)\big]\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t, \quad \mathbf{x}_0 \sim p_{\mathrm{src}}. \tag{7.2.13}$$

Let Q denote the trajectory law of Equation (7.2.13) (so $Q_0 = p_{\mathrm{src}}$ and $Q_T = p_{\mathrm{tgt}}$ by Equations (7.2.11) and (7.2.12)). Then $Q = P^*$ and the minimizer $\mathbf{v}^*$ to Equation (7.2.8) is (see (Chen *et al.*, 2021)'s Section 4.6):

$$\mathbf{v}_t^*(\mathbf{x}) = g^2(t)\nabla_{\mathbf{x}}\log\Psi(\mathbf{x},t).$$

That is, drift correction $g^2\nabla_{\mathbf{x}}\log\Psi$ is precisely the minimal KL perturbation of the reference needed to match the endpoint marginals.

Reverse-Time Schrödinger Bridge SDE. The same optimal path law can be generated reverse in time. A convenient way to see the form of the reverse-time drift is to conceptually use the standard time-reversal identity for diffusions:

$$\mathbf{b}^-(\mathbf{x},t) = \mathbf{b}^+(\mathbf{x},t) - g^2(t)\nabla_{\mathbf{x}}\log p_t^{\mathrm{SB}}(\mathbf{x}),$$

where $\mathbf{b}^+ = \mathbf{f} + g^2\nabla\log\Psi$ and $p_t = \Psi\,\widehat{\Psi}$. This gives

$$\mathbf{b}^-(\mathbf{x},t) = \mathbf{f}(\mathbf{x},t) - g^2(t)\nabla_{\mathbf{x}}\log\widehat{\Psi}(\mathbf{x},t).$$

Thus the reverse-time SDE reads

$$\mathrm{d}\mathbf{x}_t = \Big[\mathbf{f}(\mathbf{x}_t,t) - g^2(t)\nabla_{\mathbf{x}}\log\widehat{\Psi}(\mathbf{x}_t,t)\Big]\,\mathrm{d}t + g(t)\;\mathrm{d}\bar{\mathbf{w}}_t, \quad \mathbf{x}_T \sim p_{\mathrm{tgt}}. \tag{7.2.14}$$

Equivalently, reparametrizing time by $\mathbf{y}_\tau := \mathbf{x}_{T-\tau}$ so that τ increases from 0 to T. Then $\mathbf{y}_\tau$ evolves forward in τ from $\mathbf{y}_0 \sim p_{\mathrm{tgt}}$ as

$$\mathrm{d}\mathbf{y}_\tau = \big[-\mathbf{f}(\mathbf{y}_\tau, T-\tau) + g^2(T-\tau)\nabla_{\mathbf{y}}\log\widehat{\Psi}(\mathbf{y}_\tau, T-\tau)\big]\,\mathrm{d}\tau + g(T-\tau)\,\mathrm{d}\mathbf{w}_\tau. \tag{7.2.15}$$

Both the forward and reverse descriptions yield the same optimal path law P^* which are linked by

$$\nabla \log p_t^{\mathrm{SB}} = \nabla \log \Psi + \nabla \log \widehat{\Psi}, \qquad \mathbf{b}^- = \mathbf{b}^+ - g^2 \nabla \log p_t^{\mathrm{SB}},$$

so their marginals coincide with p_t^{SB} at every time. The additional drift terms $g^2 \nabla \log \Psi$ (forward) and $-g^2 \nabla \log \widehat{\Psi}$ (reverse-time) act as control forces that steer the reference diffusion to match the endpoint marginals while staying closest to the reference in relative entropy.

Practical Obstacles to the Coupled PDE Approach. To construct the generative process based on Equation (7.2.14), one must solve the coupled PDEs in Equation (7.2.12) to obtain the backward Schrödinger potential $\widehat{\Psi}$. However, these PDEs are notoriously difficult to solve, even in low-dimensional settings, which makes their direct application in generative modeling challenging. To circumvent this, several works have proposed alternative strategies: leveraging Score SDE techniques to iteratively solve each half-bridge problem ($p_{\mathrm{tgt}} \leftarrow p_{\mathrm{src}}$ and $p_{\mathrm{tgt}} \rightarrow p_{\mathrm{src}}$) (De Bortoli *et al.*, 2021); optimizing surrogate likelihood bounds (Chen *et al.*, 2022; Liu *et al.*, 2023); or designing simulation-free training based on an analytical solution of the posterior $\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_T$ for sample pairs $(\mathbf{x}_0, \mathbf{x}_T) \sim p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}$ (Liu *et al.*, 2023). We do not delve into the technical details here but briefly discuss the connection between diffusion models and SB in Section 7.4.

7.2.4 Global Pushforwards and Local Dynamics: An OT Analogy for DGMs

From the optimal-transport viewpoint (in Equation (7.2.1)), one can leverage deep generative models to learn a transport (pushforward) map from a simple prior to the data, i.e., $\mathbf{G}_{\phi\#}p_{\mathrm{prior}} \approx p_{\mathrm{data}}$. Although $\mathbf{G}_\phi$ generally does not coincide with the optimal transport map (except in works (Genevay *et al.*, 2018; Onken *et al.*, 2021) that impose an OT objective under suitable conditions), the Benamou–Brenier formulation (in Equation (7.2.3)) provides a complementary, dynamic perspective. Rather than directly learning a single global map, it describes transport as a continuous flow generated by a time-dependent local vector field, tracing a smooth path between p_{prior} and p_{data}. This dynamic formulation parallels the relationship between the static Schrödinger Bridge problem (in Equation (7.2.7)) and its stochastic-control counterpart (in Equation (7.2.8)), where the optimal coupling is realized as a controlled diffusion process. A similar analogy emerges in generative modeling: standard DGMs such as GANs or VAEs learn a global pushforward map, whereas diffusion models learn a time-dependent local vector field that drives the generative dynamics.

7.3 Relationship of Variant Optimal Transport Formulations

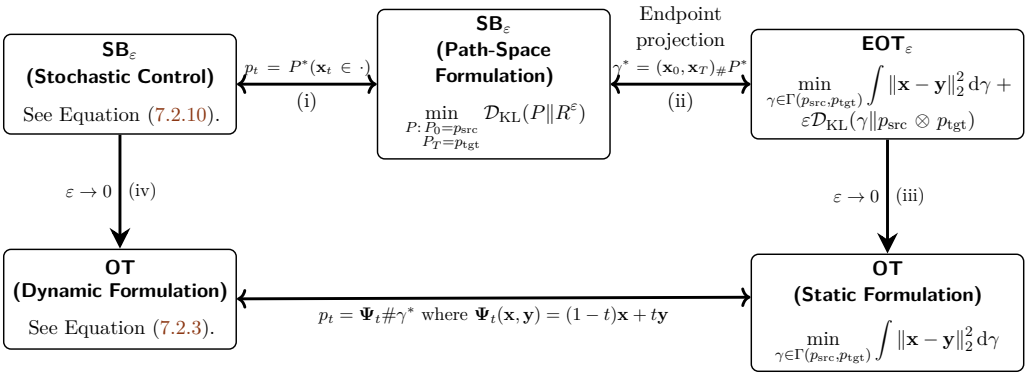


Figure 7.4: Relationship between variants of optimal transport with $c(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2$ and Reference R^ε in SB. We summarize the relationships: (i) the stochastic-control and path-space KL formulations of SB$_\varepsilon$ are equivalent; (ii) for the Brownian reference with the corresponding quadratic-cost normalization, the endpoint projection of SB$_\varepsilon$ yields EOT$_\varepsilon$; (iii) EOT$_\varepsilon$ converges to static OT as $\varepsilon \to 0$; and (iv) the appropriately rescaled Brownian-reference Schrödinger problem converges to dynamic OT as $\varepsilon \to 0$.

Source: Created by the authors.

Before delving into the technical details, it is helpful to clarify how the different formulations of optimal transport and its entropic regularizations are connected. At a high level, these problems can be viewed as related (see Figure 7.4 for their diagram for connection):

(i) **Path-Space SB $\iff$ Stochastic-Control SB.** The path-space KL formulation of SB$_\varepsilon$ and its stochastic-control formulation describe the same bridge problem relative to the chosen reference process (see Section 7.2.3).

(ii) **Brownian-Reference SB $\iff$ EOT$_\varepsilon$.** For the Brownian reference

$$\mathrm{d}\mathbf{x}_t = \sqrt{\varepsilon}\,\mathrm{d}\mathbf{w}_t,$$

with the corresponding quadratic-cost normalization, the endpoint coupling induced by the optimal SB path law solves the associated EOT$_\varepsilon$ problem (see Section 7.3.1).

(iii) **EOT$_\varepsilon$ $\longrightarrow$ Static OT.** As $\varepsilon \to 0$, the entropy-regularized endpoint coupling approaches the corresponding classical OT problem, and the optimal values converge to the OT value under the assumptions discussed in Section 7.3.2.

(iv) **SB$_\varepsilon$ $\longrightarrow$ Dynamic OT.** Under the corresponding small-noise scaling, the Brownian-reference Schrödinger problem converges as $\varepsilon \to 0$ to the Benamou–Brenier dynamic formulation of OT (see Section 7.3.3).

Together, these non-trivial relationships provide a compact view across stochastic control, entropy-regularized, and classical OT frameworks.



Figure 7.5: Effect of the regularization parameter ε on entropic optimal transport (connections (iii) and (iv)). As ε increases, two things change simultaneously: the coupling γ_ε^* spreads mass to more distant targets, and the transport paths become increasingly stochastic. Left ($\varepsilon = 0.01$): paths are nearly straight and each source point maps to a single nearby target, recovering classical OT. Center ($\varepsilon = 0.5$): paths acquire visible fluctuations and the coupling begins to spread. Right ($\varepsilon = 2.5$): paths are highly stochastic, approaching the Schrödinger bridge regime; in the limit $\varepsilon \to \infty$, the coupling degenerates to the independent product $p_{\text{src}} \otimes p_{\text{tgt}}$. The couplings are computed via the Sinkhorn algorithm (Hungarian algorithm for $\varepsilon \to 0$); conditional on endpoints, the paths are Brownian bridges with diffusion coefficient $\sigma = \sqrt{\varepsilon/2}$ from the Mikami duality (Section 7.3.1).

Source: Created by the authors with AI-assisted coding.

7.3.1 SB and EOT are (Dual) Equivalent

In this section, we present two complementary perspectives showing that SB are essentially equivalent to EOT. Unlike classical optimal transport, which produces a single deterministic map, SB yields a *stochastic* flow of particles: mass is transported probabilistically, with marginals evolving under diffusion-like dynamics.

From the static viewpoint, SB coincides with EOT, where the goal is to find a coupling between the two endpoint distributions that balances transport cost with entropy. From the dynamic viewpoint, SB describes a controlled diffusion process that remains as close as possible to a simple reference (such as Brownian motion) while still matching the desired endpoints. Each perspective independently establishes the equivalence, providing two consistent ways to understand SB/EOT as canonical formulations of distribution-to-distribution transformation.

Static Schrödinger Bridge. Let

$$\widetilde{R}^{\varepsilon}(\mathbf{x}, \mathbf{y}) := \frac{1}{Z_\varepsilon} e^{-c(\mathbf{x},\mathbf{y})/\varepsilon} p_{\text{src}}(\mathbf{x})\, p_{\text{tgt}}(\mathbf{y}),$$

with normalizing constant

$$Z_\varepsilon := \iint e^{-c(\mathbf{x},\mathbf{y})/\varepsilon} p_{\text{src}}(\mathbf{x})\, p_{\text{tgt}}(\mathbf{y})\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{y}.$$

Then, for any admissible coupling $\gamma \in \Gamma(p_{\mathrm{src}}, p_{\mathrm{tgt}})$, a direct computation gives

$$\int c \, \mathrm{d}\gamma + \varepsilon \mathcal{D}_{\mathrm{KL}}(\gamma \| p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}) = \varepsilon \mathcal{D}_{\mathrm{KL}}(\gamma \| \widetilde{R}^{\varepsilon}) - \varepsilon \log Z_{\varepsilon}.$$

Therefore,

$$\min_{\gamma \in \Gamma(p_{\mathrm{src}}, p_{\mathrm{tgt}})} \left\{ \int c \, \mathrm{d}\gamma + \varepsilon \mathcal{D}_{\mathrm{KL}}(\gamma \| p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}) \right\} = \varepsilon \min_{\gamma \in \Gamma(p_{\mathrm{src}}, p_{\mathrm{tgt}})} \mathcal{D}_{\mathrm{KL}}(\gamma \| \widetilde{R}^{\varepsilon}) - \varepsilon \log Z_{\varepsilon}, \tag{7.3.1}$$

so the entropic OT problem is equivalent, up to an additive constant, to the static Schrödinger Bridge problem

$$\min_{\gamma \in \Gamma(p_{\mathrm{src}}, p_{\mathrm{tgt}})} \mathcal{D}_{\mathrm{KL}}(\gamma \| \widetilde{R}^{\varepsilon}).$$

Dynamic Equivalence (Brownian Reference). We can also understand this equivalence from a dynamical viewpoint. A classical result (Mikami and Thieullen, 2006) states that entropic OT with quadratic cost

$$c(\mathbf{x}, \mathbf{y}) = \frac{\|\mathbf{y} - \mathbf{x}\|^2}{2T}$$

is affinely equivalent to the SB problem where the reference path law R^{ε} is Brownian motion on $[0, T]$,

$$\mathrm{d}\mathbf{x}_t = \sqrt{\varepsilon} \, \mathrm{d}\mathbf{w}_t.$$

Here, "affinely equivalent" means the optimal values differ by a positive scaling and an additive constant (independent of the decision variable), so the minimizers coincide. In particular, let P^* be the optimal path distribution for SB. The key connection to EOT is obtained by looking only at the two endpoints of this optimal stochastic process. If $\mathbf{x}_{[0:T]} \sim P^*$, then the endpoint pair $(\mathbf{x}_0, \mathbf{x}_T)$ induces the coupling

$$\gamma^* := (\mathbf{x}_0, \mathbf{x}_T)_{\#} P^*,$$

and γ^* is an optimal transport plan for the corresponding EOT problem.

Conversely, starting from an optimal EOT coupling γ^*, we may first sample a pair of endpoints $(\mathbf{x}_0, \mathbf{x}_T) \sim \gamma^*$ and then, conditioned on these endpoints, sample a trajectory from the reference diffusion conditioned to connect $\mathbf{x}_0$ to $\mathbf{x}_T$. Mixing these conditional diffusion paths according to γ^* recovers the optimal SB path distribution P^*.

In words, the optimal stochastic process from the dynamic SB problem induces the optimal endpoint coupling of the static EOT problem. Conversely, the optimal EOT coupling specifies how the endpoints should be paired, while the reference diffusion specifies the optimal stochastic path between each paired set of endpoints.

The key idea to derive this fact is that the KL divergence over paths can be broken down according to the endpoints, which means the Schrödinger bridge problem reduces to a KL divergence just over the joint distribution of $(\mathbf{x}_0, \mathbf{x}_T)$. For Brownian motion the transition density between $\mathbf{x}$ and $\mathbf{y}$ has a Gaussian form, so its negative log is quadratic:

$$-\varepsilon \log p_T(\mathbf{y} \mid \mathbf{x}) = \frac{\|\mathbf{y} - \mathbf{x}\|^2}{2T} + \text{const.}$$

This shows that the endpoint KL is exactly the same as the entropic OT objective with quadratic cost, up to an irrelevant constant.

SB with General Reference Determines the Endpoint EOT Cost. The connection between SB and EOT is not restricted to Brownian motion. For a general well-posed reference process, its transition law tells us how "natural" it is under the reference dynamics to move from $\mathbf{x}$ at time 0 to $\mathbf{y}$ at time T. Endpoint pairs that are unlikely under the reference should therefore carry a larger transport cost.

More precisely, let $p_T(\mathbf{y}|\mathbf{x})$ denote the transition density of the reference process over $[0, T]$. Up to a scaling and additive terms that depend only on the fixed endpoint marginals, the corresponding endpoint cost takes the form

$$c(\mathbf{x}, \mathbf{y}) \propto -\log p_T(\mathbf{y}|\mathbf{x}).$$

Thus, at the level of endpoint couplings, the reference dynamics induce the cost used by the corresponding EOT problem: likely transitions receive smaller cost, while unlikely transitions receive larger cost.

The full reference process contains additional information beyond this endpoint cost. Once an endpoint pair $(\mathbf{x}, \mathbf{y})$ is selected, the reference dynamics also determine the conditional stochastic paths used to connect those endpoints. In this sense, EOT determines how the endpoints are optimally paired, while SB additionally describes the stochastic transport between each paired set of endpoints.

7.3.2 EOT_ε is Reduced to OT as $\varepsilon \to 0$

Let γ_ε^* denote the optimal plan for EOT_ε, and let γ^* denote an optimal plan of the unregularized OT problem in Equation (7.2.1). Under standard assumptions, as $\varepsilon \to 0$ the EOT optimal values converge to the OT value, and any limiting entropic optimal plan is OT-optimal; when the OT optimizer is unique, the optimal plans converge to γ^*.

This convergence result is both fundamental and practically important. One of the reasons is that the entropy-regularized OT problem EOT_ε admits efficient numerical solutions via algorithms such as Sinkhorn. Thus, the result provides theoretical justification for using EOT_ε with small ε as a computationally tractable

proxy for the classical OT problem in Equation (7.2.1), even when the cost function $c(\mathbf{x}, \mathbf{y})$ is more general than the quadratic case.

Theorem 7.3.1: (Informal) EOT$_\varepsilon$ Converges to OT.

Under standard assumptions ensuring the entropic OT limit, as $\varepsilon \to 0$, the optimal values converge:

$$\lim_{\varepsilon \to 0} \mathrm{EOT}_\varepsilon(p_{\mathrm{src}}, p_{\mathrm{tgt}}) = \mathrm{OT}(p_{\mathrm{src}}, p_{\mathrm{tgt}}).$$

Moreover, any limiting optimal plan is an optimal coupling of the unregularized OT problem. In particular, when the OT optimizer is unique, the entropic optimal plans converge to that optimizer.

Proof for Theorem.

For a rigorous proof, we refer to the literature (Mikami and Thieullen, 2008; Peyré, Cuturi, *et al.*, 2019). Below we provide a heuristic derivation of the value convergence.

Let us denote the corresponding optimal values by

$$V_\varepsilon := \mathrm{EOT}_\varepsilon(p_{\mathrm{src}}, p_{\mathrm{tgt}}), \quad V_0 := \mathrm{OT}(p_{\mathrm{src}}, p_{\mathrm{tgt}})$$

for notational simplicity.

The elementary upper-bound argument below applies when an OT-optimal coupling has finite relative entropy with respect to $p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}$. In general, the rigorous value convergence can be established by approximating optimal couplings with finite-entropy couplings, or more systematically through Γ-convergence arguments.

Upper Bound. By optimality of γ^*_ε, its value V_ε is bounded by the cost of using the plan γ^*:

$$V_\varepsilon \le \int c \, \mathrm{d}\gamma^* + \varepsilon \mathcal{D}_{\mathrm{KL}}(\gamma^* \| p_{\mathrm{src}} \otimes p_{\mathrm{tgt}}).$$

Assuming the KL term is a finite constant K, we get $V_\varepsilon \le V_0 + \varepsilon K$. Taking the limit superior yields $\limsup_{\varepsilon \to 0} V_\varepsilon \le V_0$.

Lower Bound. Since the KL-divergence is non-negative, $V_\varepsilon \ge \int c \, \mathrm{d}\gamma^*_\varepsilon$. By definition of V_0 as the minimal transport cost, any plan's cost is at least V_0, so $\int c \, \mathrm{d}\gamma^*_\varepsilon \ge V_0$. This implies $V_\varepsilon \ge V_0$ for all $\varepsilon > 0$, and thus $\liminf_{\varepsilon \to 0} V_\varepsilon \ge V_0$.

Combining the upper and lower bounds shows the convergence of the optimal value, $\lim_{\varepsilon \to 0} V_\varepsilon = V_0$. The corresponding convergence of optimal plans requires additional compactness arguments: any limiting plan is OT-optimal, and full convergence follows when the OT optimizer is unique.

7.3.3 SB_ε is Reduced to OT as $\varepsilon \to 0$

For direct comparison with Equation (7.2.3), we rescale time so that $T = 1$ throughout this subsection. For each $\varepsilon > 0$, let $\mathbf{v}_t^\varepsilon$ be a minimizer of the SB problem as in Equation (7.2.10), and let p_t^ε be the marginal distribution of the controlled SDE $\mathbf{x}_t$ induced by $\mathbf{v}_t^\varepsilon$. Then p_t^ε satisfies the associated Fokker–Planck equation. In contrast, denote by $(p_t^0, \mathbf{v}_t^0)$ a minimizer of the Benamou–Brenier formulation of optimal transport (see Equation (7.2.3)).

The following theorem[7] states that as $\varepsilon \to 0$, the SB problem converges to the OT problem. This result is practically important for reasons similar to those in Theorem 7.3.1. The objective SB_ε can be efficiently solved using Sinkhorn type algorithms, yielding a numerically tractable and differentiable proxy for optimal transport. This is especially valuable in high dimensional or large scale settings, where direct solvers (e.g., based on the Benamou–Brenier formulation) become computationally expensive.

Theorem 7.3.2: (Informal) SB_ε Converges to OT.

As $\varepsilon \to 0$, we have:

$$\lim_{\varepsilon \to 0} 2\varepsilon \, \mathrm{SB}_\varepsilon(p_{\mathrm{src}}, p_{\mathrm{tgt}}) = \mathcal{W}_2^2(p_{\mathrm{src}}, p_{\mathrm{tgt}}),$$

where OT is of the Benamou–Brenier formulation as in Equation (7.2.3). Moreover, under suitable compactness and regularity assumptions, the marginals p_t^ε and the corresponding velocity fields $\mathbf{v}_t^\varepsilon$ converge, in the appropriate weak sense, along subsequences to a limiting pair $(p_t^0, \mathbf{v}_t^0)$ that solves the Benamou–Brenier problem. In particular, when the Benamou–Brenier minimizer is unique, the optimal Schrödinger dynamics converge to this OT solution in the corresponding limiting sense.

Proof for Theorem.

A full rigorous proof of the convergence result is beyond our scope; we refer the reader to Léonard (2012) and Léonard (2014) for detailed derivations. Nevertheless, we can heuristically understand why this convergence may hold.

In the stochastic control formulation of the SB problem Equation (7.2.10), the controlled SDE is given by:

$$\mathrm{d}\mathbf{x}_t = \mathbf{v}_t^\varepsilon(\mathbf{x}_t)\, \mathrm{d}t + \sqrt{\varepsilon}\, \mathrm{d}\mathbf{w}_t.$$

As $\varepsilon \to 0$, the noise term vanishes, and the SDE formally approaches a deter-

[7] A rigorous formulation uses Γ-convergence of the appropriately rescaled Schrödinger functionals. Together with suitable compactness assumptions, this yields convergence of optimal values and corresponding limit points of minimizers. Since this requires more technical background, we omit the details here and state only the conceptual result.

ministic ODE:

$$\mathrm{d}\mathbf{x}_t = \mathbf{v}_t^0(\mathbf{x}_t)\ \mathrm{d}t.$$

Together with Equation (7.2.10), this suggests that the appropriately rescaled SB objective converges to the Benamou–Brenier action:

$$\lim_{\varepsilon \to 0} 2\varepsilon\, \mathrm{SB}_\varepsilon(p_{\mathrm{src}}, p_{\mathrm{tgt}}) = \mathcal{W}_2^2(p_{\mathrm{src}}, p_{\mathrm{tgt}}).$$

In parallel, the marginal density p_t^ε satisfies the Fokker-Planck equation:

$$\partial_t p_t^\varepsilon + \nabla \cdot (p_t^\varepsilon \mathbf{v}_t^\varepsilon) = \frac{\varepsilon}{2} \Delta p_t^\varepsilon.$$

Again, as $\varepsilon \to 0$, the diffusion term vanishes, and the equation formally reduces to the continuity equation:

$$\partial_t p_t^0 + \nabla \cdot \left(p_t^0 \mathbf{v}_t^0\right) = 0.$$

Until now, we have presented the fundamental equivalences (under their respective assumptions) between EOT and SB, as well as their important connection to OT through a limiting process, illustrated in Figure 7.4. Next, we will explore how diffusion models connect to these concepts.

7.4 Is Diffusion Model's SDE Optimal Solution to SB Problem?

7.4.1 Diffusion models as a Special Case of Schrödinger Bridges

SB framework extends (score-based) diffusion models by enabling nonlinear interpolation between arbitrary source and target distributions. It achieves this by adding control drift terms derived from scalar potentials $\Psi(\mathbf{x}, t)$ and $\widehat{\Psi}(\mathbf{x}, t)$, which guide a reference diffusion process to match prescribed endpoint marginals (see Equation (7.2.12)) and follow the decomposition:

$$\nabla \log \Psi(x, t) + \nabla \log \hat{\Psi}(x, t) = \nabla \log p_t^{\mathrm{SB}}(\mathbf{x}).$$

This generalization allows the model to move beyond standard Gaussian priors and generate samples from broader distributions.

Connection to Diffusion Models. Diffusion models arise as a special case of the SB framework. Suppose the potential is constant, $\Psi(\mathbf{x}, t) \equiv 1$. Under this assumption, the second PDE in Equation (7.2.12) reduces to the standard Fokker–Planck equation, whose solution is the marginal density of the reference process:

$$\widehat{\Psi}(\mathbf{x}, t) = p_t^{\mathrm{SB}}(\mathbf{x}). \tag{7.4.1}$$

The corresponding SB forward SDE thus becomes the uncontrolled reference process:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t,$$

and the SB backward SDE simplifies to:

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla \log p_t^{\mathrm{SB}}(\mathbf{x}_t)\right]\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}_t,$$

which matches Anderson's reverse-time SDE used in diffusion models. This correspondence shows that diffusion models can be interpreted as the zero-control limit of SB, where no additional drift is introduced by the potentials.

Boundary Conditions and Generality. The above reduction is purely formal unless the boundary constraints are compatible. For arbitrary source/target $(p_{\mathrm{src}}, p_{\mathrm{tgt}})$, the PDE boundary conditions in Equation (7.2.12) are generally not satisfied by the choice $\Psi \equiv 1$. Full SB resolves this by learning nontrivial potentials that induce a nonlinear control drift, bending the reference dynamics to match any prescribed endpoints. By contrast, diffusion models fix one endpoint to a simple prior (typically Gaussian) and learn only the reverse-time score to reach the data. With this perspective, SB is the more flexible umbrella: with nontrivial potentials it bridges arbitrary endpoints; with $\Psi \equiv 1$ it collapses to the diffusion-model case above. We additionally remark that in the standard linear diffusion model, $p_T \approx p_{\mathrm{prior}}$ holds only as $T \to \infty$, so the match to the prior is merely approximate.

7.4.2 Diffusion Models as Schrödinger Half-Bridges

In this section, we explain why diffusion models are not full Schrödinger bridges, but can instead be understood through the relaxed notion of *Schrödinger half-bridges*. A half-bridge enforces only one endpoint constraint (either p_{prior} or p_{data}) rather than both, making it a one-sided variant of the full bridge. Before formalizing this connection, we introduce the definition of Schrödinger half-bridges, building on the general formulation in Equation (7.2.7) with arbitrary p_{src} and p_{tgt}. We will then return to diffusion models and show how the half-bridge viewpoint naturally applies when the endpoints are given by p_{prior} and p_{data}.

Schrödinger Half-Bridges The SB problem asks for a stochastic process whose law is closest (in KL divergence) to a simple reference process, while *matching two endpoint distributions* p_{src} and p_{tgt}. Solving the full bridge requires enforcing both boundary conditions, which is often computationally difficult. A useful relaxation is the *half-bridge* problem: instead of matching both endpoints, we match only one of them.

Formally, let R be the reference path distribution. The *forward half-bridge* seeks a path distribution P minimizing

$$\min_{P:P_0=p_{\text{src}}} \mathcal{D}_{\text{KL}}(P \,\|\, R),$$

subject to the single constraint $P_0 = p_{\text{src}}$. Similarly, the *backward half-bridge* constrains only the terminal distribution,

$$\min_{P:P_T=p_{\text{tgt}}} \mathcal{D}_{\text{KL}}(P \,\|\, R).$$

In words, the forward half-bridge asks: among all processes starting from the desired initial distribution, which one looks most like the reference? The backward half-bridge asks the same question for processes ending at the desired terminal distribution. By combining these two relaxations iteratively, one can approximate the full SB.

Diffusion Models Miss Exact Endpoint Matching. A key difference between diffusion models and the SB framework lies in the treatment of the terminal distribution p_T. In standard diffusion models, the forward SDE is typically linear in $\mathbf{x}_t$ (see Equation (4.4.2)) and designed so that p_T approximates the prior only as $T \to \infty$:

$$p_T \approx p_{\text{prior}}.$$

At finite time, however, p_T is the distribution of $\alpha_T \mathbf{x}_0 + \sigma_T \boldsymbol{\epsilon}$ and therefore generally still depends on p_{data}; it need not coincide exactly with the prescribed Gaussian prior (see Section C.1.5).

In contrast, the SB framework enforces exact marginal matching at a finite time T by introducing an additional control drift of the form $g^2(t)\nabla_{\mathbf{x}} \log \Psi(\mathbf{x}, t)$. This ensures that the terminal distribution precisely satisfies $p_T = p_{\text{prior}}$, regardless of the initial data distribution $p_0 = p_{\text{data}}$. In summary:

- **Diffusion Models:** $p_T \approx p_{\text{prior}}$, asymptotically as $T \to \infty$,
- **Schrödinger Bridge:** $p_T = p_{\text{prior}}$ exactly at finite T, enabled by solving for the control potentials Ψ and $\widehat{\Psi}$.

Diffusion Schrödinger Bridge. The fixed forward noising process can be viewed as a forward Schrödinger half-bridge anchored at p_{data} relative to the chosen reference dynamics. Its terminal marginal is the distribution produced by that reference process and is generally only approximately p_{prior}; hence it is not the full bridge between the prescribed pair $(p_{\text{data}}, p_{\text{prior}})$.

To address this, the Diffusion Schrödinger Bridge (DSB) (De Bortoli *et al.*, 2021) alternates between matching both endpoint marginals by following the idea of the Iterative Proportional Fitting (IPF) algorithm, an alternating projection method. This extends diffusion models to solve the full SB problem as follows[8]:

- **Step 0: Reference Process.** Initialize with $P^{(0)} := R_{\text{fwd}}$, the reference forward SDE:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t, \quad \mathbf{x}_0 \sim p_{\text{data}}.$$

  This ensures $P_0^{(0)} = p_{\text{data}}$, but typically $P_T^{(0)} \neq p_{\text{prior}}$.

- **Step 1: Backward Pass.** Compute the process $P^{(1)}$ that matches p_{prior} at time T while staying close to $P^{(0)}$:

$$P^{(1)} = \underset{P: P_T = p_{\text{prior}}}{\arg\min}\ \mathcal{D}_{\text{KL}}(P \| P^{(0)}).$$

  This is achieved via approximating the oracle score function with a neural network $\mathbf{s}_{\phi^\times}$, which results in the reverse-time SDE:

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}_t,$$

  simulated backward from $\mathbf{x}_T \sim p_{\text{prior}}$.

- **Iteration.** The process $P^{(1)}$ satisfies $P_T^{(1)} = p_{\text{prior}}$, but its initial marginal $P_0^{(1)}$ typically deviates from p_{data}. IPF addresses this by learning a forward SDE to adjust $P_0^{(1)}$ back to p_{data}, followed by another backward pass to enforce

[8] Although this description uses p_{data} and p_{prior}, the DSB framework applies to any pair of endpoint distributions.

p_{prior}. This alternation continues, refining the process until convergence to the optimal bridge P^*, which satisfies both $P_0^* = p_{\text{data}}$ and $P_T^* = p_{\text{prior}}$. De Bortoli *et al.* (2021) prove convergence under mild conditions.

The DSB formulation above is historically important and conceptually clear, but its practical implementation can be computationally demanding. The forward–backward projections are carried out iteratively, so that each IPF iteration requires solving a separate score-matching problem, while approximation errors may also accumulate across rounds. More broadly, a growing body of work seeks to alleviate these difficulties by developing Schrödinger bridge formulations with improved stability and scalability, and in some cases with reduced dependence on repeatedly fitting separate diffusion models throughout the iterative procedure. We do not discuss these developments in detail here, but mention them to orient interested readers to a broader literature on practical SB methods (Shi *et al.*, 2023; Liu *et al.*, 2024).

7.5 Is Diffusion Model's ODE an Optimal Map to OT Problem?

In this section, we focus on quadratic-cost optimal transport problem.

7.5.1 PF-ODE Flow Is Generally Not Optimal Transport

A natural question is whether the deterministic flow map induced by a diffusion process coincides with the optimal transport map under quadratic cost. The answer is no in general, and explicit counterexamples are already available in Gaussian settings. In particular, Tanana (2017) showed that, for a diffusion flow with a more general Gaussian reference, the resulting deterministic transport map can differ from the Brenier optimal transport map even when the endpoint distributions are Gaussian. Since both maps are explicit in this setting, their difference can be verified directly.

Below we present the stronger construction of Lavenant and Santambrogio (2022) in the standard-Gaussian Ornstein–Uhlenbeck setting that is directly relevant to diffusion models. Their argument establishes non-optimality for a broader class of initial distributions and provides structural insight into why the corresponding PF-ODE flow map need not be an optimal transport map.

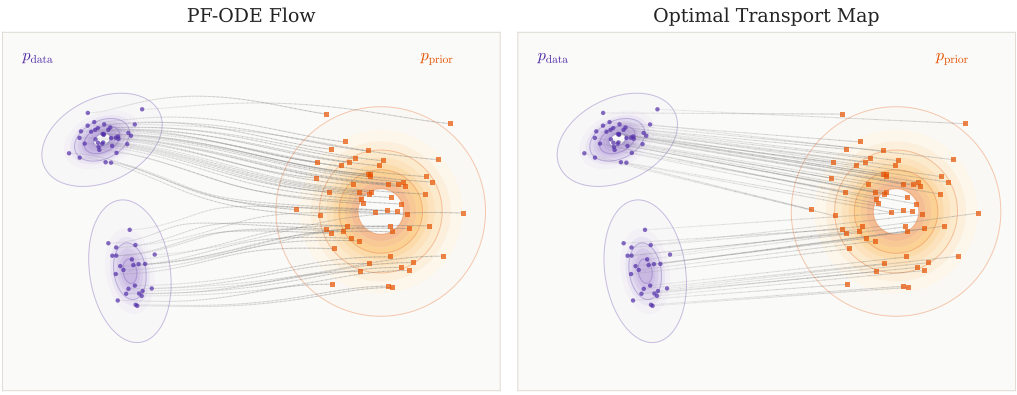


Figure 7.6: Comparing the PF-ODE flow map with the true OT map from p_{data} to p_{prior} in 2D. Right: the quadratic-cost OT map, computed exactly via the Hungarian algorithm. Left: the PF-ODE flow map obtained by integrating the PF-ODE. The two maps visibly differ, illustrating that the PF-ODE does not generally recover the optimal transport map.

Source: Created by the authors with AI-assisted coding.

Setup. We consider a VP SDE, specifically the Ornstein–Uhlenbeck process, which evolves a smooth initial density p_0 toward the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$\mathrm{d}\mathbf{x}(t) = -\mathbf{x}(t)\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}\mathbf{w}(t), \quad \mathbf{x}(0) \sim p_0.$$

The associated PF-ODE is given by

$$\frac{\mathrm{d}\mathbf{S}_t(\mathbf{x})}{\mathrm{d}t} = -\mathbf{S}_t(\mathbf{x}) - \nabla \log p_t(\mathbf{S}_t(\mathbf{x})), \quad \mathbf{S}_0(\mathbf{x}) = \mathbf{x}.$$

Here, $\mathbf{S}_t$ denotes the flow map pushing forward p_0 to the marginal p_t:

$$(\mathbf{S}_t)\,\#p_0 = p_t, \quad \text{that is,} \quad p_t(\mathbf{y}) = \int_{\mathbb{R}^D} \delta(\mathbf{y} - \mathbf{S}_t(\mathbf{x}))p_0(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

These densities p_t evolve via the Fokker-Planck equation:

$$\frac{\partial p_t}{\partial t} = \nabla \cdot (\mathbf{x}p_t) + \Delta p_t.$$

This is equivalent to a continuity equation with velocity field:

$$\mathbf{v}_t(\mathbf{x}) = -\mathbf{x} - \nabla \log p_t(\mathbf{x}),$$

whose flow is given by $\mathbf{S}_t(\mathbf{x})$. In other words, the PF-ODE can be written as:

$$\frac{\mathrm{d}\mathbf{S}_t(\mathbf{x})}{\mathrm{d}t} = \mathbf{v}_t\left(\mathbf{S}_t(\mathbf{x})\right).$$

As $t \to \infty$, the map transports the initial distribution to the prior:

$$\mathbf{S}_\infty \# p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I}) =: p_{\text{prior}}.$$

Objective of Lavenant and Santambrogio (2022)'s Argument. Lavenant and Santambrogio (2022) do not directly assess whether the terminal map $\mathbf{S}_\infty$ from p_0 to the Gaussian is optimal. Instead, they construct a specific initial distribution p_0 and examine the entire PF-ODE trajectory. Their key observation is that optimality may fail at some point along the flow.

They consider the intermediate marginal $p_t = \mathbf{S}_t \# p_0$ and define the residual transport map from p_{t_0} to the Gaussian as

$$\mathbf{T}_{t\to\infty} := \mathbf{S}_\infty \circ \mathbf{S}_t^{-1}, \quad \text{for all } t \geq 0.$$

The core of their argument shows that, for a carefully chosen p_0, there exists a time $t_0 \geq 0$ such that $\mathbf{T}_{t_0\to\infty}$ is not the quadratic-cost optimal transport map from the new starting distribution p_{t_0} to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

This result demonstrates that PF-ODE flows do not, in general, yield optimal transport maps, and that the property of optimality can break down for certain initial distributions.

Some Tools. The argument of Lavenant and Santambrogio (2022) crucially relies on the following result, known as *Brenier's theorem*:

Theorem 7.1 (Informal Brenier's Theorem)**.** Let ν_1, ν_2 be two probability distributions on $\mathbb{R}^D$ with smooth densities. A smooth map $\mathbf{T} : \mathbb{R}^D \to \mathbb{R}^D$ is the optimal transport from ν_1 to ν_2 (under quadratic cost) if and only if $\mathbf{T} = \nabla u$ for some convex function u. In this case, $\mathrm{D}\mathbf{T}$ is symmetric and positive semi-definite, and u satisfies the Monge–Ampère equation:

$$\det \mathrm{D}^2 u(\mathbf{x}) = \frac{\nu_1(\mathbf{x})}{\nu_2(\nabla u(\mathbf{x}))}.$$

The proof also implicitly uses the following fact, which we will not repeat each time: *a map is the optimal transport between two distributions if and only if its inverse is the optimal transport in the reverse direction.*

Proof Sketch: PF-ODE Is Not an OT Map in General. Lavenant and Santambrogio (2022) employ a proof by contradiction: they assume that for every $t \geq 0$, the map

$$\mathbf{T}_t = \mathbf{S}_t \circ \mathbf{S}_\infty^{-1}$$

is the quadratic-cost optimal transport map from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ to p_t.

Step 1: Brenier's Theorem. By Brenier's Theorem, the Jacobian of any optimal transport map from Gaussian must be symmetric and positive semi-definite. Thus,

$$\mathrm{D}\mathbf{T}_t(\mathbf{x}) = \mathrm{D}\mathbf{S}_t(\mathbf{S}_\infty^{-1}(\mathbf{x}))\mathrm{D}(\mathbf{S}_\infty^{-1})(\mathbf{x})$$

must be symmetric for all t and $\mathbf{x}$. Here, $\mathrm{D}\mathbf{T}_t(\mathbf{x})$ denotes the total differentiation with respect to $\mathbf{x}$.

Step 2: Time-Differentiating the Symmetry Condition. Differentiating in time:

$$\frac{\partial}{\partial t}\mathrm{D}\mathbf{T}_t(\mathbf{x}) = \left(\frac{\partial}{\partial t}\mathrm{D}\mathbf{S}_t\right)(\mathbf{S}_\infty^{-1}(\mathbf{x}))\mathrm{D}(\mathbf{S}_\infty^{-1})(\mathbf{x}).$$

Given that the symmetry holds for all t, it follows that this derivative remains symmetric.

Using the flow ODE (differentiating in $\mathbf{x}$), we obtain:

$$\frac{\partial(\mathrm{D}\mathbf{S}_t)}{\partial t} = \mathrm{D}\mathbf{v}_t(\mathbf{S}_t) \cdot \mathrm{D}\mathbf{S}_t = \left(-\mathbf{I} - \mathrm{D}^2 \log p_t(\mathbf{S}_t)\right) \cdot \mathrm{D}\mathbf{S}_t.$$

Combining the above, we see that

$$\left(-\mathbf{I} - \mathrm{D}^2 \log p_t(\mathbf{S}_t)\right) \cdot \mathrm{D}\mathbf{S}_t \cdot \mathrm{D}(\mathbf{S}_\infty^{-1})$$

is symmetric for all $t \geq 0$.

At $t = 0$, we have $\mathbf{S}_0 = \mathbf{I}$ and $\mathrm{D}\mathbf{S}_0 = \mathbf{I}$, yielding:

$$\left(-\mathbf{I} - \mathrm{D}^2 \log p_0(\mathbf{S}_\infty^{-1}(\mathbf{x}))\right) \cdot \mathrm{D}(\mathbf{S}_\infty^{-1})(\mathbf{x}) \quad \text{is symmetric.}$$

Step 3: The Commutation Condition. Since $\mathbf{T}_0 = \mathbf{S}_\infty^{-1}$ is assumed to be optimal, its Jacobian $D\mathbf{T}_0 = \mathrm{D}(\mathbf{S}_\infty^{-1})$ is symmetric. Moreover, the Hessian $\mathrm{D}^2 \log p_0$ is symmetric. Recall that two symmetric matrices multiply to a symmetric matrix if and only if they commute. Hence, for all $\mathbf{x} \in \mathbb{R}^D$,

$$\mathrm{D}^2 \log p_0(\mathbf{S}_\infty^{-1}(\mathbf{x})) \quad \text{must commute with} \quad \mathrm{D}(\mathbf{S}_\infty^{-1})(\mathbf{x}).$$

Setting $\mathbf{y} = \mathbf{S}_\infty^{-1}(\mathbf{x})$ gives the equivalent condition: for all $\mathbf{y} \in \mathbb{R}^D$,

$$\mathrm{D}^2 \log p_0(\mathbf{y}) \quad \text{must commute with} \quad \mathrm{D}\mathbf{S}_\infty(\mathbf{y}).$$

Now, we transform this condition into a more computable form. Since $\mathbf{S}_\infty$ is optimal between p_0 and $\mathcal{N}(\mathbf{0}, \mathbf{I})$, Brenier's theorem guarantees that $\mathbf{S}_\infty = \nabla u$ for some convex function u. From the Monge–Ampère equation, it follows that:

$$\log p_0(\mathbf{y}) = \log \det(\mathrm{D}^2 u(\mathbf{y})) - \frac{1}{2}\|\nabla u(\mathbf{y})\|^2 + \text{Constant}.$$

The condition becomes (with $\mathrm{D}\mathbf{S}_\infty = \mathrm{D}^2 u$):

$$\mathrm{D}^2 \left(\log \det \mathrm{D}^2 u - \frac{1}{2}\|\nabla u\|^2 \right) \quad \text{must commute with} \quad \mathrm{D}^2 u. \tag{7.5.1}$$

This yields a necessary condition for $\mathbf{T}_t$ to be optimal.

Step 4: Constructing the Counterexample. Let us show how to leverage this necessary condition to derive a contradiction.

Assume we can construct a convex function u such that

$$\mathrm{D}^2 \left(\log \det \mathrm{D}^2 u(\mathbf{x}) - \frac{1}{2}|\nabla u(\mathbf{x})|^2 \right)$$

does not commute with $\mathrm{D}^2 u(\mathbf{x})$ for some $\mathbf{x} \in \mathbb{R}^D$. Defining $p_0 = (\nabla u)^{-1} \# \mathcal{N}(\mathbf{0}, \mathbf{I})$, Brenier's theorem implies that ∇u is the optimal transport from p_0 to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. However, the condition in Equation (7.5.1) fails, leading to a contradiction. Thus, our goal is to construct such a function. Consider

$$u(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|^2 + \varepsilon \phi(\mathbf{x}), \quad \text{for a small } \varepsilon.$$

Then $\mathrm{D}^2 u(\mathbf{0}) = \mathbf{I} + \varepsilon \mathrm{D}^2 \phi(\mathbf{0})$, and the commutation condition at $\mathbf{x} = \mathbf{0}$ requires $\mathrm{D}^2 \phi(\mathbf{0})$ to commute with $\mathrm{D}^2(\Delta\phi)(\mathbf{0})$.

For example, in $\mathbb{R}^2$, choosing

$$\phi(x_1, x_2) = x_1 x_2 + x_1^4$$

provides a counterexample where the Hessian $\mathrm{D}^2 \log p_0$ and the Jacobian $\mathrm{D}^2 u$ do not commute.

This contradiction shows that $\mathbf{T}_t$ cannot be optimal for all $t \geq 0$. Therefore, there exists some $t_0 \geq 0$ such that the map $\mathbf{T}_{t_0 \to \infty}$ is not optimal.

7.5.2 Can Canonical Linear Flow and Reflow Leads to an OT Map?

We have seen that the PF-ODE (especially in VP type forward kernel) is generally not an OT map. One natural question now is:

Question 7.5.1

Does the linear interpolation flow $(1-t)\mathbf{x}_0 + t\mathbf{x}_1$ *with* $\mathbf{x}_0 \sim p_{\text{src}}$ *and* $\mathbf{x}_1 \sim p_{\text{tgt}}$, *when applied to the independent coupling* $\pi(\mathbf{x}_0, \mathbf{x}_1) = p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$, *recover the OT map?*

The answer to the question is no.

Nevertheless, combining a linear path with a given coupling offers a practical upper bound on the true OT cost. Among all possible paths, linear interpolation provides the tightest such upper bound, as we will see in the following discussion.

Canonical Linear Flow and Optimal Transport. Focusing on optimal transport with quadratic cost, we consider the equivalent form of Equation (7.2.1), the Benamou–Brenier formulation in Equation (7.2.3):

$$\mathcal{K}\left(p_{\text{src}}, p_{\text{tgt}}\right) := \min_{\substack{(p_t, \mathbf{v}_t) \text{ s.t. } \partial_t p_t + \nabla\cdot(p_t\mathbf{v}_t)=0, \\ p_0 = p_{\text{src}},\ p_1 = p_{\text{tgt}}}} \int_0^1 \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2 p_t(\mathbf{x})\, \mathrm{d}\mathbf{x}\, \mathrm{d}t.$$

Unlike the static Monge formulation (Equation (7.2.2)), where computing the optimal map requires solving the Monge–Ampère equation (Equation (7.2.4)), the Benamou–Brenier problem admits a convex reformulation after introducing the momentum field $\mathbf{m}_t := p_t\mathbf{v}_t$. Nevertheless, discretizing this convex problem on a grid in $\mathbb{R}^D$ remains intractable in high dimensions.

While solving the Benamou-Brenier formulation is generally intractable, Liu (2022) and Lipman *et al.* (2024) reveal that its kinetic energy admits a practical upper bound. This is achieved by restricting the search to a simpler family of *conditional flows*, where each path is defined by its fixed endpoints $(\mathbf{x}_0, \mathbf{x}_1)$ drawn from a coupling $\pi_{0,1}$ of the source and target distributions. Within this *conditional flow* family, the canonical linear interpolation emerges as the optimal choice, as formalized below.

Proposition 7.5.1: An Upper Bound on OT Kinetic Energy via Conditional Flows

Let $\pi_{0,1}$ be any coupling between p_{src} and p_{tgt}.

(1) The kinetic energy is bounded above by the expected path energy of any conditional flow $\boldsymbol{\Psi}_t(\mathbf{x}_0, \mathbf{x}_1)$ that connects the endpoints:

$$\mathcal{K}\left(p_{\text{src}}, p_{\text{tgt}}\right) \leq \mathbb{E}_{(\mathbf{x}_0,\mathbf{x}_1)\sim\pi_{0,1}}\left[\int_0^1 \left\|\boldsymbol{\Psi}_t'(\mathbf{x}_0, \mathbf{x}_1)\right\|^2 \mathrm{d}t\right].$$

(2) The unique conditional flow $\boldsymbol{\Psi}_t^*$ that minimizes the upper bound on the right-hand side is the linear interpolation path:

$$\boldsymbol{\Psi}_t^*(\mathbf{x}_0, \mathbf{x}_1) = (1-t)\mathbf{x}_0 + t\mathbf{x}_1.$$

Substituting this optimal path yields the tightest version of the bound:

$$\mathcal{K}\left(p_{\text{src}}, p_{\text{tgt}}\right) \leq \mathbb{E}_{(\mathbf{x}_0,\mathbf{x}_1)\sim\pi_{0,1}}\|\mathbf{x}_1 - \mathbf{x}_0\|^2.$$

Proof for Proposition.

> The proof relies on a straightforward application of Jensen's inequality and the tower property of conditional expectations, before solving the simplified variational problem with the Euler-Lagrange equation; we refer to (Lipman *et al.*, 2024)'s Section 4.7 for the complete argument. ∎

In other words, the linear interpolation $\boldsymbol{\Psi}_t^*$ (i.e., the forward kernel used by Flow Matching and Rectified Flow) minimizes an upper bound on the true kinetic energy for any chosen coupling $\pi_{0,1}$.

We emphasize that optimality within this class of conditional flows does not guarantee global optimality on the marginal distributions.

Reflow and Optimal Transport. The most naive transport plan between two distributions is to connect their samples with straight lines using a simple independent coupling. However, this approach is demonstrably not optimal, as the failure lies not in the straight-line paths themselves, but in the inefficient initial pairing of points.

The Reflow procedure may offer a constructive response. It is an iterative algorithm designed specifically to correct this pairing, and crucially, each step is guaranteed to be cost-non-increasing (Liu, Gong, *et al.*, 2022). This property suggests Reflow systematically pushes the transport plan towards a more optimal configuration, which naturally motivates the central question of its convergence.

Question 7.5.2

What happens if we apply the `Rectify` *operator iteratively? Can the resulting sequence of transport plans converge to the optimal one, or does the fixed point of the Reflow process yield the OT map?*

The short answer is no in general. Below, we explain what may go wrong. To recall, the Reflow procedure iteratively refines the coupling between p_{src} and p_{tgt} via the update:

$$\pi^{(k+1)} = \texttt{Rectify}(\pi^{(k)}),$$

initialized with the product coupling $\pi^{(0)} := p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$. More precisely, `Rectify` output the updated coupling $\pi^{(k+1)}$ via the following: At each iteration $k = 0, 1, 2, \ldots$, a velocity field $\mathbf{v}_t^{(k)}$ is learned via:

$$\mathbf{v}_t^{(k)} \in \arg\min_{\mathbf{u}_t} \mathcal{L}(\mathbf{u}_t|\pi^{(k)}),$$

where $\mathcal{L}(\mathbf{u}_t|\pi^{(k)})$ is the loss (e.g., RF or FM loss) defined in Equation (5.4.1). Here, for notational simplicity, we adopt a non-parametric formulation for the velocity field, rather than a parameterized form $\boldsymbol{\phi}$ employed in other contexts. The updated coupling is then given by:

$$\pi^{(k+1)}(\mathbf{x}_0, \mathbf{x}_1) := p_{\text{src}}(\mathbf{x}_0)\, \delta\big(\mathbf{x}_1 - \boldsymbol{\Psi}_1^{(k)}(\mathbf{x}_0)\big),$$

where $\boldsymbol{\Psi}_1^{(k)}$ denotes the solution map at time $t = 1$ obtained by integrating $\mathbf{v}_t^{(k)}$ from initial condition $\mathbf{x}_0$.

It has been observed in (Liu, Gong, *et al.*, 2022) that for a coupling π between p_{src} and p_{tgt}, the existence of a velocity field $\mathbf{v}_t$ that minimizes the Reflow loss, that is, satisfies $\mathcal{L}(\mathbf{v}_t|\pi) = 0$, does not necessarily imply that the transport is optimal.

Motivated by the Benamou–Brenier framework, where the optimal transport velocity is known to be the gradient of a potential function, Liu (2022) proposed an additional constraint: the velocity field $\mathbf{v}_t$ should be a potential field. Accordingly, the objective in Equation (5.4.1) is modified to restrict $\mathbf{v}_t$ to the space of gradient vector fields, also known as potential flows:

$$\mathbf{w}_t^{(k)} \in \arg\min_{\substack{\mathbf{u}_t:\, \mathbf{u}_t = \nabla\varphi \\ \text{for some } \varphi:\, \mathbb{R}^D \to \mathbb{R}}} \mathcal{L}(\mathbf{u}_t|\pi^{(k)}), \tag{7.5.2}$$

with the rest of the procedure remaining the same as in `Rectify`. We denote this associated operator as $\texttt{Rectify}_{\perp}$, emphasizing the projection onto irrotational vector fields.

Let π be a coupling between p_{src} and p_{tgt}. Liu, Gong, *et al.* (2022) conjecture the following equivalence characterizing optimality:

(*i*) π *is an optimal transport coupling.*

(*ii*) π *is a fixed point of the potential rectification operator:*

$$\pi = \texttt{Rectify}_{\perp}(\pi).$$

(*iii*) *There exists a gradient velocity field* $\mathbf{v}_t = \nabla\varphi_t$ *such that the rectify loss vanishes:*

$$\mathcal{L}(\mathbf{v}_t|\pi) = 0.$$

However, Hertrich *et al.* (2025) exhibit two types of counterexamples:

1. When the intermediate distributions p_t have disconnected support, one can find fixed points of $\texttt{Rectify}_{\perp}$ with zero Reflow loss and gradient velocity fields that nonetheless fail to produce the optimal coupling.

2. Even when both endpoint distributions are Gaussian, there exist couplings whose loss is arbitrarily small but whose deviation from the optimal coupling is arbitrarily large.

Therefore, while rectified flows may yield strong generative models, their reliability as optimal transport solvers remains limited. This highlights an important gap between generative modeling and principled optimal transport theory, inviting further research at their intersection.

Finally, we note that transport cost does not always correlate with downstream performance; as such, computing the exact optimal transport map may not necessarily lead to better practical outcomes. Nonetheless, variants of optimal transport remain fundamental to many problems in science and engineering. Diffusion models offer a powerful framework for exploring these challenges.

7.6 Closing Remarks

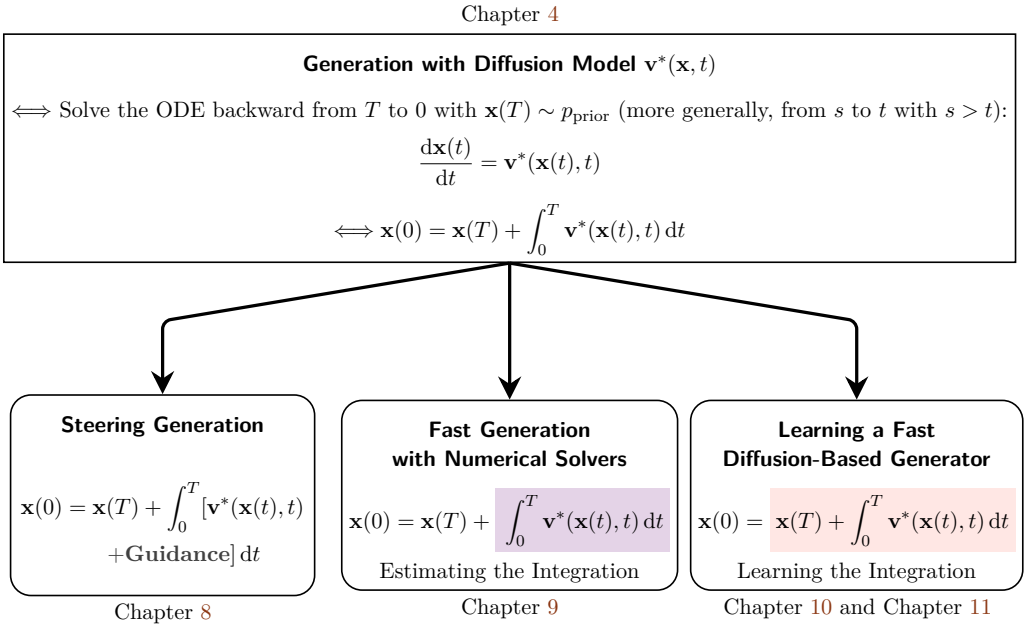
In this chapter, we revisited the classical problem of transporting one distribution to another through the lens of optimal transport and its entropic variants. We reviewed optimal transport in both its static Monge–Kantorovich form and its dynamic Benamou–Brenier formulation, where transport is realized by a deterministic flow minimizing kinetic cost. We then discussed entropy-regularized optimal transport and the Schrödinger bridge (SB), which replace deterministic transport by the most likely controlled diffusion relative to a reference process. From this perspective, diffusion models lie naturally near the SB viewpoint on the stochastic side, while their PF-ODEs define deterministic transport maps on the ODE side. As we emphasized, however, the PF-ODE transport is generally *not* the quadratic-cost optimal transport map: it is one deterministic coupling among many, rather than the minimizer of the Benamou–Brenier action.

This also highlights a more basic question: before asking whether a transport is optimal, *when can two distributions be connected by a deterministic map at all?* In many settings, existence is not the main difficulty. Under mild regularity assumptions, one can often find a measurable map $\mathbf{T}$ such that $\mathbf{T}_{\#}p_{\mathrm{src}} = p_{\mathrm{tgt}}$. The harder issue is that, beyond one dimension, relatively few classical constructions are both explicit and robust in practice. In one dimension, the monotone rearrangement gives a simple answer by matching quantiles. In higher dimensions, several important constructions are known: Brenier's theorem (Theorem 7.1) yields the quadratic-cost optimal map under suitable regularity; the Knothe–Rosenblatt rearrangement (Rosenblatt, 1952; Knothe, 1957) builds a triangular map from iterated conditional distributions; and the Dacorogna–Moser approach (Dacorogna and Moser, 1990) constructs smooth diffeomorphic transports through a velocity field solving a divergence equation. Each gives a principled route to deterministic transport, but each also relies on structural assumptions or nontrivial analytical machinery.

From this viewpoint, diffusion-based flow maps provide an appealing and flexible alternative. Rather than prescribing a closed-form transport map, diffusion models and flow-matching methods learn a time-dependent vector field whose induced flow pushes a simple source distribution toward the target. This learned map need not coincide with the Brenier map or any other classical canonical construction, and in general it does not solve a prescribed optimal transport problem. Nevertheless, it offers a practical and expressive route to map-based sampling in high dimensions, where explicit transport-map constructions are often unavailable or difficult to compute. In this sense, diffusion flows are best understood not as replacements for classical transport theory, but as modern learned transport mechanisms that sit alongside optimal transport maps, triangular rearrangements, and PDE-based constructions within the broader landscape of deterministic transport.

Part C

Sampling of Diffusion Models


Chapter 4
Generation with Diffusion Model $\mathbf{v}^*(\mathbf{x}, t)$
$\iff$ Solve the ODE backward from T to 0 with $\mathbf{x}(T) \sim p_{\text{prior}}$ (more generally, from s to t with $s > t$):
$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}^*(\mathbf{x}(t), t)$
$\iff \mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t)\,\mathrm{d}t$
Steering Generation
$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T [\mathbf{v}^*(\mathbf{x}(t), t) + \mathbf{Guidance}]\,\mathrm{d}t$
Chapter 8
Fast Generation with Numerical Solvers
$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t)\,\mathrm{d}t$
Estimating the Integration
Chapter 9
Learning a Fast Diffusion-Based Generator
$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t)\,\mathrm{d}t$
Learning the Integration
Chapter 10 and Chapter 11

8

Guidance and Controllable Generation

Diffusion models are powerful generative frameworks. In the unconditional setting, the goal is to learn $p_{\text{data}}(\mathbf{x})$ and generate samples without external input.

Many applications, however, require *conditional generation*, where outputs satisfy user-specified criteria. This can be achieved by steering an unconditional model or directly learning the conditional distribution $p_0(\mathbf{x}|\mathbf{c})$, with condition $\mathbf{c}$ (e.g., label, text description, or sketch) guiding the process.

This chapter builds on a principled view of the conditional score, which decomposes into an *unconditional direction* and a *guidance direction* that nudges samples toward the condition while preserving realism. We explain why guidance is essential, show how the conditional score serves as a unifying interface for control, and survey ways to approximate or replace the guidance term. We then distinguish *control* (meeting the condition) from *alignment* (meeting human preference under the condition), and describe how preferences can be incorporated into the same framework. Finally, we discuss direct optimization of preference without additional reward models (i.e., a learned scorer that assigns higher values to outputs better aligned with human preference).

After this chapter, you will be able to...

- understand how conditioning modifies the velocity field used during diffusion sampling;
- derive the key ideas behind classifier guidance and classifier-free guidance, and understand how guidance strength controls conditioning;
- understand how training-free guidance uses task-specific losses or measurement models to steer a pre-trained diffusion model, including for inverse problems;

- distinguish exact conditional sampling from surrogate-guided dynamics and their fixed-time tilted-density interpretations;
- understand how preference-based alignment, from RLHF to DPO and Diffusion-DPO, further reshapes a conditional generative model toward preferred outputs.

Reader route. The chapter assumes the Score SDE/PF-ODE picture from Chapters 4 and 6. Readers interested primarily in controllable generation may treat the later preference-alignment material as a second pass.

8.1 Prologue

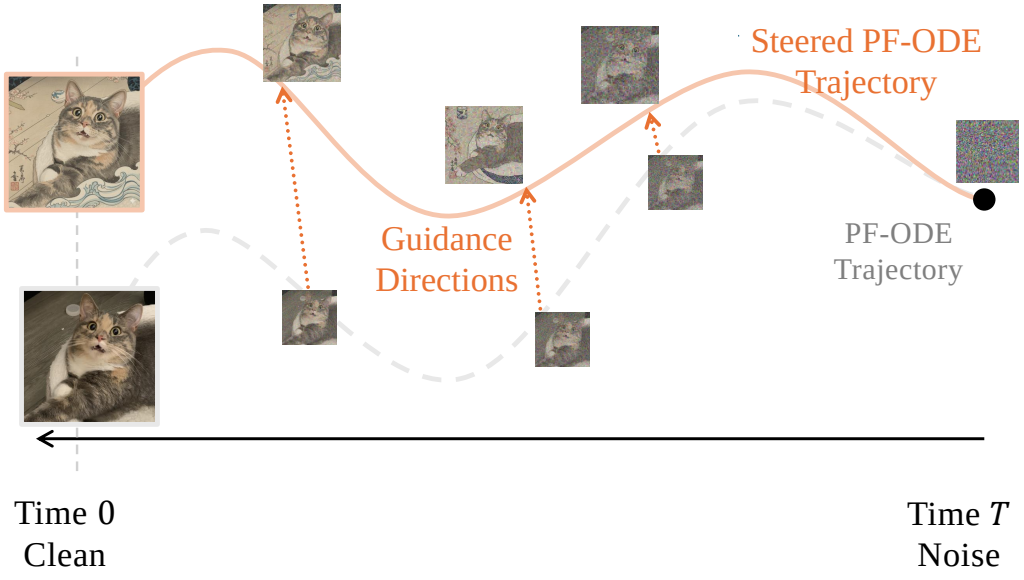


Figure 8.1: Illustration of steered diffusion sampling. Reverse-time PF-ODE sampling begins from pure noise at the right ($t = T$) and gradually evolves toward a clean sample at the left ($t = 0$). During this process, time-dependent guidance directions, weighted by w_t, modify the unconditional score field and steer the trajectory toward the desired attribute (Japanese painting style) while the sample is progressively refined from coarse to fine detail.

Source: Created by the authors.

The generation process of diffusion models proceeds in a coarse-to-fine manner, providing a flexible framework for controllable generation. At each step, a small amount of noise is removed and the sample becomes clearer, gradually revealing more structure and detail. This property enables control over the generation process: by adding a guidance term to the learned, time-dependent velocity field, we can steer the generative trajectory to reflect user intent.

A principled foundation for guidance-based sampling in diffusion models is the Bayesian decomposition of the conditional score. For each noise level t,

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}) = \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional direction}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)}_{\text{guidance direction}}. \tag{8.1.1}$$

This identity shows that the exact conditional reverse dynamics are obtained by adding the guidance term $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)$ on top of the unconditional score. We will see later on that classifier guidance (Dhariwal and Nichol, 2021) estimates this term directly, classifier-free guidance (Ho and Salimans, 2021) estimates it

implicitly as the difference between jointly learned conditional and unconditional scores, and more general training-free methods may instead use a task-specific surrogate guidance direction.

With the exact conditional score, the PF-ODE becomes

$$\begin{aligned}\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} &= f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\underbrace{\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}(t)|\mathbf{c})}_{\text{conditional score}} \\ &= f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\Big[\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}(t)) + \nabla_{\mathbf{x}_t}\log p_t(\mathbf{c}|\mathbf{x}(t))\Big].\end{aligned} \tag{8.1.2}$$

With exact scores and exact integration, these dynamics recover $p_0(\mathbf{x}_0|\mathbf{c})$ when initialized from the corresponding terminal marginal $p_T(\mathbf{x}_T|\mathbf{c})$. In standard diffusion constructions, the terminal noising is chosen so that $p_T(\mathbf{x}_T|\mathbf{c})$ is approximately matched by the usual condition-independent prior.

We formulate the discussion below in score prediction for clarity. Since the standard $\mathbf{x}$-, $\boldsymbol{\epsilon}$-, score-, and $\mathbf{v}$-prediction parameterizations are related by time-dependent linear transformations wherever these conversions are well-defined, the same guidance correction can be expressed in the other parameterizations using the relationships in Equation (6.3.1).

Instantiations of the Guidance Direction.

1. **Classifier Guidance (CG).** In Section 8.2, classifier guidance (CG) trains a time-conditional classifier $p_{\psi}(\mathbf{c}|\mathbf{x}_t, t)$ on noised data $\mathbf{x}_t$ (obtained by corrupting clean labeled samples at level t). At sampling time, its input gradient provides the guidance term:

$$\nabla_{\mathbf{x}_t}\log p_{\psi}(\mathbf{c}|\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t}\log p_t(\mathbf{c}|\mathbf{x}_t),$$

   which is then added to the unconditional score (Dhariwal and Nichol, 2021).

2. **Classifier-Free Guidance (CFG).** In Section 8.3, CFG directly trains a single conditional model

$$\mathbf{s}_{\phi}(\mathbf{x}_t, t, \mathbf{c}) \approx \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{c}),$$

   where the unconditional model is learned jointly by randomly replacing the condition with a special null token for a fraction of the training steps.

3. **Training-Free (Surrogate) Guidance.** Under the standard forward noising construction, we first draw $(\mathbf{x}_0, \mathbf{c}) \sim p_{\text{data}}$ and then sample $\mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_0)$, where the forward kernel does not depend additionally on $\mathbf{c}$. Thus $\mathbf{c} \perp \mathbf{x}_t \mid \mathbf{x}_0$[1].

[1] Indeed,

$$p_t(\mathbf{c}|\mathbf{x}_t) = \int p(\mathbf{c}, \mathbf{x}_0|\mathbf{x}_t)\,\mathrm{d}\mathbf{x}_0 = \int p(\mathbf{c}|\mathbf{x}_0, \mathbf{x}_t)\,p(\mathbf{x}_0|\mathbf{x}_t)\,\mathrm{d}\mathbf{x}_0.$$

Because the standard forward kernel satisfies $p_t(\mathbf{x}_t|\mathbf{x}_0, \mathbf{c}) = p_t(\mathbf{x}_t|\mathbf{x}_0)$, we have $p(\mathbf{c}|\mathbf{x}_0, \mathbf{x}_t) = p(\mathbf{c}|\mathbf{x}_0)$, which yields the expression below.

Consequently,

$$p_t(\mathbf{c}|\mathbf{x}_t) = \int p(\mathbf{c}|\mathbf{x}_0)\, p(\mathbf{x}_0|\mathbf{x}_t)\, \mathrm{d}\mathbf{x}_0.$$

This posterior average is generally intractable, motivating training-free methods that instead construct a task-specific surrogate guidance direction.

In Section 8.4.1, training-free (loss-based) guidance avoids evaluating the conditional likelihood $p_t(\mathbf{c}|\mathbf{x}_t)$ directly. Instead, it starts from an off-the-shelf loss $\ell(\mathbf{x}_t, \mathbf{c}; t)$ that measures how well the current sample satisfies the desired condition, with smaller loss indicating better agreement. From this loss, we define the positive *guidance potential*

$$\mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) := \exp\big(-\tau\, \ell(\mathbf{x}_t, \mathbf{c}; t)\big), \qquad \tau > 0.$$

Thus, lower-loss samples receive larger potential values. We denote the corresponding *guidance direction* by

$$\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) := \nabla_{\mathbf{x}_t} \log \mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) = -\tau \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{c}; t).$$

Classifier guidance fits naturally into the same picture. At the population level, its guidance potential is

$$\mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) = p_t(\mathbf{c}|\mathbf{x}_t),$$

which is approximated in practice by the learned classifier $p_{\psi^\times}(\mathbf{c}|\mathbf{x}_t, t)$. Equivalently, the learned classifier corresponds to the loss

$$\ell(\mathbf{x}_t, \mathbf{c}; t) = -\log p_{\psi^\times}(\mathbf{c}|\mathbf{x}_t, t), \qquad \tau = 1.$$

At the population level, the three constructions above share the same additive guidance template. Given a positive potential $\mathbf{h}_t(\mathbf{x}_t; \mathbf{c})$ and guidance strength $w_t \geq 0$, define the guided score field

$$\mathbf{s}^{\mathrm{guide}}(\mathbf{x}_t, t, \mathbf{c}; w_t) := \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + w_t \mathcal{G}_t(\mathbf{x}_t, \mathbf{c}). \tag{8.1.3}$$

For loss-based guidance, $\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) = -\tau \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{c}; t)$. For exact classifier guidance, $\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)$; classifier-free guidance expresses the same population guidance direction through the difference between conditional and unconditional scores.

At each fixed time t, Equation (8.1.3) is, whenever normalizable, the score of the tilted density

$$\widetilde{p}_t^{\mathrm{tilt}}(\mathbf{x}_t|\mathbf{c}) \propto p_t(\mathbf{x}_t) \mathbf{h}_t(\mathbf{x}_t; \mathbf{c})^{w_t}.$$

This gives a useful energy interpretation: the base density favors samples supported by the diffusion model, while the potential reweights them toward the desired condition.

In practice, the corresponding learned or surrogate version of Equation (8.1.3) is inserted into the reverse-time dynamics. Importantly, the tilted density above is only a fixed-time interpretation: the family $\{\widetilde{p}_t^{\text{tilt}}(\cdot|\mathbf{c})\}_t$ need not coincide with the marginals generated by the guided sampler. Consequently, for a generic loss or guidance schedule, integrating the guided PF-ODE from T to 0 does not in general produce samples distributed according to $\widetilde{p}_0^{\text{tilt}}(\cdot|\mathbf{c})$. The additional cross-time consistency required for this interpretation to hold is discussed in Section 8.4.2.

The effect of guidance on the sampling trajectory is illustrated in Figure 8.1. The same principle can also be applied on top of an already conditional model, allowing additional control signals to modify its sampling dynamics.

From Control to Better Alignment with Direct Preference Optimization. Strong control can be on-condition but off-preference: a sample may satisfy the conditioning signal (e.g., the prompt) yet deviate from what humans actually prefer. We formalize this by tilting the conditional target at $t = 0$ by a preference rating:

$$\widetilde{p}_0^{\text{tilt}}(\mathbf{x}_0|\mathbf{c}) \propto p_0(\mathbf{x}_0|\mathbf{c}) \exp\big(\lambda_{\text{pref}} r(\mathbf{x}_0, \mathbf{c})\big), \qquad \lambda_{\text{pref}} > 0,$$

where $r(\mathbf{x}_0, \mathbf{c})$ is a scalar alignment rating for a clean sample $\mathbf{x}_0$ and condition $\mathbf{c}$, with larger values indicating stronger preference, and λ_{pref} controls the strength of the preference reweighting. In the KL-regularized formulation introduced later, the optimizer takes this form with the reference policy as the base distribution and $\lambda_{\text{pref}} = 1/\beta$.

Existing methods for achieving such steerability typically collect human labels of the relative quality of model generations and fine-tune the conditional diffusion model to align with these preferences, often through reinforcement learning from human feedback (RLHF). However, RLHF is a complex and often unstable procedure: it first fits a reward model to capture human preferences, and then fine-tunes the conditional diffusion model with reinforcement learning to maximize this estimated reward while constraining policy drift from the original model.

This naturally raises the question: *can we remove the reward model training stage altogether?* We address this with Diffusion-DPO (Wallace *et al.*, 2024), an adaptation of Direct Preference Optimization (Rafailov *et al.*, 2023) originally developed for large language models. As described in Section 8.5, Diffusion-DPO learns the preference tilt directly from pairwise choices, so the conditional diffusion model is fine-tuned to align to preferences without a separate reward model.

8.2 Classifier Guidance

8.2.1 Foundation of Classifier Guidance

Let $\mathbf{c}$ denote a conditioning variable drawn from a distribution $p(\mathbf{c})$, such as a class label, caption, or other auxiliary information. Our goal is to draw samples from $p_0(\mathbf{x}|\mathbf{c})$. In diffusion-based conditional generation, we realize this goal by running the reverse-time dynamics whose time marginals are $p_t(\cdot|\mathbf{c})$. The drift of these dynamics depends on the conditional score

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}), \quad t \in [0, T].$$

Hence a standard and effective route[2] is to estimate this quantity.

A fundamental insight, based on Bayes' rule, is that the conditional score can be decomposed as:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}) &= \nabla_{\mathbf{x}_t} \log \left(\frac{p_t(\mathbf{x}_t) p_t(\mathbf{c}|\mathbf{x}_t)}{p(\mathbf{c})} \right) \\ &= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{c}) \\ &= \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)}_{\text{classifier gradient}}, \end{aligned} \tag{8.2.1}$$

where $p_t(\mathbf{c}|\mathbf{x}_t)$ indicates a probability of $\mathbf{c}$ conditioned on $\mathbf{x}_t$ which predicts the condition $\mathbf{c}$ from the noisy input $\mathbf{x}_t$ at time t.

This decomposition[3] motivates the *Classifier Guidance* (CG) approach proposed by Dhariwal and Nichol (2021). To separate the population identity from the learned classifier used in practice, first consider the oracle quantities. For a guidance scale $\omega \geq 0$, define at each fixed time t

$$\widetilde{p}_t^{\mathrm{CG},\omega}(\mathbf{x}_t|\mathbf{c}) \propto p_t(\mathbf{x}_t) p_t(\mathbf{c}|\mathbf{x}_t)^{\omega}. \tag{8.2.2}$$

Its score is

$$\begin{aligned} \mathbf{s}^{\mathrm{CG}}(\mathbf{x}_t, t, \mathbf{c}; \omega) &:= \nabla_{\mathbf{x}_t} \log \widetilde{p}_t^{\mathrm{CG},\omega}(\mathbf{x}_t|\mathbf{c}) \\ &= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \omega \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t). \end{aligned} \tag{8.2.3}$$

Thus ω scales the classifier-guidance direction. When $\omega = 1$, Bayes' rule gives $\widetilde{p}_t^{\mathrm{CG},1}(\mathbf{x}_t|\mathbf{c}) = p_t(\mathbf{x}_t|\mathbf{c})$, and Equation (8.2.3) recovers the exact conditional score. For a general ω, $\widetilde{p}_t^{\mathrm{CG},\omega}$ provides a fixed-time tempered interpretation of the guided field; whether it also forms the marginal path of the guided sampler is discussed in Section 8.4.2.

The scalar $\omega \geq 0$ modulates the influence of the classifier:

[2] One could in principle obtain $p_0(\mathbf{x}|\mathbf{c})$ from an unconditional generator via rejection or importance sampling if $p(\mathbf{c}|\mathbf{x})$ were available and well calibrated. This is rarely practical for high-dimensional or rare conditions.

[3] In the last identity, $\nabla_{\mathbf{x}_t} \log p(\mathbf{c}) = 0$ because $p(\mathbf{c})$ does not depend on $\mathbf{x}_t$.

- $\omega = 1$: recovers the true conditional score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c})$.
- $\omega > 1$: amplifies the classifier signal, typically increasing conditional fidelity (often at the expense of diversity).
- $0 \leq \omega < 1$: down-weights the classifier signal, typically increasing sample diversity while weakening conditioning.

Practical Approximation in CG. In practice, CG is a training-free method (w.r.t. the diffusion model) for steering a pre-trained unconditional diffusion model,

$$\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

CG is applied only at sampling time, without modifying the diffusion model itself. To enable this, a time-dependent classifier $p_{\boldsymbol{\psi}}(\mathbf{c}|\mathbf{x}_t, t)$ is trained separately to predict the condition $\mathbf{c}$ from noisy inputs $\mathbf{x}_t$ at different noise levels t. The classifier is trained in a standard way by minimizing the cross-entropy loss:

$$\mathbb{E}_{t\sim\mathcal{U}[0,T],(\mathbf{x},\mathbf{c})\sim p_{\text{data}},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\Big[-\log p_{\boldsymbol{\psi}}(\mathbf{c}|\mathbf{x}_t, t)\Big], \tag{8.2.4}$$

where $(\mathbf{x}, \mathbf{c}) \sim p_{\text{data}}$ denotes paired labeled data, and $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$ is the noisy input at time t. The classifier must be trained across noise levels and typically receives t, or an equivalent noise-level representation, as input so that it can approximate $p_t(\mathbf{c}|\mathbf{x}_t)$ throughout the diffusion path.

After training, the classifier provides scores that serves as a surrogate for the true likelihood gradient:

$$\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\psi}^\times}(\mathbf{c}|\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t).$$

8.2.2 Inference with CG

At inference time, the learned classifier gradient $\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\psi}^\times}(\mathbf{c}|\mathbf{x}_t, t)$ is added to the unconditional score estimate and scaled by ω, yielding a learned approximation to the oracle field in Equation (8.2.3):

$$\mathbf{s}^{\text{CG}}_{\text{model}}(\mathbf{x}_t, t, \mathbf{c}; \omega) := \underbrace{\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t)}_{\text{uncond. direction}} + \omega \underbrace{\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\psi}^\times}(\mathbf{c}|\mathbf{x}_t, t)}_{\text{guidance direction}} \approx \mathbf{s}^{\text{CG}}(\mathbf{x}_t, t, \mathbf{c}; \omega).$$

The unconditional score estimate in the reverse-time SDE or PF-ODE is then replaced by $\mathbf{s}^{\text{CG}}_{\text{model}}(\mathbf{x}_t, t, \mathbf{c}; \omega)$, thereby steering the generative trajectory toward samples that align with the condition $\mathbf{c}$.

8.2.3 Advantages and Limitations

CG provides a simple and flexible mechanism for conditional generation, allowing for explicit control over the strength of conditioning via ω. It can be used with any pre-trained unconditional diffusion model, requiring only an additional classifier for conditioning.

However, the approach has notable limitations:

- **Training Cost:** The classifier must be trained to operate across all noise levels, which is computationally expensive.
- **Robustness:** Classifiers must generalize well to severely corrupted inputs $\mathbf{x}_t$, especially for large t, which can be challenging.
- **Separate Training:** Since the classifier is trained independently of the diffusion model, it may not align perfectly with the learned data distribution.

8.3 Classifier-Free Guidance

8.3.1 Foundation of Classifier-Free Guidance

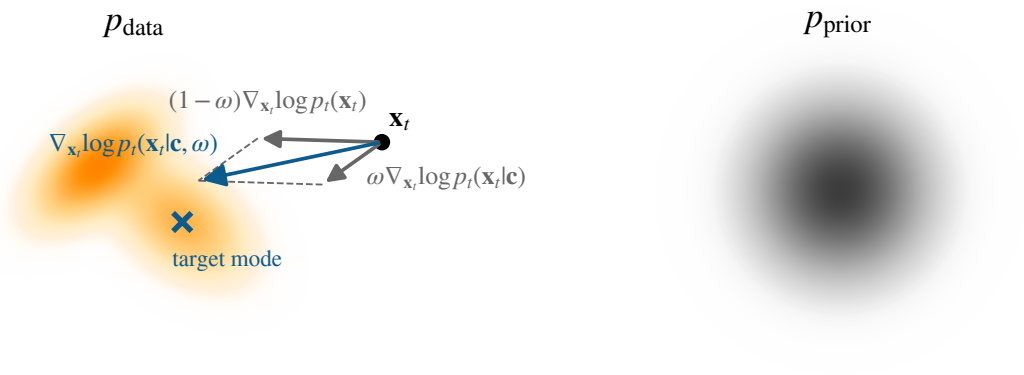


Figure 8.2: Illustration of CFG. The guided direction is a linear combination of the conditional score $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{c})$ and the unconditional score $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)$, controlled by ω. For $0 \le \omega \le 1$, this interpolates between the two directions; for $\omega > 1$, it extrapolates beyond the conditional score in the conditional direction. The resulting field is used to steer the sampling trajectory toward the conditioning signal.

Source: Created by the authors.

Classifier-free guidance (CFG) (Ho and Salimans, 2021) removes the need for a separately trained classifier by expressing the classifier-guidance direction through conditional and unconditional scores. Bayes' rule gives

$$\nabla_{\mathbf{x}_t}\log p_t(\mathbf{c}|\mathbf{x}_t) = \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{c}) - \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t). \tag{8.3.1}$$

Substituting this identity into Equation (8.2.3) gives the oracle CFG field

$$\begin{aligned}\mathbf{s}^{\text{CFG}}(\mathbf{x}_t,t,\mathbf{c};\omega) &:= \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t) + \omega\left(\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{c}) - \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)\right)\\ &= \omega\underbrace{\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{c})}_{\text{conditional score}} + (1-\omega)\underbrace{\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)}_{\text{unconditional score}}.\end{aligned} \tag{8.3.2}$$

With exact scores, Equation (8.3.2) is algebraically the same guided field as Equation (8.2.3). At $\omega = 1$, it reduces to the exact conditional score. For other guidance strengths, its fixed-time tilted-density interpretation should be distinguished from the marginals generated by the guided dynamics, as discussed in Section 8.4.2.

The hyperparameter ω again plays a critical role in controlling the influence of the conditioning information (we take $\omega \ge 0$):

- At $\omega = 0$, the model behaves as an unconditional diffusion model, completely ignoring the conditioning.
- At $\omega = 1$, the model uses the conditional score without additional guidance.

- For $\omega > 1$, the model places more emphasis on the conditional score and less on the unconditional score, strengthening alignment with $\mathbf{c}$ but typically reducing diversity.

8.3.2 Training and Sampling of CFG

Joint Training of Unconditional and Conditional Diffusion Models via CFG. CFG requires a diffusion model trained to support both conditional and unconditional predictions. Rather than training two separate models, CFG uses a single model $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t, \mathbf{c})$ and randomly replaces the conditioning input by a null token $\emptyset$ during training:

- with the null input, the model learns the unconditional prediction $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t, \emptyset)$;
- with the original condition, it learns the conditional prediction $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t, \mathbf{c})$.

The null condition is used with probability p_{uncond}, a user-specified condition-dropout probability. The full training procedure is shown in Algorithm 4, alongside standard unconditional training in Algorithm 3. The guidance weight ω is introduced only at sampling time and is not part of this training objective.

Algorithm 3 Uncond. DM

Repeat
  $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$
  $t \sim \mathcal{U}[0, T]$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$
  Take gradient step on: $\nabla_{\boldsymbol{\phi}} \left\| \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{s} \right\|^2$
until converged

Algorithm 4 CFG for Cond. DM

Input: p_{uncond}: prob. of unconditional dropout
Repeat
  $(\mathbf{x}, \mathbf{c}) \sim p_{\text{data}}(\mathbf{x}, \mathbf{c})$
  $\mathbf{c} \leftarrow \emptyset$ with prob. p_{uncond}
  $t \sim \mathcal{U}[0, T]$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$
  Take gradient step on: $\nabla_{\boldsymbol{\phi}} \left\| \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t, \mathbf{c}) - \mathbf{s} \right\|^2$
until converged

Conditioned Sampling with CFG. Once the model $\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t, \mathbf{c})$ is trained using Algorithm 4, CFG forms the learned guided field

$$\begin{aligned} \mathbf{s}^{\text{CFG}}_{\text{model}}(\mathbf{x}_t, t, \mathbf{c}; \omega) &:= \omega \underbrace{\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t, \mathbf{c})}_{\text{conditional score estimate}} + (1-\omega) \underbrace{\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t, \emptyset)}_{\text{unconditional score estimate}} \\ &\approx \mathbf{s}^{\text{CFG}}(\mathbf{x}_t, t, \mathbf{c}; \omega). \end{aligned} \tag{8.3.3}$$

During sampling, the unconditional score estimate in the reverse-time SDE or PF-ODE is replaced by $\mathbf{s}^{\text{CFG}}_{\text{model}}$. Thus CFG defines a modified sampling field by combining the learned conditional and unconditional score estimates.

This formulation enables controllable generation by adjusting ω, providing a simple post-training trade-off between conditioning strength and sample diversity without requiring a separately trained classifier or unconditional diffusion model.

8.4 Training-Free Guidance

Training-free guidance modifies the sampling dynamics of a fixed pre-trained diffusion model using an external condition, loss, or measurement objective, without retraining the diffusion model itself. The Bayes decomposition in Equation (8.1.1) provides a useful motivation in score space: guidance adds a condition-dependent direction to the unconditional score. In practice, a task-specific loss can be used to construct corrections either to the predicted clean sample or to the score used during sampling.

We first introduce these two general forms in Section 8.4.1. We then ask what distribution such surrogate-guided dynamics actually generate in Section 8.4.2, before turning to representative training-free approaches for inverse problems in Section 8.4.3.

Setup and Notations. Let $\mathbf{c}$ denote a conditioning variable. We assume access to a pre-trained diffusion model $\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)$ expressed in score prediction[4]. In addition, suppose we are given a task loss

$$\ell(\cdot, \mathbf{c}) \colon \mathbb{R}^D \to \mathbb{R}$$

that measures disagreement with the desired condition, so that smaller $\ell(\mathbf{x}, \mathbf{c})$ indicates better agreement. Examples include a perceptual distance to a reference target, or a feature-space distance or negative similarity derived from a pre-trained representation such as CLIP (Radford *et al.*, 2021).

Consider the standard linear–Gaussian forward noising kernel

$$p_t(\cdot|\mathbf{x}_0) := \mathcal{N}\big(\cdot; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}\big).$$

We borrow the DDIM update introduced later in Equation (9.2.2) as a concrete example:

$$\mathbf{x}_{t\to t-1} = \alpha_{t-1} \underbrace{\hat{\mathbf{x}}_0(\mathbf{x}_t)}_{\text{in data space}} - \sigma_{t-1}\sigma_t \underbrace{\hat{\mathbf{s}}(\mathbf{x}_t)}_{\text{in noise space}}, \tag{8.4.1}$$

where $\hat{\mathbf{x}}_0(\mathbf{x}_t) := \mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$ is the clean $\mathbf{x}$-prediction and $\hat{\mathbf{s}}(\mathbf{x}_t) := \mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)$ is the score prediction at noise level t.

8.4.1 Conceptual Framework for Training-Free Guidance

A common strategy is to use gradients of the task loss to modify either the predicted clean sample or the score term in the sampling update. To distinguish these two operations, define the data-space direction

$$\mathcal{G}_t^{\text{data}}(\mathbf{x}_t, \mathbf{c}) := -\nabla_{\mathbf{x}_0} \ell(\mathbf{x}_0, \mathbf{c})|_{\mathbf{x}_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)}.$$

[4]Here, we adopt the score and $\mathbf{x}$-prediction parameterizations for simplicity; other parameterizations such as $\boldsymbol{\epsilon}$-prediction can be handled analogously.

For the noise-space correction, we reuse the guidance direction introduced in the Prologue,

$$\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) := \nabla_{\mathbf{x}_t} \log \mathbf{h}_t(\mathbf{x}_t; \mathbf{c}),$$

for a positive guidance potential $\mathbf{h}_t$, specified below.

Using the DDIM update in Equation (8.4.1), the two corrections can be written compactly as

$$\mathbf{x}_{t\to t-1} = \alpha_{t-1} \underbrace{\left(\hat{\mathbf{x}}_0(\mathbf{x}_t) + \eta_t^{\text{data}} \mathcal{G}_t^{\text{data}}(\mathbf{x}_t, \mathbf{c})\right)}_{\text{A. data space}} - \sigma_{t-1}\sigma_t \underbrace{\left(\hat{\mathbf{s}}(\mathbf{x}_t) + \eta_t^{\text{latent}} \mathcal{G}_t(\mathbf{x}_t, \mathbf{c})\right)}_{\text{B. noise space}}, \tag{8.4.2}$$

where $\eta_t^{\text{data}}, \eta_t^{\text{latent}} \geq 0$ control the strengths of the data-space and noise-space corrections, respectively.

A. Guidance in Data Space. The data-space correction directly improves the current clean estimate by taking a local step that decreases the task loss:

$$\hat{\mathbf{x}}_0(\mathbf{x}_t) \;\longmapsto\; \hat{\mathbf{x}}_0(\mathbf{x}_t) + \eta_t^{\text{data}} \mathcal{G}_t^{\text{data}}(\mathbf{x}_t, \mathbf{c}).$$

Thus the clean prediction is first moved toward better agreement with the condition and then used in the subsequent sampling update. This correction can also be applied repeatedly when multiple optimization steps are desired.

Representative examples include MGPD (He *et al.*, 2023) and UGD (Bansal *et al.*, 2023).

B. Guidance in Noise Space. A closely related strategy applies guidance directly to the score used during sampling. Following the potential view introduced in Section 8.1, a common construction evaluates the task loss on the current denoised estimate and chooses

$$\mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) := \exp\left(-\tau\, \ell\left(\hat{\mathbf{x}}_0(\mathbf{x}_t), \mathbf{c}\right)\right), \qquad \tau > 0.$$

By the definition of $\mathcal{G}_t$ above, this gives

$$\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_t} \log \mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) = -\tau \nabla_{\mathbf{x}_t} \ell(\hat{\mathbf{x}}_0(\mathbf{x}_t), \mathbf{c}).$$

Accordingly, the score term in Equation (8.4.2) becomes

$$\hat{\mathbf{s}}(\mathbf{x}_t) + \eta_t^{\text{latent}} \mathcal{G}_t(\mathbf{x}_t, \mathbf{c}).$$

Comparing this practical update with the population-level construction in Equation (8.1.3), η_t^{latent} plays the role of the generic guidance strength w_t, while $\hat{\mathbf{s}}(\mathbf{x}_t)$ replaces the exact unconditional score by its learned estimate.

When $\mathbf{h}_t(\mathbf{x}_t; \mathbf{c})$ is proportional to the true conditional likelihood $p_t(\mathbf{c}|\mathbf{x}_t)$ up to a factor independent of $\mathbf{x}_t$, $\mathcal{G}_t(\mathbf{x}_t, \mathbf{c})$ recovers the Bayes guidance direction

$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)$. For a generic task loss, $\mathcal{G}_t$ instead defines a surrogate guidance direction.

Evaluating $\mathcal{G}_t$ requires differentiating through the $\mathbf{x}$-prediction:

$$\mathcal{G}_t(\mathbf{x}_t, \mathbf{c}) = -\tau \left[\nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)\right]^\top \nabla_{\mathbf{x}_0} \ell(\mathbf{x}_0, \mathbf{c})|_{\mathbf{x}_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)},$$

which may incur substantial computational cost in practice.

Representative examples include (Yu *et al.*, 2023; Chung *et al.*, 2022; Bansal *et al.*, 2023).

The constructions above specify how guidance modifies each sampling step. They do not, by themselves, determine the marginal distributions obtained after these modified dynamics are integrated across all noise levels. We clarify this distinction next.

8.4.2 What Distribution Does Guided Sampling Produce?

The potential view above has an exact fixed-time density interpretation at the population level. In particular, replacing the learned unconditional score estimate $\hat{\mathbf{s}}$ by the exact score, a positive potential $\mathbf{h}_t(\mathbf{x}; \mathbf{c})$ and guidance strength $w_t \geq 0$ define the ideal guided field

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + w_t \nabla_{\mathbf{x}} \log \mathbf{h}_t(\mathbf{x}; \mathbf{c}).$$

At each fixed t, this is the score of

$$\widetilde{p}_t^{\text{tilt}}(\mathbf{x}|\mathbf{c}) = \frac{1}{Z_t(\mathbf{c})} p_t(\mathbf{x}) \mathbf{h}_t(\mathbf{x}; \mathbf{c})^{w_t}, \qquad Z_t(\mathbf{c}) = \int p_t(\mathbf{x}) \mathbf{h}_t(\mathbf{x}; \mathbf{c})^{w_t} \, \mathrm{d}\mathbf{x}, \tag{8.4.3}$$

provided $0 < Z_t(\mathbf{c}) < \infty$. The learned field used in practice approximates this ideal construction through the learned score estimate $\hat{\mathbf{s}}(\mathbf{x}_t)$.

For the loss-based construction above,

$$\mathbf{h}_t(\mathbf{x}_t; \mathbf{c}) = \exp\left[-\tau \ell\left(\hat{\mathbf{x}}_0(\mathbf{x}_t), \mathbf{c}\right)\right],$$

and η_t^{latent} plays the role of w_t.

This fixed-time identity does not determine the distribution generated after the guided dynamics are integrated across noise levels. Consequently, for a generic loss and guidance schedule, solving the guided PF-ODE from T to 0 does not in general produce samples distributed according to $\widetilde{p}_0^{\text{tilt}}(\cdot|\mathbf{c})$.

The reason is cross-time consistency: densities at different noise levels cannot be chosen independently. For $\{\widetilde{p}_t^{\text{tilt}}\}_t$ to form the marginal path associated with the same forward diffusion, they must also satisfy the corresponding Fokker-Planck equation:

$$\partial_t \widetilde{p}_t^{\text{tilt}} + \nabla_{\mathbf{x}} \cdot \left(f(t)\mathbf{x} \widetilde{p}_t^{\text{tilt}}\right) - \frac{1}{2} g^2(t) \Delta_{\mathbf{x}} \widetilde{p}_t^{\text{tilt}} = 0.$$

When this condition holds, the corresponding PF-ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log \widetilde{p}_t^{\mathrm{tilt}}\left(\mathbf{x}(t)|\mathbf{c}\right)$$

has $\widetilde{p}_t^{\mathrm{tilt}}$ as its marginal path when initialized from the corresponding terminal distribution $\widetilde{p}_T^{\mathrm{tilt}}(\cdot|\mathbf{c})$.

The compatibility condition can be checked directly. Substituting Equation (8.4.3) into the Fokker-Planck equation and using the fact that p_t itself satisfies the same equation gives

$$\begin{aligned}
&\partial_t \widetilde{p}_t^{\mathrm{tilt}} + \nabla_{\mathbf{x}} \cdot \left(f(t)\mathbf{x}\widetilde{p}_t^{\mathrm{tilt}}\right) - \frac{1}{2}g^2(t)\Delta_{\mathbf{x}}\widetilde{p}_t^{\mathrm{tilt}} \\
&= \widetilde{p}_t^{\mathrm{tilt}}\Bigg[\partial_t \log(\mathbf{h}_t^{w_t}) + f(t)\mathbf{x}\cdot\nabla_{\mathbf{x}}\log(\mathbf{h}_t^{w_t}) - g^2(t)\nabla_{\mathbf{x}}\log p_t \cdot \nabla_{\mathbf{x}}\log(\mathbf{h}_t^{w_t}) \\
&\qquad\qquad - \frac{1}{2}g^2(t)\left(\Delta_{\mathbf{x}}\log(\mathbf{h}_t^{w_t}) + \left\|\nabla_{\mathbf{x}}\log(\mathbf{h}_t^{w_t})\right\|^2\right) - \frac{Z_t'}{Z_t}\Bigg],
\end{aligned}$$

where the arguments $(\mathbf{x};\mathbf{c})$ are suppressed for readability. Thus, the tilted family is a valid marginal path only when the bracketed quantity vanishes. A generic task loss and guidance schedule need not satisfy this condition. The mismatch can therefore remain even with exact score evaluation and exact ODE integration: it comes from the guided dynamics themselves rather than from numerical discretization. This is an instance of the broader Fokker-Planck consistency requirement discussed in (Lai *et al.*, 2023a).

Two simple cases clarify the distinction. When $w_t \equiv 0$, the tilt vanishes and the original unconditional path p_t is recovered. For exact classifier guidance, under the standard setting where the forward diffusion corrupts $\mathbf{x}$ while leaving $\mathbf{c}$ fixed,

$$\mathbf{h}_t(\mathbf{x};\mathbf{c}) = p_t(\mathbf{c}|\mathbf{x}), \qquad w_t \equiv 1.$$

Bayes' rule then gives

$$\widetilde{p}_t^{\mathrm{tilt}}(\mathbf{x}|\mathbf{c}) = p_t(\mathbf{x}|\mathbf{c}),$$

so the tilted family is exactly the conditional marginal path. This explains the special role of guidance weight 1 for the oracle classifier likelihood.

For a general guidance strength, for example

$$p_t(\mathbf{x})p_t(\mathbf{c}|\mathbf{x})^{\omega},$$

the expression still defines a valid fixed-time tilted density whenever it is normalizable, but it is not generally guaranteed to describe the marginals of the guided sampler. Special choices may satisfy the required compatibility condition, so $\omega \neq 1$ alone does not imply inconsistency.

Correction-based approaches such as Feynman-Kac Correctors (Skreta *et al.*, 2025) explicitly account for this cross-time mismatch when the goal is to follow a prescribed sequence of tilted distributions more faithfully.

8.4.3 Examples of Training-Free Approaches to Inverse Problems

The principle introduced in Section 8.4.1 has important applications in inverse problems. We begin with an overview of the background and then provide several concrete examples illustrating how to leverage pre-trained diffusion models for inference-time inverse problem solving.

Background on Inverse Problems. Let $\mathcal{A}$ denote a known forward operator, possibly nonlinear, and suppose that

$$\mathbf{y} = \mathcal{A}(\mathbf{x}_0) + \sigma_{\mathbf{y}}\mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{8.4.4}$$

The Bayesian objective is to characterize the posterior $p_0(\mathbf{x}_0|\mathbf{y})$, which combines the measurement model with prior knowledge about plausible clean signals. Blind inverse problems extend this setting by treating unknown parameters of $\mathcal{A}$ as additional quantities to infer.

Classical inverse-problem methods commonly combine a data-fidelity term with an analytical or hand-designed prior in a variational or Bayesian formulation. Supervised restoration methods may instead learn a direct mapping from measurements to clean signals using paired data. Pre-trained diffusion models provide a third route: a learned prior over clean data can be reused across measurement operators, with measurement information introduced only at inference time.

Pre-Trained Diffusion Models as Inverse Problem Solvers. For the measurement model above, Bayes' rule gives

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{y}) = \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{data score}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)}_{\text{measurement guidance}} . \tag{8.4.5}$$

The data score can be supplied by a pre-trained diffusion model, whereas the measurement term is generally unavailable in closed form. Training-free inverse solvers therefore introduce measurement information during sampling, but they need not all do so in the same mathematical form.

A useful distinction is between two common strategies. Some methods directly enforce measurement consistency by modifying or projecting the intermediate sample. Others approximate the likelihood score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$ and add it to the data score through Equation (8.4.5).

ILVR (Choi *et al.*, 2021). ILVR provides a simple example of the first strategy. Let ϕ_N denote its linear low-pass operator. At each reverse step, one first samples an unconditional proposal

$$\mathbf{x}'_{t-1} \sim p_{\boldsymbol{\theta}}(\mathbf{x}'_{t-1}|\mathbf{x}_t)$$

and noises the reference image to the same level,

$$\mathbf{y}_{t-1} \sim p_{t-1}(\mathbf{y}_{t-1}|\mathbf{y}).$$

The proposal is then refined according to

$$\mathbf{x}_{t-1} = \phi_N(\mathbf{y}_{t-1}) + (\mathbf{I} - \phi_N)\mathbf{x}'_{t-1}.$$

Thus, the prescribed low-frequency component comes from the noised reference, while the complementary component comes from the unconditional diffusion proposal. This is a sample-space consistency correction rather than an explicit approximation of $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$.

Diffusion Posterior Sampling (DPS) (Chung *et al.*, 2022). For a known nonlinear forward operator $\mathcal{A}$ with additive Gaussian measurement noise $\sigma_{\mathbf{y}} > 0$, *Diffusion Posterior Sampling* (DPS) approximates

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)), \tag{8.4.6}$$

where $\hat{\mathbf{x}}_0(\mathbf{x}_t) := \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t]$ is the posterior mean of the clean sample given $\mathbf{x}_t$, typically estimated from the pre-trained diffusion model using Tweedie's formula (Equation (3.3.6)).

This plug-in approximation replaces the posterior average of the measurement likelihood by its value at the posterior mean:

$$\begin{aligned} p_t(\mathbf{y}|\mathbf{x}_t) &= \int p_t(\mathbf{y}|\mathbf{x}_t, \mathbf{x}_0) p(\mathbf{x}_0|\mathbf{x}_t) \mathrm{d}\mathbf{x}_0 \\ &= \int p_t(\mathbf{y}|\mathbf{x}_0) p(\mathbf{x}_0|\mathbf{x}_t) \mathrm{d}\mathbf{x}_0 \approx p_t(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)), \end{aligned}$$

where we have used that $\mathbf{y}$ depends only on $\mathbf{x}_0$ (not on $\mathbf{x}_t$) given $\mathbf{x}_0$, and the approximation holds under the assumption that the posterior $p(\mathbf{x}_0|\mathbf{x}_t)$ is tightly peaked around its mean.

Since

$$p_t(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)) = \mathcal{N}(\mathbf{y}; \mathcal{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t)), \sigma_{\mathbf{y}}^2 \mathbf{I}),$$

we compute

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) &\approx \nabla_{\mathbf{x}_t} \log \mathcal{N}(\mathbf{y}; \mathcal{A}(\hat{\mathbf{x}}_0), \sigma_{\mathbf{y}}^2 \mathbf{I}) \\ &= -\frac{1}{2\sigma_{\mathbf{y}}^2} \nabla_{\mathbf{x}_t} \|\mathbf{y} - \mathcal{A}(\hat{\mathbf{x}}_0)\|^2 \\ &= \frac{1}{\sigma_{\mathbf{y}}^2} \left[\mathcal{J}_{\mathcal{A}}(\hat{\mathbf{x}}_0(\mathbf{x}_t)) \cdot \nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)\right]^\top (\mathbf{y} - \mathcal{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t))), \end{aligned}$$

where $\mathcal{J}_{\mathcal{A}}(\hat{\mathbf{x}}_0(\mathbf{x}_t)) := \nabla_{\mathbf{x}_0} \mathcal{A}(\mathbf{x})\big|_{\mathbf{x}=\hat{\mathbf{x}}_0(\mathbf{x}_t)}$ denotes the Jacobian of the forward operator with respect to its input. This formula propagates the gradient through the

score approximation pipeline, reflecting how the measurement likelihood changes with respect to perturbations in the noisy sample $\mathbf{x}_t$.

For linear inverse problems, this further simplifies to:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx \frac{1}{\sigma_{\mathbf{y}}^2} \left[\mathbf{A} \cdot \nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)\right]^\top \Big(\mathbf{y} - \mathbf{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t))\Big).$$

A large body of work explores diffusion-based inverse problem solvers by proposing various approximations for $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$. For a comprehensive overview, we refer readers to the survey by Daras *et al.* (2024).

A Related Supervised Iterative-Restoration Viewpoint and Its Connection to the PF-ODE. The discussion above solves inverse problems by combining a pre-trained diffusion prior with the Bayes-rule decomposition in Equation (8.4.5). More specifically, these methods use the score of the prior distribution together with the measurement model to guide the reconstruction process. At the same time, they also reflect a broader idea: instead of recovering the clean signal in one step, they gradually improve a corrupted input through many small updates. A similar high-level strategy also appears in supervised image restoration, but from a different starting point. Rather than using a pre-trained score model as a prior, these methods learn the refinement procedure directly from paired corrupted and clean examples.

A representative example is InDI (Delbracio and Milanfar, 2023). Unlike the Bayes-rule-based approaches above, InDI begins with paired data $(\mathbf{y}, \mathbf{x}_0)$, where $\mathbf{y}$ is a corrupted observation and $\mathbf{x}_0$ is the corresponding clean target. Importantly, it does not assume in general that the corruption from $\mathbf{x}_0$ to $\mathbf{y}$ is known analytically or that it is Gaussian.

The basic idea is to connect the two endpoints by the interpolation path

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{y}, \qquad t \in [0,1].$$

This path should be understood only as a convenient trajectory from the clean target to the corrupted observation. It is not a Gaussian noising process in general. Along this path, for any $0 \le s \le t$, one has the exact relation

$$\mathbf{x}_s = \Big(1 - \frac{s}{t}\Big)\mathbf{x}_0 + \frac{s}{t}\mathbf{x}_t.$$

Taking conditional expectation given $\mathbf{x}_t$ yields

$$\mathbb{E}[\mathbf{x}_s|\mathbf{x}_t] = \Big(1 - \frac{s}{t}\Big)\mathbb{E}[\mathbf{x}_0|\mathbf{x}_t] + \frac{s}{t}\mathbf{x}_t.$$

This shows that, to move from the current state $\mathbf{x}_t$ to a slightly less degraded state, it is enough to predict the clean target from $\mathbf{x}_t$.

Based on this idea, one trains a time-conditioned predictor $\mathbf{F}_{\phi}(\mathbf{x}_t, t)$ from paired examples by minimizing the regression objective

$$\min_{\phi} \mathbb{E}_{(\mathbf{x}_0,\mathbf{y})\sim p(\mathbf{x}_0,\mathbf{y}),t\sim p(t)} \Big\| \mathbf{F}_{\phi}\big((1-t)\mathbf{x}_0 + t\mathbf{y}, t\big) - \mathbf{x}_0 \Big\|_2^2,$$

where $p(\mathbf{x}_0, \mathbf{y})$ denotes the joint distribution of data pairs consisting of a clean target $\mathbf{x}_0$ and its corrupted observation $\mathbf{y}$, and $p(t)$ specifies how the interpolation time is sampled during training. That is, $\mathbf{F}_{\phi}$ is trained to predict the clean target from an intermediate degraded state along the interpolation path. For this squared-error loss, the Bayes-optimal predictor is the conditional expectation

$$\mathbf{F}^*(\mathbf{x}_t, t) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t].$$

Therefore, if the model class is sufficiently expressive and the optimization is successful, the learned predictor approximates the average clean target associated with the intermediate state $\mathbf{x}_t$ at time t. In this sense, the training objective is similar in spirit to diffusion-model training: in both cases, one learns a time-conditioned regression from an intermediate corrupted state to a cleaner target. The difference is that diffusion models generate such training pairs synthetically from a prescribed forward corruption process, whereas here the endpoint pairs $(\mathbf{y}, \mathbf{x}_0)$ come directly from supervised restoration data.

At inference time, one starts from $\hat{\mathbf{x}}_1 = \mathbf{y}$ and repeatedly refines the estimate by

$$\hat{\mathbf{x}}_{t-\Delta t} = \frac{\Delta t}{t}\mathbf{F}_{\phi}(\hat{\mathbf{x}}_t, t) + \Big(1 - \frac{\Delta t}{t}\Big)\hat{\mathbf{x}}_t.$$

Hence, the final reconstruction is produced by gradual refinement rather than by a single direct prediction. In the continuous-time limit, this leads to InDI's residual-flow ODE

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\mathbf{x}_t - \mathbf{F}_{\phi}(\mathbf{x}_t, t)}{t}.$$

Now suppose we further specialize to the Gaussian denoising setting, where the corruption is assumed to take the known analytic form

$$\mathbf{y} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

with known noise level σ. Under this specialization, InDI's residual-flow ODE reduces exactly to the probability-flow ODE. Indeed, the interpolation path becomes

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{y} = \mathbf{x}_0 + t\sigma\boldsymbol{\epsilon},$$

which is precisely the standard diffusion form

$$\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}, \qquad \text{with} \qquad \alpha_t = 1,\ \sigma_t = t\sigma.$$

Therefore, the predictor $\mathbf{F}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ is now learning to denoise a Gaussian-corrupted intermediate state $\mathbf{x}_t$. In this case, the residual-flow ODE becomes

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\mathbf{x}_t - \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]}{t} = -\dot{\sigma}_t \sigma_t \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t),$$

where the last equality follows from $\sigma_t = t\sigma$ together with Tweedie's formula,

$$\mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t] = \mathbf{x}_t + \sigma_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

This is exactly the PF-ODE for this Gaussian denoising path, written in either denoiser form or score form. Thus, under the Gaussian denoising specialization, InDI suggests an alternative interpretation of diffusion modeling: one may view it as supervised denoiser learning on Gaussian-corrupted data, with the PF-ODE arising as the continuous-time limit of gradual refinement.

8.5 From Reinforcement Learning to Direct Preference Optimization for Model Alignment

In the pursuit of aligning generative models with human intent, the prevailing paradigm has been Reinforcement Learning from Human Feedback (RLHF). While effective, RLHF is a complex, multi-stage process that can be unstable. This section introduces *Direct Preference Optimization (DPO)* (Rafailov *et al.*, 2023), a more streamlined and stable method that reaches the same goal without explicit reward modeling or reinforcement learning. We then outline its extension to diffusion models via *Diffusion-DPO* (Wallace *et al.*, 2024).

8.5.1 The Motivation: Circumventing the Pitfalls of RLHF

The goal of alignment is to steer a base, pre-trained model (e.g., an SFT model) toward outputs that humans prefer. RLHF proceeds in three stages. First, *supervised fine-tuning (SFT)* trains a base model on prompt–response pairs. Second, *reward modeling (RM)* fits a model on preference data consisting of prompts $\mathbf{c}$ and paired responses (a preferred "winner" $\mathbf{x}_w$ and a dispreferred "loser" $\mathbf{x}_l$), learning a scalar $r(\mathbf{c}, \mathbf{x})$ with $r(\mathbf{c}, \mathbf{x}_w) > r(\mathbf{c}, \mathbf{x}_l)$. Third, *RL fine-tuning* optimizes the SFT model (policy π[5]) with an algorithm such as PPO (Schulman *et al.*, 2017), maximizing expected reward from r while regularizing by a KL penalty that keeps π close to the reference/SFT distribution.

Despite its impact, this pipeline suffers from drawbacks: the RL stage is unstable and computationally expensive because it is on-policy—each update requires freshly generated samples from the current model; it also requires training and hosting multiple large models (SFT, reward, and sometimes a value model); and it optimizes only a proxy for human preferences, so flaws in the reward model can be exploited. This motivates a central question:

Question 8.5.1

Can we eliminate explicit reward modeling and the unstable RL step, directly optimizing the model on preference data?

Direct Preference Optimization (DPO) streamlines alignment by replacing the multi-stage RLHF pipeline with a single, supervised-style step. Instead of training a separate reward model and running unstable RL algorithms like PPO, DPO directly fits the policy to preference pairs using a simple logistic loss, while staying close to a fixed reference model. The key insight is that the KL-regularized RLHF objective can be rewritten so that the log-likelihood ratio between the policy and the reference acts as an implicit reward. This preserves the same regularization toward the reference policy but avoids costly rollouts and explicit reward modeling.

[5] A policy maps a prompt/history (state) to a distribution over responses/actions.

In Section 8.5.2, we briefly review the RLHF pipeline and its reliance on large reward models and RL fine-tuning. In Section 8.5.3, we present DPO, originally proposed for language models, which circumvents reward model training and simplifies alignment fine-tuning. Finally, in Section 8.5.4, we extend this idea to diffusion models, introducing Diffusion-DPO as a practical and stable alignment method in the generative modeling setting.

8.5.2 RLHF: Bradley-Terry View

Short Introduction to RLHF. RLHF begins with a learned judge: a reward model $r_{\boldsymbol{\psi}}$ that assigns a scalar preference score to candidate responses for the same prompt $\mathbf{c}$. The dataset $\mathcal{D}$ consists of pairs $(\tilde{\mathbf{x}}, \mathbf{x})$ annotated with a label y indicating whether $\tilde{\mathbf{x}}$ is preferred over $\mathbf{x}$. The label can be binary $y \in \{0, 1\}$ or a soft value $y \in [0, 1]$ obtained by aggregating multiple raters. The training objective is a simple logistic loss

$$\begin{aligned}\mathcal{L}_{\mathrm{RM}}(\boldsymbol{\psi}) = -\mathbb{E}_{(\mathbf{c},\tilde{\mathbf{x}},\mathbf{x},y)\sim\mathcal{D}}\Big[&y \log \sigma\big(r_{\boldsymbol{\psi}}(\mathbf{c}, \tilde{\mathbf{x}}) - r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x})\big) \\ &+ (1 - y) \log\big(1 - \sigma\big(r_{\boldsymbol{\psi}}(\mathbf{c}, \tilde{\mathbf{x}}) - r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x})\big)\big)\Big],\end{aligned} \tag{8.5.1}$$

where $\sigma(u) = 1/(1 + e^{-u})$. In practice, preference pairs in $\mathcal{D}$ may originate from various sources: curated responses, model snapshots at different checkpoints, or generations from a pre-trained conditional diffusion model. A standard convention is to store them in an ordered format $(\text{winner}, \text{loser})$. Under this convention we simply set $y = 1$, and Equation (8.5.1) reduces to the special case (with $\tilde{\mathbf{x}} = \mathbf{x}^w$ and $\mathbf{x} = \mathbf{x}^l$):

$$\mathcal{L}_{\mathrm{RM}}(\boldsymbol{\psi}) = -\mathbb{E}_{(\mathbf{c},\mathbf{x}_w,\mathbf{x}_l)\sim\mathcal{D}}\Big[\log \sigma\big(r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x}_w) - r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x}_l)\big)\Big]. \tag{8.5.2}$$

Bradley-Terry View and KL Connection. It is standard to interpret

$$p_{r_{\boldsymbol{\psi}}}(\tilde{\mathbf{x}} \succ \mathbf{x}|\mathbf{c}) := \sigma\big(r_{\boldsymbol{\psi}}(\mathbf{c}, \tilde{\mathbf{x}}) - r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x})\big)$$

through the Bradley-Terry (BT) model (Bradley and Terry, 1952), which converts two scalar scores into a win probability. This formulation highlights two key properties: (i) only the *difference* of scores matters (so $r_{\boldsymbol{\psi}}(\mathbf{c}, \cdot)$ is shift-invariant), and (ii) the loss pushes the predicted winner's score above the loser's score. To see (ii) intuitively, consider one pair with label $y \in \{0, 1\}$ and define

$$\Delta r := r_{\boldsymbol{\psi}}(\mathbf{c}, \tilde{\mathbf{x}}) - r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x}), \quad p := \sigma(\Delta r), \quad \sigma(u) = \tfrac{1}{1+e^{-u}}.$$

The per-example logistic loss is

$$\ell = -\big[y \log p + (1 - y) \log(1 - p)\big].$$

Then

$$\frac{\partial \ell}{\partial \Delta r} = \sigma(\Delta r) - y.$$

Under gradient descent with step size $\eta > 0$, the score gap updates as

$$\Delta r \leftarrow \Delta r - \eta(\sigma(\Delta r) - y).$$

Hence, if $y = 1$ ("$\tilde{\mathbf{x}}$ wins"), then $\sigma(\Delta r) - 1 \leq 0$, so Δr increases (winner up, loser down); if $y = 0$, Δr decreases.

Each per-example term in Equation (8.5.1) can be viewed as the cross-entropy between the observed Bernoulli label and the model's predicted win probability:

$$-\left[y \log p_{r_{\boldsymbol{\psi}}} + (1-y)\log(1-p_{r_{\boldsymbol{\psi}}})\right] = \mathcal{D}_{\mathrm{KL}}\big(\mathrm{Bern}(y) \| \mathrm{Bern}(p_{r_{\boldsymbol{\psi}}})\big) + \mathcal{H}(\mathrm{Bern}(y)),$$

where $\mathcal{H}$ is the entropy of the target Bernoulli distribution. Averaging over the dataset $\mathcal{D}$ gives

$$\mathcal{L}_{\mathrm{RM}}(\boldsymbol{\psi}) = \mathbb{E}_{\mathcal{D}}\Big[\mathcal{D}_{\mathrm{KL}}\Big(\mathrm{Bern}(y) \| \mathrm{Bern}(p_{r_{\boldsymbol{\psi}}})\Big)\Big] + \underbrace{\mathbb{E}_{\mathcal{D}}\big[\mathcal{H}(\mathrm{Bern}(y))\big]}_{\text{independent of } \boldsymbol{\psi}}. \tag{8.5.3}$$

Thus, minimizing the logistic loss is equivalent to minimizing the KL divergence between the empirical Bernoulli distribution of human labels and the model's predicted Bernoulli distribution. In the binary case ($y \in \{0, 1\}$), this equivalence is exact; for soft labels ($y \in [0, 1]$), the result holds up to an entropy constant offset. Intuitively, the reward model is trained to adjust its win probabilities until they align with the empirical human win rates observed in the dataset.

From this point onward, we adopt the most common convention where $\mathcal{D}$ stores pairs in an ordered format: $(\mathbf{x}^w, \mathbf{x}^l, \mathbf{c}) \sim \mathcal{D}$. Under this convention, the label is always $y = 1$, and the loss simplifies to the ordered form given in Equation (8.5.2), which we will use in the following discussion.

KL Regularized Policy Optimization (with Fixed Reward). With the fitted reward $r := r_{\boldsymbol{\psi}^\times}$ trained via Equation (8.5.2), and a conditional pre-trained diffusion model $p_{\boldsymbol{\phi}^\times}(\mathbf{x}|\mathbf{c})$, RLHF then adjusts a learnable policy $\pi_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$, usually fine-tuned on top of $p_{\boldsymbol{\phi}^\times}(\mathbf{x}|\mathbf{c})$, toward higher-reward responses. At the same time, the policy is regularized to stay close to a reference model, taken as the pre-trained diffusion model $\pi_{\mathrm{ref}}(\mathbf{x}|\mathbf{c}) := p_{\boldsymbol{\phi}^\times}(\mathbf{x}|\mathbf{c})$, using a $\mathcal{D}_{\mathrm{KL}}$ penalty:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{c}\sim p(\mathbf{c})}\Big[\mathbb{E}_{\mathbf{x}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{c})}\left[r_{\boldsymbol{\psi}}(\mathbf{c}, \mathbf{x})\right] - \beta \mathcal{D}_{\mathrm{KL}}\big(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{c}) \| \pi_{\mathrm{ref}}(\cdot|\mathbf{c})\big)\Big], \tag{8.5.4}$$

which makes the two forces explicit: seek samples the judge prefers, but stay close to the pre-trained reference.

We remark that the reward objective in Equation (8.5.2) uses only labeled pairs and does not require that $\mathcal{D}$ be generated by the reference model (i,e., the

pre-trained conditional diffusion model). While not required, collecting pairs from models close to the intended policy can reduce distribution shift and make the learned reward more reliable in the region where it will be used.

In summary, RLHF proceeds in two stages: first fit the reward r^* by minimizing the loss in Equation (8.5.2) (equivalently, the expected binary $\mathcal{D}_{\mathrm{KL}}$ in Equation (8.5.3)); then optimize the policy π^* by solving Equation (8.5.4).

8.5.3 DPO Framework

The Bridge from RLHF. The KL-regularized policy objective in Equation (8.5.4) has a simple closed-form solution for each prompt $\mathbf{c}$, given the fitted reward $r := r_{\psi^\times}$, expressed in the following energy-based form (Peters *et al.*, 2010):

$$\pi^*(\mathbf{x}|\mathbf{c}) = \frac{1}{Z(\mathbf{c})}\pi_{\text{ref}}(\mathbf{x}|\mathbf{c})\exp(r(\mathbf{c},\mathbf{x})/\beta), \tag{8.5.5}$$

where $\pi_{\text{ref}}(\mathbf{x}|\mathbf{c}) := p_{\phi^\times}(\mathbf{x}|\mathbf{c})$, and $Z(\mathbf{c})$ is the partition function ensuring $\int \pi^*(\mathbf{x}|\mathbf{c})\,\mathrm{d}\mathbf{x} = 1$.

For smaller β, $\exp(r/\beta)$ becomes sharper, so π^* concentrates on high reward regions: reward dominates, the policy moves farther from π_{ref}, diversity decreases, and training may become unstable or prone to reward hacking. For larger β, $\exp(r/\beta)$ flattens, keeping π^* closer to π_{ref}: the KL term dominates, updates are conservative, diversity follows the reference, but reward gains are limited.

Since our aim is to fine-tune the policy directly (without training a separate reward model), Equation (8.5.5) lets us *define* an *implicit reward* from any policy. We introduce below:

Defining an Implicit Reward Motivated by Inverting Equation (8.5.5). Equation (8.5.5) suggests an immediate inversion: for any policy π (with support contained in π_{ref}), define

$$r_\pi(\mathbf{c},\mathbf{x}) = \beta\log\frac{\pi(\mathbf{x}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}|\mathbf{c})} + \beta\log Z(\mathbf{c}). \tag{8.5.6}$$

Then Equation (8.5.5) holds with π in place of π^*, i.e., π would be the optimizer of Equation (8.5.4) for the reward function r_π. In this sense, r_π is an *implicit (policy-induced) reward*: it is identified up to the prompt-dependent constant $\beta\log Z(\mathbf{c})$, which vanishes in any pairwise comparison such as in the BT model:

$$r_\pi(\mathbf{c},\mathbf{x}_w) - r_\pi(\mathbf{c},\mathbf{x}_l) = \beta\Big(\log\frac{\pi(\mathbf{x}_w|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_w|\mathbf{c})} - \log\frac{\pi(\mathbf{x}_l|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_l|\mathbf{c})}\Big).$$

This cancellation is exactly what makes the constant irrelevant for preference learning and leads directly to the DPO loss on log-probability differences.

DPO's Training Loss. Plug the implicit reward Equation (8.5.6) into the BT model of Equation (8.5.2) for a labeled pair $(\mathbf{x}_w,\mathbf{x}_l)$ under the same prompt $\mathbf{c}$. The constants $\log Z(\mathbf{c})$ cancel between winner and loser, yielding a single logistic-loss objective on log-probability differences:

$$\mathcal{L}_{\text{DPO}}(\boldsymbol{\theta};\pi_{\text{ref}}) = -\mathbb{E}_{(\mathbf{c},\mathbf{x}_w,\mathbf{x}_l)\sim\mathcal{D}}\left[\log\sigma\Big(\beta\big(\log\frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_w|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_w|\mathbf{c})} - \log\frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_l|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_l|\mathbf{c})}\big)\Big)\right].$$

In words: DPO pushes up the (temperature-scaled) advantage of the winner over the loser, measured as the difference of log-likelihood improvements over the reference:

$$-\log\sigma\left(\beta\left[\text{log-ratio difference of } \tfrac{\pi_{\boldsymbol{\theta}}}{\pi_{\text{ref}}} \text{ at } \mathbf{x}_w \text{ vs. } \mathbf{x}_l\right]\right).$$

This achieves the goal of RLHF in a single, stable classification-style fine-tuning stage, without training an explicit reward model.

8.5.4 Diffusion-DPO

Throughout this subsection, we retain the policy notation from Section 8.5.3: $\pi_{\boldsymbol{\theta}}$ and π_{ref} denote the learned and reference diffusion policies, including their path and transition distributions, while $p_t(\mathbf{x}_t|\mathbf{x}_0)$ continues to denote the fixed forward noising process.

Why Naive DPO Fails for Diffusion Models? Evaluating the sample likelihood $\pi_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$ in diffusion models requires the instantaneous change-of-variables formula (divergence of the drift) of ODE solving (see Equation (4.3.7))[6], which is computationally intensive. Moreover, differentiating through the entire sampling trajectory can suffer from vanishing or exploding gradients. To avoid these issues, Diffusion-DPO works at the *path* level. We take the discrete-time diffusion model (e.g., DDPM) as an illustrative example; the continuous-time diffusion model is analogous.

Defining Pathwise Implicit Rewards. Let a trajectory be $\mathbf{x}_{0:T} := (\mathbf{x}_T, \ldots, \mathbf{x}_0)$ under the reverse-time Markov chain with conditionals $\pi(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})$. Here, $\mathbf{x}_T$ is the highest-noise state and $\mathbf{x}_0$ is the clean output.

Human preferences are defined on the final output through $r(\mathbf{c}, \mathbf{x}_0)$. Since $\mathbf{x}_0$ is part of every diffusion trajectory, the reward itself does not need to change when we move to path space. The useful step is instead to regularize the entire reverse process. By the chain rule for KL divergence, the KL divergence between the full path distributions is at least as large as the KL divergence between their endpoint marginals. This motivates the pathwise objective

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{c}\sim p(\mathbf{c})}\Big[\mathbb{E}_{\mathbf{x}_{0:T}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{c})}\big[r(\mathbf{c}, \mathbf{x}_0)\big] - \beta\mathcal{D}_{\text{KL}}\big(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{c})\big\|\pi_{\text{ref}}(\cdot|\mathbf{c})\big)\Big],$$

where $\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{c})$ and $\pi_{\text{ref}}(\cdot|\mathbf{c})$ denote the corresponding path distributions. The reward term is unchanged, while the KL penalty now also controls how the latent denoising trajectories deviate from the reference.

[6] In discrete-time diffusion models (e.g., DDPM), evaluating $\pi_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{c})$ requires marginalizing over the latent reverse trajectory $\mathbf{x}_{1:T}$.

At the distributional level, assuming the normalizer is finite, the optimizer for each prompt $\mathbf{c}$ has the energy-based form

$$\pi^*(\mathbf{x}_{0:T}|\mathbf{c}) = \frac{1}{Z(\mathbf{c})}\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})\exp\left(r(\mathbf{c},\mathbf{x}_0)/\beta\right). \tag{8.5.7}$$

As in ordinary DPO, we can invert this relation and define the *implicit path reward* induced by any policy π:

$$R_\pi(\mathbf{c},\mathbf{x}_{0:T}) := \beta\log\frac{\pi(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} + \beta\log Z(\mathbf{c}).$$

At the pathwise optimum, $R_{\pi^*}(\mathbf{c},\mathbf{x}_{0:T}) = r(\mathbf{c},\mathbf{x}_0)$. For a general policy, however, R_π may also depend on the latent trajectory. Since the preference data compare only the clean endpoints, we therefore average this implicit path reward over trajectories associated with each endpoint.

From Pathwise Implicit Rewards to DPO. For a labeled endpoint pair $(\mathbf{x}_0^w, \mathbf{x}_0^l)$ under the same prompt $\mathbf{c}$, the Bradley–Terry margin is obtained by comparing the conditional averages of their implicit path rewards:

$$\begin{aligned}
\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta};\pi_{\text{ref}}) &:= -\mathbb{E}_{(\mathbf{c},\mathbf{x}_0^w,\mathbf{x}_0^l)\sim\mathcal{D}}\left[\log\sigma\big(\Delta R(\mathbf{c};\boldsymbol{\theta})\big)\right], \quad \text{where} \\
\Delta R(\mathbf{c};\boldsymbol{\theta}) &:= \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}^w\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^w,\mathbf{c})}\Big[R_{\pi_{\boldsymbol{\theta}}}\big(\mathbf{c},(\mathbf{x}_0^w,\mathbf{x}_{1:T}^w)\big)\Big]}_{\text{winner path expectation}} \\
&\quad - \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}^l\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^l,\mathbf{c})}\Big[R_{\pi_{\boldsymbol{\theta}}}\big(\mathbf{c},(\mathbf{x}_0^l,\mathbf{x}_{1:T}^l)\big)\Big]}_{\text{loser path expectation}}.
\end{aligned} \tag{8.5.8}$$

Here, the expectation $\mathbb{E}_{\mathbf{x}_{1:T}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0,\mathbf{c})}[\cdot]$ means: given a fixed endpoint $\mathbf{x}_0$ (e.g., the winner $\mathbf{x}_0^w$) from the dataset, we take an expectation over latent denoising trajectories $\mathbf{x}_{1:T}$ under the model-induced conditional path distribution (the posterior over reverse-time trajectories) that, with kernels $\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{c})$, could produce $\mathbf{x}_0$. Since these intermediate states are unobserved, we average the path reward over all such trajectories.

However, Equation (8.5.8) is impractical for three practical reasons:

1. **Endpoint Conditioning Induces an Intractable Path Posterior.** The term $\mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{1:T}|\mathbf{x}_0,\mathbf{c})}[\cdot]$ averages over reverse paths constrained to hit $\mathbf{x}_0$, whereas the sampler runs $\mathbf{x}_T \to \cdots \to \mathbf{x}_0$ without this constraint. Conditioning on the endpoint creates a diffusion-bridge posterior with generally no closed form and costly sampling.

2. **Nested, $\boldsymbol{\theta}$-Coupled Expectations.** The loss $-\log\sigma(\Delta R(\mathbf{c};\boldsymbol{\theta}))$ with

$$\Delta R = \mathbb{E}_{\text{paths}|\mathbf{x}_0^w,\mathbf{c}}[R_{\pi_{\boldsymbol{\theta}}}] - \mathbb{E}_{\text{paths}|\mathbf{x}_0^l,\mathbf{c}}[R_{\pi_{\boldsymbol{\theta}}}]$$

has both the path joint distribution and the integrand $R_{\pi_{\boldsymbol{\theta}}}$ depending on $\boldsymbol{\theta}$. Thus $\nabla_{\boldsymbol{\theta}}$ must differentiate through the sampling distribution, leading to REINFORCE/pathwise couplings and high-variance gradients.

3. **Long Chains, Large Sums, and Expensive Backpropagation.** In $R_{\pi_{\boldsymbol{\theta}}}(\mathbf{c}, \mathbf{x}_{0:T})$, computing
$$\beta \left[\log \pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}|\mathbf{c}) - \log \pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})\right]$$
requires $\mathcal{O}(T)$ per-step log-densities with $T \sim 10^2$–10^3, for both policy $\pi_{\boldsymbol{\theta}}$ and reference π_{ref}, and for both winner/loser paths. Backpropagating through these stochastic chains (or bridge samplers) is memory and compute heavy and can be unstable; repeating this over many samples per pair and across all triplets pushes training beyond practical budgets.

Toward a Tractable Surrogate for Equation (8.5.8). To make this computable, we apply a key mathematical insight. By leveraging properties of diffusion models and applying Jensen's inequality, we can optimize a tractable upper bound on this loss. This transforms the problem from evaluating an entire path's likelihood to evaluating an expectation over the individual, single-step transitions within the path:

Because $-\log\sigma(\cdot)$ is convex, Jensen's inequality yields an upper bound by moving the inner expectations outside the nonlinear function:

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E}_{(\mathbf{c},\mathbf{x}_0^w,\mathbf{x}_0^l)\sim\mathcal{D}} \mathbb{E}_{\substack{\mathbf{x}_{1:T}^w\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^w,\mathbf{c}) \\ \mathbf{x}_{1:T}^l\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^l,\mathbf{c})}} \left[\log\sigma\Big(R_{\pi_{\boldsymbol{\theta}}}(\mathbf{c}, \mathbf{x}_{0:T}^w) - R_{\pi_{\boldsymbol{\theta}}}(\mathbf{c}, \mathbf{x}_{0:T}^l)\Big)\right].$$

Using the implicit-reward identity $R_{\pi_{\boldsymbol{\theta}}} = \beta \log \frac{\pi_{\boldsymbol{\theta}}}{\pi_{\text{ref}}} + \beta \log Z(\mathbf{c})$ and cancellation of the constant between winner and loser, the bound becomes

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E}\left[\log\sigma\Big(\beta\big(\log\frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}^w|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}^w|\mathbf{c})} - \log\frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}^l|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}^l|\mathbf{c})}\big)\Big)\right]. \quad (8.5.9)$$

A Tractable Surrogate (Stepwise Form). We now exploit the Markov property of the reverse process to decompose the upper bound of $\mathcal{L}_{\text{Diff-DPO}}$. This allows us to express the path-level preference as a sum of per-step contributions, converting the intractable pathwise loss into a tractable single-step estimator. The resulting form yields a DPO-style log-sigmoid loss whose inner margin reduces to a DSM-style MSE difference. Concretely, for the reverse chain,

$$\pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}|\mathbf{c}) = \pi_{\boldsymbol{\theta}}(\mathbf{x}_T|\mathbf{c}) \prod_{t=1}^{T} \pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}),$$
$$\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c}) = \pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c}) \prod_{t=1}^{T} \pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}).$$

Hence

$$\frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_T|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c})} \prod_{t=1}^{T} \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}.$$

If the prior at time T is the same for both models, $\pi_{\boldsymbol{\theta}}(\mathbf{x}_T|\mathbf{c}) = \pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c})$, then the first factor equals 1, and taking logs yields

$$\log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} = \sum_{t=1}^{T} \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}.$$

It follows that the bound in Equation (8.5.9) can be written as

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E}\left[\log \sigma\Big(\beta \sum_{t=1}^{T} \Delta_t\Big)\right],$$

where each per-step contribution is

$$\Delta_t = \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}^w|\mathbf{x}_t^w, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}^w|\mathbf{x}_t^w, \mathbf{c})} - \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}^l|\mathbf{x}_t^l, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}^l|\mathbf{x}_t^l, \mathbf{c})}.$$

To reduce the sum over timesteps to a single randomly selected transition, observe that $\beta \sum_{s=1}^{T} \Delta_s = \mathbb{E}_{t\sim\mathcal{U}\{1,\ldots,T\}}[\beta T \Delta_t]$. Since $-\log\sigma(\cdot)$ is convex, Jensen's inequality therefore gives, after taking the existing outer expectations,

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E}\left[\log \sigma(\beta T \Delta_t)\right],$$

where the expectation is over $(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D}$, $t \sim \mathcal{U}\{1, \ldots, T\}$, and $\mathbf{x}_{t-1:t}^w \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^w, \mathbf{c})$ and $\mathbf{x}_{t-1:t}^l \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_0^l, \mathbf{c})$. Although the bound now depends on only one transition, sampling these endpoint-conditioned transition pairs remains intractable.

Following Diffusion-DPO, we approximate the intractable model-induced conditional path using the fixed forward noising process. For each clean endpoint $\mathbf{x}_0$, we first sample $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$. Conditioned on $(\mathbf{x}_0, \mathbf{x}_t)$, the forward process induces the tractable posterior $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. Averaging the reverse-transition log-ratio under this posterior gives

$$\begin{aligned}
&\mathbb{E}_{p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\left[\log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}\right] \\
&= -\mathcal{D}_{\text{KL}}\left(p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, \pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})\right) + \mathcal{D}_{\text{KL}}\left(p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, \pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})\right).
\end{aligned}$$

For Gaussian reverse kernels with the same prescribed reverse variance, these KL divergences reduce to weighted denoising regression errors. Under $\boldsymbol{\epsilon}$-prediction,

$$\mathbb{E}_{p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\left[\log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}\right] = -\lambda_t \left[\|\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\|^2 - \|\hat{\boldsymbol{\epsilon}}_{\text{ref}}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\|^2\right], \tag{8.5.10}$$

where $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\lambda_t > 0$ collects the schedule-dependent weighting factors.

For notation simplicity, define for any noised sample $(\mathbf{x}_t, \boldsymbol{\epsilon})$:

$$\Delta\mathrm{MSE}(\mathbf{x}_t; \boldsymbol{\epsilon}) := \left\|\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\right\|^2 - \left\|\hat{\boldsymbol{\epsilon}}_{\mathrm{ref}}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\right\|^2.$$

This yields the practical single-step Diffusion-DPO surrogate for $\mathcal{L}_{\text{Diff-DPO}}$:

$$\tilde{\mathcal{L}}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\mathrm{ref}}) := \\ - \mathbb{E}_{\substack{(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D} \\ t \sim \mathcal{U}\{1, \ldots, T\}, \boldsymbol{\epsilon}^w, \boldsymbol{\epsilon}^l \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}} \Big[\log \sigma \Big(\beta T \lambda_t \big(\Delta\mathrm{MSE}(\mathbf{x}_t^l; \boldsymbol{\epsilon}^l) - \Delta\mathrm{MSE}(\mathbf{x}_t^w; \boldsymbol{\epsilon}^w) \big) \Big) \Big],$$

where $\mathbf{x}_t^w = \alpha_t \mathbf{x}_0^w + \sigma_t \boldsymbol{\epsilon}^w$ and $\mathbf{x}_t^l = \alpha_t \mathbf{x}_0^l + \sigma_t \boldsymbol{\epsilon}^l$. In practice, one may also use shared noise $\boldsymbol{\epsilon}^w = \boldsymbol{\epsilon}^l$ as a variance-reduction variant; this does not change the overall DPO-style form of the objective, but slightly alters the stochastic estimator.

Intuitively, minimizing $\tilde{\mathcal{L}}_{\text{Diff-DPO}}$ increases the reference-normalized winner-over-loser margin. Equivalently, it encourages the model to reduce denoising error on the winner more than on the loser, measured relative to the frozen reference model. The reference model does not contribute a direct gradient term, but it remains inside the margin within the $\log \sigma(\cdot)$ nonlinearity, so it still shapes the DPO-style per-example weighting.

8.6 Closing Remarks

This chapter has shifted our focus from foundational principles to the practical challenge of controllable generation. We established a unified framework for guidance based on the Bayesian decomposition of the conditional score, which elegantly separates the generative process into an unconditional direction and a steering term.

We saw this principle manifest in several powerful techniques. We covered methods that require dedicated training, such as Classifier Guidance (CG), which uses an external classifier, and the more efficient Classifier-Free Guidance (CFG), which learns conditional and unconditional scores within a single model. We also explored flexible training-free guidance methods, which can steer a pre-trained model at inference time by defining a surrogate guidance potential from a task-specific loss, enabling applications from artistic control to solving inverse problems without any retraining.

Beyond simple conditioning, we delved into the nuanced task of aligning model outputs with human preferences. After reviewing the standard but complex RLHF pipeline, we introduced Direct Preference Optimization (DPO) and its novel adaptation, Diffusion-DPO, as a more direct and stable alternative. This approach elegantly bypasses the need for an explicit reward model and reinforcement learning by deriving a loss directly from preference data.

Through these techniques, we have assembled a powerful toolkit for steering the generative process. However, a major practical hurdle remains untouched: the significant computational cost and latency of the iterative sampling process itself. Having addressed what to generate, we now turn to the equally important question of how fast we can generate it. The next chapter will tackle this challenge directly:

1. We will leverage the insight that sampling is equivalent to solving an ODE to explore sophisticated numerical solvers designed to drastically reduce the number of required steps.

2. We will investigate a sequence of influential methods, including DDIM, DEIS, and the DPM-Solver family, which have made diffusion models far more practical by accelerating sampling speed by orders of magnitude.

9

Numerical Solvers for Fast Diffusion Sampling

The generation process of a diffusion model, which maps noise to data samples, can be formulated through a reverse-time SDE or its associated PF-ODE. This procedure is inherently slow, since it relies on numerical solvers that approximate solution trajectories with many small integration steps (see Chapter A for a brief introduction). Accelerating inference has therefore become a central research objective. Broadly, existing approaches fall into two categories:

- **Training-Free Approaches:** The focus of this chapter. These methods develop advanced numerical solvers to improve the efficiency of diffusion sampling without additional training.
- **Training-Based Approaches:** Covered in Chapters 10 and 11. These techniques either distill a pre-trained diffusion model into a fast generator, or directly learn the ODE flow map (solution) so that only a few sampling steps are required.

SDE-based samplers (e.g., Euler–Maruyama) may yield more diverse samples due to stochasticity but typically require more steps (Xu *et al.*, 2023). Here we focus on ODE-based generation, whose principles extend naturally to the SDE setting.

After this chapter, you will be able to...

- view diffusion sampling as a numerical integration problem for the PF-ODE;
- distinguish model approximation error from numerical discretization error;
- understand the main numerical ideas behind DDIM, DEIS, DPM-Solver, and DPM-Solver++;
- connect modern diffusion samplers to classical numerical methods, includ-

ing Euler, Runge-Kutta (RK), Adams–Bashforth (AB), and exponential integrators.

Reader route. Readers unfamiliar with differential equations or numerical solvers may first refer to Chapter A. Throughout this chapter, keep one question in mind: what does each solver integrate exactly, and what does it approximate numerically?

9.1 Prologue

9.1.1 Fast Sampling as Numerical Integration

The Score SDE framework (Song *et al.*, 2020c) established a key foundation by linking discrete-time diffusion models and ELBO-based formulations (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020) with continuous-time SDE/ODE generative modeling. This connection gives a clean way to think about sampling: after a diffusion model has been trained, generation can be viewed as solving a learned differential equation from noise back to data.

The practical difficulty is speed. A naive sampler follows this reverse-time trajectory with many small numerical steps, often requiring hundreds or thousands of neural network evaluations. Fast sampling asks whether we can approximate the same generation process with far fewer model calls.

This chapter focuses on *training-free* acceleration: the pre-trained diffusion model is kept fixed, and we design better numerical solvers for sampling. Training-based acceleration methods, such as distillation or direct few-step training, are discussed later in Chapters 10 and 11.

The main message of this chapter is therefore simple:

Message

Once a diffusion model has been trained, training-free fast sampling is largely a numerical integration problem.

The literature contains many named diffusion samplers, which can make the subject appear more fragmented than it really is. In this chapter, we first introduce the small number of numerical ideas underlying these methods. Only afterward do we attach the familiar method names to them. This organization makes clear that the samplers are not unrelated tricks: they are different ways of approximating the same PF-ODE trajectory.

Sampling as Solving the Empirical PF-ODE. Suppose we have a pre-trained diffusion model

$$\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x}),$$

or equivalently a model written in one of the other standard parameterizations from Section 6.3. The empirical PF-ODE uses this learned model in place of the true score:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t) - \frac{1}{2} g^2(t)\, \mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t).$$

Starting from $\mathbf{x}(T) \sim p_{\text{prior}}$, generation integrates this ODE backward from $t = T$ to $t = 0$.

For two times $s > t$, the exact empirical update can be written in integral form as

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \left[\mathbf{f}(\mathbf{x}(\tau),\tau) - \frac{1}{2}g^2(\tau)\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}(\tau),\tau)\right] \mathrm{d}\tau. \tag{9.1.1}$$

Here, $\widetilde{\mathbf{\Psi}}_{s\to t}$ denotes the flow map of the empirical PF-ODE, obtained by replacing the oracle score with the learned model. As throughout this book, a *flow map* refers to the classical ODE solution map introduced in Equation (4.2.2). We denote the corresponding oracle flow map by $\mathbf{\Psi}_{s\to t}$. The empirical flow map is intended to approximate the oracle map:

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) \approx \mathbf{\Psi}_{s\to t}(\mathbf{x}_s).$$

Because the integral in Equation (9.1.1) cannot generally be evaluated in closed form, a numerical sampler discretizes the reverse-time interval using a decreasing grid

$$T = t_0 > t_1 > \cdots > t_M = 0, \tag{9.1.2}$$

starts from

$$\tilde{\mathbf{x}}_{t_0} = \mathbf{x}_T \sim p_{\text{prior}},$$

and produces a sequence

$$\tilde{\mathbf{x}}_{t_0}, \tilde{\mathbf{x}}_{t_1}, \ldots, \tilde{\mathbf{x}}_{t_M},$$

where $\tilde{\mathbf{x}}_{t_i}$ approximates $\widetilde{\mathbf{\Psi}}_{T\to t_i}(\mathbf{x}_T)$. The final iterate $\tilde{\mathbf{x}}_{t_M}$ is used as the generated sample.

Thus, every training-free solver in this chapter answers the same question:

Question 9.1.1

How should we approximate the PF-ODE integral accurately using as few model evaluations as possible?

Three Independent Choices in Solver Design. Once the decreasing time grid in Equation (9.1.2) has been introduced, it is useful to distinguish three choices that are often intertwined in diffusion sampling.

1. **Prediction Parameterization.** Which quantity is provided by the pre-trained diffusion model: score $\mathbf{s}_{\boldsymbol{\phi}^\times}$, noise $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}$, clean data $\mathbf{x}_{\boldsymbol{\phi}^\times}$, or velocity $\mathbf{v}_{\boldsymbol{\phi}^\times}$?

2. **Time Parameterization.** Which variable is used to parameterize progress along the same continuous trajectory? The original diffusion time t is one choice; other reparameterizations will be introduced later when they become useful.

3. **Numerical Approximation.** How does the solver approximate the unknown model-dependent quantity over a finite interval? It may use only the current evaluation, obtain additional evaluations within the current step, or reuse evaluations from previous steps.

This separation will be important throughout the chapter. The first two choices determine the numerical form presented to the solver, while the third determines how the solver uses the available model evaluations.

How Can a Solver Use Model Evaluations? A useful way to organize numerical solvers is by asking where the information for the next update comes from. Starting from the current state, a solver may either obtain additional information *within the current step* or reuse information from *previous steps*.

A *single-step method* constructs the next state using only the current state and evaluations generated within the same time interval. Euler's method is the simplest example, using only the current velocity. Higher-order Runge-Kutta (RK) methods may instead evaluate the velocity at one or more intermediate states before completing the step. Thus, "single-step" does not mean "one function evaluation"; it means that no information from earlier completed steps is required.

A *multistep method* reuses information from previous completed steps. Stored model evaluations provide information about how the dynamics have recently changed, allowing higher-order updates without several new evaluations within the current step. The Adams–Bashforth (AB) family is a representative classical example.

A separate idea is to change how updates across time are scheduled. Rather than computing all time points strictly sequentially, one may initialize states at several times and repeatedly correct them until they become consistent with the ODE. Such fixed-point iterations can allow model evaluations at different time points to be carried out in parallel.

Representative classical examples, including Euler, RK, AB, and Picard iteration, are reviewed in Section A.1.5; their structural organization is summarized in Equation (A.1.6).

Computational Cost and Practical Performance. A numerical time step need not correspond to a single neural-network evaluation. Euler uses one evaluation per step, whereas higher-order RK methods may require several. Multistep methods instead reuse stored evaluations and, after initialization, can often advance using only one new evaluation per step.

We therefore measure sampling cost by the *number of function evaluations* (NFE). If a method uses approximately m model evaluations per step over M

steps, then

$$\mathrm{NFE} \approx mM.$$

For instance, classifier-free guidance (see Section 8.3) requires both conditional and unconditional model predictions and therefore roughly doubles the neural-network evaluation cost per sampling step.

This classical taxonomy tells us *where numerical information comes from* and how many model evaluations it costs. Diffusion PF-ODEs offer an additional opportunity: not every part of the vector field is equally unknown. Depending on the prediction parameterization, part of the dynamics is determined analytically by the noise schedule, while only the remaining quantity requires a neural-network evaluation. We next examine how exposing this separation changes the numerical form of the PF-ODE and motivates diffusion-specific integrators.

9.1.2 Design of Diffusion-Specific Solvers

The classical numerical methods reviewed in Section A.1.4 treat a general ODE through evaluations of its vector field. Diffusion PF-ODEs offer additional structure: the noise schedule is known analytically, whereas the diffusion-model prediction is supplied by a neural network.

This creates a useful opportunity. Rather than numerically approximating every part of the dynamics in the same way, a solver may retain the known schedule-dependent terms analytically and approximate only the model-dependent prediction. How clearly this structure is exposed depends on the prediction parameterization.

I. Parameterizations Beyond the Score. Although the PF-ODE can be written in score form, modern implementations often use noise prediction, clean-data prediction, or velocity prediction. These parameterizations are algebraically related, but they expose different numerical structures.

There is also an important stability reason to avoid direct score prediction in many solvers. As $t \to 0$, the true score $\nabla_{\mathbf{x}} \log p_t(\cdot)$ can change very rapidly, especially when the data distribution is concentrated near a low-dimensional manifold (Kim *et al.*, 2022). This makes it difficult for a neural network trained to approximate the score directly to remain accurate near small noise levels [1].

[1] To see the scaling, recall the oracle relation from Equation (6.3.1):

$$\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = -\sigma_t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t),$$

where $\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t]$ is the oracle noise predictor, and the perturbation kernel is $\mathbf{x}_t|\mathbf{x}_0 \sim \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$. By the orthogonality property of conditional expectation in L^2,

$$\mathbb{E}\|\boldsymbol{\epsilon}\|_2^2 = \mathbb{E}\|\boldsymbol{\epsilon}^*\|_2^2 + \mathbb{E}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}^*\|_2^2,$$

For this reason, a widely used alternative is to predict the noise $\boldsymbol{\epsilon}_{\phi^\times}$, or related targets such as clean-data prediction or velocity prediction. In the noise-prediction form, the score approximation is

$$\mathbf{s}_{\phi^\times}(\mathbf{x}, t) = -\frac{1}{\sigma_t}\,\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}, t).$$

These prediction parameterizations are algebraically related, but the quantity supplied by the neural network is only one component of the solver design. We next examine how the same PF-ODE appears numerically under these different parameterizations.

II. One PF-ODE, Two Numerical Forms. Once the prediction parameterization has been chosen, the same PF-ODE can be presented to a numerical solver in two particularly useful forms.

Direct Velocity Form. The most direct representation is

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_{\phi^\times}(\mathbf{x}(t), t). \tag{9.1.3}$$

Here, under $\mathbf{v}$-prediction, the model directly supplies the complete empirical PF-ODE velocity $\mathbf{v}_{\phi^\times}(\mathbf{x}(t), t)$. Numerical integration methods therefore discretize the full right-hand side as a whole.

Factored Semilinear Form. Score, noise, and clean-data prediction naturally expose additional schedule-dependent structure. In each case, the PF-ODE can be written in the generic *semilinear* form

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underbrace{a(t)\mathbf{x}(t)}_{\substack{\text{known}\\ \text{linear part}}} + \underbrace{b(t)}_{\substack{\text{known}\\ \text{coefficient}}}\ \underbrace{\mathbf{N}_{\phi^\times}(\mathbf{x}(t), t)}_{\substack{\text{neural}\\ \text{evaluation}}}, \quad \mathbf{N}_{\phi^\times} \in \left\{\mathbf{s}_{\phi^\times}, \boldsymbol{\epsilon}_{\phi^\times}, \mathbf{x}_{\phi^\times}\right\}. \tag{9.1.4}$$

---

and therefore

$$\mathbb{E}\|\boldsymbol{\epsilon}^*\|_2^2 \le \mathbb{E}\|\boldsymbol{\epsilon}\|_2^2 = D,$$

where D is the data dimension. Thus, the oracle noise predictor has uniformly bounded expected squared norm. In contrast, the oracle score

$$\mathbf{s}^*(\mathbf{x}_t, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$$

satisfies

$$\mathbb{E}\|\mathbf{s}^*(\mathbf{x}_t, t)\|_2^2 = \sigma_t^{-2}\,\mathbb{E}\|\boldsymbol{\epsilon}^*(\mathbf{x}_t, t)\|_2^2 \le \frac{D}{\sigma_t^2}.$$

Thus, direct score prediction can become increasingly poorly scaled as σ_t becomes small, since noise prediction is converted to the score through the factor $1/\sigma_t$. This scaling can make direct score parameterization numerically less favorable near small noise levels.

Here, $a(t)$ and $b(t)$ are known functions determined by the noise schedule, whereas $\mathbf{N}_{\boldsymbol{\phi}^\times}$ denotes the neural prediction[2]. The term *semilinear* refers to this decomposition: the first term is linear in the state $\mathbf{x}(t)$, whereas the neural term may depend nonlinearly on $\mathbf{x}(t)$. This separation is not unique, but it is particularly useful numerically because the known linear part can often be handled analytically, leaving only the neural contribution to be approximated numerically.

For the three standard semilinear parameterizations[3], the corresponding choices of $a(t)$, $b(t)$, and $\mathbf{N}_{\boldsymbol{\phi}^\times}$ are listed in Table 9.1. For example, under noise prediction,

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underbrace{f(t)\mathbf{x}(t)}_{\text{linear part}} + \underbrace{\frac{1}{2}\frac{g^2(t)}{\sigma_t}\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t)}_{\text{nonlinear neural part}}. \tag{9.1.5}$$

The coefficient multiplying the neural prediction is known entirely from the noise schedule; the only learned quantity in the second term is $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t)$.

Table 9.1: Factored PF-ODE forms for score, noise, and clean-data prediction.

Prediction	$a(t)$	$b(t)$	$\mathbf{N}_{\boldsymbol{\phi}^\times}$
$\mathbf{s}$-prediction	$f(t)$	$-\frac{1}{2}g^2(t)$	$\mathbf{s}_{\boldsymbol{\phi}^\times}$
$\boldsymbol{\epsilon}$-prediction	$f(t)$	$\frac{1}{2}\frac{g^2(t)}{\sigma_t}$	$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}$
$\mathbf{x}$-prediction	$\frac{\sigma_t'}{\sigma_t}$	$\alpha_t\left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t}\right)$	$\mathbf{x}_{\boldsymbol{\phi}^\times}$

At the oracle level, score, noise, clean-data, and velocity prediction are exactly equivalent representations of the same PF-ODE. For an empirical model, the same equivalence holds when the different prediction parameterizations are obtained through exact algebraic conversion of the same model output (see Equation (6.3.1)). In this case, the same empirical PF-ODE can be expressed through score, noise, clean-data, or velocity prediction. The resulting formulas look different because they expose different numerical structures, but they describe the same learned continuous dynamics. However, if separate networks are trained independently for

[2] In standard semilinear notation, $b(t)\mathbf{N}_{\boldsymbol{\phi}^\times}$ could be absorbed into a single nonlinear term. We keep the two factors separate because $b(t)$ is known analytically from the noise schedule, whereas $\mathbf{N}_{\boldsymbol{\phi}^\times}$ requires a neural-network evaluation. This allows a solver to approximate the neural prediction inside the integral while retaining and analytically integrating the known schedule-dependent coefficient.

[3] In the following sections, we switch between the SDE coefficients $(f(t), g(t))$ and the Gaussian perturbation kernel $\mathbf{x}_t|\mathbf{x}_0 \sim \mathcal{N}\big(\mathbf{x}_t; \alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I}\big)$. They are related by

$$f(t) = \frac{\alpha_t'}{\alpha_t}, \qquad g^2(t) = \frac{\mathrm{d}}{\mathrm{d}t}\big(\sigma_t^2\big) - 2\frac{\alpha_t'}{\alpha_t}\sigma_t^2 = 2\sigma_t\sigma_t' - 2\frac{\alpha_t'}{\alpha_t}\sigma_t^2.$$

different prediction targets, their empirical PF-ODEs need not coincide exactly. In that case, differences between samplers may reflect both model approximation and numerical discretization.

III. Exponential Integrators for Semilinear PF-ODEs. The factored semilinear form in Equation (9.1.4) separates the analytically known linear dynamics from the model-dependent prediction. For $s > t$, define the integrating factor

$$\mathcal{E}(s \to t) := \exp\left(\int_s^t a(u)\,\mathrm{d}u\right).$$

The variation-of-constants formula then gives the exact empirical PF-ODE flow

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) = \mathcal{E}(s \to t)\mathbf{x}_s + \int_s^t \mathcal{E}(\tau \to t) b(\tau) \mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau. \tag{9.1.6}$$

We refer the reader to Section A.1.3 for the general derivation.

For example, under noise prediction, $a(t) = f(t)$, $b(t) = \frac{1}{2}g^2(t)/\sigma_t$, and $\mathbf{N}_{\boldsymbol{\phi}^\times} = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}$, so

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) = \underbrace{\mathcal{E}(s \to t)\mathbf{x}_s}_{\text{exact linear evolution}} + \frac{1}{2}\int_s^t \frac{g^2(\tau)}{\sigma_\tau}\mathcal{E}(\tau \to t)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau, \tag{9.1.7}$$

where $\mathcal{E}(s \to t) = \exp\left(\int_s^t f(u)\,\mathrm{d}u\right)$.

The numerical opportunity is now clear. The integrating factor and the coefficient $b(\tau)$ are determined by the diffusion schedule; the only unknown quantity inside the integral is the neural prediction evaluated along the trajectory. Keeping these known factors explicit therefore allows a first-order approximation to act only on the neural prediction, instead of absorbing the coefficient-weighted product $b(\tau)\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ into a single unknown residual.

Why the Factored Exponential-Integrator Form Can Help. To see the benefit of this separation, compare it with plain Euler over one backward step from an exact empirical PF-ODE state $\mathbf{x}_s$ at time s to $t < s$. Define $\Delta s := s - t > 0$ and $\mathbf{N}_s := \mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s)$.

Plain Euler treats the complete PF-ODE velocity as locally constant, $a(\tau)\mathbf{x}_\tau + b(\tau)\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx a(s)\mathbf{x}_s + b(s)\mathbf{N}_s$ for $\tau \in [t, s]$, giving

$$\tilde{\mathbf{x}}_t^{\text{Euler}} = \mathbf{x}_s - \Delta s\,[a(s)\mathbf{x}_s + b(s)\mathbf{N}_s]\,. \tag{9.1.8}$$

Instead, we may hold only the neural prediction fixed, $\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{N}_s$, while retaining the known schedule dependence in Equation (9.1.6). This gives the

first-order factored exponential-integrator update[4]

$$\tilde{\mathbf{x}}_t^{\mathrm{EI}} = \mathcal{E}(s \to t)\mathbf{x}_s + \left[\int_s^t \mathcal{E}(\tau \to t) b(\tau)\, \mathrm{d}\tau\right] \mathbf{N}_s. \tag{9.1.9}$$

Both methods use one neural-network evaluation per step. Plain Euler freezes the complete PF-ODE velocity, while the factored update keeps the known schedule dependence and freezes only the neural prediction. This factorization matters: if $b(\tau)\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ is instead treated as a single residual and held fixed, the result is conventional exponential Euler. Thus, the factored update retains one more known quantity, $b(\tau)$, inside the integral; the two updates coincide when $b(\tau)$ is constant over the step.

A simple frozen-prediction example makes this difference concrete. Suppose $a(t) \equiv a$, $b(t) \equiv b$, and $\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \equiv \mathbf{N}_s$ over the step. For $a \neq 0$,

$$\begin{aligned}
\tilde{\mathbf{x}}_t^{\mathrm{EI}} &= e^{-a\Delta s}\mathbf{x}_s + \frac{b}{a}\left(e^{-a\Delta s} - 1\right)\mathbf{N}_s, \\
\tilde{\mathbf{x}}_t^{\mathrm{Euler}} &= (1 - a\Delta s)\mathbf{x}_s - b\Delta s\, \mathbf{N}_s.
\end{aligned}$$

Expanding the factored update gives

$$\tilde{\mathbf{x}}_t^{\mathrm{EI}} = \tilde{\mathbf{x}}_t^{\mathrm{Euler}} + \frac{(\Delta s)^2}{2}\left[a^2\mathbf{x}_s + ab\,\mathbf{N}_s\right] + \mathcal{O}((\Delta s)^3).$$

Thus, the two updates agree to first order for small steps, while the factored update retains the higher-order effects of the known dynamics. The difference can become important for larger steps. For example, when $a > 0$, the exact multiplier $e^{-a\Delta s}$ remains positive and contractive for every $\Delta s > 0$, whereas the Euler approximation $1 - a\Delta s$ becomes negative when $a\Delta s > 1$ and has magnitude greater than one when $a\Delta s > 2$. Plain Euler can therefore distort dynamics that the factored form already handles analytically.

In the full PF-ODE, the neural prediction generally varies along the trajectory. Subtracting Equation (9.1.9) from the exact variation-of-constants formula gives

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{EI}} = \int_s^t \mathcal{E}(\tau \to t) b(\tau)\left[\mathbf{N}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) - \mathbf{N}_s\right] \mathrm{d}\tau. \tag{9.1.10}$$

Hence, once the known schedule-dependent dynamics are handled analytically, the remaining discretization error is governed by the variation of the neural prediction along the trajectory. This separation is the basic numerical principle that several diffusion-specific solvers will exploit in different ways.

[4] The update in Equation (9.1.9) is not plain Euler in the original state and time variables. After an integrating-factor transformation of the state and, on intervals where the resulting time coordinate is monotone, a corresponding time reparameterization, it can be written as an ordinary explicit Euler step. We make this connection concrete for diffusion sampling in the DDIM section.

9.1.3 Roadmap from Numerical Ideas to Diffusion Samplers

The remainder of this chapter shows how these numerical ideas appear in modern diffusion samplers. We begin with DDIM in Section 9.2, which constructs a first-order update from the current model prediction. We then study DEIS in Section 9.3, which uses predictions from previous steps to build higher-order approximations. DPM-Solver in Section 9.4 develops higher-order updates using additional model evaluations within the current step. DPM-Solver++ in Section 9.5 develops two corresponding variants based on clean-data prediction: a single-step variant, DPM-Solver++(2S), which uses additional evaluations within the current step, and a multistep variant, DPM-Solver++(2M), which reuses predictions from previous steps. Finally, Section 9.6 introduces ParaDiGMS, which updates several time points together through fixed-point iteration to enable parallel computation across time.

The table below provides a roadmap connecting these diffusion samplers with the classical numerical ideas reviewed in Equation (A.1.6).

Classical Idea	How the Update Is Constructed	Diffusion Examples	Where
Euler-type first-order update	Use the model prediction at the beginning of the step to advance to the next state.	DDIM; DPM-Solver-1	Sections 9.2 and 9.4
RK-type staged update	Evaluate the model at additional intermediate states before completing the current step.	Higher-order DPM-Solver; DPM-Solver++(2S)	Sections 9.4 and 9.5
AB-type multistep update	Reuse model predictions from previous steps to construct the next update.	DEIS; DPM-Solver++(2M)	Sections 9.3 and 9.5
Picard-type fixed-point iteration	Update several time points together and repeatedly refine their states.	ParaDiGMS	Section 9.6

These connections provide a high-level numerical roadmap for the methods developed below. Diffusion solvers further benefit from structure specific to the PF-ODE, including different prediction parameterizations, analytically known schedule-dependent coefficients, and alternative time parameterizations.

To keep the presentation systematic, each section follows the same broad structure: we first derive the diffusion solver and explain the numerical idea it exploits, then connect it to the corresponding classical solver principle, and finally relate it to other diffusion solvers developed in this chapter. This organization makes

clear both the common numerical backbone and the diffusion-specific ingredients that distinguish the resulting methods.

9.2 DDIM

Denoising Diffusion Implicit Models (DDIM) (Song *et al.*, 2020a) is one of the earliest and most widely used methods for accelerating diffusion sampling. Although DDIM was originally introduced from a probabilistic perspective, its deterministic update also admits a particularly clean interpretation through the PF-ODE. We therefore introduce DDIM here under the common numerical ODE-solving umbrella of this chapter, and defer its original variational perspective to Section 9.2.3.

The Prologue in Section 9.1 leaves us with a concrete numerical question: once the known schedule-dependent factors have been retained analytically, how should we approximate the neural prediction inside the remaining integral? DDIM takes the simplest possible answer: *use its value at the beginning of the step*. We first derive the resulting update, and then clarify its relation to classical Euler integration and to the higher-order diffusion solvers introduced later.

9.2.1 DDIM as a First-Order Diffusion Solver

Consider one sampling step from starting point[5] $\tilde{\mathbf{x}}_s$ at time s to $t < s$. Under noise prediction, Equation (9.1.7) gives the exact empirical PF-ODE flow

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \mathcal{E}(s \to t)\tilde{\mathbf{x}}_s + \frac{1}{2}\int_s^t \frac{g^2(\tau)}{\sigma_\tau}\mathcal{E}(\tau \to t)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau.$$

The schedule-dependent factors are known analytically; the only unknown quantity inside the integral is the neural prediction $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ evaluated along the unknown trajectory.

From the PF-ODE viewpoint, deterministic DDIM uses the simplest first-order approximation: it holds the noise prediction fixed at its value at the beginning of the step,

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s), \quad \text{for} \quad \tau \in [t, s],$$

while retaining the known schedule-dependent factors inside the integral. This gives

$$\tilde{\mathbf{x}}_t = \mathcal{E}(s \to t)\tilde{\mathbf{x}}_s + \left[\frac{1}{2}\int_s^t \frac{g^2(\tau)}{\sigma_\tau}\mathcal{E}(\tau \to t)\,\mathrm{d}\tau\right]\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Using $f(t) = \alpha_t'/\alpha_t$, the integrating factor simplifies to $\mathcal{E}(s \to t) = \alpha_t/\alpha_s$. The remaining schedule-dependent coefficient also has a convenient closed form: $\frac{1}{2}\frac{g^2(\tau)}{\sigma_\tau}\mathcal{E}(\tau \to t) = \alpha_t \frac{\mathrm{d}}{\mathrm{d}\tau}\left(\frac{\sigma_\tau}{\alpha_\tau}\right)$. Therefore, once the neural prediction is held fixed over the step, the remaining integral can be evaluated exactly from the noise schedule.

[5] Throughout this chapter, $\tilde{\mathbf{x}}_s$ denotes the state at time level s used to initialize the current solver step, typically obtained from a previous numerical step. The tilde distinguishes it from $\mathbf{x}_s$, which denotes an oracle sample from the exact diffusion distribution, either $\mathbf{x}_s \sim p_s$ or, conditioned on $\mathbf{x}_0$, $\mathbf{x}_s \sim p_s(\cdot \mid \mathbf{x}_0)$. Hence, $\tilde{\mathbf{x}}_s$ need not be exactly distributed according to p_s.

Proposition 9.2.1: DDIM as a First-Order Factored EI Update

For $s > t$, deterministic DDIM gives

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \alpha_t\left(\frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s}\right)\boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s). \tag{9.2.1}$$

Proof for Proposition.

The result follows by integrating $\frac{\mathrm{d}}{\mathrm{d}\tau}(\sigma_\tau/\alpha_\tau)$ from s to t.

The approximation made by DDIM is therefore very specific: the known schedule evolution is retained over the full interval, while only the change of the neural prediction is neglected.

There is an analogous derivation under clean-data prediction. The exact empirical PF-ODE flow can be written as

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \frac{\sigma_t}{\sigma_s}\tilde{\mathbf{x}}_s + \sigma_t\int_s^t \frac{\mathrm{d}}{\mathrm{d}\tau}\left(\frac{\alpha_\tau}{\sigma_\tau}\right)\mathbf{x}_{\phi^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau.$$

Using the same first-order approximation as above, we hold $\mathbf{x}_{\phi^\times}(\mathbf{x}_\tau, \tau)$ fixed at its value at time s. The remaining schedule-dependent integral is then evaluated exactly, giving

$$\tilde{\mathbf{x}}_t = \frac{\sigma_t}{\sigma_s}\tilde{\mathbf{x}}_s + \alpha_s\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)\mathbf{x}_{\phi^\times}(\tilde{\mathbf{x}}_s, s).$$

Thus, freezing either the noise prediction in the noise-prediction PF-ODE or the clean-data prediction in the clean-data PF-ODE produces the same DDIM update. This correspondence will reappear later in DPM-Solver and DPM-Solver++, which build higher-order approximations from the noise- and clean-data-prediction formulations, respectively.

Equivalent Forms and Interpretation. At the oracle level, the standard prediction targets are pointwise algebraically equivalent through Equation (6.3.1). Therefore, the deterministic DDIM update derived above from noise prediction can be rewritten at the starting point s in terms of clean-data or score prediction without changing the finite-step update. The following corollary collects these equivalent expressions.

Corollary 9.2.1: Equivalent Forms of the Deterministic DDIM Update

Let $s > t$, and consider a state $\tilde{\mathbf{x}}_s$ at time s. The oracle deterministic DDIM update admits the equivalent expressions

$$\begin{aligned}\tilde{\mathbf{x}}_t &= \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \alpha_t\left(\frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s}\right)\boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s) \\ &= \frac{\sigma_t}{\sigma_s}\tilde{\mathbf{x}}_s + \alpha_s\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)\mathbf{x}^*(\tilde{\mathbf{x}}_s, s) \\ &= \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \sigma_s^2\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)\mathbf{s}^*(\tilde{\mathbf{x}}_s, s) \\ &= \alpha_t \underbrace{\mathbf{x}^*(\tilde{\mathbf{x}}_s, s)}_{\approx \mathbf{x}_{\phi^\times}} + \sigma_t \underbrace{\boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s)}_{\approx \boldsymbol{\epsilon}_{\phi^\times}}.\end{aligned} \tag{9.2.2}$$

estimated clean; estimated noise

The last identity in Equation (9.2.2) gives a particularly intuitive view of DDIM. Starting from $\tilde{\mathbf{x}}_s$, DDIM keeps the clean and noise estimates inferred at time s fixed and recombines them using the target schedule coefficients:

$$\tilde{\mathbf{x}}_t = \alpha_t \mathbf{x}^*(\tilde{\mathbf{x}}_s, s) + \sigma_t \boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s).$$

This mirrors the shared-noise coupling of the forward Gaussian path, $\mathbf{x}_\tau = \alpha_\tau \mathbf{x}_0 + \sigma_\tau \boldsymbol{\epsilon}$: conditioned on a fixed pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$, moving from one time to another changes only the coefficients $(\alpha_\tau, \sigma_\tau)$. DDIM applies the same idea using the clean and noise estimates available at the beginning of the step.

At the conditional level, where $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$ are fixed, this recombination follows the corresponding shared-noise path exactly. At the marginal level, however, $\mathbf{x}^*(\tilde{\mathbf{x}}_s, s)$ and $\boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s)$ are posterior estimates inferred from $\tilde{\mathbf{x}}_s$. Consequently, a finite DDIM step is generally an approximation to the marginal PF-ODE flow, rather than an exact draw from p_t.

Why Does the Prediction Held Fixed Matter? At any fixed state and time, the standard prediction parameterizations are algebraically equivalent through Equation (6.3.1). A finite-step solver, however, introduces an additional approximation by holding one chosen prediction fixed over an interval. Such a freezing approximation is preserved under a change of parameterization only when the converted prediction also remains constant along the resulting approximate trajectory.

DDIM provides a particularly clean example. Let

$$\widehat{\mathbf{x}}_0 := \mathbf{x}_{\phi^\times}(\tilde{\mathbf{x}}_s, s), \qquad \widehat{\boldsymbol{\epsilon}} := \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s),$$

where, if only one quantity is directly predicted by the model, the other is defined by the algebraic conversion in Equation (6.3.1). Hence, by construction, $\tilde{\mathbf{x}}_s = \alpha_s \widehat{\mathbf{x}}_0 + \sigma_s \widehat{\boldsymbol{\epsilon}}$.

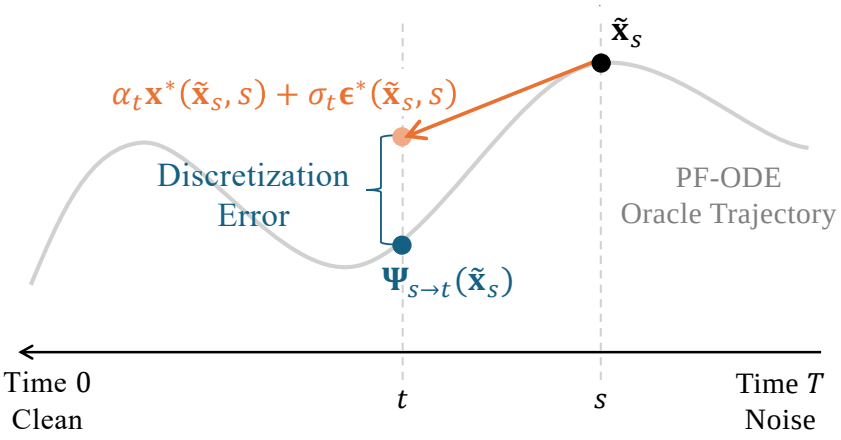


Figure 9.1: Finite-step DDIM versus the empirical PF-ODE flow. Starting from $\tilde{\mathbf{x}}_s$, the empirical PF-ODE evolves continuously to $\widetilde{\boldsymbol{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s)$, whereas DDIM retains the start-time neural prediction over the whole interval while evolving the known schedule analytically. Their discrepancy is the one-step discretization error in Equation (9.2.3).

Source: Created by the authors.

Holding this start-time clean/noise pair fixed defines the DDIM interpolation over the entire step:

$$\tilde{\mathbf{x}}_\tau^{\mathrm{DDIM}} = \alpha_\tau \widehat{\mathbf{x}}_0 + \sigma_\tau \widehat{\boldsymbol{\epsilon}}, \qquad \tau \in [t, s].$$

This requires no additional model evaluations: once $(\widehat{\mathbf{x}}_0, \widehat{\boldsymbol{\epsilon}})$ is fixed at time s, the schedule coefficients $(\alpha_\tau, \sigma_\tau)$ are known for every intermediate τ. At $\tau = s$ the expression recovers the starting state, while at $\tau = t$ it gives the DDIM update. The intermediate values simply describe the finite-step trajectory induced by the DDIM approximation.

The special role of noise and clean-data prediction is now immediate. Along this trajectory,

$$\frac{\tilde{\mathbf{x}}_\tau^{\mathrm{DDIM}} - \sigma_\tau \widehat{\boldsymbol{\epsilon}}}{\alpha_\tau} = \widehat{\mathbf{x}}_0, \qquad \frac{\tilde{\mathbf{x}}_\tau^{\mathrm{DDIM}} - \alpha_\tau \widehat{\mathbf{x}}_0}{\sigma_\tau} = \widehat{\boldsymbol{\epsilon}}.$$

Thus, the algebraically converted clean and noise predictions remain constant together along the DDIM trajectory. Consequently, holding $\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s)$ or holding $\mathbf{x}_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{x}_{\phi^\times}(\tilde{\mathbf{x}}_s, s)$ leads to the same DDIM update. In this sense, $\widehat{\mathbf{x}}_0$ and $\widehat{\boldsymbol{\epsilon}}$ are the two fixed components of the DDIM shared-noise path, while $(\alpha_\tau, \sigma_\tau)$ provide the known time-dependent weights.

However, score and velocity behave differently. DDIM keeps the inferred clean/noise pair $(\widehat{\mathbf{x}}_0, \widehat{\boldsymbol{\epsilon}})$ fixed, but the score and velocity obtained from this pair generally change with time.

For example, converting the fixed (at time s) noise estimate to score prediction at an intermediate time τ gives

$$\mathbf{s}(\tilde{\mathbf{x}}_\tau^{\mathrm{DDIM}}, \tau) = -\frac{\widehat{\boldsymbol{\epsilon}}}{\sigma_\tau} = \frac{\sigma_s}{\sigma_\tau}\mathbf{s}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Thus, even though $\widehat{\boldsymbol{\epsilon}}$ is fixed, its corresponding score generally varies with τ through the factor $1/\sigma_\tau$. Directly imposing

$$\mathbf{s}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{s}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s), \qquad \tau \in [t, s],$$

therefore makes a different finite-step approximation and generally yields a different first-order factored solver. Similarly, the fixed clean/noise pair induces the velocity

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\tilde{\mathbf{x}}_\tau^{\mathrm{DDIM}} = \alpha'_\tau \widehat{\mathbf{x}}_0 + \sigma'_\tau \widehat{\boldsymbol{\epsilon}}.$$

This velocity generally changes with τ because the schedule derivatives α'_τ and σ'_τ change. Holding the raw velocity prediction fixed instead, $\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{v}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)$, gives plain Euler in the original diffusion time; its relation to DDIM is discussed in Section 9.2.2.

Remark.

> Prediction parameterizations are pointwise algebraically equivalent, but finite-step freezing need not commute with a time-dependent conversion. DDIM keeps a consistent start-time clean/noise pair fixed; the corresponding score and complete velocity generally vary over the step. Hence directly freezing the raw score or velocity defines a different first-order method. The velocity case is quantified in Section 9.2.2.

9.2.2 Relation to Euler-Type Methods

The discussion above shows that DDIM is a first-order method, but its relation to the classical Euler method requires some care. Once the known diffusion schedule is separated from the learned prediction, different first-order schemes arise depending on *what is held fixed over a finite step.*

We compare three one-evaluation constructions:

1. *plain Euler*, which holds the complete PF-ODE velocity fixed;
2. *exponential Euler*, which treats the known linear propagation analytically but holds the remaining coefficient-weighted residual fixed;
3. *DDIM*, which keeps the start-time clean/noise estimates fixed while retaining their finite-step evolution under the known diffusion schedule.

These distinctions are easiest to see from two equivalent views of the same DDIM update. The clean/noise form makes its difference from plain Euler transparent, whereas the factored semilinear form clarifies its relation to exponential Euler. We then identify the remaining discretization error and show the precise sense in which DDIM itself can be viewed as an Euler method after a schedule-aligned change of coordinates.

DDIM versus Plain Euler. DDIM and plain Euler use the same start-time model information, but propagate it differently over a finite step. Plain Euler locally approximates the complete PF-ODE velocity, whereas DDIM retains the finite-step evolution of the known diffusion schedule while holding the model prediction fixed. Consequently, for smooth schedules, the two updates agree to first order and generally differ by terms of order $\mathcal{O}((\Delta s)^2)$. We make this difference explicit below.

Consider one backward step from time s to $t < s$, with $\Delta s := s - t > 0$, and suppose the start-time clean and noise predictions are algebraically consistent:

$$\tilde{\mathbf{x}}_s = \alpha_s \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \sigma_s \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

The corresponding empirical PF-ODE velocity at time s is

$$\mathbf{v}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) = \alpha'_s \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \sigma'_s \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Plain Euler holds this complete velocity fixed over the step, $\tau \in [t, s]$, and gives:

$$\tilde{\mathbf{x}}_t^{\text{Euler}} = \tilde{\mathbf{x}}_s - \Delta s\, \mathbf{v}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

By contrast, DDIM keeps the clean/noise-prediction pair at the starting time fixed, while evolving the known schedule coefficients from (α_s, σ_s) to (α_t, σ_t):

$$\tilde{\mathbf{x}}_t^{\text{DDIM}} = \tilde{\mathbf{x}}_s + (\alpha_t - \alpha_s)\mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + (\sigma_t - \sigma_s)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Their exact finite-step difference is therefore

$$\tilde{\mathbf{x}}_t^{\text{DDIM}} - \tilde{\mathbf{x}}_t^{\text{Euler}} = \left[\alpha_t - \alpha_s + \Delta s\, \alpha'_s\right] \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \left[\sigma_t - \sigma_s + \Delta s\, \sigma'_s\right] \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

For smooth schedules,

$$\alpha_t - \alpha_s + \Delta s\, \alpha'_s = \tfrac{(\Delta s)^2}{2}\alpha''_s + \mathcal{O}((\Delta s)^3), \quad \sigma_t - \sigma_s + \Delta s\, \sigma'_s = \tfrac{(\Delta s)^2}{2}\sigma''_s + \mathcal{O}((\Delta s)^3).$$

Hence,

$$\tilde{\mathbf{x}}_t^{\text{DDIM}} - \tilde{\mathbf{x}}_t^{\text{Euler}} = \tfrac{(\Delta s)^2}{2}\left[\alpha''_s \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \sigma''_s \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)\right] + \mathcal{O}((\Delta s)^3).$$

Thus, DDIM and plain Euler agree to first order but generally differ over a finite step. Plain Euler replaces the finite schedule changes $(\alpha_t - \alpha_s, \sigma_t - \sigma_s)$ by their local linearizations, whereas DDIM retains the finite schedule evolution analytically.

A useful special case follows immediately. If both α_τ and σ_τ are *affine in* τ *over* $[t, s]$, then

$$\alpha_t - \alpha_s = -\Delta s\, \alpha'_s, \qquad \sigma_t - \sigma_s = -\Delta s\, \sigma'_s,$$

so DDIM and plain Euler coincide exactly. This includes, for example, the schedules (see also discussion in Section 6.3.4)

$$(\alpha_t, \sigma_t) = (1, t) \quad \text{and} \quad (\alpha_t, \sigma_t) = (1 - t, t).$$

Thus, the finite-step distinction between the two methods is driven by the nonlinearity of the known schedule in the original diffusion time.

DDIM versus Exponential Euler. The comparison above used the clean/noise representation to show how DDIM differs from plain Euler in the original diffusion time. We now return to the semilinear representation from the prologue to distinguish DDIM from conventional exponential Euler. The difference is simple: both methods retain the known linear evolution analytically, but they hold different quantities fixed inside the remaining integral.

Under noise prediction, the factored PF-ODE in Equation (9.1.4) has

$$a(\tau) = f(\tau), \qquad b(\tau) = \frac{g^2(\tau)}{2\sigma_\tau}, \qquad \mathbf{N}_{\boldsymbol{\phi}^\times} = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}.$$

Its exact variation-of-constants formula contains the model-dependent term

$$b(\tau)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau).$$

Conventional exponential Euler treats this coefficient-weighted quantity as a single residual and holds the whole residual fixed at the beginning of the step:

$$b(\tau)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx b(s)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s), \quad \tau \in [t, s]$$

It therefore gives

$$\tilde{\mathbf{x}}_t^{\text{ExpEuler}} = \mathcal{E}(s \to t)\tilde{\mathbf{x}}_s + b(s)\left[\int_s^t \mathcal{E}(\tau \to t)\, \mathrm{d}\tau\right] \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

DDIM uses the finer factorization emphasized in the prologue. Since $b(\tau)$ is known from the diffusion schedule, it remains inside the integral; only the neural prediction is held fixed:

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Hence,

$$\tilde{\mathbf{x}}_t^{\text{DDIM}} = \mathcal{E}(s \to t)\tilde{\mathbf{x}}_s + \left[\int_s^t \mathcal{E}(\tau \to t) b(\tau)\, \mathrm{d}\tau\right] \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

The finite-step difference between the two updates therefore comes entirely from the known coefficient $b(\tau)$:

$$\tilde{\mathbf{x}}_t^{\text{DDIM}} - \tilde{\mathbf{x}}_t^{\text{ExpEuler}} = \left[\int_s^t \mathcal{E}(\tau \to t)\big(b(\tau) - b(s)\big)\, \mathrm{d}\tau\right] \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

For smooth b,

$$\tilde{\mathbf{x}}_t^{\mathrm{DDIM}} - \tilde{\mathbf{x}}_t^{\mathrm{ExpEuler}} = \frac{(\Delta s)^2}{2} b'(s)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \mathcal{O}((\Delta s)^3).$$

Thus, DDIM and conventional exponential Euler agree to first order. The only difference is that exponential Euler freezes $b(\tau)$ at $b(s)$ together with the neural prediction, while DDIM keeps the known variation of $b(\tau)$. Therefore, the two updates coincide exactly whenever $b(\tau)$ is constant over the step, including the constant-b setting used in the Prologue example.

Together with the preceding plain-Euler comparison, the three first-order rules differ only in how much known structure is retained before making the constant approximation on $\tau \in [t, s]$:

$$\begin{aligned} \text{Plain Euler} &: \quad f(\tau)\mathbf{x}_\tau + b(\tau)\boldsymbol{\epsilon}_\tau \;\approx\; f(s)\tilde{\mathbf{x}}_s + b(s)\widehat{\boldsymbol{\epsilon}}, \\ \text{Exponential Euler} &: \quad b(\tau)\boldsymbol{\epsilon}_\tau \;\approx\; b(s)\widehat{\boldsymbol{\epsilon}}, \\ \text{DDIM} &: \quad \boldsymbol{\epsilon}_\tau \;\approx\; \widehat{\boldsymbol{\epsilon}}, \end{aligned}$$

where $\widehat{\boldsymbol{\epsilon}} := \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)$. All three use the same single neural-network evaluation at the beginning of the step. Plain Euler approximates the complete velocity; exponential Euler first retains the known linear propagation; DDIM additionally retains the known coefficient $b(\tau)$ and approximates only the neural prediction.

This hierarchy concerns which analytically known terms are preserved, rather than a universal ordering of one-step errors. Its main numerical consequence is that the remaining approximation in DDIM can be isolated entirely to the variation of the neural prediction, as we show next.

What Error Does Each Method Still Make? The comparisons above describe how the three numerical updates differ from one another. We now compare each update with the exact empirical PF-ODE flow. For this local-error analysis, suppose the step starts from an exact empirical state $\mathbf{x}_s$ at time s.

For plain Euler, the exact empirical flow can be written in the direct velocity form as

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau.$$

Since plain Euler holds the complete velocity fixed at its start-time value, its exact one-step defect is

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{Euler}} = \int_s^t \Big[\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) - \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s)\Big]\,\mathrm{d}\tau.$$

For exponential Euler, the known linear propagation is already treated analytically, but the coefficient-weighted residual $b(\tau)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ is held fixed. Hence

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{ExpEuler}} = \int_s^t \mathcal{E}(\tau \to t)\Big[b(\tau)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) - b(s)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s)\Big]\,\mathrm{d}\tau.$$

Finally, DDIM retains both the known linear propagation and the known coefficient $b(\tau)$, and holds only the neural prediction fixed:

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{DDIM}} = \int_s^t \mathcal{E}(\tau \to t) b(\tau) \Big[\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) - \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s) \Big] \, \mathrm{d}\tau. \tag{9.2.3}$$

These identities reveal the relevant notion of smoothness for each first-order scheme. If the corresponding quantity varies slowly along the exact empirical trajectory, then the one-step approximation is accurate. More precisely, with $\Delta s := s - t$, standard smoothness gives

$$\begin{aligned}
\left\| \widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{Euler}} \right\| &\le \frac{(\Delta s)^2}{2} \sup_{\tau\in[t,s]} \left\| \tfrac{\mathrm{d}}{\mathrm{d}\tau} \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \right\|, \\
\left\| \widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{ExpEuler}} \right\| &\le \frac{(\Delta s)^2}{2} \sup_{\tau\in[t,s]} |\mathcal{E}(\tau \to t)| \cdot \sup_{\tau\in[t,s]} \left\| \tfrac{\mathrm{d}}{\mathrm{d}\tau} \left[b(\tau) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \right] \right\|, \\
\left\| \widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) - \tilde{\mathbf{x}}_t^{\mathrm{DDIM}} \right\| &\le \frac{(\Delta s)^2}{2} \sup_{\tau\in[t,s]} |\mathcal{E}(\tau \to t) b(\tau)| \cdot \sup_{\tau\in[t,s]} \left\| \tfrac{\mathrm{d}}{\mathrm{d}\tau} \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \right\|.
\end{aligned}$$

Under the usual smoothness and stability assumptions, all three methods are *globally first order* and require only one neural-network evaluation per step. Moreover, each method is exact on a step when the quantity it holds fixed is in fact constant along the exact empirical trajectory: the complete PF-ODE velocity for plain Euler, the coefficient-weighted residual $b(\tau)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ for exponential Euler, and the noise prediction $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ for DDIM.

Accordingly, there is no universal error ordering among the three methods: each is most accurate when the quantity it holds fixed varies slowly over the step. DDIM is particularly well suited to regimes where the neural prediction changes slowly but the known diffusion schedule changes appreciably, since it retains the latter analytically. Exploiting more known structure can therefore reduce discretization error, but does not guarantee a smaller error on every individual step.

The errors discussed here are numerical discretization errors relative to the empirical PF-ODE; model mismatch between the empirical and oracle PF-ODEs is a separate source of error.

A Precise Euler Interpretation of DDIM. Although DDIM and plain Euler agree to first order, they generally give different finite-step updates in the original state $\mathbf{x}$ and diffusion time t. There is, however, an exact Euler interpretation after a schedule-aligned change of both the state and the time coordinate.

Under noise prediction, define

$$\mathbf{y}_t := \frac{\mathbf{x}_t}{\alpha_t}, \qquad \rho_t := \frac{\sigma_t}{\alpha_t}.$$

The two transformations play complementary roles. The state normalization $\mathbf{x}_t \mapsto \mathbf{x}_t/\alpha_t$ factors out the known signal scaling α_t, while the new clock $\rho_t = \sigma_t/\alpha_t$

measures progress by the noise-to-signal ratio rather than by the original diffusion time.

To see this first at the conditional level, consider a fixed pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$ in the Gaussian perturbation path

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Here, $\boldsymbol{\epsilon}$ is a fixed latent noise realization, not a neural prediction. The transformed state satisfies

$$\mathbf{y}_t = \frac{\mathbf{x}_t}{\alpha_t} = \mathbf{x}_0 + \rho_t \boldsymbol{\epsilon}.$$

Hence, along this particular conditional path,

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\rho} = \boldsymbol{\epsilon}.$$

Thus, for fixed $(\mathbf{x}_0, \boldsymbol{\epsilon})$, the known schedule dependence becomes affine in the transformed coordinate ρ.

The corresponding empirical PF-ODE has the same coordinate structure, but the fixed latent noise $\boldsymbol{\epsilon}$ is replaced by the learned, state-dependent noise prediction. Using Equation (9.1.5),

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\alpha_t'}{\alpha_t}\mathbf{x}_t + \left(\sigma_t' - \frac{\alpha_t'}{\alpha_t}\sigma_t\right) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t),$$

we obtain

$$\frac{\mathrm{d}\mathbf{y}_t}{\mathrm{d}t} = \frac{1}{\alpha_t}\left(\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} - \frac{\alpha_t'}{\alpha_t}\mathbf{x}_t\right) = \frac{\mathrm{d}\rho_t}{\mathrm{d}t}\, \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t). \tag{9.2.4}$$

Hence, on any interval where $\alpha_t > 0$ and ρ_t is strictly monotone, we may use ρ as the time coordinate. Writing $t = t(\rho)$ and defining the transformed neural field

$$\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\mathbf{y}, \rho) := \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}\big(\alpha_{t(\rho)}\mathbf{y}, t(\rho)\big),$$

the empirical PF-ODE becomes

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\rho} = \bar{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\mathbf{y}, \rho). \tag{9.2.5}$$

The spatial and temporal dependence of the model therefore remains present; it is simply expressed in the transformed variables $(\mathbf{y}, \rho)$.

Explicit Euler applied to Equation (9.2.5) from ρ_s to ρ_t gives

$$\tilde{\mathbf{y}}_t = \tilde{\mathbf{y}}_s + (\rho_t - \rho_s)\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{y}}_s, \rho_s).$$

Since

$$\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{y}}_s, \rho_s) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s),$$

substituting $\mathbf{y} = \mathbf{x}/\alpha$ and $\rho = \sigma/\alpha$ yields

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \alpha_t\left(\frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s}\right)\boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s),$$

which is exactly the deterministic DDIM update.

Coordinate Choice Is Part of the Numerical Method

An invertible change of state and time coordinates gives an equivalent representation of the same continuous ODE solution. A finite-step discretization, however, is generally not invariant to that representation: "change coordinates and then apply Euler" need not equal "apply Euler and then change coordinates".

The numerical significance is that a suitable coordinate system can absorb analytically known parts of the dynamics before discretization. In DDIM, $\mathbf{x}/\alpha$ removes the known signal scaling and $\rho = \sigma/\alpha$ absorbs the corresponding schedule-dependent time scaling, leaving the transformed neural field $\bar{\boldsymbol{\epsilon}}_{\phi^\times}(\mathbf{y}, \rho)$ as the quantity approximated by Euler. Thus DDIM generally differs from plain Euler in $(\mathbf{x}, t)$, while it is exactly explicit Euler in the schedule-aligned coordinates $(\mathbf{y}, \rho)$.

A dual construction holds under clean-data prediction. Normalizing the state by σ_t and using α_t/σ_t as the time coordinate gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathbf{x}_t}{\sigma_t}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\alpha_t}{\sigma_t}\right)\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t),$$

so the clean-data form of DDIM is likewise explicit Euler in its corresponding schedule-aligned coordinates.

This *state–time change-of-coordinates viewpoint* also previews the role of coordinate choice in higher-order diffusion solvers. DPM-Solver, developed later in Section 9.4, uses the half-log SNR

$$\lambda_t := \log\frac{\alpha_t}{\sigma_t},$$

which is directly related to DDIM's Euler clock:

$$\rho_t = \frac{\sigma_t}{\alpha_t} = e^{-\lambda_t}, \qquad \mathrm{d}\rho = -e^{-\lambda}\,\mathrm{d}\lambda.$$

Therefore, the exact transformed equation

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\rho} = \bar{\boldsymbol{\epsilon}}_{\phi^\times}(\mathbf{y}, \rho)$$

can equivalently be integrated as

$$\mathbf{y}_t = \mathbf{y}_s - \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\,\mathrm{d}\lambda,$$

with the usual λ-parameterized model notation introduced later in Section 9.4.

At first order, holding the neural prediction fixed while retaining the known change-of-variable weight gives the same finite-step update, so DPM-Solver-1 is exactly DDIM. Beyond first order, however, the coordinate in which the state–time-dependent neural prediction is approximated becomes part of the numerical method. DPM-Solver uses λ because it yields an exponentially weighted integral with convenient higher-order expansions.

9.2.3 (Optional) The Original Variational Perspective on DDIM

The PF-ODE perspective above explains deterministic DDIM as a numerical solver. Historically, however, DDIM was introduced from the variational viewpoint of DDPM. Its starting question is different: *does the denoising objective uniquely determine how noisy states at different times must be coupled?*

Revisiting DDPM's Variational View. Recall that DDPM introduces a fixed Markovian inference process whose one-time perturbation marginals take the Gaussian form

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}).$$

The corresponding variational derivation involves tractable conditionals such as $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ for $t < s$, while the learned reverse model used for generation cannot condition directly on the unknown clean sample $\mathbf{x}_0$ and instead relies on the diffusion-model prediction.

The key observation behind DDIM concerns the denoising objective. We recall that a noise-prediction DDPM loss can be written as

$$\mathbb{E}_t \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, t) - \boldsymbol{\epsilon} \right\|_2^2 \right],$$

which depends on the inference process through the one-time marginals $p_t(\mathbf{x}_t|\mathbf{x}_0)$, rather than through the full joint coupling of $\mathbf{x}_1, \ldots, \mathbf{x}_T$ conditioned on $\mathbf{x}_0$.

Original DDIM Motivation. The central observation behind DDIM is that the denoising training objective does not uniquely determine how noisy states at different times are coupled. For a fixed clean sample $\mathbf{x}_0$, the usual denoising surrogate is determined by the one-time perturbation marginals $p_t(\mathbf{x}_t|\mathbf{x}_0)$, whereas many different joint processes across time can share exactly the same marginals. DDIM exploits this freedom: it preserves the perturbation marginals used during training, while changing the coupling between different noise levels.

Concretely, for any $t < s$, consider a conditional transition $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$. We require it to be *one-step marginally consistent*:

$$\int \pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)\, p_s(\mathbf{x}_s|\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_s = p_t(\mathbf{x}_t|\mathbf{x}_0). \tag{9.2.6}$$

In words, if $\mathbf{x}_s \sim p_s(\cdot|\mathbf{x}_0)$, then applying the chosen transition once produces $\mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_0)$. Hence, the marginal distribution at each noise level is preserved even though the dependence between $\mathbf{x}_s$ and $\mathbf{x}_t$ may be changed.

The Bayesian reverse posterior induced by the original Markovian DDPM forward process is one valid choice of such a transition,[6] but it is not uniquely determined by the marginals. DDIM instead allows other marginal-consistent couplings, including transitions directly between widely separated noise levels. This is the probabilistic reason that the same trained denoiser can be reused on a coarser sampling grid with skipped intermediate times.

Derivation of Discrete-Time DDIM. We now make the coupling freedom above explicit for the Gaussian perturbation model

$$p_t(\mathbf{x}_t|\mathbf{x}_0) := \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}), \qquad \mathbf{x}_0 \sim p_{\text{data}}.$$

The goal is to construct, for any $t < s$, a conditional transition $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ that satisfies the marginal-consistency condition in Equation (9.2.6), without requiring it to coincide with the Bayesian posterior associated with the original Markov forward process.

A natural choice is the Gaussian family

$$\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\, a_{t,s}\mathbf{x}_0 + b_{t,s}\mathbf{x}_s,\, c_{t,s}^2 \mathbf{I}), \tag{9.2.7}$$

where $(a_{t,s}, b_{t,s}, c_{t,s})$ determine how the state at time s, the clean sample, and newly injected noise are combined.

To impose marginal consistency, first draw

$$\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}', \qquad \boldsymbol{\epsilon}' \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

and then draw $\mathbf{x}_t$ from Equation (9.2.7). Equivalently,

$$\begin{aligned} \mathbf{x}_t &= a_{t,s}\mathbf{x}_0 + b_{t,s}\mathbf{x}_s + c_{t,s}\boldsymbol{\epsilon} \\ &= a_{t,s}\mathbf{x}_0 + b_{t,s}(\alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}') + c_{t,s}\boldsymbol{\epsilon} \\ &= (a_{t,s} + b_{t,s}\alpha_s)\mathbf{x}_0 + \sqrt{b_{t,s}^2\sigma_s^2 + c_{t,s}^2}\, \boldsymbol{\epsilon}'', \end{aligned} \tag{9.2.8}$$

[6] For the Markov forward process, $\mathbf{x}_0 \to \mathbf{x}_t \to \mathbf{x}_s$ with $t < s$, the conditional posterior $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ satisfies

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \int p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)\, p_s(\mathbf{x}_s|\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_s$$

by the law of total probability.

where $\boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon}'$ are independent standard Gaussian variables, and $\boldsymbol{\epsilon}''$ is a standard Gaussian representing their normalized linear combination.

For the resulting marginal to equal the prescribed perturbation marginal $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$, the means and variances must agree. Therefore,

$$\begin{aligned} \alpha_t &= a_{t,s} + b_{t,s}\alpha_s, \\ \sigma_t^2 &= b_{t,s}^2 \sigma_s^2 + c_{t,s}^2. \end{aligned}$$

The important point is that these are only two constraints on the three coefficients $(a_{t,s}, b_{t,s}, c_{t,s})$. Hence, the prescribed marginals do *not* determine a unique transition between s and t. We may choose $c_{t,s}$ as a free parameter, with $0 \le c_{t,s} \le \sigma_t$, and solve for the remaining coefficients. Choosing the nonnegative branch gives

$$b_{t,s} = \frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s}, \qquad a_{t,s} = \alpha_t - \alpha_s b_{t,s}. \tag{9.2.9}$$

The nonnegative choice of $b_{t,s}$ gives the standard DDIM coupling.

Substituting Equation (9.2.9) into Equation (9.2.7) yields

$$\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t;\, \alpha_t \mathbf{x}_0 + \frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s}(\mathbf{x}_s - \alpha_s \mathbf{x}_0),\, c_{t,s}^2 \mathbf{I}\right). \tag{9.2.10}$$

This expression makes the coupling freedom explicit. The term $\mathbf{x}_s - \alpha_s \mathbf{x}_0$ is the noise component already present in $\mathbf{x}_s$, while $c_{t,s}$ controls how much *new* Gaussian noise is introduced in the transition from s to t. Varying $c_{t,s}$ therefore changes the coupling between the two noise levels while preserving the same target marginal $p_t(\mathbf{x}_t|\mathbf{x}_0)$.

Remark.

> The chosen transition $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ is generally different from the Bayesian posterior associated with the original Markov forward process. That posterior is recovered by one particular choice of $c_{t,s}$, discussed below. Other choices define different marginal-consistent couplings, including the deterministic DDIM coupling obtained when $c_{t,s} = 0$.

DDIM Sampler (Step $s \to t$). The marginal-consistent kernel $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ in Equation (9.2.10) still conditions on the unknown clean sample $\mathbf{x}_0$. During generation, DDIM replaces $\mathbf{x}_0$ by the clean-data estimate obtained from the pre-trained diffusion model. Under $\boldsymbol{\epsilon}$-prediction, we define

$$\mathbf{x}_{\phi^\times}(\mathbf{x}_s, s) := \frac{\mathbf{x}_s - \sigma_s \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_s, s)}{\alpha_s},$$

and use the plug-in transition

$$p_{\phi^\times}(\mathbf{x}_t|\mathbf{x}_s) := \pi\big(\mathbf{x}_t \,\big|\, \mathbf{x}_s, \mathbf{x}_{\phi^\times}(\mathbf{x}_s, s)\big).$$

Substituting this estimate into Equation (9.2.10) yields

$$\mathbf{x}_t = \frac{\alpha_t}{\alpha_s}\mathbf{x}_s + \left(\sqrt{\sigma_t^2 - c_{t,s}^2} - \frac{\alpha_t}{\alpha_s}\sigma_s\right)\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_s, s) + c_{t,s}\boldsymbol{\epsilon}_t, \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Thus, the model prediction determines the estimated clean/noise content carried from s to t, while the free coefficient $c_{t,s} \in [0, \sigma_t]$ controls how much *new* Gaussian noise is injected during the step.

To interpret this degree of freedom, it is useful to compare with the original Markov forward process. For $t < s$, define

$$\alpha_{s|t} := \frac{\alpha_s}{\alpha_t}, \qquad \sigma_{s|t}^2 := \sigma_s^2 - \alpha_{s|t}^2\sigma_t^2,$$

so that

$$p(\mathbf{x}_s|\mathbf{x}_t) = \mathcal{N}\big(\mathbf{x}_s; \alpha_{s|t}\mathbf{x}_t, \sigma_{s|t}^2\mathbf{I}\big).$$

Gaussian conditioning gives the corresponding posterior variance

$$\mathrm{Var}(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = \frac{\sigma_t^2}{\sigma_s^2}\sigma_{s|t}^2\mathbf{I}.$$

This identifies the DDPM posterior as one particular member of the marginal-consistent family above and makes the role of $c_{t,s}$ explicit:

- **DDPM Step (Posterior Variance).** Choosing

  $$c_{t,s} = \frac{\sigma_t}{\sigma_s}\sigma_{s|t}$$

  matches the posterior variance of the original Markov forward process. Together with the nonnegative branch in Equation (9.2.9), the conditional kernel $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ then coincides with the Bayesian posterior $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ of that process.

- **Deterministic DDIM ($\eta = 0$).** At the opposite extreme, setting $c_{t,s} = 0$ removes all newly injected noise. The practical update becomes

  $$\mathbf{x}_t = \frac{\alpha_t}{\alpha_s}\mathbf{x}_s + \left(\sigma_t - \frac{\alpha_t}{\alpha_s}\sigma_s\right)\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_s, s),$$

  which is exactly the deterministic DDIM update derived earlier from the PF-ODE.

At the conditional level, its shared-noise interpretation is especially simple: if $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$, then the deterministic transition gives

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}.$$

Thus, no new noise is introduced; the same pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$ is carried across the step while only the known schedule coefficients change.

- **Interpolation.** More generally, define

$$c_{t,s} = \eta \frac{\sigma_t}{\sigma_s} \sigma_{s|t}, \qquad \eta \in [0, 1]. \tag{9.2.11}$$

The parameter η continuously controls the stochasticity of the transition: $\eta = 0$ gives deterministic DDIM, whereas $\eta = 1$ matches the posterior variance of the corresponding DDPM Markov transition.

Hence, the DDIM family separates two ingredients that are fixed differently: the perturbation marginals are kept unchanged, while the coupling between successive noise levels can be varied through $c_{t,s}$. Deterministic DDIM corresponds to preserving the existing shared noise across the step, whereas the DDPM posterior choice injects the amount of additional randomness prescribed by the original Markov forward process.

Connection to Conditional Flow Matching. The deterministic DDIM construction above also provides a simple connection to conditional flow matching (CFM) in Section 5.2. The key is to distinguish two objects associated with the same conditional Gaussian path: its *finite-time map* and its *instantaneous velocity*. DDIM is related to the former, whereas CFM is built from the latter.

Condition on a clean sample $\mathbf{x}_0$ and consider the shared-noise path

$$\mathbf{x}_\tau = \alpha_\tau \mathbf{x}_0 + \sigma_\tau \boldsymbol{\epsilon}.$$

For fixed $(\mathbf{x}_0, \boldsymbol{\epsilon})$, this is an exact conditional trajectory through the Gaussian perturbation marginals $p_\tau(\mathbf{x}_\tau | \mathbf{x}_0)$.

Finite-Time View: the Conditional DDIM Map. Suppose the trajectory is at $\mathbf{x}_s$ at time s. Since the same $\boldsymbol{\epsilon}$ is retained, eliminating it from $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ gives the exact conditional map

$$\boldsymbol{\Psi}_{s \to t}(\mathbf{x}_s | \mathbf{x}_0) = \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \left(\alpha_t - \alpha_s \frac{\sigma_t}{\sigma_s} \right) \mathbf{x}_0.$$

This map is exact *conditioned on* $\mathbf{x}_0$. It is not yet the practical DDIM sampler, because $\mathbf{x}_0$ is unknown during generation. Replacing $\mathbf{x}_0$ by its start-time model prediction gives precisely the deterministic DDIM update derived above.

Infinitesimal-Time View: the CFM Conditional Velocity. Instead of asking where the conditional path moves over a finite interval, we may ask for its instantaneous direction. Differentiating $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ gives

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \alpha_t' \mathbf{x}_0 + \sigma_t' \boldsymbol{\epsilon}.$$

Using

$$\boldsymbol{\epsilon} = \frac{\mathbf{x} - \alpha_t \mathbf{x}_0}{\sigma_t},$$

the same velocity can be written as a function of the current state:

$$\mathbf{v}_t^*(\mathbf{x}|\mathbf{x}_0) = \frac{\sigma_t'}{\sigma_t}\mathbf{x} + \left(\alpha_t' - \alpha_t \frac{\sigma_t'}{\sigma_t}\right)\mathbf{x}_0.$$

This is the conditional velocity associated with the Gaussian probability path in conditional flow matching.

CFM then connects this conditional velocity to the marginal dynamics by averaging over the posterior uncertainty about $\mathbf{x}_0$ at the current state:

$$\begin{aligned}\mathbf{v}^*(\mathbf{x}, t) &:= \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0|\mathbf{x}_t=\mathbf{x})}\left[\mathbf{v}_t^*(\mathbf{x}|\mathbf{x}_0)\right] \\ &= \frac{\sigma_t'}{\sigma_t}\mathbf{x} + \left(\alpha_t' - \alpha_t \frac{\sigma_t'}{\sigma_t}\right)\mathbf{x}^*(\mathbf{x}, t) \\ &= \alpha_t' \mathbf{x}^*(\mathbf{x}, t) + \sigma_t' \boldsymbol{\epsilon}^*(\mathbf{x}, t),\end{aligned}$$

where we used

$$\mathbf{x}^*(\mathbf{x}, t) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t = \mathbf{x}], \qquad \mathbf{x} = \alpha_t \mathbf{x}^*(\mathbf{x}, t) + \sigma_t \boldsymbol{\epsilon}^*(\mathbf{x}, t).$$

The resulting $\mathbf{v}^*(\mathbf{x}, t)$ is exactly the oracle marginal PF-ODE velocity.

DDIM and CFM

The same conditional Gaussian path gives two views: DDIM uses its finite-time map after replacing the unknown $\mathbf{x}_0$ by a model prediction, while CFM uses its instantaneous conditional velocity. Posterior averaging the latter gives the marginal PF-ODE velocity.

This connection is exact at the level of the conditional path and its instantaneous velocity. It does *not* imply that a finite DDIM step is the exact marginal PF-ODE flow: DDIM keeps the prediction inferred at the beginning of the step fixed, whereas the exact marginal PF-ODE velocity changes continuously with the evolving state.

9.3 DEIS

In the factored exponential-integrator formula (Equation (9.1.7)),

$$\int_s^t \frac{g^2(\tau)}{2\sigma_\tau}\,\mathcal{E}(\tau \to t)\,\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau,$$

all schedule-dependent factors are known; the only unknown quantity along the trajectory is the neural prediction $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$. DDIM makes the simplest first-order approximation by holding this prediction fixed at the beginning of the step,

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s), \qquad \tau \in [t, s].$$

This approximation can become inaccurate over larger steps when the neural prediction varies appreciably along the trajectory.

DEIS (Zhang and Chen, 2022) improves on this idea without introducing additional model evaluations within the current step. Inspired by classical multistep methods such as Adams–Bashforth (see Section A.1.5), it reuses model predictions from previous completed steps as *anchors* and fits a polynomial that extrapolates their recent variation into the next step. The polynomial then replaces the unknown neural prediction inside the integral, while the known schedule-dependent weight $\frac{g^2(\tau)}{2\sigma_\tau}\mathcal{E}(\tau \to t)$ is retained analytically. Thus, DEIS extends the constant approximation of DDIM to a higher-order multistep polynomial approximation.

From the classical numerical viewpoint, DEIS is therefore an Adams–Bashforth-type multistep method within the factored exponential-integrator framework: previous neural evaluations provide the higher-order information, while the known diffusion dynamics are retained in the weighted integral.

We first review the polynomial extrapolation used to construct this approximation in Section 9.3.1, then derive the resulting DEIS update in Section 9.3.2. Finally, Section 9.3.3 shows that the constant, degree-zero case recovers DDIM.

9.3.1 Background: Polynomial Extrapolation

Before deriving DEIS, we briefly review the polynomial construction it uses to reuse previous model evaluations. The idea is simple: first fit a polynomial through several known values, and then evaluate that polynomial beyond the most recent anchor to predict how the quantity may continue to change. Thus, the polynomial is an *interpolant* at the known anchors and acts as an *extrapolant* over the next solver step.

From Anchors to a Polynomial Extrapolant. Suppose a time-varying quantity is known at $n+1$ distinct times,

$$(\tau_0, \mathbf{Y}_0),\ (\tau_1, \mathbf{Y}_1),\ \ldots,\ (\tau_n, \mathbf{Y}_n), \qquad \tau_0 < \tau_1 < \cdots < \tau_n,$$

where each $\mathbf{Y}_j$ may be vector-valued. We would like to construct the simplest smooth curve that exactly matches these observed values and can then be continued beyond the most recent anchor. The unique polynomial of degree at most n passing through all anchors is the *Lagrange interpolant*

$$P_n(\tau) = \sum_{j=0}^{n} \ell_j(\tau)\, \mathbf{Y}_j,$$

with basis functions

$$\ell_j(\tau) := \prod_{\substack{k=0 \\ k\neq j}}^{n} \frac{\tau - \tau_k}{\tau_j - \tau_k}.$$

The construction of ℓ_j is simple: every factor makes it vanish at one of the other anchors, while the denominator normalizes its value to 1 at τ_j. Hence,

$$\ell_j(\tau_k) = \delta_{jk},$$

and therefore

$$P_n(\tau_j) = \mathbf{Y}_j, \qquad j = 0, \ldots, n.$$

Moreover,

$$\sum_{j=0}^{n} \ell_j(\tau) = 1.$$

Thus, $P_n(\tau)$ is a time-dependent linear combination of the observed anchors: at each anchor time it reproduces exactly the corresponding value, while between or beyond the anchors the polynomial combines them to describe the local trend suggested by their variation. In DEIS, we will use this same curve beyond the most recent anchor, turning the interpolant into an *extrapolant* for the next solver step.

The lowest-order cases illustrate the construction:

Example:

$n = 0$ **(Constant).** With one anchor, the polynomial simply keeps the most recent value fixed:

$$P_0(\tau) = \mathbf{Y}_0.$$

$n = 1$ **(Linear).** With two anchors, the polynomial is the straight line joining them:

$$P_1(\tau) = \frac{\tau - \tau_1}{\tau_0 - \tau_1}\mathbf{Y}_0 + \frac{\tau - \tau_0}{\tau_1 - \tau_0}\mathbf{Y}_1.$$

$n = 2$ **(Quadratic).** With three anchors, the interpolant becomes

$$P_2(\tau) = \ell_0(\tau)\mathbf{Y}_0 + \ell_1(\tau)\mathbf{Y}_1 + \ell_2(\tau)\mathbf{Y}_2,$$

which can additionally capture local curvature in the observed values. ■

In DEIS, the anchors will be neural predictions from previous completed sampling steps. The polynomial matches those stored predictions exactly and is then extrapolated beyond the most recent anchor over the next reverse-time interval.

9.3.2 DEIS: Lagrange Polynomial Approximation of the PF-ODE Integral

Let $n \geq 0$ be the chosen polynomial degree. At step i, DEIS approximates the unknown map $\tau \mapsto \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_\tau, \tau)$ over the next reverse-time interval $[t_i, t_{i-1}]$ using a degree-n Lagrange polynomial constructed from past model outputs. This polynomial is then substituted into the exponential–integrator update (Equation (9.1.7)) to obtain $\tilde{\mathbf{x}}_{t_i}$. The intuition is that, instead of holding the neural prediction constant as in DDIM, DEIS uses its recent history to capture how it is changing over time.

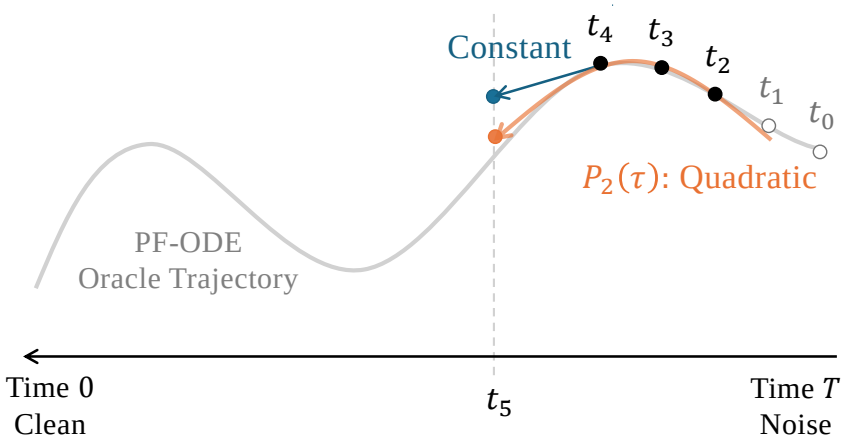


Figure 9.2: Illustration of DEIS as a multistep method. With three past anchors at t_2, t_3, t_4, DEIS builds a quadratic curve through the model outputs and extrapolates it over the step from t_4 to t_5. The extrapolating polynomial is then integrated against the known diffusion-dependent weight, capturing variation of the neural prediction that is neglected by the constant approximation used by DDIM.

Source: Created by the authors.

A degree-n update needs $n{+}1$ anchors. When they are available (*sufficient history*, $i \geq n{+}1$), we use the full degree-n scheme. During the initial steps (*insufficient history*, $i \leq n$), we use the highest feasible degree, $i{-}1$, and increase it as more anchors become available. We treat these two cases below.

Case I: $i = n+1, \ldots, M$ (Sufficient History). With sufficient history, DEIS reuses the last $n{+}1$ model evaluations as anchors,

$$(\tau_j, \mathbf{Y}_j) := (t_{i-1-j}, \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j})), \quad j = 0, \ldots, n.$$

Viewing $\tau \mapsto \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ as a smooth function along the trajectory, we construct the degree-n Lagrange polynomial

$$P_n(\tau) = \sum_{j=0}^{n} \underbrace{\Big[\prod_{\substack{k=0 \\ k\neq j}}^{n} \frac{\tau - t_{i-1-k}}{t_{i-1-j} - t_{i-1-k}}\Big]}_{=:\, \ell_j^{(i)}(\tau)} \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j})$$

which by construction satisfies $P_n(\tau_j) = \mathbf{Y}_j$ for each anchor:

$$P_n(\tau_j) = \mathbf{Y}_j = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}), \quad j = 0, \ldots, n.$$

Each $\ell_j^{(i)}$ satisfies

$$\ell_j^{(i)}(t_{i-1-m}) = \begin{cases} 1, & m = j, \\ 0, & m \neq j. \end{cases}$$

The polynomial is then extrapolated beyond the most recent anchor t_{i-1} over the new step:

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx P_n(\tau) = \sum_{j=0}^{n} \ell_j^{(i)}(\tau) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}), \quad \tau \in [t_i, t_{i-1}].$$

We then substitute $P_n(\tau)$ for $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)$ in the exponential–integrator formula (Equation (9.1.7)):

$$\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \to t_i) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau)\, \mathrm{d}\tau \approx \sum_{j=0}^{n} \underbrace{\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \to t_i) \ell_j^{(i)}(\tau)\, \mathrm{d}\tau}_{=:\, C_{i,j}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}).$$

The coefficients

$$C_{i,j} := \frac{1}{2} \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \to t_i) \ell_j^{(i)}(\tau)\, \mathrm{d}\tau$$

depend only on the noise schedule and the chosen time grid and can therefore be precomputed. For schedules admitting suitable antiderivatives, these coefficients are available in closed form; otherwise they can be evaluated numerically to high accuracy.

Integrating the linear part exactly with $\mathcal{E}(t_{i-1} \to t_i)$ gives the *AB-DEIS-n* update

$$\tilde{\mathbf{x}}_{t_i} = \mathcal{E}(t_{i-1} \to t_i) \tilde{\mathbf{x}}_{t_{i-1}} + \sum_{j=0}^{n} C_{i,j} \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}).$$

For the local truncation statements below, let h denote a characteristic local step size and assume neighboring step ratios remain bounded. Under standard smoothness assumptions on the model prediction along the empirical trajectory and regularity of the schedule-dependent weight, the full-history degree-n construction has formal order $n + 1$, with one-step local defect $\mathcal{O}(h^{n+2})$.

Case II: $i = 1, \ldots, n$ (Insufficient History). During the initial steps, only i past model evaluations are available. We therefore use all available anchors and set the polynomial degree to $i-1$:

$$P_{i-1}(\tau) = \sum_{j=0}^{i-1} \ell_j^{(i)}(\tau) \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}),$$

where $\ell_j^{(i)}$ is the Lagrange basis of degree $i-1$ built on the nodes at time $\{t_{i-1}, t_{i-2}, \ldots, t_0\}$. This matches all available anchors and transitions into the full-history formula once $i \geq n+1$.

This is the standard warm-start situation for a multistep method. The first step uses a constant approximation, the second uses a linear extrapolation, the third uses a quadratic extrapolation, and so on, until the target degree n becomes available.

Example: Special Cases of Lagrange Polynomials

Degree 0 (one anchor):

$$P_0(\tau) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}).$$

This holds the neural prediction fixed at its start-of-step value, exactly as in the first-order DDIM approximation.

Degree 1 (two anchors): the Lagrange polynomial is a linear map passing through the two pre-specified anchors.

$$P_1(\tau) = \underbrace{\frac{\tau - t_{i-2}}{t_{i-1} - t_{i-2}}}_{\ell_{i-1}(\tau)} \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) + \underbrace{\frac{\tau - t_{i-1}}{t_{i-2} - t_{i-1}}}_{\ell_{i-2}(\tau)} \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2}).$$

Here $\ell_{i-1}(\tau)$ and $\ell_{i-2}(\tau)$ are the Lagrange basis weights. They satisfy the interpolation conditions $P_1(t_{i-1}) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})$ and $P_1(t_{i-2}) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2})$, with $\ell_{i-1}(\tau) + \ell_{i-2}(\tau) = 1$.

Summary of AB-DEIS-n Update. Combining the sufficient-history and warm-start cases gives the *AB-DEIS-n* update[7] Here, n is the target polynomial degree, using up to $n+1$ past evaluations:

[7] "AB" refers to the Adams–Bashforth multistep principle, here applied within the exponential-integrator formulation (Hochbruck and Ostermann, 2010).

$$\tilde{\mathbf{x}}_{t_i} = \mathcal{E}(t_{i-1} \to t_i)\,\tilde{\mathbf{x}}_{t_{i-1}} + \sum_{j=0}^{\min\{n,\,i-1\}} C_{i,j}\,\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}},\, t_{i-1-j}),$$

with coefficients

$$C_{i,j} := \frac{1}{2}\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau}\,\mathcal{E}(\tau \to t_i)\left[\prod_{\substack{k=0\\k\neq j}}^{\min\{n,\,i-1\}} \frac{\tau - t_{i-1-k}}{t_{i-1-j} - t_{i-1-k}}\right] d\tau.$$

When $i \geq n{+}1$ (sufficient history), $\min\{n,\, i-1\} = n$, and the full-history degree-n step has one-step local truncation error $\mathcal{O}(h^{n+2})$ under the assumptions above, corresponding to a method of order $n+1$.

During warm start ($i \leq n$), the degree is $i-1$, so the one-step local truncation error is $\mathcal{O}(h^{i+1})$, and the corresponding step is formally of order i. Thus, the order increases from 1 at the first step and reaches the full order $n{+}1$ once $n{+}1$ anchors are available. As with standard multistep methods, lower-order warm-start steps do not by themselves establish a global order-$(n+1)$ guarantee; such a guarantee also requires stability and starting values of sufficient accuracy.

Very large n can degrade numerical behavior because high-degree extrapolation becomes increasingly sensitive to the spacing of the anchors, errors in the stored model predictions, and stability considerations. Relatively small polynomial degrees are therefore preferred in practice.

As we will see in the following subsection, the special case $n{=}0$ reduces exactly to deterministic DDIM, i.e., the first-order factored exponential-integrator update.

9.3.3 DDIM = AB-DEIS-0

The connection to DDIM becomes immediate at degree $n = 0$. In this case, the Lagrange polynomial is constant, so DEIS uses only the most recent model prediction throughout the next step. The corresponding coefficient simplifies to

$$C_{i0} = \frac{1}{2}\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau}\mathcal{E}(\tau \to t_i)\mathrm{d}\tau.$$

Substituting into the update formula yields the zeroth-order AB-DEIS scheme:

$$\begin{aligned}\tilde{\mathbf{x}}_{t_i} &= \mathcal{E}(t_{i-1} \to t_i)\tilde{\mathbf{x}}_{t_{i-1}} + C_{i0}\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) \\ &= e^{\int_{t_{i-1}}^{t_i} f(u)\,\mathrm{d}u}\tilde{\mathbf{x}}_{t_{i-1}} + \Big(\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} e^{\int_{\tau}^{t_i} f(u)\,\mathrm{d}u}\mathrm{d}\tau\Big)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}). \qquad (9.3.1)\end{aligned}$$

This is exactly the same first-order factored approximation used by deterministic DDIM: the neural prediction is held fixed at t_{i-1}, while the known integrating

factor and schedule-dependent coefficient remain inside the integral and are propagated over the full step. Hence, AB-DEIS-0 and DDIM are not merely first-order equivalent; they give the same finite-step update.

Proposition 9.3.1: DDIM = AB-DEIS-0

Equation (9.3.1) is identical to the DDIM update in Equation (9.2.1).

This also clarifies the role of DEIS in the solver hierarchy. We remark that the index "0" refers to the *polynomial degree*, not the numerical order: AB-DEIS-0 is still a first-order method. Higher-degree DEIS keeps the same analytic treatment of the known diffusion dynamics but replaces this constant prediction by polynomial extrapolation from additional past model evaluations. Thus, DDIM is the degree-zero base case from which the DEIS multistep construction naturally extends.

Later, in Section 9.5.4, we will encounter another route to a multistep diffusion solver; that is, DPM-Solver++(2M). It starts from a local Taylor expansion of the data prediction in log-SNR time and recycles previous model evaluations through a backward finite difference to estimate the required derivative. Although its representation and derivation differ from those of DEIS, both methods use past model evaluations to capture the variation of the neural prediction without introducing additional within-step stages. We return to this connection in Section 9.5.5, where we compare the two constructions more explicitly.

9.4 DPM-Solver

DPM-Solver (Lu *et al.*, 2022b) develops higher-order solvers for the empirical PF-ODE by exploiting the same diffusion-specific structure that appeared in our discussion of DDIM. The central goal is to obtain accurate samples using far fewer neural-network evaluations than a first-order sampler. The original DPM-Solver focuses on noise prediction and constructs first-, second-, and third-order single-step solvers; later extensions, including DPM-Solver++ (Lu *et al.*, 2022c) and DPM-Solver-v3 (Zheng *et al.*, 2023), further develop this family.

High-Level Idea of DPM-Solver. The DDIM discussion in Section 9.2 showed that, under $\boldsymbol{\epsilon}$-prediction, a first-order update can be obtained by retaining the known diffusion schedule analytically while holding the neural prediction fixed over each step,

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

DPM-Solver (Lu *et al.*, 2022b) builds naturally on this idea: rather than treating the neural prediction as constant, it seeks higher-order approximations of how the prediction varies along the trajectory. Like DEIS, it starts from the semilinear $\boldsymbol{\epsilon}$-prediction form of the PF-ODE,

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\alpha_t'}{\alpha_t}\mathbf{x}_t - \sigma_t\left(\frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t}\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t), \tag{9.4.1}$$

and its variation-of-constants representation introduced in the Prologue. Its key additional ingredient is to reparameterize time by the *half-log signal-to-noise ratio*, following the log-SNR parameterization of VDM (Kingma *et al.*, 2021),

$$\lambda_t := \log\frac{\alpha_t}{\sigma_t}. \tag{9.4.2}$$

This coordinate is directly related to DDIM's schedule-aligned clock $\rho_t = \sigma_t/\alpha_t$ (discussed around Equation (9.2.4)) through $\rho_t = e^{-\lambda_t}$, and turns the remaining model-dependent term into a particularly convenient exponentially weighted integral. Local Taylor approximations of the neural prediction in λ can then be integrated analytically, providing a systematic route from the first-order DDIM update to higher-order diffusion solvers.

9.4.1 DPM-Solver's Insight: Time Reparameterization via Log-SNR

We now make the log-SNR reparameterization in Equation (9.4.2) precise and derive the corresponding PF-ODE representation.

Change-of-Variable to Log-SNR in PF-ODE. Suppose that $\alpha_t > 0$, $\sigma_t > 0$, and the log-SNR coordinate λ_t in Equation (9.4.2) is strictly monotone on the interval of interest. It then admits an inverse function $t_\lambda(\cdot)$ satisfying

$$t = t_\lambda(\lambda(t)).$$

For schedules with $\sigma_0 = 0$, one has $\lambda_t \to +\infty$ as $t \to 0$, so formulas involving the endpoint $t = 0$ are understood through the corresponding limit or through a small positive terminal time in a numerical implementation.

We then express the state, schedule, and neural prediction in the λ-coordinate. A hat $(\hat{\cdot})$ together with λ indicates that the corresponding quantity is viewed in this reparameterized time coordinate. More precisely, define

$$\begin{aligned} \hat{\mathbf{x}}_\lambda &:= \mathbf{x}_{t_\lambda(\lambda)}, \\ \alpha_\lambda &:= \alpha_{t_\lambda(\lambda)}, \qquad \sigma_\lambda := \sigma_{t_\lambda(\lambda)}, \\ \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\mathbf{x}, \lambda) &:= \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}\big(\mathbf{x}, t_\lambda(\lambda)\big). \end{aligned} \tag{9.4.3}$$

Hence, along the empirical PF-ODE trajectory,

$$\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}\big(\mathbf{x}_{t_\lambda(\lambda)}, t_\lambda(\lambda)\big).$$

Since

$$\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} = \frac{\alpha_t'}{\alpha_t} - \frac{\sigma_t'}{\sigma_t},$$

the PF-ODE in Equation (9.4.1) can be written as

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\mathbf{x}_t - \sigma_t \frac{\mathrm{d}\lambda_t}{\mathrm{d}t}\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t).$$

Applying the chain rule,

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\mathrm{d}\hat{\mathbf{x}}_\lambda}{\mathrm{d}\lambda}\frac{\mathrm{d}\lambda_t}{\mathrm{d}t}, \qquad \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t} = \frac{\mathrm{d}\log\alpha_\lambda}{\mathrm{d}\lambda}\frac{\mathrm{d}\lambda_t}{\mathrm{d}t},$$

gives the PF-ODE in log-SNR time:

$$\frac{\mathrm{d}\hat{\mathbf{x}}_\lambda}{\mathrm{d}\lambda} = \frac{\mathrm{d}\log\alpha_\lambda}{\mathrm{d}\lambda}\hat{\mathbf{x}}_\lambda - \sigma_\lambda\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda). \tag{9.4.4}$$

Applying the same exponential-integrator (variation-of-constants) technique to Equation (9.4.4) gives the following exact solution.

Proposition 9.4.1: Exponentially Weighted Exact Solution

Given an initial value $\mathbf{x}_s$ at time $s > 0$, the exact empirical PF-ODE solution $\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s)$ at time $t \in (0, s]$ can be re-expressed as

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) = \frac{\alpha_t}{\alpha_s}\mathbf{x}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\, \mathrm{d}\lambda. \tag{9.4.5}$$

When the endpoint satisfies $\sigma_0 = 0$, the case $t = 0$ is understood through the corresponding limit.

Proof for Proposition.

From Equation (9.4.4), the integrating factor associated with the linear term satisfies

$$\exp\left(\int_{\lambda_s}^{\lambda_t} \frac{\mathrm{d}\log\alpha_\lambda}{\mathrm{d}\lambda}\,\mathrm{d}\lambda\right) = \frac{\alpha_t}{\alpha_s}.$$

Therefore, the variation-of-constants formula gives

$$\begin{aligned}\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) &= \frac{\alpha_t}{\alpha_s}\mathbf{x}_s - \int_{\lambda_s}^{\lambda_t} \frac{\alpha_t}{\alpha_\lambda}\sigma_\lambda \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\, \mathrm{d}\lambda \\ &= \frac{\alpha_t}{\alpha_s}\mathbf{x}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\, \mathrm{d}\lambda,\end{aligned}$$

where the last equality follows from $\sigma_\lambda/\alpha_\lambda = e^{-\lambda}$.

The key numerical consequence is now explicit. In λ-time, the only model-dependent quantity in the exact solution appears inside

$$\int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\, \mathrm{d}\lambda.$$

When the model output is locally approximated by a Taylor polynomial in λ, the resulting exponentially weighted polynomial terms can be integrated in closed form, yielding explicit higher-order update coefficients. Thus, as in the factored viewpoint of the Prologue, the known schedule dependence is retained analytically while the approximation is concentrated on the neural prediction.

Intuition of Why Reparameterize Time? For strictly monotone $\lambda(t)$, a local first-order change of variables gives

$$|\Delta t| \approx \frac{|\Delta\lambda|}{|\lambda'(t)|}.$$

Thus, if the numerical grid uses a fixed step size $|\Delta\lambda|$, the induced step in t is smaller where $|\lambda'(t)|$ is larger (i.e., where λ changes rapidly with t), and larger where $|\lambda'(t)|$ is smaller.

The reparameterization itself does not alter the PF-ODE solution path; it only changes how the same path is parameterized:

$$\frac{\mathrm{d}\hat{\mathbf{x}}_\lambda}{\mathrm{d}\lambda} = \frac{1}{\lambda'(t)} \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t}.$$

Hence, where $|\lambda'(t)|$ is large, the same change of the trajectory with respect to t is represented over a larger range of λ.

More importantly, $\lambda(t)$ measures progress through the diffusion process directly in terms of the relative balance between signal and noise. A fixed $\Delta\lambda$ corresponds to a fixed multiplicative change in α_t/σ_t, or equivalently to a fixed additive change in log-SNR, whereas the same Δt may correspond to very different changes in corruption level at different times. Therefore, choosing numerical nodes uniformly in λ provides a natural way to redistribute resolution along the diffusion trajectory. The precise induced spacing in the original time variable t depends on the chosen noise schedule. Two simple examples illustrate this effect:

- $(\alpha_t, \sigma_t) = (1 - t, t)$**.** This corresponds to the linear interpolation commonly used in flow matching. Then

  $$\lambda(t) = \log \frac{1-t}{t}, \qquad \lambda'(t) = -\frac{1}{t(1-t)},$$

  so locally

  $$|\Delta t| \approx |\Delta\lambda|\, t(1-t).$$

  Hence, a uniform grid in λ induces very small steps in t near both endpoints $t \to 0$ and $t \to 1$, and larger steps around the middle of the trajectory.

- $(\alpha_t, \sigma_t) = (1, t)$**.** This corresponds to the EDM parameterization (Karras *et al.*, 2022) when the independent variable t is taken directly as the noise level σ_t. Then

  $$\lambda(t) = \log \frac{1}{t}, \qquad \lambda'(t) = -\frac{1}{t},$$

  and therefore

  $$|\Delta t| \approx |\Delta\lambda|\, t.$$

  In fact, uniform spacing in λ implies

  $$t = e^{-\lambda},$$

  so the corresponding noise levels are geometrically spaced. Thus, the grid contains finer absolute spacing at small noise levels (high SNR) and coarser absolute spacing at large noise levels (low SNR).

9.4.2 Estimating the Integral with Taylor Expansion

DEIS and DPM-Solver exploit the same factored principle but obtain higher-order information in different ways. DEIS reuses predictions from previous completed steps to extrapolate the neural prediction over the next interval. DPM-Solver instead works in the log-SNR coordinate λ and uses a local Taylor expansion of the neural prediction, with the required higher-order information obtained from additional evaluations within the current step. We now derive this single-step construction.

From Equation (9.4.5), the exact empirical PF-ODE flow initialized at the current numerical state $\tilde{\mathbf{x}}_s$ is

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\,\mathrm{d}\lambda, \tag{9.4.6}$$

where $\hat{\mathbf{x}}_{\lambda_s} = \tilde{\mathbf{x}}_s$ and $\hat{\mathbf{x}}_\lambda$ denotes the corresponding exact empirical PF-ODE trajectory over the current step. Therefore, after the known schedule-dependent factors have been handled analytically, the remaining numerical problem is to approximate

$$\lambda \longmapsto \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$$

inside the weighted integral.

For $n \geq 1$, we expand this neural prediction about λ_s:

$$\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \sum_{k=0}^{n-1} \frac{(\lambda-\lambda_s)^k}{k!}\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}((\lambda-\lambda_s)^n),$$

where

$$\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) := \frac{\mathrm{d}^k}{\mathrm{d}\lambda^k}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$$

denotes the k-th *total derivative along the empirical PF-ODE trajectory*. In particular,

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \frac{\partial}{\partial\lambda}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) + J_{\mathbf{x}}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\frac{\mathrm{d}\hat{\mathbf{x}}_\lambda}{\mathrm{d}\lambda},$$

so the derivative includes both the explicit dependence on λ and the dependence through the evolving state.

Substituting the Taylor expansion into Equation (9.4.6) gives

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \alpha_t \sum_{k=0}^{n-1} \hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s)\, C_k + \mathcal{O}(h^{n+1}), \tag{9.4.7}$$

where

$$h := \lambda_t - \lambda_s, \qquad C_k := \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\frac{(\lambda-\lambda_s)^k}{k!}\,\mathrm{d}\lambda.$$

The coefficients C_k depend only on the log-SNR interval and can be evaluated analytically.

If the required trajectory derivatives were available, dropping the $\mathcal{O}(h^{n+1})$ remainder would give the corresponding formal n-th order update. DPM-Solver avoids explicitly differentiating the neural network by approximating these derivatives from additional model evaluations, as described in the next subsection.

Example: DPM-Solver-1

For $n = 1$, the exact empirical flow satisfies

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) + \mathcal{O}(h^2).$$

Dropping the $\mathcal{O}(h^2)$ remainder defines DPM-Solver-1:

$$\tilde{\mathbf{x}}_t^{\text{DPM-Solver-1}} := \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s). \tag{9.4.8}$$

This finite-step update is exactly the deterministic DDIM update, as shown in Section 9.4.4.

DPM-Solver-n with $n \geq 2$ requires evaluating the kth-derivative $\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$ for $k \leq n-1$. However, directly computing higher-order derivatives is computationally expensive in practice. Lu *et al.* (2022b) also propose efficient approximation methods for these derivatives, which will be detailed in the next subsection.

9.4.3 Implementation of DPM-Solver-n

DPM-Solver-n with $n \geq 2$. In practice, implementing a higher-order DPM-Solver-n entails the following:

- Precomputing the coefficients C_k;
- Approximating the kth derivative $\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$ for $k \leq n-1$ to circumvent the costly computation of exact higher-order derivatives—a challenge well-studied in the ODE literature (Hochbruck and Ostermann, 2005; Luan, 2021). One common strategy is finite difference approximation.

We now elaborate on the first two points.

Precomputing C_k. Let s and t denote the start and end times, respectively, and define $h := \lambda_t - \lambda_s$. The Taylor expansion of the exact empirical PF-ODE flow takes the form

$$\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) = \frac{\alpha_t}{\alpha_s}\mathbf{x}_s - \sigma_t\sum_{k=0}^{n-1} h^{k+1}\varphi_{k+1}(h)\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^{n+1}), \tag{9.4.9}$$

where, for $m \geq 1$,

$$\varphi_m(h) := \frac{e^h - \sum_{j=0}^{m-1} \frac{h^j}{j!}}{h^m}, \qquad \varphi_m(0) := \frac{1}{m!}.$$

For the first three cases,

$$\varphi_1(h) = \frac{e^h - 1}{h}, \qquad \varphi_2(h) = \frac{e^h - h - 1}{h^2}, \qquad \varphi_3(h) = \frac{e^h - \frac{h^2}{2} - h - 1}{h^3}.$$

Example: Third-Order Expansion with Exact Derivatives

For $n = 3$, Equation (9.4.9) becomes

$$\begin{aligned}\widetilde{\mathbf{\Psi}}_{s\to t}(\mathbf{x}_s) = \frac{\alpha_t}{\alpha_s}\mathbf{x}_s &- \sigma_t\left(e^h - 1\right)\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \\ &- \sigma_t\left(e^h - h - 1\right)\hat{\boldsymbol{\epsilon}}^{(1)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \\ &- \sigma_t\left(e^h - \frac{h^2}{2} - h - 1\right)\hat{\boldsymbol{\epsilon}}^{(2)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^4).\end{aligned} \tag{9.4.10}$$

Approximating $\hat{\boldsymbol{\epsilon}}^{(k)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda})$ for $k \leq n-1$. For $n \geq 2$, following the standard approach of single-step ODE solvers (Atkinson *et al.*, 2009), Lu *et al.* (2022b) use additional model evaluations within the current step to approximate the higher-order derivatives. We illustrate this idea for $n = 2$.

Let $\gamma \in (0, 1]$ specify a stage point in the log-SNR interval $[\lambda_s, \lambda_t]$. Given an estimate $\tilde{\mathbf{x}}_s$ at s, define

$$s^{\text{mid}} = t_\lambda\left(\lambda_s + \gamma h\right), \qquad h := \lambda_t - \lambda_s.$$

For $\gamma \in (0, 1)$ this is an interior point, while $\gamma = 1$ corresponds to the endpoint t. We first use a DPM-Solver-1 step to predict the state at this stage:

$$\mathbf{x}^{\text{mid}} = \frac{\alpha_{s^{\text{mid}}}}{\alpha_s}\tilde{\mathbf{x}}_s - \sigma_{s^{\text{mid}}}\left(e^{\gamma h} - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

The difference between the model predictions at the starting and stage points then provides a finite-difference approximation of the first total derivative along the trajectory:

$$\hat{\boldsymbol{\epsilon}}^{(1)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \approx \frac{\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}^{\text{mid}}, s^{\text{mid}}) - \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)}{\gamma h}.$$

Substituting this approximation into the second-order expansion gives

$$\begin{aligned}\widetilde{\mathbf{\Psi}}_{s\to t}(\tilde{\mathbf{x}}_s) = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s &- \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s) \\ &- \frac{\sigma_t}{\gamma h}\left(e^h - h - 1\right)\left[\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}^{\text{mid}}, s^{\text{mid}}) - \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)\right] + \mathcal{O}(h^3).\end{aligned} \tag{9.4.11}$$

Thus, dropping the $\mathcal{O}(h^3)$ remainder yields a second-order single-step update.

For the midpoint choice $\gamma = \frac{1}{2}$, DPM-Solver-2 admits the simpler two-stage form in Algorithm 5. This update differs from Equation (9.4.11) only by $\mathcal{O}(h^3)$ and therefore has the same second-order accuracy.

Algorithm 5 DPM-Solver-2 (with $\gamma = \frac{1}{2}$).

Input: initial value $\mathbf{x}_T$, time steps $\{t_i\}_{i=0}^{M}$, model $\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}$

$\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T$

for $i \leftarrow 1$ to M **do**

$h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$

$s_i^{\text{mid}} \leftarrow t_\lambda\left(\frac{\lambda_{t_{i-1}} + \lambda_{t_i}}{2}\right)$

$\mathbf{x}_i^{\text{mid}} \leftarrow \frac{\alpha_{s_i^{\text{mid}}}}{\alpha_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} - \sigma_{s_i^{\text{mid}}}\left(e^{h_i/2} - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})$

$\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} - \sigma_{t_i}\left(e^{h_i} - 1\right)\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i^{\text{mid}}, s_i^{\text{mid}})$

end for

return $\tilde{\mathbf{x}}_{t_M}$

Remark.

The finite difference above approximates the *total* λ-derivative of the neural prediction along the empirical PF-ODE trajectory. Under standard smoothness assumptions,

$$\frac{\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}^{\text{mid}}, s^{\text{mid}}) - \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s)}{\gamma h} = \hat{\boldsymbol{\epsilon}}^{(1)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h).$$

Here, the intermediate state is itself obtained by a first-order DPM-Solver-1 step, whose stage error is $\mathcal{O}(h^2)$; under smoothness of the model, this does not change the $\mathcal{O}(h)$ accuracy of the difference quotient.

In Equation (9.4.11), this derivative approximation is multiplied by

$$e^h - h - 1 = \mathcal{O}(h^2),$$

so its induced error is $\mathcal{O}(h^3)$. Hence, the resulting method is second-order accurate for any fixed $\gamma \in (0, 1]$.

This single-step construction uses two model evaluations: one at $(\tilde{\mathbf{x}}_s, s)$ and one at the predicted stage $(\mathbf{x}^{\text{mid}}, s^{\text{mid}})$. The choice of γ does not change the formal second-order accuracy, but it changes the stage location and the error constant. In particular, $\gamma = \frac{1}{2}$ places the second evaluation at the midpoint of the log-SNR interval, giving the commonly used midpoint version of DPM-Solver-2.

Implementation Detail: Selection of Sampling Timesteps. Since DPM-Solver constructs its local high-order approximation in log-SNR time λ, a natural choice is to place the numerical nodes uniformly in λ. For a finite terminal time $t_M > 0$, this gives

$$\lambda_{t_i} = \lambda_{t_0} + \frac{i}{M}\left(\lambda_{t_M} - \lambda_{t_0}\right), \qquad i = 0, \dots, M.$$

If $\sigma_0 = 0$, then $\lambda_0 = +\infty$, so in practice one instead uses a finite terminal log-SNR corresponding to a small positive time, with the endpoint handled by the corresponding limiting or final denoising step.

Although the derivation is carried out in λ-space, the pre-trained model and schedule (α_t, σ_t) are usually parameterized by the original diffusion time t. During sampling, each selected log-SNR node is therefore mapped back through

$$t_i = t_\lambda(\lambda_{t_i})$$

whenever the model is evaluated or the schedule coefficients are required; see, for example, Algorithm 5. Thus, λ serves as the numerical integration coordinate, while the sampler continues to evaluate the original pre-trained diffusion model at its corresponding time labels.

9.4.4 DDIM = DPM-Solver-1

The first-order DPM-Solver recovers DDIM exactly. Both methods start from an exact integral representation of the same noise-prediction PF-ODE and make the same approximation over one step:

$$\boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s).$$

The remaining schedule-dependent terms are then integrated analytically. DDIM applies this construction directly in the schedule-aligned form derived earlier, whereas DPM-Solver first rewrites the integral using the log-SNR coordinate

$$\lambda_t = \log \frac{\alpha_t}{\sigma_t}.$$

At first order, this change of numerical clock does not alter the constant approximation of the model prediction. Consequently, the two derivations produce exactly the same finite-step update.

Proposition 9.4.2: DDIM is DPM-Solver-1

For a fixed diffusion schedule (α_t, σ_t), the deterministic DDIM update in Equation (9.2.1) is identical to the DPM-Solver-1 update in Equation (9.4.8).

Proof for Proposition.

By the definition of λ,

$$\frac{\sigma_s}{\alpha_s} = e^{-\lambda_s}, \qquad \frac{\sigma_t}{\alpha_t} = e^{-\lambda_t}.$$

Let

$$h = \lambda_t - \lambda_s.$$

Then

$$e^h = \frac{\alpha_t/\sigma_t}{\alpha_s/\sigma_s} = \frac{\alpha_t \sigma_s}{\sigma_t \alpha_s},$$

and therefore

$$-\sigma_t(e^h - 1) = \sigma_t - \frac{\alpha_t}{\alpha_s}\sigma_s = \alpha_t \left(\frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s}\right).$$

Substituting this identity into the DPM-Solver-1 update gives

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \alpha_t \left(\frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s}\right) \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_s, s),$$

which is exactly the deterministic DDIM update in Equation (9.2.1). ∎

Thus, DPM-Solver-1 and DDIM are two derivations of the same first-order finite-step update. The log-SNR coordinate becomes consequential beyond first order: higher-order DPM-Solvers use it to approximate how the noise prediction changes along the trajectory, rather than treating that prediction as constant over the whole step.

9.4.5 Relation of DPM-Solver to Runge-Kutta-Type Solvers

DPM-Solver-1 is exactly the DDIM update (Section 9.4.4), so its first-order behavior is already understood. The higher-order question is how DPM-Solver improves this one-evaluation update.

The connection is to Runge-Kutta (RK) methods. Euler uses only the vector field at the beginning of a step, whereas RK methods obtain additional information through *stages*, i.e., evaluations at states and times generated within the current step; see Section A.1.5. Higher-order DPM-Solvers follow the same principle: instead of holding one neural prediction fixed over the whole step, they introduce additional within-step model evaluations to capture its variation. The diffusion-specific difference is that the known schedule-dependent factors are retained analytically, while the staged approximation is applied only to the learned prediction.

DPM-Solver-2 versus Plain Midpoint RK2. We first compare DPM-Solver-2 with classical midpoint RK2 applied directly to the complete noise-prediction PF-ODE in log-SNR time from Equation (9.4.4). For brevity, denote its complete

right-hand side by $\mathbf{F}(\hat{\mathbf{x}}_\lambda, \lambda)$.

For the midpoint choice $\gamma = \frac{1}{2}$, let

$$\lambda_{\text{mid}} := \lambda_{s^{\text{mid}}} = \lambda_s + \frac{h}{2}, \qquad \alpha_{\text{mid}} := \alpha_{s^{\text{mid}}}, \qquad \sigma_{\text{mid}} := \sigma_{s^{\text{mid}}},$$

and abbreviate

$$\hat{\boldsymbol{\epsilon}}_s := \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, \lambda_s) = \boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_s, s).$$

Classical midpoint RK2 gives

$$\mathbf{x}_{\text{mid}}^{\text{RK}} = \tilde{\mathbf{x}}_s + \frac{h}{2}\mathbf{F}(\tilde{\mathbf{x}}_s, \lambda_s),$$

followed by

$$\tilde{\mathbf{x}}_t^{\text{RK2}} = \tilde{\mathbf{x}}_s + h\,\mathbf{F}\left(\mathbf{x}_{\text{mid}}^{\text{RK}}, \lambda_{\text{mid}}\right).$$

Thus, plain RK2 applies its two-stage construction to the complete PF-ODE velocity, including both the known schedule-dependent terms and the neural prediction.

By contrast, midpoint DPM-Solver-2 in Algorithm 5 first constructs the stage

$$\mathbf{x}_{\text{mid}}^{\text{DPM2}} = \frac{\alpha_{\text{mid}}}{\alpha_s}\tilde{\mathbf{x}}_s - \sigma_{\text{mid}}\left(e^{h/2} - 1\right)\hat{\boldsymbol{\epsilon}}_s.$$

Denoting the corresponding midpoint prediction by

$$\hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{DPM2}} := \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}\left(\mathbf{x}_{\text{mid}}^{\text{DPM2}}, \lambda_{\text{mid}}\right),$$

the full update is

$$\tilde{\mathbf{x}}_t^{\text{DPM2}} = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \sigma_t\left(e^h - 1\right)\hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{DPM2}}.$$

The structural difference is immediate. Plain midpoint RK2 discretizes the complete PF-ODE velocity, whereas DPM-Solver-2 retains the known diffusion-dependent evolution analytically and uses its two model evaluations only to approximate the variation of the noise prediction.

Under standard smoothness assumptions, both are second-order methods for the same empirical PF-ODE in λ-time. Their intermediate states generally differ by $\mathcal{O}(h^2)$, and their complete one-step updates therefore satisfy

$$\tilde{\mathbf{x}}_t^{\text{DPM2}} - \tilde{\mathbf{x}}_t^{\text{RK2}} = \mathcal{O}(h^3).$$

Thus, DPM-Solver-2 follows the same two-stage midpoint principle as classical RK2, while exploiting the known diffusion structure before applying the numerical approximation.

DPM-Solver-2 versus Exponential Midpoint RK2. A closer comparison keeps the semilinear decomposition in Equation (9.4.4) explicit and treats its known linear part analytically. The corresponding *exponential midpoint RK2* construction first takes an exponential-Euler stage to the midpoint:

$$\mathbf{x}_{\text{mid}}^{\text{ExpRK}} = \frac{\alpha_{\text{mid}}}{\alpha_s}\tilde{\mathbf{x}}_s - \alpha_{\text{mid}}\sigma_s \left[\int_{\lambda_s}^{\lambda_{\text{mid}}} \frac{1}{\alpha_\lambda}\,\mathrm{d}\lambda\right] \hat{\boldsymbol{\epsilon}}_s.$$

After evaluating

$$\hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{ExpRK}} := \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}\left(\mathbf{x}_{\text{mid}}^{\text{ExpRK}}, \lambda_{\text{mid}}\right),$$

the full exponential-midpoint update is

$$\tilde{\mathbf{x}}_t^{\text{ExpRK2}} = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s - \alpha_t\sigma_{\text{mid}} \left[\int_{\lambda_s}^{\lambda_t} \frac{1}{\alpha_\lambda}\,\mathrm{d}\lambda\right] \hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{ExpRK}}.$$

When the linear coefficient is constant, these integral coefficients reduce to the usual φ_1 coefficients of explicit exponential midpoint RK2.

DPM-Solver-2 uses the same midpoint staging principle, but exploits the finer noise-prediction factorization

$$-\sigma_\lambda \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda).$$

Since σ_λ is also known from the diffusion schedule, DPM-Solver-2 retains it inside the integral and stages only the neural prediction. Its midpoint and full-step coefficients are therefore

$$\alpha_{\text{mid}} \int_{\lambda_s}^{\lambda_{\text{mid}}} \frac{\sigma_\lambda}{\alpha_\lambda}\,\mathrm{d}\lambda = \sigma_{\text{mid}}\left(e^{h/2} - 1\right), \quad \text{and} \quad \alpha_t \int_{\lambda_s}^{\lambda_t} \frac{\sigma_\lambda}{\alpha_\lambda}\,\mathrm{d}\lambda = \sigma_t\left(e^{h} - 1\right),$$

respectively, recovering the update in Algorithm 5.

Hence, the three constructions differ in what is approximated by the RK stages:

Plain RK2: stage the complete PF-ODE velocity,

ExpRK2: propagate the linear part exactly;
stage the coefficient-weighted residual $\sigma_\lambda \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}$,

DPM-Solver-2: also retain σ_λ analytically and stage only $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\phi}^\times}$.

The exponential-RK and DPM midpoint stages differ by

$$\mathbf{x}_{\text{mid}}^{\text{DPM2}} - \mathbf{x}_{\text{mid}}^{\text{ExpRK}} = -\alpha_{\text{mid}} \left[\int_{\lambda_s}^{\lambda_{\text{mid}}} \frac{\sigma_\lambda - \sigma_s}{\alpha_\lambda}\,\mathrm{d}\lambda\right] \hat{\boldsymbol{\epsilon}}_s = \mathcal{O}(h^2).$$

For the complete step, a Taylor expansion about λ_{mid} gives

$$\int_{\lambda_s}^{\lambda_t} \frac{\sigma_\lambda - \sigma_{\text{mid}}}{\alpha_\lambda}\,\mathrm{d}\lambda = \mathcal{O}(h^3),$$

while smoothness of the neural prediction gives

$$\hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{DPM2}} - \hat{\boldsymbol{\epsilon}}_{\text{mid}}^{\text{ExpRK}} = \mathcal{O}(h^2).$$

Consequently,

$$\tilde{\mathbf{x}}_t^{\text{DPM2}} - \tilde{\mathbf{x}}_t^{\text{ExpRK2}} = \mathcal{O}(h^3).$$

Thus, DPM-Solver-2 and exponential midpoint RK2 agree through second order, but DPM-Solver-2 retains one additional analytically known quantity, σ_λ, inside the weighted integral.

Remark.

> The discussion above concerns the noise-prediction formulation in which DPM-Solver is derived. If a pre-trained model uses another prediction parameterization, its output may be converted pointwise to noise prediction through Equation (6.3.1) at each stage. Just like the DDIM-case in Section 9.2.2, this should not be confused with directly applying a classical RK method to the raw prediction field: pointwise conversion and finite-step discretization are separate operations.

DPM-Solver and Runge–Kutta Methods

DPM-Solver combines the within-step staging principle of Runge–Kutta methods with the analytic treatment of known diffusion dynamics. DPM-Solver-1 is exactly DDIM, while higher-order DPM-Solvers use additional within-step evaluations to capture the variation of the neural prediction. The connection to classical RK methods is structural rather than an equality of finite-step formulas: DPM-Solver retains the known diffusion-dependent weights analytically before approximating the learned prediction.

9.5 DPM-Solver++

9.5.1 From DPM-Solver to DPM-Solver++ for Guided Sampling

DPM-Solver++ (Lu *et al.*, 2022c) was developed to improve high-order sampling under strong guidance. In this regime, two practical difficulties arise. First, large guidance scales can amplify the model output and its derivatives, narrowing the convergence region of high-order solvers and requiring smaller numerical steps. Second, the guided prediction may drive the final solution outside the range represented by the training data, leading to the train–test mismatch discussed in Chapter 8.

DPM-Solver++ addresses these issues in two complementary ways. It reformulates the diffusion ODE using data prediction, for which data-space corrections such as dynamic thresholding can be applied, and it develops a multistep variant that reuses previous model evaluations to obtain smaller effective step sizes under a fixed NFE budget. We first derive the data-prediction formulation and then introduce its single-step and multistep second-order solvers.

9.5.2 DPM-Solver++'s Methodology

DPM-Solver++ keeps the log-SNR coordinate λ introduced for DPM-Solver, but expresses the empirical PF-ODE through data prediction rather than noise prediction. Recall from Equation (6.3.1) that

$$\boldsymbol{\epsilon}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t) = \frac{\mathbf{x}_t - \alpha_t \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_t, t)}{\sigma_t}.$$

Substituting this relation into the noise-prediction exact solution in Equation (9.4.5) gives the equivalent data-prediction form

$$\widetilde{\boldsymbol{\Psi}}_{s\to t}(\mathbf{x}_s) = \frac{\sigma_t}{\sigma_s}\mathbf{x}_s + \sigma_t \int_{\lambda_s}^{\lambda_t} e^{\lambda} \hat{\mathbf{x}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\, \mathrm{d}\lambda, \tag{9.5.1}$$

where, following Equation (9.4.3), we define the λ-parameterized data-prediction field by

$$\hat{\mathbf{x}}_{\boldsymbol{\phi}^\times}(\mathbf{x}, \lambda) := \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}, t_\lambda(\lambda)).$$

Hence, along the empirical PF-ODE trajectory,

$$\hat{\mathbf{x}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_{t_\lambda(\lambda)}, t_\lambda(\lambda)).$$

Thus, the structure is exactly dual to DPM-Solver: the known schedule-dependent evolution is handled analytically, while the only model-dependent quantity inside the remaining integral is now the data prediction. DPM-Solver++ develops two second-order realizations of this exact form:

- **Single-Step Solver, DPM-Solver++(2S).** Use an additional model evaluation at an intermediate stage within the current step, analogous to the staged construction of DPM-Solver-2.
- **Multistep Solver, DPM-Solver++(2M).** Reuse data predictions from the previous two completed steps. After the warm start, only one new model evaluation is required per step.

Remark.

> Without thresholding, converting a model output to data prediction is a pointwise algebraic re-expression of the same empirical PF-ODE. The numerical method can nevertheless change because the finite-step approximation is now applied to the data prediction rather than to the noise prediction. If thresholding is used, it is an additional modification of the model output and should be viewed separately from the numerical-order analysis below.

9.5.3 DPM-Solver++ Single-Step by Taylor Expansion

Following the DPM-Solver construction in Section 9.4.2, DPM-Solver++ approximates the data prediction along the empirical trajectory by a local Taylor expansion in log-SNR time. For $k \geq 0$, denote its k-th total derivative along the empirical PF-ODE trajectory by

$$\hat{\mathbf{x}}^{(k)}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) := \frac{\mathrm{d}^k}{\mathrm{d}\lambda^k}\hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda).$$

For one step from t_{i-1} to t_i, let $h_i := \lambda_{t_i} - \lambda_{t_{i-1}} > 0$. A Taylor expansion about $\lambda_{t_{i-1}}$ gives

$$\hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \sum_{k=0}^{n-1} \frac{(\lambda - \lambda_{t_{i-1}})^k}{k!}\hat{\mathbf{x}}^{(k)}_{\phi^\times}(\hat{\mathbf{x}}_{\lambda_{t_{i-1}}}, \lambda_{t_{i-1}}) + \mathcal{O}((\lambda - \lambda_{t_{i-1}})^n).$$

Substituting this expansion into Equation (9.5.1) yields

$$\widetilde{\mathbf{\Psi}}_{t_{i-1}\to t_i}(\tilde{\mathbf{x}}_{t_{i-1}}) = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\sum_{k=0}^{n-1} h_i^{k+1}\varphi_{k+1}(-h_i)\hat{\mathbf{x}}^{(k)}_{\phi^\times}(\hat{\mathbf{x}}_{\lambda_{t_{i-1}}}, \lambda_{t_{i-1}}) + \mathcal{O}(h_i^{n+1}), \tag{9.5.2}$$

where φ_m is defined in Equation (9.4.9). Here we used

$$\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^\lambda \frac{(\lambda - \lambda_{t_{i-1}})^k}{k!}\,\mathrm{d}\lambda = e^{\lambda_{t_i}} h_i^{k+1}\varphi_{k+1}(-h_i),$$

together with $\sigma_{t_i} e^{\lambda_{t_i}} = \alpha_{t_i}$.

For $n = 1$, dropping the $\mathcal{O}(h_i^2)$ remainder gives

$$\tilde{\mathbf{x}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_{i-1}, \qquad \mathbf{D}_{i-1} := \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}),$$

which is exactly the deterministic DDIM update in data-prediction form.

For $n = 2$, the first total derivative is approximated using an additional within-step stage. For the midpoint choice, define

$$s_i^{\mathrm{mid}} := t_\lambda\left(\lambda_{t_{i-1}} + \frac{h_i}{2}\right).$$

A first-order DPM-Solver++ step predicts the midpoint state

$$\mathbf{u}_i = \frac{\sigma_{s_i^{\mathrm{mid}}}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{s_i^{\mathrm{mid}}}\left(1 - e^{-h_i/2}\right)\mathbf{D}_{i-1},$$

at which we evaluate

$$\mathbf{D}_i^{\mathrm{mid}} := \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{u}_i, s_i^{\mathrm{mid}}).$$

The midpoint DPM-Solver++(2S) update then takes the particularly simple form

$$\tilde{\mathbf{x}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_i^{\mathrm{mid}}.$$

As with midpoint DPM-Solver-2, this simplified two-stage update differs from the direct second-order Taylor/finite-difference construction only by $\mathcal{O}(h_i^3)$ and therefore retains second-order accuracy.

This yields the single-step DPM-Solver++ construction. When $n = 1$, it reduces exactly to the deterministic DDIM update in data-prediction form. When $n = 2$, approximating the first total λ-derivative using an additional within-step model evaluation gives *DPM-Solver++(2S)*, the data-prediction counterpart of DPM-Solver-2 in Algorithm 5. Its midpoint implementation is summarized in Algorithm 6.

Algorithm 6 DPM-Solver++(2S): a midpoint special case.

Input: initial value $\mathbf{x}_T$, time steps $\{t_i\}_{i=0}^{M}$, data-prediction model $\mathbf{x}_{\boldsymbol{\phi}\times}$

$\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T$
$\lambda_{t_i} \leftarrow \log(\alpha_{t_i}/\sigma_{t_i})$ for $i = 0, \dots, M$
$\mathbf{D}_0 \leftarrow \mathbf{x}_{\boldsymbol{\phi}\times}(\tilde{\mathbf{x}}_{t_0}, t_0)$ ▷ cache start prediction
for $i \leftarrow 1$ **to** M **do**
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $s_i^{\mathrm{mid}} \leftarrow t_\lambda\left(\frac{\lambda_{t_{i-1}}+\lambda_{t_i}}{2}\right)$
  $\mathbf{u}_i \leftarrow \frac{\sigma_{s_i^{\mathrm{mid}}}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{s_i^{\mathrm{mid}}}\left(1 - e^{-h_i/2}\right)\mathbf{D}_{i-1}$ ▷ predict midpoint stage
  $\mathbf{D}_i^{\mathrm{mid}} \leftarrow \mathbf{x}_{\boldsymbol{\phi}\times}(\mathbf{u}_i, s_i^{\mathrm{mid}})$ ▷ evaluate at midpoint
  $\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_i^{\mathrm{mid}}$
  if $i < M$ **then**
    $\mathbf{D}_i \leftarrow \mathbf{x}_{\boldsymbol{\phi}\times}(\tilde{\mathbf{x}}_{t_i}, t_i)$ ▷ cache for next step
  end if
end for
return $\tilde{\mathbf{x}}_{t_M}$

9.5.4 DPM-Solver++ Multistep by Recycling History

DPM-Solver++(2S) obtains second-order information by introducing an additional model evaluation *within the current step*. DPM-Solver++(2M) takes a different route: it reuses model predictions from previous completed steps to estimate the same local variation. After initialization, this requires only one new model evaluation per step, allowing more, and hence smaller, numerical steps under the same NFE budget. This is particularly useful under strong guidance, where large steps can reduce numerical stability.

The idea is simple. Recall that the second-order expansion in Equation (9.5.2) requires both the current data prediction and its first total derivative with respect to λ. The single-step method estimates this derivative using a new within-step stage; the multistep method instead estimates it from the recent history of model predictions.

For the first step, no such history is available, so we initialize with the first-order data-prediction update

$$\tilde{\mathbf{x}}_{t_1} = \frac{\sigma_{t_1}}{\sigma_{t_0}}\tilde{\mathbf{x}}_{t_0} + \alpha_{t_1}\left(1 - e^{-h_1}\right)\mathbf{D}_0, \qquad \mathbf{D}_0 := \mathbf{x}_{\boldsymbol{\phi}\times}(\tilde{\mathbf{x}}_{t_0}, t_0).$$

This is exactly the deterministic DDIM update in data-prediction form.

Once two completed-step predictions are available, for $i \geq 2$ define

$$\mathbf{D}_{i-1} := \mathbf{x}_{\boldsymbol{\phi}\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}), \qquad \mathbf{D}_{i-2} := \mathbf{x}_{\boldsymbol{\phi}\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2}).$$

Keeping the first two terms of Equation (9.5.2) gives

$$\widetilde{\mathbf{\Psi}}_{t_{i-1}\to t_i}(\tilde{\mathbf{x}}_{t_{i-1}}) = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_{i-1} + \alpha_{t_i}\left(h_i - 1 + e^{-h_i}\right)\hat{\mathbf{x}}^{(1)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_{t_{i-1}}}, \lambda_{t_{i-1}}) + \mathcal{O}(h_i^3).$$

Thus, the only additional quantity needed for a second-order update is the first total λ-derivative of the data prediction.

Instead of evaluating the model at a new intermediate stage, DPM-Solver++(2M) estimates this derivative from the previous completed step:

$$\hat{\mathbf{x}}^{(1)}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_{\lambda_{t_{i-1}}}, \lambda_{t_{i-1}}) \approx \frac{\mathbf{D}_{i-1} - \mathbf{D}_{i-2}}{h_{i-1}}.$$

Under standard smoothness assumptions, this backward difference has error $\mathcal{O}(h_{i-1})$. Substituting it into the second-order expansion and collecting the two prediction terms gives

$$\tilde{\mathbf{x}}^{\text{poly}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}^{\text{poly}}_i,$$

where

$$\mathbf{D}^{\text{poly}}_i := \mathbf{D}_{i-1} + \frac{\beta(h_i)}{h_{i-1}}\left(\mathbf{D}_{i-1} - \mathbf{D}_{i-2}\right), \qquad \beta(h) := \frac{h - 1 + e^{-h}}{1 - e^{-h}}.$$

When consecutive step ratios remain bounded, this weighted multistep update has one-step local defect $\mathcal{O}(h_i^3)$.

The Practical DPM-Solver++(2M) Update. The expression above is already a valid second-order multistep construction. DPM-Solver++(2M) uses a slightly simpler coefficient that preserves the same order. Since

$$\beta(h) = \frac{h}{2} + \frac{h^2}{12} + \mathcal{O}(h^3),$$

define the step ratio

$$r_i := \frac{h_i}{h_{i-1}}$$

and replace $\mathbf{D}^{\text{poly}}_i$ by

$$\mathbf{D}^{\text{sim}}_i := \mathbf{D}_{i-1} + \frac{r_i}{2}\left(\mathbf{D}_{i-1} - \mathbf{D}_{i-2}\right).$$

Equivalently,

$$\mathbf{D}^{\text{sim}}_i = \left(1 + \frac{r_i}{2}\right)\mathbf{D}_{i-1} - \frac{r_i}{2}\mathbf{D}_{i-2}.$$

The interpretation is straightforward: the two most recent model predictions tell us the local direction in which the data prediction is changing, and $\mathbf{D}^{\text{sim}}_i$ extrapolates that trend into the current step.

Under the same smoothness and bounded-step-ratio assumptions,

$$\mathbf{D}_i^{\text{poly}} - \mathbf{D}_i^{\text{sim}} = \mathcal{O}(h_i^2).$$

Since

$$\alpha_{t_i}\left(1 - e^{-h_i}\right) = \mathcal{O}(h_i),$$

this simplification changes the state update only by $\mathcal{O}(h_i^3)$. Hence, it preserves second-order accuracy, giving the practical DPM-Solver++(2M) update

$$\tilde{\mathbf{x}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_i^{\text{sim}}.$$

After computing $\tilde{\mathbf{x}}_{t_i}$, the model is evaluated once at the new state,

$$\mathbf{D}_i = \mathbf{x}_{\phi^\times}(\tilde{\mathbf{x}}_{t_i}, t_i),$$

and this prediction is cached for the next step. Thus, after the warm start, each DPM-Solver++(2M) step requires only one new model evaluation.

Remark.

> For uniform log-SNR steps, $r_i = 1$, so
>
> $$\mathbf{D}_i^{\text{sim}} = \frac{3}{2}\mathbf{D}_{i-1} - \frac{1}{2}\mathbf{D}_{i-2}.$$
>
> The coefficients $(\frac{3}{2}, -\frac{1}{2})$ are the familiar AB2 extrapolation weights for the *model prediction*. The complete DPM-Solver++(2M) update is nevertheless not plain AB2: the known diffusion-dependent factor $\alpha_{t_i}(1 - e^{-h_i})$ is retained analytically.

The complete warm-start and two-step procedure is summarized in Algorithm 7. The key distinction from DPM-Solver++(2S) is where the second-order information comes from: 2S obtains it from an additional stage within the current step, whereas 2M recycles model evaluations from previous completed steps.

Algorithm 7 DPM-Solver++(2M).

Input: initial value $\mathbf{x}_T$, time steps $\{t_i\}_{i=0}^M$, data-prediction model $\mathbf{x}_{\boldsymbol{\phi}^\times}$

$\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T$
$\lambda_{t_i} \leftarrow \log(\alpha_{t_i}/\sigma_{t_i})$ for $i = 0, \dots, M$
$\mathbf{D}_0 \leftarrow \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_0}, t_0)$ ▷ cache start prediction
Case I. Warm start ($i = 1$)
$h_1 \leftarrow \lambda_{t_1} - \lambda_{t_0}$
$\tilde{\mathbf{x}}_{t_1} \leftarrow \frac{\sigma_{t_1}}{\sigma_{t_0}}\tilde{\mathbf{x}}_{t_0} + \alpha_{t_1}\left(1 - e^{-h_1}\right)\mathbf{D}_0$ ▷ DDIM in data prediction
if $M > 1$ **then**
  $\mathbf{D}_1 \leftarrow \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_1}, t_1)$ ▷ cache for multistep update
end if
Case II. Two-step multistep update ($i \geq 2$)
for $i \leftarrow 2$ **to** M **do**
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $r_i \leftarrow h_i/h_{i-1}$ ▷ step ratio
  $\mathbf{D}_i^{\text{sim}} \leftarrow \left(1 + \frac{1}{2}r_i\right)\mathbf{D}_{i-1} - \frac{1}{2}r_i\mathbf{D}_{i-2}$
  $\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_i^{\text{sim}}$
  if $i < M$ **then**
    $\mathbf{D}_i \leftarrow \mathbf{x}_{\boldsymbol{\phi}^\times}(\tilde{\mathbf{x}}_{t_i}, t_i)$ ▷ one new model call; cache for next step
  end if
end for
return $\tilde{\mathbf{x}}_{t_M}$

9.5.5 Relation of DPM-Solver++(2M) to DEIS

DPM-Solver++(2M) and DEIS use the same basic multistep idea: reuse model predictions from previous completed steps to approximate how the neural prediction changes over the next interval, while keeping the known diffusion-dependent weights analytic. The difference is the representation in which this idea is applied. DEIS was formulated earlier using noise prediction in the original diffusion time, whereas DPM-Solver++(2M) uses data prediction in log-SNR time.

To expose the connection directly, we can apply the DEIS polynomial extrapolation principle to the exact data-prediction integral underlying DPM-Solver++. Recall that

$$\widetilde{\boldsymbol{\Psi}}_{t_{i-1}\to t_i}(\tilde{\mathbf{x}}_{t_{i-1}}) = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \sigma_{t_i}\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^{\lambda}\hat{\mathbf{x}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda)\,\mathrm{d}\lambda.$$

The known weight e^λ is already handled analytically. The only quantity that must be approximated over the step is therefore the data prediction

$$\lambda \longmapsto \hat{\mathbf{x}}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_\lambda, \lambda).$$

Using the two most recent cached predictions $\mathbf{D}_{i-2}$ and $\mathbf{D}_{i-1}$, the simplest nonconstant approximation is the unique affine polynomial through these two anchors. In Lagrange form,

$$P_1(\lambda) = \frac{\lambda - \lambda_{t_{i-2}}}{h_{i-1}}\mathbf{D}_{i-1} - \frac{\lambda - \lambda_{t_{i-1}}}{h_{i-1}}\mathbf{D}_{i-2}.$$

The same polynomial can be written more transparently around the most recent anchor as

$$P_1(\lambda) = \mathbf{D}_{i-1} + \frac{\lambda - \lambda_{t_{i-1}}}{h_{i-1}}\left(\mathbf{D}_{i-1} - \mathbf{D}_{i-2}\right).$$

The first expression is the Lagrange form, while the second is the backward-Newton form of the *same* affine polynomial. Thus, Lagrange interpolation and backward differences are not different numerical approximations here; they are simply two representations of the same two-anchor extrapolant.

Applying this extrapolant inside the DPM-Solver++ integral gives

$$\tilde{\mathbf{x}}^{\mathrm{poly}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \sigma_{t_i}\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^{\lambda} P_1(\lambda)\,\mathrm{d}\lambda.$$

Because the weight e^{λ} is known, the two required moments can be evaluated exactly:

$$\sigma_{t_i}\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^{\lambda}\,\mathrm{d}\lambda = \alpha_{t_i}\left(1 - e^{-h_i}\right),$$

and

$$\sigma_{t_i}\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^{\lambda}\left(\lambda - \lambda_{t_{i-1}}\right)\mathrm{d}\lambda = \alpha_{t_i}\left(h_i - 1 + e^{-h_i}\right).$$

Substituting these coefficients yields

$$\begin{aligned}\tilde{\mathbf{x}}^{\mathrm{poly}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} &+ \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}_{i-1} \\ &+ \alpha_{t_i}\left(h_i - 1 + e^{-h_i}\right)\frac{\mathbf{D}_{i-1} - \mathbf{D}_{i-2}}{h_{i-1}}.\end{aligned}$$

Equivalently,

$$\tilde{\mathbf{x}}^{\mathrm{poly}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}\left(1 - e^{-h_i}\right)\mathbf{D}^{\mathrm{poly}}_i,$$

where $\mathbf{D}^{\mathrm{poly}}_i$ is defined in Section 9.5.4.

This shows the precise connection to DEIS. Before the practical coefficient simplification, the DPM-Solver++ second-order multistep update is exactly what one obtains by applying the degree-one DEIS extrapolation principle to the data-prediction/log-SNR integral. In this sense, it is the data-prediction/log-SNR analogue of AB-DEIS-1.

The practical DPM-Solver++(2M) update further simplifies this coefficient using the leading-order expansion

$$\beta(h_i) = \frac{h_i}{2} + \mathcal{O}(h_i^2),$$

which gives

$$\frac{\beta(h_i)}{h_{i-1}} = \frac{h_i}{2h_{i-1}} + \mathcal{O}\left(\frac{h_i^2}{h_{i-1}}\right).$$

Accordingly, DPM-Solver++(2M) replaces $\beta(h_i)/h_{i-1}$ by $h_i/(2h_{i-1})$ in the effective prediction. Since

$$\mathbf{D}_{i-1} - \mathbf{D}_{i-2} = \mathcal{O}(h_{i-1}),$$

this changes the effective prediction by

$$\mathcal{O}\left(\frac{h_i^2}{h_{i-1}}\right)\mathcal{O}(h_{i-1}) = \mathcal{O}(h_i^2).$$

The coefficient multiplying the effective prediction in the state update is $\alpha_{t_i}(1 - e^{-h_i}) = \mathcal{O}(h_i)$, so the resulting change in the state update is only $\mathcal{O}(h_i^3)$. Therefore, DPM-Solver++(2M) and the degree-one weighted-polynomial update are generally different finite-step formulas, but they agree through second order.

This also highlights why the connection to DEIS is specific to DPM-Solver++(2M): 2M obtains its additional second-order information by extrapolating previous completed-step predictions, whereas DPM-Solver++(2S) obtains it from a new intermediate stage within the current step.

The complete DPM-Solver++(2M) procedure, including its first-order warm start and the caching of one new data prediction after each step, is summarized in Algorithm 7.

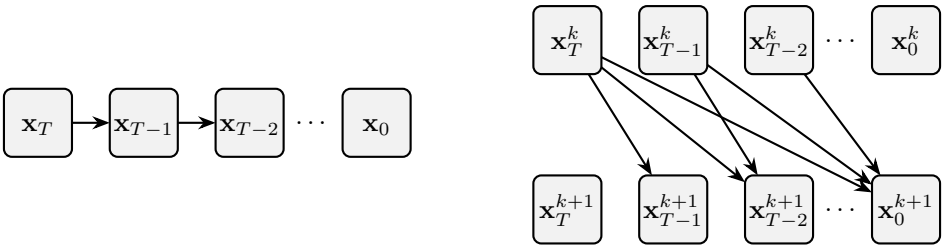
9.5.6 Closing Remark

Although our main discussion has focused on DPM-Solver and DPM-Solver++, it is useful to briefly place them in a broader context. DPM-Solver is derived in the noise-prediction formulation, whereas DPM-Solver++ reformulates the solver using data prediction, which can behave more favorably under strong CFG. DPM-Solver-v3 (Zheng *et al.*, 2023) takes this idea further by introducing a more general model representation whose coefficients are chosen using precomputed model statistics to reduce local discretization error. In this sense, it makes the representation itself part of the numerical solver design.

Another related direction is the predictor–corrector framework. In the Score SDE framework (see Chapter 4), a *predictor* advances the sample using a numerical step of the reverse-time dynamics, while a *corrector*, typically a Langevin-type update at the same noise level, further refines the sample using the score (see

Section 3.4). This predict–then–refine pattern also has a classical numerical analogue. For example, Heun's method first predicts an endpoint with Euler and then improves the numerical update using an averaged slope. Methods such as UniPC (Zhao *et al.*, 2023) develop predictor–corrector constructions specifically for diffusion sampling. We do not pursue these extensions here, but they illustrate the same broader theme of this chapter: effective diffusion solvers combine classical numerical ideas with structure specific to the diffusion PF-ODE.

9.6 (Optional) ParaDiGMs



(a) Sequential sampling by time-stepping estimation in generation process.

(b) Picard iterations with skip dependencies.

Figure 9.3: Comparisons of two computation graphs. Left: conventional time-stepping ODE solving, where the solution is propagated sequentially across time. Right: Picard iteration, which enables parallel computation by updating all time nodes simultaneously using the results from the previous iteration, thereby avoiding the strictly sequential nature of time-stepping.

Source: Adapted from Shih et al. *(2023).*

9.6.1 From Time-Stepping to Time-Parallel Solver

In the previous sections, we focused on the time–stepping approach, which estimates the trajectory by evolving from the prior time T toward an arbitrary $t \in [0, T]$. Let

$$\mathbf{v}_{\phi^\times}(\mathbf{x}, t) := \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \mathbf{s}_{\phi^\times}(\mathbf{x}, t)$$

denote the empirical PF–ODE drift from a pre-trained diffusion model. The exact evolution from T to any intermediate time t is:

$$\widetilde{\mathbf{\Psi}}_{T \to t}(\mathbf{x}(T)) = \mathbf{x}(T) + \int_T^t \mathbf{v}_{\phi^\times}(\mathbf{x}(\tau), \tau)\, \mathrm{d}\tau, \quad \mathbf{x}(T) \sim p_{\text{prior}}. \tag{9.6.1}$$

Time–stepping schemes approximate this integral using discrete updates based on past timesteps.

In this section, we turn to the *time–parallel* approach, exemplified by *ParaDiGMS*, which builds on classical Picard iteration to enable parallel integration across time. The key idea behind ParaDiGMS is to *trade computational resources for faster simulation.*

9.6.2 Methodology of ParaDiGMS

From Trajectories to Picard Iteration as a Fixed-Point Update. Fix the initial state of reverse-time sampling,

$$\mathbf{x}_T \sim p_{\text{prior}}.$$

The integral expression in Equation (9.6.1) can be viewed as a map that takes an entire candidate trajectory and produces an updated one. Formally, given any path $\{\mathbf{y}(\tau)\}_{\tau\in[0,T]}$, define

$$(\mathcal{L}[\mathbf{y}(\cdot)])(t) = \mathbf{x}_T + \int_T^t \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{y}(\tau),\tau)\,\mathrm{d}\tau, \qquad t\in[0,T].$$

That is, $\mathcal{L}$ keeps the initial state $\mathbf{x}_T$ fixed and updates the rest of the trajectory using the velocity evaluated along the candidate path.

A solution trajectory $\mathbf{x}^*(\cdot)$ of the empirical PF-ODE with $\mathbf{x}^*(T) = \mathbf{x}_T$ is precisely a *fixed point* of this map:

$$\mathbf{x}^*(t) = \mathcal{L}[\mathbf{x}^*(\cdot)](t) = \mathbf{x}_T + \int_T^t \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}^*(\tau),\tau)\,\mathrm{d}\tau.$$

Thus, instead of constructing the trajectory strictly one time step after another, we may search for an entire trajectory that is consistent with this integral equation.

Starting from an initial guess $\mathbf{x}^{(0)}(\cdot)$ satisfying $\mathbf{x}^{(0)}(T) = \mathbf{x}_T$, for example the constant path $\mathbf{x}^{(0)}(t) \equiv \mathbf{x}_T$, Picard iteration repeatedly applies this map:

$$\mathbf{x}^{(k+1)}(t) = \mathcal{L}[\mathbf{x}^{(k)}(\cdot)](t) = \mathbf{x}_T + \int_T^t \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}^{(k)}(\tau),\tau)\,\mathrm{d}\tau, \tag{9.6.2}$$

with $k = 0, 1, 2, \ldots$. Each iteration evaluates the velocity along the previous trajectory and uses these evaluations to update the trajectory at all time points.

Discrete Picard on a T to 0 Grid. To turn Equation (9.6.2) into a practical algorithm, we place a uniform, decreasing grid on $[0,T]$ by choosing a step count M, setting $\Delta t := T/M$, and defining

$$t_j := T - j\Delta t, \quad j = 0, 1, \ldots, M,$$

so $t_0 = T$ and $t_M = 0$. Denote sampled iterates by

$$\mathbf{x}_j^{(k)} := \mathbf{x}^{(k)}(t_j).$$

Because the grid runs reversely in time, the integral from T to t_j has negative orientation. Approximating it by left endpoints on the partition $\{[t_{i+1}, t_i]\}_{i=0}^{j-1}$ gives

$$\int_T^{t_j} \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}^{(k)}(\tau),\tau)\,\mathrm{d}\tau \approx -\Delta t \sum_{i=0}^{j-1} \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i^{(k)}, t_i),$$

since each small integral over $[t_{i+1}, t_i]$ equals $-\int_{t_i}^{t_{i+1}} \cdot\,\mathrm{d}\tau$. Substituting this approximation into Equation (9.6.2) yields the discrete Picard update

$$\mathbf{x}_j^{(k+1)} = \mathbf{x}_T - \Delta t \underbrace{\sum_{i=0}^{j-1} \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_i^{(k)}, t_i)}_{\substack{\text{cumulative sum}\\ \text{of drifts}}}, \qquad j = 1, \ldots, M. \tag{9.6.3}$$

This scheme is simple and parallel-friendly: each drift evaluation $\mathbf{v}_{\phi^\times}(\mathbf{x}_i^{(k)}, t_i)$ depends only on the previous iterate at the same time node t_i, so all $i = 0, \ldots, j-1$ evaluations can be computed independently across the grid. The integral is then recovered by a cumulative sum, performed either serially or via a parallel *prefix-sum (scan/sliding windows)*.

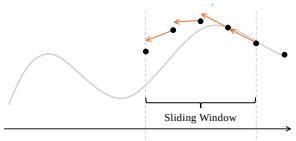


Figure 9.4: Compute the drift of $\mathbf{x}_{\ell:\ell+p}^{(k)}$ on a batch window of size $p = 4$, in parallel

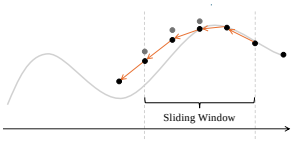


Figure 9.5: Update the values to $\mathbf{x}_{\ell:\ell+p}^{(k+1)}$ using the cumulative drift of points in the window

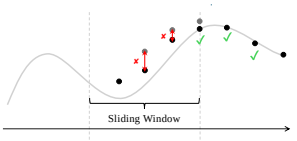


Figure 9.6: Determine how far to slide the window forward, based on the error $\|\mathbf{x}_i^{(k+1)} - \mathbf{x}_i^{(k)}\|^2$.

Source: Adapted from Shih et al. *(2023).*

Sliding Windows and Parallel Evaluation. The discrete Picard update Equation (9.6.3) expresses each $\mathbf{x}_j^{(k+1)}$ as the fixed initial value $\mathbf{x}_T$ minus a cumulative sum of drifts. To limit memory and exploit parallel hardware, it is convenient to apply the same idea locally on short, sliding blocks of indices.

Fix a window length p and a left index ℓ; the window then covers $j = \ell, \ldots, \ell+p$ with $t_\ell > t_{\ell+1} > \cdots > t_{\ell+p}$. During iteration k:

Step 1. Parallel Drift Evaluation on the Window. Compute, in parallel and using only the previous iterate,

$$\mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}), \quad i = 0, 1, \ldots, p-1.$$

These are the p local increments needed to advance from the left edge t_ℓ across the window.

Step 2. Left–Anchored Cumulative Updates. Form the windowed updates by anchoring at $j = \ell$ and accumulating the drift across subintervals:

$$\mathbf{x}_{\ell+j+1}^{(k+1)} = \mathbf{x}_\ell^{(k)} - \Delta t \sum_{i=0}^{j} \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}), \quad j = 0, 1, \ldots, p-1. \tag{9.6.4}$$

This is the local, windowed counterpart of Equation (9.6.3), with the current left-boundary value $\mathbf{x}_\ell^{(k)}$ serving as the anchor for the window. The minus sign reflects the decreasing time direction. The inner sum is a prefix sum over the windowed drifts, so all partial sums can be produced efficiently on parallel hardware.

Step 3. Progress Control and Window Advance. Having formed the left–anchored cumulative updates on the current window (Step 2), we now decide how far to slide that window. We measure local convergence by the pointwise Picard change

$$\texttt{error}_j := \|\mathbf{x}_{\ell+j}^{(k+1)} - \mathbf{x}_{\ell+j}^{(k)}\|^2, \quad j = 1, \dots, p-1,$$

and compare it against prescribed tolerances $\texttt{tol}_{\ell+j}$. That is, $\texttt{error}_j$ measures how much the iterate at node $\ell + j$ changed during the last Picard update. If this number is small, it indicates local agreement between the two successive approximations and hence local convergence of the fixed-point iteration at that node. If it is large, that node has not settled yet and needs more Picard smoothing.

The *stride* is chosen as the first index in the window that fails this test (or the full window length p if none fail):

$$\texttt{stride} := \min\Big(\{\, j \geq 1 : \ \texttt{error}_j > \texttt{tol}_{\ell+j} \,\} \ \cup \ \{p\}\Big).$$

We then slide the window by setting $\ell \leftarrow \ell + \texttt{stride}$. In words: we accept all nodes from the left edge up to (but not including) the first one that has not converged; if all nodes have converged, we accept the entire window. We then slide the window by that many accepted nodes, $\ell \leftarrow \ell + \texttt{stride}$, and continue. This advances by at most the window length p, never skipping any node that has not met its tolerance. If sliding would overrun the grid end M, we truncate the window to $p \leftarrow \min\{p, M - \ell\}$ and proceed.

When the window moves forward it uncovers new time nodes that have no values yet. To start Picard iteration there, we simply copy the value from the left boundary of the window and use it as an initial guess. This "constant extrapolation" is cheap and stable, and will be corrected by later updates. If desired, one can replace it by more accurate guesses, such as linear or polynomial extrapolations from past points.

This completes the procedure of ParaDiGMS. We summarize the algorithm in Algorithm 8.

9.6.3 Relation to Time-Stepping Solvers

Selection of Sliding Window Size. To understand the role of the window size, first consider the smallest choice $p = 1$. In this case, there is no time parallelism, and the method reduces to its sequential base update. For the left-endpoint discretization in Equation (9.6.4), this is simply a plain Euler step.

More generally, ParaDiGMS specifies how computations across different time points can be parallelized, while the numerical update used at each step can be chosen separately. For example, the Euler update can be replaced by DDIM or DPM-Solver, giving ParaDDIM or ParaDPMSolver. Increasing p then exposes more time points to parallel computation without changing the underlying time

Algorithm 8 ParaDiGMS with Sliding Windows

Input: Drift $\mathbf{v}_{\phi^\times}(\mathbf{x}, t)$; $\{t_j\}_{j=0}^{M}$; window length p; $\{\texttt{tol}_j\}_{j=1}^{M}$

Output: Approximate trajectory $\{\mathbf{x}_j^{(k)}\}_{j=0}^{M}$ with $\mathbf{x}_M^{(k)}$ at $t = 0$

$k \leftarrow 0$, $\ell \leftarrow 0$

Sample $\mathbf{x}_0^{(0)} \sim p_{\text{prior}}$; set $\mathbf{x}_j^{(0)} \leftarrow \mathbf{x}_0^{(0)}$ for $j = 1, \ldots, \min(p, M)$ ▷ constant extrapolation

while $\ell < M$ **do**

  $J \leftarrow \min(p, M - \ell)$ ▷ current window length

  Step 1: Parallel

  For $i = 0, \ldots, J-1$: $g_i \leftarrow \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i})$ ▷ drifts from previous iterate (Picard freezing)

  Compute prefix sums $S_j \leftarrow \sum_{i=0}^{j} g_i$ for $j = 0, \ldots, J-1$ ▷ scan over windowed drifts

  Step 2: Cumulative Updates

  For $j = 0, \ldots, J-1$: $\mathbf{x}_{\ell+j+1}^{(k+1)} \leftarrow \mathbf{x}_{\ell}^{(k)} - \Delta t\, S_j$ ▷ left-anchored update; cf. Equation (9.6.4)

  Step 3: Progress Control and Window Advance

  For $j = 1, \ldots, J-1$: $\texttt{error}_j \leftarrow \|\mathbf{x}_{\ell+j}^{(k+1)} - \mathbf{x}_{\ell+j}^{(k)}\|^2$ ▷ pointwise Picard change

  $\texttt{stride} \leftarrow \min\Big(\{\, j \in \{1, \ldots, J-1\} : \ \texttt{error}_j > \texttt{tol}_{\ell+j} \,\} \ \cup \ \{J\}\Big)$

  Initialize New Nodes

  For $r = 1, \ldots, \min\{\texttt{stride},\ M - \ell - J\}$: $\mathbf{x}_{\ell+J+r}^{(k+1)} \leftarrow \mathbf{x}_{\ell+J}^{(k+1)}$ ▷ constant extrapolation into newly exposed indices

  $\ell \leftarrow \ell + \texttt{stride}$; $\quad k \leftarrow k + 1$

end while

return $\{\mathbf{x}_j^{(k)}\}_{j=0}^{M}$

grid or the total number of steps M. Picard iteration repeatedly corrects these states toward consistency with the chosen base solver, with convergence controlled by the prescribed local tolerances.

Compatibility with Higher–Order Solvers (e.g., DPM). The Picard framework determines how states at different time points are updated and evaluated in parallel, while the numerical rule used to approximate each step can be chosen separately.

For example, instead of the left-endpoint rule in Equation (9.6.4), we may use the trapezoidal rule:

$$\mathbf{x}_{\ell+j+1}^{(k+1)} = \mathbf{x}_{\ell}^{(k)} - \Delta t \left[\tfrac{1}{2}\mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell}^{(k)}, t_\ell) + \sum_{i=1}^{j} \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}) + \tfrac{1}{2}\mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+j+1}^{(k)}, t_{\ell+j+1}) \right],$$

where $j = 0, \ldots, p-1$. For $j = 0$, this is the familiar one-step trapezoidal update using the two endpoint drifts. For larger j, the same rule accumulates the trapezoidal contributions across the window. Since all drifts are evaluated using the previous Picard iterate, they can still be computed in parallel, while the interior contributions can be accumulated with a prefix sum.

More generally, ParaDiGMS can use an existing sequential sampler such as DDIM or DPM-Solver as its base discretization. In that case, the Picard update is constructed from the corresponding base-solver increment rather than from the trapezoidal formula above. The precise form depends on the chosen solver, since higher-order methods may use intermediate stages or previous model evaluations.

For a ParaDiGMS construction based on a given sequential solver, exact fixed-point convergence recovers the corresponding sequential discrete trajectory. With a finite stopping tolerance, the final result also contains a remaining fixed-point iteration error in addition to the discretization error of the base solver.

9.7 Closing Remarks

This chapter has confronted one of the most significant practical limitations of diffusion models: their slow, iterative sampling process. We have explored training-free approaches that accelerate generation by designing better numerical solvers for the empirical PF-ODE:

1. We began with DDIM, a first-order factored exponential-integrator update that retains the known diffusion schedule analytically while holding the neural prediction fixed.

2. We then studied DEIS, which extends this idea to higher-order multistep updates by reusing past model evaluations to extrapolate how the neural prediction changes.

3. Finally, we examined the DPM-Solver family, which introduces the log-SNR time coordinate and constructs higher-order updates from within-step model evaluations. DPM-Solver++ further develops single-step and multistep variants in the data-prediction formulation.

Together, these methods illustrate a common principle: exploit the analytically known diffusion structure and concentrate numerical approximation on the learned model prediction. They differ in the prediction and time parameterizations, and in whether higher-order information is obtained from within-step stages or previous-step history. ParaDiGMS provides a complementary direction by reorganizing computation across time through fixed-point iteration and parallel evaluation. Such solver developments can reduce the sampling cost from hundreds or thousands of model evaluations to only tens in many practical settings.

However, these training-free methods remain fundamentally iterative: they numerically approximate a continuous generative trajectory, even when parts of the computation can be parallelized. This raises a natural question: *can we achieve high-quality generation in just one or a very few steps?*

The final Part D of this monograph explores this question through training-based acceleration. In Chapter 10, we study distillation-based methods, where a fast *student* generator learns to reproduce the behavior of a slow pre-trained *teacher*. In Chapter 11, we turn to methods that directly learn few-step generators from scratch defined by an ODE flow.

This shift from designing a better numerical solver to directly learning a fast approximation of the generative solution map aims to combine the quality of diffusion models with the speed of one- or few-step generation.

Part D

Toward Learning Fast Diffusion-Based Generators

10

Distillation-Based Methods for Fast Sampling

This chapter introduces training-based approaches that accelerate diffusion model sampling by teaching new generators to produce samples in only one or a few steps. The central idea, called *distillation*, is to let a fast student model learn from a slow, pre-trained diffusion model (teacher) sampler. While the teacher may require hundreds of steps, the student can achieve comparable quality in only a few steps[1]. Unlike solver-based acceleration, which improves the numerical integration scheme, distillation directly trains a generator to take efficient shortcuts. We highlight two main paradigms: *distribution level distillation*, which skips simulating the full trajectory and instead aligns the student's output distribution with the teacher's, and *flow map level distillation*, which trains the student to reproduce the teacher's sampling path in a faster and more compact way.

After this chapter, you will be able to...

- distinguish the two main distillation targets: the teacher's distribution and its state-to-state flow map;
- understand distribution-level distillation, where the student matches the teacher's noisy marginals through score-based supervision;
- understand flow-map distillation, where the student learns teacher-generated transitions, with Progressive Distillation progressively replacing two teacher steps by one student step.

Reader route. With a pre-trained diffusion model as the teacher, the key distinction in this chapter is what the student is trained to reproduce: the teacher's distribution or its flow map. The next chapter asks how such fast maps can be learned without a teacher.

[1] Here, distillation means reducing sampling steps rather than model size.

10.1 Prologue

A central bottleneck of diffusion models is their slow sampling speed.

As shown through Tweedie's formula (Section 6.3.1), a diffusion model can be interpreted as an "$\mathbf{x}$-prediction" model, $\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$, trained to recover the expected clean data from a noisy input $\mathbf{x}_t$ at noise level t:

$$\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t) \approx \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t],$$

where the expectation is taken with respect to $p(\mathbf{x}_0|\mathbf{x}_t)$, representing all plausible clean data corresponding to $\mathbf{x}_t$. A natural idea is to use $\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$ for one step generation. Yet because this denoiser averages over many plausible outcomes, the prediction can become overly smooth, and generation with only a few denoising steps leads to blurry, low quality samples.

On the other hand, as discussed in Section 4.3.2, diffusion sampling follows an ODE or SDE trajectory through a long sequence of iterative steps. This produces high fidelity samples, but repeated neural-network evaluations make sampling expensive. For a deterministic ODE solver of order n, under standard regularity assumptions, the local truncation error is $\mathcal{O}(h^{n+1})$ and the global discretization error over a fixed time interval is $\mathcal{O}(h^n)$, where $h = \max_i |t_i - t_{i-1}|$ is the step size. Thus, using a coarser discretization generally increases numerical error for a fixed learned vector field, although sample fidelity need not decrease monotonically with the number of solver steps. This creates a practical trade off between sampling accuracy and computational cost.

To overcome this bottleneck, a major line of research is *distillation*, which assumes access to a well-trained diffusion model (the *teacher*) and trains a generator (the *student*) to reproduce its behavior through a single feed forward or few step computation. This compresses the teacher's many sampling steps into a fast process, effectively bypassing slow iterative solvers while maintaining high sample fidelity.

Below, we introduce two perspectives on distillation: *distribution level distillation* and *flow map level distillation*[2].

10.1.1 Distribution Level Distillation

The goal of distribution-based distillation is to train a one-step generator $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$ that maps noise $\mathbf{z} \sim p_{\text{prior}}$ to a sample $\hat{\mathbf{x}} = \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$, inducing a distribution $p_{\boldsymbol{\theta}}(\hat{\mathbf{x}})$ that approximates the target data distribution $p_{\text{data}}(\mathbf{x})$. This is typically achieved

[2] Chronologically, flow map level distillation appeared earlier, with *Knowledge Distillation* (KD) (Luhman and Luhman, 2021) in 2021 and *Progressive Distillation* (PD) (Salimans and Ho, 2021) in 2022, preceding the family of distribution level distillation approaches that emerged around 2023. For smoother exposition and connection to the next chapter, however, we present distribution level distillation first.

by minimizing a statistical divergence

$$\min_{\boldsymbol{\theta}} \mathcal{D}(p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}), p_{\text{data}}(\hat{\mathbf{x}}))\,,$$

where $\mathcal{D}$ denotes a suitable divergence measurement such as KL.

In practice, distribution based methods align the generator's distribution with the empirical distribution $p_{\boldsymbol{\phi}^\times}(\mathbf{x})$ produced by a pre-trained diffusion model:

$$\min_{\boldsymbol{\theta}} \mathcal{D}(p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}), p_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}))\,,$$

where $p_{\boldsymbol{\phi}^\times}$ serves as a surrogate for p_{data}. Rather than evaluating this divergence explicitly, these methods construct tractable estimators of its gradient using the pre-trained teacher together with quantities estimated from the student's current distribution. For KL-based distribution matching, this gradient takes the form of a difference between the student and target score functions.

This formulation distills multi-step generative processes of diffusion models into a single step model through distributional alignment. We detail this approach in Section 10.2.

10.1.2 Flow Map Level Distillation

We consider the PF-ODE, which can be expressed for any prediction model (see Equation (6.3.1)):

$$\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} = f(\tau)\mathbf{x}(\tau) - \tfrac{1}{2}g^2(\tau)\nabla_{\mathbf{x}} \log p_\tau(\mathbf{x}(\tau)) =: \mathbf{v}^*(\mathbf{x}(\tau), \tau). \tag{10.1.1}$$

Its solution map (flow map), starting from $\mathbf{x}_s$ at time s and evolving reversely to time $t \le s$, is denoted by $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$; that is,

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) \coloneqq \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}(\tau), \tau)\,\mathrm{d}\tau, \tag{10.1.2}$$

where the integral solves the PF-ODE. Intuitively, $\boldsymbol{\Psi}_{s\to t}$ transports, $\mathbf{x}_s$, noise at time s to less noisy states at time t (ultimately data at $t = 0$).

Sampling from a diffusion model corresponds to evaluating $\boldsymbol{\Psi}_{T\to 0}(\mathbf{x}_T)$ for $\mathbf{x}_T \sim p_{\text{prior}}$. Typically, this integral is approximated by iterative numerical solvers leveraging the velocity field $\mathbf{v}$ (see Chapter 9), but requires many steps (e.g., at least 10-20 NEFs even in DPM-Solver), making sampling slower than classic one-step generative models such as GAN. This motivates a natural question:

Question 10.1.1

Can we learn the solution map $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$ directly?

In particular, learning a map $\boldsymbol{\Psi}_{T\to 0}(\mathbf{x}_T)$ with $\mathbf{x}_T \sim p_{\text{prior}}$ enables one-step generation.

Trajectory Distillation. Trajectory distillation seeks to train a neural generator that approximates the solution map at the instance level. Since the PF-ODE integral rarely admits a closed form, it must be approximated numerically during training. To formalize, we introduce the general solver notation

$$\texttt{Solver}_{s\to t}(\mathbf{x}_s;\, \boldsymbol{\phi}^{\times}) \quad \text{or simply} \quad \texttt{Solver}_{s\to t}(\mathbf{x}_s), \tag{10.1.3}$$

denoting numerical integration of the empirical PF-ODE from s to t starting at $\mathbf{x}_s$, with teacher parameters $\boldsymbol{\phi}^{\times}$ (omitted when clear from context).

An Early Approach: Direct Knowledge Distillation. To enable few step or even one step generation, a direct approach is to train a generator $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0)$ to imitate the output of a numerical solver evaluated along the full trajectory:

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0) \approx \texttt{Solver}_{T\to 0}(\mathbf{x}_T), \qquad \mathbf{x}_T \sim p_{\text{prior}}.$$

This idea underlies one of the earliest trajectory distillation methods, *Knowledge Distillation* (Luhman and Luhman, 2021), which uses the regression loss

$$\mathcal{L}_{\text{KD}}(\boldsymbol{\theta}) := \mathbb{E}_{\mathbf{x}_T\sim p_{\text{prior}}} \left\|\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0) - \texttt{Solver}_{T\to 0}(\mathbf{x}_T)\right\|_2^2.$$

While this approach provides direct supervision from the pre-trained teacher, it cannot leverage the strong supervision available in the original training data. In addition, it is computationally expensive if ODE integration is invoked within the training loop, since each parameter update requires solving the ODE to form targets. Finally, because the generator learns only a global mapping from T to 0, it may lose controllability for steering the generation process from intermediate states. Consequently, most controllable generation techniques introduced in Chapter 8 cannot be directly applied.

Preface to Progressive Distillation. *Progressive Distillation* (PD) (Salimans and Ho, 2021) trains a time–conditional `Student` using *local* supervision from `Teacher` fragments. Let $t_0 = T > t_1 > \cdots > t_N = 0$ be a fixed time grid. The `Teacher` provides time–stepping maps $\texttt{Teacher}_{t_k\to t_{k+1}}$ for $k = 0, \ldots, N-1$.

Rather than supervising only the one-jump $T \to 0$, PD trains the `Student` two–step skip map to match two consecutive `Teacher` steps:

$$\texttt{Student}_{t_k\to t_{k+2}} \;\approx\; \texttt{Teacher}_{t_{k+1}\to t_{k+2}} \circ \texttt{Teacher}_{t_k\to t_{k+1}},$$

for $k = 0, 2, 4, \ldots$. The matching is performed using a simple regression loss (e.g., mean squared error).

After training on locally paired fragments, the `Student` no longer follows every time interval of the original grid. Instead, it advances on every other time point,

$$t_0 \to t_2 \to t_4 \to \cdots \to t_N,$$

which means that each `Student` step effectively covers two consecutive `Teacher` steps. Consequently, the `Student` completes the same overall time span $[0, T]$ using only $N/2$ transitions.

After this stage, the trained `Student` replaces the `Teacher` to serve as the new reference model. The entire procedure is then repeated on the coarser grid, where the time step doubles ($N \to N/2 \to N/4 \to \cdots$), progressively distilling the trajectory into fewer and fewer steps until the desired number of inference steps is reached. This iterative halving preserves the global time horizon while continually compressing the temporal resolution of the generative process.

A Unified Perspective of Flow Map Learning. Various methods, including KD and PD, can be expressed within a unified loss framework:

$$\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta}) := \mathbb{E}_{s,t}\mathbb{E}_{\mathbf{x}_s\sim p_s}\left[w(s,t)d\big(\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s,s,t),\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)\big)\right], \tag{10.1.4}$$

where $\boldsymbol{\Psi}_{s\to t}$ is the oracle flow map, $w(s,t) \geq 0$ specifies how different time pairs (s,t) are weighted, $d(\cdot,\cdot)$ is a discrepancy measure such as $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2$ or $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_1$, and p_s denotes the forward noised marginal at time s. Because $\boldsymbol{\Psi}_{s\to t}$ is not available in closed form, one must rely on approximations, typically through a pre-trained diffusion model (teacher) or another tractable surrogate.

KD appears as a simple instance of Equation (10.1.4) by considering only the endpoint pair $(s,t) = (T,0)$. Equivalently, one may take the degenerate weighting $w(s,t) = \delta(s-T)\delta(t)$. Assuming $p_T = p_{\text{prior}}$[3], the oracle objective becomes

$$\mathbb{E}_{\mathbf{x}_T\sim p_T}\big\|\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T,T,0) - \boldsymbol{\Psi}_{T\to 0}(\mathbf{x}_T)\big\|_2^2 \approx \mathcal{L}_{\text{KD}}(\boldsymbol{\theta}),$$

where the practical KD target replaces the oracle flow map by $\texttt{Solver}_{T\to 0} \approx \boldsymbol{\Psi}_{T\to 0}$. An alternative perspective on this formulation is presented in Section D.5.

PD also fits this template, but instead of supervising only with the single extreme pair $(T,0)$, it uses many *nearby* time pairs and enforces a simple *local consistency* rule: a short step followed by another short step should match the direct two–step move. We return to this in Equation (10.3.3).

In practice, the main challenge is that the oracle flow map $\boldsymbol{\Psi}_{s\to t}$ generally has no closed-form expression, making direct supervision infeasible. A range of methods have been developed to approximate this target efficiently, but their success often hinges on the quality of the teacher model. We will return to Equation (10.1.4) in Chapter 11, presenting a principled framework for training-from-scratch methods that eliminate the teacher from the learning loop.

[3] This equality holds when the noise schedule reaches the prior exactly at T; otherwise, one typically has $p_T \approx p_{\text{prior}}$ for sufficiently large T.

10.2 Distribution-Based Distillation

Several works have pursued this distribution-based distillation concurrently under different names, including Distributional Matching Distillation (DMD) (Yin *et al.*, 2024b; Yin *et al.*, 2024a), Variational Score Distillation (VSD) (Poole *et al.*, 2023; Wang *et al.*, 2023; Luo *et al.*, 2023; Lu and Song, 2024), and Score Identity Distillation (SiD) (Zhou *et al.*, 2024). Despite technical differences, they share the same principle: train a generator whose forward-noised marginals match those of the teacher. We focus on VSD as a representative formulation, since the others follow similar principles.[4]

10.2.1 Formulation of VSD as a Representative Approach

Forward Process. Let $\{p_t\}_{t\in[0,T]}$ denote the marginal densities of a forward diffusion process induced by

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

with initial distribution $p_0 = p_{\text{data}}$. In contrast, let $p_0^{\boldsymbol{\theta}}$ denote the distribution of synthetic samples generated by a deterministic one-step generator $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$ from latent variables $\mathbf{z} \sim p_{\text{prior}}(\mathbf{z})$. Define $\{p_t^{\boldsymbol{\theta}}\}_{t\in[0,T]}$ as the marginal densities obtained by applying the same forward diffusion process to $p_0^{\boldsymbol{\theta}}$, that is,

$$\mathbf{x}_t^{\boldsymbol{\theta}} := \alpha_t \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) + \sigma_t \boldsymbol{\epsilon}, \tag{10.2.1}$$

where $\mathbf{z} \sim p_{\text{prior}}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Thus, both p_t and $p_t^{\boldsymbol{\theta}}$ share the same Gaussian diffusion kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ but differ in their starting distributions (p_{data} vs. $p_0^{\boldsymbol{\theta}}$ of one-step synthetic samples).

Training Objective and Gradient. The literature typically adopts the KL divergence to match the distributions p_t and $p_t^{\boldsymbol{\theta}}$, commonly by minimizing

$$\mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) := \mathbb{E}_t \left[\omega(t) \mathcal{D}_{\text{KL}}(p_t^{\boldsymbol{\theta}} \,\|\, p_t)\right] = \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \left[\omega(t) \left(\log p_t^{\boldsymbol{\theta}}(\mathbf{x}_t^{\boldsymbol{\theta}}) - \log p_t(\mathbf{x}_t^{\boldsymbol{\theta}})\right)\right],$$

where $\omega(t) \geq 0$ is a time-dependent weighting function. We will discuss in Section 10.2.3 why the KL divergence plays a special role in distribution-level distillation.

As shown in (Wang *et al.*, 2023), at the population level, the global optimum is achieved when $p_0^{\boldsymbol{\theta}^*} = p_{\text{data}}$, indicating that the generator's distribution matches the data distribution. Thus, the VSD objective provides a principled distribution-matching loss for learning p_{data}.

[4] Historically, VSD (Wang *et al.*, 2023) was introduced for text-to-3D generation, where a pre-trained 2D diffusion model provides distributional supervision through a differentiable renderer. Its single-particle (Dirac) specialization recovers the basic SDS (Poole *et al.*, 2023) update, up to the corresponding time-weighting and prediction-parameterization conventions.

However, the density-based formulation of the objective lacks an efficient training mechanism. Fortunately, by taking the gradient with respect to $\boldsymbol{\theta}$, we arrive at the expression in Equation (10.2.2), which is summarized in the following proposition. For notational simplicity, we denote $\hat{\mathbf{x}}_t := \mathbf{x}_t^{\boldsymbol{\theta}}$ as defined in Equation (10.2.1).

Proposition 10.2.1: θ-Gradient of $\mathcal{L}_{\text{VSD}}$

We have

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \left[\omega(t) \alpha_t \left(\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t) \right) \cdot \partial_{\boldsymbol{\theta}} \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) \right]. \tag{10.2.2}$$

Proof for Proposition.

The derivation applies the chain rule:

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_t \left[\omega(t) \mathcal{D}_{\text{KL}}(p_t^{\boldsymbol{\theta}} \| p_t) \right] \\
&= \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \left[\omega(t)\, \partial_{\boldsymbol{\theta}} \big(\log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) - \log p_t(\hat{\mathbf{x}}_t) \big) \right] \\
&= \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \left[\omega(t) \left(\underbrace{\partial_{\boldsymbol{\theta}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t)}_{\text{first}} + (\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t))^\top \partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t - (\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t))^\top \partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t \right) \right].
\end{aligned}$$

The first term vanishes by the score-function identity:

$$\mathbb{E}_{\hat{\mathbf{x}}_t \sim p_t^{\boldsymbol{\theta}}} \left[\partial_{\boldsymbol{\theta}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) \right] = \int \partial_{\boldsymbol{\theta}} p_t^{\boldsymbol{\theta}}(\mathbf{x})\, \mathrm{d}\mathbf{x} = \partial_{\boldsymbol{\theta}} \int p_t^{\boldsymbol{\theta}}(\mathbf{x})\, \mathrm{d}\mathbf{x} = \partial_{\boldsymbol{\theta}}(1) = 0.$$

Using the reparameterization $\hat{\mathbf{x}}_t = \alpha_t \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) + \sigma_t \boldsymbol{\epsilon}$ gives $\partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t = \alpha_t \partial_{\boldsymbol{\theta}} \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$, hence

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \left[\omega(t) \alpha_t (\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t))^\top \partial_{\boldsymbol{\theta}} \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) \right].$$

This proves Equation (10.2.2). See Section D.5 for details. ∎

We observe that the score functions naturally emerge when taking the gradient with respect to $\boldsymbol{\theta}$. Consequently, we require approximations of the score $\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t)$ for the one-step generator and $\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t)$ for the data distribution, as will be detailed in the following subsection.

10.2.2 Training Pipeline of VSD

Existing works (Yin *et al.*, 2024b; Yin *et al.*, 2024a; Poole *et al.*, 2023; Wang *et al.*, 2023; Luo *et al.*, 2023; Lu and Song, 2024) typically address this via a bi-level optimization approach: training a new diffusion model on samples from $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$ to

approximate $\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t)$, and employing a pre-trained diffusion model to provide a proxy for the intractable oracle score function $\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t)$ on synthetic samples $\hat{\mathbf{x}}_t$. More precisely, training proceeds by alternating between two phases:

- **Score Estimation Phase.** Fix $\boldsymbol{\theta}$. Let $\hat{\mathbf{x}}_0 = \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})$ and $\hat{\mathbf{x}}_t = \alpha_t \hat{\mathbf{x}}_0 + \sigma_t \boldsymbol{\epsilon}$ with $\mathbf{z} \sim p_{\text{prior}}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Train $s_{\boldsymbol{\zeta}}$ by DSM using the known Gaussian diffusion kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$:

$$\mathcal{L}_{\text{DSM}}(\boldsymbol{\zeta}; \boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \Big\| \mathbf{s}_{\boldsymbol{\zeta}}(\hat{\mathbf{x}}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\hat{\mathbf{x}}_t|\hat{\mathbf{x}}_0) \Big\|^2,$$

  which yields $\mathbf{s}_{\boldsymbol{\zeta}}(\cdot, t) \approx \nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\cdot)$ at optimum (for fixed $\boldsymbol{\theta}$).

- **Generator Update Phase.** With $s_{\boldsymbol{\zeta}}$ frozen (stop-grad), $\boldsymbol{\theta}$ is updated by using the gradient in Equation (10.2.2), replacing the individual score terms by their respective proxies:

$$\mathbf{s}_{\boldsymbol{\zeta}}(\hat{\mathbf{x}}_t, t) \approx \nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t), \text{ and } \mathbf{s}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_t, t) \approx \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t) \quad \text{(teacher)}.$$

  Equation (10.2.2) then approximately becomes:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) \approx \mathbb{E}_{t,\mathbf{z},\boldsymbol{\epsilon}} \Big[\omega(t) \alpha_t \big(\mathbf{s}_{\boldsymbol{\zeta}}(\hat{\mathbf{x}}_t, t) - \mathbf{s}_{\boldsymbol{\phi}^\times}(\hat{\mathbf{x}}_t, t) \big)^\top \partial_{\boldsymbol{\theta}} \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) \Big].$$

These two phases are alternated throughout training. Ideally, the auxiliary model tracks the current student score $\mathbf{s}_{\boldsymbol{\zeta}}(\cdot, t) \approx \nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\cdot)$, while the pre-trained teacher provides an approximation to the target score $\mathbf{s}_{\boldsymbol{\phi}^\times}(\cdot, t) \approx \nabla_{\mathbf{x}} \log p_t(\cdot)$. At the population level, if these two score functions agree at a noised level with $\sigma_t > 0$ and $\alpha_t \neq 0$, then their corresponding normalized densities agree, $p_t^{\boldsymbol{\theta}} = p_t$. Since Gaussian noising is injective when $\alpha_t \neq 0$, this further implies $p_0^{\boldsymbol{\theta}} = p_{\text{data}}$.

In practice, both scores are represented by finite learned networks, so these equalities characterize the ideal population solution rather than a guarantee of alternating optimization.

10.2.3 Beyond KL: General Divergences and Why KL Is Special

The KL divergence is not essential to the distribution-matching principle of VSD. In principle, one may use a general f-divergence (see Equation (1.1.4)):

$$\mathcal{D}_f(p_t^{\boldsymbol{\theta}} \| p_t) = \int p_t(\mathbf{x}) \, f\left(\frac{p_t^{\boldsymbol{\theta}}(\mathbf{x})}{p_t(\mathbf{x})} \right) \mathrm{d}\mathbf{x}.$$

What makes the reverse KL particularly convenient is its gradient structure. Define the density ratio

$$r_t(\mathbf{x}) := \frac{p_t^{\boldsymbol{\theta}}(\mathbf{x})}{p_t(\mathbf{x})}.$$

For a twice-differentiable convex f and positive densities, the pathwise gradient can be written as

$$\nabla_{\boldsymbol{\theta}} \mathcal{D}_f(p_t^{\boldsymbol{\theta}} \| p_t) = \mathbb{E}_{\hat{\mathbf{x}}_t \sim p_t^{\boldsymbol{\theta}}} \left[r_t(\hat{\mathbf{x}}_t) f''(r_t(\hat{\mathbf{x}}_t)) \left(\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t)\right)^\top \partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t \right].$$

Conceptually, the gradient has three components:

$$\nabla_{\boldsymbol{\theta}} \mathcal{D}_f = \mathbb{E}\Big[\underbrace{r_t f''(r_t)}_{\text{density-ratio weighting}} \times \underbrace{\left(\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}} - \nabla_{\mathbf{x}} \log p_t\right)^\top}_{\text{score mismatch}} \times \underbrace{\partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t}_{\text{parameter sensitivity}} \Big].$$

The first term determines *where and how strongly* to update, the second gives the score-mismatch direction, and the last maps this sample-space signal back to the generator parameters.

This expression reveals a useful common structure: all such f-divergences use the same *score-mismatch direction*,

$$\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log r_t(\mathbf{x}),$$

while the choice of divergence changes how strongly different regions are weighted through $r_t f''(r_t)$. Thus, for the usual strictly convex choices of f, the population distribution-matching condition remains $p_t^{\boldsymbol{\theta}} = p_t$, while the choice of divergence changes how different density-ratio regimes are weighted during optimization through $r_t f''(r_t)$.

For the reverse KL used in VSD, $f(r) = r \log r$, and therefore

$$r f''(r) = 1.$$

The density-ratio weighting disappears, recovering the score-difference gradient in Equation (10.2.2). Consequently, once the student and target scores are available, reverse-KL training does not require an additional pointwise density-ratio estimator. By contrast, a general f-divergence requires the factor $r_t f''(r_t)$; although the score difference provides $\nabla_{\mathbf{x}} \log r_t$, the ratio r_t itself is not directly available for an implicit generator and generally requires additional estimation.

For example, the forward KL corresponds to $f(r) = -\log r$, for which $r f''(r) = 1/r$. It therefore places greater weight on regions where the model density is small relative to the target density. This illustrates how different divergences can share the same underlying score-mismatch direction while emphasizing different density-ratio regimes.

Finally, this score-mismatch view connects directly to the Wasserstein gradient-flow perspective in Section B.3. At the distribution level, reverse-KL gradient flow moves particles according to the target score minus the model score. The VSD parameter gradient contains the same score mismatch, pulled back through the generator Jacobian $\partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t$. Thus, the Wasserstein view describes the ideal distribution-space motion, while VSD realizes this motion through updates of a finite-dimensional shared generator.

10.3 Progressive Distillation

Progressive Distillation (PD) (Salimans and Ho, 2021) consists of two procedures that together enable a diffusion model to learn the PF-ODE trajectory more efficiently. The key idea is to progressively reduce the number of integration steps required for high-quality sampling while retaining fidelity to the teacher trajectory.

- **Distillation Operation:** Distills a deterministic sampler (e.g., DDIM) based on a pre-trained teacher model (initially a diffusion model) into a student model that reproduces the same trajectory using only half as many sampling steps.
- **Progressive Operation:** Repeats this distillation process iteratively, each time halving the number of steps, until the student can generate high-quality samples within a small fixed budget (typically 1–4 steps).

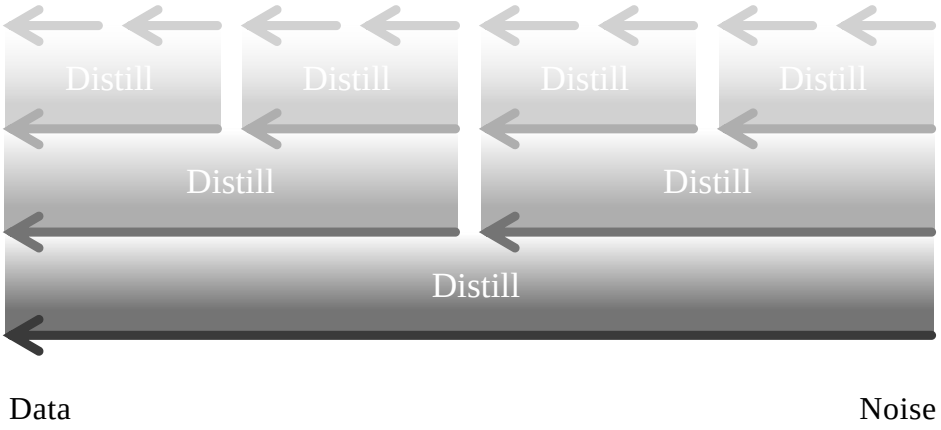


Figure 10.1: Illustration of Progressive Distillation (PD). At each round, the student model is trained so that a single step reproduces the effect of two adjacent teacher steps. This process distills N teacher steps into $N/2$ student steps, and repeating the procedure progressively halves the trajectory length until the desired step count is reached. The arrows indicate how multi-step teacher transitions are compressed into fewer student steps, moving from noise to data.

Source: Created by the authors.

We first introduce the distillation operation of PD in Section 10.3.1, and then summarize the entire training pipeline in Section 10.3.2. Section 10.3.4 presents an extension for CFG guidance.

10.3.1 Distillation Operation in PD

In this section, we fix DDIM in the $\mathbf{x}$-prediction parameterization as the time–stepping rule and still write $\texttt{Solver}_{s\to t}$ for the deterministic map obtained by plugging the

current teacher's $\mathbf{x}$-denoiser into DDIM.

In the first PD round (teacher = pre–trained diffusion model), this coincides with integrating the diffusion PF–ODE via DDIM; in later rounds (teacher = previous student), $\texttt{Solver}_{s\to t}$ is simply the DDIM transition induced by the current teacher, not the original diffusion PF–ODE.

The distillation step is as follows: starting from a noisy input $\mathbf{x}_s$ (a perturbed version of clean data, $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$), the student is trained to predict a target $\tilde{\mathbf{x}}$ so that a single student step $s \to t$ reproduces the teacher's two consecutive steps $s \to u \to t$. Let $\mathbf{x}_{\phi^\times}(\mathbf{x}, \tau)$ denote the teacher's $\mathbf{x}$-prediction denoiser in this round. Applying the teacher–induced DDIM transition twice gives

$$\tilde{\mathbf{x}}_u := \texttt{Solver}_{s\to u}(\mathbf{x}_s;\, \mathbf{x}_{\phi^\times})\,, \qquad \tilde{\mathbf{x}}_t := \texttt{Solver}_{u\to t}(\tilde{\mathbf{x}}_u;\, \mathbf{x}_{\phi^\times})\,.$$

Here, we use the notation of Equation (10.1.3) to denote the deterministic transition map from s to t (starting at $\mathbf{x}_s$) induced by plugging $\mathbf{x}_{\phi^\times}$ into DDIM.

Question 10.3.1

What is the pseudo-clean $\tilde{\mathbf{x}}$ at time s such that the solver produces the same output $\tilde{\mathbf{x}}_t$ when stepping directly $s \to t$ as it does via $s \to u \to t$? Specifically, determine $\tilde{\mathbf{x}}$ satisfying:

$$\tilde{\mathbf{x}}_t = \texttt{Solver}_{s\to t}\left(\mathbf{x}_s; \tilde{\mathbf{x}}\right).$$

Once a closed-form expression for $\tilde{\mathbf{x}}$ is obtained, we train a student model $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s)$ (also an $\mathbf{x}$-prediction model here) to approximate the "two-steps-in-one" target $\tilde{\mathbf{x}}$ by minimizing

$$\min_{\boldsymbol{\theta}} \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s}\left[w(\lambda_s)\left\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \tilde{\mathbf{x}}\right\|_2^2\right]. \tag{10.3.1}$$

In the following, we show that the DDIM rule yields $\tilde{\mathbf{x}}$ in closed form through elementary algebra (note that the result holds for both discrete and continuous time):

Lemma 10.3.1: Two-Steps-in-One Target $\tilde{\mathbf{x}}$ of DDIM

Starting from an initial condition $\mathbf{x}_s$, if the solver is taken as DDIM, then the "two-step-in-one" target $\tilde{\mathbf{x}}$ can be computed as

$$\tilde{\mathbf{x}} = \frac{\sigma_s}{\alpha_t\sigma_s - \alpha_s\sigma_t}\tilde{\mathbf{x}}_t - \frac{\sigma_t}{\alpha_t\sigma_s - \alpha_s\sigma_t}\mathbf{x}_s.$$

Here, $\tilde{\mathbf{x}}_t$ is obtained by applying DDIM (in Equation (9.2.2)) twice, from $s \to u \to t$:

$$\begin{aligned} s \to u: \quad & \tilde{\mathbf{x}}_u = \frac{\sigma_u}{\sigma_s}\mathbf{x}_s + \alpha_s\left(\frac{\alpha_u}{\alpha_s} - \frac{\sigma_u}{\sigma_s}\right)\mathbf{x}_{\phi^\times}(\mathbf{x}_s, s) \\ u \to t: \quad & \tilde{\mathbf{x}}_t = \frac{\sigma_t}{\sigma_u}\tilde{\mathbf{x}}_u + \alpha_u\left(\frac{\alpha_t}{\alpha_u} - \frac{\sigma_t}{\sigma_u}\right)\mathbf{x}_{\phi^\times}(\tilde{\mathbf{x}}_u, u). \end{aligned}$$

Proof for Lemma.

$\tilde{\mathbf{x}}_t$ must be matched with the one-step DDIM from s to t, $\tilde{\mathbf{x}}'_t$, expressed as:

$$s \to t: \quad \tilde{\mathbf{x}}'_t = \frac{\sigma_t}{\sigma_s}\mathbf{x}_s + \alpha_s\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)\tilde{\mathbf{x}}.$$

By equating $\tilde{\mathbf{x}}'_t$ and $\tilde{\mathbf{x}}_t$, we can solve for $\tilde{\mathbf{x}}$ in terms of $\tilde{\mathbf{x}}_t$, s, and t:

$$\begin{aligned} & \tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}'_t \\ \iff & \tilde{\mathbf{x}}_t = \frac{\sigma_t}{\sigma_s}\mathbf{x}_s + \alpha_s\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)\tilde{\mathbf{x}} \\ \iff & \tilde{\mathbf{x}} = \frac{\tilde{\mathbf{x}}_t - \frac{\sigma_t}{\sigma_s}\mathbf{x}_s}{\alpha_s\left(\frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s}\right)} \\ \iff & \tilde{\mathbf{x}} = \frac{\sigma_s}{\alpha_t\sigma_s - \alpha_s\sigma_t}\tilde{\mathbf{x}}_t - \frac{\sigma_t}{\alpha_t\sigma_s - \alpha_s\sigma_t}\mathbf{x}_s. \end{aligned} \tag{10.3.2}$$

With this formula, PD computes the pseudo-clean target at time s whose single DDIM step $s \to t$ lands exactly at the two-step output $\tilde{\mathbf{x}}_t$.

Practical Discrete Time Grids and Loss. In practice, we fix a decreasing grid $t_0 = T > t_1 > \cdots > t_N = 0$ and, for brevity, write $s := t_k$, $u := t_{k+1}$, $t := t_{k+2}$. The teacher provides one step maps $\texttt{Teacher}_{t_k \to t_{k+1}}$, and the student learns a two step skip map that matches the teacher composition:

$$\texttt{Student}_{t_k \to t_{k+2}} \approx \texttt{Teacher}_{t_{k+1} \to t_{k+2}} \circ \texttt{Teacher}_{t_k \to t_{k+1}}.$$

For the standard halving schedule, assume N is even and sample triplets $(s, u, t) = (t_k, t_{k+1}, t_{k+2})$ with $k \in \{0, 2, \ldots, N-2\}$. The objective Equation (10.3.1)

becomes

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{k\sim\mathcal{U}\{0,2,\ldots,N-2\}} \mathbb{E}_{\mathbf{x}_{t_k}\sim p_{t_k}}\Big[w(\lambda_{t_k})\,\big\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_k},t_k)-\tilde{\mathbf{x}}^{(k)}\big\|_2^2\Big],$$

where the teacher two-step target $\tilde{\mathbf{x}}^{(k)}$ is computed via Lemma 10.3.1. If the grid is uniform, one may write $t_k = T(1-k/N)$ so that

$$s = T\Big(1-\frac{k}{N}\Big), \quad u = T\Big(1-\frac{k+1}{N}\Big), \quad t = T\Big(1-\frac{k+2}{N}\Big),$$

corresponding to evenly spaced time steps of size $\Delta s = T/N$.

10.3.2 Entire Training Pipeline of PD and Its Sampling

After training on locally paired fragments via Equation (10.3.1), the `Student` no longer follows every interval of the original grid. Instead, each learned step covers two consecutive `Teacher` steps, so the `Student` advances on every other time point,

$$t_0 \to t_2 \to t_4 \to \cdots \to t_N,$$

and thus traverses the same horizon $[0,T]$ using only $N/2$ transitions. After this stage, the trained `Student` replaces the `Teacher` as the new denoiser model. The procedure is then repeated on the coarser grid (the time step doubles), yielding the progression

$$N \;\to\; N/2 \;\to\; N/4 \;\to\; \cdots,$$

until the desired number of inference steps is reached. At each iteration, the new `Student` is initialized from the updated `Teacher`. This iterative halving preserves the global time horizon while progressively compressing the temporal resolution of the generative process.

Sampling. At inference time, using the (DDIM) solver with the current `Student` as the denoiser, the sampler advances on the coarser grid induced by training. After the first round it takes "skip-2" jumps ($t_0 \to t_2 \to \cdots \to t_N$), after the next round "skip-4" ($t_0 \to t_4 \to \cdots \to t_N$), and so on, halving the number of sampling steps at each iteration while keeping the same start and end times.

10.3.3 Additional Discussion: Local Semigroup Matching and the Possibility of Generalized Solvers

Progressive Distillation as Local Semigroup Matching. Within the unified objective Equation (10.1.4), the intractable oracle target $\boldsymbol{\Psi}_{s\to 0}$ is replaced by a teacher–induced surrogate that uses the *semigroup property* of the ODE flow (see more details later in Equation (11.2.1)): evolving from s to t should be equivalent to going from s to any intermediate u and then from u to t,

$$\boldsymbol{\Psi}_{s\to t} = \boldsymbol{\Psi}_{u\to t} \circ \boldsymbol{\Psi}_{s\to u}.$$

PD enforces this locally by training the student's one–step map to match the teacher's composition of two adjacent one–step fragments:

$$\mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} \big\| \underbrace{\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, s - 2\Delta s)}_{\text{student one-step}} - \underbrace{\texttt{Solver}_{s-\Delta s \to s-2\Delta s}\big(\texttt{Solver}_{s \to s-\Delta s}(\mathbf{x}_s)\big)}_{\text{teacher two-step composition}} \big\|_2^2. \tag{10.3.3}$$

When the teacher solver accurately approximates the underlying ODE flow, the composition in Equation (10.3.3) approximates the corresponding semigroup relation on a short decreasing grid (take $s > u > t$ with $u = s - \Delta s$ and $t = s - 2\Delta s$):

$$\begin{aligned} \boldsymbol{\Psi}_{s \to s-2\Delta s} &= \boldsymbol{\Psi}_{s-\Delta s \to s-2\Delta s} \circ \boldsymbol{\Psi}_{s \to s-\Delta s} \\ &\approx \texttt{Solver}_{s-\Delta s \to s-2\Delta s} \circ \texttt{Solver}_{s \to s-\Delta s}, \end{aligned}$$

so training only requires short teacher fragments, rather than a full rollout from time s all the way to 0.

To connect back to the few–step denoiser view in Equation (10.1.4), define the student's few–step map as a composition of learned jumps:

$$\underbrace{\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0)}_{\text{few-step denoiser}} = \big(\mathbf{G}_{\boldsymbol{\theta}}(\,\cdot\,, 2\Delta s, 0) \circ \cdots \circ \mathbf{G}_{\boldsymbol{\theta}}(\,\cdot\,, s, s - 2\Delta s)\big)(\mathbf{x}_s).$$

Conceptually, Equation (10.3.3) provides an efficient *local surrogate* for the global regression

$$\mathbb{E}_{s, \mathbf{x}_s} \Big\| \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0) \;-\; (\texttt{Solver})^{\circ}_{s \to 0}(\mathbf{x}_s) \Big\|_2^2,$$

where $(\texttt{Solver})^{\circ}_{s \to 0}$ denotes the teacher's full composition from s to 0 on a grid with step size Δs, serving as a proxy for $\boldsymbol{\Psi}_{s \to 0}$.

Can we Use Other Solvers? Above, we used DDIM in the $\mathbf{x}$-prediction parameterization as a concrete PF–ODE sampler. The basic distillation principle is more general: several fine teacher steps produce a target endpoint, and the student is trained so that one coarser step reaches the same endpoint.

The closed-form pseudo-target derived above is available whenever the coarse solver update can be inverted with respect to the student's network prediction. For DDIM, the update is affine in a single $\mathbf{x}$-prediction, so this inversion is analytic. The same idea applies to other solvers whose one-step update admits a comparable inversion.

For more complicated solvers, such a single-output inversion may not be available. Multi-stage methods use several network evaluations, multi-step methods depend on previous solver states, and implicit methods require solving coupled equations. In such cases, one can instead compute the endpoint of the coarse student step and directly match it to the endpoint produced by the fine teacher

steps, as in Equation (10.3.3), backpropagating through the student solver when feasible. Since a coarse step may itself require multiple neural function evaluations, fewer time steps do not necessarily imply the same reduction in NFE.

For a stochastic sampler, the same pathwise logic applies after fixing or coupling the random draws used by the teacher and student. Without such coupling, individual trajectories need not share the same endpoint, and the appropriate target is generally distributional rather than pathwise matching.

10.3.4 PD with Guidance

Meng *et al.* (2023) proposed a two-stage pipeline for distilling classifier-free guided (CFG) diffusion models: (1) *distill the guidance* into a single network that takes the guidance weight as input, and (2) apply *progressive distillation (PD)* to reduce the sampling steps. They demonstrated this both in pixel space and in latent space (e.g., Stable Diffusion).

Stage-One Distillation: Distilling Guidance. Let $\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \mathbf{c})$ denote the (pre-trained) conditional diffusion model output in the "$\mathbf{x}$-prediction" parameterization (i.e., a clean estimate) at time s and condition $\mathbf{c}$; the condition can also be null, $\mathbf{c} = \emptyset$ (unconditional branch). The ω-weighted CFG combination in Equation (8.3.3) can be written as

$$\mathbf{x}^{\omega}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \mathbf{c}) := (1+\omega)\,\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \mathbf{c}) - \omega\,\mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \emptyset), \tag{10.3.4}$$

where $\omega \sim p_\omega(\omega)$ for some CFG weighting distribution p_ω, typically $p_\omega(\omega) = \mathcal{U}[\omega_{\min}, \omega_{\max}]$.

Stage-one introduces a new model $\mathbf{x}_{\boldsymbol{\theta}_1}(\mathbf{x}_s, s, \mathbf{c}, \omega)$ that directly takes ω as input and learns to reproduce the CFG output $\mathbf{x}^{\omega}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \mathbf{c})$ by supervised regression:

$$\min_{\boldsymbol{\theta}_1} \mathbb{E}_{\omega\sim p_\omega, s, \mathbf{x}\sim p_{\text{data}}, \mathbf{x}_s\sim p(\mathbf{x}_s|\mathbf{x})} \lambda(s) \big\| \mathbf{x}_{\boldsymbol{\theta}_1}(\mathbf{x}_s, s, \mathbf{c}, \omega) - \mathbf{x}^{\omega}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s, \mathbf{c}) \big\|_2^2.$$

Here $\lambda(s)$ is a standard schedule-dependent weighting; sampling ω each iteration teaches a single network to emulate CFG at arbitrary guidance strengths.

Stage-Two Distillation: PD. The stage-one model $\mathbf{x}_{\boldsymbol{\theta}_1}(\mathbf{x}_s, s, \mathbf{c}, \omega)$ serves as the teacher in PD and is progressively distilled into a student $\mathbf{x}_{\boldsymbol{\theta}_2}(\mathbf{x}_s, s, \mathbf{c}, \omega)$ with fewer sampling steps, following Section 10.3.2. At each iteration, the number of steps is halved (e.g., $N \to N/2 \to N/4 \to \cdots$).

10.4 Closing Remarks

This chapter has introduced our first major paradigm for training-based acceleration. Having exhausted training-free improvements via numerical solvers, we shifted our focus to a new strategy: training a fast *student* generator that learns to replicate the behavior of a slow, pre-trained *teacher* diffusion model.

We explored two primary distillation philosophies. First, in distribution-based distillation, represented by methods like Variational Score Distillation (VSD), the student's output distribution is trained to match the teacher's. This is achieved by aligning their respective score functions across different noise levels, providing a stable, non-adversarial objective. Second, in flow map distillation, we saw how methods like Progressive Distillation (PD) train the student to directly approximate the teacher's solution trajectory. PD's iterative approach, where each round halves the number of sampling steps, proved to be a powerful and practical method for compressing a long iterative process into just a few steps.

These distillation techniques successfully bridge the gap between the high sample quality of iterative diffusion models and the inference speed of one-step generators, offering a compelling pathway to efficient, high-fidelity synthesis.

However, the reliance on a pre-trained teacher model introduces a two-stage pipeline: first train a slow but powerful teacher, then distill it into a fast student. This raises a fundamental question at the forefront of generative modeling research: Can we bypass the teacher entirely?

Is it possible to design a standalone training principle that learns these fast, few-step generators directly from data? The final chapter of this monograph will address this question.

1. We will explore pioneering methods such as Consistency Models that learn the mapping from any point on an ODE trajectory to its destination point.

2. We will delve into generalized concepts of Consistency Models which learn to map any point on an ODE trajectory to another point in a single step.

This shift from improving the solver or distilling a solution to learning the solution map itself represents a significant step toward a new class of generative models that are both principled and highly efficient by design.

11

Learning Fast Generators from Scratch

Truth is ever to be found in simplicity, and not in the multiplicity and confusion of things.

Isaac Newton

In Chapter 10, we saw that slow iterative samplers in diffusion models can be compressed into few-step generators through distillation. From an engineering perspective, two-stage pipelines are practical because they divide a complex generative training task into clear, independent objectives. The first stage learns the data distribution, while the second accelerates sampling or enhances quality. This separation allows each stage to be optimized independently, making the overall system easier to manage, more stable, and more reliable.

In this chapter, however, the focus shifts to a central question driving the progress of deep generative modeling:

Question 11.0.1

Can we design a standalone generative principle that trains in a stable and efficient way, fast sampling, and allows users to easily guide or control what is produced?

In this chapter we pursue this direction and discuss an alternative approach: training few-step diffusion-based generators *without* relying on a pre-trained model. Our focus is the *flow-map model*. Here, a *flow map* is the classical solution map of an ODE, a notion rooted in differential equations and dynamical systems and closely connected, in continuum and fluid mechanics, to following the motion of material particles. Specifically, $\mathbf{\Psi}_{s\to t}$ takes a state at time s to the state reached at time t under the prescribed dynamics. A flow-map model learns to approximate this map directly, allowing a long trajectory to be traversed in one or a few

network evaluations[1]. This formulation provides a principled way to transport probability mass from the prior distribution p_{prior} to the data distribution p_{data}, while preserving the marginal distributions p_t specified by the forward diffusion process at each intermediate time.

After this chapter, you will be able to...

- distinguish an instantaneous velocity field from its ODE solution map (flow map);
- understand the semigroup property of exact ODE flows and how it motivates consistency-based learning objectives;
- understand Consistency Model, Consistency Trajectory Model, and Mean Flow as representative ways to learn special or general flow maps;
- distinguish oracle flow-map identities from their practical approximations, and understand how suitable parametrizations enable few-step generation while retaining access to local velocity information.

Reader route. The central object in this chapter is the ODE flow map. The individual methods should be viewed as different parametrizations and training strategies for learning this same underlying object.

[1] See Equation (4.2.2) and the surrounding discussion for the mathematical and historical background of flow maps.

11.1 Prologue

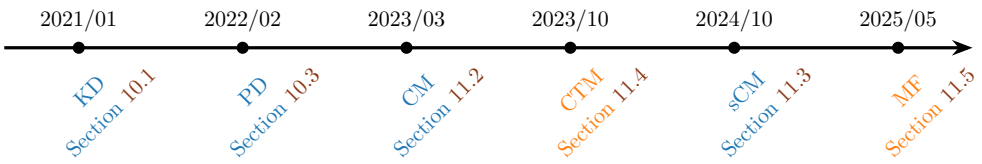


Figure 11.1: Timeline of diffusion-based methods for learning flow maps. The underlying flow map is the classical ODE solution map; the dates shown here refer only to modern machine-learning methods that distill, approximate, or learn this map. We use blue for the special case $\boldsymbol{\Psi}_{s\to 0}$ and orange for the general map $\boldsymbol{\Psi}_{s\to t}$.

Source: Created by the authors.

Motivation of Flow Map Models. In Chapter 10 we showed how the inaccessible regression target in the oracle flow-map loss $\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta})$ (see Equation (10.1.4)) can be estimated by distilling knowledge from a pre-trained diffusion model to obtain few-step generators. This route is effective and practical: a two-stage pipeline can be engineered for robustness and often remains competitive in both data and compute efficiency.

In this chapter, we shift focus to a broader challenge at the core of deep generative modeling: *Can we establish a standalone generative principle that enables stable, scalable, and efficient training, fast sampling, and generation that can be easily steered by user intentions, without relying on a pre-trained model?* Designing such standalone principles lies at the center of generative modeling.

Diffusion models offer a useful design principle: start with a continuous-time forward process that gradually transforms data into a simple prior (noise) as a reference, and frame the modeling task as learning the reverse-time transport that restores this process to match the desired marginal distributions. This time-dependent formulation also makes it easier to steer the generation process at intermediate steps, compared to one-shot generative maps. Specializing to diffusion-motivated methods, this leads to the question:

Question 11.1.1

Can we learn the flow map $\boldsymbol{\Psi}_{s\to t}(\cdot)$ with a network $\mathbf{G}_{\boldsymbol{\theta}}(\cdot, s, t)$ (a flow map model*) without access to pre-trained models, while maintaining high-fidelity generation?*

This chapter develops methods toward this goal, organized around a single objective that also underlies distillation and provides a unified view of flow-map formulations (Boffi *et al.*, 2025; Hu *et al.*, 2025):

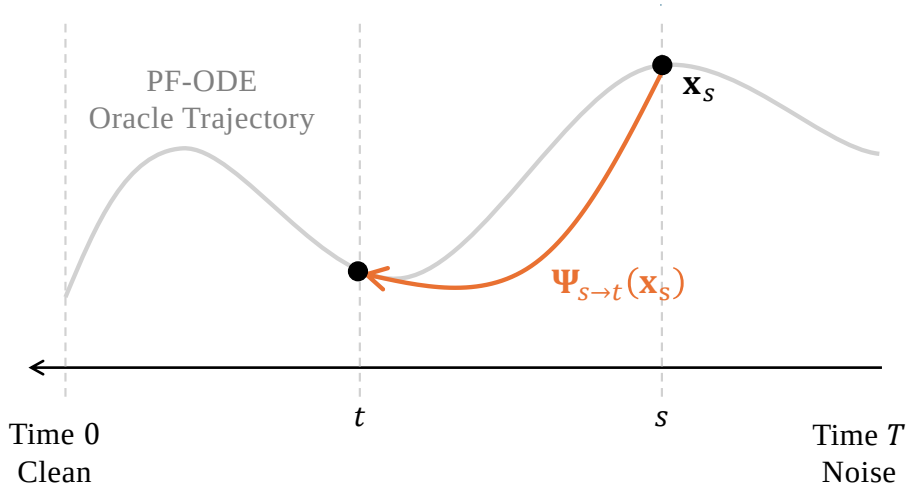


Figure 11.2: Illustration of the flow map. Starting from any state $\mathbf{x}_s$ at time s, the flow map $\mathbf{\Psi}_{s\to t}$ transports it to the corresponding point on the same PF-ODE oracle trajectory at time t.

Source: Created by the authors.

$$\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta}) := \mathbb{E}_{s,t}\,\mathbb{E}_{\mathbf{x}_s\sim p_s}\Big[w(s,t)d\big(\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s,s,t),\mathbf{\Psi}_{s\to t}(\mathbf{x}_s)\big)\Big]. \tag{10.1.4}$$

Here s,t are sampled from some time distribution (e.g., uniform), $w(s,t)\geq 0$ assigns weights to the time pairs (s,t), and $d(\cdot,\cdot)$ is a discrepancy measure such as the squared ℓ_2 norm. The oracle flow map $\mathbf{\Psi}_{s\to t}$ represents the ideal transformation that takes a state $\mathbf{x}_s$ at time s and transports it directly to time t:

$$\mathbf{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u,u)\,\mathrm{d}u,$$

where the oracle drift is given as

$$\mathbf{v}^*(\mathbf{x}_u,u) = \mathbb{E}\left[\alpha'_u\mathbf{x}_0 + \sigma'_u\boldsymbol{\epsilon}|\mathbf{x}_u\right],$$

while equivalent parametrizations are also possible (see Chapter 6), with common choices including the $\mathbf{x}$-prediction and $\mathbf{v}$-prediction forms.

At the optimum of the oracle loss, the learned model recovers the true flow map exactly:

$$\mathbf{G}^*(\mathbf{x}_s,s,t) = \mathbf{\Psi}_{s\to t}(\mathbf{x}_s), \quad \text{for all } s,t, \text{ and } \mathbf{x}_s\sim p_s.$$

Because the flow map $\mathbf{\Psi}_{s\to t}$ cannot be expressed in closed form, it must be approximated. One option, discussed in Chapter 10, is to rely on a pre-trained diffusion model. Alternatively, as we will see in this chapter, new and more

tractable surrogates can be introduced. For clarity, existing approaches can be broadly categorized according to whether the training procedure queries a teacher during the loop: *distillation*, which explicitly calls a teacher model, and *training from scratch*, which avoids teacher calls by constructing self-contained surrogates.

Building on this principled objective, we now turn to systematic approaches for learning flow map models, with the aim of developing methods that are practical while also producing generations that more accurately reflect the true data distribution and are computationally efficient. We begin with a high-level introduction to this paradigm.

Special Flow Map: Consistency Functions. *Consistency Models* (Song *et al.*, 2023) represent one of the earliest pioneering approaches to flow-map learning. They learn a few-step denoiser $\mathbf{f}_{\boldsymbol{\theta}}(\cdot, s)$ that approximates the special case of the flow map to the origin:

$$\boldsymbol{\Psi}_{s\to 0}(\cdot), \qquad s \in (0, T].$$

The key idea is that every state $\mathbf{x}_s$ is mapped to the time-0 endpoint $\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)$ of the PF-ODE trajectory passing through it. Formally, the oracle training objective for the CM family (Song *et al.*, 2023; Song and Dhariwal, 2024; Geng *et al.*, 2024; Lu and Song, 2024) is

$$\mathcal{L}_{\text{oracle-CM}}(\boldsymbol{\theta}) := \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} \left[w(s) d\left(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)\right)\right]. \tag{11.1.0}$$

In practice, however, the oracle $\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)$ is unavailable. It is therefore replaced by a *stop-gradient* target, denoted as $\mathbf{f}_{\boldsymbol{\theta}^-}$, taken from a slightly earlier step $\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s)$ on the same trajectory:

$$\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s) \approx \mathbf{f}_{\boldsymbol{\theta}^-}\left(\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s),\, s-\Delta s\right), \quad \Delta s > 0,$$

where $\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s)$ itself must also be approximated. Two practical strategies are available: (i) *distillation*, which relies on a pre-trained diffusion model, and (ii) *training from scratch*, which uses a one-point estimate without teacher guidance.

General Flow Map. Two representative approaches are the *Consistency Trajectory Model* (CTM) and *Mean Flow* (MF).

Consistency Trajectory Models. *Consistency Trajectory Model* (CTM) (Kim *et al.*, 2024a) is the first work to learn the general flow map $\boldsymbol{\Psi}_{s\to t}$ for arbitrary start and end times, and can be viewed as a concrete instance under the unified objective of Equation (10.1.4). CTM adopts an Euler-inspired parametrization by expressing the oracle flow map as

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\, \mathrm{d}u = \frac{t}{s}\mathbf{x}_s + \frac{s-t}{s} \underbrace{\left[\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\, \mathrm{d}u\right]}_{\approx\, \mathbf{g}_{\boldsymbol{\theta}}},$$

which motivates the neural parameterization

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) := \frac{t}{s}\,\mathbf{x}_s + \frac{s-t}{s}\,\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t),$$

where $\mathbf{g}_{\boldsymbol{\theta}}$ is a neural network trained so that $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) \approx \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$.

Since the oracle $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$ is inaccessible, CTM trains against a *stop-gradient* target evaluated at an intermediate time u:

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) \approx \mathbf{G}_{\boldsymbol{\theta}^-}\big(\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s),\, u,\, t\big), \qquad u \in [t, s],$$

where the intermediate state $\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s)$ is approximated in one of two ways: (i) *distillation*, which uses a few-step solver applied to a pre-trained diffusion teacher, or (ii) *training from scratch*, which constructs a self-induced teacher directly through the $\mathbf{G}_{\boldsymbol{\theta}}$ parametrization.

Mean Flow. Mean Flow (MF) (Geng *et al.*, 2025a) builds on flow matching by modeling the *average drift* over an interval $[t, s]$ (with $t \le s$):

$$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u,$$

also aligning with Equation (10.1.4). Differentiating the identity

$$(t-s)\,\mathbf{h}^*(\mathbf{x}_s, s, t) = \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u$$

with respect to s yields a self-referential relation that motivates the MF objective

$$\mathcal{L}_{\mathrm{MF}}(\boldsymbol{\theta}) := \mathbb{E}_s\,\mathbb{E}_{\mathbf{x}_s\sim p_s}\Big[w(s)\,\big\|\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \mathbf{h}^{\mathrm{tgt}}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s, t)\big\|_2^2\Big],$$

with stop-gradient target

$$\mathbf{h}^{\mathrm{tgt}}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s, t) := \mathbf{v}^*(\mathbf{x}_s, s) - (s-t)\,\big((\partial_{\mathbf{x}}\mathbf{h}_{\boldsymbol{\theta}^-})\,\mathbf{v}^*(\mathbf{x}_s, s) + \partial_s \mathbf{h}_{\boldsymbol{\theta}^-}\big)\,.$$

In practice, the oracle velocity $\mathbf{v}^*(\mathbf{x}_s, s)$ must also be approximated. Two common strategies are: (i) *distillation*, which leverages a pre-trained diffusion model trained with flow matching, or (ii) *training from scratch*, which uses the one-point conditional velocity $\alpha_s'\,\mathbf{x}_0 + \sigma_s'\,\boldsymbol{\epsilon}$ derived from the forward corruption process $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$.

Relationship Between CTM and MF. CTM and MF approximate the same path integral but parameterize different surrogates of it:

$$\begin{aligned}\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) &:= \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \\ &= \frac{t}{s}\,\mathbf{x}_s + \frac{s-t}{s}\,\underbrace{\Big[\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u\Big]}_{\approx\, \mathbf{g}_{\boldsymbol{\theta}}} \\ &= \mathbf{x}_s + (t-s)\,\underbrace{\Big[\frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u\Big]}_{\approx\, \mathbf{h}_{\boldsymbol{\theta}}}.\end{aligned}$$

In words, CTM learns an *slope displacement* through $\mathbf{g}_{\boldsymbol{\theta}}$, while MF learns the *average drift* $\mathbf{h}_{\boldsymbol{\theta}}$; both are consistent ways to approximate the same integral that defines $\boldsymbol{\Psi}_{s\to t}$.

What Happens Next? We begin with the CM family, which focuses on the specific flow map $\boldsymbol{\Psi}_{s\to 0}$. This part covers both its discrete time origin in Section 11.2 and its continuous time extension in Section 11.3. We then move on to the general flow map and provide a detailed discussion of two key representatives, CTM and MF. Their parameterizations, training strategies, and practical approximations are presented in Section 11.4 and Section 11.5, respectively.

We remark that the *Elucidating Diffusion Model* (EDM) introduced in Section D.6 offers systematic guidelines for designing the network parameterization of the $\mathbf{x}$-prediction model and has demonstrated strong empirical performance. Although this section can be considered optional, the EDM formulation serves as a valuable foundation for CM-style models.

For clarity of exposition later on, we do not strictly follow the chronological order in which these approaches appeared. Instead, we organize the discussion by conceptual relationships. Nevertheless, to acknowledge originality and respect chronology, we provide the historical timeline in Figure 11.1.

11.2 Special Flow Map: Consistency Model in Discrete Time

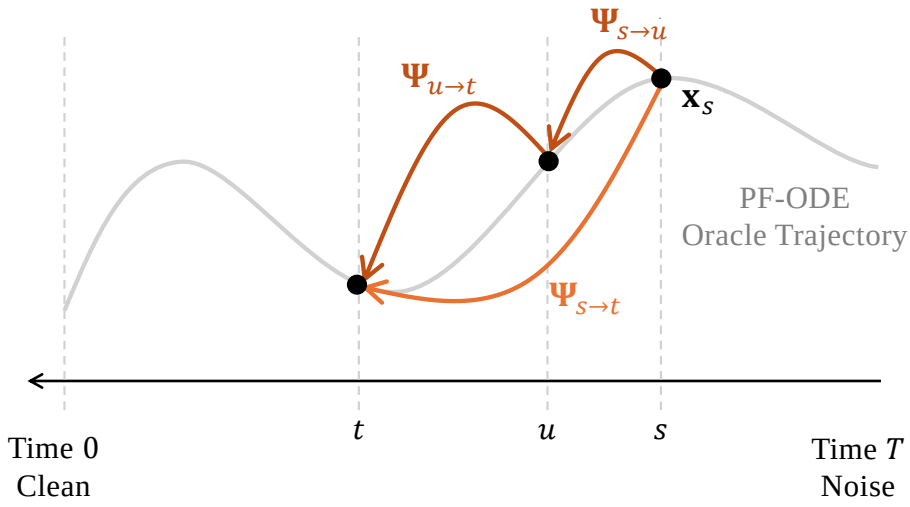


Figure 11.3: Illustration of the flow map semigroup property. This property states that transitioning from s to u and then from u to t is equivalent to transitioning directly from s to t.

Source: Created by the authors.

An Important Principle of Flow Maps: The Semigroup Property. Consistency Models (introduced in Sections 11.2 and 11.3) and their generalization, the Consistency Trajectory Model (Section 11.4), define their regression targets by exploiting a key mathematical structure of flow maps. This structure is the fundamental *semigroup property*:

$$\boldsymbol{\Psi}_{u\to t} \circ \boldsymbol{\Psi}_{s\to u} = \boldsymbol{\Psi}_{s\to t}, \quad \boldsymbol{\Psi}_{s\to s} = \mathbf{I}, \quad \text{for all } s,u,t \in [0,T]. \tag{11.2.1}$$

Intuitively, this means that if we first evolve a state from s to u (through $\boldsymbol{\Psi}_{s\to u}$) and then from u to t (through $\boldsymbol{\Psi}_{u\to t}$), we end up at exactly the same point as if we had evolved directly from s to t. This is nothing more than the basic principle of ODE solving[2]: once the starting point of a flow is specified, its future evolution is completely determined, and it follows a single well-defined path. Whether we follow this path in one long step or divide it into smaller intervals, we still move along the same trajectory and arrive at the same final state.

To build further intuition for the semigroup property, consider the solution

[2]The semigroup property follows from the uniqueness theorem for ODE initial value problems (see Chapter A).

trajectory $\{\mathbf{x}(s)\}_{s\in[0,T]}$ of the PF-ODE

$$\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} = \mathbf{v}^*(\mathbf{x}(\tau), \tau),$$

with a fixed initial condition $\mathbf{x}(T)$ at time T, solved backward in time. If we fix the terminal time at $t = 0$, the corresponding flow map can be written more simply as

$$\mathbf{f}^*(\cdot, s) := \mathbf{\Psi}_{s\to 0}(\cdot),$$

which is referred to as the *consistency function*. By construction, this function inherits several fundamental properties directly from the semigroup identity of Equation (11.2.1) with $t = 0$:

(i) **Global Consistency:** every point along the trajectory maps to the same clean endpoint,

$$\mathbf{f}^*(\mathbf{x}(s), s) = \mathbf{x}(0), \quad \text{for all } \ s \in [0, T].$$

This is because

$$\begin{aligned}\mathbf{f}^*(\mathbf{x}(s), s) &= \mathbf{\Psi}_{s\to 0}(\mathbf{\Psi}_{0\to s}(\mathbf{x}(0))) = (\mathbf{\Psi}_{s\to 0} \circ \mathbf{\Psi}_{0\to s})(\mathbf{x}(0)) \\ &= \mathbf{\Psi}_{0\to 0}(\mathbf{x}(0)) = \mathbf{x}(0).\end{aligned}$$

(ii) **Self Consistency:** any two points along the same trajectory must give identical outputs,

$$\mathbf{f}^*(\mathbf{x}(s), s) = \mathbf{f}^*(\mathbf{x}(u), u), \quad \text{for all } \ s, u \in [0, T]. \tag{11.2.2}$$

This is a direct re-interpretation of the semigroup identity: $\Psi_{s\to 0} \circ \Psi_{0\to s} = \Psi_{u\to 0} \circ \Psi_{0\to u}$.

(iii) **Local Consistency:** the consistency function is invariant with respect to s when evaluated along the trajectory,

$$\frac{\mathrm{d}}{\mathrm{d}s}\,\mathbf{f}^*(\mathbf{x}(s), s) = 0, \qquad \mathbf{f}^*(\mathbf{x}(0), 0) = \mathbf{x}(0). \tag{11.2.3}$$

This follows from global consistency, which states that $\mathbf{f}^*(\mathbf{x}(s), s)$ does not change with s along the trajectory.

The three properties are all equivalent. Each states that along any solution trajectory $s \mapsto \mathbf{x}(s)$, the flow-to-origin/consistency map $\mathbf{f}^*(\mathbf{x}(s), s) = \mathbf{\Psi}_{s\to 0}(\mathbf{x}(s))$ yields the same terminal point $\mathbf{x}(0)$, independent of the starting time.

Goal of Consistency Models. A CM aims to train a neural network $\mathbf{f}_{\boldsymbol{\theta}}\colon \mathbb{R}^D \times [0,T] \to \mathbb{R}^D$ to approximate the special flow map $\boldsymbol{\Psi}_{s\to 0}$, i.e., consistency function[3]. The key idea is to enforce consistency along PF-ODE trajectories: states lying on the same ODE trajectory should map to the same time-0 endpoint (more precisely, this corresponds to the special case $t=0$ and $u=s-\Delta s$ in Equation (11.2.1)).

There are, however, multiple ways to realize this goal. The choice depends on whether a pre-trained diffusion model is available and whether training is carried out in a discrete-time or continuous-time regime. We begin by summarizing these variants in Table 11.1 and illustrating their objectives in Figure 11.4. The subsequent sections (Sections 11.2 and 11.3) then gradually develop the details of each approach.

Table 11.1: Training Objectives of Consistency Models

	Distillation	**From Scratch**
Discrete-time	Equation (11.2.4)	Equation (11.2.6)
Continuous-time	Equation (11.3.5)	Equation (11.3.6)

11.2.1 Discrete-Time Approximations for Learning a Consistency Function

In principle, a consistency function can be learned by minimizing the oracle loss Equation (11.1.0):

$$\mathcal{L}_{\text{oracle-CM}}(\boldsymbol{\theta}) := \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s\sim p_s}\left[w(s)d\left(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s,s), \boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)\right)\right].$$

This objective enforces that every noisy sample $\mathbf{x}_s$ is mapped back to its clean endpoint $\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)$.

The challenge is that the oracle map $\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s)$ is not available in practice. To overcome this, Song *et al.* (2023) exploit the *semigroup property*: any noisy state and its consecutive step along the same PF-ODE trajectory must map to the same clean endpoint. Concretely, the oracle target is replaced by a *stop-gradient* target taken from a slightly earlier point on the trajectory:

$$\begin{aligned}\boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s) &= \boldsymbol{\Psi}_{s-\Delta s\to 0}\left(\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s)\right)\\ &\approx \mathbf{f}_{\boldsymbol{\theta}^-}\left(\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s),\, s-\Delta s\right), \qquad \Delta s>0,\end{aligned}$$

[3]The concept of a consistency function for an ODE generalizes to a function $\mathbf{f}(\mathbf{x},t)$ for an SDE such that $\mathbf{f}(\mathbf{x}_t,t)$ is a (local) martingale with respect to the SDE's natural filtration, i.e.

$$\mathbb{E}[\mathbf{f}(\mathbf{x}_t,t)|\mathbf{x}_s] = \mathbf{f}(\mathbf{x}_s,s), \quad \text{for all} \quad t\geq s.$$

This generalization was proposed/observed in (Daras *et al.*, 2023; Lai *et al.*, 2023a), and the theoretical connections are summarized in (Lai *et al.*, 2023b).

where $\boldsymbol{\theta}^-$ are parameters under the stop-gradient operator. A further difficulty is that the intermediate state $\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s)$ has no closed form either and must itself be approximated. Two practical regimes have been proposed:

With Pre-trained Diffusion Model (Consistency Distillation). Suppose we have access to a pre-trained diffusion model. Consistency Distillation (CD) leverages the teacher model to approximate the intermediate state $\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s)$ by simulating only a single backward ODE step:

$$\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s) \approx \texttt{Solver}_{s\to s-\Delta s}(\mathbf{x}_s).$$

More concretely, a pre-trained diffusion model provides an estimate of the score function $\mathbf{s}_{\phi^\times}(\mathbf{x}_s, s) \approx \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)$. Using this, one can perform a one-step DDIM update from $\mathbf{x}_s$ to obtain an approximation of the state at $s' = s - \Delta s$:

$$\begin{aligned}
\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s) &\approx \frac{\alpha_{s'}}{\alpha_s}\,\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) \\
&\approx \frac{\alpha_{s'}}{\alpha_s}\,\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\mathbf{s}_{\phi^\times}(\mathbf{x}_s, s) \\
&:= \tilde{\mathbf{x}}_{s'}^{\phi^\times}.
\end{aligned}$$

Combining this construction with the stop-gradient target yields a practical *discrete-time* proxy for the oracle loss $\mathcal{L}_{\text{oracle-CM}}(\boldsymbol{\theta})$. Formally, over a partition $0 = s_1 < s_2 < \cdots < s_N = T$, the CD training objective is given by

$$\mathcal{L}_{\text{CD}}^N(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi}^\times) := \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},i}\Big[\,\omega(s_i)\, d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_{s_{i+1}}, s_{i+1}),\, \mathbf{f}_{\boldsymbol{\theta}^-}(\tilde{\mathbf{x}}_{s_i}^{\phi^\times}, s_i)\big)\Big]. \quad (11.2.4)$$

Here, $\omega(\cdot)$ is a time-dependent weight, $d(\cdot,\cdot)$ is a distance measurement, and $\boldsymbol{\theta}^-$ indicates stop-gradient parameters, which prevent collapse to trivial solutions (e.g., constant predictions).

Without Pre-Trained Diffusion Model (Consistency Training). When no pre-trained diffusion model is available, the oracle score $\nabla_{\mathbf{x}} \log p_s(\mathbf{x}_s)$ can still be estimated directly using a simple one-point approximation (albeit with high variance). Recall that it admits the conditional expectation form:

$$\begin{aligned}
\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) &= \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0|\mathbf{x}_s)}\left[\nabla_{\mathbf{x}_s}\log p(\mathbf{x}_s|\mathbf{x}_0)\right] \\
&= \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0|\mathbf{x}_s)}\left[-\frac{\mathbf{x}_s - \alpha_s\mathbf{x}_0}{\sigma_s^2}\right].
\end{aligned}$$

The identity above suggests a simple one-sample estimator. If $\mathbf{x}_s$ is obtained from a paired sample $(\mathbf{x}_0, \boldsymbol{\epsilon})$ via $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$, then

$$\widehat{\nabla_{\mathbf{x}}\log p_s}(\mathbf{x}_s) := -\frac{\boldsymbol{\epsilon}}{\sigma_s} = -\frac{\mathbf{x}_s - \alpha_s\mathbf{x}_0}{\sigma_s^2}$$

serves as an *unbiased* estimator of the score at $\mathbf{x}_s$ (conditionally unbiased with respect to $p(\mathbf{x}_0|\mathbf{x}_s)$). It corresponds exactly to the *conditional score* used as the regression target in denoising score matching.

Plugging this estimate into the DDIM one-step update from s to $s' = s - \Delta s$ (see Equation (9.2.2)) yields

$$\begin{aligned}
\boldsymbol{\Psi}_{s\to s'}(\mathbf{x}_s) &\approx \frac{\alpha_{s'}}{\alpha_s}\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) \quad \text{(DDIM)} \\
&\approx \frac{\alpha_{s'}}{\alpha_s}\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\widehat{\nabla_{\mathbf{x}_s}\log p_s}(\mathbf{x}_s) \quad \text{(1-pt score)} \\
&= \frac{\alpha_{s'}}{\alpha_s}\mathbf{x}_s - \left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)(\mathbf{x}_s - \alpha_s\mathbf{x}_0) \\
&= \alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon},
\end{aligned} \tag{11.2.5}$$

where[4] $\mathbf{x}_0$ is the same data sample and $\boldsymbol{\epsilon}$ is the same Gaussian noise used to construct $\mathbf{x}_s$.

This leads to a teacher-free discrete-time surrogate of the oracle objective $\mathcal{L}_{\text{oracle-CM}}$, written as

$$\mathcal{L}_{\text{CT}}^N(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},i}\left[\omega(s_i)\, d\left(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_{s_{i+1}}, s_{i+1}),\, \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s_i}, s_i)\right)\right], \tag{11.2.6}$$

with $\mathbf{x}_{s_i} = \alpha_{s_i}\mathbf{x}_0 + \sigma_{s_i}\boldsymbol{\epsilon}$ and $\mathbf{x}_{s_{i+1}} = \alpha_{s_{i+1}}\mathbf{x}_0 + \sigma_{s_{i+1}}\boldsymbol{\epsilon}$.

Using $\alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon}$ directly as an approximation of $\boldsymbol{\Psi}_{s\to s'}(\mathbf{x}_s)$ without expectation introduces high variance[5]. Recall, however, the analogous conditional-expectation argument in denoising score matching (see Section 6.1). Here, the one-point state estimate is unbiased before it is passed through the nonlinear target network and discrepancy, so the resulting losses need not agree exactly. Conditioning on $\mathbf{x}_s$, the first-order error cancels in expectation; under suitable smoothness and moment assumptions, the remaining discrepancy is $\mathcal{O}(\Delta s^2)$, as formalized below.

[4] The last identity follows directly from the forward corruption process $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$ by elementary algebra.

[5] The one-point (conditional) score estimate $\widehat{\nabla_{\mathbf{x}}\log p_s}(\mathbf{x}_s)$ can be viewed as a one-sample Monte Carlo estimator, which is *conditionally unbiased* given $\mathbf{x}_s$: averaging this estimator over the (generally intractable) clean posterior $p(\cdot|\mathbf{x}_s)$ recovers the true score as

$$\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) = \mathbb{E}_{\mathbf{x}_0\sim p(\cdot|\mathbf{x}_s)}\left[\widehat{\nabla_{\mathbf{x}}\log p_s}(\mathbf{x}_s)\right].$$

Theorem 11.2.1: CM-CT Equivalence up to Error $\mathcal{O}(\Delta s^2)$

Let $s' := s - \Delta s$, and define

$$\mathcal{L}_{\text{CM}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\big[w(s)\, d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s'}^{\text{DDIM}}, s'))\big],$$
$$\mathcal{L}_{\text{CT}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\big[w(s) d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s'}, s'))\big],$$

where

$$\mathbf{x}_{s'}^{\text{DDIM}} := \frac{\alpha_{s'}}{\alpha_s}\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s)$$

is the oracle DDIM update. Both $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$ and $\mathbf{x}_{s'} = \alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon}$ share the same pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$ with $\mathbf{x}_0 \sim p_{\text{data}}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Then,

$$\mathcal{L}_{\text{CM}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \mathcal{L}_{\text{CT}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) + \mathcal{O}(\Delta s^2).$$

Proof for Theorem.

First, note that the DDIM update with the oracle score equals the conditional mean,

$$\mathbf{x}_{s'}^{\text{DDIM}} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s],$$

which can also be verified from Equation (11.2.5) by taking the expectation over $p(\cdot|\mathbf{x}_s)$. Next, perform a Taylor expansion of

$$d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\cdot, s'))$$

around $\mathbf{x}_{s'}^{\text{DDIM}} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s]$. The one-point update recovers $\mathbf{x}_{s'} = \alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon}$ pathwise for the same $(\mathbf{x}_0, \boldsymbol{\epsilon})$. Since

$$\mathbb{E}[\mathbf{x}_{s'} - \mathbf{x}_{s'}^{\text{DDIM}} \mid \mathbf{x}_s] = 0,$$

the linear Taylor term vanishes after conditioning on $\mathbf{x}_s$. Under the smoothness and moment assumptions described above, the remaining expected Taylor remainder is $\mathcal{O}(\Delta s^2)$. Hence, $\mathcal{L}_{\text{CT}} = \mathcal{L}_{\text{CM}} + \mathcal{O}(\Delta s^2)$. A detailed derivation is provided in Section D.5.

In summary, CD leverages a teacher model for initialization and supervision, which often stabilizes optimization and reduces variance. In contrast, Consistency Training (CT) requires no pre-trained model and can therefore be trained entirely from scratch. Despite this difference, CT serves as a fully standalone generative model.

Practical Considerations. In practice, Song *et al.* (2023) adopt the EDM formulation of Karras *et al.* (2022) (see Section D.6) with the forward corruption kernel $\mathbf{x}_s = \mathbf{x}_0 + s\boldsymbol{\epsilon}$, and use the neural network parameterization proposed therein (cf.

Equation (D.6.1)):

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}, s) = c_{\text{skip}}(s)\mathbf{x} + c_{\text{out}}(s)\,\mathbf{F}_{\boldsymbol{\theta}}\left(c_{\text{in}}(s)\,\mathbf{x},\, c_{\text{noise}}(s)\right),$$

where $\mathbf{F}_{\boldsymbol{\theta}}$ is a neural network and the coefficients follow Equation (D.6.5). This parameterization has the important boundary property

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}, 0) = \mathbf{x},$$

which enforces consistency at time zero and ensures the network output matches its input when no noise is present.

11.2.2 Sampling with Consistency Model

Once a consistency model $\mathbf{f}_{\boldsymbol{\theta}^\times}$ is trained, either in continuous or discrete time, it can be used to generate samples in a single step or a few steps. The algorithm is summarized in Algorithm 9.

One-Step Generation. Given an initial latent $\hat{\mathbf{x}}_T$ sampled from the prior distribution (in practice, $\mathcal{N}(\mathbf{0}, T^2\mathbf{I})$), a clean sample can be generated via a single function evaluation: $\mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_T, T)$.

Multi-Step Generation. With pre-selected timesteps $T > \tau_1 > \tau_2 > \cdots > \tau_{M-1} = 0$, start from initial noise $\hat{\mathbf{x}}_T$ and alternate between noise injection and large clean jumps via the consistency model at earlier time points, gradually refining the sample:

$$\hat{\mathbf{x}}_T \xrightarrow[\text{get a clean}]{\text{long jump}} \mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_T, T) \xrightarrow[\text{to level } \tau_1]{\text{add noise}} \hat{\mathbf{x}}_{\tau_1} \xrightarrow[\text{get a clean}]{\text{long jump}} \mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_{\tau_1}, \tau_1) \xrightarrow[\text{to level } \tau_2]{\text{add noise}} \cdots.$$

Algorithm 9 CM's Sampling with One-Step or Multi-Step Generation

Input: Consistency model $\mathbf{f}_{\boldsymbol{\theta}^\times}(\cdot,\cdot)$, sequence of time points $T > \tau_1 > \tau_2 > \cdots > \tau_{M-1} = 0$, initial noise $\hat{\mathbf{x}}_T$

if one-step **then**
  $\mathbf{x} \leftarrow \mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_T, T)$
else
  $\mathbf{x} \leftarrow \mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_T, T)$
  for $m = 1$ **to** $M - 1$ **do**
    Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\hat{\mathbf{x}}_{\tau_m} \leftarrow \alpha_{\tau_m}\mathbf{x} + \sigma_{\tau_m}\boldsymbol{\epsilon}$
    $\mathbf{x} \leftarrow \mathbf{f}_{\boldsymbol{\theta}^\times}(\hat{\mathbf{x}}_{\tau_m}, \tau_m)$
  end for
end if

Output: $\mathbf{x}$

11.3 Special Flow Map: Consistency Model in Continuous Time

We now move beyond the discrete-time setting of consistency models and consider a continuous-time perspective. Unlike the discrete approach, which fixes a time grid and trains only on those sampled points, the continuous formulation treats the flow map as defined for all times. This shift eliminates the dependence on an arbitrary discretization and provides a more principled alignment with the underlying dynamics. It also helps reduce the approximation errors that naturally arise from discretized integration, and ensures consistency is enforced globally rather than only at selected steps.

11.3.1 Continuous-Time Consistency Model

To motivate the continuous time formulation, we first revisit Equation (11.2.3), which describes the condition under which time derivatives can be taken. Using the chain rule, we arrive at

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}^*(\mathbf{x}(s),s)=0 \\ \iff &\left(\nabla_{\mathbf{x}}\mathbf{f}^*\right)(\mathbf{x}(s),s)\cdot \underbrace{\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{x}(s)}_{\text{ODE velocity}} + \left(\frac{\partial}{\partial s}\mathbf{f}^*\right)(\mathbf{x}(s),s)=0, \end{aligned} \tag{11.3.1}$$

where the trajectory $\mathbf{x}(s)$ follows the PF-ODE

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{x}(s)=\mathbf{v}^*(\mathbf{x}(s),s).$$

This relationship shows that the consistency function $\mathbf{f}^*$ remains constant along any solution trajectory of the ODE. The velocity field $\mathbf{v}^*$ can be estimated in practice either from a pre-trained diffusion model (when such a model is available) or from a direct one-point approximation, such as $\alpha'_s\mathbf{x}_0+\sigma'_s\boldsymbol{\epsilon}$, as explained in Section 11.2.

Equation (11.3.1) suggests a natural continuous-time characterization of consistency along PF-ODE trajectories. One possible approach is to enforce this differential condition directly by minimizing the residual, in a manner similar to physics-informed neural networks (PINNs) (Raissi, 2018; Boffi *et al.*, 2025):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{s,\mathbf{x}_s\sim p_s}\left[\left\|\partial_s\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s,s)+\left(\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s,s)\right)\mathbf{v}^*(\mathbf{x}_s,s)\right\|_2^2\right].$$

In practice, however, Song *et al.* (2023) and Lu and Song (2024) employ teacher-based consistency objectives rather than directly optimizing this residual. This motivates asking whether the discrete consistency loss admits a meaningful infinitesimal counterpart as $\Delta s\to 0$. The proposition below shows that, at the comparison point where the stop-gradient target coincides with the online network,

the scaled discrete-time gradient converges to the gradient of a continuous-time objective:

$$\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}\left[\omega(s)\left\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s), s-\Delta s)\right\|_2^2\right]. \tag{11.3.2}$$

For a uniform partition underlying Equations (11.2.4) and (11.2.6), let

$$\Delta s_N := \frac{T}{N-1}.$$

Then $\Delta s_N \to 0$ as $N \to \infty$, so the infinitesimal limit in Equation (11.3.2) corresponds to refining these discrete objectives.

We summarize this key idea in the following proposition.

Proposition 11.3.1: Continuous-Time Consistency Training

The following convergence result holds at $\boldsymbol{\theta} = \boldsymbol{\theta}^-$:

$$\lim_{\Delta s\to 0} \frac{1}{\Delta s}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} = \nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-}.$$

Here,

$$\mathcal{L}_{\mathrm{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[2\omega(s)\,\mathbf{f}_{\boldsymbol{\theta}}^{\top}(\mathbf{x}_s, s)\cdot\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\right],$$

and the total differentiation identity is

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) = \partial_s\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s))\mathbf{v}^*(\mathbf{x}_s, s). \tag{11.3.3}$$

Proof for Proposition.

A first-order Taylor expansion of the stop-gradient target around $(\mathbf{x}_s, s)$ shows that the loss $\mathcal{L}_{\mathrm{CM}}^{\Delta s}$ behaves, up to $\mathcal{O}(\Delta s^2)$, like an inner product between the student update $\nabla_{\boldsymbol{\theta}}\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s)$ and the tangent change $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$. Evaluating at $\boldsymbol{\theta} = \boldsymbol{\theta}^-$ removes the zeroth-order discrepancy term, so the scaled gradient satisfies

$$\lim_{\Delta s\to 0} \frac{1}{\Delta s}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\Delta s}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} = \nabla_{\boldsymbol{\theta}}\mathbb{E}\left[2\omega(s)\,\mathbf{f}_{\boldsymbol{\theta}}^{\top}(\mathbf{x}_s, s)\cdot\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\right]\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-},$$

which is the claimed identity. We defer the proof to Section D.5.

The result above is written under the gradient operator $\nabla_{\boldsymbol{\theta}}$ so that terms involving $\boldsymbol{\theta}^-$ vanish, since $\boldsymbol{\theta}^-$ is treated as constant under stop-gradient. Note that $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$ denotes the total derivative along the oracle trajectory, rather than a simple partial time derivative.

In summary, the continuous-time consistency model can be trained using the gradient of the following pseudo-objective (ignoring the factor 2):

$$\mathcal{L}_{\mathrm{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}) := \mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\omega(s)\mathbf{f}_{\boldsymbol{\theta}}^{\top}(\mathbf{x}_s, s) \cdot \frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)\right]. \tag{11.3.4}$$

11.3.2 Training Continuous-Time Consistency Model

Similar to the discrete-time case discussed in Section 11.2.1, we now clarify the practical approximation of the tangent term in Equation (11.3.4), which involves the inaccessible oracle velocity $\mathbf{v}^*$:

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) = \partial_s \mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) + \left(\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)\right)\mathbf{v}^*(\mathbf{x}_s, s).$$

After training a continuous-time CM, sampling follows the same procedure as in the discrete time case (Section 11.2.2).

Continuous-Time Consistency Distillation. If a pre-trained diffusion model is available such that $\mathbf{v}_{\boldsymbol{\phi}^{\times}} \approx \mathbf{v}^*$, then the tangent vector $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)$ in Equation (11.3.3) can be approximated by the surrogate

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) \approx \partial_s \mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) + \left(\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)\right)\mathbf{v}_{\boldsymbol{\phi}^{\times}}(\mathbf{x}_s, s). \tag{11.3.5}$$

We denote the resulting objective as $\mathcal{L}_{\mathrm{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}^{\times})$. Accordingly, Proposition 11.3.1 can be restated as

$$\lim_{N\to\infty} \frac{1}{\Delta s_N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{CD}}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}^{\times}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{CD}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}^{\times}).$$

Continuous-Time Consistency Training (from Scratch). On the other hand, if a pre-trained diffusion model is not available, the oracle velocity $\mathbf{v}^*$ can be approximated using the one point conditional estimate $\alpha_s'\mathbf{x}_0 + \sigma_s'\boldsymbol{\epsilon}$. In this case, the tangent vector $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)$ in Equation (11.3.3) is replaced by the surrogate

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) \approx \partial_s \mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s) + \left(\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}^{-}}(\mathbf{x}_s, s)\right)\left(\alpha_s'\mathbf{x}_0 + \sigma_s'\boldsymbol{\epsilon}\right). \tag{11.3.6}$$

We denote the resulting objective as $\mathcal{L}_{\mathrm{CT}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-})$, which corresponds to the training from scratch setting. Accordingly, Proposition 11.3.1 can be restated as

$$\lim_{N\to\infty} \frac{1}{\Delta s_N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{CT}}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{CT}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}).$$

So far, we have introduced all the fundamental approaches listed in Table 11.1 to realize the learning of the consistency function $\boldsymbol{\Psi}_{s\to 0}$. To provide a clearer overview, Figure 11.4 summarizes the relationships among the different loss functions for training consistency functions. The figure also indicates whether each method

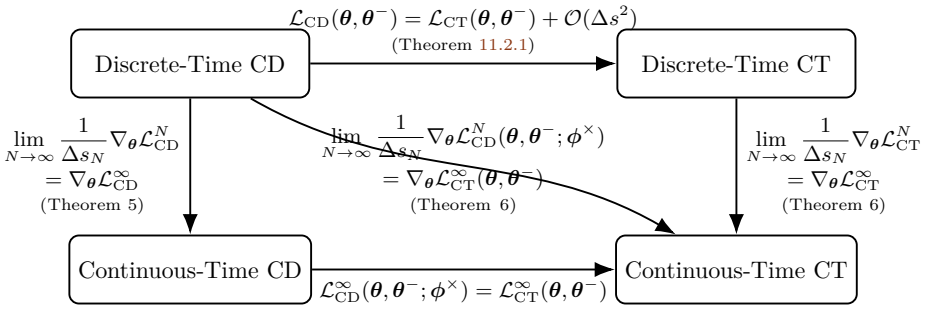


Figure 11.4: Diagram showing relationships between discrete/continuous-time CD and CT under the ℓ_2 distance metric: $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2$**.** The marked theorems follow the labeling in (Song *et al.*, 2023). Whenever the theorems involve CT, we assume a perfect score: $\mathbf{s}_{\phi^\times}(\mathbf{x},t) \equiv \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. $\mathcal{L}^\infty_{\mathrm{CT}}$ is defined in Equation (11.3.4), while $\mathcal{L}^\infty_{\mathrm{CD}}$ is defined in Equation (11.3.5).

Source: Created by the authors.

relies on a pre-trained diffusion model and distinguishes between continuous time and discrete time objectives.

However, the tangent vector $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}$ often causes instability during training. In the following optional section, we present techniques from *Simplifying, Stabilizing and Scaling Continuous Time Consistency Models* (sCM) (Lu and Song, 2024) that mitigate these issues.

11.3.3 (Optional) Practical Considerations of Continuous-Time CT

Our interest lies in the training from scratch scenario, since it yields a standalone generative model that does not rely on external pre-trained diffusion models. Hence, we focus our discussion on the continuous time case.

In practice, however, training directly with Equation (11.3.4) is often unstable, as the term $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}$ can exhibit large or unbounded time derivatives, leading to exploding gradients during optimization. To overcome this, suitable parameterizations and stabilization strategies are typically required (Geng *et al.*, 2025b; Lu and Song, 2024). As summarized in Section 6.2.2, the main factors that influence stable training include the *diffusion process*, *parameterization choices*, *time weighting function*, and *time sampling distribution*, all of which should be carefully designed and disentangled also in continuous-time CM.

Diffusion Process. Instead of using the standard diffusion parameterization $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$, Lu and Song (2024) adopt a trigonometric schedule. This schedule, although mathematically equivalent to the original form (as shown in Equation (6.3.4)), provides a cleaner structure and a better separation

in the training objective, which contributes to improved stability during training [6]. In addition, they incorporate the standard deviation σ_{d} of the data distribution p_{data}, in line with EDM's design in Section D.6.1:

$$\mathbf{x}_s := \cos(s)\mathbf{x}_0 + \sin(s)\mathbf{z}, \quad \text{where } \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{d}}^2\mathbf{I}). \tag{11.3.7}$$

This formulation is fully general. For any diffusion process of the form $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we can equivalently write:

$$\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \frac{\sigma_s}{\sigma_{\mathrm{d}}} \cdot (\sigma_{\mathrm{d}}\boldsymbol{\epsilon}),$$

by defining $\mathbf{z} := \sigma_{\mathrm{d}}\boldsymbol{\epsilon}$, $\alpha'_s := \alpha_s$, and $\sigma'_s := \frac{\sigma_s}{\sigma_{\mathrm{d}}}$. The transformed pair (α'_s, σ'_s) can then be mapped to the trigonometric form $(\cos(s), \sin(s))$ using the normalization described in Equation (6.3.5).

Parametrizations. By considering the analogous principles of EDM in Section D.6.1, Lu and Song (2024) propose the following parametrization for the neural network similar to Equation (D.6.1):

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}, s) := c_{\mathrm{skip}}(s)\mathbf{x} + c_{\mathrm{out}}(s)\mathbf{F}_{\boldsymbol{\theta}}\left(c_{\mathrm{in}}(s)\mathbf{x}, c_{\mathrm{noise}}(s)\right).$$

Here, $c_{\mathrm{skip}}(s)$, $c_{\mathrm{out}}(s)$, and $c_{\mathrm{in}}(s)$ can be derived using the same criteria presented in Section D.6.1 (see Appendix B of Lu and Song (2024) for detailed derivations), and are given by

$$c_{\mathrm{skip}}(s) = \cos(s), \quad c_{\mathrm{out}}(s) = -\sigma_{\mathrm{d}}\sin(s), \quad c_{\mathrm{in}}(s) \equiv \frac{1}{\sigma_{\mathrm{d}}}.$$

This is considered along with the default choice $c_{\mathrm{noise}}(s) = s$, where $\partial_s c_{\mathrm{noise}}(s)$ is bounded to ensure training stability, as will be discussed around Equation (11.3.10). This leads to the following parametrization under the trigonometric schedule:

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}, s) = \cos(s)\mathbf{x} - \sin(s)\sigma_{\mathrm{d}}\mathbf{F}_{\boldsymbol{\theta}}\left(\frac{\mathbf{x}}{\sigma_{\mathrm{d}}}, c_{\mathrm{noise}}(s)\right). \tag{11.3.8}$$

We note that this parametrization also ensures that the neural network always satisfies the boundary condition $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}, 0) \equiv \mathbf{x}$ for all $\mathbf{x}$, which is an essential property of a consistency function.

Techniques for Stabilizing Tangent Training. Under the trigonometric schedule and the network parametrization described in Equation (11.3.8), the gradient of the loss in Equation (11.3.4) becomes

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\infty}_{\mathrm{CT}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \nabla_{\boldsymbol{\theta}}\mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[-\omega(s)\sigma_{\mathrm{d}}\sin(s)\mathbf{F}_{\boldsymbol{\theta}}^{\top}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) \cdot \frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}(\mathbf{x}_s, s)\right]. \tag{11.3.9}$$

[6] Intuitively, both the trigonometric functions and their derivatives are bounded, which helps prevent scale explosion in terms like $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}$. A detailed discussion is provided later.

In theory, training with the gradient update in Equation (11.3.9) may be sufficient to learn a consistency function. However, Lu and Song (2024) empirically observed that the training process can become unstable in practice due to the behavior of the tangent function, given by

$$\underbrace{\frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)}{\mathrm{d}s}}_{\textbf{A.}} =$$

$$-\cos(s)\left(\sigma_{\mathrm{d}}\mathbf{F}_{\boldsymbol{\theta}^-}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) - \frac{\mathrm{d}\mathbf{x}_s}{\mathrm{d}s}\right) - \sin(s)\left(\mathbf{x}_s + \sigma_{\mathrm{d}}\frac{\mathrm{d}\mathbf{F}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, c_{\mathrm{noise}}(s)\right)\right).$$

In particular, instability was observed in the term

$$\underbrace{\sin(s)\frac{\mathrm{d}\mathbf{F}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, c_{\mathrm{noise}}(s)\right)}_{\textbf{B.}} = \sin(s)\nabla_{\mathbf{x}_s}\mathbf{F}_{\boldsymbol{\theta}^-}\frac{\mathrm{d}\mathbf{x}_s}{\mathrm{d}s} + \sin(s)\partial_s\mathbf{F}_{\boldsymbol{\theta}^-}.$$

More specifically, the instability arises from the component

$$\sin(s)\partial_s\mathbf{F}_{\boldsymbol{\theta}^-} = \sin(s)\underbrace{\frac{\partial c_{\mathrm{noise}}(s)}{\partial s}\cdot\frac{\partial \mathrm{emb}(c_{\mathrm{noise}})}{\partial c_{\mathrm{noise}}}}_{\textbf{C.}}\cdot\frac{\partial\mathbf{F}_{\boldsymbol{\theta}^-}}{\partial\mathrm{emb}(c_{\mathrm{noise}})}. \tag{11.3.10}$$

Here, we follow a common practice in the DM and CM literature by applying a positional or Fourier embedding, denoted by $\mathrm{emb}(\cdot)$, to the time variable $c_{\mathrm{noise}}(s)$:

$$s \mapsto c_{\mathrm{noise}}(s) \mapsto \mathrm{emb}(c_{\mathrm{noise}}(s)) \mapsto \mathbf{F}_{\boldsymbol{\theta}^-}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, \mathrm{emb}(c_{\mathrm{noise}}(s))\right).$$

Therefore, some additional empirical techniques are introduced to mitigate the instability:

- **A. Tangent Normalization.** Explicitly normalize the tangent function by replacing $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\theta^-}$ with $\frac{\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\theta^-}}{\left\|\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\theta^-}\right\|_2+c}$, where $c > 0$ is a constant set empirically. Alternatively, clipping the tangent within $[-1, 1]$ can also effectively cap its variance.

- **B. Tangent Warm-Up.** Since the term $\sin(s)(\mathbf{x}_s + \sigma_{\mathrm{d}}\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{F}_{\boldsymbol{\theta}^-})$ may induce instability, an optional technique can be applied by replacing the coefficient $\sin(s)$ with $r \cdot \sin(s)$, where r linearly increases from 0 to 1 over the first few training iterations.

- **C. Time Embedding.** In light of the derivative chain in Equation (11.3.10), Lu and Song (2024) opted for a smaller magnitude parameter to control the derivative $\frac{\partial\mathrm{emb}(c_{\mathrm{noise}})}{\partial c_{\mathrm{noise}}}$. For a similar reason, $c_{\mathrm{noise}}(s) = s$ is chosen, where $\partial_s c_{\mathrm{noise}}(s) = 1$, a bounded constant.

On top of these, architectural changes for improved normalization (for stability) and efficient JVP-based computation of $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\theta^-}$ are often necessary, but beyond our scope.

Time-Weighting Function. Manual design of the time-weighting function $\omega(s)$ may lead to suboptimal performance. To address this, following a similar approach to EDM-2 (Karras *et al.*, 2024), Lu and Song (2024) learn an adaptive weighting function $\omega_{\boldsymbol{\varphi}}(s)$ to balance the training loss variance across different times s (see Equation (11.3.11) for the desired outcome).

To elaborate further, we observe that the objective function in Equation (11.3.9) takes the form

$$\mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\mathbf{F}_{\boldsymbol{\theta}}^{\top}\mathbf{y}\right], \quad \text{with} \quad \mathbf{y} = -\omega(s)\sigma_{\mathrm{d}}\sin(s)\frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}.$$

Since $\mathbf{y}$ is a vector independent of $\boldsymbol{\theta}$, Equation (11.3.9) is equivalent to

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\mathbf{F}_{\boldsymbol{\theta}}^{\top}\mathbf{y}\right] = \frac{1}{2}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left\|\mathbf{F}_{\boldsymbol{\theta}} - \mathbf{F}_{\boldsymbol{\theta}^-} + \mathbf{y}\right\|_2^2.$$

Based on this observation, Lu and Song (2024) propose additionally training an adaptive weighting network $\omega_{\boldsymbol{\varphi}}(s)$ to estimate the loss norm, formulated as the following minimization problem:

$$\min_{\boldsymbol{\varphi}} \mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{e^{\omega_{\boldsymbol{\varphi}}(s)}}{D}\left\|\mathbf{F}_{\boldsymbol{\theta}} - \mathbf{F}_{\boldsymbol{\theta}^-} + \mathbf{y}\right\|_2^2 - \omega_{\boldsymbol{\varphi}}(s)\right].$$

To understand the effect of the adaptive weighting, observe that the optimal solution $\omega^*(s)$ (obtained by taking the partial derivative of the above objective with respect to $\omega_{\boldsymbol{\varphi}}$) satisfies

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{e^{\omega^*(s)}}{D}\left\|\mathbf{F}_{\boldsymbol{\theta}} - \mathbf{F}_{\boldsymbol{\theta}^-} + \mathbf{y}\right\|_2^2\right] = 1, \qquad \text{for almost every } s. \tag{11.3.11}$$

That is, for an unconstrained pointwise weighting function, the expected rescaled loss is normalized separately at each time s. As a result, the adaptive weighting effectively reduces the variance of the training loss across different time steps, leading to more balanced and stable training.

Time Sampling Distribution. Lu and Song (2024) opt to sample $\tan(s)$ from a log-normal proposal distribution (Karras *et al.*, 2022), that is,

$$\log\left(\sigma_{\mathrm{d}}\tan(s)\right) \sim \mathcal{N}(\cdot; P_{\mathrm{mean}}, P_{\mathrm{std}}^2). \tag{11.3.12}$$

Here, P_{mean} and P_{std} are two hyper-parameters. Following Lu and Song (2024), we can use the prior weighting

$$\omega(s) = \frac{1}{\sigma_{\mathrm{d}}\tan(s)},$$

so that

$$-\omega(s)\sigma_{\mathrm{d}}\sin(s)\frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}}{\mathrm{d}s} = -\cos(s)\frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}.$$

Algorithm 10 Training of Continuous-time Consistency Models (sCM)

Input: dataset $\mathcal{D}$ with std. σ_{d}, pre-trained DM $\mathbf{F}_{\text{pretrain}}$ with parameter $\boldsymbol{\theta}_{\text{pretrain}}$, model $\mathbf{F}_{\boldsymbol{\theta}}$, weighting $\omega_{\boldsymbol{\varphi}}$, learning rate η, proposal $(P_{\text{mean}}, P_{\text{std}})$, constant c, warmup iteration H

Init: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{pretrain}}$, Iters $\leftarrow 0$

Repeat

  $\mathbf{x}_0 \sim \mathcal{D}$, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{d}}^2\mathbf{I})$, $\tau \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$, $s \leftarrow \arctan\left(\frac{e^{\tau}}{\sigma_{\mathrm{d}}}\right)$

  $\mathbf{x}_s \leftarrow \cos(s)\mathbf{x}_0 + \sin(s)\mathbf{z}$

  if consistency training **then**

    $\frac{\mathrm{d}\mathbf{x}_s}{\mathrm{d}s} \leftarrow \cos(s)\mathbf{z} - \sin(s)\mathbf{x}_0$

  else

    $\frac{\mathrm{d}\mathbf{x}_s}{\mathrm{d}s} \leftarrow \sigma_{\mathrm{d}}\mathbf{F}_{\text{pretrain}}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right)$

  end if

  $r \leftarrow \min\left(1, \frac{\text{Iters}}{H}\right)$ ▷ Tangent warmup

  $\mathbf{w} \leftarrow -\cos^2(s)(\sigma_{\mathrm{d}}\mathbf{F}_{\boldsymbol{\theta}}^{-} - \frac{\mathrm{d}\mathbf{x}_s}{\mathrm{d}s}) - r\cos(s)\sin(s)\left(\mathbf{x}_s + \sigma_{\mathrm{d}}\frac{\mathrm{d}\mathbf{F}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}\right)$

  $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|+c}$ ▷ Tangent normalization

  $\mathcal{L}_{\text{sCM}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \leftarrow \frac{e^{\omega_{\boldsymbol{\varphi}}(s)}}{D}\left\|\mathbf{F}_{\boldsymbol{\theta}}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) - \mathbf{F}_{\boldsymbol{\theta}^-}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) - \mathbf{w}\right\|_2^2 - \omega_{\boldsymbol{\varphi}}(s)$ ▷ Adaptive weighting

  $(\boldsymbol{\theta}, \boldsymbol{\varphi}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi}) - \eta\nabla_{\boldsymbol{\theta},\boldsymbol{\varphi}}\mathcal{L}_{\text{sCM}}(\boldsymbol{\theta}, \boldsymbol{\varphi})$

  Iters $\leftarrow$ Iters $+ 1$

until convergence

Summary of Training Objective. In summary of the aforementioned discussion, the final training loss is expressed as:

$$\mathcal{L}_{\text{sCM}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) :=$$
$$\mathbb{E}_{s,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{e^{\omega_{\boldsymbol{\varphi}}(s)}}{D}\left\|\mathbf{F}_{\boldsymbol{\theta}}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) - \mathbf{F}_{\boldsymbol{\theta}^-}\left(\frac{\mathbf{x}_s}{\sigma_{\mathrm{d}}}, s\right) - \cos(s)\frac{\mathrm{d}\mathbf{f}_{\boldsymbol{\theta}^-}}{\mathrm{d}s}(\mathbf{x}_s, s)\right\|_2^2 - \omega_{\boldsymbol{\varphi}}(s)\right].$$

Here, s is sampled according to Equation (11.3.12), and $\mathbf{x}_s$ is computed via Equation (11.3.7). The model trained with this loss is referred to as *sCM*, and its training procedure is summarized in Algorithm 10.

11.4 General Flow Map: Consistency Trajectory Model

Consistency Trajectory Model (CTM) (Kim *et al.*, 2024a) is among the first methods to learn a *general* flow map $\boldsymbol{\Psi}_{s\to t}$.

Setup of CTM in Practice. Similar to the CM family, CTM originally follows the formulation of EDM (Karras *et al.*, 2022) (Section D.6), using the PF-ODE in $\mathbf{x}$-prediction form with the noise schedule $\alpha_t = 1$ and $\sigma_t = t$. Under this setup, the PF-ODE becomes

$$\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} = \frac{\mathbf{x}(\tau) - \mathbb{E}[\mathbf{x}|\mathbf{x}(\tau)]}{\tau}.$$

Starting from $\mathbf{x}_s$ at time s and evolving backward to an ending time $t \le s$, the corresponding flow map (solution) can be written equivalently as

$$\mathbf{x}_s + \int_s^t \frac{\mathbf{x}_\tau - \mathbb{E}[\mathbf{x}|\mathbf{x}_\tau]}{\tau}\,\mathrm{d}\tau.$$

CTM adopts an Euler-inspired parameterization: applying a single-step Euler solver to the PF-ODE[7] yields

$$\mathbf{x}_{s\to t}^{\text{Euler}} = \mathbf{x}_s - (s-t)\,\frac{\mathbf{x}_s - \mathbb{E}[\mathbf{x}|\mathbf{x}_s]}{s} = \frac{t}{s}\,\mathbf{x}_s + \Big(1 - \frac{t}{s}\Big)\mathbb{E}[\mathbf{x}|\mathbf{x}_s],$$

where $\mathbf{x}_{s\to t}^{\text{Euler}}$ approximates the solution at time t given the state $\mathbf{x}_s$ at time s.

While the EDM setup provides a simple illustrative case, CTM allows broader noise schedules defined by an arbitrary linear Gaussian forward kernel (α_t, σ_t) and expresses the PF-ODE in $\mathbf{v}$-prediction form:

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u.$$

In the discussion that follows, we focus on this general formulation.

11.4.1 CTM Parametrization for Flexible Transition Learning

Following the single-step Euler solver of the PF-ODE above, CTM rewrites the oracle flow map $\boldsymbol{\Psi}_{s\to t}$ as a convex combination of the input $\mathbf{x}_s$ and a residual function $\mathbf{g}^*$:

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u = \frac{t}{s}\,\mathbf{x}_s + \frac{s-t}{s}\underbrace{\Big[\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u\Big]}_{=:\,\mathbf{g}^*}.$$

[7] For affine-in-time schedules, such as $(\alpha_t, \sigma_t) = (1, t)$ or $(1-t, t)$, the plain Euler step in diffusion time coincides exactly with the deterministic DDIM update. For general schedules, the two agree only to first order. See discussion in Section 9.2.2.

where the residual term $\mathbf{g}^*$ is defined as

$$\mathbf{g}^*(\mathbf{x}_s, s, t) := \mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u. \tag{11.4.1}$$

This motivates the neural parameterization

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) := \frac{t}{s}\,\mathbf{x}_s + \frac{s-t}{s}\,\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t), \tag{11.4.2}$$

where $\mathbf{g}_{\boldsymbol{\theta}}$ is a neural network that aims at $\mathbf{g}_{\boldsymbol{\theta}} \approx \mathbf{g}^*$, hence $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$ is trained to approximate the oracle flow map,

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s).$$

Therefore, CTM naturally fits within the general consistency-mapping framework of Equation (10.1.4), which aligns the learned mapping with the oracle flow map.

Moreover, this formulation inherently satisfies the initial condition

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, s) = \mathbf{x}_s,$$

without requiring any explicit enforcement during training.

Advantages of CTM's Parametrizations. A crucial characteristic of $\mathbf{g}^*$ becomes evident when taking the limit as t approaches s (i.e., the same ending time as the starting time):

Proposition 11.4.1: Properties of $\mathbf{g}^*$

(i) **Recovering Diffusion Model:**

$$\mathbf{g}^*(\mathbf{x}_s, s, s) = \lim_{t\to s}\mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s).$$

(ii) **Integration Representation:**

$$\mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}(|t-s|).$$

Proof for Proposition.

From the definition of $\mathbf{g}^*$, we obtain

$$\lim_{t\to s}\mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_s - s\lim_{t\to s}\frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s).$$

This proved the first identity. For the second claim, from the Taylor expansion, we have

$$\begin{aligned}\mathbf{g}^*(\mathbf{x}_s, s, t) &= \mathbf{x}_s - \frac{s}{s-t}\int_t^s \mathbf{v}^*(\mathbf{x}_\tau, \tau)\,\mathrm{d}\tau \\ &= \mathbf{x}_s - \frac{s}{s-t}\Big[(s-t)\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}((t-s)^2)\Big] \\ &= \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}(|t-s|).\end{aligned}$$

From this proposition, we can conclude that

1. Estimating $\mathbf{g}^*$ enables approximating not only the finite s-to-t transition (for $s \geq t$) but also the *infinitesimal* s-to-s transition characterized by the instantaneous velocity $\mathbf{v}^*$.

2. $\mathbf{g}^*(\mathbf{x}_s, s, t)$ is interpreted as a denoiser-like quantity added with a residual term of the Taylor expansion.

Therefore, by leveraging CTM's parameterization in Equation (11.4.2), learning $\mathbf{G}_{\boldsymbol{\theta}} \approx \boldsymbol{\Psi}_{s\to t}$ for $t < s$ corresponds, up to the nonzero factor $(s-t)/s$, to learning $\mathbf{g}_{\boldsymbol{\theta}} \approx \mathbf{g}^*$, enabling long-jump capability through $\mathbf{G}_{\boldsymbol{\theta}}$. Moreover, the diagonal value of $\mathbf{g}^*$ encodes the diffusion model's instantaneous velocity through

$$\mathbf{v}^*(\mathbf{x}_s, s) = \frac{\mathbf{x}_s - \mathbf{g}^*(\mathbf{x}_s, s, s)}{s},$$

which can be directly supervised by the diffusion-model loss introduced below. Thus, CTM's parameterization provides a single framework for learning both finite-time transitions and local diffusion dynamics, combining the long-jump capability of flow-map models with access to the instantaneous dynamics of diffusion models.

In the next two sections, we first present CTM's consistency loss (Section 11.4.2), which supports both distillation and training-from-scratch, and enforces the semigroup property to achieve $\mathbf{G}_{\boldsymbol{\theta}}(\cdot, s, t) \approx \boldsymbol{\Psi}_{s\to t}(\cdot, s, t)$. We then describe auxiliary losses (Section 11.4.3) that arise naturally from the parametrization in Equation (11.4.2), including diffusion model loss and GAN loss, which further improve CTM's performance significantly.

11.4.2 Consistency Loss in CTM

CTM aims to approximate the oracle solution map

$$\mathbf{G}_{\boldsymbol{\theta}}(\cdot, s, t) \approx \boldsymbol{\Psi}_{s\to t}(\cdot, s, t),$$

for any $s \geq t$. Since the oracle $\boldsymbol{\Psi}_{s\to t}$ is usually not available in closed form, CTM builds a feasible regression target by enforcing the *semigroup property*

(Equation (11.2.1)): for any $s \geq u \geq t$,

$$\boldsymbol{\Psi}_{u\to t} \circ \boldsymbol{\Psi}_{s\to u} = \boldsymbol{\Psi}_{s\to t}.$$

Depending on whether a pre-trained diffusion model is available, the flow map $\boldsymbol{\Psi}_{s\to t}$ can be approximated in different ways. Throughout, we assume $s \geq u \geq t \in [0, T]$.

Training via Distillation. Assume access to a pre-trained diffusion model producing $\mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_s, s) \approx \mathbf{v}^*(\mathbf{x}_s, s)$. Then the PF-ODE is approximated by the empirical dynamics

$$\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} = \mathbf{v}_{\boldsymbol{\phi}^\times}(\mathbf{x}_\tau, \tau). \tag{11.4.3}$$

CTM trains $\mathbf{G}_{\boldsymbol{\theta}}$ to match a numerical solver $\texttt{Solver}_{s\to t}(\mathbf{x}_s; \boldsymbol{\phi}^\times)$ applied to this empirical ODE, which serves as a computable proxy for the oracle:

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \texttt{Solver}_{s\to t}(\mathbf{x}_s; \boldsymbol{\phi}^\times) \approx \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s, s, t).$$

With a strong teacher, the solver can recover $\boldsymbol{\Psi}_{s\to t}$ up to discretization error, so the optimal student closely matches the ground truth (see (Kim *et al.*, 2024a), Propositions 3 and 4).

However, solving across the full interval $[t, s]$ during training loop can be costly when s and t are far apart. To improve efficiency and provide a smoother signal, CTM introduces *soft consistency matching*, which operationalizes the semigroup property. As illustrated in Figure 11.5, CTM compares two predictions at time t: the direct student output $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$, and a mixed teacher–student path that first advances the teacher from s to a random $u \sim \mathcal{U}[t, s)$, then lets the student jump from u to t:

$$\mathbf{G}_{\boldsymbol{\theta}^-}\big(\texttt{Solver}_{s\to u}(\mathbf{x}_s; \boldsymbol{\phi}^\times),\, u,\, t\big).$$

The student is trained to match this composite prediction:

$$\underbrace{\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)}_{\approx \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)} \approx \underbrace{\mathbf{G}_{\boldsymbol{\theta}^-}\big(\texttt{Solver}_{s\to u}(\mathbf{x}_s; \boldsymbol{\phi}^\times), u, t\big)}_{\approx \boldsymbol{\Psi}_{u\to t}(\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s))}, \tag{11.4.4}$$

where $\boldsymbol{\theta}^-$ is a stop gradient copy of $\mathbf{G}_{\boldsymbol{\theta}}$.

By varying u, CTM interpolates between global and local supervision:

- **Global Consistency** ($u = t$): the teacher is solved over the full interval $s \to t$, so the target reduces to the full teacher transition at time t.
- **Local Consistency** ($u = s - \Delta s$): the student learns from a short teacher step near s; when $t = 0$, this reduces to consistency distillation.

To reinforce sample quality while aligning trajectories, both predictions are mapped to time 0 by the stop gradient student and compared in a feature space metric d:

$$\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t) := \mathbf{G}_{\boldsymbol{\theta}^-}\big(\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t),\, t,\, 0\big),$$
$$\mathbf{x}_{\text{target}}(\mathbf{x}_s, s, u, t) := \mathbf{G}_{\boldsymbol{\theta}^-}\big(\mathbf{G}_{\boldsymbol{\theta}^-}(\texttt{Solver}_{s\to u}(\mathbf{x}_s; \boldsymbol{\phi}^\times), u, t),\, t,\, 0\big).$$

The CTM consistency loss is

$$\mathcal{L}_{\text{consist}}(\boldsymbol{\theta}; \boldsymbol{\phi}^\times) := \mathbb{E}_{s\in[0,T]}\mathbb{E}_{t\in[0,s]}\mathbb{E}_{u\in[t,s)}\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{\mathbf{x}_s|\mathbf{x}_0}\Big[d\big(\mathbf{x}_{\text{est}}, \mathbf{x}_{\text{target}}\big)\Big], \quad (11.4.5)$$

which encourages the student to match the empirical PF-ODE solution while preserving generation quality.

Training from Scratch. Leveraging CTM's special parameterization (Proposition 11.4.1(i)),

$$\mathbf{g}^*(\mathbf{x}_\tau, \tau, \tau) = \mathbf{x}_\tau - \tau\, \mathbf{v}^*(\mathbf{x}_\tau, \tau) \implies \mathbf{v}^*(\mathbf{x}_\tau, \tau) = \frac{\mathbf{x}_\tau - \mathbf{g}^*(\mathbf{x}_\tau, \tau, \tau)}{\tau}.$$

We can therefore replace the oracle residual function $\mathbf{g}^*(\cdot, \tau, \tau)$ with CTM's own estimate $\mathbf{g}_{\boldsymbol{\theta}^-}(\cdot, \tau, \tau)$ for $\tau \in (0, T]$, which yields a self-induced empirical PF-ODE:

$$\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} = \frac{\mathbf{x}(\tau) - \mathbf{g}_{\boldsymbol{\theta}^-}(\mathbf{x}(\tau), \tau, \tau)}{\tau}. \quad (11.4.6)$$

We then solve this self-induced ODE to construct a tractable transition target:

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \texttt{Solver}_{s\to t}(\mathbf{x}_s; \boldsymbol{\theta}^-).$$

This target approaches the oracle flow map only insofar as the diagonal estimate $\mathbf{g}_{\boldsymbol{\theta}^-}(\cdot, \tau, \tau)$ accurately recovers the target local dynamics. The diffusion-model loss introduced below provides direct supervision for this local quantity.

As in the distillation case Equation (11.4.4), full integration over $[t, s]$ can be costly when s and t are far apart. CTM therefore enforces the semigroup property to obtain a shorter supervision path:

$$\underbrace{\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)}_{\approx \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)} \approx \underbrace{\mathbf{G}_{\boldsymbol{\theta}^-}\big(\texttt{Solver}_{s\to u}(\mathbf{x}_s; \boldsymbol{\theta}^-),\, u,\, t\big)}_{\approx \boldsymbol{\Psi}_{u\to t}(\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s))},$$

where $u \sim \mathcal{U}[t, s)$ and $\boldsymbol{\theta}^-$ is a stop gradient copy of the student. The only change from distillation is that the external teacher $\mathbf{v}_{\boldsymbol{\phi}^\times}$ is replaced by the self-induced teacher $\mathbf{g}_{\boldsymbol{\theta}^-}$.

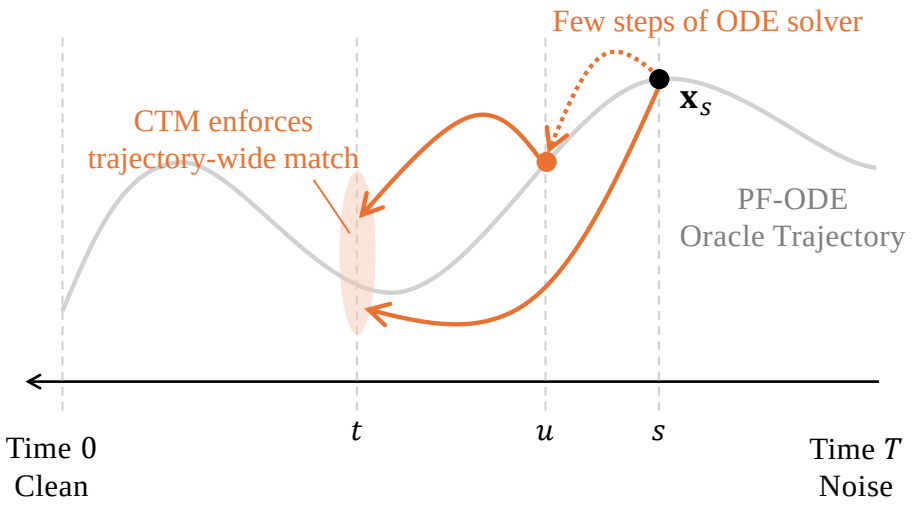


Figure 11.5: Illustration of CTM's semigroup property. For any $s \geq u \geq t$, CTM enforces $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \mathbf{G}_{\boldsymbol{\theta}^-}\big(\texttt{Solver}_{s\to u}(\mathbf{x}_s), u, t\big)$, i.e., a short solver segment $s \to u$ followed by a CTM "jump" to t matches the direct CTM map $s \to t$. The solver may be a pre-trained diffusion or a CTM's self-induced teacher.

Source: Adapted from Kim et al. *(2024a).*

To couple trajectory matching with sample quality, both predictions are mapped to time 0 using the stop gradient student and compared in feature space. The target without any pre-trained model is

$$\hat{\mathbf{x}}_{\text{target}} := \mathbf{G}_{\boldsymbol{\theta}^-}\big(\mathbf{G}_{\boldsymbol{\theta}^-}(\texttt{Solver}_{s\to u}(\mathbf{x}_s; \boldsymbol{\theta}^-),\, u,\, t),\, t,\, 0\big),$$

which replaces $\mathbf{x}_{\text{target}}$ in Equation (11.4.5), and leads to:

$$\mathcal{L}_{\text{consist}}(\boldsymbol{\theta}; \boldsymbol{\theta}^-) := \mathbb{E}_{s\in[0,T]}\mathbb{E}_{t\in[0,s]}\mathbb{E}_{u\in[t,s)}\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{\mathbf{x}_s|\mathbf{x}_0}\Big[d\big(\mathbf{x}_{\text{est}}, \hat{\mathbf{x}}_{\text{target}}\big)\Big], \quad (11.4.7)$$

Conceptually, this is self-distillation within CTM: the model supplies its own short horizon teacher signals while the student learns the full transition.

11.4.3 Auxiliary Losses in CTM

(Self-)distillation can underperform the teacher because it optimizes only teacher generated targets, lacking direct supervision from real data. By contrast, CTM can naturally incorporate data driven regularizers, for example by augmenting its objective with denoising score matching and an adversarial (GAN) term (Goodfellow *et al.*, 2014), to better learn the flow map.

Natural Integration of Diffusion Loss. The diffusion–model loss (more precisely, the conditional flow matching loss; see Equation (5.2.10)) integrates naturally into CTM and provides a fixed regression target that facilitates the learning of the flow map model. To see this, note that we have

$$\mathbf{v}^*(\mathbf{x}_s, s) = \frac{\mathbf{x}_s - \mathbf{g}^*(\mathbf{x}_s, s, s)}{s}, \qquad \mathbf{g}^*(\mathbf{x}_s, s, s) \approx \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, s).$$

This naturally induces a velocity parametrization through $\mathbf{g}_{\boldsymbol{\theta}}$:

$$\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) := \frac{1}{s}\big(\mathbf{x}_s - \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, s)\big).$$

Using the linear Gaussian path

$$\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}, \qquad \mathbf{x}_0 \sim p_{\text{data}}, \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}),$$

the diffusion model loss can be written as

$$\mathcal{L}_{\text{DM}}(\boldsymbol{\theta}) := \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}, s}\Big[w(s)\big\|\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \big(\alpha_s' \mathbf{x}_0 + \sigma_s' \boldsymbol{\epsilon}\big)\big\|_2^2\Big]. \tag{11.4.8}$$

$\mathcal{L}_{\text{DM}}$ improves accuracy when t is close to s by explicitly supervising small jumps along the trajectory. In this regime, the factor $1 - \frac{t}{s}$ in Equation (11.4.2) approaches zero, which can weaken gradients and slow learning; $\mathcal{L}_{\text{DM}}$ supplies a stronger local signal and stabilizes training.

Conceptually, Equations (11.4.5) and (11.4.7) enforce trajectory matching (zeroth order), while Equation (11.4.8) enforces slope matching (first order).

(Optional) GAN Loss. While consistency and diffusion model loss provide strong regression signals, they can yield overly smooth outputs. CTM therefore optionally adds an adversarial term to encourage sharper, more realistic samples by aligning the generator distribution with the data distribution. With a discriminator $D_{\boldsymbol{\zeta}}$ that distinguishes real $\mathbf{x}_0 \sim p_{\text{data}}$ from generated $\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t)$, the objective is

$$\begin{aligned}&\mathcal{L}_{\text{GAN}}(\boldsymbol{\theta}, \boldsymbol{\zeta}) := \\ &\mathbb{E}_{\mathbf{x}_0}\big[\log D_{\boldsymbol{\zeta}}(\mathbf{x}_0)\big] + \mathbb{E}_{s\in[0,T]}\mathbb{E}_{t\in[0,s]}\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{\mathbf{x}_s|\mathbf{x}_0}\big[\log(1 - D_{\boldsymbol{\zeta}}(\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t)))\big],\end{aligned}$$

where $D_{\boldsymbol{\zeta}}$ is maximized and $\mathbf{G}_{\boldsymbol{\theta}}$ is minimized. Intuitively, the discriminator acts as an adaptive perceptual distance that encourages realistic detail. Theoretically, the GAN term drives distributional matching (Jensen–Shannon divergence) between p_{data} and the model distribution induced by $\mathbf{G}_{\boldsymbol{\theta}}$ (Goodfellow *et al.*, 2014), which can raise fidelity beyond the teacher.

Overall CTM Objective. In summary, CTM unifies (self-)distillation, diffusion, and GAN losses into a single training framework:

$$\mathcal{L}_{\mathrm{CTM}}(\boldsymbol{\theta}, \boldsymbol{\zeta}) := \mathcal{L}_{\mathrm{consist}}(\boldsymbol{\theta}; \boldsymbol{\phi}^{\times}/\boldsymbol{\theta}^{-}) + \lambda_{\mathrm{DM}}\mathcal{L}_{\mathrm{DM}}(\boldsymbol{\theta}) + \lambda_{\mathrm{GAN}}\mathcal{L}_{\mathrm{GAN}}(\boldsymbol{\theta}, \boldsymbol{\zeta}),$$

where the teacher is either an external pre-trained model $\boldsymbol{\phi}^{\times}$ or the self-induced teacher $\boldsymbol{\theta}^{-}$. The regression style components $\mathcal{L}_{\mathrm{consist}}$ and $\mathcal{L}_{\mathrm{DM}}$ act as strong regularizers, while the optional GAN term improves fine scale detail without sacrificing stability (Kim *et al.*, 2024b).

11.4.4 Flexible Sampling with CTM

CTM learns the general flow map $\boldsymbol{\Psi}_{s\to t}$ for any $s > t$, which means it supports anytime to anytime transitions. This property enables flexible sampling strategies. For example, CTM proposes γ *sampling*, where the hyperparameter γ controls the stochasticity during generation. In addition, CTM can reuse standard inference techniques developed for diffusion models, such as ODE based solvers and exact likelihood computation.

In what follows, we fix a discrete time grid for sampling $T = \tau_0 > \tau_1 > \tau_2 > \cdots > \tau_M = 0$.

Algorithm 11 CTM's γ-sampling

Input: Trained CTM $\mathbf{G}_{\boldsymbol{\theta}^{\times}}$, $\gamma \in [0,1]$, $T = \tau_0 > \tau_1 > \tau_2 > \cdots > \tau_M = 0$.

1: Start from $\mathbf{x}_{\tau_0} \sim p_{\mathrm{prior}} = \mathcal{N}(\mathbf{0}, T^2\mathbf{I})$
2: **for** $n = 0$ to $M-1$ **do**
3: $\quad \tilde{\tau}_{n+1} \leftarrow \sqrt{1-\gamma^2}\tau_{n+1}$
4: $\quad$ Denoise $\mathbf{x}_{\tilde{\tau}_{n+1}} \leftarrow \mathbf{G}_{\boldsymbol{\theta}^{\times}}(\mathbf{x}_{\tau_n}, \tau_n, \tilde{\tau}_{n+1})$
5: $\quad$ Diffuse $\mathbf{x}_{\tau_{n+1}} \leftarrow \mathbf{x}_{\tilde{\tau}_{n+1}} + \gamma\tau_{n+1}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: **end for**

Output: $\mathbf{x}_{\tau_M}$

Methodology of γ-Sampling. CTM's γ-sampling introduces a unified family of samplers that arises naturally from learning a general flow map model. It encompasses prior approaches, such as CM's multistep sampling (see Algorithm 9) and time-stepping-style sampling, which is conceptually similar to ODE solvers. The parameter γ directly controls the degree of semantic change during generation, making γ sampling a flexible and task aware strategy for diverse downstream applications.

- Figure 11.6-(Left): When $\gamma = 1$, it coincides with the multistep sampling introduced in CM (i.e., a special flow map $\boldsymbol{\Psi}_{s\to 0}$), which is fully stochastic and results in semantic variation when the number of steps changes.

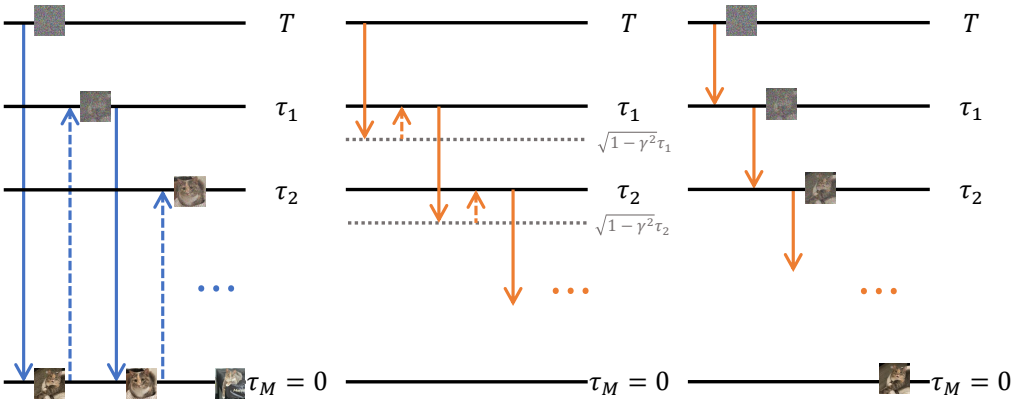


Figure 11.6: Illustration of γ-sampling with varying γ value. The procedure alternates between denoising with a network evaluation and adding noise in reverse, $(\tau_n \xrightarrow{\text{Denoise}} \sqrt{1-\gamma^2}\tau_{n+1} \xrightarrow{\text{Noisify}} \tau_{n+1})_{n=0}^{M-1}$. The leftmost panel illustrates $\gamma = 1$, corresponding to the fully stochastic case. The rightmost panel shows $\gamma = 0$, corresponding to the fully deterministic case. The middle panel depicts intermediate values $\gamma \in (0, 1)$, which interpolate between these two extremes.

Source: Adapted from Kim et al. *(2024a).*

- Figure 11.6-(Right): When $\gamma = 0$, it reduces to fully deterministic time-stepping, which estimates the solution trajectory of the PF-ODE. A key distinction between γ sampling with $\gamma = 0$ and conventional time-stepping ODE-based sampling is that CTM avoids the discretization errors of numerical solvers.

- Figure 11.6-(Middle): When $0 < \gamma < 1$, γ-sampling interpolates between the two extremes by allowing a controlled amount of stochasticity to be injected during sampling.

We highlight that values $\gamma \in [0, 1)$ require the general flow map $\boldsymbol{\Psi}_{s\to t}$, whereas the special case $\gamma = 1$ reduces to CM-style multistep sampling and requires only $\boldsymbol{\Psi}_{s\to 0}$.

Analysis of γ-Sampling. CTM empirically observed that CM's multistep sampling degrades in quality once the number of steps $M \geq 4$. To explain this phenomenon, CTM analyzed the underlying cause: when $\gamma \neq 0$, each neural "jump" introduces a small mismatch, and these mismatches accumulate as the model iteratively maps states toward time zero. This error accumulation explains why long multi-step runs can perform poorly. We formalize this idea in the following proposition.

Proposition 11.4.2: (Informal) 2-step γ-sampling

Let $\tau \in (0, T)$ and $\gamma \in [0, 1]$. Let $p_{\boldsymbol{\theta}^*,2}$ denote as the density obtained from the γ-sampler with the optimal CTM, following the transition sequence $T \to \sqrt{1-\gamma^2}\tau \to \tau \to 0$, starting from p_T. Then

$$\mathcal{D}_{\mathrm{TV}}(p_{\mathrm{data}}, p_{\boldsymbol{\theta}^*,2}) = \mathcal{O}\left(\sqrt{T - \sqrt{1-\gamma^2}\tau + \tau}\right).$$

Here, $\mathcal{D}_{\mathrm{TV}}$ denotes the total variation between distributions (see Equation (1.1.4)).

Proof for Proposition.

We refer the reader to Theorem 8 of Kim *et al.* (2024a) for the general case when the number of sampling steps is M.

The insights from the above theorem can be summarized as follows:

- **When** $\gamma = 1$ (corresponding to CM's multistep sampling)**:** The method performs iterative long-range transitions from τ_n to 0 at each step n. This leads to error accumulation on the order of

$$\mathcal{O}\left(\sqrt{T + \tau_1 + \cdots + \tau_M}\right).$$

- **When** $\gamma = 0$ (corresponding to CTM's deterministic multistep sampling)**:** Such temporal overlap between transitions is eliminated. This avoids error accumulation and yields a tighter bound of $\mathcal{O}(\sqrt{T})$. Empirically, CTM with $\gamma = 0$ provides a favorable trade-off between sampling speed and sample quality: increasing the number of sampling steps improves generation quality without introducing instability.

CTM Supports Diffusion Inference. Since CTM learns the score function (or denoiser) directly through $\mathbf{g}_{\boldsymbol{\theta}}$, thanks to its parametrization in Equation (11.4.2), it is compatible with inference techniques originally developed for diffusion models. For instance, one can compute exact likelihoods (Section 4.3.2) or apply advanced samplers such as DDIM or DPM (Chapter 9) for generation, by using $\mathbf{g}_{\boldsymbol{\theta}}(\cdot, s, s)$.

11.5 General Flow Map: Mean Flow

Just as diffusion models admit many equivalent parameterizations and training objectives, a general flow map $\mathbf{\Psi}_{s\to t}$ can also be learned in multiple plausible ways. In this section, we introduce *Mean Flow* (MF) (Geng *et al.*, 2025a), a later representative of the general flow map family $\mathbf{\Psi}_{s\to t}$ that illustrates an alternative yet principled perspective on how such maps can be effectively learned.

11.5.1 Modeling and Training of Mean Flow

In contrast to CM and CTM, which build on the EDM framework, MF is based on the flow matching formulation ($\alpha_t = 1 - t$ and $\sigma_t = t$ for $t \in [0, 1]$). Rather than directly parameterizing the flow map, MF learns the *average drift* over an interval $[t, s]$ (with $t < s$):

$$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u.$$

The corresponding oracle loss is

$$\mathbb{E}_{t<s}\mathbb{E}_{\mathbf{x}_s\sim p_s}\Big[w(s)\|\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \mathbf{h}^*(\mathbf{x}_s, s, t)\|_2^2\Big]. \tag{11.5.1}$$

In particular, when $s \to t$, the loss function reduces to the flow matching loss:

$$\mathbb{E}_t\mathbb{E}_{\mathbf{x}_t\sim p_t}\Big[w(t)\|\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, t) - \mathbf{v}^*(\mathbf{x}_t, t)\|_2^2\Big], \tag{11.5.2}$$

learning the instantaneous velocity. We will see later in Section 11.5.3 that MF remains consistent with the general objective in Equation (10.1.4), but approaches it from a different (while equivalent) perspective. Since the oracle regression target $\mathbf{h}^*(\mathbf{x}_s, s, t)$ does not admit a closed form in general, MF constructs a surrogate by exploiting an identity obtained from differentiating

$$(t-s)\,\mathbf{h}^*(\mathbf{x}_s, s, t) = \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\;\mathrm{d}u$$

with respect to s. This yields the *MF identity*:

$$\begin{aligned}\mathbf{h}^*(\mathbf{x}_s, s, t) &= \mathbf{v}^*(\mathbf{x}_s, s) - (s-t)\tfrac{\mathrm{d}}{\mathrm{d}s}\mathbf{h}^*(\mathbf{x}_s, s, t)\\ &= \mathbf{v}^*(\mathbf{x}_s, s) - (s-t)\Big[(\partial_{\mathbf{x}}\mathbf{h}^*)(\mathbf{x}_s, s, t)\mathbf{v}^*(\mathbf{x}_s, s) + \partial_s\mathbf{h}^*(\mathbf{x}_s, s, t)\Big],\end{aligned} \tag{11.5.3}$$

where the second line applies the chain rule together with

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{x}_s = \mathbf{v}^*(\mathbf{x}_s, s).$$

Motivated by this identity, MF replaces the intractable oracle with a stop-gradient surrogate, leading to the practical training objective

$$\mathcal{L}_{\mathrm{MF}}(\boldsymbol{\theta}) := \mathbb{E}_{t<s}\mathbb{E}_{\mathbf{x}_s\sim p_s}\Big[w(s)\,\|\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s,s,t) - \mathbf{h}^{\mathrm{tgt}}_{\boldsymbol{\theta}^-}(\mathbf{x}_s,s,t)\|_2^2\Big], \tag{11.5.4}$$

where the regression target is defined as

$$\mathbf{h}^{\mathrm{tgt}}_{\boldsymbol{\theta}^-}(\mathbf{x}_s,s,t) := \mathbf{v}^*(\mathbf{x}_s,s) - (s-t)\Big[\underbrace{(\partial_{\mathbf{x}}\mathbf{h}_{\boldsymbol{\theta}^-})(\mathbf{x}_s,s,t)\mathbf{v}^*(\mathbf{x}_s,s) + \partial_s\mathbf{h}_{\boldsymbol{\theta}^-}(\mathbf{x}_s,s,t)}_{\mathrm{JVP}}\Big].$$

In practice, the oracle velocity $\mathbf{v}^*$ cannot be computed in closed form and must instead be approximated. Two common strategies are available: relying on a pre-trained diffusion model (*distillation*) or constructing a direct estimator from data (*training from scratch*). Regardless of the choice, one ultimately needs to compute a *Jacobian–vector product* (JVP) of the target network $\mathbf{h}_{\boldsymbol{\theta}^-}$:

$$[\partial_{\mathbf{x}}\mathbf{h}_{\boldsymbol{\theta}^-}, \partial_s\mathbf{h}_{\boldsymbol{\theta}^-}, \partial_t\mathbf{h}_{\boldsymbol{\theta}^-}]^\top \cdot [\mathbf{v}^*, 1, 0]$$

Distillation. Use a pre-trained diffusion model with a flow matching backbone, $\mathbf{v}_{\boldsymbol{\phi}^\times} \approx \mathbf{v}^*$.

Training from scratch. Use the one point conditional velocity $\alpha_s'\mathbf{x}_0 + \sigma_s'\boldsymbol{\epsilon}$, obtained from the forward noise injection $\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This gives an unbiased single sample estimate of the instantaneous drift at level s when evaluated at paired $(\mathbf{x}_0, \boldsymbol{\epsilon})$.

11.5.2 Sampling of Mean Flow

Once a MF $\mathbf{h}_{\boldsymbol{\theta}^\times}$ is trained, it naturally recovers a proxy of the flow map. For any starting point $\mathbf{x}_s$, the map from s to t is (approximately) given by

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + (t-s)\,\mathbf{h}^*(\mathbf{x}_s,s,t) \approx \mathbf{x}_s + (t-s)\,\mathbf{h}_{\boldsymbol{\theta}^\times}(\mathbf{x}_s,s,t).$$

This enables both one-step and multi-step sampling. For example, drawing $\mathbf{x}_T \sim p_{\mathrm{prior}}$, the one-step generation of a clean sample is

$$\mathbf{x}_0 \leftarrow \mathbf{x}_T - T\,\mathbf{h}_{\boldsymbol{\theta}^\times}(\mathbf{x}_T, T, 0).$$

Alternatively, multi-step generation can be performed by preparing a time grid and applying the map sequentially, in the same time-stepping manner used in CTM. Since MF learns a general flow map, it also supports γ-sampling as in CTM, where a controllable hyperparameter γ injects stochasticity into the sampling process.

11.5.3 Relationship between CTM and MF

At first sight CTM and MF may appear unrelated. In fact, both provide different parameterizations of the same oracle flow map $\boldsymbol{\Psi}_{s\to t}$, and their corresponding squared oracle regression errors are related by a time-dependent weighting (Hu *et al.*, 2025).

Relationship of Parameterizations. Both methods operate under the same general framework but represent the learned function in distinct ways. The flow map can be written equivalently as

$$\begin{aligned}\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) &= \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \\ &= \frac{t}{s}\mathbf{x}_s + \frac{s-t}{s}\underbrace{\left[\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\ \mathrm{d}u\right]}_{\approx \mathbf{g}_{\boldsymbol{\theta}}} \\ &= \mathbf{x}_s + (t-s)\underbrace{\left[\frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\ \mathrm{d}u\right]}_{\approx \mathbf{h}_{\boldsymbol{\theta}}}.\end{aligned}$$

Here, the first is the definition of the flow map, the second form highlights the CTM parametrization through $\mathbf{g}_{\boldsymbol{\theta}}$ (see Equations (11.4.1) and (11.4.2)), while the last highlights the MF parametrization through $\mathbf{h}_{\boldsymbol{\theta}}$.

Relationship of Training Loss. Given the above reinterpretation of the oracle flow map $\boldsymbol{\Psi}_{s\to t}$ in terms of the CTM parametrization

$$\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \mathbf{g}^*(\mathbf{x}_s, s, t) := \mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u$$

and the MF parametrization

$$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u,$$

we now show that the training losses of CTM and MF are in fact equivalent.

Consider the relation

$$\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) := \mathbf{x}_s - s\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t),$$

and take $d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|^2$ as an example. Substituting into Equation (10.1.4)

and viewing $\mathbf{G}_{\boldsymbol{\theta}}$ as CTM's flow-map parameterization (Equation (11.4.2)) gives

$$
\begin{aligned}
& d(\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t), \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)) \\
=\ & \|\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)\|^2 \\
=\ & \left\| \left(\frac{t}{s}\mathbf{x}_s + \frac{s-t}{s}\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \right) - \left(\frac{t}{s}\mathbf{x}_s + \frac{s-t}{s}\left[\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \right] \right) \right\|^2 \\
=\ & \left(\frac{s-t}{s} \right)^2 \left\| \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \left(\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \right) \right\|^2 \qquad (11.5.5) \\
=\ & \left(\frac{s-t}{s} \right)^2 \left\| (\mathbf{x}_s - s\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)) - \left(\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \right) \right\|^2 \\
=\ & \left(\frac{s-t}{s} \right)^2 \left\| (\mathbf{x}_s - s\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)) - \left(\mathbf{x}_s + \frac{s}{s-t}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \right) \right\|^2 \\
=\ & (s-t)^2 \left\| \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \left(\frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_u, u)\,\mathrm{d}u \right) \right\|^2 \qquad (11.5.6)
\end{aligned}
$$

Hence,

$$
\frac{1}{s^2}\left\| \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \mathbf{g}^*(\mathbf{x}_s, s, t) \right\|^2 = \left\| \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \mathbf{h}^*(\mathbf{x}_s, s, t) \right\|^2 .
$$

Putting aside the specific algorithm used to learn the flow map, the CTM and MF losses share the same mathematical form, differing only by a weighting function. Moreover, setting $t = 0$ in either case recovers the CM setting ($\boldsymbol{\Psi}_{s\to 0}$), where each state maps directly to the clean data.

Auxiliary Loss in Practice. In CTM, training is performed with the consistency loss in Equation (11.4.7) jointly with its self-defined diffusion model loss in Equation (11.4.8). A similar strategy is adopted in MF. As shown in Equation (11.5.2), when $s \to t$, the MF loss reduces to the standard flow matching objective. In practice, MF controls the ratio between pairs with $s \neq t$ and those with $s = t$; consequently, the overall optimization becomes a mixture of the MF objective in Equation (11.5.4) and the flow matching objective in Equation (11.5.2).

Both parametrizations are able to provide a smooth transition from diffusion-model training, which learns instantaneous velocity with a fixed regression target, to flow-map learning, which employs a stop-gradient pseudo-regression target.

Both CTM and MF Parameterizations Enable Flexible Inference. Both CTM ($\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$) and MF ($\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$) aim to approximate the underlying flow map $\boldsymbol{\Psi}_{s\to t}$:

$$
\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \boldsymbol{\Psi}_{s\to t}, \quad \text{and} \quad \mathbf{x}_s + (t-s)\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \approx \boldsymbol{\Psi}_{s\to t}.
$$

Since both models learn an explicit mapping between any two time steps, they naturally support CTM's γ-sampling and remain compatible with inference techniques originally developed for diffusion models, such as guidance (Chapter 8), exact likelihood computation (Equation (4.3.7)), and accelerated sampling with higher-order solvers (Chapter 9). This compatibility arises because their parameterizations recover the instantaneous diffusion drift in the infinitesimal limit $t \to s$:

$$\mathbf{g}^*(\mathbf{x}_s, s, s) = \mathbf{x}_s - s\,\mathbf{v}^*(\mathbf{x}_s, s), \quad \text{and} \quad \mathbf{h}^*(\mathbf{x}_s, s, s) = \mathbf{v}^*(\mathbf{x}_s, s).$$

This property is not shared by specialized flow map formulations $\boldsymbol{\Psi}_{s\to 0}$, such as those in the CM family. Thus, both CTM and MF can be regarded as flexible and general flow map formulations that generalize diffusion-based inference to direct time-to-time mappings.

Conclusion. The relationship between CTM and MF is analogous to the relationship between different parametrizations of diffusion models (Section 6.3): at the oracle level, they encode the same underlying flow map. Under squared-error regression and the parametrization relation above, their oracle regression errors differ only by a known time-dependent scaling. Their practical training objectives, however, remain different because they use different surrogate targets and optimization procedures.

Although CTM and MF are related in this way, they differ conceptually and algorithmically in how the target is learned. The CTM target is defined in the "integral" form, as it acts directly on the flow map. The MF target, on the other hand, is defined in the "differential" form, as it relies on finite differences of the flow map. Because a target in the integral form is not directly tractable, CTM must solve an ODE during training. In contrast, MF leverages the MF identity (see Equation (11.5.3)) to construct a tractable target in the differential form. This relationship is analogous to that between energy-based and score-based methods: the former learns a probability density itself (in the integral form), while the latter matches the gradient of that density (in the differential form). In practice, CTM and MF may behave differently due to factors such as loss weighting, network architecture, and optimization dynamics, which can cause one method to perform better than the other under certain conditions.

This perspective suggests that CTM and MF are not the only viable formulations. Other parameterizations of the flow map could also enable efficient and stable training, potentially leading to new standalone generative models. Exploring these alternatives may further enrich the landscape of diffusion models and their flow map extensions, pushing the boundaries of few-step generation.

11.6 Closing Remarks

This final chapter has brought our exploration full circle: from slow iterative diffusion samplers to fast few-step generators (learned from scratch). The common object behind the methods in this chapter is the oracle flow map introduced in Equation (10.1.4):

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_\tau, \tau)\, \mathrm{d}\tau.$$

Learning this map replaces many small numerical updates by one or a few learned jumps.

At this point, it is useful to step back from the individual algorithms. Knowledge Distillation (Section 10.1), Progressive Distillation (Section 10.3), CM (Section 11.2), CTM (Section 11.4), and MF (Section 11.5) were introduced through different training constructions, but they all exploit, explicitly or implicitly, the same ODE solution-map structure.

The flow map itself is a classical object from differential equations and dynamical systems, with closely related particle-based and field-based descriptions in continuum and fluid mechanics; see Equation (4.2.2) and the surrounding historical discussion. Here, we use three classical descriptions of the same ODE flow, namely the *semigroup*, *Lagrangian*, and *Eulerian* views, as an organizing lens for the modern learning methods developed above[8]

Semigroup View: Split the Trip. As introduced in Equation (11.2.1), the finite-step identity is

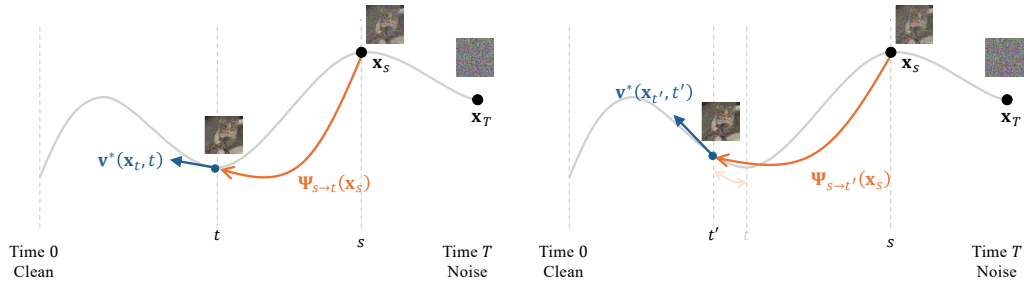
$$\boldsymbol{\Psi}_{u\to t}\big(\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s)\big) = \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s).$$

As illustrated in Figure 11.3, the intuition is the same as traveling along a fixed route: going from s to u and then from u to t lands at the same point as going directly from s to t. Within this view, KD learns the full transition $\boldsymbol{\Psi}_{T\to 0}$, PD compresses two adjacent teacher steps into one student step, CM learns the special anytime-to-clean map $\boldsymbol{\Psi}_{s\to 0}$, and CTM extends the same composition principle to the general map $\boldsymbol{\Psi}_{s\to t}$.

Lagrangian View: Move the Ending Point. Fix the starting state–time pair $(\mathbf{x}_s, s)$ and vary the ending time t. The ending point then moves along the same trajectory:

$$\partial_t \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{v}^*(\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s), t).$$

[8] In modern generative modeling, Boffi *et al.* (2025) bring together structural ideas that already appear across PD, CM, CTM, and MF, and use them to describe the learning of diffusion PF-ODE flow maps in a common language. The main ingredients therefore largely come from these existing lines of work. Our presentation and illustrations in this part are also informed by the exposition of Dieleman (2026).



(a) Base ending time t.

(b) Moved ending time t'.

Figure 11.7: Lagrangian consistency of flow maps. The starting state–time pair $(\mathbf{x}_s, s)$ is fixed, and the ending time is moved from t to t'. The output of the flow map moves along the same PF-ODE trajectory, and its infinitesimal change equals the ODE velocity: $\partial_t \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{v}^*(\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s), t)$.

Source: Adapted from Dieleman (2026).

The name "Lagrangian" follows the classical particle viewpoint: we track the ending point along the trajectory as the ending time changes. Infinitesimally, the ending point moves with the local ODE velocity evaluated at that point.

Eulerian View: Move the Starting Point. Fix the ending time t and slide the starting point along the same trajectory. The ending point at time t remains fixed:

$$\frac{\mathrm{d}}{\mathrm{d}s}\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = 0.$$

Since $\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{x}_s = \mathbf{v}^*(\mathbf{x}_s, s)$, the chain rule gives

$$\partial_s \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) + (\partial_{\mathbf{x}} \boldsymbol{\Psi}_{s\to t})(\mathbf{x}_s)\mathbf{v}^*(\mathbf{x}_s, s) = 0.$$

The term "Eulerian" invokes the classical field-based viewpoint: Here, the ending time t is fixed, while the starting state–time pair $(\mathbf{x}_s, s)$ moves along the same ODE trajectory; the ending point at time t therefore remains unchanged. For CM, the ending time is fixed at $t = 0$, so the ending point is the clean sample:

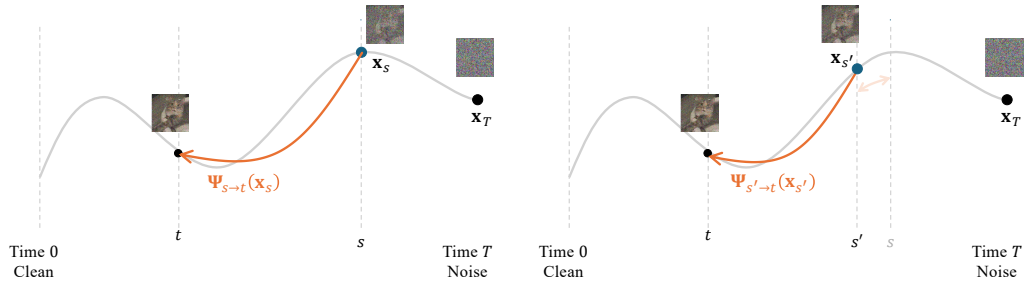
$$\mathbf{f}^*(\mathbf{x}_s, s) = \boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s).$$

Thus, Equation (11.3.1) is the starting-time consistency condition for the special flow map studied in Sections 11.2 and 11.3.

This view also recaps the relationship between CTM and MF from Section 11.5.3. Both approximate the same path integral

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_\tau, \tau)\, \mathrm{d}\tau,$$

but expose different trainable surrogates. CTM represents the ending point through a displacement-style parameterization, while MF represents the same ending point



(a) Base starting time s.

(b) Moved starting time s'.

Figure 11.8: Eulerian consistency of flow maps. The ending time t is fixed, while the starting time is moved from s to s'. Since the corresponding starting points lie on the same PF-ODE trajectory, the ending point remains unchanged: $\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \boldsymbol{\Psi}_{s'\to t}(\mathbf{x}_{s'})$. Infinitesimally, this gives $\frac{\mathrm{d}}{\mathrm{d}s}\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = 0$, or equivalently $\partial_s \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) + (\partial_{\mathbf{x}} \boldsymbol{\Psi}_{s\to t})(\mathbf{x}_s)\mathbf{v}^*(\mathbf{x}_s, s) = 0$.

Source: Adapted from Dieleman (2026).

through the average drift

$$\mathbf{h}^*(\mathbf{x}_s, s, t) = \frac{1}{t-s}\int_s^t \mathbf{v}^*(\mathbf{x}_\tau, \tau)\ \mathrm{d}\tau.$$

Substituting

$$\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{x}_s + (t-s)\mathbf{h}^*(\mathbf{x}_s, s, t)$$

into the Eulerian starting-time identity yields

$$\mathbf{h}^*(\mathbf{x}_s, s, t) = \mathbf{v}^*(\mathbf{x}_s, s) - (s-t)\left[\partial_s \mathbf{h}^*(\mathbf{x}_s, s, t) + (\partial_{\mathbf{x}}\mathbf{h}^*)(\mathbf{x}_s, s, t)\mathbf{v}^*(\mathbf{x}_s, s)\right].$$

This is the basic Mean Flow identity: the average drift over a long interval is the instantaneous drift at the starting point, corrected by how the average drift changes as the starting point slides along the same trajectory.

Mathematical Equivalence of the Three Perspectives. For an exact ODE flow, the semigroup, Lagrangian, and Eulerian identities all express the same simple principle: under standard existence, uniqueness, and smoothness assumptions, an ODE determines a unique trajectory through each state–time pair. That is, once a state at a given time is fixed, there is only one path to follow. The semigroup view splits this path into consecutive pieces, the Lagrangian view follows its ending point as the ending time changes, and the Eulerian view moves the starting point along the same path while keeping the ending point fixed. Thus, the three identities are equivalent descriptions of the same ODE flow (Arnold, 1988; Chorin and Marsden, 1993). Their connection becomes especially clear by viewing the semigroup identity as the finite-interval statement, and the Lagrangian and Eulerian identities as its two infinitesimal forms, obtained by shrinking a short interval near the ending time or near the starting time, respectively.

The semigroup identity gives the finite-interval form of this principle:

$$\boldsymbol{\Psi}_{u\to t}\big(\boldsymbol{\Psi}_{s\to u}(\mathbf{x}_s)\big) = \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s), \qquad \boldsymbol{\Psi}_{s\to s} = \mathbf{I}.$$

The two differential views are obtained by shrinking one of the two intervals.

Ending-Time Limit: the Lagrangian Identity. Fix the starting state–time pair $(\mathbf{x}_s, s)$ and shrink the final segment of the trip. For $\Delta t > 0$,

$$\boldsymbol{\Psi}_{t\to t-\Delta t}\big(\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)\big) = \boldsymbol{\Psi}_{s\to t-\Delta t}(\mathbf{x}_s).$$

Let $\mathbf{x}_t := \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)$. A first-order expansion of the short flow gives

$$\boldsymbol{\Psi}_{t\to t-\Delta t}(\mathbf{x}_t) = \mathbf{x}_t - \Delta t\, \mathbf{v}^*(\mathbf{x}_t, t) + \mathcal{O}\big((\Delta t)^2\big).$$

Therefore,

$$\frac{\boldsymbol{\Psi}_{s\to t-\Delta t}(\mathbf{x}_s) - \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s)}{-\Delta t} = \mathbf{v}^*(\mathbf{x}_t, t) + \mathcal{O}(\Delta t).$$

Taking $\Delta t \to 0$ yields the Lagrangian identity

$$\partial_t \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{v}^*\big(\boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s), t\big).$$

Thus, when the starting state–time pair $(\mathbf{x}_s, s)$ is fixed and the ending time changes, the ending point moves with the ODE velocity evaluated at that point.

Starting-Time Limit: the Eulerian Identity. Now fix the ending time t and move the starting state–time pair backward by a short interval $\Delta s > 0$ along the same trajectory:

$$\mathbf{x}_{s-\Delta s} := \boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s) = \mathbf{x}_s - \Delta s\, \mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}\big((\Delta s)^2\big).$$

The semigroup identity gives

$$\boldsymbol{\Psi}_{s-\Delta s\to t}(\mathbf{x}_{s-\Delta s}) = \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s).$$

Expanding the left-hand side in both the starting state and the starting time gives

$$\boldsymbol{\Psi}_{s-\Delta s\to t}(\mathbf{x}_{s-\Delta s}) = \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) - \Delta s\, \big(\partial_{\mathbf{x}} \boldsymbol{\Psi}_{s\to t}\big)(\mathbf{x}_s)\mathbf{v}^*(\mathbf{x}_s, s) - \Delta s\, \partial_s \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) + \mathcal{O}\big((\Delta s)^2\big).$$

Comparing with the semigroup identity and dividing by $-\Delta s$ gives

$$\partial_s \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) + \big(\partial_{\mathbf{x}} \boldsymbol{\Psi}_{s\to t}\big)(\mathbf{x}_s)\mathbf{v}^*(\mathbf{x}_s, s) = \mathcal{O}(\Delta s).$$

Taking $\Delta s \to 0$ yields the Eulerian identity

$$\partial_s \boldsymbol{\Psi}_{s\to t}(\mathbf{x}_s) + \big(\partial_{\mathbf{x}} \boldsymbol{\Psi}_{s\to t}\big)(\mathbf{x}_s)\mathbf{v}^*(\mathbf{x}_s, s) = 0.$$

Here, the ending time t is fixed, while the starting state–time pair moves along the same trajectory. The ending point therefore remains unchanged.

Recovering the Flow Map from Either Differential Identity. Conversely, either differential identity together with the boundary condition $\mathbf{\Psi}_{s\to s} = \mathbf{I}$ recovers the same flow map. The Lagrangian identity evolves the ending point from $\mathbf{\Psi}_{s\to s}(\mathbf{x}_s) = \mathbf{x}_s$. The Eulerian identity says that $\mathbf{\Psi}_{\tau\to t}(\mathbf{x}_\tau)$ is constant along the characteristic $\mathbf{x}_\tau = \mathbf{\Psi}_{s\to\tau}(\mathbf{x}_s)$, hence

$$\mathbf{\Psi}_{s\to t}(\mathbf{x}_s) = \mathbf{\Psi}_{t\to t}(\mathbf{x}_t) = \mathbf{x}_t.$$

Although these perspectives are equivalent at the oracle level, they lead to different practical algorithms once approximations are introduced: one must choose which identity to enforce, where to place stop-gradient targets, how to sample time pairs, and how to parameterize the map.

Closing. The central shift explored in this chapter is from learning a local velocity field to learning finite pieces of its ODE solution map. A conventional diffusion sampler repeatedly evaluates the local field and numerically integrates it. A flow-map model instead amortizes part of this integration during training, so that a long interval can be traversed with one or a few network evaluations.

This creates a direct route to fast generation, but moves difficulty from sampling into learning. The model must approximate a more global object, and its success depends on the parameterization of the map, the construction of tractable targets, the sampling and weighting of time pairs, and the stabilization of optimization.

Viewed in this way, flow-map learning is not a new dynamical concept but a modern learning strategy built around a classical one. It preserves the trajectory-based foundation of diffusion models while enlarging the design space for efficient generation. The objective is no longer only to integrate the generative ODE more efficiently at inference time, but to learn useful finite-time transitions of its solution map directly.

Epilogue and Outlook

12

A Unifying Principle and the Road Ahead

The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.

Edsger W. Dijkstra

Every chapter of this book has been about the same thing: how probability mass moves under a prescribed transformation, and how to reverse, approximate, or exploit that motion to generate data. In this final chapter, we step back to make this structural unity explicit. We first revisit the change-of-variable principle that underpins everything we have covered (Section 12.1), then explain how the same probability-transport viewpoint extends to discrete state spaces through Markov kernels and continuous-time Markov chains (Section 12.2), and close with reflections on the broader landscape (Section 12.3).

After this chapter, you will be able to...

- explain probability transport as the common principle underlying the generative modeling frameworks developed throughout this book;
- connect probability evolution in continuous spaces, through the continuity and Fokker–Planck equations, with its finite-state counterpart, the master equation;
- explain how reverse modeling changes in discrete state spaces, where transition kernels, probability ratios, and jump rates replace ordinary Euclidean score and velocity fields;
- compare the variational, score/ratio, and flow perspectives across continuous and discrete state spaces, identifying what remains common and what changes.

Reader route. This chapter is an epilogue rather than a new method chapter. Read it as a synthesis: focus on which principles remain unchanged, and which mathematical objects must change, when moving from continuous to discrete state spaces.

12.1 The Density-Transport Backbone

Throughout this book, we have followed two closely related lines of development. The first is *diffusion models themselves*, including variational, score-based, and flow-based formulations, which offer different mathematical descriptions of how probability laws evolve and how such evolution may be reversed for generation. The second is *diffusion-motivated fast generators*, most notably flow map models, which are built on the same transport viewpoint but aim to learn more direct and efficient generators. Despite their differences in form, all of these methods are rooted in a single organizing idea: generative modeling is fundamentally a problem of *transporting probability mass*. Starting from a simple reference distribution, such as Gaussian noise, we seek to construct a process that gradually connects it to a complex data distribution.

This viewpoint is useful because it shifts attention away from individual model families and toward a more fundamental question: *how does a probability law evolve when the underlying samples are moved by a prescribed dynamics?* The mathematical language for answering this question is the change-of-variable formula and its continuous-time extensions. In this sense, the change-of-variable principle is not merely a technical tool; it is the conceptual backbone underlying both diffusion models and the fast generators they inspire.

A Common Three-Step Structure. At a high level, the methods studied in this book all follow the same simple pattern:

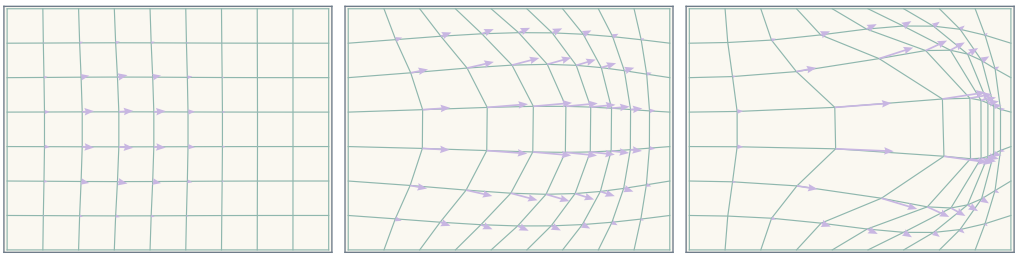
1. Define a forward process that gradually moves data toward a simple reference distribution.

2. Understand how the probability law evolves under that process through the appropriate law-evolution equation.

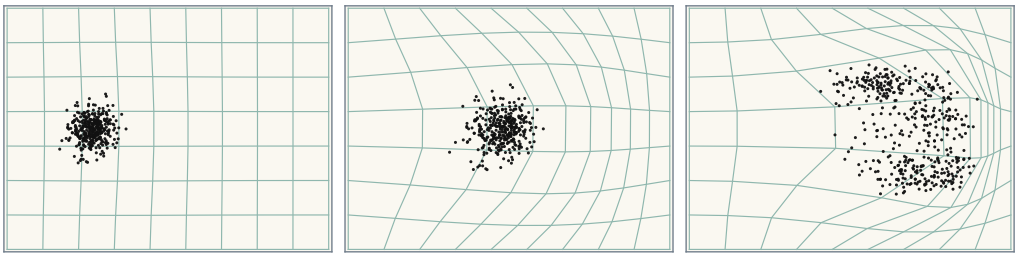
3. Learn a generative mechanism that reverses, approximates, or otherwise exploits this evolution in order to produce samples.

What varies across formulations is the choice of forward process, the mathematical form of the law-evolution equation, and the way the generative mechanism is parameterized and trained.

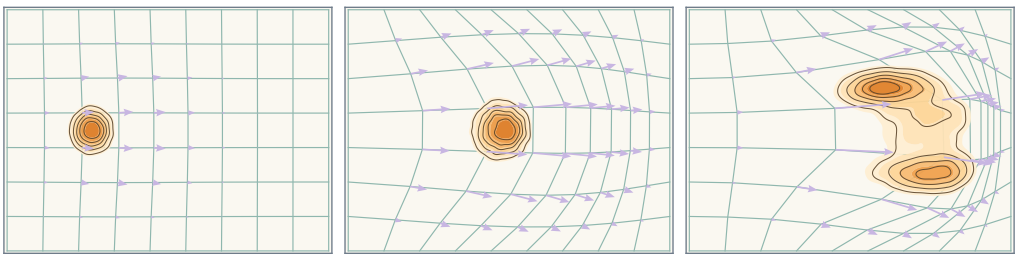
The Change-of-Variable Hierarchy. In Chapter B, we developed a systematic progression of change-of-variable formulas, from deterministic bijections to stochastic differential equations. That progression is worth recalling here, because it reveals a single transport principle appearing in several forms.



(a) Vector field illustrations. The arrows represent velocities that move particles through space, deforming the underlying grid accordingly.



(b) Particle-cloud dynamics. A predefined vector field generates a flow that transports particles from their initial state.



(c) Density evolution. As particles are advected by the vector field, the density contours deform accordingly, reflecting how the flow reshapes the underlying distribution.

Figure 12.1: Illustrations of particle and density dynamics under a vector field. Each column shows successive time snapshots from left to right.

Source: Adapted from Lipman et al. *(2024).*

1. **Single Bijection.** A smooth invertible map $\mathbf{\Psi} : \mathbb{R}^D \to \mathbb{R}^D$ transforms a density p_0 into p_1 via
$$p_1(\mathbf{x}_1) = p_0\big(\mathbf{\Psi}^{-1}(\mathbf{x}_1)\big)\left|\det \partial_{\mathbf{x}_1} \mathbf{\Psi}^{-1}(\mathbf{x}_1)\right|.$$
The Jacobian determinant records the local change of volume: when the map stretches space, the density decreases; when it compresses space, the density increases.

2. **Composed Bijections.** Chaining many bijections, with $\mathbf{x}_k = \mathbf{\Psi}_k(\mathbf{x}_{k-1})$, yields
$$\log p_L(\mathbf{x}_L) = \log p_0(\mathbf{x}_0) - \sum_{k=1}^{L} \log\left|\det \partial_{\mathbf{x}_{k-1}} \mathbf{\Psi}_k(\mathbf{x}_{k-1})\right|.$$

This is the basic principle behind normalizing flows. More importantly, it already illustrates the central message: once particle motion is specified, the evolution of the density follows accordingly.

3. **Continuous-Time Deterministic Flow (ODE).** Taking the infinitesimal-step limit leads to the continuity equation,

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot \big(p_t(\mathbf{x})\, \mathbf{v}(\mathbf{x}, t)\big) = 0,$$

where particles evolve according to

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t), t).$$

This is the continuous-time expression of mass conservation. Probability mass is not created or destroyed; it simply flows through space under the velocity field $\mathbf{v}$.

4. **Continuous-Time Stochastic Flow (SDE).** Adding Brownian noise yields the Fokker–Planck equation,

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot \big(\mathbf{v}(\mathbf{x}, t)\, p_t(\mathbf{x})\big) + \frac{1}{2} g^2(t)\, \Delta p_t(\mathbf{x}),$$

which corresponds to the stochastic dynamics

$$\mathrm{d}\mathbf{x}(t) = \mathbf{v}(\mathbf{x}(t), t)\, \mathrm{d}t + g(t)\, \mathrm{d}\mathbf{w}(t).$$

The first term describes directed transport by the drift, while the second captures the random spreading induced by noise.

These are not unrelated formulas. The multi-layer change-of-variable rule is the discrete-time version of transport under repeated deterministic transformations. Its infinitesimal deterministic limit gives the continuity equation. Adding stochastic perturbation leads to the Fokker–Planck equation. Each level extends the previous one, while preserving the same underlying principle: probability mass is conserved, and the probability law adjusts to reflect how the underlying dynamics moves samples through space.

From this perspective, the main formulations in this book are easier to place in context. Variational methods describe generation through denoising transitions on a time grid. Score-based methods work with stochastic dynamics and learn the score field governing the reverse-time SDE. Flow-based methods work with deterministic transport and learn a velocity field whose induced marginal distributions follow a prescribed family of intermediate laws. Fast generators, in turn, build on this same backbone, but seek more direct ways to exploit the learned transport for efficient sampling.

The central takeaway is therefore simple. Diffusion modeling is not best understood as a single algorithm tied to one architecture or one noise schedule. It is a *design principle*: specify a forward transport process, understand the induced evolution of probability laws, and then construct a generator on top of that structure. Once this viewpoint is in place, many seemingly different methods become natural variations of the same idea.

A natural question then arises: does this principle extend beyond continuous state spaces? In the next section, we show that it does.

12.2 Beyond Continuous States: Probability Evolution on Discrete Spaces

So far, every model in this book has operated under a common structural assumption: the state on which the generative dynamics acts lives in a continuous Euclidean space $\mathbb{R}^D$. Whether the entries of $\mathbf{x}$ represent pixel intensities, audio samples, or continuous latent variables, the mathematics is the same: particles move along continuous trajectories governed by an ODE or SDE, and the probability density evolves through the continuity or Fokker–Planck equation. The framework rests on the differential structure of $\mathbb{R}^D$: gradients and divergences are defined through derivatives, and deterministic dynamics can be described by differentiable trajectories generated by vector fields.

This distinction is about the modeling state, not about the raw modality. Discrete observations can also be handled by first choosing a continuous code and a readout. For a raw symbol a, let $\mathbf{E}(a) \in \mathbb{R}^d$ be its code, and let $\mathbf{D}$ map codes back to symbols, or more generally to a categorical distribution:

$$\text{raw discrete symbol } a \xrightarrow{\mathbf{E}} \text{continuous code } \mathbf{E}(a) \in \mathbb{R}^d \xrightarrow{\mathbf{D}} \text{discrete readout.}$$

The representation may be fixed, such as a one-hot, categorical-vector, or simplex-valued code (Richemond *et al.*, 2022); on an injective codebook, the readout can be deterministic and may satisfy $\mathbf{D}(\mathbf{E}(a)) = a$. Alternatively, the representation may be learned, such as a token embedding or an autoencoder latent (Li *et al.*, 2022; Dieleman *et al.*, 2022); then $\mathbf{D}$ is a learned readout and $\mathbf{D} \circ \mathbf{E}$ need not be exactly invertible. Once the representation is fixed, the diffusion is continuous-state: Gaussian noising in an ambient code space, or a geometry-adapted continuous process for a constrained code space, is handled by the continuous machinery developed earlier in the book. The finite-state route studied below is different: the evolving state itself remains discrete.

The present section studies the complementary case, where the state itself remains finite and discrete state throughout the forward and reverse processes. Many data types naturally fit this view: text is a sequence of vocabulary tokens, molecular graphs use finitely many atom and bond types, and protein sequences are strings over a finite alphabet. In such a finite state space there is no canonical infinitesimal displacement from one state to another, no smooth trajectory through token values, and no ordinary gradient with respect to the state itself. This raises a natural question:

Question 12.2.1

If diffusion is really about transporting probability mass, what does that idea become when the state space is finite?

The structural backbone remains the same, but the mathematical objects change. In continuous space, probability laws evolve under ODE/SDE dynamics

through the continuity or Fokker–Planck equation. In a finite state space, probability mass moves between states according to Markov transition kernels or jump rates, and the law evolves by a finite-dimensional conservation law: the *master equation*, also called the Kolmogorov forward equation. At an abstract level, both settings follow the same recipe:

Observation 12.2.1:

Choose a forward corruption process, understand its induced probability path, and learn a reverse transport mechanism.

In continuous Gaussian diffusion, this reverse mechanism appears as Gaussian reverse conditionals, scores, or velocity fields. In finite state spaces, it appears instead as reverse transition probabilities, probability ratios, or jump rates. These are different mathematical objects, but they play the same conceptual role.

The goal of this chapter is not to present one discrete diffusion algorithm as definitive. Instead, we develop a roadmap that should remain stable as the literature changes. The roadmap is organized by the object used to describe reverse transport: transition kernels, probability ratios, or jump rates. We proceed in four steps: discrete-time transitions (Section 12.2.1), their continuous-time limit (Section 12.2.2), three perspectives on reverse modeling that mirror the variational, score/ratio, and flow viewpoints of the continuous case (Section 12.2.3), and a comparison highlighting what is genuinely different from the continuous setting (Section 12.2.4).

12.2.1 Discrete-Time Transitions

Consider a forward process over K discrete categories, where K includes any special corruption tokens, such as a mask token, required by the chosen forward process. Each state is represented as a *one-hot* column vector $\mathbf{x} \in \{0,1\}^K$ with exactly one entry equal to 1; the set of all such vectors is[1]

$$\mathcal{V} = \{\mathbf{x} \in \{0,1\}^K : \textstyle\sum_{i=1}^K x_i = 1\} = \{\mathbf{e}_1, \dots, \mathbf{e}_K\}.$$

We write $\mathbf{e}_i$ for the i-th standard basis vector, namely the one-hot encoding of category i. For example, each $\mathbf{e}_i$ can represent a token, a discrete latent code, an atom type, a bond type, or an amino acid.

[1] An equivalent formulation, standard in the CTMC and stochastic-processes literature (Norris, 1998; Kelly, 1979), represents each state as a scalar index $x \in \{1, \dots, K\}$, writes transition rates as $[\mathbf{Q}_t]_{ij}$, and the marginal as $p_t(x)$. The one-hot encoding adopted here is notationally equivalent but aligns more closely with the discrete diffusion literature (Austin *et al.*, 2021; Sahoo *et al.*, 2024; Lou *et al.*, 2024), where inner products such as $\langle \mathbf{x}_\phi, \mathbf{x} \rangle$ and matrix–vector products such as $\mathbf{P}^\top \mathbf{x}$ appear naturally in training objectives.

For sequence data such as text or proteins, a sample consists of N tokens, each taking values in a vocabulary of size K. The full state space is then $\mathcal{V}^N$, whose size grows exponentially with sequence length. In many discrete diffusion models, the *forward* corruption acts independently across positions, so the single-token Markov process is the basic local building block. We therefore focus first on one token, because it exposes the core probability-transport mechanism without notational overload. For full sequences, the same formulas apply to full configurations. If the forward corruption acts independently across positions, the conditional forward kernel factorizes positionwise.[2] The learned reverse model, however, need not be independent across positions: a Transformer or graph neural network can observe the entire corrupted object and use global context to predict local or global reverse transitions.

At step k, the system occupies one of these K states. We denote the scalar probability of being in state $\mathbf{e}_i$ by

$$p_k(i) := \Pr(\mathbf{x}_k = \mathbf{e}_i), \qquad i = 1, \dots, K.$$

Collecting these scalar probabilities gives the marginal probability vector

$$\mathbf{p}_k := \big(p_k(1),\ \dots,\ p_k(K)\big)^\top = \mathbb{E}[\mathbf{x}_k] \in \Delta^{K-1} \subset \mathbb{R}^K, \qquad \sum_{i=1}^{K} p_k(i) = 1,$$

where $\Delta^{K-1} := \{\mathbf{p} \in \mathbb{R}^K : p_i \geq 0,\ \sum_{i=1}^K p_i = 1\}$ denotes the probability simplex. Equivalently, $\Pr(\mathbf{x}_k = \mathbf{e}_i) = p_k(i)$, or briefly $\mathbf{x}_k \sim \mathrm{Cat}(\mathbf{p}_k)$.

Thus $\mathbf{p}_k$ is the full law of the discrete random state $\mathbf{x}_k$, while $p_k(i)$ is its i-th scalar entry. At continuous time we use the same convention:

$$p_t(i) := \Pr(\mathbf{x}_t = \mathbf{e}_i), \qquad \mathbf{p}_t := \big(p_t(1), \dots, p_t(K)\big)^\top.$$

The vector $\mathbf{p}_t$ is the discrete analogue of a density: it tells us how the total probability mass is distributed across the K states.

Following the convention used throughout this book, bold lowercase letters ($\mathbf{p}$, $\mathbf{x}$) denote vectors, bold uppercase letters ($\mathbf{P}$, $\mathbf{Q}$) denote matrices, and unbolded indexed quantities such as $p_k(i)$, $[\mathbf{P}_k]_{ij}$, and $[\mathbf{Q}_t]_{ij}$ denote scalar entries. Unbolded p with random variables as arguments, such as $p(\mathbf{x}_{k-1}|\mathbf{x}_k)$, denotes a probability mass function or transition kernel under the fixed forward process; p_ϕ denotes the learned reverse model.

How does the system move between states? A single transition step is specified by a *transition matrix* $\mathbf{P}_k \in \mathbb{R}^{K\times K}$. Its entry $[\mathbf{P}_k]_{ij}$ gives the probability that the system, currently in state $\mathbf{e}_i$, moves to state $\mathbf{e}_j$ in one step:

$$[\mathbf{P}_k]_{ij} \ = \ \Pr(\mathbf{x}_{k+1} = \mathbf{e}_j | \mathbf{x}_k = \mathbf{e}_i).$$

[2] The resulting marginal distribution over corrupted sequences generally need not factorize, because the clean-data distribution can contain dependencies across positions.

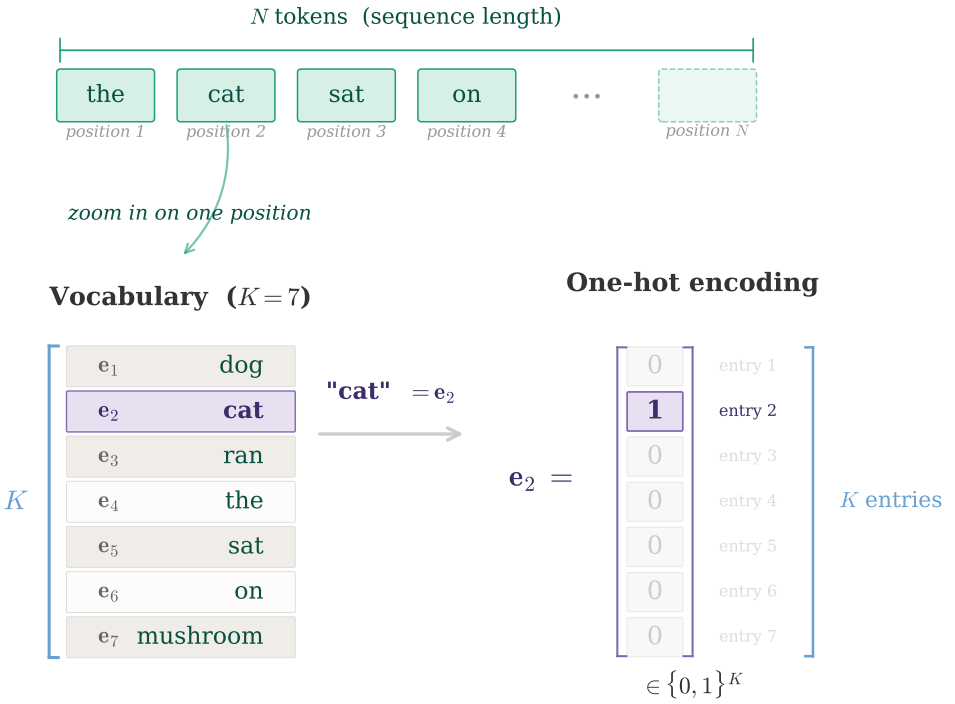


Figure 12.2: From sequences to one-hot states. A data sample such as a text or protein sequence consists of N tokens, each taking values in a finite alphabet or vocabulary of size K. This chapter focuses on the evolution of a single token among K states. Zooming in on one position, here "cat", the token is represented by the basis vector $\mathbf{e}_2$ and encoded as a one-hot vector in $\{0,1\}^K$, namely a K-dimensional vector whose second entry is 1 and all other entries are 0. The state space $\mathcal{V} = \{\mathbf{x} \in \{0,1\}^K : \sum_{i=1}^K x_i = 1\} = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$ is the set of all such one-hot vectors. The ordering of the vocabulary entries is arbitrary and unrelated to the position of a token in the sequence.

Source: Created by the authors with AI-assisted coding.

Thus the full matrix has the following structure:

$$\mathbf{P}_k = \underbrace{\begin{pmatrix} [\mathbf{P}_k]_{11} & [\mathbf{P}_k]_{12} & \cdots & [\mathbf{P}_k]_{1K} \\ [\mathbf{P}_k]_{21} & [\mathbf{P}_k]_{22} & \cdots & [\mathbf{P}_k]_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{P}_k]_{K1} & [\mathbf{P}_k]_{K2} & \cdots & [\mathbf{P}_k]_{KK} \end{pmatrix}}_{\text{columns: next state } j} \Bigg\} \text{rows: current state } i.$$

Reading across row i answers: "given that the system is currently in state $\mathbf{e}_i$, what

is the probability of landing in each possible next state?" Each row is therefore a probability distribution over destinations:

$$[\mathbf{P}_k]_{ij} \geq 0 \quad \text{for all } i, j, \qquad \sum_{j=1}^{K} [\mathbf{P}_k]_{ij} = 1 \quad \text{for each } i.$$

The non-negativity condition says every transition probability is valid, and the row-sum constraint says all mass leaving state $\mathbf{e}_i$ is accounted for: the system always ends up somewhere.

How the Distribution Evolves. With the transition matrix in hand, we now ask a basic question: if the system starts with distribution $\mathbf{p}_k$, what is the distribution after one step? The answer follows from simple probability accounting. The probability of being in state $\mathbf{e}_j$ after the transition is the total mass arriving at $\mathbf{e}_j$ from all possible source states:

$$p_{k+1}(j) = \sum_{i=1}^{K} p_k(i)\, [\mathbf{P}_k]_{ij}. \tag{12.2.1}$$

Each term in the sum represents the mass initially at state $\mathbf{e}_i$, weighted by the probability of jumping from $\mathbf{e}_i$ to $\mathbf{e}_j$. In vector form, this becomes

$$\mathbf{p}_{k+1} = \mathbf{P}_k^\top \mathbf{p}_k.$$

Equivalently, the conditional distribution of the next state given the current one-hot state $\mathbf{x}_k$ is

$$\Pr(\mathbf{x}_{k+1} | \mathbf{x}_k) = \mathrm{Cat}(\mathbf{x}_{k+1};\, \mathbf{P}_k^\top \mathbf{x}_k),$$

since $\mathbf{P}_k^\top \mathbf{e}_i$ extracts the i-th row of $\mathbf{P}_k$, which contains the transition probabilities from state $\mathbf{e}_i$.

A system evolving by this rule is called a *discrete-time Markov chain*: conditional on the current state $\mathbf{x}_k$, the distribution of $\mathbf{x}_{k+1}$ does not depend on the earlier states. This plays the finite-state role of the law-evolution formulas in continuous space. There, a bijection $\boldsymbol{\Psi}$ reshapes a density, and the Jacobian determinant compensates for local volume change. In a finite state space, there is no volume element to track. Instead, the transition matrix $\mathbf{P}_k$ directly specifies how much probability mass leaves each source state and arrives at each destination. The row-sum constraint guarantees that total probability is conserved, just as the divergence structure does in the continuity equation.

Repeating this over multiple steps, with transition matrices $\mathbf{P}_0, \ldots, \mathbf{P}_{L-1}$, yields

$$\mathbf{p}_L = \mathbf{P}_{L-1}^\top \cdots \mathbf{P}_1^\top \mathbf{P}_0^\top \, \mathbf{p}_0.$$

This mirrors the multi-layer change-of-variable formula in Equation (B.1.2). In the continuous deterministic setting, density evolution is accumulated through Jacobian

determinants; here, the probability vector is propagated through successive Markov operators. The underlying idea is the same: probability mass is tracked through a sequence of prescribed transformations.

Inverting a Single Step. Given the forward machinery, it is natural to ask how a single transition can be reversed for generation. Suppose the system is observed in state $\mathbf{e}_j$ at step $k{+}1$. What is the probability that it was in state $\mathbf{e}_i$ at the previous step k? This is a direct application of Bayes' rule:

$$\Pr(\mathbf{x}_k = \mathbf{e}_i | \mathbf{x}_{k+1} = \mathbf{e}_j) = \frac{\Pr(\mathbf{x}_{k+1} = \mathbf{e}_j | \mathbf{x}_k = \mathbf{e}_i)\ \Pr(\mathbf{x}_k = \mathbf{e}_i)}{\Pr(\mathbf{x}_{k+1} = \mathbf{e}_j)} = [\mathbf{P}_k]_{ij} \frac{p_k(i)}{p_{k+1}(j)}, \tag{12.2.2}$$

whenever $p_{k+1}(j) > 0$. This is the discrete-time reverse kernel associated with the forward transition $\mathbf{P}_k$ together with the current marginal law. Its form is intuitive: among all the ways that mass could have arrived at state $\mathbf{e}_j$, each source $\mathbf{e}_i$ contributes in proportion to how much mass was there, through $p_k(i)$, and how likely the jump was, through $[\mathbf{P}_k]_{ij}$. The numerator and denominator in Equation (12.2.2) each have a clear interpretation:

- **Numerator:** The term $[\mathbf{P}_k]_{ij}\, p_k(i)$ is the joint probability of being in state $\mathbf{e}_i$ at step k and then jumping to $\mathbf{e}_j$ at step $k{+}1$.
- **Denominator:** The term

  $$p_{k+1}(j) = \sum_{i'} p_k(i')\, [\mathbf{P}_k]_{i'j}$$

  is the total probability of arriving at state $\mathbf{e}_j$ from any source. It serves as the normalizing constant, ensuring that the reverse probabilities sum to one over i.

A crucial point is that the reverse kernel is *not* the matrix inverse of $\mathbf{P}_k$. A Markov transition may merge probability mass from many states into one state, and it need not be invertible as a linear map. The probabilistic reverse is instead a Bayesian inverse, and it depends not only on the forward transition matrix but also on the marginal distribution $\mathbf{p}_k$. This is exactly the difficulty faced in generative modeling: $\mathbf{p}_k$ is obtained by pushing the unknown data distribution through the forward process, so it is usually not available in closed form. The reverse-modeling perspectives below can be understood as different ways of representing or learning the missing information needed to implement this Bayesian inverse.

12.2.2 Continuous-Time Limit: The Master Equation

In the continuous-space story, shrinking the step size of composed bijections to zero yielded the continuity equation (Section B.1.2). The same limiting idea works

here. When the time steps become infinitesimally small, a discrete-time Markov chain becomes a continuous-time Markov chain (CTMC), and the distribution evolves according to the master equation.

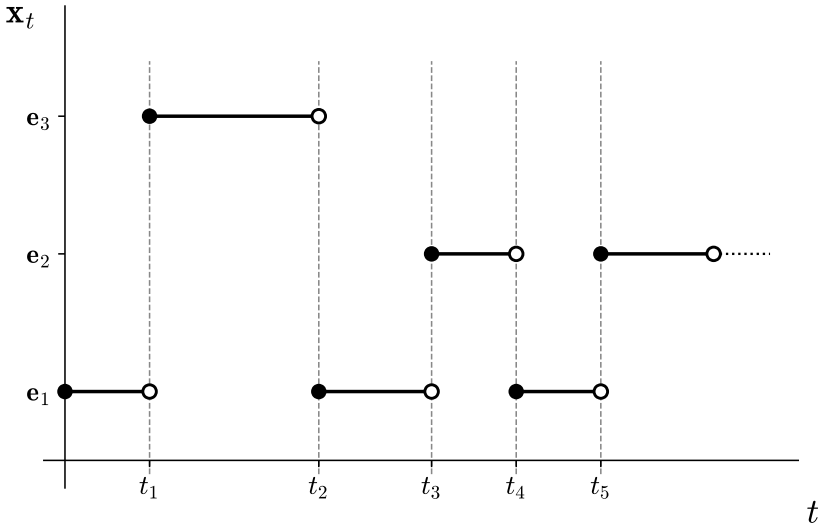


Figure 12.3: Illustration of a CTMC sample path. Let the state space be $\mathcal{V} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, labeled as S_1, S_2, S_3 on the vertical axis. The trajectory is piecewise constant: the process remains at $\mathbf{x}_t = \mathbf{e}_i$ for a random holding time, then jumps instantaneously to another state $\mathbf{e}_j$ at times $t_1, t_2, \ldots$. Filled circles mark the state occupied immediately after each jump, and open circles the state just before the jump, reflecting the right-continuous convention. The dotted segment indicates continuation beyond the displayed window. These jump times and destinations are governed by the rate matrix $\mathbf{Q}_t$: from state $\mathbf{e}_i$, the process jumps to $\mathbf{e}_j$ $(j \neq i)$ at instantaneous rate $[\mathbf{Q}_t]_{ij}$.

Source: Adapted from Campbell et al. *(2022).*

Infinitesimal Transitions. In the previous subsection, each transition matrix $\mathbf{P}_k$ described a full step from time k to time $k{+}1$. Now imagine making these steps finer and finer. Let $\mathbf{P}_{t,\Delta t}$ denote the transition matrix that moves the system forward by a small duration Δt starting at time t: its entry $[\mathbf{P}_{t,\Delta t}]_{ij}$ is the probability of moving from state $\mathbf{e}_i$ to state $\mathbf{e}_j$ during the interval $[t,\, t{+}\Delta t]$.

When Δt is small, most mass stays put and only a small fraction jumps. This means $\mathbf{P}_{t,\Delta t}$ is close to the identity, and we can expand it as

$$\mathbf{P}_{t,\Delta t} \;=\; \mathbf{I} + \mathbf{Q}_t\,\Delta t + \mathcal{O}(\Delta t^2), \tag{12.2.3}$$

where $\mathbf{Q}_t \in \mathbb{R}^{K\times K}$ is the *rate matrix*, also called the generator, at time t. If $\mathbf{Q}_t$ is sufficiently regular in time, the remainder can be written as $\mathcal{O}(\Delta t^2)$ locally. The entries of $\mathbf{Q}_t$ are instantaneous *rates* rather than probabilities. Specifically:

- $[\mathbf{Q}_t]_{ij} \geq 0$ for $i \neq j$: this is the rate at which the process jumps from state $\mathbf{e}_i$ to state $\mathbf{e}_j$. Multiplying by Δt gives the leading-order jump probability over a short interval.
- $[\mathbf{Q}_t]_{ii} = -\sum_{j \neq i} [\mathbf{Q}_t]_{ij}$: the diagonal entry is negative and records the total rate at which probability *leaves* state $\mathbf{e}_i$. This ensures that each row of $\mathbf{Q}_t$ sums to zero, which is the rate-level expression of probability conservation.[3]

Reading off the expansion, the probability of jumping from $\mathbf{e}_i$ to $\mathbf{e}_j$ with $j \neq i$ during a short interval is approximately $[\mathbf{Q}_t]_{ij}\,\Delta t$, while the probability of staying at $\mathbf{e}_i$ is approximately $1 - \sum_{j \neq i} [\mathbf{Q}_t]_{ij}\,\Delta t$.

Master Equation. With infinitesimal transitions in hand, the continuous-time limit follows immediately. Substituting the expansion in Equation (12.2.3) into $\mathbf{p}_{t+\Delta t} = \mathbf{P}_{t,\Delta t}^{\top}\,\mathbf{p}_t$ and taking $\Delta t \to 0$ gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\mathbf{p}_t = \mathbf{Q}_t^{\top}\,\mathbf{p}_t. \tag{12.2.4}$$

This is the *master equation*, also known as the *Kolmogorov forward equation* for CTMCs (Norris, 1998). Recall that $\mathbf{p}_t$ is the probability vector whose j-th entry $p_t(j)$ gives the probability of being in state $\mathbf{e}_j$ at time t. Writing out the j-th component of Equation (12.2.4) makes the meaning concrete:

$$\frac{\mathrm{d}}{\mathrm{d}t}\,p_t(j) = \underbrace{\sum_{i \neq j} p_t(i)\,[\mathbf{Q}_t]_{ij}}_{\text{inflow: mass arriving from other states}} - \underbrace{p_t(j) \sum_{\ell \neq j} [\mathbf{Q}_t]_{j\ell}}_{\text{outflow: mass departing to other states}}. \tag{12.2.5}$$

The rate of change of $p_t(j)$ is simply inflow minus outflow. This is probability conservation expressed as a differential equation.

Comparison with the Continuity Equation. In continuous space, the continuity equation

$$\partial_t\,p_t(\mathbf{x}) + \nabla \cdot \big(p_t(\mathbf{x})\,\mathbf{v}(\mathbf{x}, t)\big) = 0$$

says that density changes are driven by the net flux of probability mass under the velocity field $\mathbf{v}$. The master equation says the same thing in finite-state language: $p_t(j)$ changes because of net probability flux governed by the rate matrix $\mathbf{Q}_t$. The velocity field and the rate matrix play analogous structural roles: both prescribe

[3] This follows directly from the row-sum constraint on $\mathbf{P}_{t,\Delta t}$. Since each row of a transition matrix sums to 1, substituting the expansion $\mathbf{P}_{t,\Delta t} = \mathbf{I} + \mathbf{Q}_t\,\Delta t + o(\Delta t)$ gives $1 + \Delta t \sum_j [\mathbf{Q}_t]_{ij} + o(\Delta t) = 1$, which forces $\sum_j [\mathbf{Q}_t]_{ij} = 0$. Separating the diagonal term yields $[\mathbf{Q}_t]_{ii} = -\sum_{j \neq i} [\mathbf{Q}_t]_{ij}$.

how probability mass is redistributed, and both enforce conservation. The main difference is geometric: in $\mathbb{R}^D$, mass flows through neighboring spatial locations; in a finite state space, mass jumps along allowed edges of a transition graph.

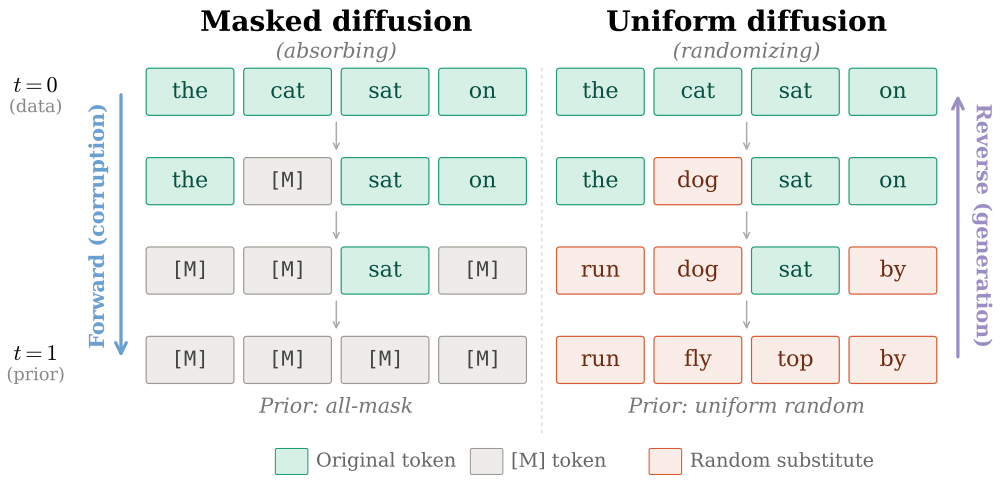


Figure 12.4: Illustration of masked and uniform diffusion forward corruption and reverse generation. The two examples shown here admit an interpolating marginal of the form $\Pr(\mathbf{x}_t|\mathbf{x}_0) = \mathrm{Cat}(\mathbf{x}_t; \alpha_t \mathbf{x}_0 + (1-\alpha_t)\boldsymbol{\pi})$, where the limiting prior $\boldsymbol{\pi}$ and decay factor α_t are determined by the chosen forward rate matrix. In masked diffusion, tokens are absorbed into the mask state, with limiting prior $\boldsymbol{\pi} = \mathbf{m} = \mathbf{e}_K$. In uniform diffusion, tokens are repeatedly randomized, with limiting prior $\boldsymbol{\pi}_{\text{unif}} = \frac{1}{K}\mathbf{1}$. Generation reverses the chosen corruption process, starting from the corresponding simple reference distribution, or from a close finite-time approximation to it.

Source: Created by the authors with AI-assisted coding.

Forward Corruption. Just as in the continuous case, we need a forward process that gradually destroys the structure of the data distribution and drives it toward a simple reference distribution that is easy to sample from. This is the discrete analogue of choosing a Gaussian noising process in continuous diffusion. The design goal is the same: the corruption should be progressive, analytically or computationally tractable, and strong enough that, by terminal time T, the data distribution has been largely washed out.

The rate matrix can be chosen in many ways. It may be uniform, absorbing, nearest-neighbor, graph-based, chemically structured, grammar-aware, or otherwise adapted to the data domain. The following two examples are therefore not meant to exhaust the taxonomy; they are canonical cases that make the mechanics transparent.

Masked Diffusion (Absorbing). Reserve the last category as a mask token, written $\mathbf{m} := \mathbf{e}_K$. Clean data $\mathbf{x}_0$ occupies one of the first $K-1$ categories, so $\langle \mathbf{x}_0, \mathbf{m} \rangle = 0$. The rate matrix is defined so that every ordinary state transitions to the mask state at rate $\beta(t)$, while the mask state never transitions out:

$$[\mathbf{Q}_t]_{iK} = \beta(t), \qquad [\mathbf{Q}_t]_{ii} = -\beta(t), \qquad [\mathbf{Q}_t]_{ij} = 0 \quad \text{for } i < K,\ j \notin \{i, K\},$$

and

$$[\mathbf{Q}_t]_{Kj} = 0 \quad \text{for all } j.$$

The idea is simple: each token is independently replaced by the mask at rate $\beta(t)$, and once masked it stays masked.

The forward marginal conditioned on a clean state $\mathbf{x}_0$ is

$$\Pr(\mathbf{x}_t | \mathbf{x}_0) = \mathrm{Cat}\big(\mathbf{x}_t;\ \alpha_t\, \mathbf{x}_0 + (1 - \alpha_t)\, \mathbf{m}\big),$$

where

$$\alpha_t = \exp\left(- \int_0^t \beta(s)\ \mathrm{d}s \right).$$

At time t, the token equals its clean value $\mathbf{x}_0$ with probability α_t and is masked with probability $1 - \alpha_t$. As time increases, more and more probability mass accumulates on the mask state. If $\int_0^T \beta(s)\, \mathrm{d}s$ is large, then α_T is small and the terminal distribution is close to the all-mask reference distribution.

Uniform Diffusion. Instead of erasing a token, this process gradually randomizes it. One convenient continuous-time normalization is to let a token jump to each of the other $K - 1$ states at the same rate:

$$[\mathbf{Q}_t]_{ij} = \frac{\beta(t)}{K - 1} \quad \text{for } i \neq j, \qquad [\mathbf{Q}_t]_{ii} = -\beta(t).$$

The first formula says that from state $\mathbf{e}_i$, the token jumps to each alternative state at the same instantaneous rate. The second says that the total rate of leaving state $\mathbf{e}_i$ is $\beta(t)$.

The forward marginal again takes the interpolating form

$$\Pr(\mathbf{x}_t | \mathbf{x}_0) = \mathrm{Cat}\big(\mathbf{x}_t;\ \alpha_t\, \mathbf{x}_0 + (1 - \alpha_t)\, \boldsymbol{\pi}_{\text{unif}}\big),$$

where $\boldsymbol{\pi}_{\text{unif}} = \frac{1}{K}\mathbf{1} \in \mathbb{R}^K$ is the uniform probability vector. Under the rate normalization above, the decay factor is

$$\alpha_t = \exp\left(- \frac{K}{K - 1} \int_0^t \beta(s)\ \mathrm{d}s \right).$$

As time passes, repeated randomization washes out the structure of the data distribution and pushes it toward the uniform distribution on $\mathcal{V}$.

More generally, the forward process need not have the simple mixture form

$$\Pr(\mathbf{x}_t|\mathbf{x}_0) = \mathrm{Cat}\big(\mathbf{x}_t;\ \alpha_t\mathbf{x}_0 + (1-\alpha_t)\boldsymbol{\pi}\big).$$

That form is useful for intuition and appears in common masked and uniform constructions, but the Markov law-evolution viewpoint is more general: transition matrices govern discrete-time evolution, and rate matrices govern continuous-time evolution through the master equation. Structured discrete diffusion models can use transition rules that respect edit distance, graph adjacency, molecular validity, vocabulary structure, or other domain knowledge. In all cases, the same principle remains: the forward Markov process defines a probability path, and generation learns to move along that path in reverse.

12.2.3 Three Perspectives on Reverse Modeling

Once the forward process and its law evolution are in place, the remaining task is the same as in the continuous case: learn a reverse process whose marginal distributions retrace the forward evolution in reverse time, turning simple reference samples into data. In Part B, we organized continuous diffusion models into three perspectives: variational (Chapter 2), score-based (Chapters 3 and 4), and flow-based (Chapter 5). The same three-way viewpoint carries over to the discrete setting, with one important caveat: these are not mutually exclusive schools or fixed paper categories. They are three ways of identifying what unknown object must be learned in order to reverse the probability path.

Variational Perspective: Learn Reverse Transition Kernels. In the continuous case (Section 2.2), the variational viewpoint treats diffusion as a hierarchical latent-variable model. The forward process defines a chain of increasingly noisy latents, and the goal is to learn reverse transitions that undo the corruption one step at a time. The true reverse kernel $\Pr(\mathbf{x}_{k-1}|\mathbf{x}_k)$ is intractable because it depends on the marginal law of $\mathbf{x}_k$, which involves the unknown data distribution. The key insight of DDPM is to condition on the clean sample $\mathbf{x}_0$: the forward posterior $\Pr(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0)$ is Gaussian in closed form, and Theorem 2.2.1 shows that training against this conditional posterior is equivalent, up to terms independent of the learned reverse model, to training against the intractable marginal reverse kernel. The ELBO then decomposes into a sum of KL terms (Equation (2.2.6)), one per noise level, each asking the learned model to match a tractable forward posterior.

The same logic applies in the discrete setting. Write p for the joint law obtained by drawing $\mathbf{x}_0$ from the data distribution and then applying the fixed forward chain, and write $p_{\boldsymbol{\phi}}$ for the learned reverse model. Consider a chain of corrupted one-hot states

$$\mathbf{x}_0 \xrightarrow{\mathbf{P}_0} \mathbf{x}_1 \xrightarrow{\mathbf{P}_1} \cdots \xrightarrow{\mathbf{P}_{L-1}} \mathbf{x}_L,$$

where $\mathbf{x}_0$ is drawn from the data distribution and $\mathbf{x}_L$ is close to a simple reference distribution, such as all-mask or uniform. The transitions are now categorical rather than Gaussian, but the variational structure is the same:

- The true reverse $p(\mathbf{x}_{k-1}|\mathbf{x}_k)$ is intractable, because it depends on the marginal at step k.
- Conditioning on $\mathbf{x}_0$ makes the forward posterior tractable via Bayes' rule:

$$p(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0) = \frac{p(\mathbf{x}_k|\mathbf{x}_{k-1})\, p(\mathbf{x}_{k-1}|\mathbf{x}_0)}{p(\mathbf{x}_k|\mathbf{x}_0)},$$

  where every factor on the right is determined by the forward process.
- Training against these conditional posteriors has the same optimal reverse kernel as training against the intractable marginal reverse kernels; the difference consists of terms independent of $p_{\boldsymbol{\phi}}$.

Schematically, the discrete diffusion ELBO contains denoising terms of the form

$$\mathbb{E}_{p(\mathbf{x}_{0:L})}\Big[\mathcal{D}_{\mathrm{KL}}\big(p(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0) \parallel p_{\boldsymbol{\phi}}(\mathbf{x}_{k-1}|\mathbf{x}_k)\big)\Big], \qquad k = 1, \ldots, L.$$

Depending on the exact model, the full ELBO also includes boundary terms such as the terminal prior term and the reconstruction term near $k = 0$. The key point is that each denoising term asks the learned reverse kernel to match a tractable Bayesian posterior determined by the forward corruption.

Here $p_{\boldsymbol{\phi}}(\mathbf{x}_{k-1} = \mathbf{e}_i|\mathbf{x}_k = \mathbf{e}_j)$ denotes the neural-network parameterized reverse transition probability: given that the system is in state $\mathbf{e}_j$ at step k, it gives the probability of having come from state $\mathbf{e}_i$ at step $k-1$. For each fixed $\mathbf{e}_j$, these values form a categorical distribution over previous states.

The main formal difference from continuous DDPM is that these KL divergences are finite sums over discrete states rather than closed-form Gaussian expressions. This reflects a special property of Gaussian distributions: the product of two Gaussian densities in the same variable is proportional to another Gaussian. In continuous DDPM, the forward posterior is computed by multiplying Gaussian factors, so the result is itself Gaussian and the KL divergence reduces to a simple algebraic expression involving only means and covariances (Section 2.2). In the discrete setting, Bayes' rule yields a categorical distribution instead, and each KL term is a sum over states. This sum is exact and conceptually simple, but for large vocabularies or full sequence spaces it may require structure, factorization, subsampling, or specialized parametrization.

A practical point about parametrization is also worth noting. The ELBO asks the learned reverse kernel $p_{\boldsymbol{\phi}}(\mathbf{x}_{k-1}|\mathbf{x}_k)$ to match a Bayesian posterior, but it does

not force one unique neural parametrization of that kernel. One common choice is a clean-state, or x_0-prediction, parametrization. The exact marginal reverse kernel satisfies the identity

$$p(\mathbf{x}_{k-1}|\mathbf{x}_k) = \sum_a p(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0 = \mathbf{e}_a)\, p(\mathbf{x}_0 = \mathbf{e}_a|\mathbf{x}_k),$$

where the sum is over valid clean categories. This is simply marginalization over the unknown clean state. Since the posterior $p(\mathbf{x}_0|\mathbf{x}_k)$ depends on the data distribution, it is not available in closed form. A model may therefore predict a clean-state distribution $\widehat{p}_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_k)$ and use it to define

$$p_{\boldsymbol{\phi}}(\mathbf{x}_{k-1}|\mathbf{x}_k) := \sum_a p(\mathbf{x}_{k-1}|\mathbf{x}_k, \mathbf{x}_0 = \mathbf{e}_a)\, \widehat{p}_{\boldsymbol{\phi}}(\mathbf{x}_0 = \mathbf{e}_a|\mathbf{x}_k).$$

If $\widehat{p}_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_k)$ were equal to the true posterior $p(\mathbf{x}_0|\mathbf{x}_k)$, this mixture would recover the exact reverse kernel. In practice, it is a useful way to tie the learned reverse transition to the known forward posterior.

This clean-state parametrization is common in multinomial/D3PM-style discrete-time diffusion models (Hoogeboom *et al.*, 2021; Austin *et al.*, 2021), but it is not the definition of discrete diffusion. D3PM itself notes that one could instead predict the reverse logits directly, and later models use other structured parametrizations tailored to the forward process (Sahoo *et al.*, 2024). Thus clean-state prediction should be viewed as one important parametrization of the variational perspective, not as the only way to model the reverse process.

Score/Ratio Perspective: Learn Local Probability Comparisons. In the continuous case (Chapters 3 and 4), the score-based viewpoint starts from a simple question: given a corrupted sample $\mathbf{x}_t \sim p_t$, what local information about the current density is needed to reverse the diffusion? The answer is the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, which appears in the reverse-time SDE and the probability-flow ODE. Learning the score with a neural network therefore suffices for generation.

In a finite state space, there is no ambient Euclidean geometry, no ordinary notion of infinitesimal displacement, and no gradient with respect to the state. A sample does not move by taking a small Euclidean step; it changes by jumping from one state to another. So the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ has no direct literal analogue. But the reverse process must still answer an analogous question: *if the system is currently in state* $\mathbf{e}_j$*, how likely is each candidate source state* $\mathbf{e}_i$ *relative to it?*

Deriving the Reverse Jump Rate. To identify the needed quantity, consider a forward CTMC $\{\mathbf{x}_t\}_{0 \le t \le T}$ with rate matrix $\mathbf{Q}_t$ and marginal $\mathbf{p}_t$. The entry $[\mathbf{Q}_t]_{ij}$ is the jump rate from state $\mathbf{e}_i$ to state $\mathbf{e}_j$. Over a short interval Δt, the probability of the infinitesimal forward event $\mathbf{x}_t = \mathbf{e}_i$ and $\mathbf{x}_{t+\Delta t} = \mathbf{e}_j$, for $i \neq j$, is

approximately

$$p_t(i)\,[\mathbf{Q}_t]_{ij}\,\Delta t.$$

Now define the reverse-time process $\bar{\mathbf{x}}_s := \mathbf{x}_{T-s}$ for $0 \le s \le T$. Its marginal at reverse time s is $\bar{\mathbf{p}}_s = \mathbf{p}_{T-s}$. If the reverse process has rate matrix $\bar{\mathbf{Q}}_s$, then the corresponding reverse jump from $\mathbf{e}_j$ to $\mathbf{e}_i$ must reproduce the same infinitesimal path probability in the reversed temporal order. Taking the limit $\Delta t \to 0$ gives the off-diagonal time-reversal formula

$$[\bar{\mathbf{Q}}_s]_{ji} = [\mathbf{Q}_{T-s}]_{ij}\,\frac{p_{T-s}(i)}{p_{T-s}(j)}, \qquad i \neq j, \tag{12.2.6}$$

whenever $p_{T-s}(j) > 0$. Equivalently, if t denotes the corresponding forward time and $\widetilde{\mathbf{Q}}_t$ denotes the reverse generator indexed by that forward time, then

$$[\widetilde{\mathbf{Q}}_t]_{ji} = [\mathbf{Q}_t]_{ij}\,\frac{p_t(i)}{p_t(j)}, \qquad i \neq j. \tag{12.2.7}$$

The diagonal entries are fixed by the row-sum-to-zero constraint,

$$[\widetilde{\mathbf{Q}}_t]_{jj} = -\sum_{i \neq j} [\widetilde{\mathbf{Q}}_t]_{ji}.$$

This is the standard time-reversal formula for CTMCs (Kelly, 1979; Anderson, 2012), applied to discrete diffusion by Campbell *et al.* (2022). The formula is understood only on states with positive marginal probability. Moreover, the reverse process only reverses probability flux along transitions allowed by the forward process: if $[\mathbf{Q}_t]_{ij} = 0$, then the corresponding reverse rate from $\mathbf{e}_j$ to $\mathbf{e}_i$ is also zero. The statement is not an equilibrium detailed-balance assumption; it is a local time-reversal identity for infinitesimal path probabilities.

The Ratio as a Discrete Analogue of the Score. Now inspect Equation (12.2.7). The forward rate $[\mathbf{Q}_t]_{ij}$ is known, since the forward process is fixed by design. The unknown quantity is the ratio $p_t(i)/p_t(j)$, which compares how likely state $\mathbf{e}_i$ is relative to state $\mathbf{e}_j$ under the current marginal. In inner-product notation, $p_t(i) = \langle \mathbf{p}_t, \mathbf{e}_i \rangle$, so the ratio can be written as $\langle \mathbf{p}_t, \mathbf{e}_i \rangle / \langle \mathbf{p}_t, \mathbf{e}_j \rangle$.

Why is this a discrete analogue of the score? In continuous space, the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ measures how the log-density changes under an infinitesimal displacement. In a finite state space, the natural local comparison is instead across an allowed jump from one state to another. Along a jump from $\mathbf{e}_j$ to $\mathbf{e}_i$, the change in log-probability is

$$\log p_t(i) - \log p_t(j) = \log \frac{p_t(i)}{p_t(j)},$$

a finite difference of log-probabilities, playing the role that a directional derivative of $\log p_t$ plays in $\mathbb{R}^D$. This comparison is naturally *edge-local*: the ratio is only needed for pairs of states connected by the forward jump structure. Thus, the discrete score is not a Euclidean vector field over $\mathbb{R}^D$; it is a collection of local probability comparisons along allowed transitions. For full sequences, the same statement applies with i and j replaced by full configurations, often neighboring configurations under a Hamming-type or graph-based transition structure.

What the Network Learns. In continuous score-based models, a neural network $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}, t)$ is trained to approximate the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, and the learned score is plugged into the reverse-time SDE to generate samples. The discrete case follows the same logic at the level of information: the network must provide the time-dependent ratio information needed to determine reverse jump rates via Equation (12.2.7). Generation then proceeds by simulating the reverse-time Markov chain or CTMC using the learned transition probabilities or rates.

Different formulations represent this information in different ways. Ratio-based models such as SEDD (Lou *et al.*, 2024) estimate ratios $p_t(i)/p_t(j)$ or their log-forms using losses inspired by score matching. Other continuous-time formulations (Campbell *et al.*, 2022; Sun *et al.*, 2023) learn equivalent reverse quantities through objectives such as matching conditional transition probabilities. The idea of treating local probability comparisons as a discrete analogue of the score is formalized by the *concrete score* of Meng *et al.* (2022). Despite these differences in parametrization, the underlying principle is shared: learn the local probability comparisons required by time reversal, construct the reverse transitions or rates, and simulate backward.

Flow Perspective: Learn a Discrete Transport Field. In the continuous case (Chapter 5), the flow-based viewpoint takes a different route: instead of learning the score and reversing an SDE, one directly learns a velocity field $\mathbf{v}_t(\mathbf{x})$ whose ODE flow transports a reference distribution to the data distribution. The training strategy is flow matching (Section 5.2): the marginal velocity field is intractable, but a *conditional* velocity field $\mathbf{v}_t(\mathbf{x}|\mathbf{x}_0)$, conditioned on a data or endpoint sample, is available in closed form. The marginal and conditional targets are related by

$$\mathbf{v}_t(\mathbf{x}) = \mathbb{E}\big[\mathbf{v}_t(\mathbf{x}|\mathbf{x}_0)|\mathbf{x}_t = \mathbf{x}\big],$$

an average over all possible conditioning variables given the current state. Computing this average explicitly would require knowing the continuous marginal density $p_t(\mathbf{x})$. The key training trick (Section 5.3.2) is that, under squared-error regression, the conditional target has the same population minimizer as the intractable marginal target. Training therefore avoids explicit marginal computation by using samples from tractable conditional paths.

The discrete case follows the same idea, with rate matrices replacing velocity fields. Since a discrete state cannot move along a continuous trajectory through token space, the role of the velocity field is played by a rate matrix, also called a Markov generator, $\mathbf{Q}_t$. Recall that the entry $[\mathbf{Q}_t]_{ij}$ specifies the jump rate from state $\mathbf{e}_i$ to state $\mathbf{e}_j$; the actual probability mass sent from $\mathbf{e}_i$ to $\mathbf{e}_j$ per unit time is the flux

$$J_t(i,j) = p_t(i)\,[\mathbf{Q}_t]_{ij}.$$

In this sense, a rate matrix is a discrete transport field: it determines how probability mass is redistributed over time.

There is one important subtlety. A probability path $\{\mathbf{p}_t\}_{t\in[0,T]}$ does not uniquely determine a rate matrix. Many different generators can produce the same derivative $\mathrm{d}\mathbf{p}_t/\,\mathrm{d}t$ in the master equation, just as different transport fields can sometimes realize the same marginal evolution under different geometric constraints. A discrete flow-matching method therefore specifies not only a probability path, but also a rule for assigning conditional jump rates or probability currents along that path.

As in continuous flow matching, the marginal rate matrix is generally intractable, but conditional versions can often be made explicit. If we condition on an endpoint, for example a clean data state and/or a reference state, one can prescribe a conditional probability path and derive tractable conditional jump rates along that path. A neural network is then trained, under an appropriate proper loss, to predict the corresponding transition or rate information from corrupted samples. The marginal generator is recovered through the same conditional-expectation principle: average the conditional transport object over the unknown endpoint distribution given the current state, but learn this average by supervised regression or classification against conditional targets rather than computing it explicitly.

The parallel with continuous flow matching is therefore:

- In continuous flow matching, the network is trained against a conditional velocity target; the marginal velocity emerges as the optimal predictor under the flow-matching loss.
- In discrete flow matching, the network is trained against conditional transition or jump-rate targets; the marginal discrete transport field emerges by the analogous conditional-expectation mechanism.

In both cases, training avoids the intractable marginal law by working with tractable conditional paths and sampled intermediate states.

This perspective is developed by Campbell *et al.* (2024) and Gat *et al.* (2024) under the heading of discrete flow models or discrete flow matching. More broadly, the flow perspective should be read as a statement about *what object is learned*: a transport field on a discrete state space, represented by transitions, rates, or probability currents.

Table 12.1: Continuous and discrete views of probability transport. The same generative modeling ingredients appear in both continuous and finite discrete state spaces, but the mathematical objects change: densities become probability vectors, ODE/SDE dynamics become Markov chains or CTMCs, and reverse modeling is expressed through transition kernels, score/ratio information, or discrete transport fields.

State	**Continuous** ($\mathbf{x} \in \mathbb{R}^D$)	**Discrete** ($\mathbf{x} \in \mathcal{V} \subset \{0,1\}^K$)
Law	Density $p_t(\mathbf{x})$	Probability vector $\mathbf{p}_t \in \Delta^{K-1}$
Sample Dynamics	ODE or SDE trajectories	Markov-chain or CTMC jumps
Law Evolution	Continuity or Fokker–Planck equation	Transition-matrix propagation or master equation
Transport Object	Velocity, drift, diffusion, or score field	Transition kernel, rate matrix, ratio, or probability current
Reference Law	Simple distribution, e.g. Gaussian	Simple distribution, e.g. uniform or all-mask
	Three Perspectives on Reverse Modeling	
Variational	ELBO with Gaussian or continuous transitions	ELBO with categorical transitions
Score/Ratio	Learn score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$	Learn ratios or log-ratios, e.g. $p_t(i)/p_t(j)$
Flow	Learn velocity field $\mathbf{v}_t(\mathbf{x})$	Learn transition, rate, or current field

12.2.4 What Is Similar and What Is Genuinely Different?

The following table summarizes the parallel between continuous and discrete diffusion models. Reading across each row shows the same conceptual ingredient instantiated in two different settings:
The parallel is clean, but the discrete setting is not simply the continuous theory with gradients replaced by ratios. Several deeper differences are worth highlighting.

No Literal Inverse Map. In a deterministic continuous normalizing flow, the reverse map is literally the inverse of the forward map. In a discrete Markov process, the forward kernel may merge mass from many states and need not be invertible. The reverse process is therefore not obtained by matrix inversion. It is a Bayesian or time-reversal construction that depends on the current marginal law. This is why reverse transition probabilities involve factors such as $p_k(i)/p_{k+1}(j)$, and reverse jump rates involve ratios such as $p_t(i)/p_t(j)$.

Geometry and Sample Paths. In continuous Euclidean space, individual samples evolve along continuous-time trajectories: ODE solutions are smooth, and SDE sample paths are continuous even though they are typically nowhere differentiable. This differential structure makes objects such as velocity fields, score functions, and divergences meaningful. In a finite state space, by contrast, an individual trajectory is a *pure-jump path*: it stays at one state for a random holding time and

then jumps abruptly to another. What evolves smoothly is not the sample path itself, but the probability vector $\mathbf{p}_t$, which evolves continuously in time on the probability simplex according to the master equation. Discrete diffusion therefore still admits a unifying description at the level of probability evolution, but its sample-level dynamics are governed by jump processes rather than by continuous ODE/SDE paths in the data space.

Forward Corruption Structure and Reverse Parametrization. In discrete diffusion, the forward corruption process is often a substantive modeling choice. In continuous Gaussian diffusion, one may vary the schedules α_t and σ_t, but the forward kernel often remains Gaussian, so the main effect is to control the rate and scale of corruption. In discrete diffusion, by contrast, the transition rule itself can change: one may use uniform corruption, masking, nearest-neighbor transitions, graph-structured transitions, domain-specific transitions, or more general probability paths. As emphasized by D3PM (Austin *et al.*, 2021), the forward process can therefore incorporate domain-specific inductive bias directly, and this in turn shapes the reverse generation problem.

This difference also changes the status of reverse parameterizations. In the Gaussian setting, common targets such as noise, denoised data, score, and velocity are linked by simple time-dependent affine transformations. In the discrete setting, reverse transition probabilities, clean-state predictors, probability ratios, and jump rates are connected through normalization, Bayes' rule, and the chosen transition graph. They are related, but not merely interchangeable Gaussian-style reparameterizations. As a result, the variational, score/ratio, and flow descriptions are genuinely different views of the reverse problem, not just different names for the same algebraic target.

Sampling Mechanism. The sampling procedure also differs. In continuous diffusion, generation is usually implemented by numerically integrating an ODE or SDE over a chosen time grid. In discrete diffusion, generation means simulating a reverse Markov chain or CTMC. In discrete time, this amounts to sampling categorical reverse transitions. In continuous time, it may involve simulating jumps from learned rates, for example by event-based simulation or time discretization. Thus, the role played by numerical ODE/SDE solvers in continuous diffusion is replaced by categorical transition sampling or jump-process simulation in the discrete setting.

Why Stop Here? Our goal in this section is not to develop the full machinery of discrete diffusion models, but to identify the principle that remains invariant across state spaces. The key point is structural: in continuous spaces, distributional

evolution is governed by change-of-variable principles and their differential forms, including the continuity equation and the Fokker–Planck equation. In discrete spaces, the corresponding role is played by Markov operators and the master equation. In both settings, the modeling logic is the same: define a forward corruption process, understand how it transports the distribution, and then learn a reverse transport mechanism for generation. This logic does not depend on whether the underlying state space is continuous or discrete.

What changes are the mathematical objects used to realize this program, not the program itself. In continuous spaces, one works with densities, gradients, velocity fields, and SDE or ODE dynamics. In discrete spaces, one works with probability vectors, transition kernels, jump rates, ratios, and probability currents. These differences matter technically, and they lead to distinct questions of parameterization, training, and sampling. But they do not alter the deeper conclusion: the probability-transport viewpoint is not tied to a particular state space, architecture, or paper lineage. It is a general organizing principle broad enough to encompass both continuous and discrete diffusion within a single conceptual framework.

12.3 Closing Remarks of the Book

Diffusion models may at first appear as a collection of acronyms, parametrizations, and algorithmic variants. Yet the central idea is simple. They provide a way to construct generators by prescribing how probability laws evolve over time. From this viewpoint, generation is not a mysterious black box, but the controlled transport of a cloud of particles under a forward process and its learned reverse.

Throughout this book, this viewpoint has appeared in several forms. In continuous state spaces, variational, score-based, and flow-based formulations provide different ways to describe and learn the reverse of a prescribed law evolution. Diffusion-motivated fast generators, such as flow-map models, belong to the same story as well: they do not abandon the transport viewpoint, but instead ask whether the underlying generative dynamics can be represented more directly and executed more efficiently. In discrete state spaces, the same structural idea reappears in the language of Markov kernels, jump processes, ratios, and master equations.

The main message of this book is therefore not that one particular formulation should replace all others. Rather, diffusion is best understood as a general principle for constructing generators from a prescribed forward process (Lai *et al.*, 2026). Under this viewpoint, different state spaces naturally lead to different reverse representations. In continuous settings, one may work with conditional Gaussians, score functions, velocity fields, or more direct flow maps. In discrete settings, one may work with reverse transition probabilities, clean-state predictors, probability ratios, jump rates, or probability currents associated with the underlying Markov process. What matters is not the label of the parametrization, but whether the forward process, the induced evolution of probability laws, and the learned generative mechanism fit together coherently.

We hope this book has made that structure clear: a forward process induces an evolution of probability laws, and generative modeling amounts to learning how to reverse, approximate, or exploit that evolution. Specific methods will continue to change, but this viewpoint gives a stable way to understand why they work, how they relate, and where new developments are likely to come from.

> *The important thing is not to stop questioning. Curiosity has its own reason for existence.*
>
> Albert Einstein

Appendices

A

Crash Course on Differential Equations

Differential equations (DEs) are fundamental tools for modeling dynamic systems and can be broadly categorized into *ordinary differential equations* (ODEs), *stochastic differential equations* (SDEs), and *partial differential equations* (PDEs).

ODEs describe how a system's state changes over time according to a precise rule, so that knowing the starting point determines the future path exactly. SDEs add randomness to this evolution, modeling how noise or uncertainty influences the system's behavior, making the outcome probabilistic rather than fixed. PDEs explain how functions depending on several variables, such as time and space, evolve together, capturing phenomena like heat spreading, waves moving, or *the time evolution of probability densities in stochastic systems* (Spoiler: Fokker-Planck equation). These types of differential equations form a fundamental language for understanding how systems evolve over time and space under both deterministic and random influences.

In this chapter, we provide essential prerequisites on differential equations.

After this chapter, you will be able to...

- interpret an ODE through its velocity field, solution trajectory, and the role of existence and uniqueness;
- use integrating factors to expose the linear structure of semilinear ODEs and understand why this structure is useful for diffusion ODE solvers;
- distinguish single-step, multistep, and parallel-in-time numerical viewpoints, and recognize Euler, Runge–Kutta, Adams–Bashforth, and Picard iteration as representative examples;
- understand how an SDE extends deterministic dynamics with Brownian noise, how its differential notation relates to the Itô integral, and how

Euler–Maruyama discretizes the resulting stochastic dynamics.

Reader route. This appendix provides the differential-equation background used throughout the book. Focus on the connection between dynamics, trajectories, and numerical approximation; the later diffusion chapters will specialize these ideas to probability-flow ODEs and stochastic diffusion processes.

A.1 Foundation of Ordinary Differential Equations

This section introduces the fundamental theory of ODEs, emphasizing the uniqueness of solutions given an initial condition. It also covers practical methods for solving ODEs using numerical solvers.

A.1.1 Intuition of Ordinary Differential Equation

The deterministic process is called an *ordinary differential equation* (ODE). In the multivariate case, we consider systems of the form:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t), t), \tag{A.1.1}$$

where $\mathbf{x}(t) \in \mathbb{R}^D$ is a vector-valued function representing the state of the system at time t, and $\mathbf{v} : \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$ is a vector field specifying the direction and magnitude of change at each point in space and time.

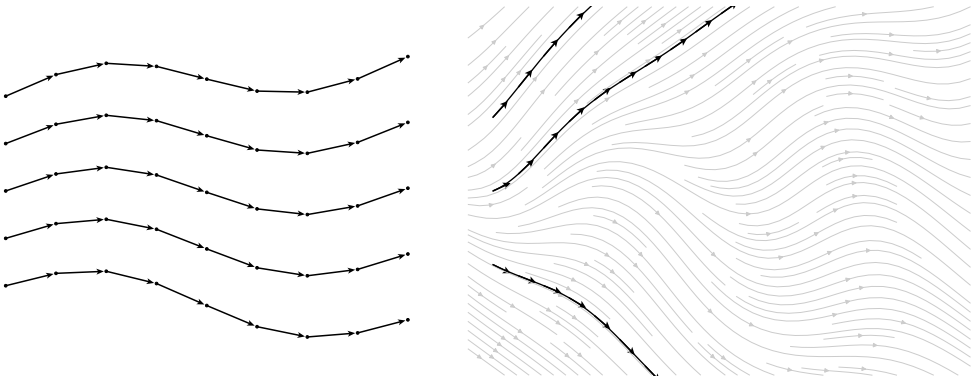


Figure A.1: ODE illustration. A velocity field $\mathbf{v}(\mathbf{x}, t)$ assigns a drift vector at every point. A solution trajectory $\mathbf{x}(t)$ is a path whose tangent always matches the local drift. The left panel shows step-by-step solver updates (dots and arrows) approximating the path, while the right panel shows the exact trajectories (black) flowing consistently with the velocity field. Without specifying an initial state $\mathbf{x}(0)$, there are infinitely many trajectories whose instantaneous changes match the same velocity field. Once $\mathbf{x}(0)$ is fixed, however, the ODE determines a unique path $\mathbf{x}(t)$ that flows according to the drift.

Source: Created by the authors.

High-Level Intuition for Solving ODEs. To build intuition, imagine the vector field $\mathbf{v}(\mathbf{x}, t)$ as a dynamic landscape of arrows that tells you how a point $\mathbf{x}$ should move at any given time t. Solving the differential equation means tracing out a curve $\mathbf{x}(t)$ through this field such that the tangent (i.e., the instantaneous velocity) of the curve at any point aligns with the vector given by $\mathbf{v}(\mathbf{x}(t), t)$.

- **Vector Field Perspective:** The function $\mathbf{v}(\mathbf{x}, t)$ defines how things should move: it gives the local "instructions" for motion or change.
- **Trajectory Perspective:** The solution $\mathbf{x}(t)$ is a path that a particle would follow if it obeys the rule set by the vector field $\mathbf{v}$ at every instant.

Thus, solving an ODE is like placing a particle in a flow field and observing where it goes over time.

A.1.2 Existence and Uniqueness of Ordinary Differential Equations

So far, we have seen that solving an ODE means finding a path that follows the directions given by the vector field at every point. Intuitively, this is like tracing the trajectory of a particle as it moves along the flow defined by the velocities.

But this picture leads to an important question:

Question A.1.1

If we pick a starting point, can we be sure there really is a path that follows these directions? And if there is, is that path unique, or could the particle suddenly jump onto a different trajectory?

Answering these questions is essential because it tells us whether the system's behavior can be reliably predicted from its starting position. The *Existence and Uniqueness Theorem* provides conditions on the vector field that guarantee exactly one path starting from any given initial point. This ensures the solution behaves consistently and forms a cornerstone of the theory of ODEs.

Local (in Time) Existence and Uniqueness Theorem. Below, we state a *local* version of the theorem, which asserts existence and uniqueness of a solution in a neighborhood of the initial time for a given initial condition.

Theorem A.1.1: Local Existence and Uniqueness

Let $\mathbf{v}(\mathbf{x}, t)$ be a continuous function with respect to $\mathbf{x}$ and t in a domain $D \subseteq \mathbb{R}^D \times \mathbb{R}$. If $\mathbf{v}$ satisfies the Lipschitz condition with respect to $\mathbf{x}$:

$$\|\mathbf{v}(\mathbf{x}_1, t) - \mathbf{v}(\mathbf{x}_2, t)\| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\| \quad \forall (\mathbf{x}_1, t), (\mathbf{x}_2, t) \in D,$$

where $L > 0$ is a constant, then for every initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$, there exists a unique solution $\mathbf{x}(t)$ to Equation (A.1.1) defined on some interval $[t_0 - \delta, t_0 + \delta]$.

Proof for Theorem.

(Proof Outline) The Existence and Uniqueness Theorem can be demonstrated constructively using the Picard-Lindelöf iteration method. The method generates a sequence of functions $\{\mathbf{x}_n(t)\}$ that converges to the solution $\mathbf{x}(t)$. The iteration is defined as:

$$\mathbf{x}_{n+1}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{v}(\mathbf{x}_n(s), s)\, \mathrm{d}s.$$

- Start with an initial guess $\mathbf{x}_0(t) = \mathbf{x}_0$.
- Iteratively refine $\mathbf{x}_n(t)$ using the integral form.
- Convergence is guaranteed under the Lipschitz condition by applying Contraction Mapping Theorem.

The essence of the proof is rooted in the Picard–Lindelöf iteration method, whose core idea is also leveraged in Section 9.6 to accelerate the sampling process of diffusion models.

Global (in Time) Existence and Uniqueness Theorem. While the Local Existence and Uniqueness Theorem guarantees the existence of solutions on a small time interval, the "global (in time) existence and uniqueness theorem" extends this result to the entire interval $[t_0, T]$ under additional regularity conditions. A well-known result in this category is the *Carathéodory theorem*, which ensures the global existence and uniqueness of solutions to ODEs under two key assumptions: local Lipschitz continuity in the state variable and a linear growth bound.

(i) **Local Lipschitz condition in $\mathbf{x}$:** There exists a function $\mathrm{Lip}(t)$, integrable on $[0, T]$, such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^D$,

$$\|\mathbf{v}(\mathbf{x}_1, t) - \mathbf{v}(\mathbf{x}_2, t)\| \leq \mathrm{Lip}(t)\|\mathbf{x}_1 - \mathbf{x}_2\|.$$

(ii) **Linear growth condition:** There exists a function $M(t)$, integrable on $[0, T]$, such that for all $\mathbf{x} \in \mathbb{R}^D$,

$$\|\mathbf{v}(\mathbf{x}, t)\| \leq M(t)(1 + \|\mathbf{x}\|).$$

We refer the reader to (Reid, 1971) for a comprehensive discussion of the assumptions, formal statement, and detailed proof of the theorem.

Remark.

To apply these theorems to the probability flow ODE in diffusion models (see Equation (4.1.7)), it may be necessary to impose additional assumptions, such as conditions (i) and (ii), on the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. These assumptions

can be reasonably accepted without further justification by readers not focused on technical details.

In summary, when an initial condition is given to an ODE defined by a time-dependent velocity field, the trajectory of the particle flow is uniquely determined.

Uniqueness Implies Non-Intersection of Solutions The uniqueness of solutions in ODEs, as guaranteed by the Local Existence and Uniqueness Theorem, implies a fundamental property: two different solution trajectories, starting from different initial conditions, cannot cross each other. This reflects the deterministic nature of ODEs, ensuring that each state evolves along a unique path. The following corollary formalizes this result.

Corollary A.1.1: Non-Intersection of Solutions

Consider two solutions $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ to the ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}\left(\mathbf{x}(t), t\right), \quad t \in [0, T].$$

Suppose they have distinct initial values $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$. Then, these solutions do not intersect on $[0, T]$, i.e.,

$$\mathbf{x}_1(t) \neq \mathbf{x}_2(t) \quad \text{for all } t \in [0, T].$$

Proof for Corollary.

Assume, for the sake of contradiction, that there exists some $t^* \in (0, T]$ such that

$$\mathbf{x}_1(t^*) = \mathbf{x}_2(t^*).$$

Define the first time at which the two solutions meet as

$$t_0 := \inf\{t \in [0, T] | \mathbf{x}_1(t) = \mathbf{x}_2(t)\}.$$

Since $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$ and t^* is contained in this set, it follows that $t_0 > 0$. By continuity of $\mathbf{x}_1$ and $\mathbf{x}_2$, we have

$$\mathbf{x}_1(t_0) = \mathbf{x}_2(t_0).$$

Consider the initial value problem

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_1(t_0).$$

By the uniqueness theorem for ODEs, both $\mathbf{x}_1$ and $\mathbf{x}_2$ must coincide on the interval $[t_0, T]$. Applying uniqueness backward in time similarly implies that the two solutions coincide on $[0, t_0]$. Therefore, the solutions satisfy

$$\mathbf{x}_1(t) = \mathbf{x}_2(t) \quad \text{for all } t \in [0, T],$$

which contradicts the assumption that $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$. Hence, we conclude that

$$\mathbf{x}_1(t) \neq \mathbf{x}_2(t) \quad \text{for all } t \in [0, T].$$

By guaranteeing non-intersecting solution paths, this theorem offers hidden yet crucial support for the flow map model (see Chapters 10 and 11).

A.1.3 Exponential Integration Factor

Even an ODE determined by a general time-varying velocity $\mathbf{v}$ does not admit a closed-form solution, in some special cases, we can solve them analytically or by reducing its formulation to a better structural one.

An Illustrative Example. Consider the following linear scalar ODE:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = L(t)\mathbf{x}(t),$$

where $L(t) \in \mathbb{R}$ is a continuous function. This equation is solvable in closed form, and its solution is well known (for any s and t):

$$\mathbf{x}(t) = \mathbf{x}(s) \cdot \exp\left(\int_s^t L(\tau)\ \mathrm{d}\tau\right).$$

This formula demonstrates how the solution evolves according to an exponential factor that accumulates the effect of the time-dependent coefficient $L(t)$. This motivates the use of *exponential integration factors*:

$$\mathcal{E}(s \to t) := \exp\left(\int_s^t L(\tau)\,\mathrm{d}\tau\right), \tag{A.1.2}$$

especially in more general settings where the dynamics include both linear and nonlinear components.

Semilinear ODEs and Exponential Integration Factors. We now consider a broader class of ODEs known as *semilinear* ODEs. These equations separate the dynamics into a linear part (in the state variable) and a nonlinear remainder:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = L(t)\mathbf{x}(t) + \mathbf{N}(\mathbf{x}(t), t), \tag{A.1.3}$$

where $\mathbf{x}(t) \in \mathbb{R}^D$ is the state vector, $L(t)$ is a scalar-valued continuous function, and $\mathbf{N} : \mathbb{R}^D \times [0, T] \to \mathbb{R}^D$ is a nonlinear vector field. This semilinear structure arises naturally in many physical and engineering systems. In particular, it also appears in the probability flow ODE formulation of diffusion models (see Equation (4.1.7)). Recognizing this structure enables integrating-factor representations that handle

the known linear dynamics analytically and leave the remaining term to be approximated numerically. This principle plays a central role in the diffusion ODE solvers studied in Chapter 9.

Step 1: Isolate the Non-Linear Term via an Integration Factor. Observing that we can isolate the nonlinear part by subtracting the linear drift from the semiliner ODE in Equation (A.1.3):

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} - L(t)\mathbf{x}(t) = \mathbf{N}(\mathbf{x}(t), t).$$

To absorb the linear term, we multiply both sides by the inverse integration factor:

$$\mathcal{E}^{-1}(s \to t) = \exp\left(-\int_s^t L(\tau)\,\mathrm{d}\tau\right).$$

Now apply the product rule to the left-hand side:

$$\mathcal{E}^{-1}(s \to t)\left(\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} - L(t)\mathbf{x}(t)\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left[\mathcal{E}^{-1}(s \to t)\mathbf{x}(t)\right].$$

Hence, the equation becomes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\mathcal{E}^{-1}(s \to t)\mathbf{x}(t)\right] = \mathcal{E}^{-1}(s \to t)\mathbf{N}(\mathbf{x}(t), t).$$

This transformation simplifies the original equation by isolating the nonlinear component, allowing us to focus entirely on the nonlinear dynamics in a transformed coordinate system.

Step 2: Integrate Over Time. We now integrate both sides from s to t:

$$\int_s^t \frac{\mathrm{d}}{\mathrm{d}\tau}\left[\mathcal{E}^{-1}(s \to \tau)\mathbf{x}(\tau)\right]\mathrm{d}\tau = \int_s^t \mathcal{E}^{-1}(s \to \tau)\mathbf{N}(\mathbf{x}(\tau), \tau)\,\mathrm{d}\tau.$$

The left-hand side is simply the difference of the transformed variable evaluated at t and s:

$$\mathcal{E}^{-1}(s \to t)\mathbf{x}(t) - \mathbf{x}(s).$$

Hence, we obtain:

$$\mathcal{E}^{-1}(s \to t)\mathbf{x}(t) = \mathbf{x}(s) + \int_s^t \mathcal{E}^{-1}(s \to \tau)\mathbf{N}(\mathbf{x}(\tau), \tau)\,\mathrm{d}\tau.$$

Step 3: Solve for $\mathbf{x}(t)$. Multiplying both sides by the exponential flow $\mathcal{E}(s \to t)$ gives the solution:

$$\mathbf{x}(t) = \underbrace{\mathcal{E}(s \to t)\mathbf{x}(s)}_{\text{linear part}} + \underbrace{\int_s^t \mathcal{E}(\tau \to t)\mathbf{N}(\mathbf{x}(\tau), \tau)\,\mathrm{d}\tau}_{\text{nonlinear part}}. \tag{A.1.4}$$

The solution naturally separates into a linear and a nonlinear component. Exponential integrators exploit this structure by solving the linear part in exactly closed form and discretizing only the nonlinear residual. This ensures that the step size is dictated by the nonlinear dynamics rather than by the potentially large linear coefficient (see, for instance, the comparison between the exponential Euler update and the vanilla Euler update in Section 9.1).

A.1.4 Numerical Solvers of Ordinary Differential Equations

We consider the ODE in Equation (A.1.1),

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t), t),$$

together with an initial condition. Solving the ODE means finding the continuous trajectory $\mathbf{x}(t)$ whose instantaneous change agrees with the velocity field $\mathbf{v}$ at every time.

Equivalently, the solution satisfies the integral equation

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{v}(\mathbf{x}(\tau), \tau)\, \mathrm{d}\tau. \tag{A.1.5}$$

If this integral cannot be evaluated in closed form, we approximate the trajectory numerically by evaluating the velocity field at finitely many points.

From a Continuous Trajectory to Numerical Updates. A numerical solver first chooses a discrete time grid

$$t_0, t_1, \ldots, t_M$$

and produces corresponding approximations

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_M$$

to the continuous trajectory.

The gap

$$h_i := t_{i+1} - t_i$$

is the *step size*. Smaller steps generally provide more local information about the trajectory but require more function evaluations. The purpose of a numerical solver is therefore to use the available evaluations of $\mathbf{v}$ efficiently so that accurate trajectories can be obtained with relatively few steps.

Different solvers mainly differ in *where the information used for the next update comes from*. This leads to the following useful organization.

A Structural View of Numerical Solvers. For the methods relevant to this book, it is useful to distinguish two broad numerical viewpoints.

- **Time-Stepping Methods:** Starting from the current state, the solver advances the trajectory from one time point to the next. Euler, Runge–Kutta, and Adams–Bashforth methods are representative examples.

  Within time-stepping methods, we further distinguish:

  - a *single-step method*, which constructs the next state from the current state and any additional evaluations generated within the current interval; and
  - a *multistep method*, which additionally reuses information from previous completed steps.

- **Parallel-in-Time Viewpoint:** Instead of completing all time points strictly one after another, one may formulate the trajectory computation so that tentative states at several times can be updated together. Fixed-point constructions such as Picard iteration provide a basic mechanism for this idea.

A single-step method should not be confused with a single *function evaluation*. For example, Euler uses one velocity evaluation per step, whereas a Runge–Kutta method may evaluate the velocity several times at intermediate *stages* while still being a single-step method. The defining distinction is whether information from earlier completed steps is required.

Mental Picture

The numerical ODE solver ideas used later for diffusion sampling can be organized as follows:

$$\begin{aligned} \textbf{Time-stepping} &\rightarrow \begin{cases} \text{single-step:} & \text{Euler, Runge–Kutta,} \\ \text{multistep} & \text{Adams–Bashforth,} \end{cases} \\ \textbf{Parallel-in-time} &\rightarrow \text{Picard fixed-point iteration.} \end{aligned}$$

$$\downarrow$$

The same numerical ideas can be applied forward or backward in time. (A.1.6)

The distinction is especially useful later in diffusion sampling: Runge–Kutta-type constructions obtain richer information by looking *within* the current step, whereas Adams–Bashforth-type constructions obtain it by looking *backward* at previously computed values.

Solving ODEs in Forward and Reverse Time. The taxonomy above does not depend on the direction in which the ODE is integrated. In ordinary forward integration,

$$t_{n+1} > t_n, \qquad h_n = t_{n+1} - t_n > 0.$$

Diffusion sampling instead typically starts from a noisy terminal state and uses a decreasing time grid,

$$t_{n+1} < t_n, \qquad h_n = t_{n+1} - t_n < 0.$$

The same numerical formulas apply in either direction when the step size is treated as a signed quantity. Thus, reverse-time integration of an ODE does not require a new family of numerical solvers; Euler, Runge–Kutta, multistep, and related methods can all be applied on a decreasing time grid.

This simple reversal should be distinguished from reversing an SDE, where the stochastic dynamics require additional analysis. We discuss that issue in the following section.

A.1.5 Representative Numerical Solvers

The taxonomy above tells us where a solver obtains information for its next update. We now make these ideas concrete through a few representative examples.

We deliberately keep the list short. Euler illustrates the simplest single-step update; Runge–Kutta methods illustrate the use of additional stages within the current step; Adams–Bashforth illustrates reuse of past history; and Picard iteration illustrates the fixed-point viewpoint behind parallel-in-time methods.

Throughout the examples below, let

$$t_{n+1} = t_n + h.$$

The step size h is signed: $h > 0$ for forward integration and $h < 0$ for reverse integration.

Euler's Method. Euler's method is the simplest single-step method. It uses only the velocity at the beginning of the current interval:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}(\mathbf{x}_n, t_n).$$

Geometrically, Euler looks at the current direction of motion and follows that direction for one complete step.

Under standard smoothness assumptions, Euler has local truncation error $\mathcal{O}(h^2)$ and global error $\mathcal{O}(h)$. Its simplicity makes it a useful first-order baseline, but its accuracy deteriorates when the velocity changes substantially over one step.

Runge–Kutta Methods. Euler uses only the velocity at the beginning of the interval. Runge–Kutta (RK) methods obtain more information by evaluating the velocity at additional *stages* generated within the same step.

They are therefore *single-step, multi-stage* methods: all information used to construct $\mathbf{x}_{n+1}$ is generated from the current starting point $\mathbf{x}_n$, but one step may contain several function evaluations.

We illustrate the basic RK principle using two second-order methods.

Explicit Midpoint Method. The explicit midpoint method first uses the current velocity to predict the state halfway through the interval:

$$\mathbf{x}_{\text{mid}} = \mathbf{x}_n + \frac{h}{2}\mathbf{v}(\mathbf{x}_n, t_n).$$

It then evaluates the velocity at this predicted midpoint and uses that direction for the complete update:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}\left(\mathbf{x}_{\text{mid}}, t_n + \frac{h}{2}\right).$$

Thus, instead of assuming that the starting velocity remains representative over the entire interval, the midpoint method "looks inside" the step before deciding how to complete it.

Heun's Method (Improved Euler). Heun's method is another two-stage, second-order Runge–Kutta method. Instead of looking at the midpoint, it first uses Euler's method to predict the endpoint:

$$\mathbf{x}_{\text{pred}} = \mathbf{x}_n + h\,\mathbf{v}(\mathbf{x}_n, t_n).$$

It then evaluates the velocity at this predicted endpoint and averages the starting and predicted ending velocities:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}\left[\mathbf{v}(\mathbf{x}_n, t_n) + \mathbf{v}(\mathbf{x}_{\text{pred}}, t_n + h)\right].$$

The explicit midpoint and Heun methods both use two stages and, under standard smoothness assumptions, have local truncation error $\mathcal{O}(h^3)$ and global error $\mathcal{O}(h^2)$. They differ only in how the additional within-step information is collected and combined.

Higher-order Runge–Kutta methods extend the same principle by using more stages; their standard formulas are omitted here. In diffusion sampling, Karras *et al.* (2022) use Heun's method directly as an ODE solver.

Adams–Bashforth Method. Runge–Kutta methods obtain additional information by looking *within* the current step. A multistep method instead looks *backward* and reuses information from previous completed steps.

For example, with a uniform step size h, the second-order Adams–Bashforth method (AB2) is

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\left[\frac{3}{2}\mathbf{v}(\mathbf{x}_n, t_n) - \frac{1}{2}\mathbf{v}(\mathbf{x}_{n-1}, t_{n-1})\right].$$

The second piece of information is therefore a stored velocity from the previous step rather than a new evaluation inside the current interval.

Under standard smoothness assumptions and sufficiently accurate initialization, AB2 has local truncation error $\mathcal{O}(h^3)$ and global error $\mathcal{O}(h^2)$. Because the first step has no previous history, a multistep method must first be initialized using another solver.

The formula above assumes a uniform grid. On nonuniform grids, the multistep coefficients depend on the neighboring step sizes. This history-reuse principle will later reappear in the diffusion-specific multistep solvers in Sections 9.3 and 9.5.4.

Picard Iteration. The methods above approximate the trajectory by constructing local time-stepping updates. Picard iteration follows a different idea: it repeatedly refines an entire candidate trajectory through the integral form of the ODE.

Starting from an initial guess $\mathbf{x}^{(0)}(t)$ satisfying $\mathbf{x}^{(0)}(0) = \mathbf{x}(0)$, Picard iteration defines

$$\mathbf{x}^{(k+1)}(t) = \mathbf{x}(0) + \int_0^t \mathbf{v}\big(\mathbf{x}^{(k)}(s), s\big)\,\mathrm{d}s.$$

Each iteration therefore evaluates the velocity along the previous candidate trajectory and uses those evaluations to construct a new one.

A fixed point of this iteration satisfies the original ODE integral equation. Under the standard conditions of the Picard–Lindelöf theorem, the iteration converges locally to the ODE solution.

For our later purposes, the important observation is computational: within one Picard iteration, the velocity at each time depends on the *previous* candidate trajectory. Evaluations at different time points can therefore be carried out independently, which opens the possibility of parallel-in-time computation. This viewpoint will be revisited in Section 9.6.

A.2 Foundation of Stochastic Differential Equations

Stochastic Differential Equations (SDEs) are an extension of ordinary differential equations (ODEs) that incorporate randomness, providing a mathematical framework for modeling systems affected by uncertainty. This chapter introduces SDEs, beginning with the discretization of ODEs, extending to the discretization of SDEs, and culminating in a discussion of general SDEs, including Ito's calculus and Ito's formula.

A.2.1 From ODEs to SDEs: An Intuitive Introduction

Let us begin with a ODE describing the deterministic evolution of a state variable $\mathbf{x}(t) \in \mathbb{R}^D$:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0. \tag{A.2.1}$$

Here, $\mathbf{f} : \mathbb{R}^D \times [0, T] \to \mathbb{R}^D$ is a time-dependent velocity field that governs the dynamics of $\mathbf{x}(t)$. The solution to this ODE is a smooth trajectory $t \mapsto \mathbf{x}(t)$, fully determined by the initial condition $\mathbf{x}_0$.

Discretization Perspective. To build intuition, consider an Euler discretization of Equation (A.2.1) over small time steps Δt:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t.$$

This approximation becomes more accurate as $\Delta t \to 0$, converging (under standard regularity conditions on $\mathbf{f}$) to the exact solution of the ODE.

Introducing Randomness: From ODE to SDE. In many real-world systems, perfect knowledge of the dynamics is unrealistic. Noise, uncertainty, or unmodeled interactions may affect the evolution. To incorporate such randomness, we augment the ODE with a stochastic term:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t + g(t)\sqrt{\Delta t} \cdot \boldsymbol{\epsilon}_t, \tag{A.2.2}$$

where

- $g : [0, T] \to \mathbb{R}$ is a diffusion coefficient (possibly dependent on both state and time, though here assumed time-dependent only),
- $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$ are i.i.d. standard Gaussian vectors.

This modified update rule reflects not just deterministic drift, but also random perturbations scaled by $\sqrt{\Delta t}$. The scaling ensures that the stochastic perturbation remains finite in the limit $\Delta t \to 0$. Importantly, this formulation gives rise to a *continuous-time stochastic process* as $\Delta t \to 0$, which leads us to the framework of SDE.

Stochastic Differential Equations. Formally, the limit of the discrete update Equation (A.2.2) as $\Delta t \to 0$ defines the SDE:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t). \tag{A.2.3}$$

Here, $\mathbf{w}(t) \in \mathbb{R}^D$ is a *Wiener process* (standard Brownian motion), a continuous-time stochastic process characterized by:

- **Initial State:** $\mathbf{w}(0) = 0$ almost surely;
- **Independent Increments:** for $0 \leq s < t$, the increment $\mathbf{w}(t) - \mathbf{w}(s)$ is independent of the past;
- **Gaussian Increments:**

$$\mathbf{w}(t) - \mathbf{w}(s) \sim \mathcal{N}(\mathbf{0}, (t-s)\mathbf{I}_D) \tag{A.2.4}$$

- **Continuity:** Sample paths $t \mapsto \mathbf{w}(t)$ are almost surely continuous but nowhere differentiable.

In addition, the notation

$$\mathrm{d}\mathbf{w}(t) := \mathbf{w}(t + \mathrm{d}t) - \mathbf{w}(t)$$

is often used to denote the infinitesimal increment of the Wiener process.

While suggestive, this notation is heuristic and should not be interpreted as a classical differential (e.g., in the Riemann or Lebesgue sense), since Brownian paths are almost surely nowhere differentiable. Instead, it serves as a formal shorthand to express the Gaussian increments property:

$$\mathrm{d}\mathbf{w}(t) \sim \mathcal{N}(0, \mathrm{d}t\,\mathrm{I}_D),$$

meaning that over an infinitesimal time interval of length $\mathrm{d}t$, the increment of the Wiener process behaves like a Gaussian random variable with zero mean and covariance $\mathrm{d}t\,\mathrm{I}_D$.

A.2.2 Further Explanation of Equation (A.2.3)

The SDE in Equation (A.2.3) should be understood in its *integral form*:

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{f}(\mathbf{x}(s), s)\,\mathrm{d}s + \int_0^t g(s)\,\mathrm{d}\mathbf{w}(s), \tag{A.2.5}$$

interpreted in the *Itô sense*. Here, the first term is a classical (Riemann or Lebesgue) integral representing the accumulated deterministic drift, while the second term is an *Itô stochastic integral*, which integrates with respect to the Wiener process $\mathbf{w}(t)$. We do not provide a full rigorous construction of the Itô integral, but offer the following intuition.

Intuition for Itô Integration. The Itô integral can be understood as the limit of discrete sums of the form[1]

$$\sum_i g(t_i)\big(w(t_{i+1}) - w(t_i)\big),$$

where the integrand $g(t)$ is evaluated at the *left* endpoint t_i of each subinterval. As the partition becomes finer, these sums converge to the Itô integral. The left-point evaluation rule is what distinguishes Itô integration from classical integrals, which may use midpoints or other evaluation rules. Because Brownian paths are continuous yet almost surely nowhere differentiable, classical integration does not directly apply. The Itô integral handles this irregularity and captures the cumulative effect of stochastic fluctuations over time.

Use of Differential Notation. Expressions such as $\mathrm{d}\mathbf{x}(t)$, $\mathrm{d}t$, and $\mathrm{d}\mathbf{w}(t)$ are not classical differentials. Instead, they are formal notations representing infinitesimal increments of the respective processes. While heuristic, they are widely used for their convenience in expressing SDEs analogously to ODEs and facilitate formal manipulations within Itô calculus.

How Itô calculus is applied in diffusion models will be explained in Chapter C.

Comparison with ODEs. In ODEs, e.g.,

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t),$$

the integral form

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \tau)\,\mathrm{d}\tau$$

is justified by the *Fundamental Theorem of Calculus*, which ensures that differentiable functions can be recovered from their derivatives.

By contrast, in SDEs such as Equation (A.2.3), there is no direct analog of this theorem because Brownian motion lacks differentiability, and stochastic integrals do not follow the classical chain rule. Instead, Itô calculus introduces alternative tools (e.g., Itô's lemma) to analyze and manipulate stochastic dynamics.

Thus, while the differential notation for SDEs is compact and intuitive, a rigorous understanding depends on interpreting them via their integral formulation using Itô integrals.

[1] For square-integrable integrands, this convergence holds in the L^2 sense (mean-square), which in turn implies convergence in probability. The L^2 viewpoint is natural because it connects directly to Itô's isometry, the key identity underlying the standard and rigorous construction of the Itô integral. This isometry links stochastic integrals to standard integrals in expectation ($\mathbb{E}\big[\big\|\int_0^T \boldsymbol{\psi}(t)\,\mathrm{d}\mathbf{w}_t\big\|^2\big] = \mathbb{E}\big[\int_0^T \|\boldsymbol{\psi}(t)\|_F^2\,\mathrm{d}t\big]$ with Frobenius norm $\|\boldsymbol{\psi}(t)\|_F^2$); we state and use it in Section C.1.5.

A.2.3 A Numerical Solver for SDE.

Like ODEs, the SDE in Equation (A.2.3) admits a unique solution[2] if $\mathbf{f}(\cdot, t)$ and $g(\cdot)$ satisfy some smoothness conditions: $\mathbf{f}(\cdot, t)$ is Lipschitz and of linear growth in $\mathbf{x}$, and $g(\cdot)$ is square integrable.

For general SDEs as in Equation (A.2.3), closed-form solutions are generally unavailable, so numerical methods are necessary. A common approach is the *Euler–Maruyama method*, which generalizes Euler's method for ODEs and, indeed, we have already seen it in Equation (A.2.2). It approximates the drift term $\mathbf{f}(\mathbf{x}(t), t)$ over a time step Δt and simulates the stochastic noise $g(t)\,\mathrm{d}\mathbf{w}(t)$ using Gaussian increments $\sqrt{\Delta t}\,\boldsymbol{\epsilon}_t$ with $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Later, in Section C.1.5, we will see that a linear SDE admits a closed-form solution.

[2] The solution is in the *strong* sense, meaning that $\mathbf{x}(t)$ satisfies the SDE in its integral form (see Equation (A.2.5)) with respect to the given Brownian motion $\mathbf{w}(t)$ on a fixed probability space. We omit the detailed technical definitions here.

B

Density Evolution: From Change of Variable to Fokker–Planck

Understanding how probability densities evolve under transformations is fundamental in both probability theory and generative modeling. In particular, diffusion models aim to construct generative processes whose induced density paths reverse a pre-defined forward process. This evolution is governed by the continuity equation or, in the stochastic case, the Fokker-Planck equation.

Although these names may sound unfamiliar or intimidating, they are in fact continuous-time analogues of the change-of-variable formula from basic calculus. In Section B.1, it builds up to them by presenting a progression of change-of-variable formulas, starting from deterministic bijections, and culminating in stochastic differential equations. This progression naturally bridges discrete mappings and continuous-time flow dynamics. See Figure B.1 for an overview of this unified framework.

In Section B.2, we provide a physical and intuitive interpretation of the continuity equation, emphasizing its connection to the conservation of density in dynamical systems.

After this chapter, you will be able to...

- connect the change-of-variable formula to the continuity equation for ODEs and the Fokker–Planck equation for SDEs;
- understand the continuity equation as conservation of probability mass, and distinguish particle motion from density evolution;
- distinguish the Lagrangian and Eulerian views of transport, including why a density path alone need not determine a unique velocity field;
- understand why an SDE and its probability-flow ODE can share the same marginal densities while having different sample paths.

Reader route. The key principle is that the change-of-variable formula describes how moving samples changes their probability distribution. This appendix develops that idea from deterministic maps to the continuity and Fokker–Planck equations, with Wasserstein gradient flow as an optional extension to distribution-level training.

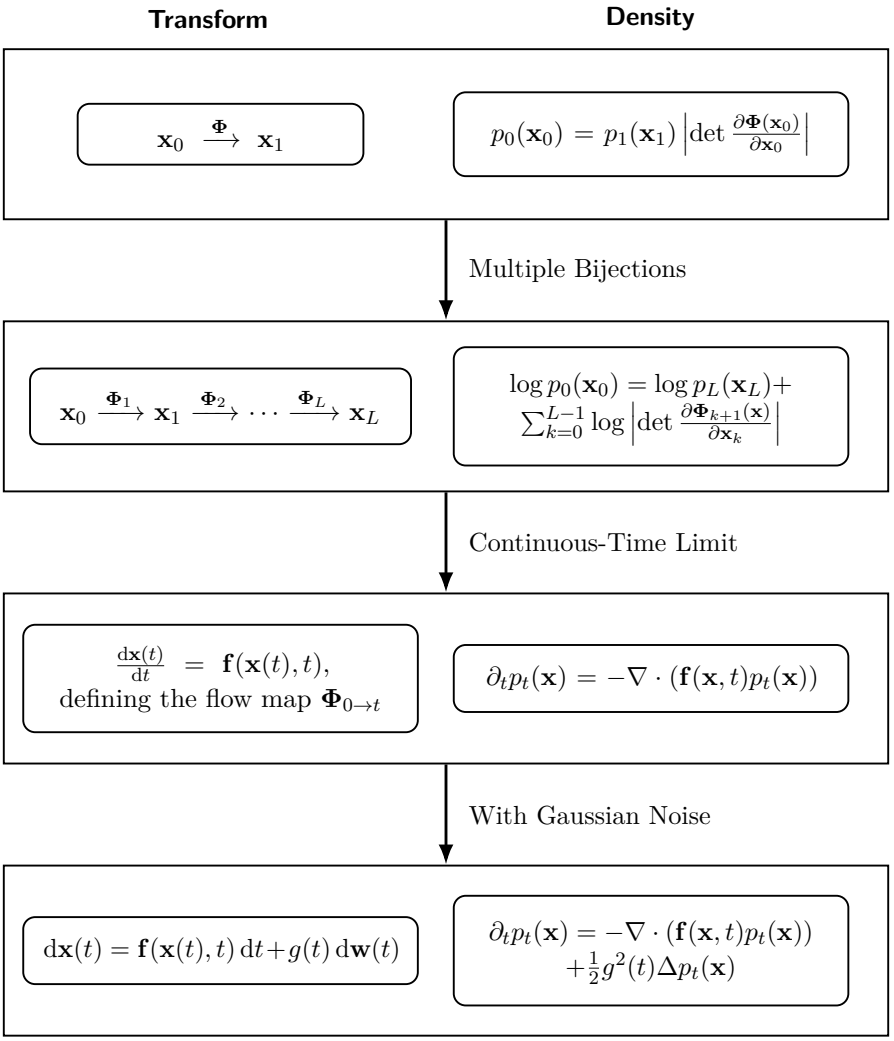


Figure B.1: A unified *change-of-variables formula*. From top to bottom: (1) a single bijection; (2) composition of multiple bijections; (3) continuous-time deterministic flow governed by an ODE and the associated continuity equation; (4) stochastic flow modeled by an SDE and the corresponding Fokker–Planck equation.

Source: Created by the authors.

B.1 Change-of-Variable Formula: From Deterministic Maps to Stochastic Flows

In this section, we aim to demystify the continuity equation and the Fokker-Planck equation by drawing analogies to the classic change-of-variable formula from calculus. We begin with the familiar single-variable case, extend it to the multivariate setting and to probability densities (Section B.1.1), then generalize to compositions of bijective maps whose continuous-time limit leads to the continuity equation (Section B.1.2). Finally, we incorporate stochasticity by introducing random noise, which naturally extends the continuity equation to the Fokker-Planck equation (Section B.1.3).

B.1.1 Change-of-Variable Formula for Deterministic Maps

We move particles according to a deterministic map and study how their *law* (density) evolves. The key principle is conservation of probability mass, grounded in a fundamental result from calculus and probability: the change-of-variable formula. This formula describes how integrals, and therefore probability densities, transform under smooth bijective mappings. To build intuition, we first consider a single update step, and then extend the discussion to sequential transformations.

Single Update. Think of a single deterministic update rule induced by applying a vector field (analogous to a force) $\mathbf{\Psi} : \mathbb{R}^D \to \mathbb{R}^D$ for one unit of time. Starting from an initial particle state $\mathbf{x}_0$, its next state is given by

$$\mathbf{x}_1 = \mathbf{\Psi}(\mathbf{x}_0).$$

Underlying Pattern (a Density) and How it Moves. If the initial states follow an underlying "law/pattern" described by a density p_0 (i.e., $\mathbf{x}_0 \sim p_0$), then applying $\mathbf{\Psi}$ produces a new density p_1 for $\mathbf{x}_1$ (i.e., $\mathbf{x}_1 \sim p_1$). Assuming $\mathbf{\Psi}$ is a smooth bijection, p_1 is obtained from p_0 via the standard *change-of-variables formula*:

$$p_1(\mathbf{x}_1) = p_0(\mathbf{\Psi}^{-1}(\mathbf{x}_1)) \cdot \left| \det \left(\frac{\partial \mathbf{\Psi}^{-1}}{\partial \mathbf{x}_1} \right) \right|. \tag{B.1.1}$$

Here $\frac{\partial \mathbf{\Psi}}{\partial \mathbf{x}}$ is the Jacobian matrix of $\mathbf{\Psi}$, denoted $\partial_{\mathbf{x}} \mathbf{\Psi}$. Equivalently, in the original coordinates,

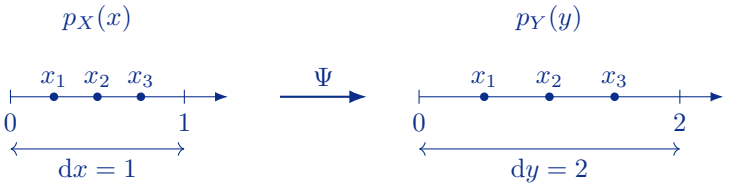
$$p_0(\mathbf{x}_0) \;=\; p_1(\mathbf{\Psi}(\mathbf{x}_0)) \; |\det \partial_{\mathbf{x}} \mathbf{\Psi}(\mathbf{x}_0)|.$$

In words, $\mathbf{\Psi}$ reshapes the density p_0 into p_1. The factor $|\det \partial_{\mathbf{x}} \mathbf{\Psi}|$ represents the local change in volume; since probability mass is conserved, the density compensates by its inverse.

As a simple case, if $\mathbf{\Psi}$ is linear with an invertible matrix $\mathbf{A}$ (i.e., $\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0$), then

$$p_1(\mathbf{x}_1) \;=\; p_0(\mathbf{A}^{-1}\mathbf{x}_1)\,\big|\det \mathbf{A}^{-1}\big|.$$

Intuition in 1D. Let us track a few particles x_1, x_2, x_3 visually. Suppose p_X lives on $[0,1]$ (width $\mathrm{d}x = 1$) and the map Ψ stretches this interval onto $[0,2]$ (width $\mathrm{d}y = 2$), carrying the particles along:



Because probability mass is conserved, the mass carried by a small interval is the same before and after the map:

$$p_X(x)\,\mathrm{d}x = p_Y(y)\,\mathrm{d}y \quad\Longrightarrow\quad p_Y(y) = \Big(\frac{\mathrm{d}y}{\mathrm{d}x}\Big)^{-1} p_X(x),$$

which is exactly Equation (B.1.1) in one dimension: the density is divided by the local stretching factor.

Schematically, we can read it as:

$$\begin{array}{lccc} \textbf{Sample:} & \mathbf{x}_0 & \xrightarrow{\mathbf{\Psi}} & \mathbf{x}_1 \\ \textbf{Density:} & p_{\mathbf{x}_0}(\mathbf{x}_0) & \xrightarrow{\mathbf{\Psi}} & p_{\mathbf{x}_1}(\mathbf{x}_1) \end{array}$$

Why is Equation (B.1.1) the Change-of-Variables Formula? This comes directly from the familiar rule in calculus, i.e., u-substitution.

Single-Variable Case. Let $y = \Psi(x)$ be smooth and invertible. Rewriting an integral over y in terms of x gives

$$\int g(y)\,\mathrm{d}y = \int g(\Psi(x)) \cdot |\Psi'(x)|\,\mathrm{d}x,$$

where $|\Psi'(x)|$ compensates for interval stretching or compression, ensuring area preservation.

But why the absolute value $|\Psi'(x)|$? Because we are integrating a probability density over a *fixed* volume: the bounds of integration cannot change (in particular, they cannot flip orientation), so only the magnitude of the stretching factor matters.

Multivariate Case. For $\mathbf{\Psi} : \mathbb{R}^D \to \mathbb{R}^D$ with $\mathbf{y} = \mathbf{\Psi}(\mathbf{x})$,

$$\int g(\mathbf{y})\, \mathrm{d}\mathbf{y} = \int g(\mathbf{\Psi}(\mathbf{x}))\, \left|\det(\partial_{\mathbf{x}} \mathbf{\Psi})\right|\, \mathrm{d}\mathbf{x},$$

so infinitesimal volumes transform as

$$\mathrm{d}\mathbf{y} = \left|\det(\partial_{\mathbf{x}} \mathbf{\Psi})\right|\, \mathrm{d}\mathbf{x}.$$

From this, the density formula in Equation (B.1.1) follows:

$$p_{\mathbf{y}}(\mathbf{y}) = \int_{\mathbb{R}^D} \delta(\mathbf{y} - \mathbf{\Psi}(\mathbf{x}))\, p_{\mathbf{x}}(\mathbf{x})\, \mathrm{d}\mathbf{x} = p_{\mathbf{x}}(\mathbf{\Psi}^{-1}(\mathbf{y}))\, \left|\det\left(\frac{\partial \mathbf{\Psi}^{-1}}{\partial \mathbf{y}}\right)\right|.$$

Proof. We use two facts.

(A) Infinitesimal volumes transform as $\mathrm{d}\mathbf{y} = \left|\det(\partial_{\mathbf{x}} \mathbf{\Psi})\right|\, \mathrm{d}\mathbf{x}$, as shown above.

(B) Lemma. Let $y = f(x)$ be smooth and invertible. Then

$$\frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(y) = \frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(f(x)) = \frac{\mathrm{d}x}{\mathrm{d}y} = \frac{1}{\mathrm{d}y/\,\mathrm{d}x} = \frac{1}{\frac{\mathrm{d}}{\mathrm{d}x} f(x)}.$$

In several dimensions the same statement reads $\partial_{\mathbf{y}} \mathbf{\Psi}^{-1}(\mathbf{y}) = \left(\partial_{\mathbf{x}} \mathbf{\Psi}(\mathbf{x})\right)^{-1}$ at $\mathbf{x} = \mathbf{\Psi}^{-1}(\mathbf{y})$, so that $\left|\det \partial_{\mathbf{y}} \mathbf{\Psi}^{-1}\right| = \left|\det \partial_{\mathbf{x}} \mathbf{\Psi}\right|^{-1}$.

Substituting $\mathbf{y}' = \mathbf{\Psi}(\mathbf{x})$, with (A) for the volume element and (B) for the inverse Jacobian determinant,

$$\begin{aligned} p_{\mathbf{y}}(\mathbf{y}) &= \int \delta(\mathbf{y} - \mathbf{\Psi}(\mathbf{x}))\, p_{\mathbf{x}}(\mathbf{x})\, \mathrm{d}\mathbf{x} \overset{(A)}{=} \int \delta(\mathbf{y} - \mathbf{y}')\, p_{\mathbf{x}}(\mathbf{\Psi}^{-1}(\mathbf{y}'))\, \left|\det \partial_{\mathbf{x}} \mathbf{\Psi}\right|^{-1}\, \mathrm{d}\mathbf{y}' \\ &\overset{(B)}{=} \int \delta(\mathbf{y} - \mathbf{y}')\, p_{\mathbf{x}}(\mathbf{\Psi}^{-1}(\mathbf{y}'))\, \left|\det \frac{\partial \mathbf{\Psi}^{-1}}{\partial \mathbf{y}'}\right|\, \mathrm{d}\mathbf{y}' \\ &= p_{\mathbf{x}}(\mathbf{\Psi}^{-1}(\mathbf{y}))\, \left|\det \frac{\partial \mathbf{\Psi}^{-1}}{\partial \mathbf{y}}\right|. \qquad \blacksquare \end{aligned}$$

Composing Multiple Bijections. We now apply several updates in sequence. Let $\mathbf{x}_k = \mathbf{\Psi}_k(\mathbf{x}_{k-1})$ for $k = 1, \ldots, L$; that is,

$$\mathbf{x}_0 \xrightarrow{\mathbf{\Psi}_1} \mathbf{x}_1 \xrightarrow{\mathbf{\Psi}_2} \cdots \xrightarrow{\mathbf{\Psi}_L} \mathbf{x}_L,$$

where each $\mathbf{\Psi}_k : \mathbb{R}^D \to \mathbb{R}^D$ is a smooth bijection. If the initial state follows density p_0 (i.e., $\mathbf{x}_0 \sim p_0$), then the sequence of updates induces densities $p_1, \ldots, p_L$ for $\mathbf{x}_1, \ldots, \mathbf{x}_L$.

Because probability mass is conserved at each step, the densities evolve according to

$$p_k(\mathbf{x}_k) \;=\; p_{k-1}(\mathbf{x}_{k-1})\, \left|\det \partial_{\mathbf{x}_{k-1}} \mathbf{\Psi}_k(\mathbf{x}_{k-1})\right|^{-1}, \quad k = 1, \ldots, L.$$

By recursion, the final density at $\mathbf{x}_L$ is

$$p_{\mathbf{x}_L}(\mathbf{x}_L) = p_{\mathbf{x}_0}(\mathbf{x}_0) \cdot \prod_{k=1}^{L} \left| \det \left(\frac{\partial \mathbf{\Psi}_k}{\partial \mathbf{x}_{k-1}} \right) \right|^{-1}. \tag{B.1.2}$$

Equivalently, in log-density form:

$$\log p_{\mathbf{x}_L}(\mathbf{x}_L) = \log p_{\mathbf{x}_0}(\mathbf{x}_0) - \sum_{k=1}^{L} \log \left| \det \left(\frac{\partial \mathbf{\Psi}_k}{\partial \mathbf{x}_{k-1}} \right) \right|.$$

This expression reflects how each transformation $\mathbf{\Psi}_k$ stretches or contracts volume, as captured by the Jacobian determinant. The accumulation of these local volume changes along the transformation path determines the final probability density under the composed map.

Equation (B.1.2) serves as the core principle underlying Normalizing Flows (see Section 5.1.2).

B.1.2 Continuous-Time Limit: Continuity Equation

We now pass from discrete updates to a continuous description. Suppose the particle motion is driven by a time-varying velocity field $\mathbf{f} : \mathbb{R}^D \times [0, T] \to \mathbb{R}^D$. Imagine evolving a particle $\mathbf{x}_0 \sim p_0$ through infinitely many small bijective updates. At each step t of length $\Delta t > 0$, the update is

$$\mathbf{x}_{t+\Delta t} = \mathbf{\Psi}(\mathbf{x}_t) := \mathbf{x}_t + \Delta t\, \mathbf{f}(\mathbf{x}_t, t).$$

As $\Delta t \to 0$, the composition of these updates converges to a continuous flow governed by a velocity field $\mathbf{f} : \mathbb{R}^D \times [0, T] \to \mathbb{R}^D$:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t), \qquad \mathbf{x}(0) = \mathbf{x}_0 \sim p_0. \tag{B.1.3}$$

Under suitable smoothness assumptions (see Chapter A), this ODE admits a unique solution for each initial condition, which defines a deterministic flow map $\mathbf{\Psi}_{0\to t} : \mathbb{R}^D \to \mathbb{R}^D$. In other words, $\mathbf{\Psi}_{0\to t}$ brings the initial state $\mathbf{x}_0$ to the solution of Equation (B.1.3) at time t:

$$\mathbf{\Psi}_{0\to t}(\mathbf{x}_0) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \tau)\ \mathrm{d}\tau.$$

As a result, the whole distribution also moves: the initial density p_0 is transported into the new density p_t, the law of $\mathbf{x}(t)$. Formally, this is written as a *pushforward*:

$$p_t = (\mathbf{\Psi}_{0\to t})_{\#} p_0.$$

When $\mathbf{\Psi}_{0\to t}$ is smooth and invertible, this reduces to the familiar change-of-variables rule:

$$p_t(\mathbf{x}) = p_0(\mathbf{\Psi}_{t\to 0}(\mathbf{x}))\, \left|\det \partial_{\mathbf{x}} \mathbf{\Psi}_{t\to 0}(\mathbf{x})\right| = \int \delta(\mathbf{x} - \mathbf{\Psi}_{0\to t}(\mathbf{x}_0))\, p_0(\mathbf{x}_0)\ \mathrm{d}\mathbf{x}_0.$$

Continuity Equation: How the Density Moves in Time. Rather than writing a separate formula for the density at each time, we can describe how it moves continuously using a differential equation in space $\mathbf{x}$ and time t. The idea is simple: probability mass is conserved, and the velocity field $\mathbf{f}$ only redistributes it in space. This gives the *continuity equation*:

$$\frac{\partial}{\partial t} p_t(\mathbf{x}) + \nabla \cdot \big(p_t(\mathbf{x})\, \mathbf{f}(\mathbf{x}, t)\big) = 0. \tag{B.1.4}$$

Here the divergence term $\nabla \cdot (p_t \mathbf{f})$ measures how the flow locally expands or compresses the density, ensuring total probability remains 1.

This partial differential equation (PDE) ensures that probability mass is conserved as the flow moves particles. In fact, it can be viewed as the continuous-time analogue of the change-of-variables formula.

Derivation of Continuity Equation via Change-of-Variables Formula. Conceptually, the continuity equation can also be obtained by taking the continuous-time limit of Equation (B.1.2). Here, however, we adopt a more direct derivation based on Equation (B.1.1).

Discretization and Change-of-Variable Formula. Consider

$$\mathbf{x}_{t+\Delta t} := \mathbf{\Psi}(\mathbf{x}_t) = \mathbf{x}_t + \Delta t\, \mathbf{f}(\mathbf{x}_t, t),$$

which is actually the forward Euler discretization of the ODE in Equation (B.1.3) over a small time interval $\Delta t > 0$. The Jacobian of the map $\mathbf{\Psi}$ with respect to $\mathbf{x}_t$ expands as

$$\frac{\partial \mathbf{\Psi}}{\partial \mathbf{x}_t} = \mathbf{I} + \Delta t \nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2),$$

so its determinant satisfies

$$\det\left(\frac{\partial \mathbf{\Psi}}{\partial \mathbf{x}_t}\right) = 1 + \Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2).$$

This uses the standard expansion $\det(\mathbf{I} + \Delta t\, \mathbf{A}) = 1 + \Delta t\, \mathrm{Tr}(\mathbf{A}) + \mathcal{O}(\Delta t^2)$ as $\Delta t \to 0$, along with $\nabla \cdot \mathbf{f} = \mathrm{Tr}(\nabla_{\mathbf{x}} \mathbf{f})$.

Applying the change-of-variables formula, the log-density evolves as

$$\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = \log p_t(\mathbf{x}_t) - \Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2).$$

Why? By Equation (B.1.1) written in the original coordinates,

$$p_t(\mathbf{x}_t) = p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) \det\Big(\frac{\partial \mathbf{\Psi}}{\partial \mathbf{x}_t}\Big),$$

where the absolute value may be dropped because the determinant equals $1 + \mathcal{O}(\Delta t) > 0$. Hence

$$\begin{aligned}\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) &= \log p_t(\mathbf{x}_t) - \log\left(1 + \Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2)\right) \\ &= \log p_t(\mathbf{x}_t) - \Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2),\end{aligned}$$

since $\log(1 + u) = u + \mathcal{O}(u^2)$ by Taylor expansion about $u = 0$, and here $u = \Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) \to 0$ as $\Delta t \to 0$. ■

That is,

$$\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t) = -\Delta t\, \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2). \tag{B.1.5}$$

Using Taylor Expansion. Now, we expand the left-hand side via multivariate Taylor expansion:

$$\begin{aligned}&\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t) \\ =&\Delta t\, \partial_t \log p_t(\mathbf{x}_t) + (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t)^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \mathcal{O}(\Delta t^2).\end{aligned}$$

(Recall that $\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t\|^2 = \mathcal{O}(\Delta t^2)$, so all second-order terms of the expansion are absorbed into $\mathcal{O}(\Delta t^2)$.) Substituting $\mathbf{x}_{t+\Delta t} - \mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\Delta t$ yields:

$$\begin{aligned}&\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t)\\ =&\Delta t\, \partial_t \log p_t(\mathbf{x}_t) + \Delta t\, \mathbf{f}(\mathbf{x}_t, t)^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \mathcal{O}(\Delta t^2).\end{aligned}$$

Matching terms with Equation (B.1.5) and letting $\Delta t \to 0$, we conclude that

$$\partial_t \log p_t(\mathbf{x}_t) = -\nabla_{\mathbf{x}_t} \cdot \mathbf{f}(\mathbf{x}_t, t) - \mathbf{f}(\mathbf{x}_t, t)^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

Exponentiating and using the product rule yields the continuity equation. *Explicitly:* since $\partial_t \log p_t = \partial_t p_t / p_t$ and $\nabla \log p_t = \nabla p_t / p_t$, multiplying the previous display through by $p_t(\mathbf{x}_t)$ gives

$$\partial_t p_t = -\, p_t\, \nabla \cdot \mathbf{f} - \mathbf{f}^\top \nabla p_t = -\, \nabla \cdot (p_t\, \mathbf{f}),$$

where the last step is the product rule for the divergence; this is exactly Equation (B.1.4). ∎

Lagrangian and Eulerian Views of Deterministic Transport. The same deterministic transport admits two complementary descriptions. The *Lagrangian* view follows individual particles through the flow map $\boldsymbol{\Psi}_{0\to t}$, whereas the *Eulerian* view describes the velocity $\mathbf{f}(\mathbf{x}, t)$ and density $p_t(\mathbf{x})$ at fixed spatial locations. Under suitable regularity, these descriptions are consistent: a prescribed velocity field generates a particle flow, whose pushforward density satisfies the continuity equation.

There is nevertheless an important asymmetry. Once the velocity field and initial density are fixed, the resulting flow and density evolution are determined. A prescribed density path alone, however, generally does not determine a unique velocity field or particle flow.

From Velocity to Flow and Density. So far, we have assumed that the velocity field $\mathbf{f}$ is given. The particle ODE

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t)$$

describes how each particle moves, while the density PDE

$$\partial_t p_t + \nabla \cdot (p_t \mathbf{f}_t) = 0$$

describes how the entire distribution of particles evolves. These two views are connected: moving particles according to the ODE automatically produces a density that satisfies the PDE. In this case, the particle flow $\boldsymbol{\Psi}_{0\to t}$ is uniquely determined: starting from $\mathbf{x}(0) \sim p_0$, each trajectory $\mathbf{x}(t)$ is fixed, and the resulting density p_t follows the continuity equation. Here, particle dynamics and density dynamics are fully consistent.

From Density Path to Velocity: Non-Uniqueness. If instead we begin only with the density path $t \mapsto p_t$ (e.g., Section 5.3.2 in flow matching), the velocity field is no longer uniquely determined. For example, if a vector field $\mathbf{w}_t$ satisfies

$$\nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x})\, \mathbf{w}_t(\mathbf{x})) = 0 \qquad \text{(no net flux w.r.t. } p_t\text{)},$$

then both $\mathbf{f}_t$ and $\mathbf{f}_t + \mathbf{w}_t$ give rise to the same density evolution. Thus a single density path may correspond to many different flows, and choosing one particular particle flow $\boldsymbol{\Psi}_{0\to t}$ amounts to picking a specific velocity field among these possibilities.

Not every given path p_t can actually arise from particles moving under some velocity field. The continuity equation (Equation (B.1.4)) provides the consistency check for whether a density path can be "generated by a flow". We say that p_t is *realizable* (or *generated by* $\mathbf{f}$) if there exists a velocity field $\mathbf{f}$ such that particles following

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), t)$$

produce exactly the densities p_t through the flow map $\boldsymbol{\Psi}_{0\to t}$. That is, realizability holds when p_t and $\mathbf{f}$ together satisfy Equation (B.1.4).

Intuitively, realizability means that the snapshots of p_t over time can be explained by particles moving under some velocity field, rather than being an arbitrary sequence of distributions.

When this condition holds, the density p_t is nothing more than the pushforward of the initial density p_0 along the flow map $\boldsymbol{\Psi}_{0\to t}$. In this case, the familiar change-of-variables formula applies:

$$p_t = (\boldsymbol{\Psi}_{0\to t})_{\#} p_0 = p_0\big(\boldsymbol{\Psi}_{t\to 0}(\mathbf{x})\big)\, \big|\det \partial_{\mathbf{x}} \boldsymbol{\Psi}_{t\to 0}(\mathbf{x})\big| = \int \delta\left(\mathbf{x} - \boldsymbol{\Psi}_{0\to t}(\mathbf{x}_0)\right) p_0(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0.$$

(Optional) Conditioning. If an additional conditioning variable $\mathbf{z} \sim \pi(\mathbf{z})$ is introduced, the same reasoning applies for each fixed $\mathbf{z}$:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}_t(\mathbf{x}(t)|\mathbf{z})$$

with pushforward $p_t(\cdot|\mathbf{z}) = (\boldsymbol{\Psi}_{0\to t}(\cdot; \mathbf{z}))_{\#} p_0$, and continuity equation

$$\partial_t p_t(\mathbf{x}|\mathbf{z}) + \nabla \cdot (p_t(\mathbf{x}|\mathbf{z})\, \mathbf{v}_t(\mathbf{x}|\mathbf{z})) = 0$$

The marginal density is then

$$p_t(\mathbf{x}) = \int p_t(\mathbf{x}|\mathbf{z}) \pi(\mathbf{z})\, \mathrm{d}\mathbf{z}.$$

B.1.3 Stochastic Processes: Fokker-Planck Equation

When noise is added, the dynamics follow the SDE as in Equation (A.2.3):

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t).$$

Then, the density $p_t(\mathbf{x})$ satisfies the Fokker-Planck equation:

$$\begin{aligned}\frac{\partial p_t(\mathbf{x})}{\partial t} &= -\nabla \cdot (\mathbf{f}(\mathbf{x}, t)\, p_t(\mathbf{x})) + \frac{1}{2} g^2(t)\, \Delta p_t(\mathbf{x}) \\ &= -\nabla \cdot \left(\left(\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right) p_t(\mathbf{x}) \right).\end{aligned}$$

$\Delta p_t = \nabla \cdot \nabla_{\mathbf{x}} p_t$ is the Laplacian operator. In the first equality, the first term describes transport of probability mass by the deterministic drift $\mathbf{f}$, while the second term models the spreading (diffusion) of the density due to stochastic noise with variance proportional to $\frac{1}{2} g^2(t)$. The second equality rewrites the same density evolution as a continuity equation with velocity

$$\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Therefore, the deterministic ODE driven by this velocity therefore evolves through the same one-time marginal densities p_t as the SDE. This is the probability-flow ODE used throughout the book.

Remark.

> The key point is that the SDE and PF-ODE agree on the *density path*, but generally not on how samples at different times are coupled. At any fixed time t, both produce the same distribution p_t; equivalently, their particle "snapshots" have the same histogram.
>
> Across time, however, their sample paths are different. For the PF-ODE, once the state $\mathbf{x}_s$ is fixed, the later state is determined by $\boldsymbol{\Psi}_{s \to t}(\mathbf{x}_s)$. For the SDE, fresh Brownian noise is injected after time s, so the later state remains random even after conditioning on $\mathbf{x}_s$.
>
> Consequently, the two processes may have different transition distributions, temporal correlations, and pathwise events, even though their marginal density at every individual time is identical. Thus, the PF-ODE reproduces the SDE's *marginal density evolution*, rather than its full stochastic path law. See Figure 4.5 for illustration.

The derivation of the Fokker-Planck equation is more involved; we refer the reader to Section C.1.4.

B.2 Intuition of the Continuity Equation

In this section, we give a physical interpretation of the continuity equation, highlighting its role as a conservation law for probability density in a dynamical system.

B.2.1 Derivation of the Continuity Equation from Conservation Laws

We write the conservation principle in a coordinate-free form over an arbitrary control volume.

The continuity equation formalizes the conservation of a physical quantity, such as mass or charge, in a dynamical system. Let $p(\mathbf{x}, t)$ denote the density of the conserved quantity at position $\mathbf{x} \in \mathbb{R}^D$ and time $t \in [0, T]$, and let $\mathbf{v}(\mathbf{x}, t)$ denote the velocity field. When the quantity is transported by particles moving with velocity $\mathbf{v}(\mathbf{x}, t)$, the flux is

$$\mathbf{j}(\mathbf{x}, t) = p(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t).$$

Step 1: Rate of Change within a Control Volume. Consider an arbitrary control volume $V \subset \mathbb{R}^D$ with boundary ∂V. The total amount of the conserved quantity in V is

$$\int_V p(\mathbf{x}, t)\, \mathrm{d}V,$$

whose time derivative gives the rate of accumulation:

$$\frac{\partial}{\partial t} \int_V p(\mathbf{x}, t)\, \mathrm{d}V.$$

Step 2: Net Flux Through the Boundary. The quantity exits V through ∂V with outward normal vector $\mathbf{n}$. The net outward flux is

$$\int_{\partial V} p(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t) \cdot \mathbf{n}\, \mathrm{d}S.$$

Step 3: Conservation Principle. Conservation implies that the rate of accumulation within V equals the negative of the net outward flux:

$$\frac{\partial}{\partial t} \int_V p\, \mathrm{d}V + \int_{\partial V} p\mathbf{v} \cdot \mathbf{n}\, \mathrm{d}S = 0.$$

Step 4: Divergence Theorem. Applying the divergence theorem (see Section E.1) to convert the surface integral to a volume integral:

$$\int_{\partial V} p\mathbf{v} \cdot \mathbf{n}\, \mathrm{d}S = \int_V \nabla \cdot (p\mathbf{v})\, \mathrm{d}V.$$

Hence,

$$\frac{\partial}{\partial t}\int_V p\,\mathrm{d}V + \int_V \nabla\cdot(p\mathbf{v})\,\mathrm{d}V = 0.$$

Equivalently,

$$\int_V \left(\frac{\partial p}{\partial t} + \nabla\cdot(p\mathbf{v})\right)\mathrm{d}V = 0.$$

Since the control volume V is arbitrary, the integrand must vanish pointwise. This yields the continuity equation:

$$\frac{\partial p}{\partial t} + \nabla\cdot(p\mathbf{v}) = 0.$$

Using $\mathbf{j} = p\mathbf{v}$, this is the same as

$$\frac{\partial p}{\partial t} + \nabla\cdot\mathbf{j} = 0.$$

Particle-Level Intuition. Expanding the divergence term gives

$$\frac{\partial p}{\partial t} + \mathbf{v}\cdot\nabla p + p\,\nabla\cdot\mathbf{v} = 0.$$

The first two terms describe the total change of the density along a moving particle:

$$\frac{\mathrm{d}}{\mathrm{d}t}p(\mathbf{x}_t,t) = \frac{\partial p}{\partial t}(\mathbf{x}_t,t) + \mathbf{v}(\mathbf{x}_t,t)\cdot\nabla p(\mathbf{x}_t,t).$$

Therefore,

$$\frac{\mathrm{d}}{\mathrm{d}t}p(\mathbf{x}_t,t) = -p(\mathbf{x}_t,t)\nabla\cdot\mathbf{v}(\mathbf{x}_t,t).$$

This provides another useful interpretation: density decreases when nearby particles spread apart, and density increases when nearby particles contract. Thus, the velocity field tells us how individual particles move, while the continuity equation tells us how the entire density evolves.

B.3 (Optional) Wasserstein Gradient Flows as Distribution-Level Training

We now step back from diffusion-specific dynamics and take a broader distribution-level view of deep generative modeling. We have discussed that the continuity equation describes how a probability distribution moves when its particles move. The same language can also be used to understand the training of deep generative models: a training algorithm updates the model parameters, the updated generator moves generated particles, and the movement of these particles changes the induced model distribution $p_{\boldsymbol{\phi}}$.

From this viewpoint, different generative modeling methods can be organized according to how the model distribution $p_{\boldsymbol{\phi}}$ is driven toward the data distribution p_{data}. Wasserstein gradient flow gives one particularly clean way to describe this movement: it treats training as an idealized flow of probability mass in distribution space, which is then approximated by finite-dimensional neural network updates.

A neural generator $\mathbf{G}_{\boldsymbol{\phi}}$ defines both a function and a probability distribution. If

$$\mathbf{z} \sim p_{\text{prior}}, \qquad \mathbf{x} = \mathbf{G}_{\boldsymbol{\phi}}(\mathbf{z}),$$

then the generated distribution is the pushforward

$$p_{\boldsymbol{\phi}} := (\mathbf{G}_{\boldsymbol{\phi}})_{\#} p_{\text{prior}}.$$

From the generative modeling viewpoint, the object we ultimately care about is how $p_{\boldsymbol{\phi}}$ approaches the data distribution p_{data}.

This motivates a distribution-level question:

Question B.3.1

Can we first describe the ideal way to move the model distribution $p_{\boldsymbol{\phi}}$ toward p_{data}, and then update the neural network parameters so that the generator approximately realizes this movement?

Throughout this optional section, τ denotes *training time*: an idealized continuous-time version of the discrete training iteration index. It should not be confused with diffusion/noising time. We write $\boldsymbol{\phi}_\tau$ for the parameter at training time τ, and $p_{\boldsymbol{\phi}_\tau}$ for the corresponding model distribution.

Moving Model Distributions by Moving Particles. We first describe an ideal distribution-level evolution, before accounting for the finite-dimensional parameterization of the neural generator. Suppose particles from the current model distribution move according to a training-time velocity field $\mathbf{w}_\tau$:

$$\frac{\mathrm{d}\mathbf{x}_\tau}{\mathrm{d}\tau} = \mathbf{w}_\tau(\mathbf{x}_\tau), \qquad \mathbf{x}_\tau \sim p_{\boldsymbol{\phi}_\tau}.$$

Then the model distribution evolves according to the continuity equation

$$\partial_\tau p_{\boldsymbol{\phi}_\tau}(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot \left(p_{\boldsymbol{\phi}_\tau}(\mathbf{x}) \mathbf{w}_\tau(\mathbf{x})\right) = 0.$$

Thus, $\mathbf{w}_\tau$ describes how each generated particle moves, while the continuity equation describes how the entire generated distribution moves[1].

Choosing the Velocity: Gradient Descent in Distribution Space. Now we need to choose a velocity field $\mathbf{w}_\tau$ that moves p_{ϕ_τ} toward p_{data}. Let

$$\mathcal{D}(p_{\phi_\tau}, p_{\text{data}})$$

be a discrepancy between the current model distribution and the data distribution. *Wasserstein gradient flow* (Jordan *et al.*, 1998; Ambrosio *et al.*, 2005) chooses the particle velocity that decreases this discrepancy most directly under the geometry of moving probability mass.

To choose the velocity field, we first ask how the discrepancy changes when the current distribution is infinitesimally perturbed. Think of $\mathcal{D}(p, p_{\text{data}})$ as a global energy of the distribution p. Its first variation gives a pointwise energy potential: it tells us which regions of space are costly for the current distribution and which regions are favorable.

Concretely, for a small perturbation $p+\epsilon h$, where $\epsilon > 0$ is small and $\int h(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0$, the first variation is defined by

$$\mathcal{D}(p + \epsilon h, p_{\text{data}}) = \mathcal{D}(p, p_{\text{data}}) + \epsilon \int \frac{\delta \mathcal{D}}{\delta p}(p, p_{\text{data}})(\mathbf{x})\, h(\mathbf{x})\,\mathrm{d}\mathbf{x} + \mathrm{o}(\epsilon).$$

Here p is a dummy distribution variable. After taking the variation, we evaluate it at the current model distribution $p = p_{\phi_\tau}$.

We define the resulting pointwise energy potential by

$$E_\tau(\mathbf{x}) := \frac{\delta \mathcal{D}}{\delta p}(p_{\phi_\tau}, p_{\text{data}})(\mathbf{x}).$$

Intuitively, if $E_\tau(\mathbf{x})$ is large, then placing more probability mass near $\mathbf{x}$ would increase the discrepancy. If $E_\tau(\mathbf{x})$ is small, then moving mass toward $\mathbf{x}$ is favorable. Therefore, the Wasserstein gradient-flow velocity moves particles downhill in this potential:

$$\mathbf{w}_\tau(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\tau(\mathbf{x}).$$

With this choice,

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\mathcal{D}(p_{\phi_\tau}, p_{\text{data}}) = -\int p_{\phi_\tau}(\mathbf{x})\, \|\nabla_{\mathbf{x}} E_\tau(\mathbf{x})\|_2^2\,\mathrm{d}\mathbf{x} \le 0.$$

Therefore, Wasserstein gradient flow is gradient descent in the 2-Wasserstein geometry of probability distributions[2]. The global discrepancy $\mathcal{D}$ defines the energy

[1] This velocity should not be confused with the diffusion-time PF-ODE velocity $\mathbf{v}^*(\mathbf{x}, t)$. The variable t or s describes noise/denoising time, whereas τ describes training-time evolution of the model distribution.

[2] We remark that this ideal distribution-space dynamics should not be confused with ordinary gradient descent on the neural parameters ϕ: the latter is constrained by the finite-dimensional generator family and generally realizes only an approximation to the desired particle motion.

of the whole distribution, the first variation E_τ defines the local energy landscape seen by particles, and $-\nabla_{\mathbf{x}} E_\tau$ gives the ideal particle velocity for decreasing the discrepancy.

Realizing the Distributional Velocity with a Neural Network. The velocity above is a distribution-level object: it says how generated particles should move. A neural generator, however, cannot move every particle independently. It can only move particles through a shared parameter update of $\boldsymbol{\phi}$.

At training time τ, the current generator produces particles

$$\mathbf{x}_i = \mathbf{G}_{\boldsymbol{\phi}_\tau}(\mathbf{z}_i), \qquad \mathbf{z}_i \sim p_{\text{prior}}, \qquad i = 1, \dots, N.$$

A small Wasserstein step moves these particles to

$$\widetilde{\mathbf{x}}_i = \mathbf{x}_i + \eta \mathbf{w}_\tau(\mathbf{x}_i),$$

where $\eta > 0$ is a small step size. The neural network is then updated so that its new outputs match these moved particles:

$$\boldsymbol{\phi}_{\tau+\eta} \approx \arg\min_{\boldsymbol{\phi}} \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{G}_{\boldsymbol{\phi}}(\mathbf{z}_i) - \operatorname{sg}\left(\mathbf{x}_i + \eta \mathbf{w}_\tau(\mathbf{x}_i)\right) \right\|_2^2 .$$

Here $\operatorname{sg}(\cdot)$ denotes stop-gradient, so the moved particles are treated as fixed targets.

For a small parameter update $\Delta\boldsymbol{\phi}$, we have the linearization

$$\mathbf{G}_{\boldsymbol{\phi}_\tau + \Delta\boldsymbol{\phi}}(\mathbf{z}_i) \approx \mathbf{G}_{\boldsymbol{\phi}_\tau}(\mathbf{z}_i) + \left(\partial_{\boldsymbol{\phi}} \mathbf{G}_{\boldsymbol{\phi}_\tau}(\mathbf{z}_i)\right) \Delta\boldsymbol{\phi}.$$

Therefore, the parameter update approximately solves

$$\min_{\Delta\boldsymbol{\phi}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left(\partial_{\boldsymbol{\phi}} \mathbf{G}_{\boldsymbol{\phi}_\tau}(\mathbf{z}_i)\right) \Delta\boldsymbol{\phi} - \eta \mathbf{w}_\tau(\mathbf{x}_i) \right\|_2^2 .$$

In words, Wasserstein gradient flow specifies the desired distribution-level particle motion, while neural network training projects this motion onto the set of movements that can be realized by changing the finite-dimensional parameter $\boldsymbol{\phi}$.

Forward-KL Training: Data-Side Likelihood Learning. A classical distribution-level objective is the data-side, or forward, KL divergence:

$$\mathcal{D}_{\text{FKL}}(p_{\boldsymbol{\phi}}, p_{\text{data}}) := \mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\boldsymbol{\phi}}) = \int p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\boldsymbol{\phi}}(\mathbf{x})} \, \mathrm{d}\mathbf{x}.$$

Since p_{data} is fixed, minimizing this objective is equivalent to maximum likelihood (see Equation (1.1.2)):

$$\min_{\boldsymbol{\phi}} \mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\boldsymbol{\phi}}) \quad \Longleftrightarrow \quad \max_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log p_{\boldsymbol{\phi}}(\mathbf{x}) \right].$$

This is the distribution-level view behind many likelihood-based models. Autoregressive models and normalizing flows optimize exact likelihoods; likelihood-based VAEs optimize variational lower bounds; energy-based models optimize likelihood through positive and negative phases; and diffusion models can be related to likelihood or score-matching surrogates through their variational and denoising objectives.

From the Wasserstein gradient-flow viewpoint, the forward KL has first variation

$$\frac{\delta}{\delta p}\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}}\|p) = -\frac{p_{\mathrm{data}}}{p}.$$

Evaluating at $p = p_{\phi_\tau}$ gives the velocity

$$\mathbf{w}_\tau^{\mathrm{FKL}}(\mathbf{x}) = \nabla_{\mathbf{x}}\left(\frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\phi_\tau}(\mathbf{x})}\right).$$

This expression involves the density ratio $p_{\mathrm{data}}/p_{\phi_\tau}$, which is usually difficult to estimate for implicit generators. In practice, forward-KL or maximum-likelihood training is most direct when likelihoods, variational bounds, or suitable approximations to the likelihood gradient are available. Score matching provides a different route to distribution learning by matching score functions rather than directly realizing the forward-KL Wasserstein flow.

The main strength of forward-KL training is that it is data-covering and statistically well grounded when a likelihood or a variational surrogate is available. Its limitation is that it often requires tractable densities, variational bounds, score-matching surrogates, or expensive negative sampling, which makes it less directly compatible with arbitrary implicit generators.

Reverse-KL Training: Model-Side Score-Mismatch Dynamics. The model-side, or reverse, KL divergence is

$$\mathcal{D}_{\mathrm{RKL}}(p_{\boldsymbol{\phi}}, p_{\mathrm{data}}) := \mathcal{D}_{\mathrm{KL}}(p_{\boldsymbol{\phi}}\|p_{\mathrm{data}}) = \int p_{\boldsymbol{\phi}}(\mathbf{x}) \log \frac{p_{\boldsymbol{\phi}}(\mathbf{x})}{p_{\mathrm{data}}(\mathbf{x})}\,\mathrm{d}\mathbf{x}.$$

Its first variation is

$$\frac{\delta}{\delta p}\mathcal{D}_{\mathrm{KL}}(p\|p_{\mathrm{data}}) = \log p - \log p_{\mathrm{data}} + 1.$$

Evaluating at $p = p_{\phi_\tau}$ gives the Wasserstein velocity

$$\mathbf{w}_\tau^{\mathrm{RKL}}(\mathbf{x}) := \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\phi_\tau}(\mathbf{x}).$$

Thus, reverse-KL training induces a score-mismatch force:

$$\text{target score} - \text{model score}.$$

The target score attracts generated particles toward high-density regions of the data distribution, while the model score repels particles from regions where the current model distribution is already too concentrated.

In diffusion distillation and distribution matching (see Section 10.2), this idea is usually applied at a noise level s. Let p_s denote the teacher or noised-data distribution, and let $p_{\phi_\tau,s}$ denote the noised distribution of current generator samples. The reverse-KL objective

$$\mathbb{E}_s \left[w(s) \mathcal{D}_{\mathrm{KL}}(p_{\phi_\tau,s} \| p_s) \right]$$

induces the score-mismatch velocity

$$\mathbf{w}^{\mathrm{RKL}}_{\tau,s}(\mathbf{x}) := \nabla_{\mathbf{x}} \log p_s(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\phi_\tau,s}(\mathbf{x}).$$

At the parameter level, for generated noisy samples $\hat{\mathbf{x}}_s \sim p_{\phi_\tau,s}$, one obtains gradients of the form

$$\nabla_\phi \mathcal{D}_{\mathrm{KL}}(p_{\phi,s} \| p_s)\big|_{\phi=\phi_\tau} = \mathbb{E}_{\hat{\mathbf{x}}_s \sim p_{\phi_\tau,s}} \left[\left(\nabla_{\mathbf{x}} \log p_{\phi_\tau,s}(\hat{\mathbf{x}}_s) - \nabla_{\mathbf{x}} \log p_s(\hat{\mathbf{x}}_s) \right)^\top \partial_\phi \hat{\mathbf{x}}_s \right].$$

Thus, the score difference

$$\nabla_{\mathbf{x}} \log p_s(\hat{\mathbf{x}}_s) - \nabla_{\mathbf{x}} \log p_{\phi_\tau,s}(\hat{\mathbf{x}}_s)$$

provides the distribution-level descent direction that drives the parameter gradient. The actual movement of individual generated samples is mediated by the shared neural parameterization, as discussed above. This score-difference gradient is the basic distribution-level mechanism behind reverse-KL-based methods such as DMD and VSD: estimate the target and model scores on generated samples and use their difference to update the generator.

The main strength of reverse-KL training is that it is naturally compatible with implicit generators once the relevant scores can be estimated. It is also mode-seeking, which can improve visual fidelity in one-step or few-step generation. Its limitation is that it can drop modes, depends strongly on the quality of score estimates, and may require an additional model-score network or a strong teacher.

The same attraction-repulsion interpretation also appears in drifting-style particle updates (Weber, 2023; Deng *et al.*, 2026). In the terminology of Lai *et al.* (2026), DMD/VSD/SiD provide parametric score-based instantiations of this score-mismatch dynamics, while drifting-style approaches (Weber, 2023; Deng *et al.*, 2026) provide a nonparametric kernel-based instantiation.

Closing Remarks. In summary, the continuity equation provides the common language behind both diffusion dynamics and distribution-level training. In diffusion models, it describes how the intermediate distributions p_t evolve along a prescribed noise or probability-flow dynamics. In Wasserstein gradient flow, the same equation

describes how the model distribution p_{ϕ_τ} evolves during training as generated particles move toward the data distribution p_{data}.

This viewpoint therefore broadens the role of the continuity equation beyond diffusion models. It lets us interpret generative modeling as the design of distributional dynamics: different methods choose different discrepancies, different particle velocities, and different parameter updates to drive p_ϕ toward p_{data}. From this perspective, diffusion models, likelihood-based models, distribution-matching methods, and score-mismatch-based approaches can be viewed under a shared distribution-level taxonomy.

C

Behind the Scenes of Diffusion Models: Itô's Calculus and Girsanov's Theorem

(Score-based) Diffusion models are built on SDEs: a drift that pushes states and a Brownian term that jitters them. Unlike ODE paths, Brownian paths are nowhere differentiable, so the ordinary chain rule fails. In this section, we introduce two fundamental tools that make the math precise:

- **Itô's Formula** is the correct chain rule for stochastic trajectories. It tells us how a function $\mathbf{h}(\mathbf{x}_t, t)$ evolves when $\mathbf{x}_t$ follows an SDE. It enables derivations of the Fokker–Planck equation, moment dynamics, the Itô product rule, and the identities used in score-based training.
- **Girsanov's Theorem** is a change-of-measure result on *path probabilities*. It quantifies how likelihoods change when the noise is fixed but the drift is altered. This links likelihood-weighted score matching to path-space KL divergence and, through a variational bound, to data likelihood.

With these tools, the standard diffusion model derivations (Fokker–Planck, reverse time SDE, training objectives, and likelihood relations) follow cleanly and without hand waving.

After this chapter, you will be able to...

- understand why ordinary calculus is not enough for SDEs, and how Itô's formula gives the correct chain rule for stochastic trajectories;
- see how Itô's formula works behind the scenes in important diffusion-model derivations, including the Fokker–Planck equation and linear SDEs;
- understand Girsanov's theorem as a way to compare the probabilities of stochastic paths when their drifts are different but their diffusion is the same;

- distinguish matching distributions at each time from matching the full stochastic path, and see why likelihood-weighted score matching is related to path-space KL divergence and data likelihood.

Reader route. This appendix develops some of the mathematical tools that operate behind the scenes of diffusion models. Itô calculus makes stochastic differentiation precise, while Girsanov's theorem explains how changing the drift changes the probability of entire stochastic paths. These tools underlie several standard diffusion-model derivations that are often used without being shown explicitly.

C.1 Itô's Formula: The Chain Rule for Random Processes

Standard calculus does not directly apply to stochastic processes because Wiener processes are not differentiable in the classical sense. Instead, we use Itô's calculus, which provides rules for working with stochastic integrals.

C.1.1 Motivation: Why Do We Need a Special Chain Rule?

Consider a deterministic time-varying function $\mathbf{y}_t$ that evolves smoothly with time t (e.g., an ODE). If we have a function $\mathbf{h}(\mathbf{y}_t, t)$, the usual chain rule tells us:

$$\frac{\mathrm{d}\mathbf{h}}{\mathrm{d}t} = \frac{\partial \mathbf{h}}{\partial t} + \nabla_{\mathbf{y}}\mathbf{h}\frac{\mathrm{d}\mathbf{y}_t}{\mathrm{d}t}.$$

Here, $\nabla_{\mathbf{y}}\mathbf{h}$ is the Jacobian of $\mathbf{h}$. This works perfectly for deterministic paths $\mathbf{y}_t$.

Question C.1.1

But what happens if $\mathbf{x}_t$ is a stochastic process, say, driven by an SDE

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t$$

as in Equation (A.2.3)? What SDE does the process $\mathbf{h}(\mathbf{x}_t, t)$ satisfy?

Why the Ordinary Chain Rule Fails? Naïvely applying the classical chain rule yields

$$\mathrm{d}\mathbf{h} = \frac{\partial \mathbf{h}}{\partial t}\,\mathrm{d}t + \nabla_{\mathbf{y}}\mathbf{h} \cdot \mathrm{d}\mathbf{x}_t.$$

However, Brownian increments scale as $\mathcal{O}(\sqrt{\mathrm{d}t})$, while their quadratic variation is encoded in Itô calculus by

$$(\mathrm{d}\mathbf{w}_t)^2 = \mathrm{d}t.$$

Thus, second-order terms in $\mathrm{d}\mathbf{w}_t$ *do not* vanish in stochastic calculus, unlike classical calculus where $(\mathrm{d}t)^2$ terms are negligible.

Example: Simple Example–$h(x_t) = x_t^2$

To see the intuition, let us consider the simple real-valued function $h(x_t) = x_t^2 \in \mathbb{R}$ where the random variable $x_t \in \mathbb{R}$ satisfies

$$\mathrm{d}x_t = \sigma\,\mathrm{d}w_t, \quad x_0 = 0,$$

with a constant $\sigma > 0$, so that $x_t = \sigma w_t$ (integrating both sides w.r.t. time and using $x_0 = 0$). If we try the classical chain rule,

$$\mathrm{d}h = 2x_t\,\mathrm{d}x_t = 2x_t\sigma\,\mathrm{d}w_t.$$

If this were true, the expectation of $h(x_t)$ would be constant in time because

$$\mathbb{E}[\mathrm{d}h] = 2\sigma\mathbb{E}[x_t\,\mathrm{d}w_t] = 2\sigma\mathbb{E}[x_t]\underbrace{\mathbb{E}[\mathrm{d}w_t]}_{=0} = 0.$$

But we know from classical Brownian motion properties (see Equation (A.2.4)) that

$$\mathbb{E}[x_t^2] = \sigma^2 t,$$

which grows linearly in time. So the ordinary chain rule misses an important term. Indeed, $\mathbb{E}[x_t^2] = \mathrm{Var}[x_t] + \mathbb{E}[x_t]^2 = \mathrm{Var}[\sigma w_t] + 0 = \sigma^2 t$.

C.1.2 Deriving 1D Itô's Formula from Taylor Expansion

Note. What follows is not a formal proof; it is the intuition behind Itô's formula, obtained by keeping track of the orders of magnitude in a Taylor expansion.

Deterministic Chain Rule via Taylor Expansion. To understand why the classical chain rule fails for stochastic processes defined by SDEs, we first revisit it in the deterministic setting using Taylor expansion. We consider the scalar case: $y_t \in \mathbb{R}$ and $h(\cdot,\cdot) \in \mathbb{R}$. Formally treating $\mathrm{d}y_t = y_{t+\mathrm{d}t} - y_t$, with $\mathrm{d}t \approx 0$, we expand:

$$\begin{aligned} &h(y_{t+\mathrm{d}t}, t+\mathrm{d}t) - h(y_t, t) \\ &= \frac{\partial h}{\partial t}\,\mathrm{d}t + \frac{\partial h}{\partial y}\,\mathrm{d}y_t + \frac{1}{2}\left(\frac{\partial^2 h}{\partial y^2}(\mathrm{d}y_t)^2 + 2\frac{\partial^2 h}{\partial t \partial y}\,\mathrm{d}t\,\mathrm{d}y_t + \frac{\partial^2 h}{\partial t^2}(\mathrm{d}t)^2\right) + \mathcal{O}(\mathrm{d}t^3), \end{aligned}$$

Here, $\mathrm{d}t\,\mathrm{d}y_t = \left(\frac{\mathrm{d}y_t}{\mathrm{d}t}\right)(\mathrm{d}t)^2 = \mathcal{O}(\mathrm{d}t^2)$, and similarly $(\mathrm{d}t)^2 = \mathcal{O}(\mathrm{d}t^2)$. Likewise, $(\mathrm{d}y_t)^2 = \left(\frac{\mathrm{d}y_t}{\mathrm{d}t}\right)^2(\mathrm{d}t)^2 = \mathcal{O}(\mathrm{d}t^2)$. Therefore, all the gray parts are ignorable, and the full differential is:

$$\mathrm{d}h = \frac{\partial h}{\partial t}\,\mathrm{d}t + \frac{\partial h}{\partial y}\,\mathrm{d}y_t + \mathcal{O}(\mathrm{d}t^2).$$

Itô's Formula via Stochastic Taylor Expansion. Now consider a stochastic process $x_t \in \mathbb{R}$ governed by the SDE:

$$\mathrm{d}x_t = f(x_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}w_t,$$

where w_t is standard Brownian motion. We aim to compute the differential of a scalar-valued function $h(x_t, t)$.

Using the stochastic Taylor expansion (Kloeden *et al.*, 1992), which retains

second-order terms in $\mathrm{d}x_t$, we have:

$$\begin{aligned} & h(x_{t+\mathrm{d}t}, t+\mathrm{d}t) - h(x_t, t) \\ &= \frac{\partial h}{\partial t}\,\mathrm{d}t + \frac{\partial h}{\partial x}\,\mathrm{d}x_t + \frac{1}{2}\left(\underbrace{\frac{\partial^2 h}{\partial x^2}(\mathrm{d}x_t)^2}_{\text{extra term}} + 2\frac{\partial^2 h}{\partial t \partial x}\,\mathrm{d}t\,\mathrm{d}x_t + \frac{\partial^2 h}{\partial t^2}(\mathrm{d}t)^2 \right) + \cdots \end{aligned} \tag{C.1.1}$$

Negligible Cross Terms. By the scaling property of Brownian motion (Equation (A.2.4)),

$$\mathrm{d}w_t = \mathcal{O}(\sqrt{\mathrm{d}t}) \quad \Rightarrow \quad \mathrm{d}t \cdot \mathrm{d}w_t = \mathcal{O}((\mathrm{d}t)^{3/2}).$$

Therefore,

$$\mathrm{d}t \cdot \mathrm{d}x_t = \mathrm{d}t\,(f\,\mathrm{d}t + g\,\mathrm{d}w_t) = f(\mathrm{d}t)^2 + g \cdot \mathrm{d}t \cdot \mathrm{d}w_t = \mathcal{O}((\mathrm{d}t)^{3/2}).$$

So, the gray terms are negligible: $\mathcal{O}((\mathrm{d}t)^{3/2})$ or smaller.

Second-Order Term $(\mathrm{d}x_t)^2$. Expanding using the SDE:

$$\begin{aligned} (\mathrm{d}x_t)^2 &= (f\,\mathrm{d}t + g\,\mathrm{d}w_t)^2 \\ &= f^2(\mathrm{d}t)^2 + 2fg\,\mathrm{d}t\,\mathrm{d}w_t + g^2(\mathrm{d}w_t)^2 \\ &= g^2(t)\,\mathrm{d}t, \end{aligned}$$

where the last equality uses the Itô differential rules $(\mathrm{d}t)^2 = 0$, $\mathrm{d}t\,\mathrm{d}w_t = 0$, and $(\mathrm{d}w_t)^2 = \mathrm{d}t$, with the last identity understood in the quadratic-variation sense.

Combining terms, we obtain the differential:

$$\mathrm{d}h(x_t, t) = \frac{\partial h}{\partial t}\,\mathrm{d}t + \frac{\partial h}{\partial x}\,\mathrm{d}x_t + \frac{1}{2}\frac{\partial^2 h}{\partial x^2} g^2(t)\,\mathrm{d}t.$$

Substituting $\mathrm{d}x_t = f(x_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}w_t$ yields:

$$\mathrm{d}h(x_t, t) = \left(\frac{\partial h}{\partial t} + f\frac{\partial h}{\partial x} + \frac{1}{2}g^2\frac{\partial^2 h}{\partial x^2} \right)\mathrm{d}t + g\frac{\partial h}{\partial x}\,\mathrm{d}w_t.$$

This is the 1D version of *Itô's formula*.

Example: Simple Example–$h(x_t) = x_t^2$

> We revisit the simple example: $h(x_t) = x_t^2$, where the stochastic process $x_t \in \mathbb{R}$ satisfies
>
> $$\mathrm{d}x_t = \sigma\,\mathrm{d}w_t,$$
>
> with a constant $\sigma > 0$. Applying Itô's formula correctly to $h(x_t) = x_t^2$, we obtain:
>
> $$\mathrm{d}h(x_t) = \mathrm{d}(x_t^2) = 2x_t\,\mathrm{d}x_t + \sigma^2\,\mathrm{d}t.$$

Substituting $\mathrm{d}x_t = \sigma\,\mathrm{d}w_t$, this becomes:

$$\mathrm{d}(x_t^2) = 2x_t\sigma\,\mathrm{d}w_t + \sigma^2\,\mathrm{d}t.$$

C.1.3 Itô's Formula: The Chain Rule for SDEs

We summarize the one-dimensional Itô's formula derived above. Using similar arguments, the result extends naturally to the multi-dimensional setting. While we omit the detailed derivation, we state the general formula for completeness.

Finally, we illustrate an application of Itô's formula by deriving the *Itô product rule*, which enables computation of $\mathrm{d}(\mathbf{x}_t^\top \mathbf{y}_t)$ for stochastic processes $\mathbf{x}_t$ and $\mathbf{y}_t$.

1D Itô's Formula. Let $x_t \in \mathbb{R}$ be a stochastic process satisfying the SDE:

$$\mathrm{d}x_t = f(x_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}w_t.$$

For a scalar function $h\colon \mathbb{R} \times [0,T] \to \mathbb{R}$, the process $h(x_t, t)$ satisfies:

$$\mathrm{d}h(x_t,t) = \left(\frac{\partial h}{\partial t} + f\frac{\partial h}{\partial x} + \frac{1}{2}g^2\frac{\partial^2 h}{\partial x^2}\right)\mathrm{d}t + g\frac{\partial h}{\partial x}\,\mathrm{d}w_t.$$

Multidimensional Itô's Formula with Scalar Output. Let $\mathbf{x}_t \in \mathbb{R}^D$ satisfy the SDE:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t,$$

where $\mathbf{f}\colon \mathbb{R}^D \times [0,T] \to \mathbb{R}^D$, $g\colon [0,T] \to \mathbb{R}$, and $\mathbf{w}_t \in \mathbb{R}^D$ is a D-dimensional Brownian motion. Let $h\colon \mathbb{R}^D \times [0,T] \to \mathbb{R}$ be a scalar-valued function. Then $h(\mathbf{x}_t, t)$ satisfies:

$$\mathrm{d}h(\mathbf{x}_t,t) = \left(\frac{\partial h}{\partial t} + \nabla_{\mathbf{x}} h^\top \mathbf{f} + \frac{1}{2}g^2(t)\,\mathrm{Tr}\left(\nabla_{\mathbf{x}}^2 h\right)\right)\mathrm{d}t + g(t)\nabla_{\mathbf{x}} h^\top\,\mathrm{d}\mathbf{w}_t, \quad \text{(C.1.2)}$$

where $\nabla_{\mathbf{x}} h \in \mathbb{R}^D$ is the gradient and $\nabla_{\mathbf{x}}^2 h \in \mathbb{R}^{D\times D}$ is the Hessian matrix of h with respect to $\mathbf{x}$.

Example: Itô's Product Rule

Let $\mathbf{x}_t, \mathbf{y}_t \in \mathbb{R}^D$ be vector-valued stochastic processes governed by the SDEs:

$$\mathrm{d}\mathbf{x}_t = \mathbf{a}(\mathbf{x}_t, t)\,\mathrm{d}t + b(t)\,\mathrm{d}\mathbf{w}_t,$$
$$\mathrm{d}\mathbf{y}_t = \mathbf{c}(\mathbf{y}_t, t)\,\mathrm{d}t + d(t)\,\mathrm{d}\mathbf{w}_t,$$

where $\mathbf{a}, \mathbf{c} : \mathbb{R}^D \times [0,T] \to \mathbb{R}^D$ are vector fields, and $b(t), d(t) \in \mathbb{R}$ are scalar-

valued functions. Here, $\mathbf{w}_t \in \mathbb{R}^D$ denotes a standard D-dimensional Brownian motion.

We aim to derive the SDE for the scalar-valued process

$$z(t) := \mathbf{x}_t^\top \mathbf{y}_t .$$

Applying the multivariate Itô formula to the bilinear function $h(\mathbf{x}, \mathbf{y}) := \mathbf{x}^\top \mathbf{y}$, we obtain:

$$\mathrm{d}(\mathbf{x}^\top \mathbf{y}) = (\mathrm{d}\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top \,\mathrm{d}\mathbf{y} + \mathrm{Tr}\left[\mathrm{d}\mathbf{x} \cdot (\mathrm{d}\mathbf{y})^\top\right].$$

Why? Stack the two processes into $\mathbf{z}_t := (\mathbf{x}_t; \mathbf{y}_t) \in \mathbb{R}^{2D}$ and expand $h(\mathbf{z}) := \mathbf{x}^\top \mathbf{y}$ as in the stochastic Taylor expansion Equation (C.1.1). The gradient and Hessian of h are

$$\nabla_{\mathbf{z}} h = \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}, \qquad \nabla^2_{\mathbf{z}} h = \begin{pmatrix} \mathbf{0} & \mathbf{I}_D \\ \mathbf{I}_D & \mathbf{0} \end{pmatrix},$$

and the quadratic covariation of $\mathbf{z}_t$ is, in block form,

$$\mathrm{d}\mathbf{z}_t \,\mathrm{d}\mathbf{z}_t^\top = \begin{pmatrix} \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{x}^\top & \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y}^\top \\ \mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x}^\top & \mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{y}^\top \end{pmatrix}.$$

Hence, by Equation (C.1.1),

$$\begin{aligned} \mathrm{d}(\mathbf{x}^\top \mathbf{y}) &= \nabla_{\mathbf{z}} h^\top \,\mathrm{d}\mathbf{z}_t + \frac{1}{2}\,\mathrm{Tr}\left(\nabla^2_{\mathbf{z}} h \;\mathrm{d}\mathbf{z}_t \,\mathrm{d}\mathbf{z}_t^\top\right) \\ &= \mathbf{y}^\top \,\mathrm{d}\mathbf{x} + \mathbf{x}^\top \,\mathrm{d}\mathbf{y} + \frac{1}{2}\,\mathrm{Tr}\begin{pmatrix} \mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x}^\top & \mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{y}^\top \\ \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{x}^\top & \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y}^\top \end{pmatrix} \\ &= (\mathrm{d}\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top \,\mathrm{d}\mathbf{y} + \frac{1}{2}\Big(\mathrm{Tr}\left[\mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x}^\top\right] + \mathrm{Tr}\left[\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y}^\top\right]\Big) \\ &= (\mathrm{d}\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top \,\mathrm{d}\mathbf{y} + \mathrm{Tr}\left[\mathrm{d}\mathbf{x} \cdot (\mathrm{d}\mathbf{y})^\top\right], \end{aligned}$$

using $\mathrm{Tr}[\mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x}^\top] = \mathrm{Tr}[\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y}^\top]$. The off-diagonal blocks of the Hessian are exactly what pick out the cross-covariation of $\mathbf{x}_t$ and $\mathbf{y}_t$. Note that nothing about the specific noise structure has been used so far; that enters only when $\mathrm{Tr}[\mathrm{d}\mathbf{x} \cdot (\mathrm{d}\mathbf{y})^\top]$ is evaluated next.

The Itô correction term is computed as:

$$\begin{aligned} \mathrm{d}\mathbf{x} \cdot (\mathrm{d}\mathbf{y})^\top &= b(t)\,\mathrm{d}\mathbf{w}_t \cdot [d(t)\,\mathrm{d}\mathbf{w}_t]^\top \\ &= b(t)d(t)\,\mathrm{d}\mathbf{w}_t \cdot \mathrm{d}\mathbf{w}_t^\top \\ &= b(t)d(t)\,\mathrm{d}t \cdot \mathbf{I}_D. \end{aligned}$$

Here all $(\mathrm{d}t)^2$ and $\mathrm{d}t\,\mathrm{d}\mathbf{w}_t$ terms have been dropped (they are $\mathcal{O}(\mathrm{d}t^{3/2})$ or smaller), and $\mathrm{d}\mathbf{w}_t\,\mathrm{d}\mathbf{w}_t^\top = \mathbf{I}_D\,\mathrm{d}t$ because the components of $\mathbf{w}_t$ are independent Brownian motions with $(\mathrm{d}w_t^{(i)})^2 = \mathrm{d}t$. Thus,

$$\mathrm{Tr}\left[\mathrm{d}\mathbf{x} \cdot (\mathrm{d}\mathbf{y})^\top\right] = b(t)d(t)\,\mathrm{Tr}(\mathbf{I}_D)\,\mathrm{d}t = Db(t)d(t)\,\mathrm{d}t.$$

Putting everything together, the resulting SDE is:

$$\mathrm{d}(\mathbf{x}^\top \mathbf{y}) = (\mathrm{d}\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top \,\mathrm{d}\mathbf{y} + Db(t)d(t)\,\mathrm{d}t \tag{C.1.3}$$

C.1.4 Itô's Formula's Application: Derivation of Fokker-Planck Equation

In this section, we apply Itô's formula from Equation (C.1.2) to derive the Fokker–Planck equation, a PDE that characterizes the time evolution of the probability density $p_t(\mathbf{x})$ associated with the D-dimensional diffusion process defined by the SDE in Equation (A.2.3).

Step 1: Apply Itô's Formula. Let $\phi(\mathbf{x}, t)$ be a smooth test function $\phi\colon \mathbb{R}^D \times [0, T] \to \mathbb{R}$. By Itô's Formula:

$$\mathrm{d}\phi(\mathbf{x}_t, t) = \left(\frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}}\phi^\top \mathbf{f}(\mathbf{x}_t, t) + \frac{1}{2}g^2(t)\,\mathrm{Tr}[\nabla^2_{\mathbf{x}}\phi]\right)\mathrm{d}t + g(t)\nabla_{\mathbf{x}}\phi^\top \,\mathrm{d}\mathbf{w}_t.$$

Step 2: Take Expectation. Taking expectation and using that the Itô integral has zero expectation under the standard integrability conditions:

$$\mathbb{E}[\mathrm{d}\phi(\mathbf{x}_t, t)] = \mathbb{E}\left[\left(\frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}}\phi^\top \mathbf{f}(\mathbf{x}_t, t) + \frac{1}{2}g^2(t)\,\mathrm{Tr}[\nabla^2_{\mathbf{x}}\phi]\right)\mathrm{d}t\right].$$

Step 3: Express Expectation via Density. This expectation can be written as:

$$\mathbb{E}[\mathrm{d}\phi(\mathbf{x}_t, t)] = \int \left(\frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}}\phi^\top \mathbf{f}(\mathbf{x}, t) + \frac{1}{2}g^2(t)\,\mathrm{Tr}[\nabla^2_{\mathbf{x}}\phi]\right) p_t(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t.$$

Step 4: Integrate by Parts. By multivariate integration by parts (see Section E.2) with $\mathbf{v} = \mathbf{f}p_t$, using $p_t \to 0$ as $\|\mathbf{x}\| \to \infty$,

$$\int \nabla_{\mathbf{x}}\phi^\top \mathbf{f} p_t \,\mathrm{d}\mathbf{x} = -\int \phi\, \nabla_{\mathbf{x}} \cdot (\mathbf{f}p_t)\,\mathrm{d}\mathbf{x}.$$

Applying it twice more, with $\mathbf{v} = \nabla_{\mathbf{x}}\phi$ against p_t and then $\mathbf{v} = \nabla_{\mathbf{x}}p_t$ against ϕ (using $p_t, \nabla_{\mathbf{x}}p_t \to 0$),

$$\int \mathrm{Tr}[\nabla^2_{\mathbf{x}}\phi]\, p_t \,\mathrm{d}\mathbf{x} = \int \Delta\phi\, p_t \,\mathrm{d}\mathbf{x} = -\int \nabla_{\mathbf{x}}\phi^\top \nabla_{\mathbf{x}} p_t \,\mathrm{d}\mathbf{x} = \int \phi\, \Delta p_t \,\mathrm{d}\mathbf{x}.$$

Step 5: Substitute and Rearrange. Note that

$$\mathbb{E}[\phi(\mathbf{x}_t, t)] = \int \phi(\mathbf{x}, t) p_t(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

Differentiating in time, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}[\phi(\mathbf{x}_t,t)] = \frac{\mathrm{d}}{\mathrm{d}t}\int \phi(\mathbf{x},t)p_t(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int \Big(\partial_t\phi(\mathbf{x},t)p_t(\mathbf{x}) + \phi(\mathbf{x},t)\partial_t p_t(\mathbf{x})\Big)\,\mathrm{d}\mathbf{x}.$$

Therefore,

$$\mathbb{E}[\mathrm{d}\phi(\mathbf{x}_t,t)] = \frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}[\phi(\mathbf{x}_t,t)]\,\mathrm{d}t = \int \Big(\partial_t\phi(\mathbf{x},t)p_t(\mathbf{x}) + \phi(\mathbf{x},t)\partial_t p_t(\mathbf{x})\Big)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t.$$

Equating this expression with Step 3, the terms $\int \partial_t\phi\, p_t\ \mathrm{d}\mathbf{x}\ \mathrm{d}t$ cancel. Applying Step 4 to the remaining spatial terms gives

$$\int \phi(\mathbf{x},t)\left[\frac{\partial p_t}{\partial t} + \nabla_{\mathbf{x}}\cdot(\mathbf{f}(\mathbf{x},t)p_t(\mathbf{x})) - \frac{1}{2}g^2(t)\Delta p_t(\mathbf{x})\right]\mathrm{d}\mathbf{x} = 0.$$

Since this holds for every smooth test function ϕ, we obtain the Fokker–Planck equation

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla_{\mathbf{x}}\cdot(\mathbf{f}(\mathbf{x},t)p_t(\mathbf{x})) + \frac{1}{2}g^2(t)\Delta p_t(\mathbf{x}).$$

C.1.5 Itô's Formula Application: Closed-Form Solution of a Linear SDE

This subsection demonstrates how to obtain a closed-form solution for a linear SDE by using an integration factor (similar to the ODE case) and Itô's formula. The approach mirrors classical techniques for solving linear ODEs, but adapted to the stochastic setting.

We consider a linear SDE of the form

$$\mathrm{d}\mathbf{x}_t = f(t)\mathbf{x}_t\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t, \tag{C.1.4}$$

where $f(t)$ and $g(t)$ are deterministic functions and $\mathbf{w}_t$ is a standard Wiener process.

Closed-Form Solution of a Linear ODE. For the deterministic analogue $\frac{\mathrm{d}x_t}{\mathrm{d}t} = f(t)\,x_t + g(t)$, define the integrating factor $\Psi(t) := \exp\big(-\int_0^t f(s)\,\mathrm{d}s\big)$, so that $\Psi'(t) = -f(t)\Psi(t)$. Then

$$\begin{aligned}\Psi(t)\frac{\mathrm{d}x_t}{\mathrm{d}t} = \Psi(t)f(t)\,x_t + \Psi(t)g(t) \quad &\Longleftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t}\big(\Psi(t)\,x_t\big) = \Psi(t)\,g(t)\\ &\Longleftrightarrow \quad x_t = \Psi(t)^{-1}\Big[x_0 + \int_0^t \Psi(s)\,g(s)\,\mathrm{d}s\Big].\end{aligned}$$

Equation (C.1.5) below is the stochastic analogue: the same steps go through with $g(s)\,\mathrm{d}s$ replaced by $g(s)\,\mathrm{d}\mathbf{w}(s)$.

Closed-Form Solution of a Linear SDE. We derive the explicit solution to the linear SDE in Equation (C.1.4) using the method of integrating factors. This type of forward SDE commonly arises in diffusion models (see Section 4.1).

Step 1: Define an Integration Factor. Let

$$\Psi(t) := \exp\left(-\int_0^t f(s)\mathrm{d}s\right), \quad \text{and define} \quad \mathbf{y}_t := \Psi(t)\mathbf{x}_t.$$

Step 2: Apply Itô's Formula. We apply Itô's formula to the function $\mathbf{h}(\mathbf{x}, t) := \Psi(t)\mathbf{x}$. This is actually a special case of the Itô product rule in Equation (C.1.3). Since $\Psi(t)$ is deterministic, there is no cross-variation term, and the formula simplifies to:

$$\begin{aligned}
\mathrm{d}\mathbf{y}_t &= \mathrm{d}[\Psi(t)\mathbf{x}_t] \\
&= \Psi'(t)\mathbf{x}_t\mathrm{d}t + \Psi(t)\mathrm{d}\mathbf{x}_t \\
&= -f(t)\Psi(t)\mathbf{x}_t\mathrm{d}t + \Psi(t)\left[f(t)\mathbf{x}_t\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t\right] \\
&= \Psi(t)g(t)\mathrm{d}\mathbf{w}_t.
\end{aligned}$$

Hence,

$$\mathbf{y}_t = \mathbf{y}_0 + \int_0^t \Psi(s)g(s)\mathrm{d}\mathbf{w}(s) = \mathbf{x}_0 + \int_0^t \Psi(s)g(s)\mathrm{d}\mathbf{w}(s),$$

since $\Psi(0) = 1$.

Step 3: Solve for $\mathbf{x}_t$. Using $\mathbf{x}_t = \Psi(t)^{-1}\mathbf{y}_t$, we obtain

$$\mathbf{x}_t = e^{\int_0^t f(s)\mathrm{d}s}\left[\mathbf{x}_0 + \int_0^t e^{-\int_0^s f(r)\mathrm{d}r} g(s)\mathrm{d}\mathbf{w}(s)\right]. \tag{C.1.5}$$

This provides an explicit solution to the vector-valued SDE.

Below, we demonstrate two alternative approaches to compute the analytical form of $p_t(\mathbf{x}_t|\mathbf{x}_0)$.

Analysis of the Closed-Form Solution. Equation (C.1.5) reconfirms that $p_t(\mathbf{x}_t|\mathbf{x}_0)$ is Gaussian. To see this, define

$$\phi(s) := e^{-\int_0^s f(u)\,\mathrm{d}u} g(s),$$

which is a deterministic matrix-valued function of time (assuming $f(u)$ and $g(s)$ are deterministic). The Itô integral $\int_0^t \phi(s)\,\mathrm{d}\mathbf{w}_s$ is then a zero-mean Gaussian random variable, as it is the stochastic integral of a deterministic function with respect to Brownian motion. Therefore, $\mathbf{x}_t|\mathbf{x}_0$ is an affine transformation of a Gaussian random variable and hence itself Gaussian. Its distribution is fully characterized by its conditional mean and covariance.

We define the conditional mean and covariance (given initial condition $\mathbf{x}_0$) as

$$\mathbf{m}(t) := \mathbb{E}[\mathbf{x}_t|\mathbf{x}_0], \quad \mathbf{P}(t) := \mathbb{E}[(\mathbf{x}_t - \mathbf{m}(t))(\mathbf{x}_t - \mathbf{m}(t))^\top|\mathbf{x}_0].$$

Mean. Using linearity of expectation and the fact that the Itô integral of a deterministic function has zero mean:

$$\mathbf{m}(t) = \mathbb{E}\left[e^{\int_0^t f(s)\,\mathrm{d}s}\left(\mathbf{x}_0 + \int_0^t \phi(s)\,\mathrm{d}\mathbf{w}_s\right)\middle|\mathbf{x}_0\right] = e^{\int_0^t f(s)\,\mathrm{d}s}\mathbf{x}_0.$$

Covariance. Let $\mathbf{z}_t := \int_0^t \phi(s)\,\mathrm{d}\mathbf{w}_s$. Then $\mathbf{x}_t - \mathbf{m}(t) = A(t)\mathbf{z}_t$, so

$$\mathbf{P}(t) = e^{2\int_0^t f(s)\,\mathrm{d}s}\mathbb{E}[\mathbf{z}_t\mathbf{z}_t^\top].$$

By Itô isometry[1],

$$\mathbb{E}[\mathbf{z}_t\mathbf{z}_t^\top] = \left(\int_0^t \phi^2(s)\,\mathrm{d}s\right)\mathbf{I}_D,$$

hence,

$$\mathbf{P}(t) = e^{2\int_0^t f(s)\,\mathrm{d}s}\left(\int_0^t \left(e^{-\int_0^s f(u)\,\mathrm{d}u}g(s)\right)^2 \mathrm{d}s\right)\mathbf{I}_D.$$

This shows the conditional covariance is isotropic.

Derivation of Mean and Variance ODEs in Equation (4.4.3). Alternatively, we can derive the moment evolution equations directly from the linear SDE Equation (C.1.4).

Mean Evolution. Taking the conditional expectation of both sides of the SDE and using linearity:

$$\frac{\mathrm{d}\mathbf{m}(t)}{\mathrm{d}t} = \mathbb{E}[f(t)\mathbf{x}_t|\mathbf{x}_0] = f(t)\mathbb{E}[\mathbf{x}_t|\mathbf{x}_0] = f(t)\mathbf{m}(t).$$

Solving this linear ODE gives $\mathbf{m}(t) = \exp\left(\int_0^t f(s)\,\mathrm{d}s\right)\mathbf{x}_0$, in agreement with the direct computation above.

[1] Itô's isometry links stochastic integrals to standard integrals in expectation; we omit the proof as it requires the full machinery of Itô calculus. For a process $\boldsymbol{\psi} : [0,T] \to \mathbb{R}^{D\times D}$, Itô isometry states

$$\mathbb{E}\left[\left\|\int_0^T \boldsymbol{\psi}(t)\mathrm{d}\mathbf{w}_t\right\|^2\right] = \mathbb{E}\left[\int_0^T \|\boldsymbol{\psi}(t)\|_F^2\mathrm{d}t\right],$$

where $\|\boldsymbol{\psi}(t)\|_F^2 = \sum_{i,j=1}^D |\psi_{ij}(t)|^2$ is the Frobenius norm. For $\boldsymbol{\psi}(t) \in \mathbb{R}^D$ (a vector), the integral is scalar and the isometry simplifies to

$$\mathbb{E}\left[\left(\int_0^T \boldsymbol{\psi}(t)\mathrm{d}\mathbf{w}_t\right)^2\right] = \mathbb{E}\left[\int_0^T \|\boldsymbol{\psi}(t)\|^2\mathrm{d}t\right].$$

Covariance Evolution. Define the centered process $\tilde{\mathbf{x}}_t := \mathbf{x}_t - \mathbf{m}(t)$. Applying Itô's product rule (see Equation (C.1.3)):

$$\mathrm{d}\left(\tilde{\mathbf{x}}_t \tilde{\mathbf{x}}_t^\top\right) = \mathrm{d}\tilde{\mathbf{x}}_t \cdot \tilde{\mathbf{x}}_t^\top + \tilde{\mathbf{x}}_t \cdot \mathrm{d}\tilde{\mathbf{x}}_t^\top + \mathrm{d}\tilde{\mathbf{x}}_t \cdot \mathrm{d}\tilde{\mathbf{x}}_t^\top.$$

From the SDE, we compute:

$$\mathrm{d}\tilde{\mathbf{x}}_t = \mathrm{d}\mathbf{x}_t - \mathrm{d}\mathbf{m}(t) = f(t)\tilde{\mathbf{x}}_t\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t.$$

Substituting into the product rule and taking expectation:

$$\begin{aligned}
\frac{\mathrm{d}\mathbf{P}(t)}{\mathrm{d}t} &= \frac{1}{\mathrm{d}t}\,\mathbb{E}\big[\,\mathrm{d}(\tilde{\mathbf{x}}_t\tilde{\mathbf{x}}_t^\top)\,\big|\,\mathbf{x}_0\big] \\
&= \frac{1}{\mathrm{d}t}\,\mathbb{E}\big[f(t)\,\tilde{\mathbf{x}}_t\tilde{\mathbf{x}}_t^\top\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t\,\tilde{\mathbf{x}}_t^\top + f(t)\,\tilde{\mathbf{x}}_t\tilde{\mathbf{x}}_t^\top\,\mathrm{d}t \\
&\qquad\qquad + g(t)\,\tilde{\mathbf{x}}_t\,\mathrm{d}\mathbf{w}_t^\top + g^2(t)\,\mathbf{I}_D\,\mathrm{d}t + \mathcal{O}(\mathrm{d}t^{3/2})\,\big|\,\mathbf{x}_0\big] \\
&= \mathbb{E}[f(t)\tilde{\mathbf{x}}_t\tilde{\mathbf{x}}_t^\top + \tilde{\mathbf{x}}_t\tilde{\mathbf{x}}_t^\top f(t) + g^2(t)\mathbf{I}_D\,|\,\mathbf{x}_0] \\
&= 2f(t)\mathbf{P}(t) + g^2(t)\mathbf{I}_D.
\end{aligned}$$

The two $\mathrm{d}\mathbf{w}_t$ terms vanish because the increment $\mathrm{d}\mathbf{w}_t$ is independent of $\tilde{\mathbf{x}}_t$ and of $\mathbf{x}_0$, and has zero mean. Thus, we recover the moment evolution equations in Equation (4.4.3).

C.2 Change-of-Variable For Measures: Girsanov's Theorem in Diffusion Models

Diffusion models harness SDEs to transform simple noise into rich data distributions. At the heart of this transformation lies a profound idea: we can reinterpret randomness by modifying only the deterministic part of an SDE (the drift) while preserving its underlying stochasticity. This is precisely where Girsanov's theorem enters the picture.

The Core Idea. Consider an observed continuous trajectory that describes the data's evolution from time $t = 0$ to $t = T$, denoted as $\mathbf{x}_{0:T} := \{\mathbf{x}_t | t \in [0, T]\}$. Girsanov's theorem addresses a fundamental question:

Question C.2.1

Given this single observed path, what is its likelihood if we assume it was generated by one SDE, versus if we assume it was generated by a different SDE?

We compare two hypothetical models for generating the same trajectory. They have the same initial distribution and diffusion coefficient, but differ in their deterministic "push" or drift function. We assume $\mathbf{x}_0$ has the same initial distribution for both assumed generating processes.

To build intuition, imagine $\mathbf{x}_{0:T}$ as a wiggly line drawn on paper. One hypothesis assigns trajectory probabilities according to drift $\mathbf{f}$, while another uses a different drift $\tilde{\mathbf{f}}$ with the same diffusion coefficient. Girsanov's theorem compares the corresponding path measures and quantifies how changing the drift changes the relative weighting assigned to the same trajectories.

The Setup. Let $P_{\mathbf{f}}$ and $P_{\tilde{\mathbf{f}}}$ denote the path measures induced by two SDEs with the same initial distribution and the same diffusion coefficient, but different drifts:

$$\begin{aligned} \mathrm{d}\mathbf{x}_t &= \mathbf{f}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t, \\ \mathrm{d}\mathbf{x}_t &= \tilde{\mathbf{f}}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t. \end{aligned}$$

For simplicity, assume $g(t) > 0$ on the interval of interest and the standard integrability conditions required by Girsanov's theorem. Take $P_{\tilde{\mathbf{f}}}$ as the reference path measure, and let $\mathbf{w}_t$ denote the Brownian motion under this reference measure. Define

$$\boldsymbol{\delta}_t := \mathbf{f}(\mathbf{x}_t, t) - \tilde{\mathbf{f}}(\mathbf{x}_t, t).$$

Girsanov's Likelihood Ratio. Girsanov's theorem gives the Radon–Nikodym derivative between the two path measures:

$$\frac{\mathrm{d}P_{\mathbf{f}}}{\mathrm{d}P_{\tilde{\mathbf{f}}}}(\mathbf{x}_{0:T}) = \exp\left(\int_0^T \boldsymbol{\delta}_t^\top g(t)^{-1}\,\mathrm{d}\mathbf{w}_t - \frac{1}{2}\int_0^T \left\|g(t)^{-1}\boldsymbol{\delta}_t\right\|^2 \mathrm{d}t\right).$$

This compact formula is an exponential of two integrals. The first is an Itô integral, while the second is a standard Riemann integral. This ratio is crucial in diffusion models, allowing us to compare different stochastic dynamics and to connect path-space discrepancies with likelihood-based training objectives.

Girsanov's theorem is best understood as a change-of-variable formula for *measures*. Just as a change of variables in calculus transforms an integral between coordinate systems via the Jacobian determinant, Girsanov's theorem provides the corresponding factor (the Radon–Nikodym derivative) to transform probabilities or expectations between two stochastic processes, when the drift changes but the diffusion remains the same.

C.2.1 Girsanov's Theorem as a Bridge Between Likelihood Training and Score Matching

After understanding how Girsanov's theorem relates the likelihoods of a single path under different drift assumptions, we now delve into its implications for diffusion models.

Recall the forward SDE in diffusion models:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}_t,$$

which induces a *path distribution* P over full trajectories $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t=0}^T$ (that is, the joint law of the process over the entire time interval). The reverse-time SDE, parameterized by a learnable score function $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$, is given by

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)\right]\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}_t,$$

which in turn defines another path distribution $P_{\boldsymbol{\phi}}$ over trajectories.

Two Notions in Diffusion Models. It is useful to distinguish two levels at which two stochastic processes can agree:

- **Concept 1. Marginal Distribution Matching:** The processes have the same distribution $p_t(\mathbf{x}_t)$ at every fixed time t. Thus, their time-indexed "snapshots" agree.
- **Concept 2. Joint Path Distribution Matching:** The processes have the same probability law over entire trajectories. This stronger notion also matches how states at different times are coupled, including their transition laws and temporal dependencies.

Matching the full path law automatically implies marginal matching, since each p_t is obtained by projecting the path law onto the state at time t. The converse is generally false: two processes can have identical marginals at every time while inducing different couplings between those marginals. The SDE and its associated PF-ODE provide an important example: they share the same marginal density path, but generally have different sample trajectories and path laws.

Girsanov Connects Score Error to Path-Law Mismatch. The exact reverse-time SDE satisfies the stronger notion above. Under standard regularity conditions, when initialized from the exact terminal marginal p_T and equipped with the true score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, it is the time reversal of the forward diffusion and therefore reproduces its full time-reversed path law.

Let P^{rev} denote this exact reverse-time path law, and let P_ϕ denote the learned reverse-time path law obtained by replacing the true score with $\mathbf{s}_\phi$ and initializing from p_{prior}. Given $\mathbf{x}_T$, both conditional path laws are reverse-time SDEs with the same $g(t)$ and drifts $\mathbf{f} - g^2\nabla_{\mathbf{x}} \log p_t$ (exact) and $\mathbf{f} - g^2\mathbf{s}_\phi$ (learned). Let

$$\boldsymbol{\delta}_t := [\mathbf{f} - g^2\mathbf{s}_\phi] - [\mathbf{f} - g^2\nabla_{\mathbf{x}} \log p_t] = g^2(t)(\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) - \mathbf{s}_\phi(\mathbf{x}_t, t)),$$

so $\|g^{-1}\boldsymbol{\delta}_t\|^2 = g^2 \|\mathbf{s}_\phi - \nabla_{\mathbf{x}} \log p_t\|^2$, and note $\mathbf{x}_t \sim p_t$ under P^{rev}. Take P^{rev} as the reference measure, with Brownian motion $\bar{\mathbf{w}}_t$ (in the reverse-time clock both are forward SDEs). Then

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}(P^{\mathrm{rev}} \| P_\phi) &= \mathbb{E}_{P^{\mathrm{rev}}}\Big[\log \frac{\mathrm{d}P^{\mathrm{rev}}}{\mathrm{d}P_\phi}\Big] \\
&= \mathbb{E}_{P^{\mathrm{rev}}}\Big[\log \frac{p_T(\mathbf{x}_T)}{p_{\mathrm{prior}}(\mathbf{x}_T)}\Big] - \mathbb{E}_{P^{\mathrm{rev}}}\Big[\log \frac{\mathrm{d}P_\phi(\cdot \,|\, \mathbf{x}_T)}{\mathrm{d}P^{\mathrm{rev}}(\cdot \,|\, \mathbf{x}_T)}\Big] \\
&\overset{\text{Girsanov}}{=} \mathcal{D}_{\mathrm{KL}}(p_T \| p_{\mathrm{prior}}) - \mathbb{E}_{P^{\mathrm{rev}}}\Big[\int_0^T \boldsymbol{\delta}_t^\top g(t)^{-1} \, \mathrm{d}\bar{\mathbf{w}}_t\Big] \\
&\qquad + \frac{1}{2}\int_0^T \mathbb{E}_{P^{\mathrm{rev}}}\big[\|g(t)^{-1}\boldsymbol{\delta}_t\|^2\big] \, \mathrm{d}t \\
&\overset{\text{Itô}}{=} \mathcal{D}_{\mathrm{KL}}(p_T \| p_{\mathrm{prior}}) + \frac{1}{2}\int_0^T \mathbb{E}_{P^{\mathrm{rev}}}\big[\|g(t)^{-1}\boldsymbol{\delta}_t\|^2\big] \, \mathrm{d}t \\
&= \mathcal{D}_{\mathrm{KL}}(p_T \| p_{\mathrm{prior}}) \\
&\qquad + \frac{1}{2}\int_0^T g^2(t) \, \mathbb{E}_{\mathbf{x}_t \sim p_t}\big[\|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|^2\big] \, \mathrm{d}t,
\end{aligned}$$

where the Itô integral against $\bar{\mathbf{w}}_t$ has zero mean under P^{rev}. That is,

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}(P^{\mathrm{rev}} \| P_\phi) &= \mathcal{D}_{\mathrm{KL}}(p_T \| p_{\mathrm{prior}}) \\
&\quad + \frac{1}{2}\int_0^T g^2(t) \mathbb{E}_{\mathbf{x}_t \sim p_t}\Big[\|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|^2\Big] \, \mathrm{d}t.
\end{aligned} \tag{C.2.1}$$

The first term measures the mismatch between the true terminal distribution and the prior used to initialize generation; it vanishes when $p_T = p_{\text{prior}}$. The second term measures the accumulated mismatch between the learned and exact reverse drifts.

Thus, score matching with the particular likelihood weighting $g^2(t)$ has a direct path-space interpretation: minimizing the weighted score error minimizes the KL divergence between the exact and learned reverse path laws, up to the fixed terminal-distribution term. When denoising score matching is used instead of marginal score matching, the corresponding objectives differ by a parameter-independent constant. A different time weighting generally does not correspond to this same path-space KL.

Connection to Data Likelihood. The path-space KL also controls the discrepancy between the generated and data marginals. Since taking the endpoint of a trajectory is a measurable projection and KL divergence cannot increase under marginalization,

$$\mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\boldsymbol{\phi}}) \leq \mathcal{D}_{\text{KL}}(P^{\text{rev}} \| P_{\boldsymbol{\phi}}), \tag{C.2.2}$$

where $p_{\boldsymbol{\phi}}$ denotes the generated marginal at data time.

Combining this with Equation (C.2.1) shows that likelihood-weighted score matching, together with the terminal-distribution mismatch, provides an upper bound on the data-side KL divergence and therefore on the expected negative log-likelihood up to a constant independent of $\boldsymbol{\phi}$ (Song *et al.*, 2021).

D

Supplementary Materials and Proofs

D.1 Variational Perspective

D.1.1 Theorem 2.2.1: Equivalence Between Marginal and Conditional KL Minimization

Proof. **Derivation of Equation (2.2.3).**

We start by expanding the right-hand side expectation:

$$\begin{aligned}&\mathbb{E}_{p(\mathbf{x}_0,\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))\right]\\ =&\int\int p(\mathbf{x}_0,\mathbf{x}_i)\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))\,\mathrm{d}\mathbf{x}_0\,\mathrm{d}\mathbf{x}_i.\end{aligned}$$

By the definition of KL divergence,

$$\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))=\int p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\log\frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\,\mathrm{d}\mathbf{x}_{i-1}.$$

Substituting this into the expectation, we have

$$\int\int\int p(\mathbf{x}_0,\mathbf{x}_i)p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\log\frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\,\mathrm{d}\mathbf{x}_{i-1}\,\mathrm{d}\mathbf{x}_0\,\mathrm{d}\mathbf{x}_i.$$

Using the chain rule of probability,

$$p(\mathbf{x}_0,\mathbf{x}_i)=p(\mathbf{x}_i)p(\mathbf{x}_0|\mathbf{x}_i),$$

we rewrite the integral as

$$\int p(\mathbf{x}_i)\int p(\mathbf{x}_0|\mathbf{x}_i)\int p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)\log\frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\,\mathrm{d}\mathbf{x}_{i-1}\,\mathrm{d}\mathbf{x}_0\,\mathrm{d}\mathbf{x}_i.$$

This allows us to express the expectation in nested form:

$$\mathbb{E}_{p(\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}\left[\log\frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\right]\right]\right].$$

Next, we apply the decomposition of the logarithm:

$$\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} = \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p(\mathbf{x}_{i-1}|\mathbf{x}_i)} + \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}.$$

Substituting this back into the expectation gives two terms:

$$\begin{aligned}&\mathbb{E}_{p(\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}\right]\right]\right] \\ &+ \mathbb{E}_{p(\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\right]\right]\right].\end{aligned}$$

Since the second logarithmic term does not depend on $\mathbf{x}_0$, by the law of total probability $\left(\int p(\mathbf{x}_0|\mathbf{x}_i)\, p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0 = p(\mathbf{x}_{i-1}|\mathbf{x}_i)\right)$

$$\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i,\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\right]\right] = \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}\left[\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}\right].$$

Similarly, the first term is the KL divergence

$$\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\|p(\mathbf{x}_{i-1}|\mathbf{x}_i))\right].$$

Putting it all together, we obtain the decomposition:

$$\begin{aligned}&\mathbb{E}_{p(\mathbf{x}_0,\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))\right] \\ =&\mathbb{E}_{p(\mathbf{x}_i)}\left[\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\|p(\mathbf{x}_{i-1}|\mathbf{x}_i))\right]\right] \quad \text{(does not depend on } \phi\text{)} \\ &+ \mathbb{E}_{p(\mathbf{x}_i)}\left[\mathcal{D}_{\mathrm{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i)\|p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))\right].\end{aligned}$$

Proof of Optimality. To prove:

$$p^*(\mathbf{x}_{i-1}|\mathbf{x}_i) = p(\mathbf{x}_{i-1}|\mathbf{x}_i) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)}\left[p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)\right], \qquad \mathbf{x}_i \sim p_i.$$

The first identity follows from the fact that the KL divergence $\mathcal{D}_{\mathrm{KL}}(p\|p_\phi)$ is minimized when $p^* = p$, assuming the parameterization is sufficiently expressive. The second identity follows directly from the law of total probability. ■

D.1.2 Theorem 2.2.3: ELBO of Diffusion Model

Proof. For notational simplicity, we denote

$$\mathbf{x}_{1:L} := (\mathbf{x}_1, \ldots, \mathbf{x}_L).$$

Step 1: Apply Jensen's Inequality. The marginal log-likelihood is

$$\log p_\phi(\mathbf{x}_0) = \log \int p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L})\, \mathrm{d}\mathbf{x}_{1:L},$$

where the joint generative distribution is

$$p_{\boldsymbol{\phi}}(\mathbf{x}_0, \mathbf{x}_{1:L}) = p_{\text{prior}}(\mathbf{x}_L) \prod_{i=1}^{L} p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i).$$

We introduce the forward noising distribution (the only difference from the HVAE is that the encoder is fixed, and it is factored slightly differently for better tractability)

$$p(\mathbf{x}_{1:L}|\mathbf{x}_0) = \prod_{i=1}^{L} p(\mathbf{x}_i|\mathbf{x}_{i-1}),$$

and rewrite

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}_0) = \log \int p(\mathbf{x}_{1:L}|\mathbf{x}_0) \frac{p_{\boldsymbol{\phi}}(\mathbf{x}_0, \mathbf{x}_{1:L})}{p(\mathbf{x}_{1:L}|\mathbf{x}_0)} \, \mathrm{d}\mathbf{x}_{1:L}.$$

Applying Jensen's inequality (see Section E.3) yields

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}_0) \geq \mathbb{E}_{p(\mathbf{x}_{1:L}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\phi}}(\mathbf{x}_0, \mathbf{x}_{1:L})}{p(\mathbf{x}_{1:L}|\mathbf{x}_0)} \right] =: \mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \boldsymbol{\phi}),$$

and hence

$$-\log p_{\boldsymbol{\phi}}(\mathbf{x}_0) \leq -\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \boldsymbol{\phi}).$$

Step 2: Expand the ELBO. Substituting the factorizations into the ELBO gives

$$\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \boldsymbol{\phi}) = \mathbb{E}_{p(\mathbf{x}_{1:L}|\mathbf{x}_0)} \left[\log p_{\text{prior}}(\mathbf{x}_L) + \sum_{i=1}^{L} \log p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i) - \sum_{i=1}^{L} \log p(\mathbf{x}_i|\mathbf{x}_{i-1}) \right].$$

Therefore,

$$-\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \boldsymbol{\phi}) = \mathbb{E}_{p(\mathbf{x}_{1:L}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_L|\mathbf{x}_0)}{p_{\text{prior}}(\mathbf{x}_L)} + \sum_{i=2}^{L} \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)} - \log p_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_1) \right].$$

Here we used the identity

$$p(\mathbf{x}_{1:L}|\mathbf{x}_0) = p(\mathbf{x}_L|\mathbf{x}_0) \prod_{i=2}^{L} p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0),$$

which follows from the Markov structure of the forward process.

Now group the terms according to their dependence:

$$-\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \boldsymbol{\phi}) = \underbrace{\mathbb{E}_{p(\mathbf{x}_L|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_L|\mathbf{x}_0)}{p_{\text{prior}}(\mathbf{x}_L)} \right]}_{\mathcal{L}_{\text{prior}}(\mathbf{x}_0)} + \underbrace{\mathbb{E}_{p(\mathbf{x}_1|\mathbf{x}_0)} \left[-\log p_{\boldsymbol{\phi}}(\mathbf{x}_0|\mathbf{x}_1) \right]}_{\mathcal{L}_{\text{recon.}}(\mathbf{x}_0; \boldsymbol{\phi})}$$
$$+ \underbrace{\sum_{i=2}^{L} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \left[\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_{\boldsymbol{\phi}}(\mathbf{x}_{i-1}|\mathbf{x}_i)) \right]}_{\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \boldsymbol{\phi})}.$$

This proves the claimed decomposition. Since the encoder is fixed, only $\mathcal{L}_{\text{recon.}}$ and $\mathcal{L}_{\text{diffusion}}$ depend on $\boldsymbol{\phi}$. ■

D.2 Score-Based Perspective

D.2.1 Proposition 3.2.1: Tractable Score Matching via Integration by Parts

Proof. **Expanding $\mathcal{L}_{\mathsf{SM}}(\phi)$.** Let us expand the squared difference inside the expectation:

$$\begin{aligned}\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi}) &= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2 - 2\langle \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{s}(\mathbf{x})\rangle + \|\mathbf{s}(\mathbf{x})\|_2^2\right] \\ &= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\|_2^2\right] - \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\langle \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{s}(\mathbf{x})\rangle\right] \\ &\quad + \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\|\mathbf{s}(\mathbf{x})\|_2^2\right].\end{aligned}$$

We now focus on the cross-product term:

$$\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\langle \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{s}(\mathbf{x})\rangle\right].$$

Using the fact that

$$\nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p_{\mathrm{data}}(\mathbf{x})}{p_{\mathrm{data}}(\mathbf{x})},$$

and assuming $p_{\mathrm{data}}(\mathbf{x})$ is not zero (e.g., on its support), the cross-product term becomes:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\langle \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{s}(\mathbf{x})\rangle\right] &= \int \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})^\top \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}) p_{\mathrm{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\ &= \int \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})^\top \nabla_{\mathbf{x}} p_{\mathrm{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\ &= \sum_{i=1}^{D} \int s_{\boldsymbol{\phi}}^{(i)}(\mathbf{x}) \partial_{x_i} p_{\mathrm{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x},\end{aligned}$$

where $s_{\boldsymbol{\phi}}^{(i)}(\mathbf{x})$ is the i-th component of the score function

$$\mathbf{s}_{\boldsymbol{\phi}} = \left(s_{\boldsymbol{\phi}}^{(1)}, s_{\boldsymbol{\phi}}^{(2)}, \ldots, s_{\boldsymbol{\phi}}^{(D)}\right).$$

Integration by Parts. We apply the multivariate integration by parts of Section E.2 to the scalar field p_{data} and the vector field $\mathbf{s}_{\boldsymbol{\phi}}$, assuming that

$$|s_{\boldsymbol{\phi}}^{(i)}(\mathbf{x})\, p_{\mathrm{data}}(\mathbf{x})| \to 0 \quad \text{sufficiently fast as } \|\mathbf{x}\| \to \infty.$$

Since $\nabla_{\mathbf{x}} \cdot \mathbf{s}_{\boldsymbol{\phi}} = \sum_{i=1}^{D} \partial_{x_i} s_{\boldsymbol{\phi}}^{(i)}$, we get:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\langle \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{s}(\mathbf{x})\rangle\right] &= -\sum_{i=1}^{D} \int \partial_{x_i} s_{\boldsymbol{\phi}}^{(i)}(\mathbf{x}) p_{\mathrm{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\ &= -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\mathrm{Tr}\left(\nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\right)\right].\end{aligned}$$

Integration by parts works here precisely because we are dealing with the score: it moves the partial derivative off the unknown p_{data} and onto the known $s_{\boldsymbol{\phi}}^{(i)}$.

Combining all results, we have:

$$\mathcal{L}_{\text{SM}}(\boldsymbol{\phi}) = \underbrace{\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\text{Tr}\left(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\right) + \frac{1}{2}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})\right\|_2^2\right]}_{\widetilde{\mathcal{L}}_{\text{SM}}(\boldsymbol{\phi})} + \underbrace{\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\left\|\mathbf{s}(\mathbf{x})\right\|_2^2\right]}_{=:C},$$

where C depends only on the distribution p_{data}, which concludes the proof. ■

D.2.2 Theorem 3.3.1: Equivalence Between SM and DSM Minimization

Proof. Expanding both $\mathcal{L}_{\text{SM}}(\boldsymbol{\phi};\sigma)$ and $\mathcal{L}_{\text{DSM}}(\boldsymbol{\phi};\sigma)$, we have:

$$\mathcal{L}_{\text{SM}}(\boldsymbol{\phi};\sigma) = \frac{1}{2}\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\Big[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2 - 2\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}) + \left\|\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}})\right\|_2^2\Big],$$

$$\mathcal{L}_{\text{DSM}}(\boldsymbol{\phi};\sigma) = \frac{1}{2}\mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\Big[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2 - 2\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) + \left\|\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\right\|_2^2\Big].$$

Subtracting the two equations yields:

$$\begin{aligned}
&\mathcal{L}_{\text{SM}}(\boldsymbol{\phi};\sigma) - \mathcal{L}_{\text{DSM}}(\boldsymbol{\phi};\sigma)\\
&= \frac{1}{2}\left(\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2\right)\\
&\quad - \left(\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\left[\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}})\right]\right.\\
&\qquad\qquad \left. - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left[\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\right]\right)\\
&\quad + \frac{1}{2}\left(\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\left\|\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}})\right\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left\|\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\right\|_2^2\right).
\end{aligned}$$

Next, we address the three terms one at a time. For the first term, since $p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x}$, we can rewrite it as:

$$\begin{aligned}
\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2 &= \int\left(\int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2\,\mathrm{d}\tilde{\mathbf{x}}\\
&= \int p_{\text{data}}(\mathbf{x})\int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2\,\mathrm{d}\tilde{\mathbf{x}}\,\mathrm{d}\mathbf{x}\\
&= \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left\|\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)\right\|_2^2.
\end{aligned}$$

Thus, the first term is zero. For the second term:

$$\begin{aligned}
&\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})}\left[\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\right] \\
=&\int p_\sigma(\tilde{\mathbf{x}})\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top \frac{\nabla_{\tilde{\mathbf{x}}} p_\sigma(\tilde{\mathbf{x}})}{p_\sigma(\tilde{\mathbf{x}})}\,\mathrm{d}\tilde{\mathbf{x}} \\
=&\int \mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top \nabla_{\tilde{\mathbf{x}}} \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\tilde{\mathbf{x}} \\
=&\int\int \mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top \nabla_{\tilde{\mathbf{x}}} p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\,\mathrm{d}\tilde{\mathbf{x}}\,\mathrm{d}\mathbf{x} \\
=&\mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left[\mathbf{s}_{\boldsymbol{\phi}}(\tilde{\mathbf{x}};\sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\right].
\end{aligned} \tag{D.2.1}$$

Thus, it is also zero. For the third term, note that:

$$C := \frac{1}{2}\left(\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma(\tilde{\mathbf{x}})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2\right)$$

depends only on $p_{\text{data}}(\mathbf{x})$ and $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$, and hence it is constant with respect to $\boldsymbol{\phi}$. ■

D.2.3 Lemma 3.3.2: Tweedie's Formula

We first state a more general form of Tweedie's formula, which considers time-dependent Gaussian perturbations, and we provide its proof below.

Tweedie's Identity with Time-Dependent Parameters. Let $\mathbf{x}_t|\mathbf{x}_0 \sim \mathcal{N}(\cdot; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$. Then Tweedie's formula says

$$\alpha_t \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t] = \mathbf{x}_t + \sigma_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t),$$

where the expectation is taken over the posterior distribution $p(\mathbf{x}_0|\mathbf{x}_t)$ of $\mathbf{x}_0$ given the observed $\mathbf{x}_t$, and $p_t(\mathbf{x}_t)$ is the marginal density of $\mathbf{x}_t$.

Proof.

Marginal Density and Its Score. We recall that the marginal density of $\mathbf{x}_t$ is given by

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t|\mathbf{x}_0) p_0(\mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0.$$

We now compute the score function:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} = \frac{1}{p_t(\mathbf{x}_t)} \int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t|\mathbf{x}_0) p_0(\mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0.$$

We therefore need to compute the gradient of the conditional density.

Gradient of the Conditional and Rearrangement. The gradient of the conditional Gaussian density (note: different from the score, since there is no log) is:

$$\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t|\mathbf{x}_0) = -p_t(\mathbf{x}_t|\mathbf{x}_0) \cdot \sigma_t^{-2}(\mathbf{x}_t - \alpha_t \mathbf{x}_0).$$

Substituting this into the previous expression, we have:

$$\begin{aligned}
\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t) &= \int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t|\mathbf{x}_0) p_0(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0 \\
&= -\sigma_t^{-2} \int (\mathbf{x}_t - \alpha_t \mathbf{x}_0) p_t(\mathbf{x}_t|\mathbf{x}_0) p_0(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0 \\
&= -\sigma_t^{-2} \int (\mathbf{x}_t - \alpha_t \mathbf{x}_0) p(\mathbf{x}_0|\mathbf{x}_t) p_t(\mathbf{x}_t)\, \mathrm{d}\mathbf{x}_0 \\
&= -p_t(\mathbf{x}_t)\sigma_t^{-2} \left(\mathbf{x}_t - \alpha_t \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t]\right).
\end{aligned}$$

Dividing both sides by $p_t(\mathbf{x}_t)$, we obtain:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = -\sigma_t^{-2} \left(\mathbf{x}_t - \alpha_t \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t]\right).$$

Rearranging yields:

$$\mathbf{x}_t + \sigma_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \alpha_t \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t].$$

This completes the derivation.

D.2.4 Stein's Identity and Surrogate SURE Objective

Stein's Identity. Stein's identity is the integration-by-parts technique that turns expectations under an unknown density into expectations of observable functions and their derivatives, which cancels the partition function and enables unbiased, tractable objectives and tests without ever evaluating the unknown density or the partition function. We begin with the simplest one-dimensional case and then extend it to the form needed to prove the surrogate loss for SURE.

1D, Standard Normal Case. If $z \sim \mathcal{N}(0,1)$ and f has suitable decay, then Stein's identity states:

$$\mathbb{E}[f'(z)] = \mathbb{E}[z f(z)].$$

Denote $\phi(z) := \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$, the one-dimensional standard normal density. The proof follows by integration by parts, using $\phi'(z) = -z\phi(z)$, together with the vanishing boundary term. To see this precisely, we compute

$$\mathbb{E}[f'(z)] = \int_{-\infty}^{\infty} f'(z)\phi(z)\, \mathrm{d}z.$$

By integration by parts, with $u = \phi(z)$ and $\mathrm{d}v = f'(z)\, \mathrm{d}z$, we obtain

$$\int f'(z)\phi(z)\, \mathrm{d}z = \Big[f(z)\phi(z)\Big]_{-\infty}^{\infty} - \int f(z)\phi'(z)\, \mathrm{d}z.$$

Since $\phi'(z) = -z\phi(z)$ and $f(z)\phi(z) \to 0$ as $|z| \to \infty$ (decay condition), the boundary term vanishes and we have

$$\mathbb{E}[f'(z)] = \int f(z)z\phi(z)\,\mathrm{d}z = \mathbb{E}[zf(z)].$$

This completes the derivation and proves the one-dimensional case of Stein's identity.

Multivariate, Standard Normal Case. If $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$ and $g : \mathbb{R}^D \to \mathbb{R}$, then Stein's identity is

$$\mathbb{E}[\nabla g(\mathbf{z})] = \mathbb{E}[\mathbf{z}g(\mathbf{z})].$$

Equivalently, for $\mathbf{u} : \mathbb{R}^D \to \mathbb{R}^D$,

$$\mathbb{E}[\nabla_{\mathbf{z}} \cdot \mathbf{u}(\mathbf{z})] = \mathbb{E}[\mathbf{z}^\top \mathbf{u}(\mathbf{z})]. \tag{D.2.2}$$

Identity Needed for SURE.

Lemma. With $\tilde{\mathbf{x}} = \mathbf{x} + \sigma\mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$, and any vector function $\mathbf{a}$ of suitable regularity,

$$\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \mathbf{a}(\tilde{\mathbf{x}})|\mathbf{x}] = \sigma^2 \mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{a}(\tilde{\mathbf{x}})|\mathbf{x}]. \tag{D.2.3}$$

Proof. With $\tilde{\mathbf{x}} = \mathbf{x} + \sigma\mathbf{z}$,

$$\begin{aligned}\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \mathbf{a}(\tilde{\mathbf{x}}) \,|\, \mathbf{x}] &= \sigma\, \mathbb{E}[\mathbf{z}^\top \mathbf{a}(\mathbf{x} + \sigma\mathbf{z}) \,|\, \mathbf{x}] \overset{\text{Stein}}{=} \sigma\, \mathbb{E}[\nabla_{\mathbf{z}} \cdot \mathbf{a}(\mathbf{x} + \sigma\mathbf{z}) \,|\, \mathbf{x}] \\ &= \sigma^2\, \mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{a}(\tilde{\mathbf{x}}) \,|\, \mathbf{x}],\end{aligned}$$

where the middle step is Equation (D.2.2) and the last step uses $\nabla_{\mathbf{z}} = \sigma\nabla_{\tilde{\mathbf{x}}}$ (chain rule). ∎

Deriving SURE from the Conditional MSE. Let $\mathbf{D} : \mathbb{R}^D \to \mathbb{R}^D$ be a denoiser and define

$$R(\mathbf{D}; \mathbf{x}) := \mathbb{E}\left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|_2^2 | \mathbf{x}\right].$$

Expand around $\tilde{\mathbf{x}}$:

$$
\begin{aligned}
& R(\mathbf{D};\mathbf{x}) \\
&= \mathbb{E}\left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}} + \tilde{\mathbf{x}} - \mathbf{x}\|^2 | \mathbf{x}\right] \\
&= \mathbb{E}\left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 | \mathbf{x}\right] + 2\mathbb{E}\left[(\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}})^\top (\tilde{\mathbf{x}} - \mathbf{x}) | \mathbf{x}\right] + \mathbb{E}\left[\|\tilde{\mathbf{x}} - \mathbf{x}\|^2 | \mathbf{x}\right] \\
&= \mathbb{E}\left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 | \mathbf{x}\right] + 2\Big(\underbrace{\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \mathbf{D}(\tilde{\mathbf{x}}) | \mathbf{x}]}_{\substack{\sigma^2 \mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) | \mathbf{x}] \\ \text{by Equation (D.2.3)}}} - \underbrace{\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \tilde{\mathbf{x}} | \mathbf{x}]}_{\substack{\sigma\, \mathbb{E}[\mathbf{z}]^\top \mathbf{x} + \sigma^2 \mathbb{E}[\mathbf{z}^\top \mathbf{z}] \\ = \sigma^2 D}} \Big) \\
&\qquad + \underbrace{\mathbb{E}[\|\tilde{\mathbf{x}} - \mathbf{x}\|^2 | \mathbf{x}]}_{\substack{\sigma^2 \mathbb{E}[\|\mathbf{z}\|^2] \\ = \sigma^2 D}} \\
&= \mathbb{E}\left[\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 + 2\sigma^2 \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2 | \mathbf{x}\right].
\end{aligned}
$$

Therefore the *observable* surrogate

$$
\mathrm{SURE}(\mathbf{D};\tilde{\mathbf{x}}) := \|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|_2^2 + 2\sigma^2 \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2
$$

satisfies $\mathbb{E}\left[\mathrm{SURE}(\mathbf{D};\tilde{\mathbf{x}}) | \mathbf{x}\right] = R(\mathbf{D};\mathbf{x})$. Minimizing SURE (in expectation or empirically) is thus equivalent to minimizing the true conditional MSE while using only noisy observations.

D.2.5 Theorem 4.2.1: Marginal Alignment via Fokker–Planck

Proof.

Part 1: Fokker-Planck Equation for the Forward SDE. Consider the forward SDE:

$$
\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t).
$$

The diffusion matrix is $\sigma(t) = g(t) I_D$, so $\sigma(t)\sigma(t)^T = g^2(t) I_D$. The Fokker-Planck equation for the marginal density $p_t(\mathbf{x})$ of $\mathbf{x}(t)$ is:

$$
\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})\right] + \frac{1}{2} \sum_{i,j=1}^{D} \frac{\partial^2}{\partial x_i \partial x_j} \left[(g^2(t)\delta_{ij}) p_t(\mathbf{x})\right].
$$

Compute the diffusion term:

$$
\sum_{i,j=1}^{D} \frac{\partial^2}{\partial x_i \partial x_j} \left[g^2(t)\delta_{ij} p_t(\mathbf{x})\right] = \sum_{i=1}^{D} \frac{\partial^2}{\partial x_i^2} \left[g^2(t) p_t(\mathbf{x})\right] = g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).
$$

Thus:

$$
\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})\right] + \frac{1}{2} g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).
$$

Now, rewrite using:

$$\mathbf{v}(\mathbf{x},t) = \mathbf{f}(\mathbf{x},t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Since $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})}$, compute:

$$\nabla_{\mathbf{x}} \cdot [\mathbf{v}(\mathbf{x},t)p_t(\mathbf{x})] = \nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x},t)p_t(\mathbf{x}) - \frac{1}{2}g^2(t)\frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})}p_t(\mathbf{x})\right].$$

The second term is:

$$\nabla_{\mathbf{x}} \cdot \left[-\frac{1}{2}g^2(t)\nabla_{\mathbf{x}} p_t(\mathbf{x})\right] = -\frac{1}{2}g^2(t)\Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Thus:

$$\nabla_{\mathbf{x}} \cdot [\mathbf{v}(\mathbf{x},t)p_t(\mathbf{x})] = \nabla_{\mathbf{x}} \cdot [\mathbf{f}(\mathbf{x},t)p_t(\mathbf{x})] - \frac{1}{2}g^2(t)\Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Therefore:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\mathbf{v}(\mathbf{x},t)p_t(\mathbf{x})],$$

verifying the Fokker-Planck equation in both forms.

Part 2: PF-ODE Marginal Densities. Consider the PF-ODE:

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{v}(\tilde{\mathbf{x}}(t),t), \quad \mathbf{v}(\mathbf{x},t) = \mathbf{f}(\mathbf{x},t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Forward Direction: $\tilde{\mathbf{x}}(0) \sim p_0$. Let $\tilde{\mathbf{x}}(t)$ follow the PF-ODE with $\tilde{\mathbf{x}}(0) \sim p_0$. The flow map $\boldsymbol{\Psi}_t : \mathbb{R}^D \to \mathbb{R}^D$ is defined by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Psi}_t(\mathbf{x}_0) = \mathbf{v}(\boldsymbol{\Psi}_t(\mathbf{x}_0),t), \quad \boldsymbol{\Psi}_0(\mathbf{x}_0) = \mathbf{x}_0.$$

($\boldsymbol{\Psi}_t(\mathbf{x}_0)$ is just $\tilde{\mathbf{x}}(t)$ for the given initial condition $\mathbf{x}_0$.) Since $\tilde{\mathbf{x}}(t) = \boldsymbol{\Psi}_t(\tilde{\mathbf{x}}(0))$, the density $\tilde{p}_t(\mathbf{x})$ of $\tilde{\mathbf{x}}(t)$ satisfies the continuity equation:

$$\partial_t \tilde{p}_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\mathbf{v}(\mathbf{x},t)\tilde{p}_t(\mathbf{x})].$$

Since $\tilde{\mathbf{x}}(0) \sim p_0$, we have $\tilde{p}_0(\mathbf{x}) = p_0(\mathbf{x})$. From Part 1, $p_t(\mathbf{x})$ satisfies:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\mathbf{v}(\mathbf{x},t)p_t(\mathbf{x})].$$

Both $\tilde{p}_t$ and p_t satisfy the same continuity equation with the same initial condition p_0. Assuming sufficient smoothness (e.g., $\mathbf{v} \in C^1$), the solution is unique in some appropriate function space, so $\tilde{p}_t = p_t$. Thus, $\tilde{\mathbf{x}}(t) \sim p_t$ for all $t \in [0,T]$.

Backward Direction: $\tilde{\mathbf{x}}(T) \sim p_T$. Now, let $\tilde{\mathbf{x}}(t)$ follow the PF-ODE backward from $t = T$ to $t = 0$, with $\tilde{\mathbf{x}}(T) \sim p_T$. The ODE is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\tilde{\mathbf{x}}(t) = \mathbf{v}(\tilde{\mathbf{x}}(t),t).$$

Let $s = T - t$, so the backward ODE becomes:

$$\frac{\mathrm{d}}{\mathrm{d}s}\tilde{\mathbf{x}}(T-s) = -\mathbf{v}(\tilde{\mathbf{x}}(T-s), T-s).$$

The density $\tilde{p}_{T-s}(\mathbf{x})$ of $\tilde{\mathbf{x}}(T-s)$ satisfies:

$$\partial_s \tilde{p}_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} \cdot \big[\mathbf{v}(\mathbf{x}, T-s)\tilde{p}_{T-s}(\mathbf{x})\big].$$

Since $\tilde{\mathbf{x}}(T) \sim p_T$, we have $\tilde{p}_T = p_T$. The Fokker-Planck equation for p_t at $t = T - s$ is:

$$\partial_t p_{T-s}(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \big[\mathbf{v}(\mathbf{x}, T-s)p_{T-s}(\mathbf{x})\big].$$

Since $\partial_t = -\partial_s$, we get:

$$\partial_s p_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} \cdot \big[\mathbf{v}(\mathbf{x}, T-s)p_{T-s}(\mathbf{x})\big].$$

Both $\tilde{p}_{T-s}$ and p_{T-s} satisfy the same PDE with the same initial condition at $s = 0$ ($\tilde{p}_T = p_T$). Uniqueness implies $\tilde{p}_{T-s} = p_{T-s}$, so $\tilde{\mathbf{x}}(t) = \tilde{\mathbf{x}}(T-s) \sim p_{T-s} = p_t$, for all $t \in [0, T]$.

Part 3: Reverse-Time SDE Marginal Densities. We verify the marginal distributions using the forward-running reverse clock

$$s := T - t.$$

This is a change of variables so that both SDEs are integrated forward, from 0 to T. The Fokker–Planck equation requires the SDE to run forward in time: when we set $\mathrm{d}\mathbf{w}_t^\top \mathrm{d}\mathbf{w}_t = \mathrm{d}t\,\mathbf{I}$ in its derivation, we assume $\mathrm{d}t > 0$. Define

$$\mathbf{y}(s) := \bar{\mathbf{x}}(T-s), \qquad s \in [0, T].$$

Since the reverse process starts from $\bar{\mathbf{x}}(T) \sim p_T$, we have

$$\mathbf{y}(0) \sim p_T.$$

Under this change of clock, the reverse-time SDE becomes

$$\mathrm{d}\mathbf{y}(s) = \Big[-\mathbf{f}(\mathbf{y}(s), T-s) + g^2(T-s)\nabla_{\mathbf{x}} \log p_{T-s}(\mathbf{y}(s))\Big]\,\mathrm{d}s + g(T-s)\,\mathrm{d}\mathbf{w}^{\mathrm{rev}}(s),$$

where $\mathbf{w}^{\mathrm{rev}}$ is a standard Wiener process under the forward-running clock s.

Let q_s denote the density of $\mathbf{y}(s)$. Its Fokker–Planck equation is

$$\begin{aligned}\partial_s q_s(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \Big(\big[-\mathbf{f}(\mathbf{x}, T-s) + g^2(T-s)\nabla_{\mathbf{x}} \log p_{T-s}(\mathbf{x})\big] q_s(\mathbf{x})\Big) \\ + \tfrac{1}{2} g^2(T-s)\Delta_{\mathbf{x}} q_s(\mathbf{x}).\end{aligned}$$

We now check that

$$q_s(\mathbf{x}) = p_{T-s}(\mathbf{x})$$

satisfies this equation. Using

$$\nabla_{\mathbf{x}} \log p_{T-s}(\mathbf{x})\, p_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} p_{T-s}(\mathbf{x}),$$

the right-hand side becomes

$$\begin{aligned}
&\nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, T-s) p_{T-s}(\mathbf{x})\right] - g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}) \\
&\quad + \tfrac{1}{2} g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}) \\
&= \nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, T-s) p_{T-s}(\mathbf{x})\right] - \tfrac{1}{2} g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}).
\end{aligned}$$

On the other hand, the forward Fokker–Planck equation gives

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})\right] + \tfrac{1}{2} g^2(t)\Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Since $t = T - s$, the chain rule gives

$$\begin{aligned}
\partial_s p_{T-s}(\mathbf{x}) &= -\partial_t p_t(\mathbf{x})\big|_{t=T-s} \\
&= \nabla_{\mathbf{x}} \cdot \left[\mathbf{f}(\mathbf{x}, T-s) p_{T-s}(\mathbf{x})\right] - \tfrac{1}{2} g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}).
\end{aligned}$$

This is exactly the Fokker–Planck equation of the reversed process.

Finally, $q_0 = p_T$, which matches the initial distribution $\mathbf{y}(0) \sim p_T$. Under the regularity assumptions ensuring uniqueness of the Fokker–Planck solution,

$$q_s = p_{T-s}, \qquad s \in [0, T].$$

Returning to the original diffusion clock $t = T - s$, we obtain

$$\bar{\mathbf{x}}(t) \sim p_t.$$

Thus, when the reverse-time SDE is solved backward from $t = T$ to $t = 0$, it follows the same marginal distributions as the forward SDE, but in the reverse order. ■

D.2.6 Proposition 4.3.1: Minimizer of SM and DSM

Proof. To find the minimizer $\mathbf{s}^*$, we first consider a fixed time t and analyze the inner expectation in the objective function:

$$\mathcal{J}(t, \boldsymbol{\phi}) := \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{\mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_0)} \left[\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2 \right].$$

For this expectation to be minimized, we need to find $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$ that minimizes the expected squared error for each $\mathbf{x}_t$. We can rewrite this expectation using the joint distribution of X_0 and X_t:

$$\mathcal{J}(t, \boldsymbol{\phi}) = \iint p_{\text{data}}(\mathbf{x}_0) p_t(\mathbf{x}_t|\mathbf{x}_0) \, \|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)\|_2^2 \, \mathrm{d}\mathbf{x}_0 \, \mathrm{d}\mathbf{x}_t.$$

For each fixed $\mathbf{x}_t$, we need to minimize:

$$\int p(\mathbf{x}_0|X_t = \mathbf{x}_t) p_t(\mathbf{x}_t) \, \|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)\|_2^2 \, \mathrm{d}\mathbf{x}_0.$$

Since $p_t(\mathbf{x}_t)$ is constant with respect to $\mathbf{s}_\phi(\mathbf{x}_t, t)$, this is equivalent to minimizing:

$$\int p(\mathbf{x}_0|X_t = \mathbf{x}_t) \left\|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2 \mathrm{d}\mathbf{x}_0$$

This is minimized when $\mathbf{s}_\phi(\mathbf{x}_t, t)$ equals the conditional expectation:

$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{X_0 \sim p(X_0|X_t=\mathbf{x}_t)} \left[\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|X_0)\right].$$

Now we need to prove that this equals $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$. By Bayes' rule and the definition of marginal probability:

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0.$$

Taking the logarithm and then the gradient with respect to $\mathbf{x}_t$:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} = \frac{\nabla_{\mathbf{x}_t} \int p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0}{\int p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0}.$$

Under suitable regularity conditions, we can exchange the gradient and integral:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0}{\int p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0}.$$

Using

$$\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t|\mathbf{x}_0) = p_t(\mathbf{x}_t|\mathbf{x}_0) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0),$$

and Bayes' rule, we obtain

$$\begin{aligned}
\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) &= \frac{\int p_t(\mathbf{x}_t|\mathbf{x}_0)\, \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)\ \mathrm{d}\mathbf{x}_0}{\int p_t(\mathbf{x}_t|\mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)\ \mathrm{d}\mathbf{x}_0} \\
&= \frac{\int p(\mathbf{x}_0|\mathbf{x}_t)\, p_t(\mathbf{x}_t)\, \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\ \mathrm{d}\mathbf{x}_0}{\int p(\mathbf{x}_0|\mathbf{x}_t)\, p_t(\mathbf{x}_t)\ \mathrm{d}\mathbf{x}_0} \\
&= \int p(\mathbf{x}_0|\mathbf{x}_t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\, \mathrm{d}\mathbf{x}_0 \\
&= \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} \left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\right] = \mathbf{s}^*(\mathbf{x}_t, t),
\end{aligned}$$

which completes the proof. ■

D.2.7 Closed-Form Score Function of a Gaussian

For future reference, we summarize the formula for the score of a general multivariate normal distribution as the following lemma:

Lemma D.2.1: Score of Gaussian

Let $\mathbf{x} \in \mathbb{R}^D$ and consider the multivariate normal distribution

$$p(\tilde{\mathbf{x}}|\mathbf{x}) := \mathcal{N}(\tilde{\mathbf{x}}; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean and $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ is an invertible covariance matrix. Its score function is

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}). \tag{D.2.4}$$

Proof for Lemma.

The density function of $p(\tilde{\mathbf{x}}|\mathbf{x})$ is given by:

$$p(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu})\right).$$

To compute the score function, we first take the log of $p(\tilde{\mathbf{x}}|\mathbf{x})$:

$$\log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}).$$

Now, we compute the gradient of $\log p(\tilde{\mathbf{x}}|\mathbf{x})$ with respect to $\tilde{\mathbf{x}}$:

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\frac{1}{2}\nabla_{\tilde{\mathbf{x}}}\left((\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu})\right).$$

Using the chain rule, we get:

$$\nabla_{\tilde{\mathbf{x}}}\left((\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu})\right) = 2\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}).$$

Thus, the score function is:

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}). \tag{D.2.5}$$

D.3 Flow-Based Perspective

D.3.1 Lemma 5.1.1: Instantaneous Change of Variables

Proof. **Approach 1: Change-of-Variables Formula.** We denote $p(\mathbf{x}(t), t)$ by $p_t(\mathbf{x}_t)$. Starting from the ODE discretization

$$\mathbf{z}_{t+\Delta t} = \mathbf{z}_t + \Delta t \mathbf{F}(\mathbf{z}_t, t),$$

the change-of-variables formula for normalizing flows (Equation (5.1.1)) gives

$$\begin{aligned}
\log p_{t+\Delta t}(\mathbf{z}_{t+\Delta t}) &= \log p_t(\mathbf{z}_t) - \log \left| \det \left(\mathbf{I} + \Delta t \nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t) \right) \right| \\
&= \log p_t(\mathbf{z}_t) - \operatorname{Tr} \Big(\log(\mathbf{I} + \Delta t \nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t)) \Big) \\
&= \log p_t(\mathbf{z}_t) - \Delta t \operatorname{Tr} \left(\nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t) \right) + \mathcal{O}(\Delta t^2),
\end{aligned}$$

where we used $\log\det = \operatorname{Tr}\log$ and the expansion for small Δt. The expansion keeps the first term of the power series $\log(\mathbf{I} + \mathbf{A}) = \mathbf{A} - \frac{1}{2}\mathbf{A}^2 + \cdots$ with $\mathbf{A} = \Delta t \nabla_{\mathbf{z}} \mathbf{F}$. Taking the limit $\Delta t \to 0$ yields the continuous-time differential equation for the log-density. Indeed, the same trick is applied in Equation (5.1.6).

Approach 2: Continuity Equation. We can also leverage the continuity equation, which essentially serves as the change-of-variables formula:

$$\partial_t p(\mathbf{z}, t) = -\nabla_{\mathbf{z}} \cdot \left(\mathbf{F}(\mathbf{z}, t) p(\mathbf{z}, t) \right).$$

Expanding the divergence,

$$\partial_t p = -\left((\nabla_{\mathbf{z}} \cdot \mathbf{F}) p + \mathbf{F} \cdot \nabla_{\mathbf{z}} p \right).$$

Along trajectories $\mathbf{z}(t)$ satisfying $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{F}(\mathbf{z}(t), t)$, the total time derivative is

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} p(\mathbf{z}(t), t) &= \nabla_{\mathbf{z}} p \cdot \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} + \partial_t p \\
&= \nabla_{\mathbf{z}} p \cdot \mathbf{F} - \left((\nabla_{\mathbf{z}} \cdot \mathbf{F}) p + \mathbf{F} \cdot \nabla_{\mathbf{z}} p \right) \\
&= -(\nabla_{\mathbf{z}} \cdot \mathbf{F}) p.
\end{aligned}$$

Dividing by $p(\mathbf{z}(t), t)$, we conclude

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p(\mathbf{z}(t), t) = -\nabla_{\mathbf{z}} \cdot \mathbf{F}(\mathbf{z}(t), t).$$

■

D.3.2 Theorem 5.2.2: Mass Conservation Criterion

Some Prerequisites: Flow Map and Flow-Induced Density. For any initial position $\mathbf{x}_0 \in \mathbb{R}^D$, the flow map $\boldsymbol{\Psi}_t : \mathbb{R}^D \to \mathbb{R}^D$ is the unique solution of the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t} \boldsymbol{\Psi}_t(\mathbf{x}_0) = \mathbf{v}_t(\boldsymbol{\Psi}_t(\mathbf{x}_0)), \quad \boldsymbol{\Psi}_0(\mathbf{x}_0) = \mathbf{x}_0.$$

Under our regularity assumptions, $\mathbf{\Psi}_t$ is continuously differentiable in both t and $\mathbf{x}_0$.

The flow-induced density p_t^{fwd} is the pushforward of the initial density p_0 by $\mathbf{\Psi}_t$:

$$p_t^{\text{fwd}}(\mathbf{x}) = p_0(\mathbf{\Psi}_t^{-1}(\mathbf{x}))\left|\det(\nabla \mathbf{\Psi}_t^{-1}(\mathbf{x}))\right|.$$

This gives the density at $\mathbf{x}$ and time t of particles that started from $p_0 = p_{\text{data}}$ and evolved under the velocity field $\mathbf{v}_t$.

Informal Proof: **Sufficient Condition: $p_t^{\text{fwd}} = p_t \Rightarrow$ Continuity Equation.** In Section B.1.2 we obtained a *strong solution* of the continuity equation by taking the continuous time limit of the discrete change of variable formula. In that approach, the density p_t is assumed smooth enough that all derivatives exist classically and the PDE holds pointwise. Here, we offer a complementary derivation in the *weak sense*: the continuity equation is imposed only after integrating against arbitrary smooth test functions, which relaxes the regularity requirements on both p_t and the velocity field $\mathbf{v}_t$. This weak formulation is not only more rigorous since it accommodates less regular solutions but is also the standard framework in PDE theory and numerical analysis.

For any smooth test function $\varphi(\mathbf{x})$ with compact support, the pushforward property gives:

$$\begin{aligned}\int p_t^{\text{fwd}}(\mathbf{x})\varphi(\mathbf{x})\,\mathrm{d}\mathbf{x} &= \int p_0(\mathbf{\Psi}_t^{-1}(\mathbf{x}))\left|\det\left(\nabla \mathbf{\Psi}_t^{-1}(\mathbf{x})\right)\right|\varphi(\mathbf{x})\,\mathrm{d}\mathbf{x} \\ &= \int p_0(\mathbf{y})\varphi(\mathbf{\Psi}_t(\mathbf{y}))\,\mathrm{d}\mathbf{y},\end{aligned}$$

where the second equality follows from the change of variables $\mathbf{x} = \mathbf{\Psi}_t(\mathbf{y})$, with $\mathrm{d}\mathbf{y} = \left|\det\left(\nabla \mathbf{\Psi}_t^{-1}(\mathbf{x})\right)\right|\mathrm{d}\mathbf{x}$.

Differentiate both sides with respect to t:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int p_t^{\text{fwd}}(\mathbf{x})\varphi(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{\mathrm{d}}{\mathrm{d}t}\int p_0(\mathbf{y})\varphi(\mathbf{\Psi}_t(\mathbf{y}))\,\mathrm{d}\mathbf{y}.$$

The left-hand side is:

$$\int \frac{\partial p_t^{\text{fwd}}}{\partial t}(\mathbf{x})\varphi(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

On the right-hand side:

$$\int p_0(\mathbf{y})\nabla\varphi(\mathbf{\Psi}_t(\mathbf{y}))\cdot\mathbf{v}_t(\mathbf{\Psi}_t(\mathbf{y}))\,\mathrm{d}\mathbf{y},$$

since $\frac{\partial \mathbf{\Psi}_t}{\partial t}(\mathbf{y}) = \mathbf{v}_t(\mathbf{\Psi}_t(\mathbf{y}))$. Change variables to $\mathbf{x} = \mathbf{\Psi}_t(\mathbf{y})$, so

$$\mathrm{d}\mathbf{y} = \left|\det\left(\nabla \mathbf{\Psi}_t^{-1}(\mathbf{x})\right)\right|\mathrm{d}\mathbf{x},$$

and

$$p_0(\mathbf{y}) = p_t^{\text{fwd}}(\mathbf{x}) \left|\det\left(\nabla \boldsymbol{\Psi}_t(\mathbf{y})\right)\right| = \frac{p_t^{\text{fwd}}(\mathbf{x})}{\left|\det\left(\nabla \boldsymbol{\Psi}_t^{-1}(\mathbf{x})\right)\right|}.$$

Thus, the right-hand side becomes:

$$\int p_t^{\text{fwd}}(\mathbf{x}) \nabla \varphi(\mathbf{x}) \cdot \mathbf{v}_t(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

Apply integration by parts, using the compact support of φ:

$$\int p_t^{\text{fwd}}(\mathbf{x}) \nabla \varphi(\mathbf{x}) \cdot \mathbf{v}_t(\mathbf{x})\, \mathrm{d}\mathbf{x} = -\int \varphi(\mathbf{x}) \nabla \cdot (p_t^{\text{fwd}}(\mathbf{x}) \mathbf{v}_t(\mathbf{x}))\, \mathrm{d}\mathbf{x}.$$

Equating both sides:

$$\int \left[\frac{\partial p_t^{\text{fwd}}}{\partial t}(\mathbf{x}) + \nabla \cdot (p_t^{\text{fwd}}(\mathbf{x}) \mathbf{v}_t(\mathbf{x}))\right] \varphi(\mathbf{x})\, \mathrm{d}\mathbf{x} = 0.$$

Since φ is arbitrary, we conclude:

$$\frac{\partial p_t^{\text{fwd}}}{\partial t} + \nabla \cdot (p_t^{\text{fwd}} \mathbf{v}_t) = 0.$$

Given $p_t^{\text{fwd}} = p_t$, this implies:

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t \mathbf{v}_t) = 0.$$

Necessary Condition: Continuity Equation $\Rightarrow p_t^{\text{fwd}} = p_t$. Suppose p_t satisfies the continuity equation:

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t \mathbf{v}_t) = 0,$$

with the initial condition $p_0(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$. We know p_t^{fwd} satisfies the same continuity equation, as shown above, and:

$$p_0^{\text{fwd}}(\mathbf{x}) = p_0(\boldsymbol{\Psi}_0^{-1}(\mathbf{x})) \left|\det\left(\nabla \boldsymbol{\Psi}_0^{-1}(\mathbf{x})\right)\right| = p_0(\mathbf{x}) \left|\det \mathbf{I}\right| = p_0(\mathbf{x}),$$

since $\boldsymbol{\Psi}_0(\mathbf{x}) = \mathbf{x}$. Thus, both densities share the same initial condition $p_0 = p_{\text{data}}$.

The continuity equation can be rewritten as:

$$\frac{\partial p}{\partial t} + \mathbf{v}_t \cdot \nabla p + p \nabla \cdot \mathbf{v}_t = 0.$$

This is a first-order linear PDE. Assuming $\mathbf{v}_t$ is continuously differentiable and globally Lipschitz, and p_t is sufficiently smooth, the method of characteristics guarantees a unique solution in the space of smooth functions. Since p_t and p_t^{fwd} satisfy the same PDE and initial condition, we conclude:

$$p_t(\mathbf{x}) = p_t^{\text{fwd}}(\mathbf{x})$$

for all $t \in [0, 1]$ and $\mathbf{x} \in \mathbb{R}^D$.

This completes the proof of the equivalence. ■

D.4 Theoretical Supplement: A Unified and Systematic View on Diffusion Models

D.4.1 Proposition 6.3.1: Equivalence of Parametrizations

Proof: As in Theorem 4.3.1 on the DSM loss, the global optimum of the matching loss

$$\mathbb{E}_t \left[\omega(t) \, \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\| \cdot \|_2^2 \right] \right]$$

is attained when the inner expectation

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\| \cdot \|_2^2 \right]$$

is minimized for each fixed t. Since this is a standard mean squared error problem, the minimizer is unique. From denoising score matching (Vincent, 2011), Theorem 4.3.1 shows the optimal score function satisfies

$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} \left[\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t|\mathbf{x}_0) \right] = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

Using the identity $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{1}{\sigma_t}\boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we obtain

$$\mathbf{s}^*(\mathbf{x}_t, t) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}^*(\mathbf{x}_t, t),$$

where $\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\boldsymbol{\epsilon}|\mathbf{x}_t]$ is the optimal $\boldsymbol{\epsilon}$-prediction for $\mathcal{L}_{\text{noise}}(\boldsymbol{\phi})$. For the $\mathbf{x}$-prediction loss $\mathcal{L}_{\text{clean}}$, the optimal estimator is

$$\mathbf{x}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t],$$

which, by Tweedie's formula, relates to the score via

$$\alpha_t \, \mathbf{x}^*(\mathbf{x}_t, t) = \mathbf{x}_t + \sigma_t^2 \, \mathbf{s}^*(\mathbf{x}_t, t).$$

For the $\mathbf{v}$-prediction loss $\mathcal{L}_{\text{velocity}}$, the optimal estimator is

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\alpha_t' \mathbf{x}_0 + \sigma_t' \boldsymbol{\epsilon}|\mathbf{x}_t] = \alpha_t' \mathbf{x}^* + \sigma_t' \boldsymbol{\epsilon}^*.$$

Substituting the expressions for $\mathbf{x}^*$ and $\boldsymbol{\epsilon}^*$ in terms of $\mathbf{s}^*$ yields

$$\mathbf{v}^*(\mathbf{x}_t, t) = \frac{\alpha_t'}{\alpha_t} \mathbf{x}_t + \left(\frac{\alpha_t'}{\alpha_t} \sigma_t^2 - \sigma_t \sigma_t' \right) \mathbf{s}^*(\mathbf{x}_t, t) = f(t)\mathbf{x}_t - \frac{1}{2} g^2(t) \mathbf{s}^*(\mathbf{x}_t, t),$$

which completes the derivation.

■

D.5 Theoretical Supplement: Learning Fast Diffusion-Based Generators

D.5.1 Knowledge Distillation Loss as an Instance of the General Framework Equation (10.1.4)

We begin with the oracle KD loss

$$\mathcal{L}_{\mathrm{KD}}^{\mathrm{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_T \sim p_T} \left\| \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0) - \boldsymbol{\Psi}_{T\to 0}(\mathbf{x}_T) \right\|_2^2,$$

where $p_T = p_{\mathrm{prior}}$. For the deterministic ODE flow map $\boldsymbol{\Psi}$ (which satisfies the semigroup property and is bijective along trajectories), the marginals satisfy the pushforward identities

$$p_t = \boldsymbol{\Psi}_{0\to t} \sharp\, p_{\mathrm{data}} = \boldsymbol{\Psi}_{T\to t} \sharp\, p_{\mathrm{prior}}.$$

Hence,

$$\boldsymbol{\Psi}_{s\to T} \sharp\, p_s = p_T \quad \text{and} \quad \boldsymbol{\Psi}_{T\to 0} \circ \boldsymbol{\Psi}_{s\to T} = \boldsymbol{\Psi}_{s\to 0}.$$

Changing variables $\mathbf{x}_T = \boldsymbol{\Psi}_{s\to T}(\mathbf{x}_s)$ with $\mathbf{x}_s \sim p_s$ gives

$$\mathcal{L}_{\mathrm{KD}}^{\mathrm{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_s \sim p_s} \left\| \mathbf{G}_{\boldsymbol{\theta}}\big(\boldsymbol{\Psi}_{s\to T}(\mathbf{x}_s), T, 0\big) - \boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s) \right\|_2^2.$$

Define the pulled-back student $\widetilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0) := \mathbf{G}_{\boldsymbol{\theta}}(\boldsymbol{\Psi}_{s\to T}(\mathbf{x}_s), T, 0)$ to express the same loss in the unified flow-map form (at $t = 0$):

$$\mathcal{L}_{\mathrm{KD}}^{\mathrm{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_s \sim p_s} \left\| \widetilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0) - \boldsymbol{\Psi}_{s\to 0}(\mathbf{x}_s) \right\|_2^2.$$

This derivation relies on change of variables through the oracle flow and the semigroup property.

■

Transport Interpretation. Proposition 10.2.1 also admits a distributional interpretation. For

$$\hat{\mathbf{x}}_t = \alpha_t \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) + \sigma_t \boldsymbol{\epsilon},$$

an infinitesimal parameter perturbation induces the data-space displacement $(\partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t)\, \mathrm{d}\boldsymbol{\theta}$. Accordingly, the VSD gradient can be written as

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{VSD}} = \mathbb{E}\Big[\omega(t) (\partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t)^\top \Big(\nabla_{\mathbf{x}} \log p_t^{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t) \Big) \Big].$$

Thus, the score mismatch provides the data-space gradient signal, while the generator Jacobian maps this signal back to parameter space; gradient descent then updates the parameters in the opposite direction. This is the parametric counterpart of the distribution-level transport view in Section B.3.

D.5.2 Theorem 11.2.1: CM Equals CT up to $\mathcal{O}(\Delta s^2)$

Proof: **Step 1: DDIM Update (with Oracle Score) Is the Conditional Mean.**

$$\begin{aligned}
\hat{\mathbf{x}}_{s'} &:= \frac{\alpha_{s'}}{\alpha_s}\mathbf{x}_s + \sigma_s^2\left(\frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s}\right)\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) \\
&= \frac{\alpha_{s'}}{\alpha_s}(\mathbf{x}_s + \sigma_s^2\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s)) - \sigma_{s'}\sigma_s\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s) \\
&= \alpha_{s'}\mathbb{E}[\mathbf{x}_0|\mathbf{x}_s] + \sigma_{s'}\Big(\frac{\mathbf{x}_s - \alpha_s\mathbb{E}[\mathbf{x}_0|\mathbf{x}_s]}{\sigma_s}\Big) \\
&= \alpha_{s'}\mathbb{E}[\mathbf{x}_0|\mathbf{x}_s] + \sigma_{s'}\mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_s] \\
&= \mathbb{E}[\alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon}|\mathbf{x}_s] \\
&= \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s].
\end{aligned}$$

Here we use Tweedie's formula

$$\mathbb{E}[\mathbf{x}_0|\mathbf{x}_s] = \frac{\mathbf{x}_s + \sigma_s^2\nabla_{\mathbf{x}_s}\log p_s(\mathbf{x}_s)}{\alpha_s},$$

together with the shared-noise coupling

$$\mathbf{x}_s = \alpha_s\mathbf{x}_0 + \sigma_s\boldsymbol{\epsilon}, \qquad \mathbf{x}_{s'} = \alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon} \quad (s' = s - \Delta s),$$

in the third and fourth equalities. We also note that, more precisely, $\hat{\mathbf{x}}_{s'} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s]$ should be read as $\hat{\mathbf{x}}_{s'} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s, s]$, since $s' = s - \Delta s$ is deterministically fixed once s is sampled.

Step 2: Expand CT Around the Conditional Mean $\hat{\mathbf{x}}_{s'}$.

Recall that s is sampled first and we set $s' := s - \Delta s$ deterministically.

$$\begin{aligned}
\mathcal{L}_{\mathrm{CT}} &= \mathbb{E}_{s,\mathbf{x}_s}\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s}\Big[w(s)d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s'}, s')\big)\Big] \\
&\overset{(1)}{=} \mathbb{E}_{s,\mathbf{x}_s}\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s}\Big[w(s)d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\big) \\
&\qquad + w(s)\partial_2 d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\big)\big[\partial_1\mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')(\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})\big] \\
&\qquad + w(s)\mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2)\Big] \\
&\overset{(2)}{=} \mathbb{E}_{s,\mathbf{x}_s}\Big[w(s)d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\big)\Big] \\
&\quad + \mathbb{E}_{s,\mathbf{x}_s}\Big[w(s)\partial_2 d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\big)\big[\partial_1\mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s}(\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})\big]\Big] \\
&\quad + \mathbb{E}_{s,\mathbf{x}_s}\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s}\Big[w(s)\mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2)\Big] \\
&\overset{(3)}{=} \mathbb{E}_{s,\mathbf{x}_s}\Big[w(s)d\big(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{s'}, s')\big)\Big] + \mathbb{E}_{s,\mathbf{x}_s}\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s}\Big[w(s)\mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2)\Big] \\
&= \mathcal{L}_{\mathrm{CM}} + \mathcal{O}(\mathbb{E}\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \\
&\overset{(4)}{=} \mathcal{L}_{\mathrm{CM}} + \mathcal{O}(\Delta s^2).
\end{aligned}$$

Here, (1) applies a second-order Taylor expansion of

$$h(\mathbf{x}') := d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}', s'))$$

around $\hat{\mathbf{x}}_{s'}$ in its second argument. (2) uses the *tower property* (a.k.a. the law of total expectation):

$$\mathbb{E}_{s,\mathbf{x}_s}\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s,s}[\cdot] = \mathbb{E}_{s,\mathbf{x}_s}[\cdot],$$

and notes that inside $\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s,s}$, the terms

$$w(s),\ \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s),\ \hat{\mathbf{x}}_{s'},\ s'$$

do not depend on the random variable $\mathbf{x}_{s'}$ (since $s' = s - \Delta s$ and $\hat{\mathbf{x}}_{s'} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s, s]$ are fixed once $(s, \mathbf{x}_s)$ is fixed). Hence the only $\mathbf{x}_{s'}$-dependence is through $(\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})$, and the inner expectations reduce to $\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s,s}(\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})$ and $\mathbb{E}_{\mathbf{x}_{s'}|\mathbf{x}_s,s}\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2$. (3) uses

$$\mathbb{E}[\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}|\mathbf{x}_s, s] = \mathbf{0} \quad \text{because} \quad \hat{\mathbf{x}}_{s'} = \mathbb{E}[\mathbf{x}_{s'}|\mathbf{x}_s, s].$$

For (4), note that

$$\mathbb{E}\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2 = \mathbb{E}\,\mathrm{Var}(\mathbf{x}_{s'}|\mathbf{x}_s, s) \le \mathbb{E}\|\mathbf{x}_{s'} - \mathbf{x}_s\|^2.$$

Moreover,

$$\mathbf{x}_{s'} - \mathbf{x}_s = (\alpha_{s'} - \alpha_s)\mathbf{x}_0 + (\sigma_{s'} - \sigma_s)\boldsymbol{\epsilon},$$

and for a smooth scheduler, $\alpha_{s'} - \alpha_s = \mathcal{O}(\Delta s)$ and $\sigma_{s'} - \sigma_s = \mathcal{O}(\Delta s)$, hence

$$\mathbb{E}\|\mathbf{x}_{s'} - \mathbf{x}_s\|^2 = \mathcal{O}(\Delta s^2),$$

which yields

$$\mathbb{E}\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2 = \mathcal{O}(\Delta s^2).$$

■

D.5.3 Proposition 11.3.1: Continuous-Time Limit of the CT Gradient

Proof: We simplify the notation by working with Equation (11.3.2) in the form

$$\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}\left[\omega(s)\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s),\ s - \Delta s)\|_2^2\right].$$

For notational simplicity, write

$$\tilde{\mathbf{x}}_{s-\Delta s} := \boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s).$$

Define

$$\delta\mathbf{f} := \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s).$$

Then

$$\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s - \Delta s)\|_2^2 = \left\|\delta\mathbf{f} + \left(\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s - \Delta s)\right)\right\|_2^2.$$

Applying a Taylor expansion at $(\mathbf{x}_s, s)$ gives

$$\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s-\Delta s) = \frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\,\Delta s + \mathcal{O}(|\Delta s|^2),$$

where we use the first-order expansion of the oracle flow map,

$$\mathbf{x}_s - \boldsymbol{\Psi}_{s\to s-\Delta s}(\mathbf{x}_s) = \mathbf{v}^*(\mathbf{x}_s, s)\,\Delta s + \mathcal{O}(|\Delta s|^2),$$

together with the total differentiation identity

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) = \partial_s \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + \big(\partial_{\mathbf{x}}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\big)\,\mathbf{v}^*(\mathbf{x}_s, s).$$

Therefore,

$$\big\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s-\Delta s)\big\|_2^2 = \|\delta\mathbf{f}\|_2^2 + 2\,\delta\mathbf{f}^\top \frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\,\Delta s + \mathcal{O}(|\Delta s|^2).$$

Now differentiate with respect to $\boldsymbol{\theta}$. Since $\boldsymbol{\theta}^-$ is treated as fixed with respect to $\nabla_{\boldsymbol{\theta}}$ (i.e., no gradient is taken through the stop-gradient target), there is no gradient through $\mathbf{f}_{\boldsymbol{\theta}^-}$. Hence

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \nabla_{\boldsymbol{\theta}}\mathbb{E}\Big[\omega(s)\|\delta\mathbf{f}\|_2^2\Big] + 2\,\mathbb{E}\Big[\omega(s)\,\nabla_{\boldsymbol{\theta}}\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s)^\top \frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\Big]\,\Delta s + \mathcal{O}(|\Delta s|^2).$$

At the comparison point where the target network is the stop-gradient copy of the online network, i.e., $\boldsymbol{\theta} = \boldsymbol{\theta}^-$, we have

$$\delta\mathbf{f} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) = 0,$$

and therefore

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}\Big[\omega(s)\|\delta\mathbf{f}\|_2^2\Big] = 0.$$

Thus,

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} = 2\,\mathbb{E}\Big[\omega(s)\,\nabla_{\boldsymbol{\theta}}\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s)^\top \frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\Big]\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-}\Delta s + \mathcal{O}(|\Delta s|^2).$$

Dividing by Δs and taking the limit yields

$$\lim_{\Delta s\to 0}\frac{1}{\Delta s}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} = \nabla_{\boldsymbol{\theta}}\mathbb{E}\Big[2\,\omega(s)\,\mathbf{f}_{\boldsymbol{\theta}}^\top(\mathbf{x}_s, s)\,\frac{\mathrm{d}}{\mathrm{d}s}\mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)\Big]\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-}.$$

This proves the identity at $\boldsymbol{\theta} = \boldsymbol{\theta}^-$.

■

D.6 (Optional) Elucidating Diffusion Model (EDM)

We introduce specific criteria for neural network parameterization design in the $\mathbf{x}$-prediction model, as proposed in Elucidating Diffusion Models (EDM) (Karras *et al.*, 2022). EDM provides a simple recipe that makes the training process easier to optimize and more reliable. The $\mathbf{x}$-prediction model is expressed as a time-dependent skip connection combined with a scaled residual (Equation (D.6.1)). The central idea is to normalize both the inputs and the regression targets to unit variance at all times, and to adjust the skip path so that residual errors are not amplified as time evolves. This recipe has become a widely adopted default in diffusion model implementations and extends naturally to flow map learning, especially the family of Consistency Models.

D.6.1 Criteria for Neural Network $\mathbf{x}_\phi$ Design

EDM considers a parametrization of the $\mathbf{x}$-prediction model, denoted with a slight abuse of notation as $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}, t)$[1], in the following form:

$$\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}, t) := c_{\text{skip}}(t)\mathbf{x} + c_{\text{out}}(t)\mathbf{F}_{\boldsymbol{\phi}}\left(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)\right). \tag{D.6.1}$$

Here, $c_{\text{skip}}(t)$, $c_{\text{out}}(t)$, $c_{\text{in}}(t)$, and $c_{\text{noise}}(t)$ are time-dependent functions. They are chosen to enhance stability and performance during training, based on practical considerations that will be introduced shortly.

Plugging this in Equation (6.2.5), then the objective function becomes after straightforward algebraic manipulation[2]:

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\omega(t)c_{\text{out}}^2(t)\left\|\mathbf{F}_{\boldsymbol{\phi}}\left(c_{\text{in}}(t)\mathbf{x}_t, c_{\text{noise}}(t)\right) - \mathbf{x}_{\text{tgt}}(t)\right\|_2^2\right]. \tag{D.6.2}$$

Here, the regression target $\mathbf{x}_{\text{tgt}}(t)$ is obtained as:

$$\mathbf{x}_{\text{tgt}}(t) = \frac{\left(1 - c_{\text{skip}}(t)\alpha_t\right)\mathbf{x}_0 - c_{\text{skip}}(t)\sigma_t\boldsymbol{\epsilon}}{c_{\text{out}}(t)}.$$

[1] As discussed, all four prediction types are equivalent and can be reduced to the $\mathbf{x}$-prediction case. EDM adopts this formulation, which is both well-studied and naturally aligned with the goal of generating clean samples of flow map models.

[2] We start from the $\mathbf{x}$-prediction diffusion loss by substituting the parameterization given in Equation (D.6.1):

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\omega(t)\left\|\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) - \mathbf{x}_0\right\|_2^2\right] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\omega(t)\left\|c_{\text{skip}}(t)\underbrace{\left(\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}\right)}_{\mathbf{x}_t} + c_{\text{out}}(t)\mathbf{F}_{\boldsymbol{\phi}}\left(c_{\text{in}}(t)\mathbf{x}_t, c_{\text{noise}}(t)\right) - \mathbf{x}_0\right\|_2^2\right] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\omega(t)c_{\text{out}}^2(t)\left\|\mathbf{F}_{\boldsymbol{\phi}}\left(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)\right) - \underbrace{\left(\frac{\left(1 - c_{\text{skip}}(t)\alpha_t\right)\mathbf{x}_0 - c_{\text{skip}}(t)\sigma_t\boldsymbol{\epsilon}}{c_{\text{out}}(t)}\right)}_{\mathbf{x}_{\text{tgt}}(t)}\right\|_2^2\right] \\
&= \text{Equation (D.6.2)}.
\end{aligned}$$

By incorporating the standard deviation of p_{data}, denoted as σ_{d}, EDM proposes the following design criterion for network parameterization, which may be described as the *unit variance criterion*.

Unit Variance of Input.

$$\begin{aligned} &\text{Var}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[c_{\text{in}}(t)\mathbf{x}_t\right] = 1 \\ \iff &c_{\text{in}}^2(t)\text{Var}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}\right] = 1 \\ \iff &c_{\text{in}}(t) = \frac{1}{\sqrt{\sigma_{\text{d}}^2\alpha_t^2 + \sigma_t^2}}, \end{aligned}$$

taking the positive root.

Unit Variance of Training Target.

$$\begin{aligned} &\text{Var}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\mathbf{x}_{\text{tgt}}(t)\right] = 1 \\ \iff &c_{\text{out}}^2(t) = \left(1 - c_{\text{skip}}(t)\alpha_t\right)^2\sigma_{\text{d}}^2 + c_{\text{skip}}^2(t)\sigma_t^2, \end{aligned} \tag{D.6.3}$$

with centered data ($\mathbb{E}[\mathbf{x}_0] = \mathbf{0}$).

Minimize the Error Amplification from $\mathbf{F}_{\boldsymbol{\phi}}$ to $\mathbf{x}_{\boldsymbol{\phi}}$. EDM aims to mitigate the amplification of the network's learning error from $\mathbf{F}_{\boldsymbol{\phi}}$ to $\mathbf{x}_{\boldsymbol{\phi}}$. This is achieved by selecting c_{skip} to minimize c_{out}:

$$c_{\text{skip}}^* \in \arg\min_{c_{\text{skip}}} c_{\text{out}}^2.$$

Using the standard approach of solving $\frac{\partial c_{\text{out}}}{\partial c_{\text{skip}}} = 0$ for the critical point c_{skip}^*, we obtain

$$c_{\text{skip}}^*(t) = \frac{\alpha_t\sigma_{\text{d}}^2}{\alpha_t^2\sigma_{\text{d}}^2 + \sigma_t^2}.$$

Substituting this into Equation (D.6.3), the optimal value is given by

$$c_{\text{out}}^*(t) = \pm\frac{\sigma_t\sigma_{\text{d}}}{\sqrt{\alpha_t^2\sigma_{\text{d}}^2 + \sigma_t^2}}.$$

By convention, we use the nonnegative branch for the output scale:

$$c_{\text{out}}^*(t) = \frac{\sigma_t\sigma_{\text{d}}}{\sqrt{\alpha_t^2\sigma_{\text{d}}^2 + \sigma_t^2}} \quad (\geq 0),$$

which ensures $c_{\text{out}}(0) = 0$ and $c_{\text{out}}(t) \to \sigma_{\text{d}}$ as σ_t is sufficiently large, yielding the intuitive limits $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, 0) \approx \mathbf{x}_t$ and $\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}_t, t) \approx \sigma_{\text{d}}\,\mathbf{F}_{\boldsymbol{\phi}}(\cdot)$.

We summarize these coefficients as follows:

Denoting $R_t := \alpha_t^2\sigma_{\mathrm{d}}^2 + \sigma_t^2$, we have the following selections:

$$c_{\mathrm{in}}(t) = \frac{1}{\sqrt{R_t}}, \quad c_{\mathrm{skip}}(t) = \frac{\alpha_t\sigma_{\mathrm{d}}^2}{R_t}, \quad c_{\mathrm{out}}(t) = \frac{\sigma_t\sigma_{\mathrm{d}}}{\sqrt{R_t}}. \tag{D.6.4}$$

With Equation (D.6.4), the regression target $\mathbf{x}_{\mathrm{tgt}}(t)$ simplifies to

$$\mathbf{x}_{\mathrm{tgt}}(t) = \frac{1}{\sigma_{\mathrm{d}}}\frac{\sigma_t\mathbf{x}_0 - \alpha_t\sigma_{\mathrm{d}}^2\boldsymbol{\epsilon}}{\sqrt{R_t}}.$$

Additionally, substituting these expressions into Equation (D.6.1) and Equation (D.6.2) allows us to simplify both the parametrization and the loss function, yielding:

$$\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}, t) = \frac{\alpha_t\sigma_{\mathrm{d}}^2}{R_t}\mathbf{x} + \frac{\sigma_t\sigma_{\mathrm{d}}}{\sqrt{R_t}}\mathbf{F}_{\boldsymbol{\phi}}\left(\frac{1}{\sqrt{R_t}}\mathbf{x}, c_{\mathrm{noise}}(t)\right),$$

and

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\omega(t)\frac{\sigma_t^2}{R_t}\left\|\sigma_{\mathrm{d}}\mathbf{F}_{\boldsymbol{\phi}}\left(\frac{1}{\sqrt{R_t}}\mathbf{x}_t, c_{\mathrm{noise}}(t)\right) - \left(\frac{\sigma_t\mathbf{x}_0 - \alpha_t\sigma_{\mathrm{d}}^2\boldsymbol{\epsilon}}{\sqrt{R_t}}\right)\right\|_2^2\right].$$

With this parameterization and the conditions $\alpha_0 \approx 1$ and $\sigma_0 \approx 0$, we observe that

$$c_{\mathrm{skip}}(0) \approx 1 \quad \text{and} \quad c_{\mathrm{out}}(0) \approx 0.$$

Selection of $c_{\mathrm{noise}}(t)$. It provides a noise-level embedding to $\mathbf{F}_{\boldsymbol{\phi}}$; any one-to-one mapping of the noise level σ_t (e.g., $c_{\mathrm{noise}}(t) = \log\sigma_t$) is suitable.

D.6.2 A Common EDM Special Case: $\alpha_t = 1$, $\sigma_t = t$

We consider the simplified noise schedule used in EDM, where $\alpha_t = 1$ and $\sigma_t = t$, which also appears in the discussion of CTM in Section 11.4. Under this setting, the forward process becomes

$$\mathbf{x}_t = \mathbf{x}_0 + t\boldsymbol{\epsilon}, \quad \text{with } \mathbf{x}_0 \sim p_{\mathrm{data}},\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

corresponding to the perturbation kernel

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_0, t^2\mathbf{I}).$$

The resulting marginal density is

$$p_t(\mathbf{x}) = \int \mathcal{N}(\mathbf{x}; \mathbf{x}_0, t^2\mathbf{I})p_{\mathrm{data}}(\mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0 = [p_{\mathrm{data}} * \mathcal{N}(\mathbf{0}, t^2\mathbf{I})](\mathbf{x}).$$

Thus, the exact terminal marginal is $p_T = p_{\text{data}} * \mathcal{N}(\mathbf{0}, T^2\mathbf{I})$. When T is sufficiently large relative to the scale of the data, this marginal is commonly approximated by $\mathcal{N}(\mathbf{0}, T^2\mathbf{I})$, which is used as the sampling prior:

$$p_{\text{prior}}(\mathbf{x}_T) := \mathcal{N}(\mathbf{x}_T; \mathbf{0}, T^2\mathbf{I}).$$

Under this setup, the PF-ODE based on $\mathbf{x}$-prediction $\mathbf{x}_{\boldsymbol{\phi}^\times}$ (see Equation (6.3.2)) simplifies to

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \frac{\mathbf{x}(t) - \mathbf{x}_{\boldsymbol{\phi}^\times}(\mathbf{x}(t), t)}{t}.$$

Substituting this formulation into Equation (D.6.4), the neural network parameterization coefficients become

$$c_{\text{in}}(t) = \frac{1}{\sqrt{\sigma_{\text{d}}^2 + t^2}}, \quad c_{\text{skip}}(t) = \frac{\sigma_{\text{d}}^2}{\sigma_{\text{d}}^2 + t^2}, \quad c_{\text{out}}(t) = \pm\frac{t\sigma_{\text{d}}}{\sqrt{\sigma_{\text{d}}^2 + t^2}}. \tag{D.6.5}$$

From these expressions, we observe:

- When $t \approx 0$, the noise level is negligible, so $c_{\text{skip}} \approx 1$ and $c_{\text{out}} \approx 0$. In this limit, the skip path dominates and the network essentially passes through its input,
$$\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}, t) \approx \mathbf{x}.$$
- When $t \gg 0$, the input is heavily corrupted by noise, so $c_{\text{skip}} \approx 0$ and $c_{\text{out}} \approx \sigma_{\text{d}}$. In this regime, the skip path vanishes and the model output is determined entirely by the learned residual,
$$\mathbf{x}_{\boldsymbol{\phi}}(\mathbf{x}, t) \approx \sigma_{\text{d}}\mathbf{F}_{\boldsymbol{\phi}}\left(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)\right),$$
meaning that the network $\mathbf{F}_{\boldsymbol{\phi}}$ predicts a scaled proxy of the clean signal from a normalized noisy input; at high noise levels the model output is therefore determined entirely by the learned denoising function.

In short, the parameterization smoothly interpolates from an identity map at small t to a scaled residual predictor on standardized inputs at large t.

E

Miscellaneous Proofs

This appendix collects short, self-contained proofs of standard facts that are used throughout the book but would interrupt the flow of the main derivations.

E.1 Two Perspectives on the Divergence Theorem

The divergence theorem underlies the multivariate integration by parts of Section E.2. Here are two ways to see why it holds. Let $\mathbf{F} = (P, Q, R)$ be a smooth vector field on $\mathbb{R}^3$ and $\mathbf{n}$ the outward unit normal.

(1) Intuition. Consider a tiny box $V = [a, b] \times [c, d] \times [e, f]$ and the net flux perpendicular to the x-axis. On the right face ($x = b$) the outward normal is $+\hat{\mathbf{x}}$, so the flux through it is $\int_e^f \int_c^d P(b, y, z)\, \mathrm{d}y\, \mathrm{d}z$. On the left face ($x = a$) the outward normal is $-\hat{\mathbf{x}}$, so the flux through it is $\int_e^f \int_c^d -P(a, y, z)\, \mathrm{d}y\, \mathrm{d}z$. Hence the net flux perpendicular to the x-axis is

$$\int_e^f \int_c^d \big[P(b, y, z) - P(a, y, z)\big]\, \mathrm{d}y\, \mathrm{d}z \overset{\text{FTC}}{=} \int_e^f \int_c^d \int_a^b \frac{\partial P}{\partial x}\, \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z = \int_V \frac{\partial P}{\partial x}\, \mathrm{d}V.$$

By symmetry, the y- and z-faces contribute $\int_V \partial Q/\partial y\, \mathrm{d}V$ and $\int_V \partial R/\partial z\, \mathrm{d}V$. Summing the three,

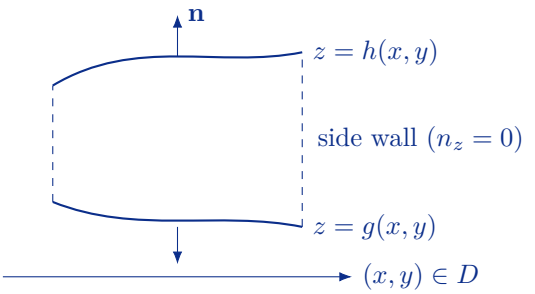
$$\int_{\partial V} \mathbf{F} \cdot \mathbf{n}\, \mathrm{d}S \;=\; \int_V \nabla \cdot \mathbf{F}\, \mathrm{d}V. \qquad \checkmark$$

(2) "Rigorous" Proof. We want to show

$$\int_{\partial V} \mathbf{F} \cdot \mathbf{n}\, \mathrm{d}S \;=\; \int_V \Big(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}\Big)\, \mathrm{d}V.$$

Assume V is a *type-1 region*, i.e., it is sandwiched between two graphs over a planar domain D:

$$V = \big\{(x, y, z) :\ (x, y) \in D,\ g(x, y) \le z \le h(x, y)\big\}.$$



Consider only the R-term:

$$\int_V \frac{\partial R}{\partial z}\,\mathrm{d}V = \int_D \Big[\int_{g(x,y)}^{h(x,y)} \frac{\partial R}{\partial z}\,\mathrm{d}z\Big]\,\mathrm{d}x\,\mathrm{d}y$$
$$= \int_D \big[R(x,y,h(x,y)) - R(x,y,g(x,y))\big]\,\mathrm{d}x\,\mathrm{d}y = \int_{\partial V} R\,n_z\,\mathrm{d}S,$$

where the last equality holds because on the top surface $n_z\,\mathrm{d}S = +\,\mathrm{d}x\,\mathrm{d}y$, on the bottom surface $n_z\,\mathrm{d}S = -\,\mathrm{d}x\,\mathrm{d}y$, and the vertical side walls contribute nothing since $n_z = 0$ there. By symmetry, assuming V is also a type-2 and a type-3 region (so that the same argument applies to P and Q), we conclude

$$\int_V \nabla \cdot \mathbf{F}\,\mathrm{d}V \;=\; \int_{\partial V} \mathbf{F}\cdot\mathbf{n}\,\mathrm{d}S. \qquad \blacksquare$$

E.2 Multivariate Integration by Parts

Multivariate integration by parts is a direct consequence of the divergence theorem.

Lemma (Multivariate Integration by Parts). Let $\phi\colon \mathbb{R}^D \to \mathbb{R}$ be a smooth scalar field and $\mathbf{v}\colon \mathbb{R}^D \to \mathbb{R}^D$ a smooth vector field such that the product $\phi\,\mathbf{v}$ vanishes sufficiently fast at infinity (faster than $1/R^{D-1}$, so that the flux through the sphere of radius R vanishes as $R \to \infty$). Then

$$\int_{\mathbb{R}^D} \nabla_{\mathbf{x}}\phi^{\top}\mathbf{v}\,\mathrm{d}\mathbf{x} = -\int_{\mathbb{R}^D} \phi\,\nabla_{\mathbf{x}}\cdot\mathbf{v}\,\mathrm{d}\mathbf{x}.$$

Proof. For $\mathbf{G} := \phi\,\mathbf{v}$, the product rule gives

$$\nabla_{\mathbf{x}}\cdot\mathbf{G} = \nabla_{\mathbf{x}}\phi^{\top}\mathbf{v} + \phi\,\nabla_{\mathbf{x}}\cdot\mathbf{v}.$$

Integrating both sides over the ball $B_R := \{\mathbf{x}\in\mathbb{R}^D : \|\mathbf{x}\| \le R\}$ and applying the divergence theorem to the left-hand side,

$$\int_{B_R} \nabla_{\mathbf{x}}\phi^{\top}\mathbf{v}\,\mathrm{d}\mathbf{x} + \int_{B_R} \phi\,\nabla_{\mathbf{x}}\cdot\mathbf{v}\,\mathrm{d}\mathbf{x} = \int_{B_R} \nabla_{\mathbf{x}}\cdot\mathbf{G}\,\mathrm{d}\mathbf{x} = \int_{\partial B_R} \mathbf{G}\cdot\mathbf{n}\,\mathrm{d}S,$$

where $\mathbf{n}$ is the outward unit normal. Since $\mathbf{G} = \phi\,\mathbf{v}$ vanishes sufficiently fast at infinity, the surface integral tends to zero:

$$\int_{\mathbb{R}^D} \nabla_{\mathbf{x}} \cdot \mathbf{G}\, \mathrm{d}\mathbf{x} = \lim_{R\to\infty} \int_{\partial B_R} \mathbf{G}\cdot\mathbf{n}\, \mathrm{d}S = 0.$$

Rearranging yields the claim. ∎

A useful special case: taking $\mathbf{v} := \nabla_{\mathbf{x}}\psi$ for a smooth $\psi\colon \mathbb{R}^D \to \mathbb{R}$ (with $\phi\,\nabla_{\mathbf{x}}\psi$ vanishing at infinity) gives Green's identity

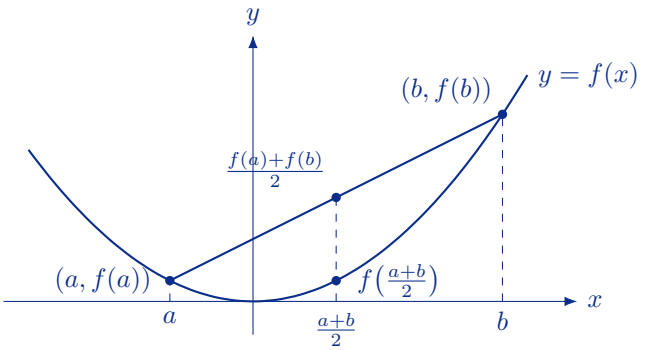
$$\int \nabla_{\mathbf{x}}\phi^{\top}\nabla_{\mathbf{x}}\psi\, \mathrm{d}\mathbf{x} = -\int \phi\,\Delta\psi\, \mathrm{d}\mathbf{x}.$$

E.3 Jensen's Inequality

Intuition. A function f is convex when the chord connecting any two points on its graph lies *above* the graph. This single geometric fact is Jensen's inequality in miniature. Take $f(x) = x^2$ and two points $a < b$. At the midpoint $\frac{a+b}{2}$, the graph sits at height $f\big(\frac{a+b}{2}\big)$, while the chord sits at the average height $\frac{f(a)+f(b)}{2}$. Since the chord lies above the graph,

$$f\Big(\frac{a+b}{2}\Big) \le \frac{f(a)+f(b)}{2},$$

i.e., *the function of the average is at most the average of the function.* Jensen's inequality is exactly this statement, generalized from the two-point average to arbitrary convex combinations and expectations.



Lemma (Jensen's Inequality). Let $f\colon \mathbb{R} \to \mathbb{R}$ be convex and let X be a random variable with finite expectation. Then

$$f\big(\mathbb{E}[X]\big) \le \mathbb{E}\big[f(X)\big].$$

Proof. Assume f is differentiable, and set $\mu := \mathbb{E}[X]$. The key fact is that a convex function lies above every tangent line; in particular, above its tangent at μ:

$$f(x) \geq f(\mu) + f'(\mu)\,(x - \mu) \qquad \text{for all } x \in \mathbb{R}.$$

In particular, at $x = X$,

$$f(X) \geq f(\mu) + f'(\mu)\,(X - \mu).$$

Taking expectations, and using that $f(\mu)$ and $f'(\mu)$ are constants and $\mathbb{E}[X] = \mu$,

$$\mathbb{E}[f(X)] \geq f(\mu) + f'(\mu)\,(\mathbb{E}[X] - \mu) = f(\mathbb{E}[X]). \qquad \blacksquare$$

(If f is not differentiable, the same argument runs verbatim with the tangent replaced by a supporting line at μ, which every convex function admits.)

For a *concave* f the inequality reverses: $f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$.

References

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski. (1985). "A learning algorithm for Boltzmann machines". *Cognitive science*. 9(1): 147–169.

Albergo, M. S., N. M. Boffi, and E. Vanden-Eijnden. (2023). "Stochastic interpolants: A unifying framework for flows and diffusions". *arXiv preprint arXiv:2303.08797*.

Albergo, M. S. and E. Vanden-Eijnden. (2023). "Building Normalizing Flows with Stochastic Interpolants". In: *The Eleventh International Conference on Learning Representations*.

Altschuler, J., J. Niles-Weed, and P. Rigollet. (2017). "Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration". *Advances in neural information processing systems*. 30.

Ambrosio, L., N. Gigli, and G. Savaré. (2005). *Gradient flows: in metric spaces and in the space of probability measures*. Springer.

Anderson, B. D. (1982). "Reverse-time diffusion equation models". *Stochastic Processes and their Applications*. 12(3): 313–326.

Anderson, W. J. (2012). *Continuous-Time Markov Chains: An Applications-Oriented Approach*. Springer.

Arnold, V. I. (1988). *Geometrical Methods in the Theory of Ordinary Differential Equations*. 2nd ed. Vol. 250. *Grundlehren der mathematischen Wissenschaften*. New York: Springer-Verlag. ISBN: 978-0-387-96649-6. DOI: 10.1007/978-1-4612-1037-5.

Atkinson, K., W. Han, and D. E. Stewart. (2009). *Numerical solution of ordinary differential equations*. Vol. 81. John Wiley & Sons.

Austin, J., D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg. (2021). "Structured Denoising Diffusion Models in Discrete State-Spaces". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. URL: https://openreview.net/forum?id=h7-XixPCAL.

Bansal, A., H.-M. Chu, A. Schwarzschild, S. Sengupta, M. Goldblum, J. Geiping, and T. Goldstein. (2023). "Universal guidance for diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 843–852.

Behrmann, J., W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. (2019). "Invertible residual networks". In: *International conference on machine learning*. PMLR. 573–582.

Benamou, J.-D. and Y. Brenier. (2000). "A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem". *Numerische Mathematik*. 84(3): 375–393.

Boffi, N. M., M. S. Albergo, and E. Vanden-Eijnden. (2025). "Flow map matching with stochastic interpolants: A mathematical framework for consistency models". *Transactions on Machine Learning Research*. ISSN: 2835-8856. URL: https://openreview.net/forum?id=cqDH0e6ak2.

Bradley, R. A. and M. E. Terry. (1952). "Rank analysis of incomplete block designs: I. the method of paired comparisons". *Biometrika*. 39(3/4): 324–345.

Caluya, K. F. and A. Halder. (2021). "Wasserstein proximal algorithms for the Schrödinger bridge problem: Density control with nonlinear drift". *IEEE Transactions on Automatic Control*. 67(3): 1163–1178.

Campbell, A., J. Benton, V. De Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet. (2022). "A continuous time framework for discrete denoising models". *Advances in Neural Information Processing Systems*. 35: 28266–28279.

Campbell, A., J. Yim, R. Barzilay, T. Rainforth, and T. Jaakkola. (2024). "Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design". In: *International Conference on Machine Learning*. PMLR. 5453–5512.

Chen, R. T., J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen. (2019). "Residual flows for invertible generative modeling". *Advances in Neural Information Processing Systems*. 32.

Chen, R. T., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. (2018). "Neural ordinary differential equations". *Advances in neural information processing systems*. 31.

Chen, T., G.-H. Liu, and E. Theodorou. (2022). "Likelihood Training of Schrödinger Bridge using Forward-Backward SDEs Theory". In: *International Conference on Learning Representations*.

Chen, Y., T. T. Georgiou, and M. Pavon. (2016). "On the relation between optimal transport and Schrödinger bridges: A stochastic control viewpoint". *Journal of Optimization Theory and Applications*. 169: 671–691.

Chen, Y., T. T. Georgiou, and M. Pavon. (2021). "Stochastic control liaisons: Richard Sinkhorn meets gaspard Monge on a Schrodinger bridge". *Siam Review*. 63(2): 249–313.

Choi, J., S. Kim, Y. Jeong, Y. Gwon, and S. Yoon. (2021). "ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. 14347–14356.

Chorin, A. J. and J. E. Marsden. (1993). *A Mathematical Introduction to Fluid Mechanics*. 3rd ed. Vol. 4. *Texts in Applied Mathematics*. New York: Springer. DOI: 10.1007/978-1-4612-0883-9.

Chung, H., J. Kim, M. T. Mccann, M. L. Klasky, and J. C. Ye. (2022). "Diffusion posterior sampling for general noisy inverse problems". *arXiv preprint arXiv:2209.14687*.

Csiszár, I. (1963). "Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizität von Markoffschen Ketten". *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*. 8(1-2): 85–108.

Cuturi, M. (2013). "Sinkhorn distances: Lightspeed computation of optimal transport". *Advances in neural information processing systems*. 26.

Dacorogna, B. and J. Moser. (1990). "On a Partial Differential Equation Involving the Jacobian Determinant". *Annales de l'Institut Henri Poincaré. Analyse Non Linéaire*. 7(1): 1–26.

Dai Pra, P. (1991). "A stochastic control approach to reciprocal diffusion processes". *Applied mathematics and Optimization*. 23(1): 313–329.

Daras, G., H. Chung, C.-H. Lai, Y. Mitsufuji, J. C. Ye, P. Milanfar, A. G. Dimakis, and M. Delbracio. (2024). "A survey on diffusion models for inverse problems". *arXiv preprint arXiv:2410.00083*.

Daras, G., Y. Dagan, A. Dimakis, and C. Daskalakis. (2023). "Consistent diffusion models: Mitigating sampling drift by learning to be consistent". *Advances in Neural Information Processing Systems*. 36: 42038–42063.

De Bortoli, V. (2022). "Convergence of denoising diffusion models under the manifold hypothesis". *arXiv preprint arXiv:2208.05314*.

De Bortoli, V., J. Thornton, J. Heng, and A. Doucet. (2021). "Diffusion schrödinger bridge with applications to score-based generative modeling". *Advances in Neural Information Processing Systems*. 34: 17695–17709.

Delbracio, M. and P. Milanfar. (2023). "Inversion by Direct Iteration: An Alternative to Denoising Diffusion for Image Restoration". *Transactions on Machine Learning Research*. ISSN: 2835-8856. URL: https://openreview.net/forum?id=VmyFF5lL3F.

Deng, M., H. Li, T. Li, Y. Du, and K. He. (2026). "Generative Modeling via Drifting". *arXiv preprint arXiv:2602.04770*.

Dhariwal, P. and A. Nichol. (2021). "Diffusion models beat gans on image synthesis". *Advances in Neural Information Processing Systems*. 34: 8780–8794.

Dieleman, S. (2026). "Learning the integral of a diffusion model". URL: https://sander.ai/2026/05/06/flow-maps.html.

Dieleman, S., L. Sartran, A. Roshannai, N. Savinov, Y. Ganin, P. H. Richemond, A. Doucet, R. Strudel, C. Dyer, C. Durkan, C. Hawthorne, R. Leblond, W. Grathwohl, and J. Adler. (2022). "Continuous Diffusion for Categorical Data". *arXiv preprint arXiv:2211.15089*.

Efron, B. (2011). "Tweedie's formula and selection bias". *Journal of the American Statistical Association*. 106(496): 1602–1614.

Esser, P., S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, *et al.* (2024). "Scaling rectified flow transformers for high-resolution image synthesis". In: *Forty-first International Conference on Machine Learning*.

Euler, L. (1757). "Principes généraux du mouvement des fluides". *Mémoires de l'Académie des Sciences de Berlin*. 11: 274–315.

Fokker, A. D. (1914). "Die mittlere Energie rotierender elektrischer Dipole im Strahlungsfeld". *Annalen der Physik*. 348(5): 810–820.

Fournier, N. and A. Guillin. (2015). "On the rate of convergence in Wasserstein distance of the empirical measure". *Probability theory and related fields*. 162(3): 707–738.

Frey, B. J., G. E. Hinton, and P. Dayan. (1995). "Does the wake-sleep algorithm produce good density estimators?" *Advances in neural information processing systems*. 8.

Gat, I., T. Remez, N. Shaul, F. Kreuk, R. T. Chen, G. Synnaeve, Y. Adi, and Y. Lipman. (2024). "Discrete Flow Matching". *arXiv preprint arXiv:2407.15595*.

Genevay, A., G. Peyré, and M. Cuturi. (2018). "Learning generative models with sinkhorn divergences". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 1608–1617.

Geng, Z., M. Deng, X. Bai, J. Z. Kolter, and K. He. (2025a). "Mean Flows for One-step Generative Modeling". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. URL: https://openreview.net/forum?id=uWj4s7rMnR.

Geng, Z., A. Pokle, W. Luo, J. Lin, and J. Z. Kolter. (2025b). "Consistency Models Made Easy". In: *The Thirteenth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=xQVxo9dSID.

Geng, Z., A. Pokle, W. Luo, J. Lin, and J. Z. Kolter. (2024). "Consistency models made easy". *arXiv preprint arXiv:2406.14548*.

Gibbs, J. W. (1902). *Elementary principles in statistical mechanics: developed with especial reference to the rational foundations of thermodynamics*. C. Scribner's sons.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. (2014). "Generative adversarial nets". *Advances in neural information processing systems*. 27.

He, Y., N. Murata, C.-H. Lai, Y. Takida, T. Uesaka, D. Kim, W.-H. Liao, Y. Mitsufuji, J. Z. Kolter, R. Salakhutdinov, *et al.* (2023). "Manifold preserving guided diffusion". In: *International Conference on Learning Representations*.

Heitz, E., L. Belcour, and T. Chambon. (2023). "Iterative α-(de) blending: A minimalist deterministic diffusion model". In: *ACM SIGGRAPH 2023 Conference Proceedings*. 1–8.

Hertrich, J., A. Chambolle, and J. Delon. (2025). "On the Relation between Rectified Flows and Optimal Transport". *arXiv preprint arXiv:2505.19712*.

Hinton, G. E. (2002). "Training products of experts by minimizing contrastive divergence". *Neural computation*. 14(8): 1771–1800.

Ho, J., A. Jain, and P. Abbeel. (2020). "Denoising diffusion probabilistic models". *Advances in Neural Information Processing Systems*. 33: 6840–6851.

Ho, J. and T. Salimans. (2021). "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.

Hochbruck, M. and A. Ostermann. (2005). "Explicit exponential Runge–Kutta methods for semilinear parabolic problems". *SIAM Journal on Numerical Analysis*. 43(3): 1069–1090.

Hochbruck, M. and A. Ostermann. (2010). "Exponential integrators". *Acta Numerica*. 19: 209–286.

Hoogeboom, E., D. Nielsen, P. Jaini, P. Forré, and M. Welling. (2021). "Argmax flows and multinomial diffusion: Learning categorical distributions". *Advances in neural information processing systems*. 34: 12454–12465.

Hu, Z., C.-H. Lai, Y. Mitsufuji, and S. Ermon. (2025). "CMT: Mid-Training for Efficient Learning of Consistency, Mean Flow, and Flow Map Models". *arXiv preprint arXiv:2509.24526*.

Huang, C.-W., R. T. Chen, C. Tsirigotis, and A. Courville. (2021). "Convex Potential Flows: Universal Probability Distributions with Optimal Transport and Convex Optimization". In: *International Conference on Learning Representations*.

Hutchinson, M. F. (1989). "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". *Communications in Statistics-Simulation and Computation*. 18(3): 1059–1076.

Hyvärinen, A. and P. Dayan. (2005). "Estimation of non-normalized statistical models by score matching." *Journal of Machine Learning Research*. 6(4).

Jordan, R., D. Kinderlehrer, and F. Otto. (1998). "The variational formulation of the Fokker–Planck equation". *SIAM Journal on Mathematical Analysis*. 29(1): 1–17. DOI: 10.1137/S0036141096303359.

Karras, T., M. Aittala, T. Aila, and S. Laine. (2022). "Elucidating the design space of diffusion-based generative models". *Advances in Neural Information Processing Systems*. 35: 26565–26577.

Karras, T., M. Aittala, J. Lehtinen, J. Hellsten, T. Aila, and S. Laine. (2024). "Analyzing and Improving the Training Dynamics of Diffusion Models". In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 24174–24184.

Kelly, F. P. (1979). *Reversibility and Stochastic Networks*. Cambridge University Press.

Kim, D., C.-H. Lai, W.-H. Liao, N. Murata, Y. Takida, T. Uesaka, Y. He, Y. Mitsufuji, and S. Ermon. (2024a). "Consistency trajectory models: Learning probability flow ode trajectory of diffusion". In: *International Conference on Learning Representations*.

Kim, D., C.-H. Lai, W.-H. Liao, Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon. (2024b). "PaGoDA: Progressive Growing of a One-Step Generator from a Low-Resolution Diffusion Teacher". *arXiv preprint arXiv:2405.14822*.

Kim, D., S. Shin, K. Song, W. Kang, and I.-C. Moon. (2022). "Soft truncation: A universal training technique of score-based diffusion model for high precision score estimation". In: *International Conference on Machine Learning*. PMLR. 11201–11228.

Kingma, D., T. Salimans, B. Poole, and J. Ho. (2021). "Variational diffusion models". *Advances in neural information processing systems*. 34: 21696–21707.

Kingma, D. P. and R. Gao. (2023). "Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation". In: *Thirty-seventh Conference on Neural Information Processing Systems*.

Kingma, D. P. and M. Welling. (2013). "Auto-encoding variational bayes". *arXiv preprint arXiv:1312.6114*.

Kloeden, P. E., E. Platen, P. E. Kloeden, and E. Platen. (1992). *Stochastic differential equations*. Springer.

Knothe, H. (1957). "Contributions to the Theory of Convex Bodies". *Michigan Mathematical Journal*. 4(1): 39–52.

Kolmogoroff, A. (1931). "Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung". *Mathematische Annalen*. 104: 415–458. DOI: 10.1007/BF01457949.

Kuhn, H. W. (1955). "The Hungarian Method for the Assignment Problem". *Naval Research Logistics Quarterly*. 2(1-2): 83–97. DOI: 10.1002/nav.3800020109.

Lagrange, J.-L. (1783). "Mémoire sur la théorie du mouvement des fluides". *Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Berlin*: 151–198.

Lai, C.-H., B. Nguyen, N. Murata, Y. Takida, T. Uesaka, Y. Mitsufuji, S. Ermon, and M. Tao. (2026). "A unified view of drifting and score-based models". *arXiv preprint arXiv:2603.07514*.

Lai, C.-H., Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon. (2023a). "FP-Diffusion: Improving score-based diffusion models by enforcing the underlying score Fokker-Planck equation". In: *International Conference on Machine Learning*. PMLR. 18365–18398.

Lai, C.-H., Y. Takida, T. Uesaka, N. Murata, Y. Mitsufuji, and S. Ermon. (2023b). "On the Equivalence of Consistency-Type Models: Consistency Models, Consistent Diffusion Models, and Fokker-Planck Regularization". *arXiv preprint arXiv:2306.00367*.

Langevin, P. (1908). "Sur la théorie du mouvement brownien". *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*. 146: 530–533.

Larochelle, H. and I. Murray. (2011). "The neural autoregressive distribution estimator". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 29–37.

Lavenant, H. and F. Santambrogio. (2022). "The flow map of the fokker–planck equation does not provide optimal transport". *Applied Mathematics Letters*. 133: 108225.

LeCun, Y., S. Chopra, R. Hadsell, M. Ranzato, F. Huang, *et al.* (2006). "A tutorial on energy-based learning". *Predicting structured data*. 1(0).

Léonard, C. (2012). "From the Schrödinger problem to the Monge–Kantorovich problem". *Journal of Functional Analysis*. 262(4): 1879–1920.

Léonard, C. (2014). "A survey of the Schrödinger problem and some of its connections with optimal transport". *Discrete and Continuous Dynamical Systems-Series A*. 34(4): 1533–1574.

Li, X. L., J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto. (2022). "Diffusion-LM Improves Controllable Text Generation". In: *Advances in Neural Information Processing Systems*. Vol. 35.

Lipman, Y., R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. (2022). "Flow Matching for Generative Modeling". In: *The Eleventh International Conference on Learning Representations*.

Lipman, Y., M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat. (2024). "Flow matching guide and code". *arXiv preprint arXiv:2412.06264*.

Liu, G.-H., Y. Lipman, M. Nickel, B. Karrer, E. Theodorou, and R. T. Q. Chen. (2024). "Generalized Schrödinger Bridge Matching". In: *International Conference on Learning Representations*.

Liu, G.-H., A. Vahdat, D.-A. Huang, E. Theodorou, W. Nie, and A. Anandkumar. (2023). "IÂSB: Image-to-Image Schrödinger Bridge". In: *International Conference on Machine Learning*. PMLR. 22042–22062.

Liu, Q. (2022). "Rectified flow: A marginal preserving approach to optimal transport". *arXiv preprint arXiv:2209.14577*.

Liu, X., C. Gong, *et al.* (2022). "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow". In: *The Eleventh International Conference on Learning Representations*.

Lou, A., C. Meng, and S. Ermon. (2024). "Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution". In: *International Conference on Machine Learning*.

Lu, C. and Y. Song. (2024). "Simplifying, stabilizing and scaling continuous-time consistency models". *arXiv preprint arXiv:2410.11081*.

Lu, C., K. Zheng, F. Bao, J. Chen, C. Li, and J. Zhu. (2022a). "Maximum Likelihood Training for Score-based Diffusion ODEs by High Order Denoising Score Matching". In: *International Conference on Machine Learning*. PMLR. 14429–14460.

Lu, C., Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. (2022b). "Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps". *Advances in Neural Information Processing Systems*. 35: 5775–5787.

Lu, C., Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. (2022c). "Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models". *arXiv preprint arXiv:2211.01095*.

Luan, V. T. (2021). "Efficient exponential Runge–Kutta methods of high order: construction and implementation". *BIT Numerical Mathematics*. 61(2): 535–560.

Luhman, E. and T. Luhman. (2021). "Knowledge distillation in iterative generative models for improved sampling speed". *arXiv preprint arXiv:2101.02388*.

Luo, W., T. Hu, S. Zhang, J. Sun, Z. Li, and Z. Zhang. (2023). "Diff-instruct: A universal approach for transferring knowledge from pre-trained diffusion models". *Advances in Neural Information Processing Systems*. 36: 76525–76546.

Ma, N., M. Goldstein, M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, and S. Xie. (2024). "Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers". In: *European Conference on Computer Vision*. Springer. 23–40.

Maoutsa, D., S. Reich, and M. Opper. (2020). "Interacting particle solutions of Fokker–Planck equations through gradient–log–density estimation". *Entropy*. 22(8): 802.

Meng, C., K. Choi, J. Song, and S. Ermon. (2022). "Concrete score matching: Generalized score matching for discrete data". *Advances in Neural Information Processing Systems*. 35: 34532–34545.

Meng, C., R. Rombach, R. Gao, D. Kingma, S. Ermon, J. Ho, and T. Salimans. (2023). "On distillation of guided diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14297–14306.

Meng, C., Y. Song, W. Li, and S. Ermon. (2021a). "Estimating high order gradients of the data distribution by denoising". *Advances in Neural Information Processing Systems*. 34: 25359–25369.

Meng, C., Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon. (2021b). "Sdedit: Image synthesis and editing with stochastic differential equations". *arXiv preprint arXiv:2108.01073*.

Mikami, T. and M. Thieullen. (2008). "Optimal transportation problem by stochastic optimal control". *SIAM Journal on Control and Optimization*. 47(3): 1127–1139.

Mikami, T. and M. Thieullen. (2006). "Duality theorem for the stochastic optimal control problem". *Stochastic processes and their applications*. 116(12): 1815–1835.

Mokady, R., A. Hertz, K. Aberman, Y. Pritch, and D. Cohen-Or. (2023). "Null-text inversion for editing real images using guided diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6038–6047.

Munkres, J. (1957). "Algorithms for the Assignment and Transportation Problems". *Journal of the Society for Industrial and Applied Mathematics*. 5(1): 32–38. DOI: 10.1137/0105003.

Neklyudov, K., R. Brekelmans, D. Severo, and A. Makhzani. (2023). "Action matching: Learning stochastic dynamics from samples". In: *International Conference on Machine Learning*. PMLR. 25858–25889.

Norris, J. R. (1998). *Markov chains*. No. 2. Cambridge university press.

Nowozin, S., B. Cseke, and R. Tomioka. (2016). "f-gan: Training generative neural samplers using variational divergence minimization". *Advances in neural information processing systems*. 29.

Øksendal, B. (2003). "Stochastic differential equations". In: *Stochastic differential equations*. Springer. 65–84.

Onken, D., S. W. Fung, X. Li, and L. Ruthotto. (2021). "Ot-flow: Fast and accurate continuous normalizing flows via optimal transport". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. No. 10. 9223–9232.

Pavon, M. and A. Wakolbinger. (1991). "On free energy, stochastic control, and Schrödinger processes". In: *Modeling, Estimation and Control of Systems with Uncertainty: Proceedings of a Conference held in Sopron, Hungary, September 1990*. Springer. 334–348.

Peters, J., K. Mulling, and Y. Altun. (2010). "Relative entropy policy search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. No. 1. 1607–1612.

Peyré, G., M. Cuturi, *et al.* (2019). "Computational optimal transport: With applications to data science". *Foundations and Trends® in Machine Learning*. 11(5-6): 355–607.

Planck, V. (1917). "Über einen Satz der statistischen Dynamik und seine Erweiterung in der Quantentheorie". *Sitzungberichte der*.

Pontryagin, L. S. (2018). *Mathematical theory of optimal processes*. Routledge.

Poole, B., A. Jain, J. T. Barron, and B. Mildenhall. (2023). "DreamFusion: Text-to-3D using 2D Diffusion". In: *The Eleventh International Conference on Learning Representations*.

Pra, P. D. and M. Pavon. (1990). "On the Markov processes of Schrödinger, the Feynman-Kac formula and stochastic control". In: *Realization and Modelling in System Theory: Proceedings of the International Symposium MTNS-89, Volume I*. Springer. 497–504.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.* (2021). "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 8748–8763.

Rafailov, R., A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. (2023). "Direct preference optimization: Your language model is secretly a reward model". *Advances in neural information processing systems*. 36: 53728–53741.

Raissi, M. (2018). "Deep hidden physics models: Deep learning of nonlinear partial differential equations". *Journal of Machine Learning Research*. 19(25): 1–24.

Reid, W. (1971). *Ordinary Differential Equations. Applied mathematics series*. Wiley.

Rezende, D. and S. Mohamed. (2015). "Variational inference with normalizing flows". In: *International conference on machine learning*. PMLR. 1530–1538.

Richemond, P. H., S. Dieleman, and A. Doucet. (2022). "Categorical SDEs with Simplex Diffusion". *arXiv preprint arXiv:2210.14784*.

Rosenblatt, M. (1952). "Remarks on a Multivariate Transformation". *The Annals of Mathematical Statistics*. 23(3): 470–472. DOI: 10.1214/aoms/1177729394.

Sahoo, S. S., M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. T. Chiu, A. Rush, and V. Kuleshov. (2024). "Simple and effective masked diffusion language models". *Advances in Neural Information Processing Systems*. 37: 130136–130184.

Salimans, T. and J. Ho. (2021). "Progressive Distillation for Fast Sampling of Diffusion Models". In: *International Conference on Learning Representations*.

Särkkä, S. and A. Solin. (2019). *Applied stochastic differential equations*. Vol. 10. Cambridge University Press.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. (2017). "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*.

Shi, Y., V. D. Bortoli, A. Campbell, and A. Doucet. (2023). "Diffusion Schrödinger Bridge Matching". In: *Advances in Neural Information Processing Systems*.

Shih, A., S. Belkhale, S. Ermon, D. Sadigh, and N. Anari. (2023). "Parallel Sampling of Diffusion Models". *arXiv preprint arXiv:2305.16317*.

Sinkhorn, R. (1964). "A relationship between arbitrary positive matrices and doubly stochastic matrices". *The annals of mathematical statistics*. 35(2): 876–879.

Skreta, M., T. Akhound-Sadegh, V. Ohanesian, R. Bondesan, A. Aspuru-Guzik, A. Doucet, R. Brekelmans, A. Tong, and K. Neklyudov. (2025). "Feynman-Kac Correctors in Diffusion: Annealing, Guidance, and Product of Experts". In: *International Conference on Machine Learning*. PMLR. 55906–55949.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli. (2015). "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. PMLR. 2256–2265.

Song, J., C. Meng, and S. Ermon. (2020a). "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations*.

Song, Y. and P. Dhariwal. (2024). "Improved Techniques for Training Consistency Models". In: *The Twelfth International Conference on Learning Representations*.

Song, Y., P. Dhariwal, M. Chen, and I. Sutskever. (2023). "Consistency models". *arXiv preprint arXiv:2303.01469*.

Song, Y., C. Durkan, I. Murray, and S. Ermon. (2021). "Maximum likelihood training of score-based diffusion models". *Advances in Neural Information Processing Systems*. 34: 1415–1428.

Song, Y. and S. Ermon. (2019). "Generative modeling by estimating gradients of the data distribution". *Advances in Neural Information Processing Systems*. 32.

Song, Y., S. Garg, J. Shi, and S. Ermon. (2020b). "Sliced score matching: A scalable approach to density and score estimation". In: *Uncertainty in Artificial Intelligence*. PMLR. 574–584.

Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. (2020c). "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*.

Su, X., J. Song, C. Meng, and S. Ermon. (2022). "Dual diffusion implicit bridges for image-to-image translation". *arXiv preprint arXiv:2203.08382*.

Sun, H., L. Yu, B. Dai, D. Schuurmans, and H. Dai. (2023). "Score-based Continuous-time Discrete Diffusion Models". In: *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BYWWwSY2G5s.

Tabak, E. G. and E. Vanden-Eijnden. (2010). "Density estimation by dual ascent of the log-likelihood". *Commun. Math. Sci.* 8(1): 217–233.

Tanana, A. (2017). "Comparison of transport map generated by heat flow interpolation and the optimal transport Brenier map". *arXiv preprint arXiv:1709.06464*.

Tong, A., K. FATRAS, N. Malkin, G. Huguet, Y. Zhang, J. Rector-Brooks, G. Wolf, and Y. Bengio. (2024). "Improving and generalizing flow-based generative models with minibatch optimal transport". *Transactions on Machine Learning Research*.

Turner, C. V. (2013). "A family of nonparametric density estimation algorithms". *Communications on Pure and Applied Mathematics*. 66(2): 145–164.

Uria, B., M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. (2016). "Neural autoregressive distribution estimation". *Journal of Machine Learning Research*. 17(205): 1–37.

Vahdat, A. and J. Kautz. (2020). "NVAE: A deep hierarchical variational autoencoder". *Advances in neural information processing systems*. 33: 19667–19679.

Villani, C. *et al.* (2008). *Optimal transport: old and new*. Vol. 338. Springer.

Vincent, P. (2011). "A connection between score matching and denoising autoencoders". *Neural computation*. 23(7): 1661–1674.

Wallace, B., M. Dang, R. Rafailov, L. Zhou, A. Lou, S. Purushwalkam, S. Ermon, C. Xiong, S. Joty, and N. Naik. (2024). "Diffusion model alignment using direct preference optimization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8228–8238.

Wang, Y., Y. He, and M. Tao. (2024). "Evaluating the design space of diffusion-based generative models". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. URL: https://openreview.net/forum?id=9CMOrofB75.

Wang, Z., C. Lu, Y. Wang, F. Bao, C. Li, H. Su, and J. Zhu. (2023). "Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation". *Advances in Neural Information Processing Systems*. 36: 8406–8441.

Weber, R. M. (2023). "The Score-Difference Flow for Implicit Generative Modeling". *Transactions on Machine Learning Research*. ISSN: 2835-8856. URL: https://openreview.net/forum?id=dpGSNLUCzu.

Xu, Y., M. Deng, X. Cheng, Y. Tian, Z. Liu, and T. S. Jaakkola. (2023). "Restart Sampling for Improving Generative Processes". In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: https://openreview.net/forum?id=wFuemocyHZ.

Yin, T., M. Gharbi, T. Park, R. Zhang, E. Shechtman, F. Durand, and B. Freeman. (2024a). "Improved distribution matching distillation for fast image synthesis". *Advances in neural information processing systems*. 37: 47455–47487.

Yin, T., M. Gharbi, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, and T. Park. (2024b). "One-Step Diffusion with Distribution Matching Distillation". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 6613–6623.

Yu, J., Y. Wang, C. Zhao, B. Ghanem, and J. Zhang. (2023). "Freedom: Training-free energy-guided conditional diffusion model". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 23174–23184.

Zhang, Q. and Y. Chen. (2022). "Fast Sampling of Diffusion Models with Exponential Integrator". In: *The Eleventh International Conference on Learning Representations*.

Zhao, W., L. Bai, Y. Rao, J. Zhou, and J. Lu. (2023). "Unipc: A unified predictor-corrector framework for fast sampling of diffusion models". *Advances in Neural Information Processing Systems*. 36: 49842–49869.

Zheng, K., C. Lu, J. Chen, and J. Zhu. (2023). "Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics". *Advances in Neural Information Processing Systems*. 36: 55502–55542.

Zhou, M., H. Zheng, Z. Wang, M. Yin, and H. Huang. (2024). "Score identity distillation: Exponentially fast distillation of pretrained diffusion models for one-step generation". In: *Forty-first International Conference on Machine Learning*.