Accelerating and Privatizing Diffusion Models

by

Tim Dockhorn

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	David J Fleet Professor Dept. of Comp. Science University of Toronto (UofT) & Dept. of Comp. and Mathematical Sciences UofT Scarborough
Supervisor:	Yaoliang Yu Associate Professor School of Comp. Science, University of Waterloo
Internal Members:	Pascal Poupart Professor School of Comp. Science, University of Waterloo
	Gautam Kamath Assistant Professor School of Comp. Science, University of Waterloo
Internal-External Member:	Jun Liu Associate Professor Dept. of Applied Mathematics, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Diffusion models (DMs) have emerged as a powerful class of generative models. DMs offer both state-of-the-art synthesis quality and sample diversity in combination with a robust and scalable learning objective. DMs rely on a diffusion process that gradually perturbs the data towards a normal distribution, while the neural network learns to denoise. Formally, the problem reduces to learning the score function, i.e., the gradient of the log-density of the perturbed data. The reverse of the diffusion process can be approximated by a differential equation, defined by the learned score function, and can therefore be used for generation when starting from random noise. In this thesis, we give a thorough and beginner-friendly introduction to DMs and discuss their history starting from early work on score-based generative models. Furthermore, we discuss connections to other statistical models and lay out applications of DMs, with a focus on image generative modeling.

We then present CLD: a new DM based on critically-damped Langevin dynamics. CLD can be interpreted as running a joint diffusion in an extended space, where the auxiliary variables can be considered "velocities" that are coupled to the data variables as in Hamiltonian dynamics. We derive a novel score matching objective for CLD-based DMs and introduce a fast solver for the reverse diffusion process which is inspired by methods from the statistical mechanics literature. The CLD framework provides new insights into DMs and generalizes many existing DMs which are based on overdamped Langevin dynamics.

Next, we present GENIE, a novel higher-order numerical solver for DMs. Many existing higher-order solvers for DMs built on finite difference schemes which break down in the large step size limit as approximations become too crude. GENIE, on the other hand, learns neural network-based models for higher-order derivatives whose precision do not depend on the step size. The additional networks in GENIE are implemented as small output heads on top of the neural backbone of the original DM, keeping the computational overhead minimal. Unlike recent sampling distillation methods that fundamentally alter the generation process in DMs, GENIE still solves the true generative differential equation, and therefore naturally enables applications such as encoding and guided sampling.

The fourth chapter presents differentially private diffusion models (DPDMs), DMs trained with strict differential privacy guarantees. While modern machine learning models rely on increasingly large training datasets, data is often limited in privacy-sensitive domains. Generative models trained on sensitive data with differential privacy guarantees can sidestep this challenge, providing access to synthetic data instead. DPDMs enforce privacy by using differentially private stochastic gradient descent for training. We thoroughly study the

design space of DPDMs and propose noise multiplicity, a simple yet powerful modification of the DM training objective tailored to the differential privacy setting. We motivate and show numerically why DMs are better suited for differentially private generative modeling than one-shot generators such as generative adversarial networks or normalizing flows.

Finally, we propose to distill the knowledge of large pre-trained DMs into smaller student DMs. Large-scale DMs have achieved unprecedented results across several domains, however, they generally require a large amount of GPU memory and are slow at inference time, making it difficult to deploy them in real-time or on resource-limited devices. In particular, we propose an approximate score matching objective that regresses the student model towards predictions of the teacher DM rather than the clean data as is done in standard DM training. We show that student models outperform the larger teacher model for a variety of compute budgets. Additionally, the student models may also be deployed on GPUs with significantly less memory than was required for the original teacher model.

Acknowledgements

First and foremost I would like to thank my supervisor Yaoliang Yu, whose support has been invaluable throughout my PhD journey. Yaoliang has not only provided me with the freedom to explore my research interests but has also consistently prioritized my growth and development. I am especially grateful for him giving me the opportunity to gain industry experience during my studies and to relocate during the final stages of my PhD.

I would like to extend my thanks to Iain Murray, from whom I have learned an immense amount. His willingness to connect and offer guidance from afar has been tremendously beneficial. I am very much looking forward to finally meeting in person and catching up.

To Karsten Kreis and Arash Vahdat, who introduced me to the beautiful world of Diffusion Models and taught me so much. I have had an incredible time with you both at Nvidia. Also thanks to Sanja Fidler and the entire Nvidia Toronto AI Lab.

To all the friends I have made during my time in Waterloo, many of whom were part of the amazing football pickup community. Special shoutouts to Kent, Ibra, Ali, and Ashwin; I guess it's time for somebody else to win intramurals now anyways. Thanks to the Supersonics team for an incredible season in my last year in Waterloo, and a well deserved league title.

Thank you to the amazing staff at UW and in particular Denise Shantz; in four years, I could not find a single questions she could not answer.

Maddy, thanks for always supporting me, sharing a "mega-desk" with me during the pandemic and taking the journey to move across the country. To an incredible time out West.

Lastly, I want to express my deepest appreciation to my family. The majority of my achievements and successes are a result of their constant support and selfless love. No amount of gratitude can truly capture the extent of their contributions and support during this thesis and beyond.

Dedication

To my family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of unsupervised learning is to unveil the hidden underlying structure of a given dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$. Commonly used methods in unsupervised learning are, for example, clustering, dimensionality reduction and generative modeling. In generative modeling, we attempt to learn a model for the unknown *data generating process*, i.e., the process that generated the given dataset $\mathcal{D}$. A *generative model* allows for a wide range of downstream tasks, such as *probabilistic inference*, *data completion*, *density evaluation*, *outlier detection*, and *sampling*. As such, modeling the data generating process can be considered the pinnacle of unsupervised learning. Generative models are used to model the data commonly encountered in artificial intelligence, such as images, audio speech, and natural language.

If not stated otherwise, we assume that the dataset $\mathcal{D}$ is a subset of $\mathbb{R}^d$ and that the unknown data generating process admits a *density* (function) $p\colon \mathbb{R}^d \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the *set of non-negative numbers*, i.e., $\mathbb{R}_+ = \{x \in \mathbb{R}\colon x \geq 0\}$. Informally, the integral $\int_A p(\mathbf{x})\, d\mathbf{x}$, $A \subseteq \mathbb{R}^d$, quantifies the *probability* of any point $\mathbf{x}_i$ being an element of A. Consequently, the density p is normalized, i.e., $\int_{\mathbb{R}^d} p(\mathbf{x})\, d\mathbf{x} = 1$. The density p uniquely defines the data generating process and we indicate that the elements of the dataset $\mathcal{D}$ are *sampled* from the particular data generating process p by $\mathbf{x}_i \sim p(\mathbf{x})$.

We focus on generative models that are *parameterized* by a *neural network* and the parameters $\boldsymbol{\theta}$ of the neural network are *trained* using only the dataset $\mathcal{D}$. After training, the samples $\mathbf{x} \sim p_{\boldsymbol{\theta}}$ generated by our model should closely resemble those generated by the data generating process p. In addition, we may have (several) secondary objectives such as fast sampling (on *resource limited devices*), preventing the leakage of the training data $\mathcal{D}$ through samples, and being able to (approximately) evaluate the model log-density $\log p_{\boldsymbol{\theta}}$.

1.1 The Origins of Score-Based Generative Modeling

1.1.1 Energy-Based Models

Likelihood-based generative models directly model the density p of the unknown data generating process. Arguably the most flexible likelihood-based generative model is the *energy-based model* (EBM). Let $e_{\boldsymbol{\theta}} \colon \mathbb{R}^d \to \mathbb{R}$ be an *energy function*, with a set of (learnable) parameters $\boldsymbol{\theta}$, then an EBM is defined as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{e^{-e_{\boldsymbol{\theta}}(\mathbf{x})}}{Z_{\boldsymbol{\theta}}}, \tag{1.1}$$

where $Z_{\boldsymbol{\theta}} = \int_{\mathbb{R}^d} e^{-e_{\boldsymbol{\theta}}(\mathbf{x})}\, d\mathbf{x}$ is a, generally intractable, normalizing constant. The parameters $\boldsymbol{\theta}$ of the energy function $e_{\boldsymbol{\theta}}$ can be trained via *maximum likelihood learning*,

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x})], \tag{1.2}$$

where $p_{\text{data}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \delta(\mathbf{x} = \mathbf{x}_i)$ is the *empirical data distribution* and the expectation $\mathbb{E}$ is defined by

$$\mathbb{E}_{\mathbf{x} \sim q(\mathbf{x})}[h(\mathbf{x})] = \int_{\mathbb{R}^d} h(\mathbf{x}) q(\mathbf{x})\, d\mathbf{x}, \tag{1.3}$$

for any function $h \colon \mathbb{R}^d \to \mathbb{R}$ and any density q. Note that evaluating $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ in Equation (1.2) requires us to compute the logarithm of the normalizing constant $Z_{\boldsymbol{\theta}}$. Thus, to make maximum likelihood training feasible, EBMs either restrict the (function) space of the energy function $e_{\boldsymbol{\theta}}$ using, for example, invertible networks in *normalizing flows* [231], or approximate the normalizing constant $Z_{\boldsymbol{\theta}}$ in Equation (1.2) using, for example, *(amortized) variational inference* in *variational autoencoders* [149, 232] or *Markov chain Monte Carlo* (MCMC) sampling in *contrastive divergence* [107].

Sampling from EBMs: For differentiable energy functions $e_{\boldsymbol{\theta}}$, we can generate samples from the EBM $p_{\boldsymbol{\theta}}$ by numerically simulating a *stochastic differential equation* (SDE), namely *overdamped Langevin dynamics*,

$$d\mathbf{x}_t = \nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\theta}}(\mathbf{x}_t)\, dt + \sqrt{2}\, d\mathbf{w}_t, \quad t \in [0, \infty), \tag{1.4}$$

where $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$, $\mathbf{w}_t$ is the standard *Wiener process* and $\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\theta}} \colon \mathbb{R}^d \to \mathbb{R}^d$ is the *score function* [123]. Importantly, for EBMs, the score function $\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\theta}} = -\nabla_{\mathbf{x}_t} e_{\boldsymbol{\theta}}(\mathbf{x}_t)$

is independent of the normalizing constant $Z_{\boldsymbol{\theta}}$. For any smooth density function p_0, the diffusion process converges to the target distribution (under some mild conditions), i.e., $\lim_{t\to\infty} \mathbf{x}_t \sim p_{\boldsymbol{\theta}}(\mathbf{x})$ [154]. To make sampling via overdamped Langevin dynamics tractable, we need to make two approximations: Firstly, we can only simulate the SDE for a finite time interval $t \in [0, T]$, and secondly, we cannot solve the SDE analytically, and therefore need to use a *numerical method*. Applying, for example, the Euler–Maruyama method to the overdamped Langevin dynamics in Equation (1.4) (with $t \in [0, T]$ instead of $t \in [0, \infty)$) results in the following *numerical scheme*:

$$\mathbf{x}_{t_{n+1}} = \mathbf{x}_{t_n} + h_n \nabla_{\mathbf{x}_{t_n}} \log p_{\boldsymbol{\theta}}\left(\mathbf{x}_{t_n}\right) + \sqrt{2h_n}\boldsymbol{\varepsilon}_n, \tag{1.5}$$

where $0 = t_0 < t_1 < \cdots < t_N = T$ is the *time discretization* of the numerical scheme, $\boldsymbol{\varepsilon}_n \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$, and $h_n = t_{n+1} - t_n$ are the step sizes. If T is sufficiently large and the step sizes h_n are sufficiently small, then the density of $\mathbf{x}_T$ will be close to $p_{\boldsymbol{\theta}}$ in distribution [154]. Rather than considering Equation (1.5) as an *approximate sampling scheme* for the EBM $p_{\boldsymbol{\theta}}$ it can be useful to consider the *combination* of the EBM $p_{\boldsymbol{\theta}}$ and the particular choice of the numerical scheme as an independent generative model in itself. Decreasing the number of steps N while keeping T (and $p_{\boldsymbol{\theta}}$) fixed may therefore result in a worse generative model.

1.1.2 Learning Score Models

Rather than modeling the density p of the data generating process (for example via an EBM), we may decide to model its score function $\nabla_{\mathbf{x}} \log p$ directly with a neural network $\boldsymbol{s}_{\boldsymbol{\theta}} \colon \mathbb{R}^d \to \mathbb{R}^d$. To generate samples from the (learned) score model $\boldsymbol{s}_{\boldsymbol{\theta}}$ we can utilize the numerical scheme defined in Equation (1.5), replacing the score of the EBM with the score model, i.e.,

$$\mathbf{x}_{t_{n+1}} = \mathbf{x}_{t_n} + h_n \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}) + \sqrt{2h_n}\boldsymbol{\varepsilon}_n. \tag{1.6}$$

Note that we generally do not restrict our model $s_{\boldsymbol{\theta}}$ to be the gradient of a log-density function, and therefore the above scheme may not converge. That being said, as long as the distance of $\boldsymbol{s}_{\boldsymbol{\theta}}$ to the gradient of *any* log-density is small compared to the step size h_n, which should be the case for well-trained score models, the numerical scheme should result in a useful generative model. While a more in-depth discussion on the convergence of numerical schemes (for SDEs) is out of the scope of this thesis, we refer the reader to Kloeden and Platen [154] for further discussion.

Furthermore, because $\boldsymbol{s}_{\boldsymbol{\theta}}$ may not be the gradient of a log-density function, we cannot

learn the parameters $\boldsymbol{\theta}$ of the score model $s_{\boldsymbol{\theta}}$ via maximum likelihood learning (Equation (1.2)). So how can we learn the score model instead? Let us assume for a moment that we have access to the true score function $\nabla_{\mathbf{x}} \log p$. In this case, we could minimize the expected difference of the true score function and the score model using L_2-regression,

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[\| \nabla_{\mathbf{x}} \log p(\mathbf{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}) \|_2^2 \right]. \tag{1.7}$$

Of course, the above is intractable since we do not have access to the true score function. However, (approximately) equivalent *score matching* objectives exist; for example:

Implicit score matching: In their seminal work, Hyvärinen [123] proved that the following objective is equivalent to Equation (1.7):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x}) + \tfrac{1}{2} \| s_{\boldsymbol{\theta}}(\mathbf{x}) \|_2^2 \right]. \tag{1.8}$$

The main reason why implicit score matching is not commonly used in deep learning is that computing the divergence $\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x})$ is very expensive if the score model $s_{\boldsymbol{\theta}}$ is parameterized by a large neural network and the dimensionality d of $\mathbf{x} \in \mathbb{R}^d$ is large. In particular, it requires d forward and backward passes through the network:

$$\nabla_{\mathbf{x}} \cdot \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{j=1}^{d} \partial_{\mathrm{x}_j} (\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}))_j. \tag{1.9}$$

Sliced score matching: The divergence $\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}$ in implicit score matching [123] can be rewritten using *Hutchinson's trace estimator* [122], i.e., $\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)} \left[\mathbf{v}^\top \nabla_{\mathbf{x}} s_{\boldsymbol{\theta}}(\mathbf{x})\, \mathbf{v} \right]$. In sliced score matching, the expectation over the standard normal distribution is estimated using a single *Monte Carlo* sample, i.e., $\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x}) \approx \mathbf{v}^\top \nabla_{\mathbf{x}} s_{\boldsymbol{\theta}}(\mathbf{x})\, \mathbf{v}$, $\mathbf{v} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$ [261]. The *Jacobian-vector product* $\nabla_{\mathbf{x}} s_{\boldsymbol{\theta}}(\mathbf{x})\, \mathbf{v}$ can be efficiently computed using only a *single* forward and backward pass:

$$\nabla_{\mathbf{x}} \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x})\, \mathbf{v} = \nabla_{\mathbf{x}} \left(\boldsymbol{s}_{\boldsymbol{\theta}}^\top(\mathbf{x})\, \mathbf{v} \right) = \nabla_{\mathbf{x}} h(\mathbf{x}), \tag{1.10}$$

where $h \colon \mathbb{R}^d \to \mathbb{R}$. The main drawback of sliced score matching is that the variance of the single-sample Monte Carlo estimator can be (very) large.

Denoising score matching: Denoising score matching (DSM) [282] entirely circumvents the computation of the divergence $\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}$ by modeling the score function of a perturbed density $q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}$. DSM is defined by

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}),\, \tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})} \left[\| \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) \|_2^2 \right], \tag{1.11}$$

and can be considered an approximate score matching objective. The *perturbation kernel* $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ is generally chosen to be a *normal* distribution, i.e., $\mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \boldsymbol{I}_d)$, in which case DSM can be written as

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}),\, \tilde{\mathbf{x}} \sim \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \boldsymbol{I}_d)} \left[\left\| \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2} - \boldsymbol{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) \right\|_2^2 \right], \tag{1.12}$$

Note that $\boldsymbol{s}_{\boldsymbol{\theta}^\star}$, where $\boldsymbol{\theta}^\star$ is the minimizer of the above equation, is only a good approximation to the true score function $\nabla_{\mathbf{x}} \log p$ if σ is sufficiently small.

1.1.3 Practical Denoising Score Matching

As discussed above, the variance σ in the perturbation kernel $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ of DSM needs to be sufficiently small to obtain a useful score model $\boldsymbol{s}_{\boldsymbol{\theta}}$. However, this makes both learning and sampling challenging [259]: In DSM, the L_2-difference between the score model $\boldsymbol{s}_{\boldsymbol{\theta}}$ and the score of the perturbation kernel $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ is effectively weighted by $p_{\text{data}}(\mathbf{x})\, q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$. For small σ, we can approximate $p_{\text{data}}(\mathbf{x})\, q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$, which implies that differences of the score model and the score of the perturbation kernel are mostly ignored in *small density regions*. This is especially problematic for densities with multiple (highly peaked) *modes*. Even if one can successfully train a score model, accurate sampling with overdamped Langevin dynamics (for *multimodal* densities) requires extremely small step sizes h_n, making it very computationally expensive [259, 289].

In their seminal work, Song and Ermon [259] propose to learn a set of score models with DSM for a variety of noise levels $\{\sigma_0, \ldots, \sigma_M\}$, with $\sigma_0 < \sigma_1 < \ldots, \sigma_M$. While they eventually want to generate samples from the score model $\boldsymbol{s}_{\boldsymbol{\theta}}(\cdot, \sigma = \sigma_0)$, inspired by *simulated annealing* [153] and *annealed importance sampling* [208], the other score models can be used to build a fast numerical scheme, i.e.,

$$\mathbf{x}_{t_{n+1}} = \mathbf{x}_{t_n} + h_n \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, \sigma_{j(n)}) + \sqrt{2h_n} \boldsymbol{\varepsilon}_n, \tag{1.13}$$

where $j(n)\colon \{0, \ldots, N\} \to \{0, \ldots, M\}$ is a monotonically decreasing function with $j(0) = M$ and $j(N) = 0$. Importantly, the step size h_n should be decreased as the noise level σ_m is decreased [259]. Furthermore, Song and Ermon [259] propose to parameterize all score models using a single shared neural network that is conditioned on the noise level σ. This network is trained to minimize the DSM loss for all noise levels jointly:

$$\min_{\boldsymbol{\theta}} \sum_{m=0}^{M} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}),\, \tilde{\mathbf{x}} \sim \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma_m^2 \boldsymbol{I}_d)} \left[\lambda(m) \left\| \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma_m^2} - \boldsymbol{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}, \sigma_m) \right\|_2^2 \right], \tag{1.14}$$

for some weighting function $\lambda : \{0, \dots, M\} \to \mathbb{R}_+$.

1.2 Diffusion Models

In the previous section, we saw that multiple noise levels are a crucial ingredient for score-based generative models trained with DSM. The following question arises: what would happen in the limit of continuous noise levels? For perturbation kernels based on normal distributions, this results in a *diffusion process* which can be described by the following SDE:

$$d\mathbf{x}_t = \boldsymbol{f}(\mathbf{x}_t, t)\, dt + \boldsymbol{G}(\mathbf{x}_t, t)\, d\mathbf{w}_t, \quad t \in [t_{\min}, t_{\max}], \tag{1.15}$$

with drift coefficient $\boldsymbol{f} \colon \mathbb{R}^d \times [t_{\min}, t_{\max}] \to \mathbb{R}^d$, diffusion coefficient $\boldsymbol{G} \colon \mathbb{R}^d \times [t_{\min}, t_{\max}] \to \mathbb{R}^{d \times d}$, standard Wiener process $\mathbf{w}_t$, and $\mathbf{x}_{t_{\min}} \sim q_{t_{\min}}(\mathbf{x}_{t_{\min}})$. For convenience, we set $t_{\min} = 0$, and therefore $q_{t_{\min}} = p_{\text{data}}$, and abbreviate $t_{\max} = T$. Note that setting $t_{\min} > 0$ is equivalent to setting $t_{\min} = 0$ and adding an appropriate amount of Gaussian noise to the data $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$. So far, most research on *diffusion models* (DMs) considers drift and diffusion coefficients of the form $\boldsymbol{f}(\mathbf{x}_t, t) = f(t)\mathbf{x}_t$ and $\boldsymbol{G}(\mathbf{x}_t, t) = g(t)\boldsymbol{I}_d$, respectively. For these diffusion processes, the perturbation kernel is given by

$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d), \tag{1.16}$$

with $f(t) = \dot{\alpha}_t / \alpha_t$ and $g(t) = \alpha_t \sqrt{2\dot{\gamma}_t \gamma_t}$, where $\gamma_t = \sigma_t / \alpha_t$ and $\dot{\alpha}(t)$ denotes the derivative with respect to time, i.e., $\dot{\alpha}(t) = \frac{d}{dt}\alpha(t)$; see Appendix B of Karras et al. [141] for a derivation. Note that by definition we have $\alpha_0 = 1$ and $\sigma_0 = 0$. The functions α_t and σ_t are generally chosen such that the diffusion process in Equation (1.15) approximately converges to a tractable normal distribution $\mathcal{N}(\mathbf{0}_d, \sigma_{\max}^2 \boldsymbol{I}_d)$ at final time T. The quantity γ_t^{-2} is often referred to as the *signal-to-noise ratio* [151].

1.2.1 Generative Modeling with Diffusion Models

Remarkably, the *reverse of a diffusion process* can be written as another diffusion process [7, 104, 263],

$$d\mathbf{x}_t = \Big(\boldsymbol{f}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \cdot \Big[\boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \Big] - \boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \Big)\, dt + \boldsymbol{G}(\mathbf{x}_t, t)\, d\mathbf{w}_t, \tag{1.17}$$

where dt is now an infinitesimal negative timestep and $\mathbf{w}_t$ is a Wiener process for which time "flows backwards" from T to 0. Song et al. [263] propose to learn a time-dependent score model $\boldsymbol{s}_{\boldsymbol{\theta}}$ for the unknown $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ via DSM:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0),\, t \sim p(t),\, \mathbf{x}_t \sim p_t(\mathbf{x}_t \mid \mathbf{x}_0)} \left[\lambda(t) \| \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0) \|_2^2 \right], \tag{1.18}$$

for some weighting function $\lambda \colon [0, T] \to \mathbb{R}_+$ and distribution $p(t)$ over the interval $[0, T]$. After training, we can generate samples by numerically simulating Equation (1.17) and replacing the unknown true score function $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ with the score model $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$:

$$d\mathbf{x}_t = \left(\boldsymbol{f}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \cdot \left[\boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \right] - \boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \right) dt + \boldsymbol{G}(\mathbf{x}_t, t)\, d\mathbf{w}_t. \tag{1.19}$$

To make DMs tractable, we make the approximation of sampling $\mathbf{x}_T$ from the tractable normal distribution $p_T = \mathcal{N}(\mathbf{0}_d, \sigma_{\max}^2 \boldsymbol{I}_d)$. For the perturbation kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d)$, the above simplifies to

$$d\mathbf{x}_t = \left(\frac{\dot{\alpha}_t}{\alpha_t} \mathbf{x}_t - 2\alpha_t^2 \dot{\gamma}_t \gamma_t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \right) dt + \alpha_t \sqrt{2 \dot{\gamma}_t \gamma_t}\, d\mathbf{w}_t. \tag{1.20}$$

1.2.2 Likelihood-Based Generative Modeling after all?

The SDE in Equation (1.19) is *marginally equivalent* to the following *ordinary differential equation* (ODE) [263]:

$$\frac{d\mathbf{x}_t}{dt} = \underbrace{\boldsymbol{f}(\mathbf{x}_t, t) - \frac{1}{2} \nabla_{\mathbf{x}_t} \cdot \left[\boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \right] - \frac{1}{2} \boldsymbol{G}(\mathbf{x}_t, t) \boldsymbol{G}(\mathbf{x}_t, t)^\top \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{=: \widetilde{\boldsymbol{f}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}. \tag{1.21}$$

The above equation is referred to as the *probability flow* ODE [263], an instance of *continuous normalizing flows* [50, 100]. Marginal equivalence means that any intermediate sample $\mathbf{x}_t$, $t \in [0, T]$, is as likely under the SDE in Equation (1.19) as it is under the probability flow ODE. For the perturbation kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d)$, the probability flow ODE simplifies to

$$\frac{d\mathbf{x}_t}{dt} = \frac{\dot{\alpha}_t}{\alpha_t} \mathbf{x}_t - \alpha_t^2 \dot{\gamma}_t \gamma_t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{1.22}$$

Continuous normalizing flows can evaluate the log-density using the *instantaneous change of variables formula* [50],

$$\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \int_0^T \nabla \cdot \widetilde{\boldsymbol{f}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\, dt. \tag{1.23}$$

Computing the divergence $\nabla \cdot \widetilde{\boldsymbol{f}}_{\boldsymbol{\theta}}$ is expensive, and therefore we may approximate it using Hutchtinson's trace estimator [50, 100, 122] as is also done in sliced score matching [261]; see Section 1.1.2. This approximation results in an *unbiased* estimate of the log-density $\log p_0$ (and a *biased* estimate of the density p_0) for *any* $\mathbf{x}_0$. Furthermore, for particular combinations of the weighting function λ and the distribution $p(t)$ over the time interval $[0, T]$, DSM (Equation (1.18)) is equivalent to maximum likelihood learning [120, 262]. That being said, at least for image data other combinations have been shown to be considerably more effective in training models that can generate high perceptual quality samples [53, 115, 262].

1.2.3 A Unified Learning Framework

In DSM (Equation (1.18)) we learn a time-dependent score model $\boldsymbol{s}_{\boldsymbol{\theta}}$ by regressing it towards the time-dependent score of the perturbation kernel $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$. Plugging in the common choice $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d)$ results in

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0), t \sim p(t), \mathbf{x}_t \sim p_t(\mathbf{x}_t \mid \mathbf{x}_0)} \left[\lambda(t) \left\| \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \frac{\alpha_t \mathbf{x}_0 - \mathbf{x}_t}{\sigma_t^2} \right\|_2^2 \right]. \tag{1.24}$$

For this choice, the score model is effectively learning a scaled difference between the clean data $\mathbf{x}_0$ and the noisy data $\mathbf{x}_t$. Alternatively, we could also learn a *denoiser* $D_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ that predicts the clean data $\mathbf{x}_0$ [141, 244]. This would still allow us to define a score model via

$$s_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{\alpha_t D_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \mathbf{x}_t}{\sigma_t^2}. \tag{1.25}$$

It has been shown that it can be highly beneficial to not parameterize the denoiser $D_{\boldsymbol{\theta}}$ directly with a neural network, but rather use a *mixed parameterization* [75, 141, 244, 280]:

$$D_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = c_{\text{skip}}(t)\, \mathbf{x}_t + c_{\text{out}}(t)\, F_{\boldsymbol{\theta}}\left(c_{\text{in}}(t)\mathbf{x}_t, c_{\text{noise}}(t)\right). \tag{1.26}$$

where $F_{\boldsymbol{\theta}}$ is the neural network to be trained. In this unified framework, we may rewrite DSM as learning a denoiser:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0),\, t \sim p(t),\, \mathbf{n} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)} [\lambda(t) \| D_{\boldsymbol{\theta}}(\alpha_t \mathbf{x}_0 + \sigma_t \mathbf{n}, t) - \mathbf{x}_0 \|_2^2]. \tag{1.27}$$

Many existing DM frameworks can be recovered by setting $c_{\text{skip}}(t)$, $c_{\text{out}}(t)$, $c_{\text{in}}(t)$, $c_{\text{noise}}(t)$, $p(t)$ and $\lambda(t)$ appropriately; see Karras et al. [141] for some examples (in a slightly modified framework).

1.2.4 Discrete-Time Diffusion Models

Although we have introduced DMs in a continuous-time framework, they were originally proposed in a discrete-time framework through the lens of *Markov chain* models [109, 257]. In the following, we show that the original discrete-time DMs can be recovered by restricting the support of the distribution $p(t)$, i.e., the set $\{t \in [0,T] \colon p(t) > 0\}$, to the discrete set $\{t_0 = 0, t_1, \dots, t_N = T\}$: In the original discrete-time framework [109, 257], DMs were presented as *latent variable models* of the form $p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0}) = \int_{\mathbf{x}_{t_1}} \cdots \int_{\mathbf{x}_{t_N}} p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0:t_N})\, d\mathbf{x}_{1:T}$. The joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0:t_N})$ is a Markov chain,

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0:t_N}) = p(\mathbf{x}_{t_N}) \prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n-1}} \mid \mathbf{x}_{t_n}), \quad p_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n-1}} \mid \mathbf{x}_{t_n}) = \mathcal{N}(\mathbf{x}_{t_{n-1}}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n), \beta_{t_n} \boldsymbol{I}_d), \tag{1.28}$$

where $p(\mathbf{x}_{t_N}) = \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$, $\{\beta_{t_n}\}_{n=1}^N$ are fixed variances, and $\boldsymbol{\mu}_{\boldsymbol{\theta}} \colon \mathbb{R}^d \times \{t_0, \dots, t_N\} \to \mathbb{R}^d$ is a learnable neural network. The *inference distribution* is fixed to a Markov chain that gradually adds Gaussian noise,

$$q(\mathbf{x}_{t_1:t_N} \mid \mathbf{x}_{t_0}) = \prod_{n=1}^{N} q(\mathbf{x}_{t_n} \mid \mathbf{x}_{t_{n-1}}), \quad q(\mathbf{x}_{t_n} \mid \mathbf{x}_{t_{n-1}}) = \mathcal{N}(\mathbf{x}_{t_n}; \sqrt{1-\beta_{t_n}}\mathbf{x}_{t_{n-1}}, \beta_{t_n} \boldsymbol{I}_d). \tag{1.29}$$

In the limit of small variances $\{\beta_{t_n}\}_{n=1}^N$, the generative Markov chain and the inference distribution have the same functional form [86, 257]. The latent variable model $p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0})$ is then trained by maximizing the *evidence lower bound*, a lower bound to maximum likelihood learning:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_{t_0} \sim p_{\text{data}}(\mathbf{x}_{t_0})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0})] \tag{1.30}$$

$$= \max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_{t_0} \sim p_{\text{data}}(\mathbf{x}_{t_0})}[\log \int_{\mathbf{x}_{t_1}} \cdots \int_{\mathbf{x}_{t_N}} p_{\boldsymbol{\theta}}(\mathbf{x}_{t_0:t_N})\, d\mathbf{x}_{t_1:t_N}] \tag{1.31}$$

$$\geq \max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_{t_0} \sim p_{\text{data}}(\mathbf{x}_{t_0}),\, \mathbf{x}_{t_1:t_N} \sim q(\mathbf{x}_{t_1:t_N} \mid \mathbf{x}_{t_0})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) - \log q(\mathbf{x}_{t_1:t_N} \mid \mathbf{x}_{t_0})] \tag{1.32}$$

$$= C + \max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{\mathbf{x}_{t_0} \sim p_{\text{data}}(\mathbf{x}_{t_0}),\, \mathbf{x}_{t_n} \sim q(\mathbf{x}_{t_n} \mid \mathbf{x}_{t_0})} \left[\tfrac{1}{\beta_{t_n}} \| \widetilde{\mu}_{t_n}(\mathbf{x}_{t_n}, \mathbf{x}_{t_0}) - \mu_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) \|_2^2 \right], \tag{1.33}$$

where C is a constant with respect to $\boldsymbol{\theta}$ and

$$\widetilde{\mu}_{t_n}(\mathbf{x}_{t_n}, \mathbf{x}_{t_0}) = \underbrace{\frac{\sqrt{\bar{\rho}_{tn-1}}(1-\rho_{t_n})}{1-\bar{\rho}_{t_n}}}_{l_{t_n}} \mathbf{x}_{t_0} + \underbrace{\frac{\sqrt{\rho_{t_n}}(1-\bar{\rho}_{t_{n-1}})}{1-\rho_{t_n}}}_{r_{t_n}} \mathbf{x}_{t_n}, \tag{1.34}$$

for $n > 1$ and $\widetilde{\mu}_{t_1}(\mathbf{x}_{t_1}, \mathbf{x}_{t_0}) = \mathbf{x}_{t_0}$ [109]. The marginal $q(\mathbf{x}_{t_n} \mid \mathbf{x}_{t_0})$ of the above inference distribution can be computed in closed-form as $q(\mathbf{x}_{t_n} \mid \mathbf{x}_{t_0}) = \mathcal{N}(\mathbf{x}_{t_n}; \sqrt{\bar{\rho}_{t_n}}\mathbf{x}_{t_0}, (1-\bar{\rho}_{t_n})\boldsymbol{I}_d)$, with $\rho_{t_n} = 1 - \beta_{t_n}$ and $\bar{\rho}_{t_n} = \prod_{s=1}^{n} \rho_{t_s}$ [109]. Maximizing the above evidence lower bound, up to the additive constant C which is irrelevant for training the parameters $\boldsymbol{\theta}$, can actually be exactly recovered in our unified framework (see Section 1.2.3) by setting

$$p(t) = \mathcal{U}(\{t_1, \dots, t_N\}),\ \lambda(t) = N\frac{l_t^2}{\beta_t},\ c_{\text{skip}}(t) = -\frac{r_t}{l_t},\ c_{\text{out}}(t) = \tfrac{1}{l_t}. \tag{1.35}$$

Furthermore, the generative Markov chain in Equation (1.28) can be recovered by simulating the reverse diffusion SDE (Equation (1.17)) using the Euler–Maruyama method with time discretization

$$h_n = t_{n-1} - t_n = \frac{1-\rho_{t_n}}{2\bar{\rho}_{t_n}\dot{\gamma}_{t_n}\gamma_{t_n}}, \quad \gamma_{t_n} = \sqrt{\frac{1-\bar{\rho}_{t_n}}{\bar{\rho}_{t_n}}}. \tag{1.36}$$

Importantly, the above connection shows that discrete-time DMs learn a model for the score function at the discrete points $\{t_0 = 0, t_1, \dots, t_N = T\}$. Consequently, we can define new generative models using other numerical schemes for the reverse diffusion SDE (Equation (1.17)) or the probability flow ODE (Equation (1.21)) with the restriction of using the above step sizes; see Section 3.1.1 for an example.

1.2.5 Connections to other Statistical Models

1.2.5.1 Variational Autoencoders

Variational autoencoders [149, 232] are frameworks to train a latent variable model $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})$ using amortized variational inference. Similar to discrete-time DMs, variational autoencoders are trained by optimizing the evidence lower bound, i.e,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) + \log p(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})], \tag{1.37}$$

where $q_{\boldsymbol{\theta}}$ is a learnable inference distribution. Therefore, discrete-time DMs are a special case of variational autoencoders, in particular a special case of *hierarchical variational*

autoencoders [279]. An important difference of discrete-time DMs to more general (hierarchical) variational autoencoders is that the inference model q has no learnable parameters and all information in the latent variables is progressively destroyed, whereas the latent variables $\mathbf{z}$ of variational autoencoders generally capture global factors of the data.

1.2.5.2 Denoising Autoencoders

Autoencoders are neural network architectures $f_{\boldsymbol{\theta}} \colon \mathbb{R}^d \to \mathbb{R}^d$ that try do predict their inputs, i.e., learn the identity function. Without any restrictions on $f_{\boldsymbol{\theta}}$, this task is obviously trivial, however, the problem becomes more challenging when the neural network has a bottleneck structure, i.e., $f_{\boldsymbol{\theta}}(\mathbf{x}) = h_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}))$, where $g_{\boldsymbol{\theta}} \colon \mathbb{R}^d \to \mathbb{R}^{d_\text{b}}$ and $h_{\boldsymbol{\theta}} \colon \mathbb{R}^{d_\text{b}} \to \mathbb{R}^d$ with $d_\text{b} \ll d$. After training the autoencoder, the $h_{\boldsymbol{\theta}}$ part can be discarded and the network $g_{\boldsymbol{\theta}}$ can be used to extract useful representations of the data. *Denoising autoencoders* [283] modify the input to autoencoders by adding noise, i.e., predict $\mathbf{x}$ given $\mathbf{x} + \boldsymbol{\varepsilon}_\sigma$, $\boldsymbol{\varepsilon}_\sigma \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_d)$:

$$\mathbb{E}_{\mathbf{x} \sim p_\text{data}(\mathbf{x}), \boldsymbol{\varepsilon}_\sigma \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_d)} \left[\| \boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\varepsilon}_\sigma) - \mathbf{x} \|_2^2 \right]. \tag{1.38}$$

Recall that for Gaussian noise, denoiser functions define a score model, and therefore denoising autoencoders are score-based generative models for a single noise level trained via DSM. However, for score-based generative modeling we would generally not parameterize the network with a bottleneck structure since the learned representations of the network are not of (primary) interest. Interestingly, there have been several works that propose to either progressively decrease the noise level σ of denoising autoencoders during training [43, 93, 311] or to train denoising autoencoders for multiple noise levels [94]. These modifications are similar to the ones we discussed in Section 1.1.3 for score-based generative models. For more ideas and intuition on the connections between denoising autoencoders and DMs, we refer the reader to Dieleman [67].

1.2.5.3 Continuous Normalizing Flows

Continuous normalizing flows [50, 100] model the probability density function $p_{\boldsymbol{\theta}}(\mathbf{x}_0)$ using an ODE

$$\frac{d\mathbf{x}_t}{dt} = f_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \tag{1.39}$$

$$\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}_d, \boldsymbol{I}_d). \tag{1.40}$$

Table 1.1: Applications of DMs in image and video modeling.

Application	References
Text-to-image synthesis	[211],[242],[230],[235], [16]
Personalized image synthesis	[237], [92], [91]
Image editing	[197], [185], [31], [143]
Image super-resolution	[110], [240],[238]
Image-to-image translation	[249], [239],[267]
Inverse (medical) image problems	[264],[299],[187],[59],[117]
Text-to-video synthesis	[112], [252],[145], [292]
Video editing	[204], [85], [229], [177]
Personalized video synthesis	[204]

where $f_{\boldsymbol{\theta}}$ is a deep neural network and the above ODE is simulated backwards from $t \in [T, 0]$. As already pointed out in Section 1.2.2, the probability flow ODE of DMs (Equation (1.21)) is a special case of continuous normalizing flows and this fact can be exploited to compute unbiased estimates of the log-density of the DM. Compared to DMs, however, more general continuous normalizing flows have to be trained by maximum likelihood learning, rather than DSM, which generally involves expensive backpropagation through an ODE solver [50, 100]. Furthermore, (general) continuous normalizing flows can only be sampled from by simulating an ODE effectively resulting in a bijection g between latents, i.e., $g_{t,t'}(\mathbf{x}_t) = \mathbf{x}_{t'}$ and $g^{-1}_{t,t'}(\mathbf{x}_{t'}) = \mathbf{x}_t$ for $t, t' \in [0, T]$, which can be restrictive. The stochastic sampling procedure of DMs, on the other hand, naturally allows for secondary applications such as image editing (see SDEdit [197] in the next section).

1.3 Applications of Diffusion Models

1.3.1 Image and Video Modeling

Although the methods introduced in this thesis are applicable to DMs of many domains they are primarily tested for image applications. In Table 1.1, we list some image and video applications together as they are closely related. In the remainder of this section, we briefly summarize some important concepts and methods of this area which are used in the following chapters:

U-Net: The U-Net architecture is a neural network architecture that was initially developed for semantic image segmentation [236]. The U-Net is comprised of downsampling and upsampling blocks. The downsampling blocks sequentially decrease the spatial resolution of an image, each block generally by a factor of two, while the upsampling blocks subsequently bring the spatial resolution back up to the one of the input image. Generally, each upsampling block receives the output of the downsampling block at the same resolution as an additional input via a *skip connection*. The U-Net architecture is by far the most widely used architecture in image generative modeling with DMs. However, recently, some competing transformer-based architectures have been proposed [125, 223]. The U-Net architecture has recently also been adapted for video modeling by adding temporal attention and convolution layers [27, 111, 112].

SDEdit: SDEdit [197] is a general DM-based editing technique that has been widely used in image generative modeling. The core idea is to bring any input (image) closer to the data distribution defined by a pre-trained DM by adding some amount of noise (using the perturbation kernel of the DM) to it. The particular amount of noise is very important: if too much noise is added all prior information is lost, whereas if too little noise is added the noisy image will not coincide with the distribution defined by the DM. After noise is added, the pre-trained DM can be used to denoise. Since the input (image) was only partially noised the coarse structure of the output (image) will remain the same as the input. In contrast, the high-detail features of the output (image) follow the DM distribution. This technique can then be used, for example, to synthesize realistic images from brush strokes [197]. SDEdit is widely used for text-based image editing [235].

Classifier-free guidance: Classifier-free guidance [108] is a technique to guide the iterative sampling process of a DM towards a particular conditioning signal $\mathbf{c}$ by mixing the predictions of a conditional and an unconditional model

$$D^w(\mathbf{x}_t, t, \mathbf{c}) = (1 + w)D(\mathbf{x}_t, t, \mathbf{c}) - wD(\mathbf{x}_t, t), \tag{1.41}$$

where $w \geq 0$ is the *guidance strength*. In practice, the unconditional model can be trained jointly alongside the conditional model in a single network by randomly replacing the conditioning signal $\mathbf{c}$ with a (learnable) null embedding, e.g., 10% of the time [108]. Classifier-free guidance is widely used to improve the sampling quality, at the cost of reduced diversity, of text-to-image DMs [211, 235]. Alternatively, noise-aware classifiers can be also be used to guide the diffusion process [66, 263].

Latent diffusion models: Latent diffusion models (LDMs) [235] are DMs trained in the feature space of an autoencoder rather than in data space directly. Samples are generated

by feeding *latents*, which are sampled from the DM, through the decoder component of the autoencoder. The main advantage of LDMs compared to data space DMs is that they use less GPU memory during inference and training, enabling, for example, generation of very-high resolution images.

Cascaded diffusion models: *Cascaded diffusion models* [110] are a sequence of DMs used for image generation. The idea is to generate a low-resolution image using a *base DM*, followed by gradually increasing the resolution using *upsamplers*, e.g., one upsampler from 64×64 to 256×256 and another upsampler from 256×256 to 1024×1024. The upsamplers are conditioned on the low-resolution image generated by the previous upsampler (or the base DM). The upsamplers can be made more robust against imperfections of the conditioning signal via *conditioning augmentation* [110], i.e., adding some noise to the conditioning image during training and inference. Generally, most parameters of cascaded diffusion models are allocated to the base DM, and less and less parameters are assigned to increasingly higher-resolution upsamplers. Similar to LDMs, this allows for very-high resolution image generation at lower GPU memory cost when compared to direct pixel space generation.

DreamBooth: *DreamBooth* [237] is a technique to personalize large text-to-image DMs. The models are fined-tuned on a small set of images (of a particular subject or style). To prevent overfitting, DreamBooth uses a technique called *prior preservation loss*. *Textual inversion* [90], *low-rank adaptation* [118], and *encoder-based domain tuning* [91] are some alternative fine-tuning techniques for text-to-image personalization.

1.3.2 Other Applications

While this thesis is mainly focused on image modeling, we list some other applications in Table 1.2.

1.4 Contributions

Each of the next four chapters of this thesis is structured around a central (workshop) paper, for which I am the sole first author. Besides the publications themselves, I outline the explicit contributions, and describe novel insights and discuss follow-up work where applicable. The chapters are purposefully ordered chronologically to provide the reader

Table 1.2: Some applications of DMs outside image modeling.

Primary application	Secondary application	References
Computer vision	Text-to-3D synthesis	[226], [174]
	Text-to-4D synthesis	[253]
	3D shape generation	[188], [37]. [319],[190],[82]
	Semantic segmentation	[21]
Natural language processing	Text synthesis	[68],[12], [170], [54]
Audio generative modeling	Speech synthesis	[48],[157],[130],[49],[227],[178]
Temporal data modeling	Time series imputation	[272]
	Time series forecasting	[5]
Molecule & protein generation	Small-scale molecular generation	[251],[300],[298],[133],[114]
	Large-scale protein synthesis	[287],[124]
	Molecular 3D density map generation	[159]

with some insight about the development of this fast moving field. In my opinion most strikingly is the development of the inference time of DMs which has roughly decreased by a factor of 100 within the last two years. All four chapters follow a common theme of improving secondary objectives of DMs (such as inference acceleration and privatization) while optimizing the main objective of high-quality diverse sampling.

The complexity of the learning problem in DMs depends, other than on the training data itself, only on the perturbation kernel. In Chapter 2, we introduce a new type of DM based on *critically-damped Langevin dynamics*. This particular choice allows for fast numerical sampling schemes from the statistical mechanics and molecular dynamics literature. Dockhorn et al. [75] was published at ICLR 2022.

In Chapter 3, we introduce a fast higher-order numerical sampling scheme for DMs. Rather than the common procedure of approximating higher-order derivatives with *finite difference* schemes we learn them via *distillation*. Dockhorn et al. [74] was published at NeurIPS 2022.

In Chapter 4, we propose to learn DMs with strict differential privacy guarantees. We build on existing techniques form the differential privacy literature and tailor them to the training of DMs. Dockhorn et al. [73] was submitted to TMLR in April 2023.

Lastly, in Chapter 5 we present a technique to distill the knowledge in DMs into small *student models*. Given a fixed budget of floating point operations, we show that the distilled student models considerably outperform their teachers. Furthermore, the student

models have the additional benefit that they need less GPU memory, making them potentially suitable for deployment on resource limited devices. Dockhorn et al. [76] was presented at the CVPR 2023 workshop for "Generative Models for Computer Vision".

Three (workshop) papers on generative modeling for which I was a shared first author are not included in this thesis. In Dockhorn et al. [71], we introduce a method for *density deconvolution* based on *normalizing flows* (ICML 2021 workshop for "Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models"). In Kreis et al. [159], we train latent DMs for *Cryo-EM* data (NeurIPS 2022 workshop for "Machine Learning for Structural Biology Workshop"). Lastly, in Blattmann et al. [27] we insert additional layers into pre-trained image latent DMs and fine-tune them for video generation (CVPR 2023).

Before working in generative modeling, I dabbled my feet in quantization of neural networks. The work resulted in a first author paper at NeurIPS 2021: Dockhorn et al. [72].

Chapter 2

Score-Based Generative Modeling With Critically-Damped Langevin Diffusion

2.1 Choosing the Diffusion Process in Diffusion Models

Recall the general diffusion process from Section 1.2, i.e.,

$$d\mathbf{x}_t = \boldsymbol{f}(\mathbf{x}_t, t)\, dt + \boldsymbol{G}(\mathbf{x}_t, t)\, d\mathbf{w}_t, \quad t \in [0, T], \tag{2.1}$$

with drift coefficient $\boldsymbol{f}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$, diffusion coefficient $\boldsymbol{G}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^{d\times d}$, standard Wiener process $\mathbf{w}_t$, and $\mathbf{x}_0 \sim q_0(\mathbf{x}_0)$. Much of the research on DMs has considered drift and diffusion coefficients of the form $\boldsymbol{f}(\mathbf{x}_t, t) = f(t)\mathbf{x}_t$ and $\boldsymbol{G}(\mathbf{x}_t, t) = g(t)\boldsymbol{I}_d$, respectively. One popular choice is the *variance-preserving* diffusion process for which $f(t) = -\beta_t$ and $g(t) = \sqrt{2\beta_t}$, with positive monotonically-increasing β_t [263]. Considering β_t as a time rescaling reveals that the variance-preserving diffusion process is equivalent to the overdamped Langevin dynamics (Equation (1.4)) for the standard normal distribution $\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$ [75].

Langevin dynamics are widely used in physics and the natural sciences. They play, for example, a crucial role in the world of MCMC. Within this field it is widely known that discretizations of overdamped Langevin dynamics converge relatively slow when compared

to discretizations of *underdamped Langevin dynamics* [55],

$$d\mathbf{x}_t = M^{-1}\mathbf{v}_t\,dt, \tag{2.2}$$

$$d\mathbf{v}_t = \nabla_{\mathbf{x}_t}\log p(\mathbf{x}_t)\,dt + M^{-1}\Gamma\,\mathbf{x}_t\,dt + \sqrt{2\Gamma}\,d\mathbf{w}_t, \tag{2.3}$$

where $\mathbf{v}_t \in \mathbb{R}^d$ are *auxiliary variables*. Here, the mass parameter $M \in \mathbb{R}_+$ determines the coupling between $\mathbf{x}_t$ and $\mathbf{v}_t$, and $\Gamma \in \mathbb{R}_+$ is a friction coefficient that determines the strength of the noise injection into the $\mathbf{v}_t$ variables. Discretizations of the underdamped Langevin dynamics can be viewed as a form of Hamiltonian MCMC [209], a very popular MCMC algorithm. In DMs, the learning problem depends, other than on the data itself, only on the diffusion process. This makes it imperative to explore the wide variety of possible diffusion processes and go beyond overdamped Langevin dynamics.

2.2 Preface

This section presents the paper "Score-Based Generative Modeling with Critically-Damped Langevin Diffusion". In this work, we propose a new DM (referred to as CLD in the remainder of this work) which is based on *critically-damped Langevin dynamics*, an instance of underdamped Langevin dynamics for which the mass and friction parameters are constrained to $\Gamma^2 = 4M$. We motivate the particular choice of critically-damped Langevin dynamics and propose a fast numerical scheme for the reverse diffusion process, motivated by the statistical mechanics and the molecular dynamics literature. The paper was initially put on arXiv in December 2021, and then accepted and presented at the *International Conference on Learning Representations* in April 2022. The paper has an average reviewer score of 8.5/10 (99th percentile).

Naming: At the time of publication, DMs were more commonly referred to as *score-based generative models* and if the paper had been published today, the title would have likely been "Critically-Damped Langevin Diffusion" instead.

Contributions: I was the sole first author of this work. Arash Vahdat and Karsten Kreis were co-authors, and Karsten supervised this project. Karsten initially proposed to use underdamped Langevin dynamics for DMs. Eventually, I suggested to use *critically-damped Langevin dynamics*, a special form of underdamped Langevin dynamics, for which I found a closed-form perturbation kernel. Karsten and I jointly came up with the *hybrid score matching* loss function and the *symmetric splitting CLD sampler*. I implemented all

code and ran all experiments. The paper was written jointly.

Differences in notation: The notation in the reverse diffusion process is different as the time variable is running from $t = 0$ to $t = T$ rather than from $t = T$ to $t = 0$.

Reproducibility: The code and models of this work have been open-sourced. See https://github.com/nv-tlabs/CLD-SGM for instructions to reproduce our results.

arXiv: The paper "Score-Based Generative Modeling with Critically-Damped Langevin Diffusion" is available on ArXiV (v4). The following version is simply reformatted into the style of the thesis.

Abstract: Score-based generative models (SGMs) have demonstrated remarkable synthesis quality. SGMs rely on a diffusion process that gradually perturbs the data towards a tractable distribution, while the generative model learns to denoise. The complexity of this denoising task is, apart from the data distribution itself, uniquely determined by the diffusion process. We argue that current SGMs employ overly simplistic diffusions, leading to unnecessarily complex denoising processes, which limit generative modeling performance. Based on connections to statistical mechanics, we propose a novel *critically-damped Langevin diffusion* (CLD) and show that CLD-based SGMs achieve superior performance. CLD can be interpreted as running a joint diffusion in an extended space, where the auxiliary variables can be considered "velocities" that are coupled to the data variables as in Hamiltonian dynamics. We derive a novel score matching objective for CLD and show that the model only needs to learn the score function of the conditional distribution of the velocity given data, an easier task than learning scores of the data directly. We also derive a new sampling scheme for efficient synthesis from CLD-based diffusion models. We find that CLD outperforms previous SGMs in synthesis quality for similar network architectures and sampling compute budgets. We show that our novel sampler for CLD significantly outperforms solvers such as Euler–Maruyama. Our framework provides new insights into score-based denoising diffusion models and can be readily used for high-resolution image synthesis. Project page and code: https://nv-tlabs.github.io/CLD-SGM.

2.3 Main Paper

2.3.1 Introduction

Score-based generative models (SGMs) and denoising diffusion probabilistic models have emerged as a promising class of generative models [150, 257, 262, 263, 280]. SGMs offer high quality synthesis and sample diversity, do not require adversarial objectives, and have found applications in image [66, 109, 110, 212], speech [48, 130, 157], and music synthesis [202], image editing [89, 197, 255], super-resolution [168, 240], image-to-image translation [249], and 3D shape generation [188, 319]. SGMs use a diffusion process to gradually add noise to the data, transforming a complex data distribution into an analytically tractable prior distribution. A neural network is then utilized to learn the score function—the gradient of the log probability density—of the perturbed data. The learnt scores can be used to solve a stochastic differential equation (SDE) to synthesize new samples. This corresponds to an iterative denoising process, inverting the forward diffusion.

In the seminal work by Song et al. [263], it has been shown that the score function that needs to be learnt by the neural network is uniquely determined by the forward diffusion process. Consequently, the complexity of the learning problem depends, other than on the data itself, only on the diffusion. Hence, the diffusion process is the key component of SGMs that needs to be revisited to further improve SGMs, for example, in terms of synthesis quality or sampling speed.

Inspired by statistical mechanics [277], we propose a novel forward diffusion process, the *critically-damped Langevin diffusion (CLD)*. In CLD, the data variable, $\mathbf{x}_t$ (time t along the diffusion), is augmented with an additional "velocity" variable $\mathbf{v}_t$ and a diffusion process is run in the joint data-velocity space. Data and velocity are coupled to each other as in Hamiltonian dynamics, and noise is injected only into the velocity variable. As in Hamiltonian Monte Carlo [81, 209], the Hamiltonian component helps to efficiently traverse the joint data-velocity space and to transform the data distribution into the prior distribution more smoothly. We derive the corresponding score matching objective and show that for CLD the neural network is tasked with learning only the score of the conditional distribution of velocity given data $\nabla_{\mathbf{v}_t} \log p_t(\mathbf{v}_t|\mathbf{x}_t)$, which is arguably easier than learning the score of diffused data directly. Using techniques from molecular dynamics [34, 165, 277], we also derive a new SDE integrator tailored to CLD's reverse-time synthesis SDE.

We extensively validate CLD and the novel SDE solver: **(i)** We show that the neural

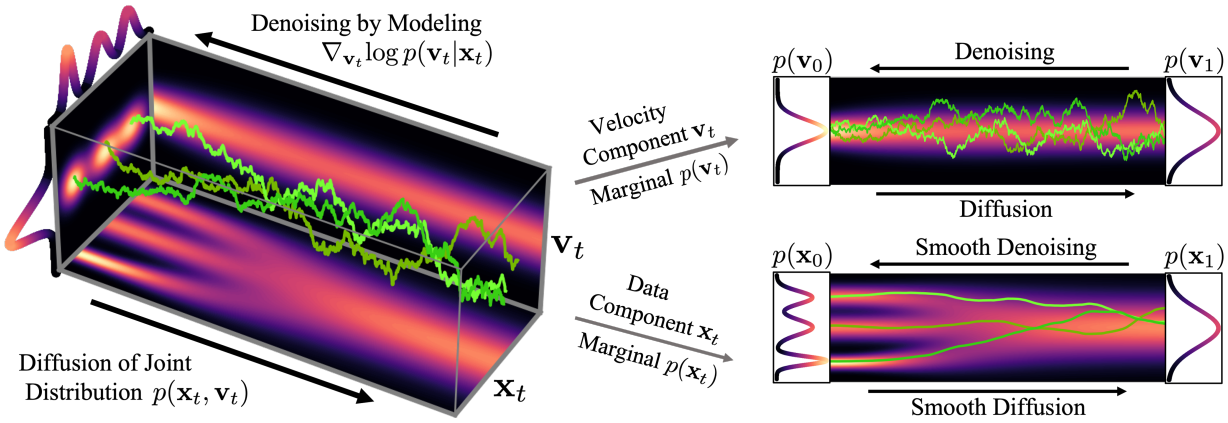


Figure 2.1: In critically-damped Langevin diffusion, the data $\mathbf{x}_t$ is augmented with a velocity $\mathbf{v}_t$. A diffusion coupling $\mathbf{x}_t$ and $\mathbf{v}_t$ is run in the joint data-velocity space (probabilities in **red**). Noise is injected only into $\mathbf{v}_t$. This leads to smooth diffusion trajectories (**green**) for the data $\mathbf{x}_t$. Denoising only requires $\nabla_{\mathbf{v}_t} \log p(\mathbf{v}_t|\mathbf{x}_t)$.

networks learnt in CLD-based SGMs are smoother than those of previous SGMs. **(ii)** On the CIFAR-10 image modeling benchmark, we demonstrate that CLD-based models outperform previous diffusion models in synthesis quality for similar network architectures and sampling compute budgets. We attribute these positive results to the Hamiltonian component in the diffusion and to CLD's easier score function target, the score of the velocity-data conditional distribution $\nabla_{\mathbf{v}_t} \log p_t(\mathbf{v}_t|\mathbf{x}_t)$. **(iii)** We show that our novel sampling scheme for CLD significantly outperforms the popular Euler–Maruyama method. **(iv)** We perform ablations on various aspects of CLD and find that CLD does not have difficult-to-tune hyperparameters.

In summary, we make the following technical contributions: **(i)** We propose CLD, a novel diffusion process for SGMs. **(ii)** We derive a score matching objective for CLD, which requires only the conditional distribution of velocity given data. **(iii)** We propose a new type of denoising score matching ideally suited for scalable training of CLD-based SGMs. **(iv)** We derive a tailored SDE integrator that enables efficient sampling from CLD-based models. **(v)** Overall, we provide novel insights into SGMs and point out important new connections to statistical mechanics.

2.3.2 Background

Consider a diffusion process $\mathbf{u}_t \in \mathbb{R}^d$ defined by the Itô SDE

$$d\mathbf{u}_t = \boldsymbol{f}(\mathbf{u}_t, t)\, dt + \boldsymbol{G}(\mathbf{u}_t, t)\, d\mathbf{w}_t, \quad t \in [0, T], \tag{2.4}$$

with continuous time variable $t \in [0, T]$, standard Wiener process $\mathbf{w}_t$, drift coefficient $\boldsymbol{f}\colon \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ and diffusion coefficient $\boldsymbol{G}\colon \mathbb{R}^d \times [0, T] \to \mathbb{R}^{d\times d}$. Defining $\bar{\mathbf{u}}_t := \mathbf{u}_{T-t}$, a corresponding reverse-time diffusion process that inverts the above forward diffusion can be derived [7, 104, 263] (with positive dt and $t \in [0, T]$):

$$d\bar{\mathbf{u}}_t = \left[-\boldsymbol{f}(\bar{\mathbf{u}}_t, T-t) + \boldsymbol{G}(\bar{\mathbf{u}}_t, T-t)\boldsymbol{G}(\bar{\mathbf{u}}_t, T-t)^\top \nabla_{\bar{\mathbf{u}}_t} \log p_{T-t}(\bar{\mathbf{u}}_t)\right] dt + \boldsymbol{G}(\bar{\mathbf{u}}_t, T-t) d\mathbf{w}_t, \tag{2.5}$$

where $\nabla_{\bar{\mathbf{u}}_t} \log p_{T-t}(\bar{\mathbf{u}}_t)$ is the score function of the marginal distribution over $\bar{\mathbf{u}}_t$ at time $T - t$.

The reverse-time process can be used as a generative model. In particular, Song et al. [263] model data $\mathbf{x}$, setting $p(\mathbf{u}_0){=}p_{\text{data}}(\mathbf{x})$. Currently used SDEs [146, 263] have drift and diffusion coefficients of the simple form $\boldsymbol{f}(\mathbf{x}_t, t){=}f(t)\mathbf{x}_t$ and $\boldsymbol{G}(\mathbf{x}_t, t){=}g(t)\boldsymbol{I}_d$. Generally, $\boldsymbol{f}$ and $\boldsymbol{G}$ are chosen such that the SDE's marginal, equilibrium density is approximately Normal at time T, i.e., $p(\mathbf{u}_T){\approx}\mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$. We can then initialize $\mathbf{x}_0$ based on a sample drawn from a complex data distribution, corresponding to a far-from-equilibrium state. While the state $\mathbf{x}_0$ relaxes towards equilibrium via the forward diffusion, we can learn a model $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ for the score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, which can be used for synthesis via the reverse-time SDE in Eq. (2.5). If $\boldsymbol{f}$ and $\boldsymbol{G}$ take the simple form from above, the denoising score matching [282] objective for this task is:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[0,T]} \mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0)} \mathbb{E}_{\mathbf{x}_t\sim p_t(\mathbf{x}_t|\mathbf{x}_0)} \left[\lambda(t) \|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2\right] \tag{2.6}$$

If $\boldsymbol{f}$ and $\boldsymbol{G}$ are affine, the conditional distribution $p_t(\mathbf{x}_t|\mathbf{x}_0)$ is Normal and available analytically [247]. Different $\lambda(t)$ result in different trade-offs between synthesis quality and likelihood in the generative model defined by $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ [262, 280].

2.3.3 Critically-Damped Langevin Diffusion

We propose to augment the data $\mathbf{x}_t \in \mathbb{R}^d$ with auxiliary *velocity*[1] variables $\mathbf{v}_t \in \mathbb{R}^d$ and utilize a diffusion process that is run in the joint $\mathbf{x}_t$-$\mathbf{v}_t$-space. With $\mathbf{u}_t = (\mathbf{x}_t, \mathbf{v}_t)^\top \in \mathbb{R}^{2d}$, we set

$$\boldsymbol{f}(\mathbf{u}_t, t) \coloneqq \left(\begin{pmatrix} 0 & \beta M^{-1} \\ -\beta & -\Gamma \beta M^{-1} \end{pmatrix} \otimes \boldsymbol{I}_d \right) \mathbf{u}_t \,, \qquad \boldsymbol{G}(\mathbf{u}_t, t) \coloneqq \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{2\Gamma\beta} \end{pmatrix} \otimes \boldsymbol{I}_d, \tag{2.7}$$

where $\otimes$ denotes the Kronecker product. The coupled SDE that describes the diffusion process is

$$\begin{pmatrix} d\mathbf{x}_t \\ d\mathbf{v}_t \end{pmatrix} = \underbrace{\begin{pmatrix} M^{-1}\mathbf{v}_t \\ -\mathbf{x}_t \end{pmatrix} \beta dt}_{\text{Hamiltonian component}=:H} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\mathbf{v}_t \end{pmatrix} \beta dt + \begin{pmatrix} 0 \\ \sqrt{2\Gamma\beta} \end{pmatrix} d\mathbf{w}_t}_{\text{Ornstein-Uhlenbeck process}=:O}, \tag{2.8}$$

which corresponds to *Langevin dynamics* in each dimension. That is, each x_i is independently coupled to a velocity v_i, which explains the blockwise structure of $\boldsymbol{f}$ and $\boldsymbol{G}$. The *mass* $M \in \mathbb{R}^+$ is a hyperparameter that determines the coupling between the $\mathbf{x}_t$ and $\mathbf{v}_t$ variables; $\beta \in \mathbb{R}^+$ is a constant time rescaling chosen such that the diffusion converges to its equilibrium distribution within $t \in [0, T]$ (in practice, we set T=1) when initialized from a data-defined non-equilibrium state and is analogous to $\beta(t)$ in previous diffusions (we could also use time-dependent $\beta(t)$, but found constant β's to work well, and therefore opted for simplicity); $\Gamma \in \mathbb{R}^+$ is a *friction* coefficient that determines the strength of the noise injection into the velocities. Notice that the SDE in Eq. (2.8) consists of two components. The H term represents a Hamiltonian component. Hamiltonian dynamics are frequently used in Markov chain Monte Carlo methods to accelerate sampling and efficiently explore complex probability distributions [209]. The Hamiltonian component in our diffusion process plays a similar role and helps to quickly and smoothly converge the initial joint data-velocity distribution to the equilibrium, or prior (see Fig. 2.1). Furthermore, Hamiltonian dynamics on their own are trivially invertible [277], which intuitively is also beneficial in our situation when using this diffusion for training SGMs. The O term corresponds to an Ornstein-Uhlenbeck process [247] in the velocity component, which injects noise such that the diffusion dynamics properly converge to equilibrium for any Γ>0. It can be shown that the equilibrium distribution of this diffusion is $p_{\text{EQ}}(\mathbf{u}) = \mathcal{N}(\mathbf{x}; \mathbf{0}_d, \boldsymbol{I}_d)\mathcal{N}(\mathbf{v}; \mathbf{0}_d, M\boldsymbol{I}_d)$ (see App. 2.4.2.2).

[1] We call the auxiliary variables *velocities*, as they play a similar role as velocities in physical systems. Formally, our velocity variables would rather correspond to physical momenta, but the term momentum is already widely used in machine learning and our mass M is unitless anyway.

There is a crucial balance between M and Γ [193]: For $\Gamma^2<4M$ (*underdamped* Langevin dynamics) the Hamiltonian component dominates, which implies oscillatory dynamics of $\mathbf{x}_t$ and $\mathbf{v}_t$ that slow down convergence to equilibrium. For $\Gamma^2>4M$ (*overdamped* Langevin dynamics) the O-term dominates which also slows down convergence, since the accelerating effect by the Hamiltonian component is suppressed due to the strong noise injection. For $\Gamma^2=4M$ (*critical damping*), an ideal balance is achieved and convergence to $p_{\text{EQ}}(\mathbf{u})$ occurs as fast as possible in a smooth manner without oscillations (also see discussion in App. 2.4.1.1) [193]. Hence, we propose to set $\Gamma^2=4M$ and call the resulting diffusion *critically-damped Langevin diffusion (CLD)* (see Fig. 2.1).

Diffusions such as the VPSDE [263] correspond to overdamped Langevin dynamics with high friction coefficients Γ (see App. 2.4.1.2). Furthermore, in previous works noise is injected directly into the data variables (pixels, for images). In CLD, only the velocity variables are subject to direct noise and the data is perturbed only indirectly due to the coupling between $\mathbf{x}_t$ and $\mathbf{v}_t$.

2.3.3.1 Score Matching Objective

Considering the appealing convergence properties of CLD, we propose to utilize CLD as forward diffusion process in SGMs. To this end, we initialize the joint distribution $p(\mathbf{u}_0)=p(\mathbf{x}_0)\,p(\mathbf{v}_0)=p_{\text{data}}(\mathbf{x}_0)\mathcal{N}(\mathbf{v}_0;\mathbf{0}_d,\gamma M\boldsymbol{I}_d)$ with hyperparameter $\gamma<1$ and let the distribution diffuse towards the tractable equilibrium—or prior—distribution $p_{\text{EQ}}(\mathbf{u})$. We can then learn the corresponding score functions and define CLD-based SGMs. Following a similar derivation as [262], we obtain the score matching (SM) objective (see App. 2.4.2.3):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[0,T]}\mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t)}\left[\lambda(t)\|s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)-\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)\|_2^2\right] \tag{2.9}$$

Notice that this objective requires only the velocity gradient of the log-density of the joint distribution, i.e., $\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)$. This is a direct consequence of injecting noise into the velocity variables *only*. Without loss of generality, $p_t(\mathbf{u}_t)=p_t(\mathbf{x}_t,\mathbf{v}_t)=p_t(\mathbf{v}_t|\mathbf{x}_t)p_t(\mathbf{x}_t)$. Hence,

$$\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)=\nabla_{\mathbf{v}_t}\left[\log p_t(\mathbf{v}_t|\mathbf{x}_t)+\log p_t(\mathbf{x}_t)\right]=\nabla_{\mathbf{v}_t}\log p_t(\mathbf{v}_t|\mathbf{x}_t) \tag{2.10}$$

This means that in CLD the neural network-defined score model $s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)$ only needs to learn the score of the conditional distribution $p_t(\mathbf{v}_t|\mathbf{x}_t)$, an arguably easier task than learning the score of $p_t(\mathbf{x}_t)$, as in previous works, or of the joint $p_t(\mathbf{u}_t)$. This is the case,

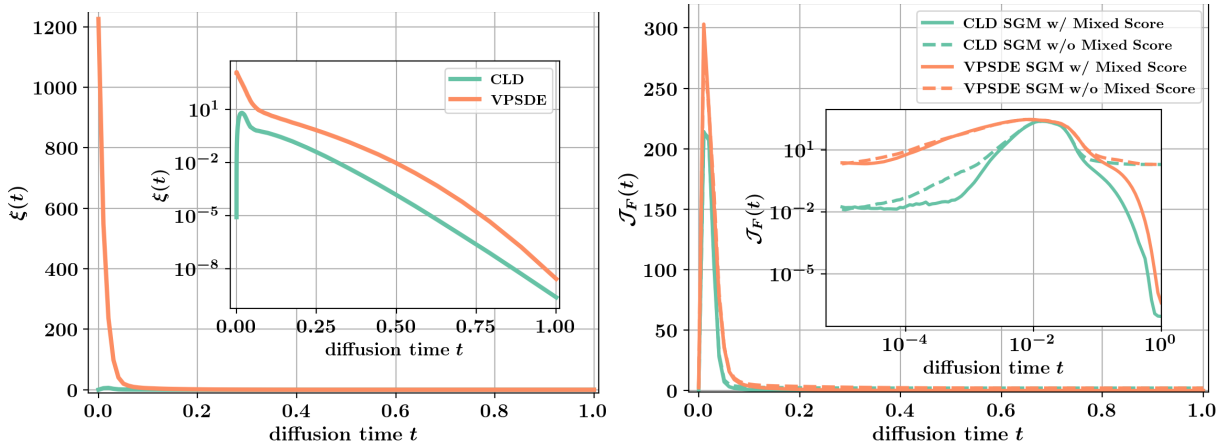


Figure 2.2: *Left:* Difference $\xi(t)$ (via $L2$ norm) between score of diffused data and score of Normal distribution. *Right:* Frobenius norm of Jacobian $\mathcal{J}_F(t)$ of the neural network defining the score function for different t. The underlying data distribution is a mixture of Normals. *Insets:* Different axes (see App. 2.4.5.1 for detailed definitions of $\xi(t)$ and $\mathcal{J}_F(t)$).

because our velocity distribution is initialized from a simple Normal distribution, such that $p_t(\mathbf{v}_t|\mathbf{x}_t)$ is closer to a Normal distribution for all $t{\geq}0$ (and for any $\mathbf{x}_t$) than $p_t(\mathbf{x}_t)$ itself. This is most evident at $t{=}0$: The data and velocity distributions are independent at $t{=}0$ and the score of $p_0(\mathbf{v}_0|\mathbf{x}_0){=}p_0(\mathbf{v}_0)$ simply corresponds to the score of the Normal distribution $p_0(\mathbf{v}_0)$ from which the velocities are initialized, whereas the score of the data distribution $p_0(\mathbf{x}_0)$ is highly complex and can even be unbounded [146]. We empirically verify the reduced complexity of the score of $p_t(\mathbf{v}_t|\mathbf{x}_t)$ in Fig. 2.2. We find that the score that needs to be learnt by the model is more similar to a score corresponding to a Normal distribution for CLD than for the VPSDE. We also measure the complexity of the neural networks that were learnt to model this score via the squared Frobenius norm of their Jacobians. We find that the CLD-based SGMs have significantly simpler and smoother neural networks than VPSDE-based SGMs for most t, in particular when leveraging a mixed score formulation (see next section).

2.3.3.2 Scalable Training

A Practical Objective. We cannot train directly with Eq. (2.9), since we do not have access to the marginal distribution $p_t(\mathbf{u}_t)$. As presented in Sec. 2.3.2, we could employ denoising score matching (DSM) and instead sample $\mathbf{u}_0$, and diffuse those samples, which

would lead to a tractable objective. However, recall that in CLD the distribution at $t{=}0$ is the product of a complex data distribution and a Normal distribution over the initial velocity. Therefore, we propose a hybrid version of score matching [123] and denoising score matching [282], which we call *hybrid score matching* (HSM). In HSM, we draw samples from $p_0(\mathbf{x}_0){=}p_{\text{data}}(\mathbf{x}_0)$ as in DSM, but then diffuse those samples while marginalizing over the full initial velocity distribution $p_0(\mathbf{v}_0){=}\mathcal{N}(\mathbf{v};\mathbf{0}_d,\gamma M\boldsymbol{I}_d)$ as in regular SM (HSM is discussed in detail in App. 2.4.3). Since $p_0(\mathbf{v}_0)$ is Normal (and $\boldsymbol{f}$ and $\boldsymbol{G}$ affine), $p(\mathbf{u}_t|\mathbf{x}_0)$ is also Normal and this remains tractable. We can write this HSM objective as:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\in[0,T]}\mathbb{E}_{\mathbf{x}_0\sim p_0(\mathbf{x}_0)}\mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{x}_0)}\left[\lambda(t)\|s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)-\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t|\mathbf{x}_0)\|_2^2\right]. \tag{2.11}$$

In HSM, the expectation over $p_0(\mathbf{v}_0)$ is essentially solved analytically, while DSM would use a sample-based estimate. Hence, HSM reduces the variance of training objective gradients compared to pure DSM, which we validate in App. 2.4.3.1. Furthermore, when drawing a sample $\mathbf{u}_0$ to diffuse in DSM, we are essentially placing an infinitely sharp Normal with unbounded score [146] at $\mathbf{u}_0$, which requires undesirable modifications or truncation tricks for stable training [263, 280]. Hence, with DSM we could lose some benefits of the CLD framework discussed in Sec. 2.3.3.1, whereas HSM is tailored to CLD and fundamentally avoids such unbounded scores. Closed form expressions for the perturbation kernel $p_t(\mathbf{u}_t|\mathbf{x}_0)$ are provided in App. 2.4.2.1.

Score Model Parametrization. **(i)** Ho et al. [109] found that it can be beneficial to parameterize the score model to predict the noise that was used in the reparametrized sampling to generate perturbed samples $\mathbf{u}_t$. For CLD, $\mathbf{u}_t=\boldsymbol{\mu}_t(\mathbf{x}_0)+\boldsymbol{L}_t\boldsymbol{\epsilon}_{2d}$, where $\boldsymbol{\Sigma}_t=\boldsymbol{L}_t\boldsymbol{L}_t^\top$ is the Cholesky decomposition of $p_t(\mathbf{u}_t|\mathbf{x}_0)$'s covariance matrix, $\boldsymbol{\epsilon}_{2d}\sim\mathcal{N}(\boldsymbol{\epsilon}_{2d};\mathbf{0}_{2d},\boldsymbol{I}_{2d})$, and $\boldsymbol{\mu}_t(\mathbf{x}_0)$ is $p_t(\mathbf{u}_t|\mathbf{x}_0)$'s mean. Furthermore, $\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t|\mathbf{x}_0)=-\ell_t\boldsymbol{\epsilon}_{d:2d}$, where $\boldsymbol{\epsilon}_{d:2d}$ denotes those d components of $\boldsymbol{\epsilon}_{2d}$ that actually affect $\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t|\mathbf{x}_0)$ (since we take velocity gradients only, not all are relevant).

$$\text{With}\quad \boldsymbol{\Sigma}_t=\underbrace{\begin{pmatrix}\Sigma_t^{xx} & \Sigma_t^{xv}\\ \Sigma_t^{xv} & \Sigma_t^{vv}\end{pmatrix}}_{\text{“per-dimension” covariance matrix}}\otimes\boldsymbol{I}_d,\quad \text{we have}\quad \ell_t\coloneqq\sqrt{\frac{\Sigma_t^{xx}}{\Sigma_t^{xx}\Sigma_t^{vv}-(\Sigma_t^{xv})^2}}.$$

(ii) Vahdat et al. [280] showed that it can be beneficial to assume that the diffused marginal distribution is Normal at all times and parametrize the model with a Normal score and a residual “correction”. For CLD, the score is indeed Normal at $t=0$ (due to the independently initialized $\mathbf{x}$ and $\mathbf{v}$ at $t{=}0$). Similarly, the target score is close to Normal for large t, as we approach the equilibrium.

Based on **(i)** and **(ii)**, we parameterize $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = -\ell_t \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ with $\alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = \ell_t^{-1}\mathbf{v}_t/\Sigma_t^{vv} + \alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$, where Σ_t^{vv} corresponds to the v-v component of the "per-dimension" covariance matrix of the Normal distribution $p_t(\mathbf{u}_t|\mathbf{x}_0 = \mathbf{0}_d)$. In other words, we assumed $p_0(\mathbf{x}_0) = \delta(\mathbf{x})$ when defining the analytic term of the score model. Formally, $-\mathbf{v}/\Sigma_t^{vv}$ is the score of a Normal distribution with covariance $\hat{\Sigma}_t^{vv}\boldsymbol{I}_d$. Following [280], we refer to this parameterization as *mixed score parameterization*. Alternative model parameterizations are possible, but we leave their exploration to future work. With this definition, the HSM training objective becomes (details in App. 2.4.2.3):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[0,T]}\mathbb{E}_{\mathbf{x}_0\sim p_0(\mathbf{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}_{2d}\sim\mathcal{N}(\boldsymbol{\epsilon}_{2d};\mathbf{0}_{2d},\boldsymbol{I}_{2d})}\left[\lambda(t)\ell_t^2\|\boldsymbol{\epsilon}_{d:2d} - \alpha_{\boldsymbol{\theta}}(\boldsymbol{\mu}_t(\mathbf{x}_0) + \boldsymbol{L}_t\boldsymbol{\epsilon}_{2d}, t)\|_2^2\right], \tag{2.12}$$

which corresponds to training the model to predict the noise only injected into the velocity during reparametrized sampling of $\mathbf{u}_t$, similar to noise prediction in Ho et al. [109], Song et al. [263].

Objective Weightings. For $\lambda(t) = \Gamma\beta$, the objective corresponds to maximum likelihood learning [262] (see App. 2.4.2.3). Analogously to prior work [109, 262, 280], an objective better suited for high quality image synthesis can be obtained by setting $\lambda(t) = \ell_t^{-2}$, which corresponds to "dropping the variance prefactor" ℓ_t^2.

2.3.3.3 Sampling from CLD-based SGMs

To sample from the CLD-based SGM we can either directly simulate the reverse-time diffusion process (Eq. (2.5)) or, alternatively, solve the corresponding probability flow ODE [262, 263] (see App. 2.4.2.5). To simulate the SDE of the reverse-time diffusion process, previous works often relied on Euler-Maruyama (EM) [154] and related methods [109, 134, 263]. We derive a new solver, tailored to CLD-based models. Here, we provide the high-level ideas and derivations (see App. 2.4.4 for details).

Our generative SDE can be written as (with $\bar{\mathbf{u}}_t = \mathbf{u}_{T-t}$, $\bar{\mathbf{x}}_t = \mathbf{x}_{T-t}$, $\bar{\mathbf{v}}_t = \mathbf{v}_{T-t}$):

$$\begin{pmatrix} d\bar{\mathbf{x}}_t \\ d\bar{\mathbf{v}}_t \end{pmatrix} = \underbrace{\begin{pmatrix} -M^{-1}\bar{\mathbf{v}}_t \\ \bar{\mathbf{x}}_t \end{pmatrix}\beta dt}_{A_H} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\bar{\mathbf{v}}_t \end{pmatrix}\beta dt + \begin{pmatrix} \mathbf{0}_d \\ \sqrt{2\Gamma\beta}d\mathbf{w}_t \end{pmatrix}}_{A_O} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ 2\Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix}\beta dt}_{S}$$

It consists of a Hamiltonian component A_H, an Ornstein-Uhlenbeck process A_O, and the score model term S. We could use EM to integrate this SDE; however, standard Euler methods are not well-suited for Hamiltonian dynamics [167, 209]. Furthermore, if S was 0,

we could solve the SDE in closed form. This suggests the construction of a novel integrator.

We use the Fokker-Planck operator[2] formalism [165, 166, 277]. Using a similar notation as Leimkuhler and Matthews [165], the Fokker-Planck equation corresponding to the generative SDE is $\partial p_t(\bar{\mathbf{u}}_t)/\partial t = (\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*) p_t(\bar{\mathbf{u}}_t)$, where $\hat{\mathcal{L}}_A^*$ and $\hat{\mathcal{L}}_S^*$ are the non-commuting Fokker-Planck operators corresponding to the $A := A_H + A_O$ and S terms, respectively. Expressions for $\hat{\mathcal{L}}_A^*$ and $\hat{\mathcal{L}}_S^*$ can be found in App. 2.4.4. We can construct a formal, but intractable solution of the generative SDE as $\bar{\mathbf{u}}_t = e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)} \bar{\mathbf{u}}_0$, where the operator $e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)}$ (known as the *classical propagator* in statistical physics) propagates states $\bar{\mathbf{u}}_0$ for time t according to the dynamics defined by the combined operators $\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*$. Although this operation is not analytically tractable, it can serve as starting point to derive a practical integrator. Using the symmetric Trotter theorem or Strang splitting formula as well as the Baker–Campbell–Hausdorff formula [266, 275, 277], it can be shown that:

$$e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)} = \lim_{N \to \infty} \left[e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*} e^{\delta t \hat{\mathcal{L}}_S^*} e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*} \right]^N \approx \left[e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*} e^{\delta t \hat{\mathcal{L}}_S^*} e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*} \right]^N + \mathcal{O}(N \delta t^3), \tag{2.13}$$

for large $N \in \mathbb{N}^+$ and time step $\delta t := t/N$. The expression suggests that instead of directly evaluating the intractable $e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)}$, we can discretize the dynamics over t into N pieces of step size δt, such that we only need to apply the *individual* $e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*}$ and $e^{\delta t \hat{\mathcal{L}}_S^*}$ many times one after another for small steps δt. A finer discretization results in a smaller error (since $N = t/\delta t$, the error effectively scales as $\mathcal{O}(\delta t^2)$ for fixed t). Hence, this implies an integration method. Indeed, $e^{\frac{\delta t}{2} \hat{\mathcal{L}}_A^*} \bar{\mathbf{u}}_t$ is available in closed form, as mentioned before; however, $e^{\delta t \hat{\mathcal{L}}_S^*} \bar{\mathbf{u}}_t$ is not. Therefore, we approximate this latter component of the integrator via a standard Euler step. Thus, the integrator formally has an error of the same order as standard EM methods. Nevertheless, as long as the dynamics is not dominated by the S component, our proposed integration scheme is expected to be more accurate than EM, since we split off the analytically tractable part and only use an Euler approximation for the S term. Recall that the model only needs to learn the score of the conditional distribution $p_t(\mathbf{v}_t | \mathbf{x}_t)$, which is close to Normal for much of the diffusion, in which case the S term will indeed be small. This suggests that the generative SDE dynamics are in fact dominated by A_H and A_O in practice. Note that only the propagator $e^{\delta t \hat{\mathcal{L}}_S^*}$ is computationally expensive, as it involves evaluating the neural network. We coin our novel SDE integrator for CLD-based SGMs *Symmetric Splitting CLD Sampler* (SSCS). A detailed derivation, analyses, and a formal algorithm are presented in App. 2.4.4.

[2] The *Fokker-Planck operator* is also known as *Kolmogorov operator*. If the underlying dynamics is fully Hamiltonian, it corresponds to the *Liouville operator* [166, 277].

2.3.4 Related Work

Relations to Statistical Mechanics and Molecular Dynamics. Learning a mapping between a simple, tractable and a complex distribution as in SGMs is inspired by annealed importance sampling [208] and the Jarzynski equality from non-equilibrium statistical mechanics [14, 127–129]. However, after Sohl-Dickstein et al. [257], little attention has been given to the origins of SGMs in statistical mechanics. Intuitively, in SGMs the diffusion process is initialized in a non-equilibrium state $\mathbf{u}_0$ and we would like to bring the system to equilibrium, i.e., the tractable prior distribution, *as quickly and as smoothly as possible* to enable efficient denoising. This "equilibration problem" is a much-studied problem in statistical mechanics, particularly in molecular dynamics, where a molecular system is often simulated in thermodynamic equilibrium. Algorithms to quickly and smoothly bring a system to and maintain at equilibrium are known as *thermostats*. In fact, CLD is inspired by the Langevin thermostat [34]. In molecular dynamics, advanced thermostats are required in particular for "multiscale" systems that show complex behaviors over multiple time- and length-scales. Similar challenges also arise when modeling complex data, such as natural images. Hence, the vast literature on thermostats [6, 35, 41, 42, 116, 121, 192, 215, 277] may be valuable for the development of future SGMs. Also the framework for developing SSCS is borrowed from statistical mechanics. The same techniques have been used to derive molecular dynamics algorithms [34, 42, 158, 165, 166, 276].

Further Related Work. Generative modeling by learning stochastic processes has a long history [4, 23, 28, 99, 109, 189, 206, 256, 259]. We build on Song et al. [263], which introduced the SDE framework for modern SGMs. Nachmani et al. [207] recently introduced non-Gaussian diffusion processes with different noise distributions. However, the noise is still injected directly into the data, and no improved sampling schemes or training objectives are introduced. Vahdat et al. [280] proposed LSGM, which is complementary to CLD: we improve the diffusion process itself, whereas LSGM "simplifies the data" by first embedding it into a smooth latent space. LSGM is an overall more complicated framework, as it is trained in two stages and relies on additional encoder and decoder networks. Recently, techniques to accelerate sampling from pre-trained SGMs have been proposed [156, 245, 258, 285]. Importantly, these methods usually do not permit straightforward log-likelihood estimation. Furthermore, they are originally not based on the continuous time framework, which we use, and have been developed primarily for discrete-step diffusion models. A complementary work to CLD is "Gotta Go Fast" (GGF) [134], which introduces an adaptive SDE solver for SGMs, tuned towards image synthesis. GGF uses standard Euler-based methods under the hood [154, 234], in contrast to our SSCS that is derived from first principles. Furthermore, our SDE integrator for CLD does not make

any data-specific assumptions and performs extremely well even without adaptive step sizes. Some works study SGMs for maximum likelihood training [120, 150, 262]. Note that we did not focus on training our models towards high likelihood. Furthermore, Chen et al. [47] and Huang et al. [119] recently trained augmented Normalizing Flows, which have conceptual similarities with our velocity augmentation. Methods leveraging auxiliary variables similar to our velocities are also used in statistics—such as Hamiltonian Monte Carlo [209]—and have found applications, for instance, in Bayesian machine learning [52, 69, 250]. As shown in Ma et al. [191], our velocity is equivalent to momentum in gradient descent and related methods [148, 225]. Momentum accelerates optimization; the velocity in CLD accelerates mixing in the diffusion process. Lastly, our CLD method can be considered as a second-order Langevin algorithm, but even higher-order schemes are possible [205] and could potentially further improve SGMs.

2.3.5 Experiments

Architectures. We focus on image synthesis and implement CLD-based SGMs using NCSN++ and DDPM++ [263] with 6 input channels (for velocity and data) instead of 3.

Relevant Hyperparameters. CLD's hyperparameters are chosen as β=4, Γ=1 (or equivalently M^{-1}=4) in all experiments. We set the variance scaling of the inital velocity distribution to γ=0.04 and use the proposed HSM objective with the weighting $\lambda(t)=\ell_t^{-2}$, which promotes image quality.

Sampling. We generate model samples via: **(i)** Probability flow using a Runge–Kutta 4(5) method; reverse-time generative SDE sampling using either **(ii)** EM or **(iii)** our SSCS. For methods without adaptive stepsize (EM and SSCS), we use evaluation times chosen according to a quadratic function, like previous work [156, 258, 285] (indicated by QS).

Evaluation. We measure image sample quality for CIFAR-10 via Fréchet inception distance (FID) with 50k samples [105]. We also evaluate an upper bound on the negative log-likelihood (NLL): $-\log p(\mathbf{x}_0) \leq -\mathbb{E}_{\mathbf{v}_0 \sim p(\mathbf{v}_0)} \log p_\varepsilon(\mathbf{x}_0, \mathbf{v}_0) - H$, where H is the entropy of $p(\mathbf{v}_0)$ and $\log p_\varepsilon(\mathbf{x}_0, \mathbf{v}_0)$ is an unbiased estimate of $\log p(\mathbf{x}_0, \mathbf{v}_0)$ from the probability flow ODE [100, 263]. As in Vahdat et al. [280], the stochasticity of $\log p_\varepsilon(\mathbf{x}, \mathbf{v})$ prevents us from performing importance weighted NLL estimation over the velocity distribution [33]. We also record the number of function—neural network—evaluations (NFEs) during synthesis when comparing sampling methods. All implementation details in App. 2.4.2.5 and 2.4.5.

2.3.5.1 Image Generation

Following Song et al. [263], we focus on the widely used CIFAR-10 unconditional image generation benchmark. Our CLD-based SGM achieves an FID of 2.25 based on the probability flow ODE and an FID of 2.23 via simulating the generative SDE (Tab. 2.1). The only models marginally outperforming CLD are LSGM [280] and NSCN++/VESDE with 2,000 step predictor-corrector (PC) sampling [263]. However, LSGM uses a model with $\approx 475M$ parameters to achieve its high performance, while we obtain our numbers with a model of $\approx 100M$ parameters. For a fairer comparison, we trained a smaller LSGM also with $\approx 100M$ parameters, which is reported as "LSGM-100M" in Tab. 2.1 (details in App. 2.4.5.2.7). Our model has a significantly better FID score than "LSGM-100M". In contrast to NSCN++/VESDE, we achieve extremely strong results with much fewer NFEs (for example, see $n \in \{150, 275, 500\}$ in Tab. 2.3 and also Tab. 2.2)—the VESDE performs poorly in this regime. We conclude that when comparing models with similar network capacity and under NFE budgets ≤ 500, our CLD-SGM outperforms all published results in terms of FID. We attribute these positive results to our easier score matching task. Furthermore, our model reaches an NLL bound of 3.31, which is on par with recent works such as Austin et al. [12], Nichol and Dhariwal [212], Vahdat et al. [280] and indicates that our model is not dropping modes. However, our bound is potentially quite loose (see discussion in App. 2.4.2.5) and the true NLL might be significantly lower. We did not focus on training our models towards high likelihood.

To demonstrate that CLD is also suitable for high-resolution image synthesis, we additionally trained a CLD-SGM on CelebA-HQ-256, but without careful hyperparameter tuning due to limited compute resources. Model samples in Fig. 4 appear diverse and high-quality (additional samples in App. 2.4.6).

2.3.5.2 Sampling Speed and Synthesis Quality Trade-Offs

We analyze the sampling speed vs. synthesis quality trade-off for CLD-SGMs and study SSCS's performance under different NFE budgets (Tabs. 2.2 and 2.3). We compare to Song et al. [263] and use EM to solve the generative SDE for their VPSDE and PC (reverse-diffusion + Langevin sampler) for the VESDE model. We also compare to the GGF [134] solver for the generative SDE as well as probability flow ODE sampling with a higher-order adaptive step size solver. Further, we compare to LSGM [280] (using our LSGM-100M), which also uses probability flow sampling. With one exception (VESDE with 2,000 NFE) our CLD-SGM outperforms all baselines, both for adaptive and fixed-step size methods.

Table 2.1: Unconditional CIFAR-10 generative performance.

Class	Model	NLL↓	FID↓
Score	CLD-SGM (Prob. Flow) *(ours)*	≤3.31	2.25
	CLD-SGM (SDE) *(ours)*	-	2.23
Score	DDPM++, VPSDE (Prob. Flow) [263]	3.13	3.08
	DDPM++, VPSDE (SDE) [263]	-	2.41
	DDPM++, sub-VP (Prob. Flow) [263]	2.99	2.92
	DDPM++, sub-VP (SDE) [263]	-	2.41
	NCSN++, VESDE (SDE) [263]	-	2.20
	LSGM [280]	≤3.43	2.10
	LSGM-100M [280]	≤2.96	4.60
	DDPM [109]	≤3.75	3.17
	NCSN [259]	-	25.3
	Adversarial DSM [135]	-	6.10
	Likelihood SDE [262]	2.84	2.87
	DDIM (100 steps) [258]	-	4.16
	FastDDPM (100 steps) [156]	-	2.86
	Improved DDPM [212]	3.37	2.90
	VDM [150]	≤2.49	7.41 (4.00)
	UDM [146]	3.04	2.33
	D3PM [12]	≤3.44	7.34
	Gotta Go Fast [134]	-	2.44
	DDPM Distillation [186]	-	9.36
GANs	SNGAN [203]	-	21.7
	SNGAN+DGflow [10]	-	9.62
	AutoGAN [96]	-	12.4
	TransGAN [131]	-	9.26
	StyleGAN2 w/o ADA [138]	-	8.32
	StyleGAN2 w/ ADA [138]	-	2.92
	StyleGAN2 w/ Diffaug [317]	-	5.79
Aut.-Reg., Flows, VAEs, EBMs	DistAug [136]	2.53	42.90
	PixelCNN [216]	3.14	65.9
	Glow [152]	3.35	48.9
	Residual Flow [51]	3.28	46.37
	NVAE [279]	2.91	23.5
	NCP-VAE [8]	-	24.08
	DC-VAE [221]	-	17.90
	IGEBM [80]	-	40.6
	VAEBM [294]	-	12.2
	Recovery EBM [92]	3.18	9.58

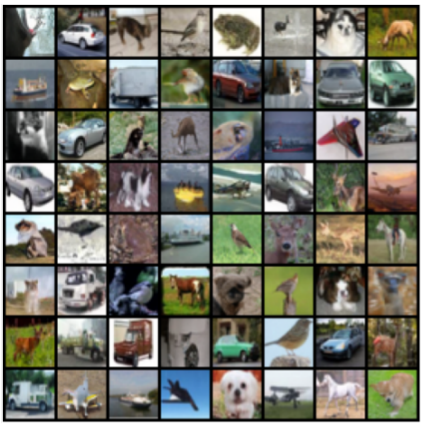


Figure 2.3: CIFAR-10 samples.

Figure 2.4: CelebA-HQ-256 samples.

Table 2.2: *(right)* Performance using adaptive stepsize solvers (ODE is based on probability flow, GGF simulates generative SDE). †: taken from Jolicoeur-Martineau et al. [134]. LSGM corresponds to the small LSGM-100M model for fair comparison (details in App. 2.4.5.2.7). Error tolerances were chosen to obtain similar NFEs.

Model	Solver	NFEs↓	FID↓
CLD	ODE	312	**2.25**
VPSDE	GGF	330	2.56†
VESDE	GGF	488	2.99†
CLD	ODE	147	**2.71**
VPSDE	ODE	141	2.76
VPSDE	GGF	151	2.73†
VESDE	ODE	182	7.63
VESDE	GGF	170	10.15†
LSGM	ODE	131	4.60

Table 2.3: *(bottom)* Performance using non-adaptive stepsize solvers (for PC, QS performed poorly). †: 2.23 FID is our evaluation, Song et al. [263] reports 2.20 FID. See Tab. 2.9 in App. 2.4.6.2 for extended results.

		FID at n function evaluations ↓					
Model	Sampler	n=50	n=150	n=275	n=500	n=1000	n=2000
CLD	EM-QS	52.7	7.00	3.24	2.41	**2.27**	**2.23**
CLD	SSCS-QS	**20.5**	**3.07**	**2.38**	**2.25**	2.30	2.29
VPSDE	EM-QS	28.2	4.06	2.65	2.47	2.66	2.60
VESDE	PC	460	216	11.2	3.75	2.43	**2.23**†

More results in App. 2.4.6.2.

Several observations stand out: **(i)** As expected (Sec. 2.3.3.3), for CLD, SSCS significantly outperforms EM under limited NFE budgets. When using a fine discretization of the SDE (high NFE), the two perform similarly, which is also expected, as the errors of both methods will become negligible. **(ii)** In the adaptive solver setting, using a simpler ODE solver, we even outperform GGF, which is tuned towards image synthesis. **(iii)** Our CLD-SGM also outperforms the LSGM-100M model in terms of FID. It is worth noting, however, that LSGM was designed primarily for faster synthesis, which it achieves by modeling a smooth distribution in latent space instead of the more complex data distribution directly. This suggests that it would be promising to combine LSGM with CLD and train a CLD-based LSGM, combining the strengths of the two approaches. It would also be interesting to develop a more advanced, adaptive SDE solver that leverages SSCS as the backbone and, for example, potentially test our method within a framework like GGF. Our current SSCS only allows for fixed step sizes—nevertheless, it achieves excellent performance.

2.3.5.3 Ablation Studies

We perform ablation studies to study CLD's new hyperparameters (run with a smaller version of our CLD-SGM used above; App. 2.4.5 for details).

Mass Parameter: Tab. 2.4 shows results for a CLD-SGM trained with different M^{-1} (also recall that M^{-1} and Γ are tied together via $\Gamma^2 = 4M$; we are always in the critical-damping regime). Different mass values perform mostly similarly. Intuitively, training with smaller M^{-1} means that noise flows from the velocity variables $\mathbf{v}_t$ into the data $\mathbf{x}_t$ more slowly, which necessitates a larger time rescaling β. We found that simply tying M^{-1} and β together via $\beta{=}8\sqrt{M}$ works well and did not further fine-tune. **Initial Velocity Distribution:** Tab. 2.5 shows results for a CLD-SGM trained with different initial velocity variance scalings γ. Varying γ similarly has only a small effect, but small γ seems slightly beneficial for FID, while the NLL bound suffers a bit. Due to our focus on synthesis quality as measued by FID, we opted for small γ. Intuitively, this means that the data at $t{=}0$ is "at rest", and noise flows from the velocity into the data variables only slowly.

Table 2.4: Mass hyperparameter.

M^{-1}	NLL↓	FID↓
1	≤3.30	3.23
4	≤3.37	3.14
16	≤3.26	3.16

Table 2.5: Initial velocity distribution width.

γ	NLL↓	FID↓
0.04	≤3.37	3.14
0.4	≤3.15	3.21
1	≤3.15	3.27

Mixed Score: Similar to previous work [280], we find training with the mixed score (MS) parametrization (Sec. 2.3.3.2) beneficial. With MS, we achieve an FID of 3.14, without only 3.56.

Hybrid Score Matching: We also tried training with regular DSM, instead of HSM. However, training often became unstable. As discussed in Sec. 2.3.3.2, this is likely because when using standard DSM our CLD would suffer from unbounded scores close to $t{=}0$, similar to previous SDEs [146]. Consequently, we consider our novel HSM a crucial element for training CLD-SGMs.

We conclude that CLD does not come with difficult-to-tune hyperparameters. We expect our chosen hyperparameters to immediately translate to new tasks and models. In fact, we used the same M^{-1}, γ, MS and HSM settings for CIFAR-10 and CelebA-HQ-256 experiments without fine-tuning.

2.3.6 Conclusion

We presented *critically-damped Langevin diffusion*, a novel diffusion process for training SGMs. CLD diffuses the data in a smoother, easier-to-denoise manner compared to previous SGMs, which results in smoother neural network-parametrized score functions, fast synthesis, and improved expressivity. Our experiments show that CLD outperforms previ-

ous SGMs on image synthesis for similar-capacity models and sampling compute budgets, while our novel SSCS is superior to EM in CLD-based SGMs. From a technical perspective, in addition to proposing CLD, we derive CLD's score matching objective termed as HSM, a variant of denoising score matching suited for CLD, and we derive a tailored SDE integrator for CLD. Inspired by methods used in statistical mechanics, our work provides new insights into SGMs and implies promising directions for future research.

We believe that CLD can potentially serve as the backbone diffusion process of next generation SGMs. Future work includes using CLD-based SGMs for generative modeling tasks beyond images, combining CLD with techniques for accelerated sampling from SGMs, adapting CLD-based SGMs towards maximum likelihood, and utilizing other thermostating methods from statistical mechanics.

2.3.7 Ethics and Reproducibility

Our paper focuses on fundamental algorithmic advances to improve the generative modeling performance of SGMs. As such, the proposed CLD does not imply immediate ethical concerns. However, we validate CLD on image synthesis benchmarks. Generative modeling of images has promising applications, for example for digital content creation and artistic expression [15], but can also be used for nefarious purposes [201, 210, 278]. It is worth mentioning that compared to generative adversarial networks [97], a very popular class of generative models, SGMs have the promise to model the data more faithfully, without dropping modes and introducing problematic biases. Generally, the ethical impact of our work depends on its application domain and the task at hand.

To aid reproducibility of the results and methods presented in our paper, we made source code to reproduce the main results of the paper publicly available, including detailed instructions; see our project page https://nv-tlabs.github.io/CLD-SGM and the code repository https://github.com/nv-tlabs/CLD-SGM. Furthermore, all training details and hyperparameters are already in detail described in the Appendix, in particular in App. 2.4.5.

2.4 Appendix

2.4.1 Langevin Dynamics

Here, we discuss different aspects of Langevin dynamics. Recall the Langevin dynamics, Eq. (2.8), from the main paper:

$$\begin{pmatrix} d\mathbf{x}_t \\ d\mathbf{v}_t \end{pmatrix} = \underbrace{\begin{pmatrix} M^{-1}\mathbf{v}_t \\ -\mathbf{x}_t \end{pmatrix} \beta dt}_{\text{Hamiltonian component}=:H} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\mathbf{v}_t \end{pmatrix} \beta dt + \begin{pmatrix} 0 \\ \sqrt{2\Gamma\beta} \end{pmatrix} d\mathbf{w}_t}_{\text{Ornstein-Uhlenbeck process}=:O}. \tag{2.14}$$

2.4.1.1 Different Damping Ratios

As discussed in Sec. 2.3.3, Langevin dynamics can be run with different ratios between mass M and squared friction Γ^2. To recap from the main paper:

(i) For $\Gamma^2 < 4M$ (*underdamped* Langevin dynamics), the Hamiltonian component dominates, which implies oscillatory dynamics of $\mathbf{x}_t$ and $\mathbf{v}_t$ that slow down convergence to equilibrium.

(ii) For $\Gamma^2 > 4M$ (*overdamped* Langevin dynamics), the O-term dominates which also slows down convergence, since the accelerating effect by the Hamiltonian component is suppressed due to the strong noise injection.

(iii) For $\Gamma^2 = 4M$ (*critical-damping*), an ideal balance is achieved and convergence to $p_{\text{EQ}}(\mathbf{u})$occurs quickly in a smooth manner without oscillations.

In Fig. 2.5, we visualize diffusion trajectories according to Langevin dynamics run in the different damping regimes. We observe that underdamped Langevin dynamics show undesired oscillatory behavior, while overdamped Langevin dynamics perform very inefficiently, too. Critical-damping achieves a good balance between the two and mixes and converges quickly. In fact, it can be shown to be optimal in terms of convergence; see, for example, McCall [193].

Consequently, we propose to set $\Gamma^2 = \Gamma^2_{\text{critical}} := 4M$ in CLD.

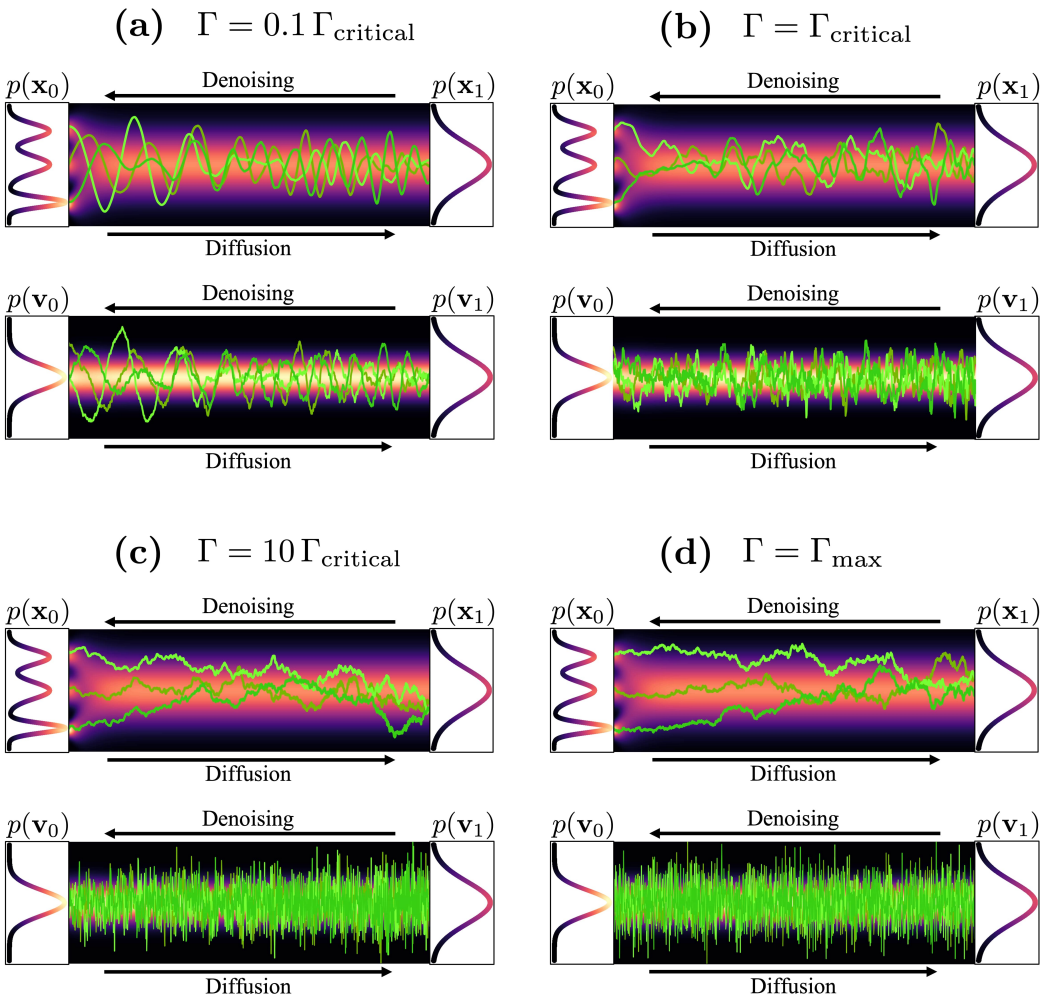


Figure 2.5: Langevin dynamics in different damping regimes. Each pair of visualizations corresponds to the (coupled) evolution of data $\mathbf{x}_t$ and velocities $\mathbf{v}_t$. We show the marginal **(red) probabilities** and the projections of the **(green) trajectories**. The probabilities always correspond to the same optimal setting $\Gamma = \Gamma_{\text{critical}}$ (recall that $\Gamma_{\text{critical}} = 2\sqrt{M}$ and $\Gamma_{\text{max}} = M/(\beta(t)\delta t)$; see Sec. 2.4.1.2). The trajectories correspond to different Langevin trajectories run in the different regimes with indicated friction coefficients Γ. We see in **(b)**, that for *critical damping* the $\mathbf{x}_t$ trajectories quickly explore the space and converge according to the distribution indicated by the underlying probability. In the *under-damped regime* **(a)**, even though the trajectories mix quickly we observe undesired oscillatory behavior. For *over-damped Langevin dynamics*, **(c)** and **(d)**, the $\mathbf{x}_t$ trajectories mix and converge only very slowly. Note that the visualized diffusion uses different hyperparameters compared to the diffusion shown in Fig. 2.1 in the main text: Here, we have chosen a much larger β, such that also the slow overdamped Langevin dynamics trajectories shown here mix a little bit over the visualized diffusion time (while the probability distribution and the trajectories for critical damping converge almost instantly).

2.4.1.2 Very High Friction Limit and Connections to previous SDEs in SGMs

Let us re-write the above Langevin dynamics and consider the more general case with time-dependent $\beta(t)$:

$$d\mathbf{x}_t = M^{-1}\mathbf{v}_t\beta(t)dt, \tag{2.15}$$

$$d\mathbf{v}_t = -\underbrace{\mathbf{x}_t\beta(t)dt}_{\textbf{(ii)}\text{: potential term}} - \underbrace{\Gamma M^{-1}\mathbf{v}_t\beta(t)dt}_{\textbf{(iii)}\text{: friction term}} + \underbrace{\sqrt{2\Gamma\beta(t)}d\mathbf{w}_t}_{\textbf{(iv)}\text{: noise term}}. \tag{2.16}$$

To solve this SDE, let us assume a simple Euler-based integration scheme, with the update equation for a single step at time t (this integration scheme would not be optimal, as discussed in Sec. 2.3.3.3., however, it would be accurate for sufficiently small time steps and we just need this to make the connection to previous works like the VPSDE):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \beta(t)M^{-1}\mathbf{v}_{n+1}\delta t, \tag{2.17}$$

$$\mathbf{v}_{n+1} = \underbrace{\mathbf{v}_n}_{\textbf{(i)}\text{: current step velocity}} - \underbrace{\beta(t)\mathbf{x}_n\delta t}_{\textbf{(ii)}\text{: potential term}} - \underbrace{\beta(t)\Gamma M^{-1}\mathbf{v}_n\delta t}_{\textbf{(iii)}\text{: friction term}} + \underbrace{\sqrt{2\beta(t)\Gamma}\mathcal{N}(\mathbf{0}_d, \delta t\boldsymbol{I}_d)}_{\textbf{(iv)}\text{: noise term}}, \tag{2.18}$$

Now, let us assume a friction coefficient $\Gamma = \Gamma_{\text{max}} := \frac{M}{\beta(t)\delta t}$. Since the time step δt is usually very small, this correspond to a very high friction. In fact, it can be considered the maximum friction limit, at which the friction is so large that the current step velocity **(i)** is completely cancelled out by the friction term **(iii)**. We obtain:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \beta(t)M^{-1}\mathbf{v}_{n+1}\delta t \tag{2.19}$$

$$\mathbf{v}_{n+1} = -\beta(t)\mathbf{x}_t\delta t + \sqrt{2\frac{M}{\delta t}}\mathcal{N}(\mathbf{0}_d, \delta t\boldsymbol{I}_d). \tag{2.20}$$

Now the velocity update, Eq. (2.20), does not depend on the current step velocity on the right-hand-side anymore. Hence, we can insert Eq. (2.20) directly into Eq. (2.19) and obtain:

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n - \beta(t)^2M^{-1}\mathbf{x}_n\delta t^2 + \sqrt{2\beta(t)^2\delta tM^{-1}}\mathcal{N}(\mathbf{0}_d, \delta t\boldsymbol{I}_d) \\ &= \mathbf{x}_n - \beta(t)^2M^{-1}\mathbf{x}_n\delta t^2 + \sqrt{2\beta(t)^2\delta t^2M^{-1}}\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d).\end{aligned} \tag{2.21}$$

Re-defining $\delta t' := \delta t^2$ and $\beta'(t) := \beta(t)^2$, we obtain

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \beta'(t)M^{-1}\mathbf{x}_n\delta t' + \sqrt{2\beta'(t)\delta t'M^{-1}}\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d), \tag{2.22}$$

which corresponds to the high-friction overdamped Langevin dynamics that are frequently run, for example, to train energy-based generative models [80, 294]. Let's further absorb the mass M^{-1} and the time step $\delta t'$ into the time rescaling, defining $\hat{\beta}(t) := 2\beta'(t)M^{-1}\delta t'$. We obtain:

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n - \frac{1}{2}\hat{\beta}(t)\mathbf{x}_n + \sqrt{\hat{\beta}(t)}\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d) \\ &= (1 - \frac{1}{2}\hat{\beta}(t))\mathbf{x}_n + \sqrt{\hat{\beta}(t)}\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d) \\ &\approx \sqrt{1 - \hat{\beta}(t)}\mathbf{x}_n + \sqrt{\hat{\beta}(t)}\mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d),\end{aligned} \tag{2.23}$$

where the last approximation is true for sufficiently small $\hat{\beta}(t)$. However, this expression corresponds to

$$\mathbf{x}_{n+1} \sim \mathcal{N}(\mathbf{x}_{n+1}; \sqrt{1 - \hat{\beta}(t)}\mathbf{x}_n, \hat{\beta}(t)\boldsymbol{I}_d) \tag{2.24}$$

which is exactly the transition kernel of the VPSDE's Markov chain [109, 263]. We see that the VPSDE corresponds to the high-friction limit of a more general Langevin dynamics-based diffusion process of the form of Eq. (2.14).

If we assume a diffusion as above but with the potential term **(ii)** set to 0, we can similarly derive the VESDE [263] as a high-friction limit of the corresponding diffusion. Generally, all previously used diffusions that inject noise directly into the data variables correspond to such high-friction diffusions.

In conclusion, we see that previous high-friction diffusions require an excessive amount of noise to be injected to bring the dynamics to the prior, which intuitively makes denoising harder. For our CLD in the critical damping regime we can run the diffusion for a much shorter time or, equivalently, can inject less noise to converge to the equilibrium, i.e., the prior.

2.4.2 Critically-Damped Langevin Diffusion

Here, we present further details about our proposed critically-damped Langevin diffusion (CLD). We provide the derivations and formulas that were not presented in the main paper in the interest of brevity.

2.4.2.1 Perturbation Kernel

To recap from the main text, in this work we propose to augment the data $\mathbf{x}_t \in \mathbb{R}^d$ with auxiliary *velocity* variables $\mathbf{v}_t \in \mathbb{R}^d$. We then run the following diffusion process in the joint $\mathbf{x}_t$-$\mathbf{v}_t$-space

$$d\mathbf{u}_t \coloneqq \begin{pmatrix} d\mathbf{x}_t \\ d\mathbf{v}_t \end{pmatrix} = \boldsymbol{f}(\mathbf{u}_t, t)dt + \boldsymbol{G}(\mathbf{u}_t, t)d\mathbf{w}_t \tag{2.25}$$

$$\boldsymbol{f}(\mathbf{u}_t, t) = (f(t) \otimes \boldsymbol{I}_d)\mathbf{u}_t, \quad f(t) \coloneqq \begin{pmatrix} 0 & \beta(t)M^{-1} \\ -\beta(t) & -\beta(t)\Gamma M^{-1} \end{pmatrix}, \tag{2.26}$$

$$\boldsymbol{G}(\mathbf{u}_t, t) = G(t) \otimes \boldsymbol{I}_d, \quad G(t) \coloneqq \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{2\Gamma\beta(t)} \end{pmatrix}, \tag{2.27}$$

where $\mathbf{w}_t$ is a standard Wiener process in $\mathbb{R}^{2d}$ and $\beta\colon [0, T] \to \mathbb{R}_0^+$ is a time rescaling.[3] In particular, we consider the *critically-damped Langevin diffusion* which can be obtained by setting $M = \Gamma^2/4$, resulting in the following drift kernel

$$\boldsymbol{f}_{\text{CLD}}(\mathbf{u}_t, t) = (f_{\text{CLD}}(t) \otimes \boldsymbol{I}_d)\mathbf{u}_t, \quad f_{\text{CLD}}(t) \coloneqq \begin{pmatrix} 0 & 4\beta(t)\Gamma^{-2} \\ -\beta(t) & -4\beta(t)\Gamma^{-1} \end{pmatrix}. \tag{2.28}$$

Since we only consider the critically-damped case in this work, we redefine $\boldsymbol{f} \coloneqq \boldsymbol{f}_{\text{CLD}}$ and $f \coloneqq f_{\text{CLD}}$ for simplicity. Since our drift $\boldsymbol{f}$ and diffusion $\boldsymbol{G}$ coefficients are affine, $\mathbf{u}_t$ is Normally distributed for all $t \in [0, T]$ if $\mathbf{u}_0$ is Normally distributed at $t = 0$ [247]. In particular, given that $\mathbf{u}_0 \sim \mathcal{N}(\mathbf{u}_0; \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0 = \Sigma_0 \otimes \boldsymbol{I}_d)$, where $\Sigma_0 = \text{diag}(\Sigma_0^{xx}, \Sigma_0^{vv})$ is a positive semi-definite diagonal 2-by-2 matrix (we restrict our derivation to diagonal covariance matrices at $t = 0$ for simplicity, since in our situation velocity and data are generally independent at $t = 0$), we derive expressions for $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, the mean and the covariance matrix of $\mathbf{u}_t$, respectively.

Following Särkkä and Solin [247] (Section 6.1), the mean and covariance matrix of $\mathbf{u}_t$ obey the following respective ordinary differential equations (ODEs)

$$\frac{d\boldsymbol{\mu}_t}{dt} = (f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\mu}_t, \tag{2.29}$$

$$\frac{d\boldsymbol{\Sigma}_t}{dt} = (f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\Sigma}_t + [(f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\Sigma}_t]^\top + \left(G(t)G(t)^\top\right) \otimes \boldsymbol{I}_d. \tag{2.30}$$

[3] For our experiments, we only used constant β; however, for generality, we present all derivations for time-dependent $\beta(t)$.

Notating $\boldsymbol{\mu}_0 = (\mathbf{x}_0, \mathbf{v}_0)^\top$, the solutions to the above ODEs are

$$\boldsymbol{\mu}_t = \begin{pmatrix} 2\mathcal{B}(t)\Gamma^{-1}\mathbf{x}_0 + 4\mathcal{B}(t)\Gamma^{-2}\mathbf{v}_0 + \mathbf{x}_0 \\ -\mathcal{B}(t)\mathbf{x}_0 - 2\mathcal{B}(t)\Gamma^{-1}\mathbf{v}_0 + \mathbf{v}_0 \end{pmatrix} e^{-2\mathcal{B}(t)\Gamma^{-1}}, \tag{2.31}$$

and

$$\boldsymbol{\Sigma}_t = \Sigma_t \otimes \boldsymbol{I}_d, \tag{2.32}$$

$$\Sigma_t = \begin{pmatrix} \Sigma_t^{xx} & \Sigma_t^{xv} \\ \Sigma_t^{xv} & \Sigma_t^{vv} \end{pmatrix} e^{-4\mathcal{B}(t)\Gamma^{-1}}, \tag{2.33}$$

$$\Sigma_t^{xx} = \Sigma_0^{xx} + e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 + 4\mathcal{B}(t)\Gamma^{-1}\left(\Sigma_0^{xx} - 1\right) + 4\mathcal{B}^2(t)\Gamma^{-2}\left(\Sigma_0^{xx} - 2\right) + 16\mathcal{B}(t)^2\Gamma^{-4}\Sigma_0^{vv}, \tag{2.34}$$

$$\Sigma_t^{xv} = -\mathcal{B}(t)\Sigma_0^{xx} + 4\mathcal{B}(t)\Gamma^{-2}\Sigma_0^{vv} - 2\mathcal{B}^2(t)\Gamma^{-1}\left(\Sigma_0^{xx} - 2\right) - 8\mathcal{B}^2(t)\Gamma^{-3}\Sigma_0^{vv}, \tag{2.35}$$

$$\Sigma_t^{vv} = \tfrac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma + \Sigma_0^{vv}\left(1 + 4\mathcal{B}(t)^2\Gamma^{-2} - 4\mathcal{B}(t)\Gamma^{-1}\right) + \mathcal{B}(t)^2\left(\Sigma_0^{xx} - 2\right), \tag{2.36}$$

where $\mathcal{B}(t) = \int_0^t \beta(\hat{t})\, d\hat{t}$. For constant $\beta(t) = \beta$ (as is used in all our experiments), we simply have $\mathcal{B}(t) = t\beta$. The correctness of the proposed mean and covariance matrix can be verified by simply plugging them back into their respective ODEs; see App. 2.4.7.1.

With the above derivations, we can find analytical expressions for the perturbation kernel $p(\mathbf{u}_t|\cdot)$. For example, when conditioning on initial data and velocity samples $\mathbf{x}_0$ and $\mathbf{v}_0$ (as in denoising score matching (DSM)), the mean and covariance matrix of the perturbation kernel $p(\mathbf{u}_t|\mathbf{u}_0)$ can be obtained by setting $\boldsymbol{\mu}_0 = (\mathbf{x}_0, \mathbf{v}_0)^\top$, $\Sigma_0^{xx} = 0$, and $\Sigma_0^{vv} = 0$.

In our experiments, the initial velocity distribution is set to $\mathcal{N}(\mathbf{0}_d, \gamma M \boldsymbol{I}_d)$. Conditioning only on initial data samples $\mathbf{x}_0$ and marginalizing over the full initial velocity distribution (as in our hybrid score matching (HSM), see Sec. 2.4.3), the mean and covariance matrix of the perturbation kernel $p(\mathbf{u}_t|\mathbf{x}_0)$ can be obtained by setting $\boldsymbol{\mu}_0 = (\mathbf{x}_0, \mathbf{0}_d)^\top$, $\Sigma_0^{xx} = 0$, and $\Sigma_0^{vv} = \gamma M$.

2.4.2.2 Convergence and Equilibrium

Our CLD-based training of SGMs—as well as denoising diffusion models more generally—relies on the fact that the diffusion converges towards an analytically tractable equilibrium distribution for sufficiently large t. In fact, from the above equations we can easily see

that,

$$\lim_{t\to\infty} \Sigma_t^{xx} = 1, \tag{2.37}$$

$$\lim_{t\to\infty} \Sigma_t^{xv} = 0, \tag{2.38}$$

$$\lim_{t\to\infty} \Sigma_t^{vv} = \frac{\Gamma^2}{4} = M, \tag{2.39}$$

$$\lim_{t\to\infty} \boldsymbol{\mu}_t = \mathbf{0}_{2d}, \tag{2.40}$$

which establishes $p_{\mathrm{EQ}}(\mathbf{u}) = \mathcal{N}(\mathbf{x}; \mathbf{0}_d, \boldsymbol{I}_d)\,\mathcal{N}(\mathbf{v}; \mathbf{0}_d, M\boldsymbol{I}_d)$.

Notice that our CLD is an instantiation of the more general Langevin dynamics defined by

$$\begin{pmatrix} d\mathbf{x}_t \\ d\mathbf{v}_t \end{pmatrix} = \begin{pmatrix} M^{-1}\mathbf{v}_t \\ \nabla_{\mathbf{x}_t} \log p_{\mathrm{pot}}(\mathbf{x}_t) \end{pmatrix} \beta dt + \begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\mathbf{v}_t \end{pmatrix} \beta dt + \begin{pmatrix} 0 \\ \sqrt{2\Gamma\beta} \end{pmatrix} d\mathbf{w}_t. \tag{2.41}$$

which has the equilibrium distribution $\hat{p}_{\mathrm{EQ}}(\mathbf{u}) = p_{\mathrm{pot}}(\mathbf{x})\mathcal{N}(\mathbf{v}; \mathbf{0}_d, M\boldsymbol{I}_d)$ [166, 277]. However, the perturbation kernel of this Langevin dynamics is not available analytically anymore for arbitrary $p_{\mathrm{pot}}(\mathbf{x})$. In our case, however, we have the analytically tractable $p_{\mathrm{pot}}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}_d, \boldsymbol{I}_d)$. Note that this corresponds to the classical "harmonic oscillator" problem from physics.

2.4.2.3 CLD Objective

To derive the objective for training CLD-based SGMs, we start with a derivation that targets maximum likelihood training in a similar fashion to Song et al. [262]. Let p_0 and q_0 be two densities, then

$$\begin{aligned} D_{\mathrm{KL}}(p_0 \parallel q_0) &= D_{\mathrm{KL}}(p_0 \parallel q_0) - D_{\mathrm{KL}}(p_T \parallel q_T) + D_{\mathrm{KL}}(p_T \parallel q_T) \\ &= -\int_0^T \frac{\partial D_{\mathrm{KL}}(p_t \parallel q_t)}{\partial t} dt + D_{\mathrm{KL}}(p_T \parallel q_T), \end{aligned} \tag{2.42}$$

where p_t and q_t are the marginal densities of p_0 and q_0, respectively, diffused by our critically-damped Langevin diffusion. As has been shown in Song et al. [262], Eq. (2.42) can be written as a mixture (over t) of score matching losses. To this end, let us consider the Fokker–Planck equation associated with the critically-damped Langevin diffusion:

$$\begin{aligned} \frac{\partial p_t(\mathbf{u}_t)}{\partial t} &= \nabla_{\mathbf{u}_t} \cdot \left[\tfrac{1}{2}\left(G(t)G(t)^\top \otimes \boldsymbol{I}_d\right) \nabla_{\mathbf{u}_t} p_t(\mathbf{u}_t) - p_t(\mathbf{u}_t)(f(t) \otimes \boldsymbol{I}_d)\mathbf{u}_t\right] \\ &= \nabla_{\mathbf{u}_t} \cdot \left[\boldsymbol{h}_p(\mathbf{u}_t, t) p_t(\mathbf{u}_t)\right], \quad \boldsymbol{h}_p(\mathbf{u}_t, t) \coloneqq \tfrac{1}{2}\left(G(t)G(t)^\top \otimes \boldsymbol{I}_d\right) \nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t) - (f(t) \otimes \boldsymbol{I}_d)\mathbf{u}_t. \end{aligned} \tag{2.43}$$

Similarly, we have $\frac{\partial q_t(\mathbf{u}_t)}{\partial t} = \nabla_{\mathbf{u}_t} \cdot [\boldsymbol{h}_q(\mathbf{u}_t, t) q_t(\mathbf{u}_t)]$. Assuming $\log p_t(\mathbf{u}_t)$ and $\log q_t(\mathbf{u}_t)$ are smooth functions with at most polynomial growth at infinity, we have

$$\lim_{\mathbf{u}_t \to \infty} \boldsymbol{h}_p(\mathbf{u}_t, t) p_t(\mathbf{u}_t) = \lim_{\mathbf{u}_t \to \infty} \boldsymbol{h}_q(\mathbf{u}_t, t) q_t(\mathbf{u}_t) = 0. \tag{2.44}$$

Using the above fact, we can compute the time-derivative of the Kullback–Leibler divergence between p_t and q_t as

$$\begin{aligned}
\frac{\partial D_{\mathrm{KL}}(p_t \parallel q_t)}{\partial t} &= \frac{\partial}{\partial t} \int p_t(\mathbf{u}_t) \log \frac{p_t(\mathbf{u}_t)}{q_t(\mathbf{u}_t)} \, d\mathbf{u}_t \\
&= -\int p_t(\mathbf{u}_t) \left[\boldsymbol{h}_p(\mathbf{u}_t, t) - \boldsymbol{h}_q(\mathbf{u}_t, t)\right]^\top \left[\nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{u}_t} \log q_t(\mathbf{u}_t)\right] \, d\mathbf{u}_t \\
&= -\frac{1}{2} \int p_t(\mathbf{u}_t) \left[\nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{u}_t} \log q_t(\mathbf{u}_t)\right]^\top \left(G(t) G(t)^\top \otimes \boldsymbol{I}_d\right) \left[\nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t)\right. \\
&\qquad \left. - \nabla_{\mathbf{u}_t} \log q_t(\mathbf{u}_t)\right] d\mathbf{u}_t \\
&= -\beta(t) \Gamma \int p_t(\mathbf{u}_t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2 \, d\mathbf{u}_t.
\end{aligned} \tag{2.45}$$

Notice that due to the form of $G(t)$, we now have only gradients with respect to the velocity component $\mathbf{v}_t$. Combining the above with Eq. (2.42), we have

$$\begin{aligned}
D_{\mathrm{KL}}(p_0 \parallel q_0) &= \mathbb{E}_{t \sim \mathcal{U}[0,T], \mathbf{u}_t \sim p_t(\mathbf{u})} \left[\Gamma \beta(t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2\right] + D_{\mathrm{KL}}(p_T \parallel q_T) \\
&\approx \mathbb{E}_{t \sim \mathcal{U}[0,T], \mathbf{u}_t \sim p_t(\mathbf{u})} \left[\Gamma \beta(t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2\right],
\end{aligned} \tag{2.46}$$

Note that the approximation holds if p_T is sufficiently "close" to q_T. We obtain a more general objective function by replacing $\Gamma \beta(t)$ with an arbitrary function $\lambda(t)$, i.e.,

$$\mathbb{E}_{t \sim \mathcal{U}[0,T], \mathbf{u}_t \sim p_t(\mathbf{u})} \left[\lambda(t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2\right] \tag{2.47}$$

As shown in App. 2.4.3, the above can be rewritten, up to irrelevant constant terms, as either of the following two objectives:

$$\mathrm{HSM}(\lambda(t)) \coloneqq \mathbb{E}_{t \sim \mathcal{U}[0,T], \mathbf{x}_0 \sim p_0(\mathbf{x}_0), \mathbf{u}_t \sim p_t(\mathbf{u}_t \mid \mathbf{x}_0)} \left[\lambda(t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{x}_0) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2\right], \tag{2.48}$$

$$\mathrm{DSM}(\lambda(t)) \coloneqq \mathbb{E}_{t \sim \mathcal{U}[0,T], \mathbf{u}_0 \sim p_0(\mathbf{u}_0), \mathbf{u}_t \sim p_t(\mathbf{u}_t \mid \mathbf{u}_0)} \left[\lambda(t) \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{u}_0) - \nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)\|_2^2\right]. \tag{2.49}$$

For both HSM and DSM, we have shown in App. 2.4.2.1 that the perturbation kernels $p_t(\mathbf{u}_t \mid \mathbf{x}_0)$ and $p_t(\mathbf{u}_t \mid \mathbf{u}_0)$ are Normal distributions with the following structure of the covariance matrix:

$$\boldsymbol{\Sigma}_t = \Sigma_t \otimes \boldsymbol{I}_d, \quad \Sigma_t = \begin{pmatrix} \Sigma_t^{xx} & \Sigma_t^{xv} \\ \Sigma_t^{xv} & \Sigma_t^{vv} \end{pmatrix}. \tag{2.50}$$

We can use this fact to compute the gradient $\nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t \mid \cdot)$

$$\begin{aligned} \nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t \mid \cdot) &= -\nabla_{\mathbf{u}_t} \tfrac{1}{2}(\mathbf{u}_t - \boldsymbol{\mu}_t)\boldsymbol{\Sigma}_t^{-1}(\mathbf{u}_t - \boldsymbol{\mu}_t) \\ &= -\boldsymbol{\Sigma}_t^{-1}(\mathbf{u}_t - \boldsymbol{\mu}_t) \\ &= -\boldsymbol{L}_t^{-\top}\boldsymbol{L}_t^{-1}(\mathbf{u}_t - \boldsymbol{\mu}_t) \\ &= -\boldsymbol{L}_t^{-\top}\boldsymbol{\epsilon}_{2d}, \end{aligned} \tag{2.51}$$

where $\boldsymbol{\epsilon}_{2d} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{2d})$ and $\boldsymbol{\Sigma}_t = \boldsymbol{L}_t\boldsymbol{L}_t^\top$ is the Cholesky factorization of the covariance matrix $\boldsymbol{\Sigma}_t$. Note that the structure of $\boldsymbol{\Sigma}_t$ implies that $\boldsymbol{L}_t = L_t \otimes \boldsymbol{I}_d$, where $L_t L_t^\top$ is the Cholesky factorization of Σ_t, i.e,

$$L_t = \begin{pmatrix} L_t^{xx} & L_t^{xv} \\ L_t^{xv} & L_t^{vv} \end{pmatrix} = \begin{pmatrix} \sqrt{\Sigma_t^{xx}} & 0 \\ \frac{\Sigma_t^{xv}}{\sqrt{\Sigma_t^{xx}}} & \sqrt{\frac{\Sigma_t^{xx}\Sigma_t^{vv} - (\Sigma_t^{xv})^2}{\Sigma_t^{xx}}} \end{pmatrix}. \tag{2.52}$$

Furthermore, we have

$$\begin{aligned} \boldsymbol{L}_t^{-\top} &= L_t^{-\top} \otimes \boldsymbol{I}_d \\ &= \begin{pmatrix} \sqrt{\Sigma_t^{xx}} & \frac{\Sigma_t^{xv}}{\sqrt{\Sigma_t^{xx}}} \\ 0 & \sqrt{\frac{\Sigma_t^{xx}\Sigma_t^{vv} - (\Sigma_t^{xv})^2}{\Sigma_t^{xx}}} \end{pmatrix}^{-1} \otimes \boldsymbol{I}_d \\ &= \begin{pmatrix} \frac{1}{\sqrt{\Sigma_t^{xx}}} & \frac{-\Sigma_t^{xz}}{\sqrt{\Sigma_t^{xx}}\sqrt{\Sigma_t^{xx}\Sigma_t^{zz} - (\Sigma_t^{xv})^2}} \\ 0 & \sqrt{\frac{\Sigma_t^{xx}}{\Sigma_t^{xx}\Sigma_t^{vv} - (\Sigma_t^{xv})^2}} \end{pmatrix} \otimes \boldsymbol{I}_d. \end{aligned} \tag{2.53}$$

Using the above, we can compute

$$\begin{aligned} \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \cdot) &= [\nabla_{\mathbf{u}_t} \log p_t(\mathbf{u}_t \mid \cdot)]_{d:2d} \\ &= \left[-\boldsymbol{L}_t^{-\top}\boldsymbol{\epsilon}_{2d}\right]_{d:2d} \\ &= -\ell_t \boldsymbol{\epsilon}_{d:2d}, \end{aligned} \tag{2.54}$$

where

$$\ell_t := \sqrt{\frac{\Sigma_t^{xx}}{\Sigma_t^{xx}\Sigma_t^{vv} - \left(\Sigma_t^{xv}\right)^2}}, \tag{2.55}$$

and $\boldsymbol{\epsilon}_{d:2d}$ denotes those (latter) d components of $\boldsymbol{\epsilon}_{2d}$ that actually affect $\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t|\cdot)$.

Note that ℓ_t depends on the conditioning in the perturbation kernel, and therefore ℓ_t is different for DSM, which is based on $p(\mathbf{u}_t \mid \mathbf{u}_0)$, and HSM, which is based on $p(\mathbf{u}_t \mid \mathbf{x}_0)$. Therefore, we will henceforth refer to ℓ_t^{HSM} and ℓ_t^{DSM} if distinction of the two cases is necessary (otherwise we will simply refer to ℓ_t for both).

As discussed in Section 2.3.3.2, we model $\nabla_{\mathbf{v}_t} \log q_t(\mathbf{u}_t)$ as $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = -\ell_t \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$. Plugging everything back into our objective functions, Eq. (2.48) and Eq. (2.49), we obtain

$$\text{HSM}(\lambda(t)) = \mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{x}_0\sim p_0(\mathbf{x}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{x}_0)} \left[\lambda(t) \left(\ell_t^{\text{HSM}}\right)^2 \|\boldsymbol{\epsilon}_{d:2d} - \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2\right], \tag{2.56}$$

$$\text{DSM}(\lambda(t)) = \mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{u}_0\sim p_0(\mathbf{u}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{u}_0)} \left[\lambda(t) \left(\ell_t^{\text{DSM}}\right)^2 \|\boldsymbol{\epsilon}_{d:2d} - \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2\right], \tag{2.57}$$

where $\mathbf{u}_t$ is sampled via reparameterization:

$$\mathbf{u}_t = \boldsymbol{\mu}_t + \boldsymbol{L}_t \boldsymbol{\epsilon} = \boldsymbol{\mu}_t + \begin{pmatrix} L_t^{xx}\boldsymbol{\epsilon}_{0:d} \\ L_t^{xv}\boldsymbol{\epsilon}_{0:d} + L_t^{vv}\boldsymbol{\epsilon}_{d:2d} \end{pmatrix}. \tag{2.58}$$

Note again that L_t is different for HSM and DSM.

Analogously to prior work [109, 262, 280] an objective better suited for high quality image synthesis can be obtained by "dropping the variance prefactor":

$$\text{HSM}\left(\lambda(t) = \left(\ell_t^{\text{HSM}}\right)^{-2}\right) = \mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{x}_0\sim p_0(\mathbf{x}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{x}_0)} \left[\|\boldsymbol{\epsilon}_{d:2d} - \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2\right], \tag{2.59}$$

$$\text{DSM}\left(\lambda(t) = \left(\ell_t^{\text{DSM}}\right)^{-2}\right) = \mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{u}_0\sim p_0(\mathbf{u}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{u}_0)} \left[\|\boldsymbol{\epsilon}_{d:2d} - \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2\right]. \tag{2.60}$$

2.4.2.4 CLD-specific Implementation Details

Analytically, ℓ_t^{HSM} is bounded (in particular, $\ell_0^{\text{HSM}} = 1/\sqrt{\gamma M}$), whereas ℓ_t^{DSM} is diverging for $t \to 0$. In practice, however, we found that computation of ℓ_t^{HSM} can also be numerically unstable, even when using double precision. As is common practice for computing Cholesky

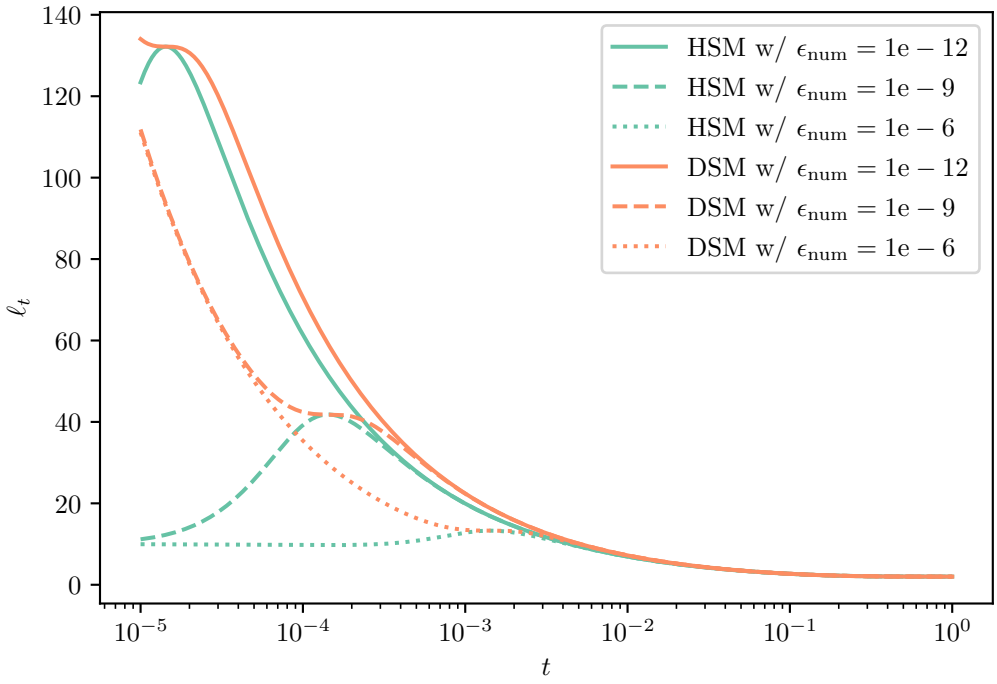


Figure 2.6: Comparison of ℓ_t^{HSM} (in green) and ℓ_t^{DSM} (in orange) for our main hyperparameter setting with $M = 0.25$ and $\gamma = 0.04$. In contrast to ℓ_t^{DSM}, ℓ_t^{HSM} is analytically bounded. Nevertheless, numerical computation can be unstable (even when using double precision) in which case adding a numerical stabilization of $\epsilon_{\text{num}} = 10^{-9}$ to the covariance matrix before computing ℓ_t suffices to make HSM work (see App. 2.4.2.4).

decompositions, we add a numerical stabilization matrix $\epsilon_{\text{num}}\boldsymbol{I}_{2d}$ to $\boldsymbol{\Sigma}_t$ before computing ℓ_t. In Fig. 2.6, we visualize ℓ_t^{HSM} and ℓ_t^{DSM} for different values of ϵ_{num} using our main experimental setup of $M = 0.25$ and $\gamma = 0.04$ (also, recall that in practice we have $T = 1$). Note that a very small numerical stabilization of $\epsilon_{\text{num}} = 10^{-9}$ in combination with the use of double precision makes HSM work in practice.

2.4.2.5 Lower Bounds and Probability Flow ODE

Given the score model $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$, we can synthesize novel samples via simulating the reverse-time diffusion SDE, Eq. (2.5) in the main text. This can be achieved, for example, via our novel SSCS, Euler-Maruyama, or methods such as GGF [134]. However, [262, 263] have shown that a corresponding ordinary differential equation can be defined that generates samples from the same distribution, in case $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ models the ground truth scores perfectly. This ODE is:

$$d\bar{\mathbf{u}}_t = \left[-\boldsymbol{f}(\bar{\mathbf{u}}_t, T-t) + \frac{1}{2}\boldsymbol{G}(\bar{\mathbf{u}}_t, T-t)\boldsymbol{G}(\bar{\mathbf{u}}_t, T-t)^\top \nabla_{\bar{\mathbf{u}}_t} \log p_{T-t}(\bar{\mathbf{u}}_t) \right] dt \tag{2.61}$$

This ODE is often referred to as the probability flow ODE. We can use it to generate novel data by sampling the prior and solving this ODE, like previous works [263]. Note that in practice $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ won't be a perfect model, though, such that the generative models defined by simulating the reverse-time SDE and the probability flow ODE are not exactly equivalent [262]. Nevertheless, they are very closely connected and it has been shown that their performance is usually very similar or almost the same, when we have learnt a good $s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$. In addition to sampling the generative SDE in our paper, we also sample from our CLD-based SGMs via this probability flow approach.

With the definition of our CLD, the ODE becomes:

$$\begin{pmatrix} d\bar{\mathbf{x}}_t \\ d\bar{\mathbf{v}}_t \end{pmatrix} = \underbrace{\begin{pmatrix} -M^{-1}\bar{\mathbf{v}}_t \\ \bar{\mathbf{x}}_t \end{pmatrix} \beta dt}_{A'_H} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ \Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix} \beta dt}_{S'} \tag{2.62}$$

Notice the interesting form of this probability flow ODE for CLD: It corresponds to Hamiltonian dynamics (A'_H) plus the score function term S'. Compared to the generative SDE (Sec. 2.3.3.3), the Ornstein-Uhlenbeck term disappears. Generally, symplectic integrators are best suited for integrating Hamiltonian systems [167, 209, 277]. However, our ODE is not perfectly Hamiltonian, due to the score term, and modern non-symplectic methods,

such as the higher-order adaptive-step size Runge-Kutta 4(5) ODE integrator [77], which we use in practice to solve the probability flow ODE, can also accurately simulate Hamiltonian systems over limited time horizons.

Importantly, the ODE formulation also allows us to estimate the log-likelihood of given test data, as it essentially defines a continuous Normalizing flow [50, 100], that we can easily run in either direction. However, in CLD the input into this ODE is not just the data $\mathbf{x}_0$, but also the velocity variable $\mathbf{v}_0$. In this case, we can still calculate a lower bound on the log-likelihood:

$$\begin{aligned}
\log p(\mathbf{x}_0) &= \log \left(\int p(\mathbf{x}_0, \mathbf{v}_0) d\mathbf{v}_0 \right) \\
&= \log \left(\int p(\mathbf{v}_0) \frac{p(\mathbf{x}_0, \mathbf{v}_0)}{p(\mathbf{v}_0)} d\mathbf{v}_0 \right) \\
&\geq \mathbb{E}_{\mathbf{v}_0 \sim p(\mathbf{v}_0)} \left[\log p(\mathbf{x}_0, \mathbf{v}_0) - \log p(\mathbf{v}_0) \right] \\
&= \mathbb{E}_{\mathbf{v}_0 \sim p(\mathbf{v}_0)} \left[\log p(\mathbf{x}_0, \mathbf{v}_0) \right] + H(p(\mathbf{v}_0))
\end{aligned} \tag{2.63}$$

where $H(p(\mathbf{v}_0))$ denotes the entropy of $p(\mathbf{v}_0)$ (we have $H(p(\mathbf{v}_0)) = \frac{1}{2} \log (2\pi e \gamma M)$). We can obtain a stochastic, but unbiased estimate of $\log p(\mathbf{x}_0, \mathbf{v}_0) \approx \log p_\varepsilon(\mathbf{x}_0, \mathbf{v}_0)$ via solving the probability flow ODE with initial conditions $(\mathbf{x}_0, \mathbf{v}_0)$ and calculating a stochastic estimate of the log-determinant of the Jacobian via Hutchinson's trace estimator (and also calculating the probability of the output under the prior), as done in Normalizing flows [50, 100] and previous works on SGMs [262, 263]. In the main paper, we report the negative of Eq. (2.63) as our upper bound on the negative log-likelihood (NLL).

Note that this bound can be potentially quite loose. In principle, it would be desirable to perform an importance-weighted estimation of the log-likelihood, as in importance-weighted autoencoders [33], using multiple samples from the velocity distribution. However, this isn't possible, as we only have access to a *stochastic* estimate $\log p_\varepsilon(\mathbf{x}_0, \mathbf{v}_0)$. The problems arising from this are discussed in detail in Appendix F of Vahdat et al. [280]. We could consider training a velocity encoder network, somewhat similar to [47], to improve our bound, but we leave this for future research.

2.4.2.6 On Introducing a Hamiltonian Component into the Diffusion

Here, we provide additional high-level intuitions and motivations about adding the Hamiltonian component to the diffusion process, as is done in our CLD.

Let us recall how the data distribution evolves in the forward diffusion process of SGMs: The role of the diffusion is to bring the initial non-equilibrium state quickly towards the equilibrium or prior distribution. Suppose for a moment, we could do so with "pure" Hamiltonian dynamics (no noise injection). In that case, we could generate data from the backward model without learning a score or neural network at all, because Hamiltonian dynamics is analytically invertible (flipping the sign of the velocity, we can just integrate backwards in reverse time direction). Now, this is not possible in practice, since Hamiltonian dynamics alone usually cannot convert the non-equilibrium distribution to the prior distribution. Nevertheless, Hamiltonian dynamics essentially achieves a certain amount of mixing on its own; moreover, since it is deterministic and analytically invertible, this mixing comes at no cost in the sense that we do not have to learn a complex score function to invert the Hamiltonian dynamics. Our thought experiment shows that we should strive for a diffusion process that behaves as deterministically (meaning that deterministic implies easily invertible) as possible with as little noise injection as possible. And this is exactly what is achieved by adding the Hamiltonian component in the overall diffusion process. In fact, recall that it is the diffusion coefficient $\boldsymbol{G}$ of the forward SDE that ultimately scales the score function term of the backward generative SDE (and it is the score function that is hard to approximate with complex neural nets). Therefore, in other words, relying more on a deterministic Hamiltonian component for enhanced mixing (mixing just like in MCMC in that it brings us quickly towards the target distribution, in our case the prior) and less on pure noise injection will lead to a nicer generative SDE that relies less on a score function that requires complex and approximate neural network-based modeling, but more on a simple and analytical Hamiltonian component. Such an SDE could then be solved easier with an appropriate integrator (like our SSCS). In the end, we believe that this is the reason why our networks are "smoother" and why given the same network capacity and limited compute budgets we essentially outperform all previous results in the literature (on CIFAR-10).

We would also like to offer a second perspective, inspired by the Markov chain Monte Carlo (MCMC) literature. In MCMC, "mixing" helps to quickly traverse the high probability parts of the target distribution and, if an MCMC chain is initialized far from the high probability manifold, to quickly converge to this manifold. However, this is precisely the situation we are in with the forward diffusion process of SGMs: The system is initialized in a far-from-equilibrium state (the data distribution) and we need to traverse the space as efficiently as possible to converge to the equilibrium distribution, this is, the prior. Without efficient mixing, it takes longer to converge to the prior, which also implies a longer generation path in the reverse direction—which intuitively corresponds to a harder problem.

Therefore, we believe that ideas from the MCMC literature that accelerate mixing and traversal of state space may be beneficial also for the diffusions in SGMs. In fact, leveraging Hamiltonian dynamics to accelerate sampling is popular in the MCMC field [209]. Note that this line of reasoning extends to thermostating techniques from statistical mechanics and molecular dynamics, which essentially tackle similar problems like MCMC methods from the statistics literature (see discussion in Sec. 2.3.4).

2.4.3 HSM: Hybrid Score Matching

We begin by recalling our objective function from App. 2.4.2.3 (Eq. (2.47)):

$$\mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u})}[\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)-s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2]\right], \tag{2.64}$$

where $s_{\boldsymbol{\theta}}(\mathbf{u},t)$ is our score model. In the following, we dissect the "score matching" part of the above objective:

$$\begin{aligned}\mathcal{L}_{\text{SM}} &\coloneqq \mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t)}\left[\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)-s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2\right]\\ &= \mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t)}\|s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2 - 2S(\boldsymbol{\theta})+C_2(t).\end{aligned} \tag{2.65}$$

where $C_2(t)\coloneqq\mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t)}\left[\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t)\|_2^2\right]$ and $S(\boldsymbol{\theta})$ is the cross term discussed below. Following Vincent [282], we can rewrite $\mathcal{L}_{\text{SM}}$ as an equivalent (up to addition of a time-dependent constant) denoising score matching objective $\mathcal{L}_{\text{DSM}}$:

$$\begin{aligned}\mathcal{L}_{\text{DSM}} &\coloneqq \mathbb{E}_{\mathbf{u}_0\sim p(\mathbf{u}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{u}_0)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t\mid\mathbf{u}_0)-s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2\\ &= \mathcal{L}_{\text{SM}}+C_3(t)-C_2(t),\end{aligned} \tag{2.66}$$

where $C_3(t)\coloneqq\mathbb{E}_{\mathbf{u}_0\sim p(\mathbf{u}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{u}_0)}\left[\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t\mid\mathbf{u}_0)\|_2^2\right]$. Something that might not necessarily be quite obvious is that there is no fundamental need to "denoise" with the distribution $p(\mathbf{u}_0)$ (this is, use samples from the joint $\mathbf{x}_0$-$\mathbf{v}_0$ distribution $p(\mathbf{u}_0)$, perturb them, and learn the score for denoising).

Instead, we can "denoise" only with the data distribution $p(\mathbf{x}_0)$ and marginalize over the entire initial velocity distribution $p(\mathbf{v}_0)$, which results in

$$\begin{aligned}&\mathcal{L}_{\text{HSM}}\coloneqq\mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{x}_0)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t\mid\mathbf{x}_0)-s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2\\ &=\mathbb{E}_{\mathbf{u}_t\sim p_t(\mathbf{u}_t)}\|s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2-2\mathbb{E}_{\mathbf{x}_0\sim p(\mathbf{x}_0),\mathbf{u}_t\sim p_t(\mathbf{u}_t|\mathbf{x}_0)}[\langle\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t\mid\mathbf{x}_0),s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\rangle]+C_4(t),\end{aligned} \tag{2.67}$$

where $C_4(t) \coloneqq \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0), \mathbf{u}_t \sim p_t(\mathbf{u}_t|\mathbf{x}_0)} \left[\|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{x}_0)\|_2^2 \right]$ and $\langle \cdot, \cdot \rangle$ donates the inner product (notation chosen to be consistent with Vincent [282]). In our case, this makes sense since $p(\mathbf{v}_0)$ is Normal, and therefore (as shown in App 2.4.2.1), the perturbation kernel $p_t(\mathbf{u}_t \mid \mathbf{x}_0)$ is still Normal.

In the following, for completeness, we redo the derivation of Vincent [282] and show that $\mathcal{L}_{\text{SM}}$ is equivalent to $\mathcal{L}_{\text{HSM}}$ (up to addition of a constant). Starting from $S(\boldsymbol{\theta})$, we have

$$
\begin{aligned}
S(\boldsymbol{\theta}) &= \mathbb{E}_{\mathbf{u}_t \sim p_t(\mathbf{u}_t)} \left\langle \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t), s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \\
&= \int_{\mathbf{u}_t} p_t(\mathbf{u}_t) \left\langle \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t), s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \, d\mathbf{u}_t \\
&= \int_{\mathbf{u}_t} \left\langle \nabla_{\mathbf{v}_t} p_t(\mathbf{u}_t), s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \, d\mathbf{u}_t \\
&= \int_{\mathbf{u}_t} \left\langle \nabla_{\mathbf{v}_t} \int_{\mathbf{x}_0} p_t(\mathbf{u}_t \mid \mathbf{x}_0) p_0(\mathbf{x}_0) \, d\mathbf{x}_0, s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \, d\mathbf{u}_t \\
&= \int_{\mathbf{u}_t} \left\langle \int_{\mathbf{x}_0} p_t(\mathbf{u}_t \mid \mathbf{x}_0) p_0(\mathbf{x}_0) \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{x}_0) \, d\mathbf{x}_0, s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \, d\mathbf{u}_t \\
&= \int_{\mathbf{u}_t} \int_{\mathbf{x}_0} p_t(\mathbf{u}_t \mid \mathbf{x}_0) p_0(\mathbf{x}_0) \left\langle \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{x}_0), s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \, d\mathbf{x}_0 \, d\mathbf{u}_t \\
&= \mathbb{E}_{\mathbf{x}_0 \sim p_0(\mathbf{x}_0), \mathbf{u}_t \sim p(\mathbf{u}_t|\mathbf{x}_0)} \left[\left\langle \nabla_{\mathbf{v}_t} \log p_t(\mathbf{u}_t \mid \mathbf{x}_0), s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) \right\rangle \right].
\end{aligned}
\tag{2.68}
$$

Hence, we have that

$$
\mathcal{L}_{\text{HSM}} = \mathcal{L}_{\text{SM}} + C_4(t) - C_2(t). \tag{2.69}
$$

This further implies that

$$
\mathcal{L}_{\text{HSM}} = \mathcal{L}_{\text{DSM}} + C_4(t) - C_3(t). \tag{2.70}
$$

Using the analysis from App 2.4.2.1, we realize that C_3 and C_4 can be simplified to $d \left(\ell_t^{\text{DSM}}\right)^2$ and $d \left(\ell_t^{\text{HSM}}\right)^2$, respectively. Here, we used the fact that the expected squared norm of a multivariate standard Normal random variable is equal to its dimension, i.e., $\mathbb{E}_{\varepsilon \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)} \|\varepsilon\|_2^2 = d$. This analysis then simplifies Eq. (2.70) to

$$
\mathcal{L}_{\text{HSM}} = \mathcal{L}_{\text{DSM}} + d \left(\left(\ell_t^{\text{DSM}}\right)^2 - \left(\ell_t^{\text{HSM}}\right)^2 \right). \tag{2.71}
$$

Using this relation, we can also find a connection between our CLD objective functions from App. 2.4.2.3. In particular, we have

$$\begin{aligned}\mathrm{HSM}(\lambda(t)) &= \mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\mathcal{L}_{\mathrm{HSM}}\right] \\ &= \mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\mathcal{L}_{\mathrm{DSM}}\right] + d\,\mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\left(\left(\ell_t^{\mathrm{DSM}}\right)^2 - \left(\ell_t^{\mathrm{HSM}}\right)^2\right)\right], \\ &= \mathrm{DSM}(\lambda(t)) + d\,\mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\left(\left(\ell_t^{\mathrm{DSM}}\right)^2 - \left(\ell_t^{\mathrm{HSM}}\right)^2\right)\right].\end{aligned} \tag{2.72}$$

2.4.3.1 Gradient Variance Reduction via HSM

Above, we derived that $\mathcal{L}_{\mathrm{HSM}} = \mathcal{L}_{\mathrm{DSM}} + const$, so one might wonder why we advocate for HSM over DSM. As discussed in Sec. 2.3.3.2, one advantage of HSM is that it avoids unbounded scores at $t \to 0$. However, there is a second advantage: In practice, we never solve expectations analytically but rather approximate them using Monte Carlo estimates. In the remainder of this section, we will show that in practice (Monte Carlo) gradients based on HSM have lower variance than those based on DSM.

From Eq. (2.72), we have

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathrm{HSM}(\lambda(t)) &= \mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{HSM}}\right] \\ &= \mathbb{E}_{t\sim\mathcal{U}[0,T]}\left[\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{DSM}}\right], \\ &= \nabla_{\boldsymbol{\theta}}\mathrm{DSM}(\lambda(t)),\end{aligned} \tag{2.73}$$

where $\boldsymbol{\theta}$ are the learnable parameters of the neural network. Instead of comparing the above expectations directly, we instead compare $\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{HSM}}$ with $\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{DSM}}$ for $t \in [0, 1]$ (we use $T = 1$ in all experiments) at discretized time values (as is done in practice). Replacing $\mathcal{L}_{\mathrm{HSM}}$ and $\mathcal{L}_{\mathrm{DSM}}$ with a single Monte Carlo estimate (as is used in practice), we have

$$\begin{aligned}\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{HSM}} \approx \lambda(t)\nabla_{\boldsymbol{\theta}}s_\theta(\mathbf{u}_t,t)\nabla_{s_\theta(\mathbf{u}_t,t)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{x}_0) - s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2, \\ \mathbf{x}_0 \sim p(\mathbf{x}_0), \mathbf{u}_t \sim p_t(\mathbf{u}_t \mid \mathbf{x}_0),\end{aligned} \tag{2.74}$$

$$\begin{aligned}\lambda(t)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{DSM}} \approx \lambda(t)\nabla_{\boldsymbol{\theta}}s_\theta(\mathbf{u}_t,t)\nabla_{s_\theta(\mathbf{u}_t,t)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{u}_0) - s_{\boldsymbol{\theta}}(\mathbf{u}_t,t)\|_2^2, \\ \mathbf{u}_0 \sim p(\mathbf{u}_0), \mathbf{u}_t \sim p_t(\mathbf{u}_t \mid \mathbf{u}_0),\end{aligned} \tag{2.75}$$

where we applied the chain-rule. Note that in Eq. (2.74) and Eq. (2.75), $\mathbf{u}_t$ is sampled from the same distribution. Hence, $\lambda(t)\nabla_{\boldsymbol{\theta}}s_\theta(\mathbf{u}_t,t)$ acts as a common scaling factor, with the variance difference between HSM and DSM originating from the squared norm term.

Hence, we ignore $\lambda(t)\nabla_{\boldsymbol{\theta}} s_\theta(\mathbf{u}_t, t)$ and only focus our analysis on the gradient of the norm terms, which we can further simplify:

$$\frac{1}{2}\nabla_{s_\theta(\mathbf{u}_t,t)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{x}_0) - s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2 = s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) - \nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{x}_0) =: \mathcal{K}_{\text{HSM}}, \tag{2.76}$$

and

$$\frac{1}{2}\nabla_{s_\theta(\mathbf{u}_t,t)}\|\nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{u}_0) - s_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\|_2^2 = s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) - \nabla_{\mathbf{v}_t}\log p_t(\mathbf{u}_t \mid \mathbf{u}_0) =: \mathcal{K}_{\text{DSM}}. \tag{2.77}$$

We explore this difference in a realistic setup; in particular, we evaluate $\mathcal{K}_{\text{HSM}}$ and $\mathcal{K}_{\text{DSM}}$ for all data points in the CIFAR-10 training set. We choose $s_{\boldsymbol{\theta}}$ to be our trained ablation CLD model (with the standard setup of $M^{-1} = \beta = 4$, see Sec. 2.4.5.2.1 for model details). We then use these samples to compute the empirical covariance matrices Cov_{HSM} and Cov_{DSM} of the random variables $\mathcal{K}_{\text{HSM}}$ and $\mathcal{K}_{\text{DSM}}$, respectively.

As is common practice in statistics, we consider only the trace of the estimated covariance matrices.[4] The trace of the covariance matrix (of a random variable) is also commonly referred to as the total variation (of a random variable).

We visualize our results in Fig. 2.7. For HSM, there is barely any visual difference in $\text{Tr}(\text{Cov})$ for $\gamma = 0.04$ and $\gamma = 1$. For DSM, both $\gamma = 0.04$ and $\gamma = 1$ result in very large $\text{Tr}(\text{Cov})$ values for small t. For large t, $\text{Tr}(\text{Cov})$ is considerably smaller for $\gamma = 0.04$ than for $\gamma = 1$. However, in practice, we found that DSM is even unstable for small γ. Given this analysis, we believe this is due to the large gradient variance for small t. In conclusion, these results demonstrate a clear variance reduction by the HSM objective, in particular for large γ. Ultimately, this is expected: In HSM, we are effectively integrating out the initial velocity distribution when estimating gradients, while in DSM we use noisy samples for the initial velocity.

Note that re-introducing $\lambda(t)$ weightings would allow us to scale the $\text{Tr}(\text{Cov})$ curves according to the "reweighted" objective or the maximum likelihood objective. However, we believe it is most instructive to directly analyze the gradient of the relevant norm term itself.

[4] Arguably, the most prominent algorithm that follows this practice is principal component analysis (PCA).

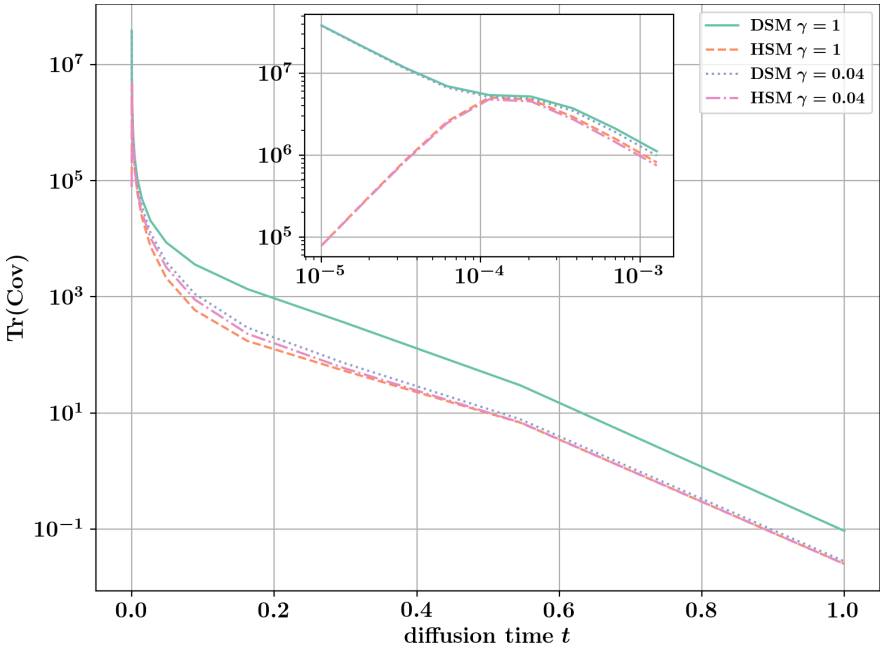


Figure 2.7: Traces of the estimated covariance matrices.

2.4.4 Symmetric Splitting CLD Sampler (SSCS)

In this section, we present a more complete derivation and analysis of our novel Symmetric Splitting CLD Sampler (SSCS).

2.4.4.1 Background

Our derivation is inspired by methods from the statistical mechanics and molecular dynamics literature. In particular, we are leveraging symmetric splitting techniques as well as (Fokker–Planck) operator concepts. The high-level idea of symmetric splitting as well as the operator formalism are well-explained in [277], in particular in their Section 3.10, which includes simple examples. Symmetric splitting methods for stochastic dynamics in particular are discussed in detail in Leimkuhler and Matthews [166]. We also recommend Leimkuhler and Matthews [165], which discusses splitting methods for Langevin dynamics in a concise but insightful manner.

2.4.4.2 Derivation and Analysis

Generative SDE. From Sec. 2.3.3.3, recall that our generative SDE can be written as (with $\bar{\mathbf{u}}_t = \mathbf{u}_{T-t}$, $\bar{\mathbf{x}}_t = \mathbf{x}_{T-t}$, $\bar{\mathbf{v}}_t = \mathbf{v}_{T-t}$):

$$\begin{pmatrix} d\bar{\mathbf{x}}_t \\ d\bar{\mathbf{v}}_t \end{pmatrix} = \underbrace{\begin{pmatrix} -M^{-1}\bar{\mathbf{v}}_t \\ \bar{\mathbf{x}}_t \end{pmatrix} \beta dt}_{A_H} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\bar{\mathbf{v}}_t \end{pmatrix} \beta dt + \begin{pmatrix} \mathbf{0}_d \\ \sqrt{2\Gamma\beta} d\mathbf{w}_t \end{pmatrix}}_{A_O} + \underbrace{\begin{pmatrix} \mathbf{0}_d \\ 2\Gamma \left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix} \beta dt}_{S}. \tag{2.78}$$

Fokker–Planck Equation and Fokker–Planck Operators. The evolution of the probability distribution $p_{T-t}(\bar{\mathbf{u}}_t)$ is described by the general Fokker–Planck equation [247]:

$$\frac{\partial p_{T-t}(\bar{\mathbf{u}}_t)}{\partial t} = -\sum_{i=1}^{2d} \frac{\partial}{\partial \bar{u}_i} \left[\mu_i(\bar{\mathbf{u}}_t, T-t) p_{T-t}(\bar{\mathbf{u}}_t)\right] + \sum_{i=1}^{2d}\sum_{j=1}^{2d} \frac{\partial^2}{\partial \bar{u}_i \partial \bar{u}_j} \left[D_{ij}(\bar{\mathbf{u}}_t, T-t) p_{T-t}(\bar{\mathbf{u}}_t)\right], \tag{2.79}$$

with

$$\boldsymbol{\mu}(\bar{\mathbf{u}}_t, T-t) = \begin{pmatrix} -M^{-1}\bar{\mathbf{v}}_t \\ \bar{\mathbf{x}}_t \end{pmatrix} \beta + \begin{pmatrix} \mathbf{0}_d \\ -\Gamma M^{-1}\bar{\mathbf{v}}_t \end{pmatrix} \beta + \begin{pmatrix} \mathbf{0}_d \\ 2\Gamma \left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix} \beta, \tag{2.80}$$

$$D(\bar{\mathbf{u}}_t, T-t) = \begin{pmatrix} 0 & 0 \\ 0 & \Gamma\beta \end{pmatrix} \otimes \boldsymbol{I}_d. \tag{2.81}$$

For our SDE, we can write the Fokker–Planck equation in short form as

$$\frac{\partial p_{T-t}(\bar{\mathbf{u}}_t)}{\partial t} = (\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*) p_{T-t}(\bar{\mathbf{u}}_t), \tag{2.82}$$

with the Fokker–Planck operators (defined via their action on functions of the variables $\phi(\bar{\mathbf{u}}_t)$):

$$\hat{\mathcal{L}}_A^* \phi(\bar{\mathbf{u}}_t) := \beta M^{-1}\bar{\mathbf{v}}_t \nabla_{\bar{\mathbf{x}}_t}\phi(\bar{\mathbf{u}}_t) - \beta\bar{\mathbf{x}}_t \nabla_{\bar{\mathbf{v}}_t}\phi(\bar{\mathbf{u}}_t) + \Gamma\beta M^{-1}\nabla_{\bar{\mathbf{v}}_t}\left[\bar{\mathbf{v}}_t\phi(\bar{\mathbf{u}}_t)\right] + \Gamma\beta\Delta_{\bar{\mathbf{v}}_t}\phi(\bar{\mathbf{u}}_t), \tag{2.83}$$

$$\hat{\mathcal{L}}_S^* \phi(\bar{\mathbf{u}}_t) := -2\Gamma\beta\nabla_{\bar{\mathbf{v}}_t}\left[\left(\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right)\phi(\bar{\mathbf{u}}_t)\right], \tag{2.84}$$

$$\Delta_{\bar{\mathbf{v}}_t} := \sum_{i=1}^{d} \left(\frac{\partial^2}{\partial \bar{x}_i^2} + \frac{\partial^2}{\partial \bar{v}_i^2}\right). \tag{2.85}$$

We are providing these formulas for transparency and completeness. We do not directly leverage them. However, working with these operators can be convenient. In particular, the operators describe the time evolution of states $\bar{\mathbf{u}}_t$ under the stochastic dynamics defined by the SDE. Given an initial state $\bar{\mathbf{u}}_0$, we can construct a formal solution to the generative SDE via [166, 277]:

$$\bar{\mathbf{u}}_t = e^{t\left(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*\right)} \bar{\mathbf{u}}_0, \tag{2.86}$$

where the operator $e^{t\left(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*\right)}$ is known as the classical propagator that propagates states $\bar{\mathbf{u}}_0$ for time t according to the dynamics defined by the combined Fokker–Planck operators $\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*$ (to avoid confusion, note that in Eq. (2.86) the operator $e^{t\left(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*\right)}$ is applied on $\bar{\mathbf{u}}_0$ in an element-wise or "vectorized" fashion on all elements of $\bar{\mathbf{u}}_0$ in parallel). The problem with that expression is that we cannot analytically evaluate it. However, we can leverage it to design an integration method.

Symmetric Splitting Integration. Using the symmetric Trotter theorem or Strang splitting formula as well as the Baker–Campbell–Hausdorff formula [266, 275, 277], it can be shown that:

$$e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)} = \lim_{N\to\infty} \left[e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*} e^{\delta t \hat{\mathcal{L}}_S^*} e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*} \right]^N \approx \left[e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*} e^{\delta t \hat{\mathcal{L}}_S^*} e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*} \right]^N + \mathcal{O}(N\delta t^3), \tag{2.87}$$

for large $N \in \mathbb{N}^+$ and time step $\delta t := t/N$. The expression suggests that instead of directly evaluating the intractable $e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)}$, we can discretize the dynamics over t into N pieces of step size δt, such that we only need to apply the *individual* $e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}$ and $e^{\delta t \hat{\mathcal{L}}_S^*}$ many times one after another for small time steps δt. A finer discretization implies a smaller error (since $N{=}t/\delta t$ the error effectively scales as $\mathcal{O}(\delta t^2)$ for fixed t). Hence, this implies an integration method. The general idea of such splitting schemes is to split an initially intractable propagator into separate terms, each of which is analytically tractable. In that case, the overall integration error for many steps is only due to the splitting scheme error,[5] but not due to the evaluation of the individual updates. Such techniques are, for example, popular in molecular dynamics to develop symplectic integrators as well as accurate samplers [34, 165, 166, 276, 277].

Analyzing the Splitting Terms. Next, we need to analyze the two individual terms:

[5] In principle, the error of the splitting scheme is defined more specifically by the commutator of the non-commuting Fokker–Planck operators. See, for example Leimkuhler and Matthews [165, 166], Tuckerman [277].

(i) Let us first analyze $e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\bar{\mathbf{u}}_t$: This term describes the stochastic evolution of $\bar{\mathbf{u}}_t$ under the dynamics of an SDE like Eq. (2.78), but with S set to zero. However, if S is set to zero, the remaining SDE has affine drift and diffusion coefficients. In that case, if the input is Normal (or a discrete state corresponding to a Normal with 0 variance) then the distribution is Normal at all times and we can calculate the evolution analytically. In particular, we can solve the differential equations for the mean $\bar{\boldsymbol{\mu}}_{\frac{\delta t}{2}}$ and covariance $\bar{\boldsymbol{\Sigma}}_{\frac{\delta t}{2}}$ of the Normal (see Sec. 2.4.2.1), and obtain

$$\bar{\boldsymbol{\mu}}_{\frac{\delta t}{2}}(\bar{\mathbf{u}}_t) = \begin{pmatrix} 2\beta\frac{\delta t}{2}\Gamma^{-1}\bar{\mathbf{x}}_t - 4\beta\frac{\delta t}{2}\Gamma^{-2}\bar{\mathbf{v}}_t + \bar{\mathbf{x}}_t \\ \beta\frac{\delta t}{2}\bar{\mathbf{x}}_t - 2\beta\frac{\delta t}{2}\Gamma^{-1}\bar{\mathbf{v}}_t + \bar{\mathbf{v}}_t \end{pmatrix} e^{-2\beta\frac{\delta t}{2}\Gamma^{-1}}, \tag{2.88}$$

as well as

$$\bar{\boldsymbol{\Sigma}}_{\frac{\delta t}{2}} = \bar{\Sigma}_{\frac{\delta t}{2}} \otimes \boldsymbol{I}_d, \tag{2.89}$$

$$\bar{\Sigma}_{\frac{\delta t}{2}} = \begin{pmatrix} \bar{\Sigma}^{xx}_{\frac{\delta t}{2}} & \bar{\Sigma}^{xv}_{\frac{\delta t}{2}} \\ \bar{\Sigma}^{xv}_{\frac{\delta t}{2}} & \bar{\Sigma}^{vv}_{\frac{\delta t}{2}} \end{pmatrix} e^{-4\beta\frac{\delta t}{2}\Gamma^{-1}}, \tag{2.90}$$

$$\bar{\Sigma}^{xx}_{\frac{\delta t}{2}} = e^{4\beta\frac{\delta t}{2}\Gamma^{-1}} - 1 - 4\beta\frac{\delta t}{2}\Gamma^{-1} - 8\left(\beta\frac{\delta t}{2}\right)^2\Gamma^{-2}, \tag{2.91}$$

$$\bar{\Sigma}^{xv}_{\frac{\delta t}{2}} = -4\left(\beta\frac{\delta t}{2}\right)^2\Gamma^{-1}, \tag{2.92}$$

$$\bar{\Sigma}^{vv}_{\frac{\delta t}{2}} = \frac{\Gamma^2}{4}\left(e^{4\beta\frac{\delta t}{2}\Gamma^{-1}} - 1\right) + \beta\frac{\delta t}{2}\Gamma - 2\left(\beta\frac{\delta t}{2}\right)^2. \tag{2.93}$$

The correctness of the proposed mean and covariance matrix can be verified by simply plugging them back in their respective ODEs; see App. 2.4.7.2.

Now, we can write the action of the the propagator $e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}$ on a state $\bar{\mathbf{u}}_t$ as:

$$e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\bar{\mathbf{u}}_t \sim \mathcal{N}(\bar{\mathbf{u}}_{t+\frac{\delta t}{2}}; \bar{\boldsymbol{\mu}}_{\frac{\delta t}{2}}(\bar{\mathbf{u}}_t), \bar{\boldsymbol{\Sigma}}_{\frac{\delta t}{2}}). \tag{2.94}$$

(ii): Next, we need to analyze $e^{\delta t\hat{\mathcal{L}}_S^*}\bar{\mathbf{u}}_t$. Unfortunately, we cannot calculate the action of the propagator $e^{\delta t\hat{\mathcal{L}}_S^*}$ on $\bar{\mathbf{u}}_t$ analytically and we need to make an approximation. From Eq. (2.78), we can easily see that the propagator $e^{\delta t\hat{\mathcal{L}}_S^*}$ describes the evolution of the velocity component $\bar{\mathbf{v}}_t$ for time step δt under the ODE (this can be easily seen by noticing

that the S term in Eq. (2.78) only acts on the velocity component of the joint state $\bar{\mathbf{u}}_t$):

$$d\bar{\mathbf{v}}_t = 2\beta\Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] dt. \tag{2.95}$$

We propose to simply solve this ODE for the step δt via a simple step of Euler's method, resulting in:

$$\begin{aligned} e^{\delta t \hat{\mathcal{L}}_S^*}\bar{\mathbf{u}}_t &\approx \bar{\mathbf{u}}_t + \delta t \begin{pmatrix} \mathbf{0}_d \\ 2\beta\Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix} + \mathcal{O}(\delta t^2) \\ &= e^{\delta t \hat{\mathcal{L}}_S^{*\,\text{Euler}}}\bar{\mathbf{u}}_t + \mathcal{O}(\delta t^2), \end{aligned} \tag{2.96}$$

with the informal definition

$$e^{\delta t \hat{\mathcal{L}}_S^{*\,\text{Euler}}}\bar{\mathbf{u}}_t := \bar{\mathbf{u}}_t + \delta t \begin{pmatrix} \mathbf{0}_d \\ 2\beta\Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix}. \tag{2.97}$$

Error Analysis. It is now instructive to study the overall error of our proposed integrator. With the additional Euler integration in one of the splitting terms, we have

$$\begin{aligned} e^{t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)} &\approx \left[e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\left(e^{\delta t \hat{\mathcal{L}}_S^{*\,\text{Euler}}} + \mathcal{O}(\delta t^2)\right) e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\right]^N + \mathcal{O}(N\delta t^3) \\ &= \left[e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\left(e^{\delta t \hat{\mathcal{L}}_S^{*\,\text{Euler}}}\right) e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\right]^N + N\mathcal{O}(\delta t^2) \\ &= \left[e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\left(e^{\delta t \hat{\mathcal{L}}_S^{*\,\text{Euler}}}\right) e^{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*}\right]^N + \mathcal{O}(\delta t), \end{aligned} \tag{2.98}$$

where we used $N = \frac{t}{\delta t}$ and only kept the dominating error terms of lowest order in δt. We see that, just like Euler's method, also our SSCS is a first-order integrator with local error $\sim\delta t^2$ and global error $\sim\delta t$, which can be also seen from the last two lines of Eq. (2.98). This is expected, considering that we used an Euler step for the S term. Nevertheless, as long as the dynamics is not dominated by the S component, our proposed integration scheme is still expected to be more accurate than EM, since we split off the analytically tractable part and only use an Euler approximation for the S term.

To this end, recall that the model only needs to learn the score of the conditional distribution $p_t(\mathbf{v}_t|\mathbf{x}_t)$, which is close to Normal for much of the diffusion, in which case the S term will indeed be small. This suggests that the generative SDE dynamics are in fact dominated by A_H and A_O in practice. From another perspective, note that (recalling that

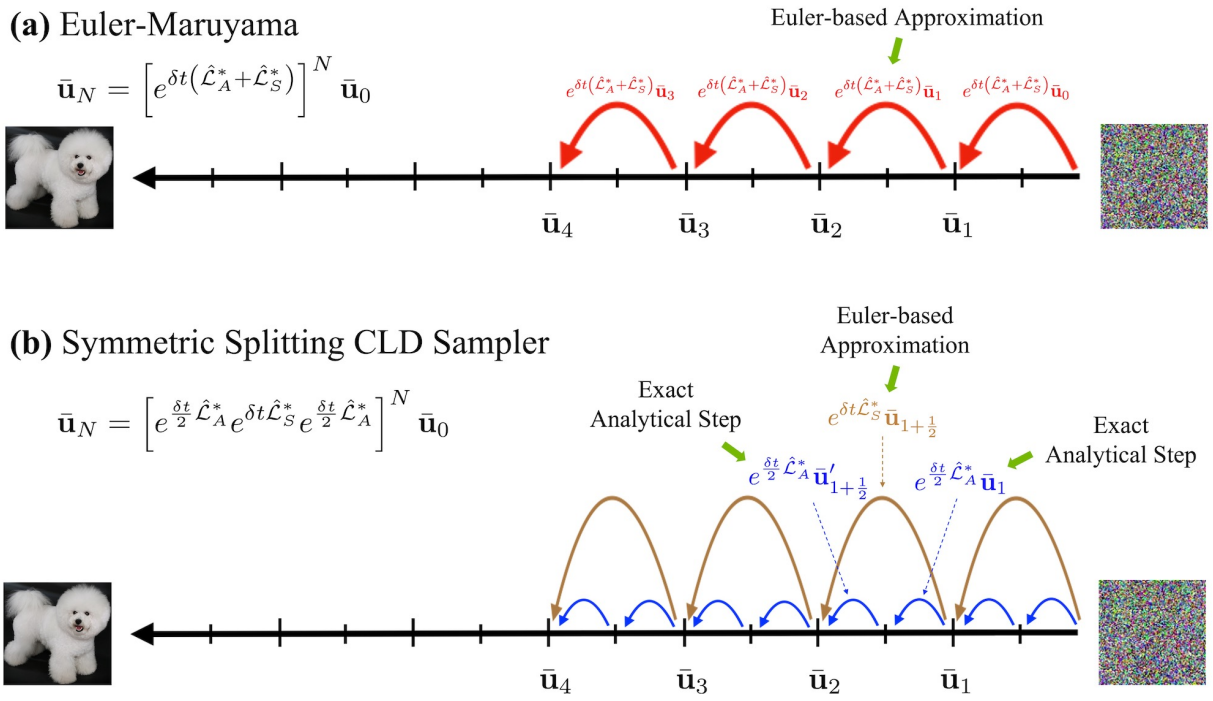


Figure 2.8: Conceptual visualization of our new SSCS sampler and comparison to Euler-Maruyama (for image synthesis): **(a)** In EM-based sampling, in each integration step the entire SDE is integrated using an Euler-based approximation. This can be formally expressed as solving the full-step propagator $\exp\left\{\delta t(\hat{\mathcal{L}}_A^* + \hat{\mathcal{L}}_S^*)\right\}$ via Euler-based approximation for N small steps of size δt (see **red** steps; for simplicity, this visualization assumes constant δt). **(b)**: In contrast, in our SSCS the propagator is partitioned into an analytically tractable component $\exp\left\{\frac{\delta t}{2}\hat{\mathcal{L}}_A^*\right\}$ (**blue**) and the score model term $\exp\left\{\delta t\hat{\mathcal{L}}_S^*\right\}$ (**brown**). Only the latter requires numerical approximation, which results in an overall more accurate integration scheme.

$s_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = -\ell_t \alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ with $\alpha_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = \ell_t^{-1}\mathbf{v}_t/\Sigma_t^{vv} + \alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ from Sec. 2.3.3.2):

$$\begin{aligned} \mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t &= -\ell_t \alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t, t) - \frac{\mathbf{v}_t}{\Sigma_t^{vv}} + M^{-1}\bar{\mathbf{v}}_t \\ &= -\ell_t \alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t, t) + \bar{\mathbf{v}}_t \left(\frac{1}{M} - \frac{1}{\Sigma_t^{vv}}\right). \end{aligned} \tag{2.99}$$

For large parts of the diffusion, Σ_t^{vv} is indeed close to M, such that the $\bar{\mathbf{v}}_t$ term is very small (this cancellation is the reason why we pulled the $M^{-1}\bar{\mathbf{v}}_t$ term into the S component). In Sec. 2.3.3, we have also seen that our neural network component $\alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$ can be much smoother than that of previous SGMs. Overall, this suggests that the error of SSCS indeed might be smaller than the error we would obtain when applying a naive Euler–Maruyama integrator to the full generative SDE. Our positive experimental results in Sec. 2.3.5.2 validate that. Only in the limit for very small steps, both our SSCS and EM make only very small errors and are expected to perform equally well, which is exactly what we observe in our experiments. Our SSCS turns out to be well-suited for integrating the generative SDE of CLD-SGMs with relatively few synthesis steps.

Note that error analysis of stochastic differential equation solvers is usually performed in terms of weak and strong convergence [154]. Due to the use of Euler's method for the S component, as argued above, we expect our SSCS to formally have the same weak and strong convergence properties like EM, this is, weak convergence of order 1 and strong convergence of order 1 as well, since the noise is additive in our case (and assuming appropriate smoothness conditions for the drift and diffusion coefficients; furthermore, without additive noise, we would have strong convergence of order 0.5). We leave a more detailed analysis to future work.

In practice, we do not use SSCS to integrate all the way from t=0 to t=T, but only up to t=$T - \epsilon$, and perform a denoising step, similar to previous works [134, 263]. It is worth noting that our SSCS scheme would also be applicable when we used time-dependent $\beta(t)$, as in our more general derivation of the CLD perturbation kernel in App. 2.4.2. However, since we only used constant β in the main paper, we also presented SSCS in that way.

A promising direction for future work would be to extend SSCS to adaptive step sizes and to use techniques to facilitate higher-order integration, while still leveraging the advantages of SSCS.

SSCS Algorithm. Finally, we summarize SSCS in terms of a concise algorithm:

Algorithm 1 *Symmetric Splitting CLD Sampler (SSCS)*

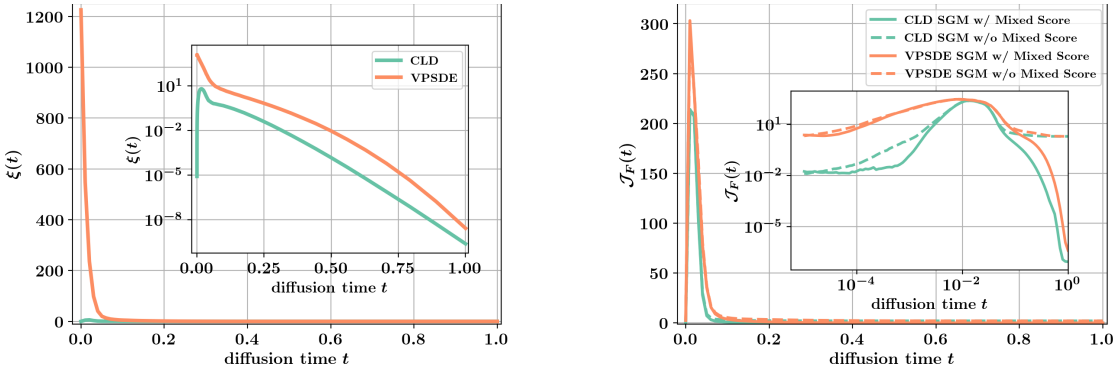
Input: Score function $\mathbf{s}_{\boldsymbol{\theta}}(\bar{\mathbf{u}}_t, T-t)$, CLD parameters Γ, β, $M = \Gamma^2/4$, number of sampling steps N, step sizes $\{\delta t_n \geq 0\}_{n=0}^{N-1}$ chosen such that $\epsilon := T - \sum_{n=0}^{N-1} \delta t_n \geq 0$ (stepsizes can vary, for example in QS).
Output: Synthesized model sample $\bar{\mathbf{x}}'_N$, along with a velocity sample $\bar{\mathbf{v}}'_N$.

$\bar{\mathbf{x}}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0; \mathbf{0}_d, \boldsymbol{I}_d)$, $\bar{\mathbf{v}}_0 \sim \mathcal{N}(\bar{\mathbf{v}}_0; \mathbf{0}_d, M\boldsymbol{I}_d)$, $\bar{\mathbf{u}}_0 = (\bar{\mathbf{x}}_0, \bar{\mathbf{v}}_0)$ ▷ Draw initial prior samples from $p_{\mathrm{EQ}}(\mathbf{u})$
$t = 0$ ▷ Initialize time
for $n = 0$ **to** $N-1$ **do**
  $\bar{\mathbf{u}}_{n+\frac{1}{2}} \sim \mathcal{N}(\bar{\mathbf{u}}_{n+\frac{1}{2}}; \bar{\boldsymbol{\mu}}_{\frac{\delta t_n}{2}}(\bar{\mathbf{u}}_n), \bar{\boldsymbol{\Sigma}}_{\frac{\delta t_n}{2}})$ ▷ First half-step: apply $\exp\{\frac{\delta t_n}{2}\hat{\mathcal{L}}_A^*\}$ on $\bar{\mathbf{u}}_n$
  $\bar{\mathbf{u}}'_{n+\frac{1}{2}} \leftarrow \bar{\mathbf{u}}_{n+\frac{1}{2}} + \delta t_n \begin{pmatrix} \mathbf{0}_d \\ 2\beta\Gamma\left[\mathbf{s}(\bar{\mathbf{u}}_t, T-t) + M^{-1}\bar{\mathbf{v}}_t\right] \end{pmatrix}$ ▷ Full step: apply $\exp\{\delta t_n \hat{\mathcal{L}}_S^*\}$ on $\bar{\mathbf{u}}_{n+\frac{1}{2}}$
  $\bar{\mathbf{u}}_{n+1} \sim \mathcal{N}(\bar{\mathbf{u}}_{n+1}; \bar{\boldsymbol{\mu}}_{\frac{\delta t_n}{2}}(\bar{\mathbf{u}}'_{n+\frac{1}{2}}), \bar{\boldsymbol{\Sigma}}_{\frac{\delta t_n}{2}})$ ▷ Second half-step: apply $\exp\{\frac{\delta t_n}{2}\hat{\mathcal{L}}_A^*\}$ on $\bar{\mathbf{u}}'_{n+\frac{1}{2}}$
  $t \leftarrow t + \delta t_n$ ▷ Update time
end for
$\bar{\mathbf{u}}'_N \leftarrow \bar{\mathbf{u}}_N - \epsilon \left(\begin{pmatrix} 0 & \beta M^{-1} \\ -\beta & -\Gamma\beta M^{-1} \end{pmatrix} \otimes \boldsymbol{I}_d \right) \bar{\mathbf{u}}_N + \epsilon \begin{pmatrix} \mathbf{0}_d \\ 2\beta\Gamma\mathbf{s}(\bar{\mathbf{u}}_t, \epsilon) \end{pmatrix}$ ▷ Denoising
$(\bar{\mathbf{x}}'_N, \bar{\mathbf{v}}'_N) = \bar{\mathbf{u}}'_N$ ▷ Extract output data and velocity samples

Note that the algorithm uses the expressions in Eqs. (2.88) and (2.89) for $\bar{\boldsymbol{\mu}}_t$ and $\bar{\boldsymbol{\Sigma}}_t$. Furthermore, in practice in the denoising step at the end, we usually only update the $\bar{\mathbf{x}}'_N$ component of $\bar{\mathbf{u}}'_N$, since we are only interested in the data sample. This saves us the final neural network call during denoising, which only affects the $\bar{\mathbf{v}}'_N$ component (also see App. 2.4.5.2.4). However, we wrote the algorithm in the general way, which also allows to correctly generate the velocity sample $\bar{\mathbf{v}}'_N$. In Fig. 2.8, we show a conceptual visualization of our SSCS and contrast it to EM.

Also note that we could combine the second half-step from one iteration of SSCS with the first half-step from the next iteration of SSCS. This is commonly done in the Leapfrog integrator [166, 167, 209, 277],[6] which follows a similar structure as our SSCS. However, it is not important in our case, as the only computationally costly operation is in the center full step, which involves the neural network evaluation. The first and last half-steps come at virtually no computational cost.

[6] The *Leapfrog integrator* corresponds to the *velocity Verlet integrator* in molecular dynamics.



(a) Difference $\xi(t)$ (via $L2$ norm) between score of diffused data and score of Normal distribution.

(b) Frobenius norm of Jacobian $\mathcal{J}_F(t)$ of the neural network defining the score function for different t.

Figure 2.9: Toy experiments for mixture of Normals dataset.

2.4.5 Implementation and Experiment Details

2.4.5.1 Score and Jacobian Experiments

In this section, we provide details for the experiments presented in Sec. 2.3.3.1. For both experiments, we consider a two-dimensional simple mixture of Normals of the form

$$p_{\text{data}}(\mathbf{x}) = \sum_{k=1}^{9} \frac{1}{9} p^{(k)}(\mathbf{x}), \tag{2.100}$$

where $p^{(k)}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k; 0.04^2 \boldsymbol{I}_2)$ and

$$\begin{aligned}
\boldsymbol{\mu}_1 &= \begin{pmatrix} -a \\ 0 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = \begin{pmatrix} -a/2 \\ a/2 \end{pmatrix}, \quad \boldsymbol{\mu}_3 = \begin{pmatrix} 0 \\ a \end{pmatrix}, \\
\boldsymbol{\mu}_4 &= \begin{pmatrix} -a/2 \\ -a/2 \end{pmatrix}, \quad \boldsymbol{\mu}_5 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{\mu}_6 = \begin{pmatrix} a/2 \\ a/2 \end{pmatrix}, \\
\boldsymbol{\mu}_7 &= \begin{pmatrix} 0 \\ -a \end{pmatrix}, \quad \boldsymbol{\mu}_8 = \begin{pmatrix} a/2 \\ -a/2 \end{pmatrix}, \quad \boldsymbol{\mu}_9 = \begin{pmatrix} a \\ 0 \end{pmatrix},
\end{aligned}$$

and $a = 2^{-\frac{1}{2}}$. The choice of this data distribution is not arbitrary. In fact, mixture of Normal distributions are diffused by simply diffusing the components, i.e., setting $p_0(\mathbf{x}_0) =$

$p_{\text{data}}(\mathbf{x})$, we have

$$p_t(\mathbf{x}_t) = \sum_{k=1}^{9} \frac{1}{9} p_t^{(k)}(\mathbf{x}_t), \tag{2.101}$$

where $p_t^{(k)}$ are the diffused components (analogously for CLD with velocity augmentation). This means that for both CLD as well as VPSDE [263] we can diffuse $p_{\text{data}}(\mathbf{x})$ with analytical access to the diffused marginal $p_t(\mathbf{x}_t)$ or $p_t(\mathbf{u}_t)$. This allows us to perform interesting analyses that would be impossible when working, for example, with image data. We visualize the data distribution in Fig. 2.10.

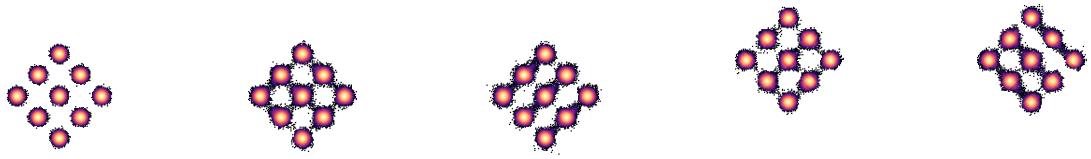
Score experiment: We empirically verify the reduced complexity of the score of $p_t(\mathbf{v}_t|\mathbf{x}_t)$, which is learned in CLD, compared to the score of $p_t(\mathbf{x}_t)$, which is learned in VPSDE. To avoid scaling issues between VPSDE and CLD, we chose $M = \gamma = 1$ for CLD in this experiment; this results in an equilibrium distribution of $\mathcal{N}(\mathbf{0}_2, \boldsymbol{I}_2)$ (for both data and velocity components, which are independent at equilibrium), which is the same as the equilibrium distribution of the VPSDE. We then measure the difference of the respective scores at time t and the equilibrium (or prior) scores, i.e. (recall that the score of a Normal distribution $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}_2, \boldsymbol{I}_2)$ is simply $\nabla_{\mathbf{x}} \log p(\mathbf{x}) = -\mathbf{x}$),

$$\xi^{\text{VPSDE}}(t) := \mathbb{E}_{\mathbf{x}_t \sim p(\mathbf{x}_t)} \|\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \mathbf{x}_t\|_2^2, \tag{2.102}$$

$$\xi^{\text{CLD}}(t) := \mathbb{E}_{\mathbf{u}_t \sim p(\mathbf{u}_t)} \|\nabla_{\mathbf{v}_t} \log p_t(\mathbf{v}_t \mid \mathbf{x}_t) + \mathbf{v}_t\|_2^2. \tag{2.103}$$

The expectations are approximated using 10^5 samples from $p(\mathbf{x}_t)$ and $p(\mathbf{u}_t)$ for VPSDE and CLD, respectively. As can be seen in Fig. 2.9a, $\xi^{\text{CLD}}(t)$ is smaller than $\xi^{\text{VPSDE}}(t)$ for all $t \in [0, T]$. The difference is particularly striking for small time values t. Other previous SDEs, such as the VESDE, sub-VPSDE, etc., are expected to behave similarly. This result implies that the ground truth scores that need to be learnt in CLD are closer to Normal scores than the ground truth scores in previous SDEs like the VPSDE. Since the score of a Normal is very simple—and indeed directly leveraged in our mixed score formulation—we would intuitively expect that the CLD training task is easier.

Complexity experiment: Therefore, to understand the above observations in terms of learning neural networks, we train a small ResNet architecture (less than 100k parameters) for each of the following four setups: both CLD and VPSDE each with and without a mixed score parameterization. The mixed score of the VPSDE simply assumes a standard Normal data distribution (which is also the equilibrium distribution of VPSDE) resulting in adding $-\mathbf{x}_t$ to the score function. Formally, $-\mathbf{x}_t$ is the score of a Normal distribution



(a) Data p_{data} (b) CLD w/ MS (c) CLD w/o MS (d) VPSDE w/ MS (e) VPSDE w/o MS

Figure 2.10: Mixture of Normals: data and trained models (samples).

with unit variance.

We train the models for 1M iterations using fresh data synthesized from p_{data} at a batch size of 512. The model and data distributions are visualized in Fig. 2.10. We see that all models have learnt good representations of the data. We measure the complexity of the trained neural networks using the squared Frobenius norm of the networks' Jacobians. For CLD, we have

$$\mathcal{J}_{\text{F}}^{\text{CLD}}(t) \coloneqq \mathbb{E}_{\mathbf{u}_t \sim p(\mathbf{u}_t)} \|\nabla_{\mathbf{u}_t} \alpha'_{\boldsymbol{\theta}}(\mathbf{u}_t)\|_F^2. \tag{2.104}$$

Similarly, for the VPSDE we compute

$$\mathcal{J}_{\text{F}}^{\text{VPSDE}}(t) \coloneqq \mathbb{E}_{\mathbf{x}_t \sim p(\mathbf{x}_t)} \|\nabla_{\mathbf{x}_t} \alpha'_{\boldsymbol{\theta}}(\mathbf{x}_t)\|_F^2. \tag{2.105}$$

For both CLD and VPSDE, expectations are again approximated using 10^5 samples. As can be seen in Fig. 2.9b the neural network complexity is significantly lower for CLD compared to VPSDE. A mixed score formulation further helps decreasing the neural network complexity for both CLD and VPSDE. This result implies that the arguably simpler training task in CLD indeed also translates to reduced model complexity in that the neural network is smoother as measured by $\mathcal{J}_{\text{F}}(t)$. In large-scale experiments, this would mean that, given similar model capacity, a CLD-based SGM could potentially have a higher expressivity. Or, on the other hand, similar performance could be achieved with a smoother and potentially smaller model. Indeed these findings are in line with our strong results on the CIFAR-10 benchmark.

2.4.5.2 Image Modeling Experiments

We perform image modeling experiments on CIFAR-10 as well as CelebA-HQ-256. We report FID scores on CIFAR-10 for our main model for various different solvers; see Tab. 2.3 and Tab. 2.2. We further present generated samples for both models in Sec. 2.3.5 using Euler–Maruyama with 2000 quadratic striding steps and Runge–Kutta 4(5) for CIFAR10 and CelebA-HQ-256, respectively. We present additional samples for various solver settings in App. 2.4.6. All (average) NFEs for the Runge–Kutta solver are computed using a batch size of 128.

2.4.5.2.1 Training Details and Model Architectures Our models are based on the NCSN++ and the DDPM++ architectures from Song et al. [263]. Importantly, we changed the number of input channels from three to six to facilitate the additional velocity variables. Note that the number of additional neural network parameters due to this change is negligible.

For fair a comparison, we train our models using the same t-sampling cutoff during training as is used for VESDE and VPSDE in Song et al. [263]. Note, however, that this is not strictly necessary for CLD as we do not have any "blow-up" of the SDE due to unbounded scores as $t \to 0$ (also see Fig. 2.18 and Fig. 2.19).

We summarize our three model architectures as well as our SDE and training setups in Tab. 2.6.

2.4.5.2.2 CIFAR-10 Results for VESDE and VPSDE The results reported for VESDE and VPSDE using the GGF sampler are taken from Jolicoeur-Martineau et al. [134]. All other results for VESDE and VPSDE are generated using the provided PyTorch code as well as the provided checkpoints from Song et al. [263].[7] We used EM and PC to sample from the VPSDE and VESDE models, respectively (see Sec. 2.3.5.2), since these choices correspond to their recommended settings.[8]

Furthermore, in App. 2.4.6.2 we also used DDIM [258] to sample the VPSDE. DDIM's update rule is

$$\mathbf{x}_{t-1} = \frac{\alpha_{t-1}}{\alpha_t}\left[\mathbf{x}_t + \sigma_t^2 s_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right] - \sigma_{t-1}\sigma_t s_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \tag{2.106}$$

[7] https://github.com/yang-song/score_sde_pytorch

[8] https://colab.research.google.com/drive/1dRR_0gNRmfLtPavX2APzUggBuXyjWW55

Hyperparameter	CIFAR10 (Main)	CelebA (Qualitative)	CIFAR10 (Ablation)
Model			
EMA rate	0.9999	0.9999	0.9999
# of ResBlocks per Resolution	8	2	2
Normalization	Group Normalization	Group Normalization	Group Normalization
Scaling by σ	✗	✗	✗
Nonlinearity	Swish	Swish	Swish
Attention resolution	16	16	16
Embedding type	Fourier	Positional	Positional
Progressive	None	None	None
Progressive input	Residual	None	None
Progressive combine	Sum	N/A	N/A
Finite Impulse Response [316]	✓	✗	✗
# of parameters	$\approx 108M$	$\approx 68M$	$\approx 39M$
Training			
# of iterations	800k	320k	1M
# of warmup iterations	100k	100k	100k
Optimizer	Adam	Adam	Adam
Mixed precision	✓	✓	✓
Learning rate	$2 \cdot 10^{-4}$	10^{-4}	$2 \cdot 10^{-4}$
Gradient norm clipping	1.0	1.0	1.0
Dropout	0.1	0.1	0.1
Batch size per GPU	8	4	8
# of GPUs	16	16	16
t-sampling cutoff during training	10^{-5}	10^{-5}	10^{-5}
SDE			
M	0.25	0.25	varies
γ	0.04	0.04	varies
β	4	4	varies
ϵ_{num}	10^{-9}	10^{-6}	10^{-9}

Table 2.6: Model architectures as well as SDE and training setups for our experiments on CIFAR-10 and CelebA-HQ-256.

where $\alpha_t = \exp\Big(-0.5 \int_0^t \beta(t)\, dt\Big)$, $\sigma_t^2 = 1 - \exp\Big(-\int_0^t \beta(t)\, dt\Big)$, and $\beta(t) = 0.1 + 19.9t$.

2.4.5.2.3 Quadratic Striding When we simulate our generative SDE numerically, using for example EM or our SSCS, we need to choose a time discretization. Given a certain NFE budget N_{NFE}, how do we choose time step sizes? The standard approach is to use an equidistant discretization, corresponding to a set of evaluation time steps $\{\delta t_i \geq 0\}_{i=0}^{N_{\text{NFE}}-1}$ with $\delta t_i = \frac{1}{N_{\text{NFE}}} \forall\, i \in [0, N_{\text{NFE}} - 1]$. However, prior work [156, 258, 285] has shown that it can be beneficial to focus function evaluations (neural network calls) on times t "close to the data". This is because the diffusion process distribution is most complex close to the data and almost perfectly Normal close to the prior. Among other techniques, these works used a useful heuristic, denoted as *quadratic striding* (QS), which discretizes the integration interval such that the evaluation times follow a quadratic schedule and the individual time steps follow a linear schedule. We also used this QS approach in our experiments.

We can formally define it as follows (assuming a time interval $[0.0, 1.0]$ here for simplicity): Denote the evaluation times as τ_i (including 0.0 and 1.0) and define:

$$\tau_i = c_\tau i^2 \quad \forall i \in [0, N_{\text{NFE}}]. \tag{2.107}$$

Hence,

$$\delta t_i = \tau_i - \tau_{i-1} = c_\tau (2i - 1) \quad \forall i \in [1, N_{\text{NFE}}], \tag{2.108}$$

and $c_\tau = \frac{1}{N_{\text{NFE}}^2}$ to ensure that $\tau_{\text{NFE}} = 1.0$.

This describes the time steps as going from $t = 0$ to $t = 1$. During synthesis, however, we are going backwards. Hence, we can define our time steps as

$$\delta t_j = c_\tau \left[2N_{\text{NFE}} - 2j + 1\right] \quad \forall j \in [1, N_{\text{NFE}}], \tag{2.109}$$

where j now counts time steps in the other direction. Note that this can be easily adapted to general integration intervals $[\epsilon, T]$.

2.4.5.2.4 Denoising As has been pointed out in Jolicoeur-Martineau et al. [135], samples that are generated with models similar to ours can contain noise that is hard to detect visually but worsens FID scores significantly.

Denoising Formulas. For a fair comparison we use the same denoising setup for all

experiments we conducted (including VESDE (PC/ODE) and VPSDE (EM/ODE)) except for LSGM.[9] We simulate the underlying generative ODE/SDE until the time cutoff $\varepsilon = 10^{-3}$ and then take a single denoising step of the form

$$\mathbf{u}_0 = \mathbf{u}_\varepsilon - \varepsilon \boldsymbol{f}(\mathbf{u}_\varepsilon, \varepsilon) + \varepsilon \boldsymbol{G}(\mathbf{u}_\varepsilon, \varepsilon)\boldsymbol{G}(\mathbf{u}_\varepsilon, \varepsilon)^\top \begin{pmatrix} \mathbf{0}_d \\ \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{u}_\varepsilon, \varepsilon) \end{pmatrix}. \tag{2.110}$$

This denoising step can be considered as an Euler–Maruyama step without noise injection. For SDEs acting on data directly (VESDE, VPSDE, etc.) the corresponding denoising formula is

$$\mathbf{x}_0 = \mathbf{x}_\varepsilon - \varepsilon \boldsymbol{f}(\mathbf{x}_\varepsilon, \varepsilon) + \varepsilon \boldsymbol{G}(\mathbf{x}_\varepsilon, \varepsilon)\boldsymbol{G}(\mathbf{x}_\varepsilon, \varepsilon)^\top \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_\varepsilon, \varepsilon). \tag{2.111}$$

Influence of Denoising on Results. For SDEs acting in the data space directly, it has been reported that this denoising step is crucial to obtain good FID scores Jolicoeur-Martineau et al. [135], Song et al. [263]. When we simulate the generative probability flow ODE we found that denoising is important in order for the Runge–Kutta solver not to "blow-up" as $t \to 0$. On the other hand, when simulating CLD using our new SSCS solver, we found that denoising only slightly influences FID (see Tab. 2.7). We believe that this might be because the neural network does not have any influence on the denoising step for CLD. More specifically, the neural network only denoises the velocity component. However, we are primarily interested in the data component. Putting the drift and diffusion coefficients of CLD in the denoising formula in Eq. (2.110), we obtain

$$\mathbf{u}_0 = \mathbf{u}_\varepsilon - \varepsilon \left(\begin{pmatrix} 0 & \beta M^{-1} \\ -\beta & -\beta \Gamma M^{-1} \end{pmatrix} \otimes \boldsymbol{I}_d \right) \mathbf{u}_\varepsilon + \varepsilon \begin{pmatrix} \mathbf{0}_d \\ 2\Gamma \beta s_{\boldsymbol{\theta}}(\mathbf{u}_\varepsilon, \varepsilon) \end{pmatrix} \implies \mathbf{x}_0 = \mathbf{x}_\varepsilon - \varepsilon \beta M^{-1} \mathbf{v}_\varepsilon. \tag{2.112}$$

2.4.5.2.5 Solver Error Tolerances for Runge–Kutta 4(5) In Tab. 2.2, we report FID scores for a Runge–Kutta 4(5) solver [77] as well as the "Gotta Go Fast" solver from Jolicoeur-Martineau et al. [134] (see their Table 1). For simulating CLD with Runge–Kutta 4(5) we chose the solver error tolerances to hit certain regimes of NFEs to facilitate comparisons with VPSDE and VESDE. We obtain a mean number of function evaluations of 312 and 137 using Runge–Kutta 4(5) solver error tolerances of 10^{-5} and 10^{-3},

[9] Denoising has not been used in the original LSGM work [280] and is not needed in their case, since the output of the latent SGM lives in a smooth latent space and is further processed by a decoder.

Table 2.7: Influence of denoising step on FID scores (using our main CIFAR-10 model).

Sampler	Denoising	FID at n function evaluations ↓ n=50	n=500
SSCS	✓	81.1	2.30
SSCS-QS	✓	20.5	2.25
SSCS	✗	78.9	2.32
SSCS-QS	✗	28.5	2.3

respectively. For VESDE, VPSDE and LSGM we used 10^{-5} as the ODE solver error tolerance, following the recommended default setups [263, 280]. These values are used for both relative and absolute error tolerances.

2.4.5.2.6 Ablation Experiments The model architecture used for all ablation experiments can be found in Tab. 2.6. As pointed out in Sec. 2.3.5 we found that the hyperparameters γ and M only have small effects on CIFAR-10 FID scores. On the other hand, we found that the mixed score parameterization helps significantly in obtaining competitive FIDs.

2.4.5.2.7 LSGM-100M Model Our CLD-based SGM has $\approx 108M$ parameters, while the original CIFAR-10 Latent SGM from [280], to which we compare in Tab. 2.1, uses $\approx 476M$ parameters. To establish a fairer comparison between our CLD-based SGMs and LSGM [280], we train another smaller LSGM model with $\approx 109M$ parameters. To do this, we followed the exact setup of the "CIFAR-10 (balanced)" model from LSGM (see Table 7 in Vahdat et al. [280]), with a few minor modifications: We used a VAE backbone model with only 10 groups instead of 20, which corresponds to a reduction in parameters by a factor of 2 in the encoder and decoder networks. We also reduced the convolutional channels in the latent space SGM from 512 to 256 and reduced the number of the residual cells per scale from 8 to 4. With these modifications the resulting "LSGM-100M" uses only $\approx 109M$ parameters overall with approximately half of them in the encoder and decoder networks and the other half in the latent SGM. Other than these architecture modifications, our model is trained in exactly the same way as the bigger, original models in Vahdat et al. [280].

For evaluation, we follow the recommended setting by Vahdat et al. [280] and use the

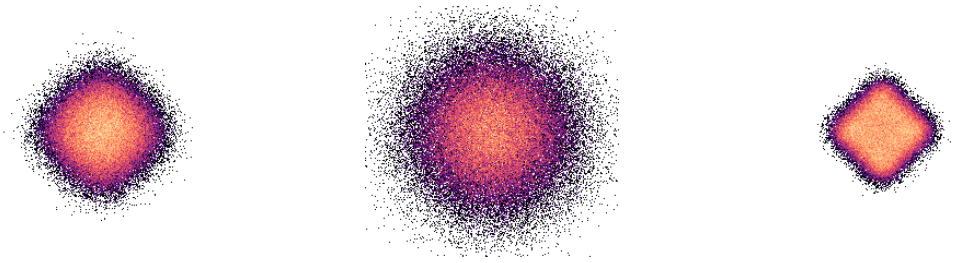
same Runge-Kutta 4(5) ODE solver with an error tolerance of 10^{-5} to solve the probability flow ODE in LSGM's latent space. LSGM-100M achieves an FID of 4.60, an NLL bound of 2.96 bpd, and requires on average 131 NFE for sampling new images. We report these results in Tabs. 2.1 and 2.2 in the main text.

Note that we also tried training a model following the "CIFAR-10 (best FID)" setup, but found training to be unstable (however, the orignal "CIFAR-10 (best FID)" model from Vahdat et al. [280] only performs marginally better in FID than their "CIFAR-10 (balanced)" model anyway). Furthermore, we also tried training another small LSGM with a similar number of parameters but with more parameters in the latent SGM and less in the encoder and decoder networks, compared to the reported LSGM-100M. However, this model performed significantly worse.

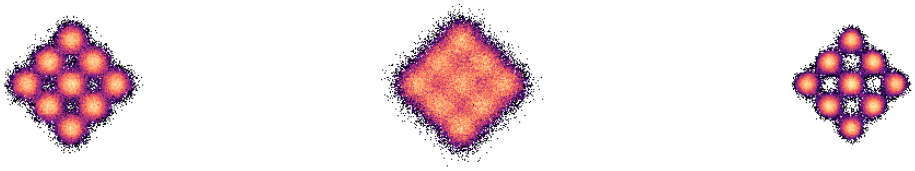
2.4.6 Additional Experiments

2.4.6.1 Toy Experiments

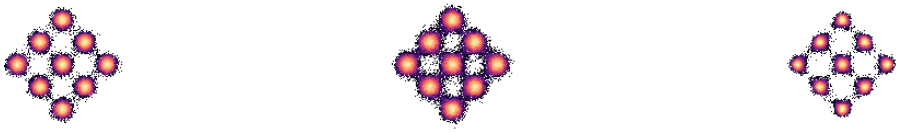
2.4.6.1.1 Analytical Sampling In order to test combinations of diffusions and numerical samplers in isolation, we consider a dataset for which we know the ground truth score function (for all t) analytically. In particular, we use the mixture of Normals introduced in App. 2.4.5.1; see Fig. 2.10a for a visualization of the data distribution. In Fig. 2.11, we show samples for VPSDE (Euler–Maruyama (EM) sampler) and CLD (EM and SSCS samplers). For quantitative comparison, we also compute negative log-likelihoods for the three combinations (which can be done easily due to our access to the ground truth distribution): as can be seen in Tab. 2.8, for each number of steps $n \in \{20, 50, 100, 200\}$ CLD with SCSS outperforms both VPSDE and CLD with EM. As discussed in Sec. 2.3.3.3, we can see in Tab. 2.8 that EM is not well-suited for CLD. This is true, in particular, when using a small number of synthesis steps n (function evaluations). In Fig. 2.11, we see that CLD with EM leads to sampling distributions which are too broad. These results are exactly in line with the "diverging" dynamics that is observed when solving Hamiltonian dynamics with a non-symplectic integrator, such as the standard Euler method [209]. This problematic behavior of Euler-based techniques is more pronounced when using fewer steps with larger stepsizes, which is also what we observe in our experiments. These results further motivate the use of our novel SSCS, which addresses these challenges, for sampling from our CLD-based SGMs.



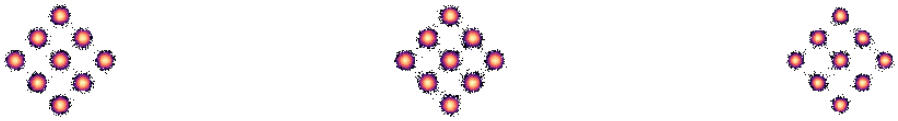
(a) VPSDE + EM, $n = 20$ (b) CLD + EM, $n = 20$ (c) CLD + SSCS, $n = 20$



(d) VPSDE + EM, $n = 50$ (e) CLD + EM, $n = 50$ (f) CLD + SSCS, $n = 50$



(g) VPSDE + EM, $n = 100$ (h) CLD + EM, $n = 100$ (i) CLD + SSCS, $n = 100$



(j) VPSDE + EM, $n = 200$ (k) CLD + EM, $n = 200$ (l) CLD + SSCS, $n = 200$

Figure 2.11: Mixture of Normals: numerical simulation with analytical score function for different diffusions (VPSDE with EM vs. CLD with EM/SSCS) and number of synthesis steps n. A visualization of the data distribution can be found in Fig. 2.10a.

Table 2.8: Performance (measured in negative log-likelihood) using analytical scores for non-adaptive stepsize solvers for varying numbers of synthesis steps n (function evaluations).

		$-\log p(x)$ at n function evaluations ↓			
Model	Sampler	n=20	n=50	n=100	n=200
CLD	EM	60.6	9.71	0.72	-1.04
CLD	SSCS	**10.5**	**1.55**	**-1.25**	**-1.54**
VPSDE	EM	14.2	4.68	-0.35	-1.11

2.4.6.1.2 Maximum Likelihood Training For maximum likelihood training, models based on overdamped Langevin dynamics such as VPSDE need to learn an unbounded score for $t \to 0$. Our model, on the other hand, only ever needs to learn a bounded score even for $t = 0$. For our image data experiments, we use a reweighted objective function to improve visual quality of samples (as is general practice).

Here, we also study training towards maximum likelihood on toy dataset tasks. To explore this, we repeat the neural network complexity experiment from App. 2.4.5.1 with maximum likelihood training (instead of the reweighted objective). Furthermore, we also train VPSDE-based and CLD-based SGMs on a challenging toy dataset and find that CLD significantly outperforms VPSDE. We leave the study of CLD with maximum likelihood training for high-dimensional (image) datasets to future work.

Complexity Experiment. The setup of this experiment is equivalent to the setup in App. 2.4.5.1 up to the training objective: in this experiment we do maximum likelihood learning, i.e., we train CLD models with the objective from Eq. (2.11) with $\lambda(t) = \Gamma\beta$.[10] Furthermore, we test CLD in this setup for three different values of γ. The results of this experiment can be found in Fig. 2.12. For CLD, we find that larger values of γ generally lead to less complex networks, in particular for smaller times t. However, even for $\gamma = 0.04$ the learned neural network is still significantly smoother than the network learned for the VPSDE when a mixed score parameterization is used.[11] **Challenging Toy Dataset.** Using the same simple ResNet architecture (less than 100k parameters) from the above experiment, we trained a VPSDE-based as well as a CLD-based SGM to maximize the likelihood of a more challenging toy dataset (the dataset is essentially "multi-scale", as it involves both large scale—the placement of the swiss rolls—and fine scale—the swiss rolls

[10] For the ML objective of the VPSDE, we refer the reader to Song et al. [262].

[11] The VPSDE-based model with mixed score parameterization did not converge to the target distribution, and therefore is not included in Fig. 2.12.

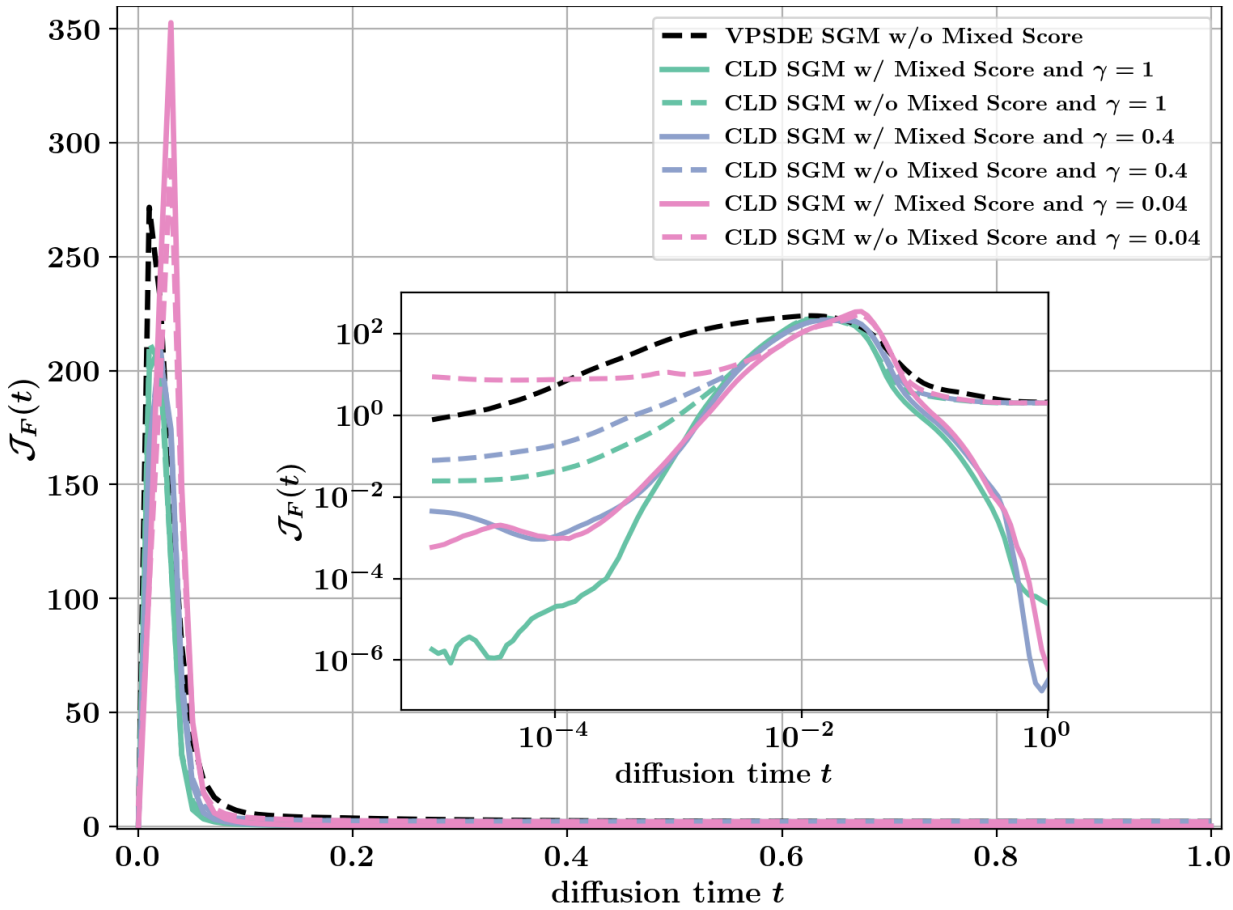
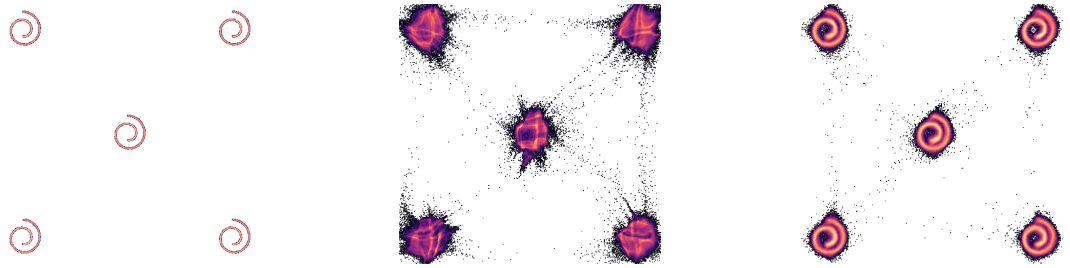


Figure 2.12: Frobenius norm $\mathcal{J}_F(t)$ of the neural network defining the score function for different t.



(a) Data (b) VPSDE (c) CLD

Figure 2.13: Data distribution and model samples for multi-scale toy experiment.

themselves—structure). Similar to the other toy datasets, the models are trained for 1M iterations using fresh data synthesized from the data distribution in each batch at a batch size of 512.

In Fig. 2.13, we compare samples of the models to the data distribution. Even with our simple model architecture, CLD is able to capture the multi-scale structure of the dataset: the five rolls are adequately resembled and only a few samples are in between modes. VPSDE, on the other hand, only captures the main modes, but not the fine structure. Furthermore, VPSDE has the undesired behavior of "connecting" the modes.

Overall, we conclude that also in the maximum likelihood training setting CLD is a promising diffusion showing superior behavior compared to the VPSDE in our toy experiments.

2.4.6.2 CIFAR-10 — Extended Results

In this section, we provide additional results on the CIFAR-10 image modeling benchmark.

An extended version of Tab. 2.3 (sampling the generative SDE with different fixed-step size solvers for different compute budgets) including additional baselines can be found in Tab. 2.9. Note that time stepping with quadratic striding (QS) improves sampling from VPSDE- and CLD-based models for all settings except for the combination of VPSDE and EM sampling in the setting $n = \{1000, 2000\}$. For the VESDE (using PC sampling), QS significantly worsens FID scores. The reason for this could be that the variance of the VESDE already follows an exponential schedule (see Fig. 5 in Song et al. [263]). We additionally present results for the VPSDE using the DDIM (Denoising Diffusion Implicit

Table 2.9: Performance using non-adaptive step size solvers. Extended version of Tab. 2.3.

		FID at n function evaluations ↓					
Model	Sampler	n=50	n=150	n=275	n=500	n=1000	n=2000
CLD	EM	143	31.5	10.9	3.96	2.50	2.27
CLD	EM-QS	52.7	7.00	3.24	2.41	**2.27**	**2.23**
CLD	SSCS	81.1	10.5	2.86	2.30	2.32	2.29
CLD	SSCS-QS	20.5	**3.07**	**2.38**	**2.25**	2.30	2.29
VPSDE	EM	92.0	30.3	13.1	4.42	2.46	2.43
VPSDE	EM-QS	28.2	4.06	2.65	2.47	2.66	2.60
VPSDE	DDIM	6.04	4.04	3.53	3.26	3.09	3.01
VPSDE	DDIM-QS	**3.78**	3.15	3.05	2.99	2.96	2.95
VESDE	PC	460	216	11.2	3.75	2.43	**2.23**
VESDE	PC-QS	461	388	155	5.47	11.4	11.2

Models) sampler [258]. As was observed by [258], QS also helps for DDIM. Importantly, for any $n \geq 150$, our CLD with our novel SSCS (and QS) even outperforms DDIM. Only for $n = 50$, DDIM performs better. It needs to be mentioned, however, that the DDIM sampler was specifically designed for few-step sampling, whereas our CLD with SSCS is derived in a general fashion without this particular regime in mind. In particular, DDIM sampling can be interpreted as a non-Markovian sampling method and it is not clear how to calculate the log-likelihood of hold-out validation data under this non-Markovian synthesis approach. Nevertheless, it would be interesting to also explore non-Markovian DDIM-inspired techniques for CLD-SGMs to further improve sampling speed in CLD-SGMs.

Note that our DDIM results shown in Tab. 2.9 are better than those presented in [258] itself, because we are relying on the DDPM++ model trained in [263], whereas [258] uses the DDPM model from [109].

Finally, we present additional generated samples from our CLD-SGM model: see Fig. 2.14 and Fig. 2.15 for samples from EM-QS with 2000 evaluations and SSCS-QS with 150 evaluations, respectively.

Figure 2.14: Additional samples using EM-QS with 2000 function evaluations. This setup gave us our best FID score of 2.23.

Figure 2.15: Additional samples using SSCS-QS. This setup resulted in an FID score of 3.07 using only 150 function evaluations.

2.4.6.3 CelebA-HQ-256 — Extended Results

In this section, we provide additional qualitative results on CelebA-HQ-256. For high quality samples using our new SSCS solver see Fig. 2.16.

Samples generated with an adaptive step size Runge–Kutta 4(5) solver at different solver tolerances can be found in Fig. 2.17. We found that our model still generates very good samples even for a solver error tolerance of 10^{-3} using an average of 129 neural network evaluations.

Lastly, we show "generation paths" of samples from our CelebA-HQ-256 model: see Fig. 2.18 and Fig. 2.19 for samples from the probability flow ODE and the generative SDE, respectively. We visualize the continuous generation paths via snapshots of data and velocity variables at eight different time steps. Interestingly, we can see that the velocity variables "encode" the data at intermediate t. On the other hand, at time $t = 1.0$, by construction, both data and velocity are distributed according to the "equilibrium distribution" of the diffusion, namely, $p_{\mathrm{EQ}}(\mathbf{u}) = \mathcal{N}(\mathbf{x}; \mathbf{0}_d, \boldsymbol{I}_d)\,\mathcal{N}(\mathbf{v}; \mathbf{0}_d, M\boldsymbol{I}_d)$. Furthermore, as $t \to 0$ the data variables approximately converge to the data distribution, while the velocity variables approximately converge to another Normal distribution $\mathcal{N}(\mathbf{v}; \mathbf{0}_d, \gamma M\boldsymbol{I}_d)$ (with $\gamma = 0.04$ in our experiments).

Recall that for CLD, the neural network approximates the score $\nabla_{\mathbf{v}_t} \log p_t(\mathbf{v}_t|\mathbf{x}_t)$. We believe that the generation paths are further evidence that CLD-SGMs need to learn simpler models: for fixed t the velocity variable $\mathbf{v}_t$ appears to be a "noisy" version of the data $\mathbf{x}_t$, and therefore we believe $p_t(\mathbf{v}_t|\mathbf{x}_t)$ to be relatively smooth and simple when compared to the marginal $p_t(\mathbf{x}_t)$.

Finally, note that in Figs. 2.18 and 2.19, when visualizing the velocity variables, we used a colorization scheme that corresponds exactly to the inverse of the color scheme used for visualizing the images themselves. Alternatively, we could also interpret this in such a way that we are not actually visualizing velocities, but negative velocities with flipped signs. When using this inverse colorization scheme for the velocities, we see that at intermediate t, where the velocities encode the data, the color values visualizing image data and velocities are, apart from the additional noise in the velocities, similar (i.e. the velocities appear as noisy versions of the actual images). This implies that image pixel values $\mathbf{x}_t$ translate into corresponding *negative* velocities $\mathbf{v}_t$ that pull the pixel values back towards the mean of the equilibrium distribution. This is a consequence of the Hamiltonian coupling between the data and velocity variables. In other words, it is a result of the negative sign in front of

$\mathbf{x}_t$ in the H term in Eq. (2.8) (and analogously for the reverse-time generative SDE). Also see the visualizations on our project page (https://nv-tlabs.github.io/CLD-SGM).

2.4.7 Proofs of Perturbation Kernels

In this section, we prove the correctness of the perturbation kernels of the forward diffusion (App. 2.4.2.1) as well as for the analytical splitting term in our SSCS (App. 2.4.4.2). All derivations are presented for general time-dependent $\beta(t)$.

2.4.7.1 Forward Diffusion

We have the following ODEs describing the evolution of the mean and the covariance matrix

$$\frac{d\boldsymbol{\mu}_t}{dt} = (f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\mu}_t, \tag{2.113}$$

$$\frac{d\boldsymbol{\Sigma}_t}{dt} = (f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\Sigma}_t + [(f(t) \otimes \boldsymbol{I}_d)\boldsymbol{\Sigma}_t]^\top + \left(G(t)G^\top(t)\right) \otimes \boldsymbol{I}_d, \tag{2.114}$$

where

$$f(t) \coloneqq \begin{pmatrix} 0 & 4\beta(t)\Gamma^{-2} \\ -\beta(t) & -4\beta(t)\Gamma^{-1} \end{pmatrix}, \tag{2.115}$$

$$G(t) \coloneqq \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{2\Gamma\beta(t)} \end{pmatrix}. \tag{2.116}$$

In App. 2.4.2.1, we claim the following solutions:

$$\boldsymbol{\mu}_t \coloneqq C_t \hat{\boldsymbol{\mu}}_t, \tag{2.117}$$

$$\hat{\boldsymbol{\mu}}_t \coloneqq \begin{pmatrix} \boldsymbol{\mu}_t^x \\ \boldsymbol{\mu}_t^v \end{pmatrix}, \tag{2.118}$$

$$C_t \coloneqq e^{-2\mathcal{B}(t)\Gamma^{-1}}, \tag{2.119}$$

$$\boldsymbol{\mu}_t^x \coloneqq 2\mathcal{B}(t)\Gamma^{-1}\mathbf{x}_0 + 4\mathcal{B}(t)\Gamma^{-2}\mathbf{v}_0 + \mathbf{x}_0, \tag{2.120}$$

$$\boldsymbol{\mu}_t^v \coloneqq -\mathcal{B}(t)\mathbf{x}_0 - 2\mathcal{B}(t)\Gamma^{-1}\mathbf{v}_0 + \mathbf{v}_0, \tag{2.121}$$

Figure 2.16: Samples generated by our model on the CelebA-HQ-256 dataset using our SSCS solver.

(a) ODE solver error tolerance 10^{-5}; 273 average NFE.

(b) ODE solver error tolerance 10^{-4}; 190 average NFE.

(c) ODE solver error tolerance 10^{-3}; 129 average NFE.

(d) ODE solver error tolerance 10^{-2}; 99.4 average NFE.

Figure 2.17: Samples generated by our model on the CelebA-HQ-256 dataset using a Runge–Kutta 4(5) adaptive ODE solver to solve the probability flow ODE. We show the effect of the ODE solver error tolerance on the quality of samples ((a), (b), (c) and (d) were generated using the same prior samples). Little visual differences can be seen between 10^{-5} and 10^{-4}. Low frequency artifacts can be observed at 10^{-3}. Deterioration starts to set in at 10^{-2}.

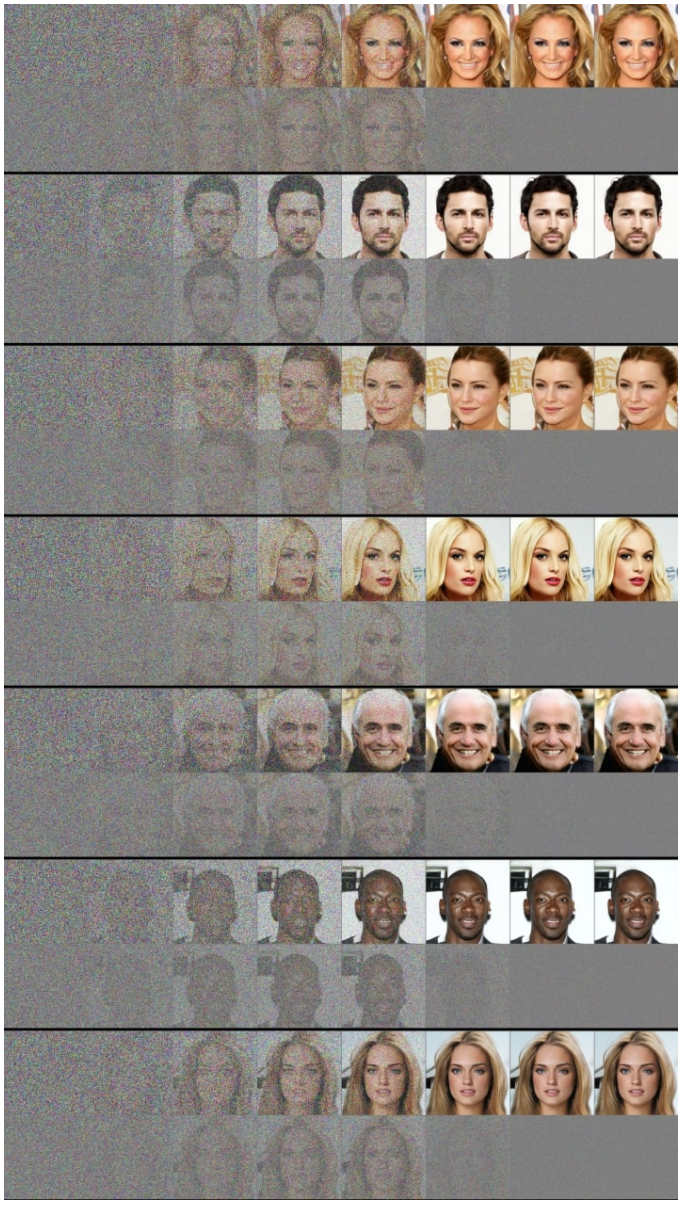


Figure 2.18: Generation paths of samples from our CelebA-HQ-256 model (Runge–Kutta 4(5) solver; mean NFE: 288). Odd and even rows visualize data and velocity variables, respectively. The eight columns correspond to times $t \in \{1.0, 0.5, 0.3, 0.2, 0.1, 10^{-2}, 10^{-3}, 10^{-5}\}$ (from left to right). The velocity distribution converges to a Normal (different variances) for both $t \to 0$ and $t \to 1$. See App. 2.4.6.3 for visualization details and discussion.

Figure 2.19: Generation paths of samples from our CelebA-HQ-256 model (SSCS-QS using only 150 steps). Odd and even rows visualize data and velocity variables, respectively. The eight columns correspond to times $t \in \{1.0, 0.5, 0.3, 0.2, 0.1, 10^{-2}, 10^{-3}, 10^{-5}\}$ (from left to right). The velocity distribution converges to a Normal (different variances) for both $t \to 0$ and $t \to 1$. See App. 2.4.6.3 for visualization details and discussion.

and

$$\boldsymbol{\Sigma}_t \coloneqq \Sigma_t \otimes \boldsymbol{I}_d, \tag{2.122}$$

$$\Sigma_t \coloneqq D_t \hat{\Sigma}_t, \tag{2.123}$$

$$\hat{\Sigma}_t \coloneqq \begin{pmatrix} \Sigma_t^{xx} & \Sigma_t^{xv} \\ \Sigma_t^{xv} & \Sigma_t^{vv} \end{pmatrix}, \tag{2.124}$$

$$D_t \coloneqq e^{-4\mathcal{B}(t)\Gamma^{-1}}, \tag{2.125}$$

$$\Sigma_t^{xx} \coloneqq \Sigma_0^{xx} + e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 + 4\mathcal{B}(t)\Gamma^{-1}\left(\Sigma_0^{xx} - 1\right) + 4\mathcal{B}^2(t)\Gamma^{-2}\left(\Sigma_0^{xx} - 2\right) + 16\mathcal{B}^2(t)\Gamma^{-4}\Sigma_0^{vv}, \tag{2.126}$$

$$\Sigma_t^{xv} \coloneqq -\mathcal{B}(t)\Sigma_0^{xx} + 4\mathcal{B}(t)\Gamma^{-2}\Sigma_0^{vv} - 2\mathcal{B}^2(t)\Gamma^{-1}\left(\Sigma_0^{xx} - 2\right) - 8\mathcal{B}^2(t)\Gamma^{-3}\Sigma_0^{vv}, \tag{2.127}$$

$$\Sigma_t^{vv} \coloneqq \frac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma + \Sigma_0^{vv}\left(1 + 4\mathcal{B}^2(t)\Gamma^{-2} - 4\mathcal{B}(t)\Gamma^{-1}\right) + \mathcal{B}^2(t)\left(\Sigma_0^{xx} - 2\right), \tag{2.128}$$

where $\mathcal{B}(t) = \int_0^t \beta(\hat{t})\, d\hat{t}$ and $\boldsymbol{\mu}_0 = [\mathbf{x}_0, \mathbf{v}_0]^\top$ as well as Σ_0^{xx} and Σ_0^{vv} are initial conditions.

2.4.7.1.1 Proof of Correctness of the Mean

Plugging the claimed solution (Eqs. (2.117)-(2.121)) back into the ODE (Eq. 2.113), we obtain

$$\hat{\boldsymbol{\mu}}_t \frac{dC_t}{dt} + \frac{d\hat{\boldsymbol{\mu}}_t}{dt} C_t = C_t (f(t) \otimes \boldsymbol{I}_d)\hat{\boldsymbol{\mu}}_t. \tag{2.129}$$

The above can be decomposed into two equations:

$$-2\beta(t)\Gamma^{-1}\boldsymbol{\mu}_t^x + \frac{d\boldsymbol{\mu}_t^x}{dt} = 4\beta(t)\Gamma^{-2}\boldsymbol{\mu}_t^v, \tag{2.130}$$

$$-2\beta(t)\Gamma^{-1}\boldsymbol{\mu}_t^v + \frac{d\boldsymbol{\mu}_t^v}{dt} = -\beta(t)\boldsymbol{\mu}_t^x - 4\beta(t)\Gamma^{-1}\boldsymbol{\mu}_t^v, \tag{2.131}$$

where we used the fact that $\frac{dC_t}{dt} = -2\beta(t)\Gamma^{-1}C_t$.

Eq. (2.130): Plugging the claimed solution into Eq. (2.130), we obtain:

$$-2\beta(t)\Gamma^{-1}\left[2\mathcal{B}(t)\Gamma^{-1}\mathbf{x}_0 + 4\mathcal{B}(t)\Gamma^{-2}\mathbf{v}_0 + \mathbf{x}_0\right] + \left[2\beta(t)\Gamma^{-1}\mathbf{x}_0 + 4\beta(t)\Gamma^{-2}\mathbf{v}_0\right] = 4\beta(t)\Gamma^{-2}\left[-\mathcal{B}(t)\mathbf{x}_0 - 2\mathcal{B}(t)\Gamma^{-1}\mathbf{v}_0 + \mathbf{v}_0\right]. \tag{2.132}$$

Eq. (2.131): After simplification, plugging in the claimed solution into Eq. (2.131), we obtain:

$$2\beta(t)\Gamma^{-1}\left[-\mathcal{B}(t)\mathbf{x}_0 - 2\mathcal{B}(t)\Gamma^{-1}\mathbf{v}_0 + \mathbf{v}_0\right] + \left[-\beta(t)\mathbf{x}_0 - 2\beta(t)\Gamma^{-1}\mathbf{v}_0\right] = -\beta(t)\left[2\mathcal{B}(t)\Gamma^{-1}\mathbf{x}_0 + 4\mathcal{B}(t)\Gamma^{-2}\mathbf{v}_0 + \mathbf{x}_0\right]. \tag{2.133}$$

This completes the proof of the correctness of the mean.

2.4.7.1.2 Proof of Correctness of the Covariance Plugging the claimed solution (Eqs. (2.122)-(2.128)) back in the ODE (Eq. (2.114)), we obtain

$$\left[\frac{d\hat{\Sigma}_t}{dt}D_t + \frac{dD_t}{dt}\Sigma_t\right] \otimes \boldsymbol{I}_d = D_t(f(t)\otimes \boldsymbol{I}_d)(\hat{\Sigma}_t \otimes \boldsymbol{I}_d) + D_t\left[(f(t)\otimes \boldsymbol{I}_d)(\hat{\Sigma}_t \otimes \boldsymbol{I}_d)\right]^\top + \left(G(t)G^\top(t)\right)\otimes \boldsymbol{I}_d. \quad (2.134)$$

Noting that

$$\begin{aligned}(f(t)\otimes \boldsymbol{I}_d)(\hat{\Sigma}_t \otimes \boldsymbol{I}_d) &= (f(t)\hat{\Sigma}_t)\otimes \boldsymbol{I}_d \\ &= \beta(t)\begin{pmatrix} 4\Gamma^{-2}\Sigma_t^{xv} & 4\Gamma^{-2}\Sigma_t^{vv} \\ -\Sigma_t^{xx} - 4\Gamma^{-1}\Sigma_t^{xv} & -\Sigma_t^{xv} - 4\Gamma^{-1}\Sigma^{vv}\end{pmatrix}\otimes \boldsymbol{I}_d,\end{aligned} \quad (2.135)$$

and

$$G(t)G^\top(t) = \begin{pmatrix} 0 & 0 \\ 0 & 2\Gamma\beta(t)\end{pmatrix}, \quad (2.136)$$

as well as the fact that $\frac{dD_t}{dt} = -4\beta(t)\Gamma^{-1}D_t$, we can decompose Eq. (2.134) into three equations:

$$-4\beta(t)\Gamma^{-1}\Sigma_t^{xx} + \frac{d\Sigma^{xx}}{dt} = 8\beta(t)\Gamma^{-2}\Sigma_t^{xv}, \quad (2.137)$$

$$-4\beta(t)\Gamma^{-1}\Sigma_t^{xv} + \frac{d\Sigma^{xv}}{dt} = \beta(t)\left[-\Sigma_t^{xx} - 4\Gamma^{-1}\Sigma_t^{xv} + 4\Gamma^{-2}\Sigma_t^{vv}\right], \quad (2.138)$$

$$-4\beta(t)\Gamma^{-1}\Sigma_t^{vv} + \frac{d\Sigma^{vv}}{dt} = \beta(t)\left[-2\Sigma_t^{xv} - 8\Gamma^{-1}\Sigma_t^{vv}\right] + 2\Gamma\beta(t)D_t^{-1}. \quad (2.139)$$

Eq. (2.137): Plugging the claimed solution into Eq. (2.137), we obtain

$$\begin{aligned}&-4\beta(t)\Gamma^{-1}\left[\Sigma_0^{xx} + e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 + 4\mathcal{B}(t)\Gamma^{-1}(\Sigma_0^{xx}-1) + 4\mathcal{B}^2(t)\Gamma^{-2}(\Sigma_0^{xx}-2) + 16\mathcal{B}^2(t)\Gamma^{-4}\Sigma_0^{vv}\right] \\ &+\left[4\beta(t)\Gamma^{-1}e^{4\mathcal{B}(t)\Gamma^{-1}} + 4\beta(t)\Gamma^{-1}(\Sigma_0^{xx}-1) + 8\beta(t)\mathcal{B}(t)\Gamma^{-2}(\Sigma_0^{xx}-2) + 32\beta(t)\mathcal{B}(t)\Gamma^{-4}\Sigma_0^{vv}\right] \\ &= 8\beta(t)\Gamma^{-2}\left[-\mathcal{B}(t)\Sigma_0^{xx} + 4\mathcal{B}(t)\Gamma^{-2}\Sigma_0^{vv} - 2\mathcal{B}^2(t)\Gamma^{-1}(\Sigma_0^{xx}-2) - 8\mathcal{B}^2(t)\Gamma^{-3}\Sigma_0^{vv}\right].\end{aligned} \quad (2.140)$$

Eq. (2.138): After simplification, plugging the claimed solution into Eq. (2.138), we obtain

$$\begin{aligned}&\left[-\beta(t)\Sigma_0^{xx} + 4\beta(t)\Gamma^{-2}\Sigma_0^{vv} - 4\beta(t)\mathcal{B}(t)\Gamma^{-1}(\Sigma_0^{xx}-2) - 16\beta(t)\mathcal{B}(t)\Gamma^{-3}\Sigma_0^{vv}\right] \\ &= -\beta(t)\left[\Sigma_0^{xx} + e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 + 4\mathcal{B}(t)\Gamma^{-1}(\Sigma_0^{xx}-1) + 4\mathcal{B}^2(t)\Gamma^{-2}(\Sigma_0^{xx}-2) + 16\mathcal{B}^2(t)\Gamma^{-4}\Sigma_0^{vv}\right] \\ &+ 4\beta(t)\Gamma^{-2}\left[\frac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma + \Sigma_0^{vv}\left(1 + 4\mathcal{B}^2(t)\Gamma^{-2} - 4\mathcal{B}(t)\Gamma^{-1}\right) + \mathcal{B}^2(t)(\Sigma_0^{xx}-2)\right].\end{aligned} \quad (2.141)$$

Eq. (2.139): After simplification, plugging the claimed solution into Eq. (2.139), we obtain

$$\begin{aligned}
&4\beta(t)\Gamma^{-1}\left[\frac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}}-1\right)+\mathcal{B}(t)\Gamma+\Sigma_0^{vv}\left(1+4\mathcal{B}^2(t)\Gamma^{-2}-4\mathcal{B}(t)\Gamma^{-1}\right)+\mathcal{B}^2(t)\left(\Sigma_0^{xx}-2\right)\right]\\
&+\left[\frac{\Gamma^2}{4}4\beta(t)\Gamma^{-1}e^{4\mathcal{B}(t)\Gamma^{-1}}+\beta(t)\Gamma+\Sigma_0^{vv}\left(8\mathcal{B}(t)\beta(t)\Gamma^{-2}-4\beta(t)\Gamma^{-1}\right)+2\beta(t)\mathcal{B}(t)\left(\Sigma_0^{xx}-2\right)\right]\\
&=-2\beta(t)\left[-\mathcal{B}(t)\Sigma_0^{xx}+4\mathcal{B}(t)\Gamma^{-2}\Sigma_0^{vv}-2\mathcal{B}^2(t)\Gamma^{-1}\left(\Sigma_0^{xx}-2\right)-8\mathcal{B}^2(t)\Gamma^{-3}\Sigma_0^{vv}\right]\\
&+2\Gamma\beta(t)e^{4\mathcal{B}(t)\Gamma^{-1}}.
\end{aligned} \tag{2.142}$$

This completes the proof of the correctness of the covariance.

2.4.7.2 Analytical Splitting Term of SSCS

We have the following ODEs describing the evolution of the mean and the covariance matrix

$$\frac{d\bar{\boldsymbol{\mu}}_t}{dt}=(f(T-t)\otimes\boldsymbol{I}_d)\bar{\boldsymbol{\mu}}_t, \tag{2.143}$$

$$\frac{d\bar{\boldsymbol{\Sigma}}_t}{dt}=(f(T-t)\otimes\boldsymbol{I}_d)\bar{\boldsymbol{\Sigma}}_t+\left[(f(T-t)\otimes\boldsymbol{I}_d)\bar{\boldsymbol{\Sigma}}_t\right]^\top+\left(G(T-t)G^\top(T-t)\right)\otimes\boldsymbol{I}_d, \tag{2.144}$$

where

$$f(T-t)\coloneqq\begin{pmatrix}0 & -4\beta(T-t)\Gamma^{-2}\\ +\beta(T-t) & -4\beta(T-t)\Gamma^{-1}\end{pmatrix}, \tag{2.145}$$

$$G(T-t)\coloneqq\begin{pmatrix}0 & 0\\ 0 & \sqrt{2\Gamma\beta(T-t)}\end{pmatrix}. \tag{2.146}$$

These ODEs are very similar to the ODEs of the forward diffusion in App. 2.4.7.1, the only difference being flipped signs in the off-diagonal terms of $f(T-t)$ (highlighted in red).

In App. 2.4.4.2, we claim the following solutions

$$\bar{\boldsymbol{\mu}}_t=C_t\tilde{\boldsymbol{\mu}}_t, \tag{2.147}$$

$$\tilde{\boldsymbol{\mu}}_t=\begin{pmatrix}\bar{\boldsymbol{\mu}}_t^x\\ \bar{\boldsymbol{\mu}}_t^v\end{pmatrix}, \tag{2.148}$$

$$C_t=e^{-2\mathcal{B}(t)\Gamma^{-1}}, \tag{2.149}$$

$$\bar{\boldsymbol{\mu}}_t^x=2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{x}}_{t'}-4\mathcal{B}(t)\Gamma^{-2}\bar{\mathbf{v}}_{t'}+\bar{\mathbf{x}}_{t'}, \tag{2.150}$$

$$\bar{\boldsymbol{\mu}}_t^v=+\mathcal{B}(t)\bar{\mathbf{x}}_{t'}-2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{v}}_{t'}+\bar{\mathbf{v}}_{t'}, \tag{2.151}$$

and

$$\bar{\boldsymbol{\Sigma}}_t = \bar{\Sigma}_t \otimes \boldsymbol{I}_d, \tag{2.152}$$

$$\bar{\Sigma}_t = D_t \tilde{\Sigma}_t, \tag{2.153}$$

$$\tilde{\Sigma}_t = \begin{pmatrix} \bar{\Sigma}_t^{xx} & \bar{\Sigma}_t^{xv} \\ \bar{\Sigma}_t^{xv} & \bar{\Sigma}_t^{vv} \end{pmatrix}, \tag{2.154}$$

$$D_t = e^{-4\mathcal{B}(t)\Gamma^{-1}}, \tag{2.155}$$

$$\bar{\Sigma}_t^{xx} = e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 - 4\mathcal{B}(t)\Gamma^{-1} - 8\mathcal{B}^2(t)\Gamma^{-2}, \tag{2.156}$$

$$\bar{\Sigma}_t^{xv} = -4\mathcal{B}^2(t)\Gamma^{-1}, \tag{2.157}$$

$$\bar{\Sigma}_t^{vv} = \frac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma - 2\mathcal{B}^2(t), \tag{2.158}$$

where $\mathcal{B}(t) = \int_{t'}^{t} \beta(T - \hat{t})\, d\hat{t}$ and $\bar{\boldsymbol{\mu}}_{t'} = [\bar{\mathbf{x}}_{t'}, \bar{\mathbf{v}}_{t'}]^\top$ is an initial condition. Differences of the above solution to the solutions of the forward diffusion are again highlighted in red. Note that by construction the initial covariance for the analytical splitting term of SSCS is the zero matrix, i.e., $\bar{\Sigma}_{t'}^{xx} = \bar{\Sigma}_{t'}^{xv} = \bar{\Sigma}_{t'}^{vv} = 0$, since we always initialize from an "updated sample", which itself does not have any uncertainty. Also note that in this derivation we use general initial t' (whereas in App. 2.4.7.1 we set $t' = 0$ for simplicity).

2.4.7.2.1 Proof of Correctness of the Mean Plugging the claimed solution (Eqs. (2.147)-(2.151)) into the ODE (Eq. (2.143)), we obtain

$$\tilde{\boldsymbol{\mu}}_t \frac{dC_t}{dt} + \frac{d\tilde{\boldsymbol{\mu}}_t}{dt} C_t = C_t (f(T-t) \otimes \boldsymbol{I}_d)\tilde{\boldsymbol{\mu}}_t. \tag{2.159}$$

The above can be decomposed into two equations:

$$-2\beta(T-t)\Gamma^{-1}\bar{\boldsymbol{\mu}}_t^x + \frac{d\bar{\boldsymbol{\mu}}_t^x}{dt} = -4\beta(T-t)\Gamma^{-2}\bar{\boldsymbol{\mu}}_t^v, \tag{2.160}$$

$$-2\beta(T-t)\Gamma^{-1}\bar{\boldsymbol{\mu}}_t^v + \frac{d\bar{\boldsymbol{\mu}}_t^v}{dt} = \beta(T-t)\bar{\boldsymbol{\mu}}_t^x - 4\beta(T-t)\Gamma^{-1}\bar{\boldsymbol{\mu}}_t^v, \tag{2.161}$$

where we used the fact that $\frac{dC_t}{dt} = -2\beta(T-t)\Gamma^{-1}C_t$.

Eq. (2.160): Plugging the claimed solution into Eq. (2.160), we obtain:

$$\begin{aligned} &-2\beta(T-t)\Gamma^{-1}\left[2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{x}}_{t'} - 4\mathcal{B}(t)\Gamma^{-2}\bar{\mathbf{v}}_{t'} + \bar{\mathbf{x}}_{t'}\right] + \left[2\beta(T-t)\Gamma^{-1}\bar{\mathbf{x}}_{t'} - 4\beta(T-t)\Gamma^{-2}\bar{\mathbf{v}}_{t'}\right] \\ &= -4\beta(T-t)\Gamma^{-2}\left[\mathcal{B}(t)\bar{\mathbf{x}}_{t'} - 2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{v}}_{t'} + \bar{\mathbf{v}}_{t'}\right]. \end{aligned} \tag{2.162}$$

Eq. (2.131): After simplification, plugging the claimed solution into Eq. (2.131), we obtain:

$$\begin{aligned}&2\beta(T-t)\Gamma^{-1}\left[\mathcal{B}(t)\bar{\mathbf{x}}_{t'}-2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{v}}_{t'}+\bar{\mathbf{v}}_{t'}\right]+\left[\beta(T-t)\bar{\mathbf{x}}_{t'}-2\beta(T-t)\Gamma^{-1}\bar{\mathbf{v}}_{t'}\right]\\&=\beta(T-t)\left[2\mathcal{B}(t)\Gamma^{-1}\bar{\mathbf{x}}_{t'}-4\mathcal{B}(t)\Gamma^{-2}\bar{\mathbf{v}}_{t'}+\bar{\mathbf{x}}_{t'}\right].\end{aligned}\tag{2.163}$$

This completes the proof of the correctness of the mean.

2.4.7.2.2 Proof of Correctness of the Covariance Plugging the claimed solution (Eqs. (2.152)-(2.158)) into the ODE (Eq. (2.144)), we obtain

$$\left[\frac{d\tilde{\Sigma}_t}{dt}D_t+\frac{dD_t}{dt}\tilde{\Sigma}_t\right]\otimes\boldsymbol{I}_d=D_t(f\otimes\boldsymbol{I}_d)(\tilde{\Sigma}_t\otimes\boldsymbol{I}_d)+D_t\left[(f\otimes\boldsymbol{I}_d)(\tilde{\Sigma}_t\otimes\boldsymbol{I}_d)\right]^\top+\left(GG^\top\right)\otimes\boldsymbol{I}_d\tag{2.164}$$

with $f=f(T-t)$ and $G=G(T-t)$.

Noting that

$$\begin{aligned}(f(T-t)\otimes\boldsymbol{I}_d)(\tilde{\Sigma}_t\otimes\boldsymbol{I}_d)&=(f(T-t)\tilde{\Sigma}_t)\otimes\boldsymbol{I}_d\\&=\beta(T-t)\begin{pmatrix}-4\Gamma^{-2}\bar{\Sigma}_t^{xv} & -4\Gamma^{-2}\bar{\Sigma}_t^{vv}\\ \bar{\Sigma}_t^{xx}-4\Gamma^{-1}\bar{\Sigma}_t^{xv} & \bar{\Sigma}_t^{xv}-4\Gamma^{-1}\bar{\Sigma}^{vv}\end{pmatrix}\otimes\boldsymbol{I}_d,\end{aligned}\tag{2.165}$$

and

$$G(T-t)G^\top(T-t)=\begin{pmatrix}0 & 0\\ 0 & 2\Gamma\beta(T-t)\end{pmatrix},\tag{2.166}$$

as well as the fact $\frac{dD_t}{dt}=-4\beta(T-t)\Gamma^{-1}D_t$, we can decompose Eq. (2.164) into three equations:

$$-4\beta(T-t)\Gamma^{-1}\bar{\Sigma}_t^{xx}+\frac{d\bar{\Sigma}^{xx}}{dt}=-8\beta(T-t)\Gamma^{-2}\bar{\Sigma}_t^{xv},\tag{2.167}$$

$$-4\beta(T-t)\Gamma^{-1}\bar{\Sigma}_t^{xv}+\frac{d\bar{\Sigma}^{xv}}{dt}=\beta(T-t)\left[\bar{\Sigma}_t^{xx}-4\Gamma^{-1}\bar{\Sigma}_t^{xv}-4\Gamma^{-2}\bar{\Sigma}^{vv}\right],\tag{2.168}$$

$$-4\beta(T-t)\Gamma^{-1}\bar{\Sigma}_t^{vv}+\frac{d\bar{\Sigma}^{vv}}{dt}=\beta(T-t)\left[2\bar{\Sigma}_t^{xv}-8\Gamma^{-1}\bar{\Sigma}_t^{vv}\right]+2\Gamma\beta(T-t)D_t^{-1}.\tag{2.169}$$

Eq. (2.167): Plugging the claimed solution into Eq. (2.167), we obtain

$$\begin{aligned}&-4\beta(T-t)\Gamma^{-1}\left[e^{4\mathcal{B}(t)\Gamma^{-1}}-1-4\mathcal{B}(t)\Gamma^{-1}-8\mathcal{B}^2(t)\Gamma^{-2}\right]\\&\left[4\beta(T-t)\Gamma^{-1}e^{4\mathcal{B}(t)\Gamma^{-1}}-4\beta(T-t)\Gamma^{-1}-16\beta(T-t)\mathcal{B}(t)\Gamma^{-2}\right]\\&=-8\beta(T-t)\Gamma^{-2}\left[-4\mathcal{B}^2(t)\Gamma^{-1}\right].\end{aligned}\tag{2.170}$$

Eq. (2.168): After simplification, plugging the claimed solution into Eq. (2.168), we obtain

$$\begin{aligned}
&-8\beta(T-t)\mathcal{B}(t)\Gamma^{-1} \\
&= \beta(T-t)\left[e^{4\mathcal{B}(t)\Gamma^{-1}} - 1 - 4\mathcal{B}(t)\Gamma^{-1} - 8\mathcal{B}^2(t)\Gamma^{-2}\right] \\
&- 4\Gamma^{-2}\beta(T-t)\left[\tfrac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma - 2\mathcal{B}^2(t)\right].
\end{aligned} \tag{2.171}$$

Eq. (2.169): After simplification, plugging the claimed solution into Eq. (2.169), we obtain

$$\begin{aligned}
&4\beta(T-t)\Gamma^{-1}\left[\tfrac{\Gamma^2}{4}\left(e^{4\mathcal{B}(t)\Gamma^{-1}} - 1\right) + \mathcal{B}(t)\Gamma - 2\mathcal{B}^2(t)\right] \\
&\left[\Gamma\beta(T-t)e^{4\mathcal{B}(t)\Gamma^{-1}} + \beta(T-t)\Gamma - 4\beta(T-t)\mathcal{B}(t)\right] \\
&= 2\beta(T-t)\left[-4\mathcal{B}^2(t)\Gamma^{-1}\right] + 2\Gamma\beta(T-t)e^{4\mathcal{B}(t)\Gamma^{-1}}.
\end{aligned} \tag{2.172}$$

This completes the proof of the correctness of the covariance.

To connect back to the SSCS as presented in App. 2.4.4.2, recall that in practice we use constant β (and $T = 1$) and that we solve for small time steps of size $\frac{\delta t}{2}$, such that $\mathcal{B}(t) = \beta\frac{\delta t}{2}$, which leads to the expressions presented in App. 2.4.4.2.

2.5 Epilogue

2.5.1 Hybrid Score Matching Trains a Denoiser

In Section 1.2.3 we present a unified learning framework for DMs that use the perturbation kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t\mathbf{x}_0, \sigma_t^2\boldsymbol{I}_d)$. Now, recall the hybrid score matching loss from the paper (Equation (9)):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0\sim p_{\text{data}}(\mathbf{x}_0), t\sim\mathcal{U}[0,1], \boldsymbol{\varepsilon}_{2d}\sim\mathcal{N}(\boldsymbol{\varepsilon}_{2d};\mathbf{0}_{2d},\boldsymbol{I}_{2d})}[\lambda(t)\ell_t^2\|\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{u}_t, t) - \boldsymbol{\varepsilon}_{d:2d}\|_2^2], \tag{2.173}$$

where $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = \ell_t^{-1}\mathbf{v}_t/\Sigma_t^{vv} + F_{\boldsymbol{\theta}}(\mathbf{u}_t, t)$, with $F_{\boldsymbol{\theta}}$ being the learnable neural network, and

$$\mathbf{u}_t = \boldsymbol{\mu}_t(\mathbf{x}_0) + \boldsymbol{L}_t\boldsymbol{\varepsilon}_{2d}. \tag{2.174}$$

Plugging in the definitions of $\boldsymbol{\mu}_t$ and $\boldsymbol{L}_t$ (see Equation (2.31) and Equation (2.52), respectively), we obtain

$$\mathbf{x}_t = (2\mathcal{B}(t)\Gamma^{-1} + 1)e^{-2\mathcal{B}(t)\Gamma^{-1}}\mathbf{x}_0 + L_t^{xx}\boldsymbol{\varepsilon}_{0:d}, \tag{2.175}$$

$$\mathbf{v}_t = -\mathcal{B}(t)e^{-2\mathcal{B}(t)\Gamma^{-1}}\mathbf{x}_0 + L_t^{xv}\boldsymbol{\varepsilon}_{0:d} + L_t^{vv}\boldsymbol{\varepsilon}_{d:2d}. \tag{2.176}$$

Approximating $\boldsymbol{\varepsilon}_{d:2d}$ by our model $\boldsymbol{\alpha}_{\boldsymbol{\theta}}$ and solving for $\mathbf{x}_0$ gives

$$\mathbf{x}_0 \approx D_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = \frac{(\mathbf{v}_t - L_t^{vv}\left(\ell_t^{-1}\mathbf{v}_t/\Sigma_t^{vv} + F_{\boldsymbol{\theta}}(\mathbf{u}_t, t)\right) - \frac{L_t^{xv}}{L_t^{xx}}\mathbf{x}_t)}{g(t) - f(t)\frac{L_t^{xv}}{L_t^{xx}}}, \tag{2.177}$$

$$g(t) = -\mathcal{B}(t)e^{-2\mathcal{B}(t)\Gamma^{-1}}, \tag{2.178}$$

$$f(t) = (2\Gamma^{-1}\mathcal{B}(t) + 1)e^{-2\mathcal{B}(t)\Gamma^{-1}}. \tag{2.179}$$

Thus, hybrid score matching effectively learns a denoiser model $D_{\boldsymbol{\theta}}$ for the data-velocity pair $\mathbf{u}_t = (\mathbf{x}_t, \mathbf{v}_t)^\top$. We may extend the unified framework from Equation (1.26) to CLD:

$$D_{\boldsymbol{\theta}}(\mathbf{u}_t, t) = c_{\text{skip}}^x(t)\,\mathbf{x}_t + c_{\text{skip}}^v(t)\mathbf{v}_t + c_{\text{out}}(t)\,F_{\boldsymbol{\theta}}\left(c_{\text{in}}(t)\,\mathbf{u}_t, c_{\text{noise}}(t)\right). \tag{2.180}$$

While the original scaling factors in CLD were chosen subjectively, having this unified framework may allow us to use more principled strategies as has been done for DMs that use the perturbation kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \alpha_t\mathbf{x}_0, \sigma_t^2\boldsymbol{I}_d)$ [141].

2.5.2 Additional Related Work

Zhang et al. [315] developed a fast sampling scheme for CLD using the *exponential integrator* formalism. Singhal et al. [254] generalize CLD to *any* number of auxiliary variables. Furthermore, Singhal et al. [254] propose a method to learn the parameters of the diffusion process and automate the process of finding the perturbation kernel, which was done manually (and very time-consuming) in CLD. Doucet et al. [79] use CLD to improve *annealed importance sampling* [208]. Wizadwongsa and Suwajanakorn [291] propose a solver for DMs using *splitting methods* [35, 42, 158, 165, 276] similar to the symmetric splitting CLD sampler introduced in the paper. Not directly related to CLD, other novel diffusion processes for DMs have been explored [18, 63, 113, 132, 233].

Chapter 3

GENIE: Higher-Order Denoising Diffusion Solvers

3.1 An Introduction to Accelerated Sampling for Diffusion Models

When DMs were originally proposed [109, 257, 263], their main drawback, compared to other generative models, was slow sampling time. Since then, there has been a vast amount of work on accelerating the sampling process of DMs. Acceleration has primarily focused on deterministic sampling, i.e., sampling based on the probability flow ODE (Equation (1.21)).

3.1.1 Denoising Diffusion Implicit Models

Denoising diffusion implicit models (DDIMs) [258] were originally proposed as a non-Markovian generalization of discrete-time DMs (Section 1.2.4). Despite having the same objective function as discrete-time DMs, DDIMs are more flexible at inference time allowing, for example, for deterministic sampling. Recently, several works independently found that the deterministic sampling algorithm of DDIM can be recovered with continuous-time DMs by reparameterizing the probability flow ODE and simulating it using Euler's method with a particular time discretization [74, 244, 258, 312]. This finding allows for even further accelerated sampling by keeping the DDIM reparameterization but using more sophisticated ODE solvers than Euler's method. In the following, we derive the reparameterization

for the general perturbation kernel $\mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d)$ in the notation of Chapter 1. Let us recall the probability flow ODE (Equation (1.22))

$$\frac{d\mathbf{x}_t}{dt} = \frac{\dot{\alpha}_t}{\alpha_t}\mathbf{x}_t - \alpha_t^2 \dot{\gamma}_t \gamma_t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \tag{3.1}$$

where $\gamma_t = \sigma_t / \alpha_t$. Let us define

$$\widetilde{\mathbf{x}}_t = \frac{\mathbf{x}_t}{\alpha_t}, \tag{3.2}$$

then

$$\frac{d\widetilde{\mathbf{x}}_t}{dt} = \frac{\frac{d\mathbf{x}_t}{dt}}{\alpha_t} - \mathbf{x}_t \frac{\dot{\alpha}_t}{\alpha_t^2} = -\alpha_t \dot{\gamma}_t \gamma_t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = -\dot{\gamma}_t \sigma_t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{3.3}$$

Plugging the DM (Equation (1.25)) into the above equation results in

$$\frac{d\widetilde{\mathbf{x}}_t}{dt} = -\frac{\dot{\gamma}_t}{\gamma_t}(D(\mathbf{x}_t, t) - \widetilde{\mathbf{x}}_t), \tag{3.4}$$

and therefore

$$\frac{d\widetilde{\mathbf{x}}_t}{d\gamma_t} = \frac{\widetilde{\mathbf{x}}_t - D(\mathbf{x}_t, t)}{\gamma_t}. \tag{3.5}$$

Note that a single step in Euler's method from *any* $t_1 > 0$ to $t_0 = 0$ results in

$$\widetilde{\mathbf{x}}_0 = \widetilde{\mathbf{x}}_1 - \gamma_1 \frac{\widetilde{\mathbf{x}}_1 - D(\mathbf{x}_1, t_1)}{\gamma_1} = D(\mathbf{x}_1, t_1), \quad (\text{using } \gamma_0 = 0), \tag{3.6}$$

that is, the tangent of the ODE trajectory always points towards the predicted clean data point $\mathbf{x}_0$ which is expected to change slowly with time t and may explain why this particular reparameterization works so well [141]. Furthermore, the DDIM reparameterization can also be recovered by applying the *exponential integrator* formalism to the probability flow ODE [312]; another possible explanation for its success.

3.1.2 An Introduction to Fast ODE Solvers

There has been a long line of research on solving ODEs efficiently. Arguably, the two most prominent classes of methods are *linear multistep* methods and *Runge–Kutta* methods.

To better understand these methods, let us apply the p-th order *Taylor polynomial* to a generic ODE $\frac{d\mathbf{y}}{dt} = \boldsymbol{f}(\mathbf{y}, t)$ at the point $\mathbf{y}_{t_n}$, i.e.,

$$\mathbf{y}_{t_{n+1}} - \mathbf{y}_{t_n} = h_n \frac{d\mathbf{y}}{dt}|_{(\mathbf{y}_{t_n}, t_n)} + \frac{1}{2} h_n^2 \frac{d^2\mathbf{y}}{dt^2}|_{(\mathbf{y}_{t_n}, t_n)} \cdots + \frac{1}{p!} h_n^p \frac{d^p\mathbf{y}}{dt^p}|_{(\mathbf{y}_{t_n}, t_n)}, \tag{3.7}$$

where $h_n = t_{n+1} - t_n$ is the step size. Note that we can only directly evaluate the first term on the right-hand-side of Equation (3.7). Both linear multistep and Runge–Kutta methods (of p-th order) approximate the higher-order derivative terms in Equation (3.7) (up to p-th derivative) using *finite difference* schemes. Runge–Kutta methods approximate the higher order terms using evaluations of the first-order derivative $\boldsymbol{f}$ at additional points $\mathbf{y}_{t_n} < \mathbf{y}_{t_n'}, \mathbf{y}_{t_n''}, \cdots < \mathbf{y}_{t_{n+1}}$. Linear multistep methods, on the other hand, use additional evaluations of the first-order derivative at "past evaluations" $\mathbf{y}_{t_{n-1}}, \mathbf{y}_{t_{n-2}}, \ldots$, etc. Note that linear multistep methods of p-th order need to be *warm started*, that is, all points up to $\mathbf{y}_{t_p}$ need to be evaluated using an alternative method, e.g., a Runge–Kutta method or a linear multistep method of lower order. Both linear multistep methods and Runge–Kutta methods have been widely used to solve the probability flow ODE (or the DDIM reparameterization thereof) [176, 183, 184, 312]. Methods based on finite differences, however, generally break down in the *few-step limit* for large h_n since the approximations of the higher-order derivatives become increasingly crude.

3.2 Preface

This section presents the paper "GENIE: Higher-Order Denoising Diffusion Solvers". Rather than approximating higher-order derivatives using finite difference schemes, as is done in linear multistep methods and Runge–Kutta methods, in GENIE we propose to learn an efficient neural model for the second-order derivative, i.e., $\mathbf{k}_{\boldsymbol{\phi}} \approx \frac{d^2\mathbf{y}}{dt^2}$. Evaluating the model $\boldsymbol{k}_{\boldsymbol{\phi}}$ is cheap as it is implemented as a small prediction head on top of the neural network backbone of the original DM model. Our method can in principle also be extended to any higher-order derivative, and we show initial results for the third-order derivative. The paper was initially put on arXiv in October 2022, and then accepted and presented at the Conference on *Neural Information Processing Systems* in December 2022.

Contributions: I was the sole first author of this work. Arash Vahdat and Karsten Kreis were co-authors, and Karsten supervised the project. Karsten initially proposed to learn higher-order score functions using higher-order denoising score matching [195]. After little success in this direction, I pitched the idea to instead learn higher-order derivatives

via *distillation*. I implemented all code and ran all experiments, derived the loss functions, and wrote the majority of the paper. Karsten, Arash, and I jointly came up with the mixed network parameterization and the idea of learning the second-order derivative model $\mathbf{k}_\phi$ as a small prediction head which makes use of the internal representations of the neural network backbone of the DM.

Reproducibility: The code and models for this work have been open-sourced. See https://github.com/nv-tlabs/GENIE for instructions to reproduce our results.

arXiv: The paper "GENIE: Higher-Order Denoising Diffusion Solvers" is available on ArXiV (v1). The following version is simply reformatted into the style of the thesis.

Abstract: Denoising diffusion models (DDMs) have emerged as a powerful class of generative models. A forward diffusion process slowly perturbs the data, while a deep model learns to gradually denoise. Synthesis amounts to solving a differential equation (DE) defined by the learnt model. Solving the DE requires slow iterative solvers for high-quality generation. In this work, we propose *Higher-Order Denoising Diffusion Solvers* (GENIE): Based on truncated Taylor methods, we derive a novel higher-order solver that significantly accelerates synthesis. Our solver relies on higher-order gradients of the perturbed data distribution, that is, higher-order score functions. In practice, only Jacobian-vector products (JVPs) are required and we propose to extract them from the first-order score network via automatic differentiation. We then distill the JVPs into a separate neural network that allows us to efficiently compute the necessary higher-order terms for our novel sampler during synthesis. We only need to train a small additional head on top of the first-order score network. We validate GENIE on multiple image generation benchmarks and demonstrate that GENIE outperforms all previous solvers. Unlike recent methods that fundamentally alter the generation process in DDMs, our GENIE solves the true generative DE and still enables applications such as encoding and guided sampling. Project page and code: https://nv-tlabs.github.io/GENIE.

3.3 Main Paper

3.3.1 Introduction

Denoising diffusion models (DDMs) offer both state-of-the-art synthesis quality and sample diversity in combination with a robust and scalable learning objective. DDMs have been

used for image [66, 109, 110, 212, 235] and video [112, 301] synthesis, super-resolution [168, 240], deblurring [142, 290], image editing and inpainting [185, 197, 235, 239], text-to-image synthesis [13, 211, 230], conditional and semantic image generation [22, 56, 179, 218, 228], image-to-image translation [239, 249, 267] and for inverse problems in medical imaging [59, 60, 117, 187, 224, 264, 299]. They also enable high-quality speech synthesis [48, 49, 130, 157, 178, 227], 3D shape generation [37, 82, 188, 190, 319], molecular modeling [133, 251, 298, 300], maximum likelihood training [120, 151, 262, 280], and more [29, 213, 214, 246, 272, 305]. In DDMs, a diffusion process gradually perturbs the data towards random noise, while a deep neural network learns to denoise. Formally, the problem reduces to learning the *score function*, i.e., the gradient of the log-density of the perturbed data. The (approximate) inverse of the forward diffusion can be described by an ordinary or a stochastic differential equation (ODE or SDE, respectively), defined by the learned score function, and can therefore be used for generation when starting from random noise [262, 263].

A crucial drawback of DDMs is that the generative ODE or SDE is typically difficult to solve, due to the complex score function. Therefore, efficient and tailored samplers are required for fast synthesis. In this work, building on the generative ODE [258, 262, 263], we rigorously derive a novel second-order ODE solver using *truncated Taylor methods* [154]. These higher-order methods require higher-order gradients of the ODE—in our case this includes higher-order gradients of the log-density of the perturbed data, i.e., higher-order score functions. Because such higher-order scores are usually not available, existing works typically use simple first-order solvers or samplers with low accuracy [75, 109, 258, 263], higher-order methods that rely on suboptimal finite difference or other approximations [134, 176, 269], or alternative approaches [20, 156, 286] for accelerated sampling. Here, we fundamentally avoid such approximations and directly model the higher-order gradient terms: Importantly, our novel *Higher-Order Denoising Diffusion Solver (GENIE)* relies on Jacobian-vector products (JVPs) involving second-order scores. We propose to calculate these JVPs by automatic differentiation of the regular learnt first-order scores. For computational efficiency, we then distill the entire higher-order gradient of the ODE, including the JVPs, into a separate neural network. In practice, we only need to add a small head to the first-order score network to predict the components of the higher-order ODE gradient. By directly modeling the JVPs we avoid explicitly forming high-dimensional higher-order scores. Intuitively, the higher-order terms in GENIE capture the local curvature of the ODE and enable larger steps when iteratively solving the generative ODE (Fig. 3.1).

Experimentally, we validate GENIE on multiple image modeling benchmarks and achieve

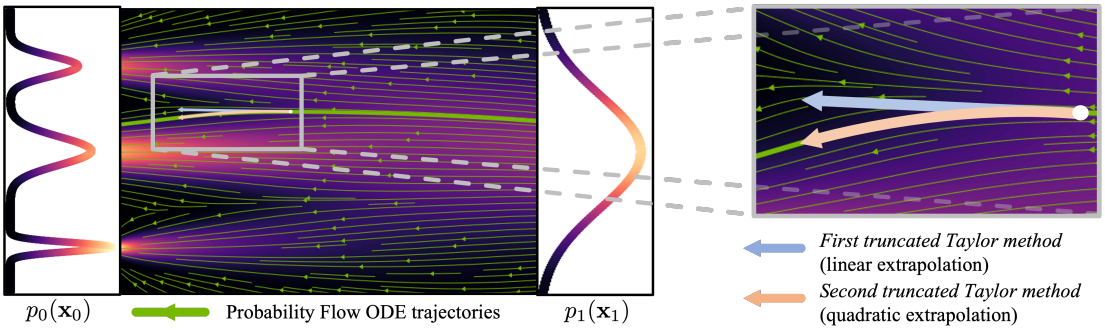


Figure 3.1: Our novel *Higher-Order Denoising Diffusion Solver* (GENIE) relies on the second *truncated Taylor method* (TTM) to simulate a (re-parametrized) Probability Flow ODE for sampling from denoising diffusion models. The second TTM captures the local curvature of the ODE's gradient field and enables more accurate extrapolation and larger step sizes than the first TTM (Euler's method), which previous methods such as DDIM [258] utilize.

state-of-the-art performance in solving the generative ODE of DDMs with few synthesis steps. In contrast to recent methods that fundamentally modify the generation process of DDMs by training conditional GANs [295] or by distilling the full sampling trajectory [186, 244], GENIE solves the true generative ODE. Therefore, we also show that we can still encode images in the DDM's latent space, as required for instance for image interpolation, and use techniques such as guided sampling [66, 108, 263].

We make the following contributions: **(i)** We introduce GENIE, a novel second-order ODE solver for fast DDM sampling. **(ii)** We propose to extract the required higher-order terms from the first-order score model by automatic differentiation. In contrast to existing works, we explicitly work with higher-order scores without finite difference approximations. To the best of our knowledge, GENIE is the first method that *explicitly* uses higher-order scores for generative modeling with DDMs. **(iii)** We propose to directly model the necessary JVPs and distill them into a small neural network. **(iv)** We outperform all previous solvers and samplers for the generative differential equations of DDMs.

3.3.2 Background

We consider continuous-time DDMs [109, 257, 263] whose forward process can be described by

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}), \tag{3.8}$$

where $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$ is drawn from the empirical data distribution and $\mathbf{x}_t$ refers to diffused data samples at time $t \in [0, 1]$ along the diffusion process. The functions α_t and σ_t

are generally chosen such that the *logarithmic signal-to-noise ratio* [151] $\log \frac{\alpha_t^2}{\sigma_t^2}$ decreases monotonically with t and the data diffuses towards random noise, i.e., $p_1(\mathbf{x}_1) \approx \mathcal{N}(\mathbf{x}_1; \mathbf{0}, \boldsymbol{I})$. We use *variance-preserving* [263] diffusion processes for which $\sigma_t^2 = 1 - \alpha_t^2$ (however, all methods introduced in this work are applicable to more general DDMs). The diffusion process can then be expressed by the (variance-preserving) SDE

$$d\mathbf{x}_t = -\tfrac{1}{2}\beta_t \mathbf{x}_t \, dt + \sqrt{\beta_t} \, d\mathbf{w}_t, \tag{3.9}$$

where $\beta_t = -\frac{d}{dt} \log \alpha_t^2$, $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$ and $\mathbf{w}_t$ is a standard Wiener process. A corresponding reverse diffusion process that effectively inverts the forward diffusion is given by [7, 104, 263]

$$d\mathbf{x}_t = -\tfrac{1}{2}\beta_t \left[\mathbf{x}_t + 2\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt + \sqrt{\beta_t} \, d\mathbf{w}_t, \tag{3.10}$$

and this reverse-time generative SDE is marginally equivalent to the generative ODE [262, 263]

$$d\mathbf{x}_t = -\tfrac{1}{2}\beta_t \left[\mathbf{x}_t + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt, \tag{3.11}$$

where $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ is the *score function*. Equation (3.11) is referred to as the *Probability Flow* ODE [263], an instance of continuous Normalizing flows [50, 100]. To generate samples from the DDM, one can sample $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{x}_1; \mathbf{0}, \boldsymbol{I})$ and numerically simulate either the Probability Flow ODE or the generative SDE, replacing the unknown score function by a learned score model $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$.

The DDIM solver [258] has been particularly popular to simulate DDMs due to its speed and simplicity. It has been shown that DDIM is *Euler's method* applied to an ODE based on a re-parameterization of the Probability Flow ODE [244, 258]: Defining $\gamma_t = \sqrt{\frac{1-\alpha_t^2}{\alpha_t^2}}$ and $\bar{\mathbf{x}}_t = \mathbf{x}_t \sqrt{1+\gamma_t^2}$, we have

$$\frac{d\bar{\mathbf{x}}_t}{d\gamma_t} = \sqrt{1+\gamma_t^2} \frac{d\mathbf{x}_t}{dt} \frac{dt}{d\gamma_t} + \mathbf{x}_t \frac{\gamma_t}{\sqrt{1+\gamma_t^2}} = -\frac{\gamma_t}{\sqrt{1+\gamma_t^2}} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \tag{3.12}$$

where we inserted Equation (3.11) for $\frac{d\mathbf{x}_t}{dt}$ and used $\beta(t)\frac{dt}{d\gamma_t} = \frac{2\gamma_t}{\gamma_t^2+1}$. Letting $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$ denote a parameterization of the score model, the approximate generative DDIM ODE is then given by

$$d\bar{\mathbf{x}}_t = \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}\left(\mathbf{x}_t, t\right) d\gamma_t, \tag{3.13}$$

where we used $\sigma_t = \sqrt{1-\alpha_t^2} = \frac{\gamma_t}{\sqrt{\gamma_t^2+1}}$ (see Section 3.4.1 for a more detailed derivation of Equation (3.13)). The model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ can be learned by minimizing the score matching objective [109, 282]

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[g(t)\|\boldsymbol{\varepsilon}-\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)\|_2^2\right], \quad \mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}, \tag{3.14}$$

for small $0 < t_{\text{cutoff}} \ll 1$. As is standard practice, we set $g(t) = 1$. Other weighting functions $g(t)$ are possible; for example, setting $g(t) = \frac{\beta_t}{2\sigma_t^2}$ recovers maximum likelihood learning [120, 151, 262, 280].

3.3.3 Higher-Order Denoising Diffusion Solver

As discussed in Section 3.3.2, the so-known DDIM solver [258] is simply Euler's method applied to the DDIM ODE (cf. Equation (3.13)). In this work, we apply a higher-order method to the DDIM ODE, building on the *truncated Taylor method* (TTM) [154]. The p-th TTM is simply the p-th order *Taylor polynomial* applied to an ODE. For example, for the general $\frac{d\mathbf{y}}{dt} = \boldsymbol{f}(\mathbf{y}, t)$, the p-th TTM reads as

$$\mathbf{y}_{t_{n+1}} = \mathbf{y}_{t_n} + h_n\frac{d\mathbf{y}}{dt}|_{(\mathbf{y}_{t_n},t_n)} + \cdots + \frac{1}{p!}h_n^p\frac{d^p\mathbf{y}}{dt^p}|_{(\mathbf{y}_{t_n},t_n)}, \tag{3.15}$$

where $h_n = t_{n+1} - t_n$ (see Section 3.4.2.1 for a truncation error analysis with respect to the exact ODE solution). Note that the first TTM is simply Euler's method. Applying the second TTM to the DDIM ODE results in the following scheme:

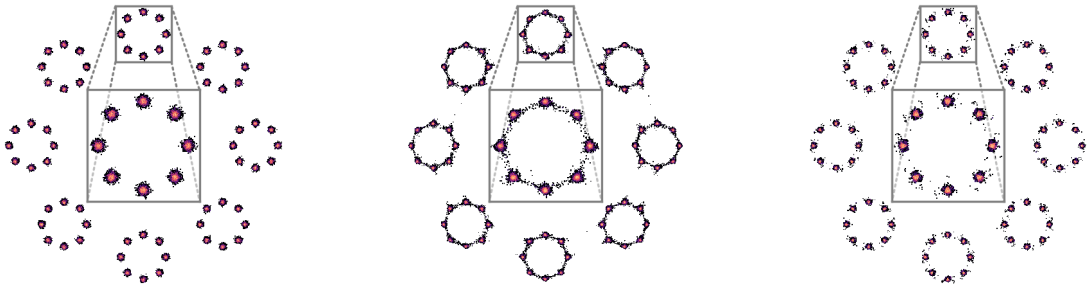
$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + h_n\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2}h_n^2\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t}|_{(\mathbf{x}_{t_n},t_n)}, \tag{3.16}$$

where $h_n = \gamma_{t_{n+1}} - \gamma_{t_n}$. Recall that $\gamma_t = \sqrt{\frac{1-\alpha_t^2}{\alpha_t^2}}$, where the function α_t is a time-dependent hyperparameter of the DDM. The total derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}} := \frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t}$ can be decomposed as follows

$$d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{\partial\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial t}\frac{dt}{d\gamma_t}, \tag{3.17}$$

where $\frac{\partial\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t}$ denotes the Jacobian of $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ and

$$\frac{d\mathbf{x}_t}{d\gamma_t} = \frac{\partial\mathbf{x}_t}{\partial\bar{\mathbf{x}}_t}\frac{d\bar{\mathbf{x}}_t}{d\gamma_t} + \frac{\partial\mathbf{x}_t}{\partial\gamma_t} = \frac{1}{\sqrt{\gamma_t^2+1}}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t) - \frac{\gamma_t}{1+\gamma_t^2}\mathbf{x}_t. \tag{3.18}$$



(a) Ground truth (b) DDIM (c) GENIE

Figure 3.3: Modeling a complex 2D toy distribution: Samples in *(b)* and *(c)* are generated via DDIM and GENIE, respectively, with 25 solver steps using the analytical score function of the ground truth distribution.

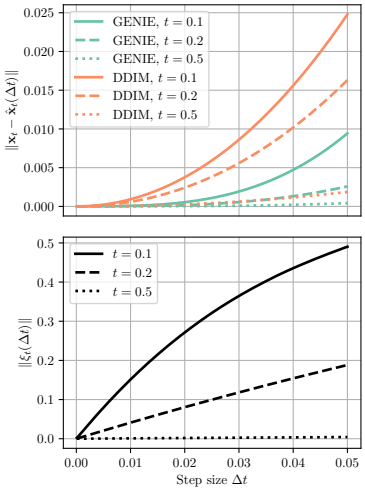


Figure 3.2: *Top*: Single step error using analytical score function. *Bottom*: Norm of difference $\xi_t(\Delta t)$ between analytical and approximate derivative computed via finite difference method.

If not explicitly stated otherwise, we refer to the second TTM applied to the DDIM ODE, i.e., the scheme in Equation (3.16), as *Higher-Order Denoising Diffusion Solver* (GENIE). Intuitively, the higher-order gradient terms used in the second TMM model the local curvature of the ODE. This translates into a Taylor formula-based extrapolation that is quadratic in time (cf. Equations (3.15) and (3.16)) and more accurate than linear extrapolation, as in Euler's method, thereby enabling larger time steps (see Figure 3.1 for a visualization). In Section 3.4.2, we also discuss the application of the third TTM to the DDIM ODE. We emphasize that TTMs are not restricted to the DDIM ODE and could just as well be applied to the Probability Flow ODE [263] (also see Section 3.4.2) or neural ODEs [50, 100] more generally.

The Benefit of Higher-Order Methods: We showcase the benefit of higher-order methods on a 2D toy distribution (Figure 3.3a) for which we know the score function as well as all higher-order derivatives necessary for GENIE analytically. We generate 1k different accurate "ground truth" trajectories $\mathbf{x}_t$ using DDIM with 10k steps. We compare these "ground truth" trajectories to *single* steps of DDIM and GENIE for varying step sizes Δt. We then measure the mean L_2-distance of the single steps $\hat{\mathbf{x}}_t(\Delta t)$ to the "ground truth" trajectories $\mathbf{x}_t$, and we repeat this experiment for three starting points $t \in \{0.1, 0.2, 0.5\}$. We see (Figure 3.2 (*top*)) that GENIE can use larger step sizes to stay within a certain error tolerance for all starting points t. We further show samples for DDIM and GENIE, using 25 solver

steps, in Figure 3.3. DDIM has the undesired behavior of sampling low-density regions between modes, whereas GENIE looks like a slightly noisy version of the ground truth distribution (Figure 3.3a).

Comparison to Multistep Methods: Linear multistep methods are an alternative higher-order method to solve ODEs. Liu et al. [176] applied the well-established Adams–Bashforth [AB, 36] method to the DDIM ODE. AB methods can be derived from TTMs by approximating higher-order derivatives $\frac{d^p \mathbf{y}}{dt^p}$ using the finite difference method [154]. For example, the second AB method is obtained from the second TTM by replacing $\frac{d^2 \mathbf{y}}{dt^2}$ with the first-order forward difference approximation $(f(\mathbf{y}_{t_n}, t_n) - f(\mathbf{y}_{t_{n-1}}, t_{n-1}))/h_{n-1}$. In Figure 3.2 (*bottom*), we visualize the mean L_2-norm of the difference $\xi_t(\Delta t)$ between the analytical derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ and its first-order forward difference approximation for varying step sizes Δt for the 2D toy distribution. The approximation is especially poor at small t for which the score function becomes complex (Section 3.4.5 for details on all toy experiments).

3.3.3.1 Learning Higher-Order Derivatives

The above observations inspire to apply GENIE to DDMs of more complex and high-dimensional data such as images. Regular DDMs learn a model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ for the first-order score; however, the higher-order gradient terms required for GENIE (cf. Equation (3.17)) are not immediately available to us, unlike in the toy example above. Let us insert Equation (3.18) into Equation (3.17) and analyze the required terms more closely:

$$d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{1}{\sqrt{\gamma_t^2 + 1}} \underbrace{\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{\text{JVP}_1} - \frac{\gamma_t}{1 + \gamma_t^2} \underbrace{\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \mathbf{x}_t}_{\text{JVP}_2} + \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} \frac{dt}{d\gamma_t}. \tag{3.19}$$

We see that the full derivative decomposes into two JVP terms and one simpler time derivative term. The term $\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}$ plays a crucial role in Equation (3.19). It can be expressed as

$$\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} = -\sigma_t \frac{\partial \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \approx -\sigma_t \nabla_{\mathbf{x}_t}^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \tag{3.20}$$

which means that GENIE relies on *second-order score functions* $\nabla_{\mathbf{x}_t}^\top \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ under the hood.

Given a DDM, that is, given $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$, we could compute the derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ for the GENIE scheme in Equation (3.16) using automatic differentiation (AD). This would, however, make a single step of GENIE at least twice as costly as DDIM, because we would need a

forward pass through the $\boldsymbol{\varepsilon_\theta}$ network to compute $\boldsymbol{\varepsilon_\theta}(\mathbf{x}_t, t)$ itself, and another pass to compute the JVPs and the time derivative in Equation (3.19). These forward passes cannot be parallelized, since the vector-part of JVP_1 in Equation (3.19) involves $\boldsymbol{\varepsilon_\theta}$ itself, and needs to be known before computing the JVP. To accelerate sampling, this overhead is too expensive.

Gradient Distillation: To avoid this overhead, we propose to first distill $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ into a separate neural network. During distillation training, we can use the slow AD-based calculation of $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$, but during synthesis we call the trained neural network. We build on the observation that the internal representations of the neural network modeling $\boldsymbol{\varepsilon_\theta}$ (in our case a U-Net [236] architecture) can be used for downstream tasks [21, 57]: specifically, we provide the last feature layer from the $\boldsymbol{\varepsilon_\theta}$ network together with its time embedding as well as $\mathbf{x}_t$ and the output $\boldsymbol{\varepsilon_\theta}(\mathbf{x}_t, t)$ to a small prediction head $\boldsymbol{k_\psi}(\mathbf{x}_t, t)$ that models the different terms in Equation (3.19) (see Figure 3.4). The overhead generated by $\boldsymbol{k_\psi}$ is small, for instance less than 2% for our CIFAR-10 model (also see Section 3.3.5), and we found this approach to provide excellent performance. Note that in principle we could also train an independent deep neural network, which does not make use of the internal representations of $\boldsymbol{\varepsilon_\theta}$ and could therefore theoretically be run in parallel to the $\boldsymbol{\varepsilon_\theta}$ model. We justify using small prediction heads over independent neural networks because AD-based distillation training is slow: in each training iteration we first need to call the $\boldsymbol{\varepsilon_\theta}$ network, then calculate the JVP terms, and only then can we call the distillation model. By modeling $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ via small prediction heads, while reusing the internal representation of the score model, we can make training relatively fast: we only need to train $\boldsymbol{k_\psi}$ for up to 50k iterations. In contrast, training score models from scratch takes roughly an order of magnitude more iterations. We leave training of independent networks to predict $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ to future work.

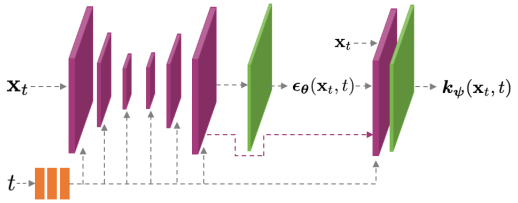


Figure 3.4: Our distilled model $\boldsymbol{k_\psi}$ that predicts the gradient $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ is implemented as a small additional output head on top of the first-order score model $\boldsymbol{\varepsilon_\theta}$. Purple layers are used both in $\boldsymbol{\varepsilon_\theta}$ and $\boldsymbol{k_\psi}$; green layers are specific for $\boldsymbol{\varepsilon_\theta}$ and $\boldsymbol{k_\psi}$.

Mixed Network Parameterization: We found that learning $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ directly as single output of a neural network can be challenging. Assuming a single data point distribution

$p_0(\mathbf{x}_0) = \delta(\mathbf{x}_0 = \mathbf{0})$, for which we know the diffused score function and all higher-order derivatives analytically, we found that the terms in Equation (3.19) all behave very differently within the $t \in [0, 1]$ interval (for instance, the prefactor of JVP_1 in Equation (3.19) approaches 1 as $t \to 0$, while JVP_2's prefactor vanishes). As outlined in detail in Paragraph 3.4.3.2.3, this simple single data point assumption implies an effective *mixed network parameterization*, an approach inspired by the "mixed score parametrizations" in Vahdat et al. [280] and Dockhorn et al. [75]. In particular, we model

$$\boldsymbol{k}_{\boldsymbol{\psi}} = -\frac{1}{\gamma_t}\boldsymbol{k}_{\boldsymbol{\psi}}^{(1)} + \frac{\gamma_t}{1+\gamma_t^2}\boldsymbol{k}_{\boldsymbol{\psi}}^{(2)} + \frac{1}{\gamma_t(1+\gamma_t^2)}\boldsymbol{k}_{\boldsymbol{\psi}}^{(3)} \approx d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}, \tag{3.21}$$

where $\boldsymbol{k}_{\boldsymbol{\psi}}^{(i)}(\mathbf{x}_t, t)$, $i \in \{1, 2, 3\}$, are different output channels of the neural network (i.e. the additional head on top of the $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ network). The three terms in Equation (3.21) exactly correspond to the three terms of Equation (3.19), in the same order. We show the superior performance of this parameterization in Section 3.3.5.3.

Learning Objective: Ideally, we would like our model $\boldsymbol{k}_{\boldsymbol{\psi}}$ to match $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ exactly, for all $t \in [0, T]$ and $\mathbf{x}_t$ in the diffused data distribution, which the generative ODE trajectories traverse. This suggests a simple (weighted) L_2-loss, similar to regular score matching losses for DDMs [109, 263]:

$$\min_{\boldsymbol{\psi}} \mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[g_{\text{d}}(t)\|\boldsymbol{k}_{\psi}(\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}, t) - d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}, t)\|_2^2\right] \tag{3.22}$$

for diffused data points $\alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}$ and $g_{\text{d}}(t) = \gamma_t^2$ to counteract the $1/\gamma_t$ in the first and third terms of Equation (3.21). This leads to a roughly constant loss over different time values t. During training we compute $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ via AD; however, at inference time we use the learned prediction head $\boldsymbol{k}_{\boldsymbol{\psi}}$ to approximate $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$. In Paragraph 3.4.3.2.4, we provide pseudo code for training and sampling with heads $\boldsymbol{k}_{\boldsymbol{\psi}}$. Note that our distillation objective is consistent and principled: if $\boldsymbol{k}_{\psi}$ matches $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ exactly, the resulting GENIE algorithm recovers the second TTM *exactly* (extended discussion in App. 3.4.2.4).

Alternative Learning Approaches: As shown in Equation (3.20), GENIE relies on second-order score functions. Recently, Meng et al. [195] directly learnt such higher-order scores with higher-order score matching objectives. Directly applying these techniques has the downside that we would need to explicitly form the higher-order score terms $\nabla_{\mathbf{x}_t}^{\top}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, which are very high-dimensional for data such as images. Low-rank approximations are possible, but potentially insufficient for high performance. In our approach,

we are avoiding this complication by directly modeling the lower-dimensional JVPs. We found that the methods from Meng et al. [195] can be modified to provide higher-order score matching objectives for the JVP terms required for GENIE and we briefly explored this (see Section 3.4.4). However, our distillation approach with AD-based higher-order gradients worked much better. Nevertheless, this is an interesting direction for future research. To the best of our knowledge, GENIE is the first solver for the generative differential equations of DDMs that *directly* uses higher-order scores (in the form of the distilled JVPs) for generative modeling without finite difference or other approximations.

3.3.4 Related Work

Accelerated Sampling from DDMs. Several previous works address the slow sampling of DDMs: One line of work reduces and readjusts the timesteps [156, 212] used in time-discretized DDMs [109, 257]. This can be done systematically by grid search [48] or dynamic programming [285]. Bao et al. [20] speed up sampling by defining a new DDM with optimal reverse variances. DDIM [258], discussed in Sec. 3.3.2, was also introduced as a method to accelerate DDM synthesis. Further works leverage modern ODE and SDE solvers for fast synthesis from (continuous-time) DDMs: For instance, higher-order Runge–Kutta methods [77, 263] and adaptive step size SDE solvers [134] have been used. These methods are not optimally suited for the few-step synthesis regime, in which GENIE shines; see also Section 3.3.5. Most closely related to our work is Liu et al. [176], which simulates the DDIM ODE [258] using a higher-order linear multistep method [36]. As shown in Section 3.3.3, linear multistep methods can be considered an approximation of the TTMs used in GENIE. Furthermore, Tachibana et al. [269] solve the generative SDE via a higher-order Itô–Taylor method [154] and in contrast to our work, they propose to use an "ideal derivative trick" to approximate higher-order score functions. In Section 3.4.2.2, we show that applying this ideal derivative approximation to the DDIM ODE does not have any effect: the "ideal derivatives" are zero by construction. Note that in GENIE, we in fact use the DDIM ODE, rather than, for example, the regular Probability Flow ODE [263], as the base ODE for GENIE.

Alternatively, sampling from DDMs can also be accelerated via learning: For instance, Watson et al. [286] learn parameters of a generalized family of DDMs by optimizing for perceptual output quality. Luhman and Luhman [186] and Salimans and Ho [244] distill a DDIM sampler into a student model, which enables sampling in as few as a single step. Xiao et al. [295] replace DDMs' Gaussian samplers with expressive generative ad-

versarial networks, similarly allowing for few-step synthesis. GENIE can also be considered a learning-based approach, as we distill a derivative of the generative ODE into a separate neural network. However, in contrast to the mentioned methods, GENIE still solves the true underlying generative ODE, which has major advantages: for instance, it can still be used easily for classifier-guided sampling [66, 108, 263] and to efficiently encode data into latent space—a prerequisite for likelihood calculation [262, 263] and editing applications [230]. Note that the learnt sampler [286] defines a proper probabilistic generalized DDM; however, it isn't clear how it relates to the generative SDE or ODE and therefore how compatible the method is with applications such as classifier guidance.

Other approaches to accelerate DDM sampling change the diffusion itself [75, 163, 207] or train DDMs in the latent space of a Variational Autoencoder [280]. GENIE is complementary to these methods.

Higher-Order ODE Gradients beyond DDMs. TTMs [154] and other methods that leverage higher-order gradients are also applied outside the scope of DDMs. For instance, higher-order derivatives can play a crucial role when developing solvers [70] and regularization techniques [87, 144] for neural ODEs [50, 100]. Outside the field of machine learning, higher-order TTMs have been widely studied, for example, to develop solvers for stiff [44] and non-stiff [44, 62] systems.

Concurrent Works. Zhang and Chen [314] motivate the DDIM ODE from an exponential integrator perspective applied to the Probability Flow ODE and propose to apply existing solvers from the numerical ODE literature, namely, Runge–Kutta and linear multistepping, to the DDIM ODE directly. Lu et al. [183] similarly recognize the semi-linear structure of the Probability Flow ODE, derive dedicated solvers, and introduce new step size schedulers to accelerate DDM sampling. Karras et al. [141] propose new fast solvers, both deterministic and stochastic, specifically designed for the differential equations arising in DDMs. Both Zhang et al. [315] and Karras et al. [141] realize that the DDIM ODE has "straight line solution trajectories" for spherical normal data and single data points—this exactly corresponds to our derivation that the higher-order terms in the DDIM ODE are zero in such a setting (see Section 3.4.2.2). Bao et al. [19] learn covariance matrices for DDM sampling using prediction heads somewhat similar to the ones in GENIE; in Section 3.4.7.1, we thoroughly discuss the differences between GENIE and the method proposed in Bao et al. [19].

3.3.5 Experiments

Datasets: We run experiments on five datasets: CIFAR-10 [160] (resolution 32), LSUN Bedrooms [308] (128), LSUN Church-Outdoor [308] (128), (conditional) ImageNet [65] (64), and AFHQv2 [58] (512). On AFHQv2 we only consider the subset of cats; referred to as "Cats" in the remainder of this work.

Architectures: Except for CIFAR-10 (we use a checkpoint by Song et al. [263]), we train our own score models using architectures introduced by previous works [66, 109]. The architecture of our prediction heads is based on (modified) BigGAN residual blocks [30, 263]. To minimize computational overhead, we only use a single residual block. See Section 3.4.3 for training and architecture details.

Evaluation: We measure sample quality via Fréchet Inception Distance [FID, 105] (see Section 3.4.6.1).

Synthesis Strategy: We simulate the DDIM ODE from t=1 up to t=10^{-3} using evaluation times following a quadratic function (*quadratic striding* [258]). For variance-preserving DDMs, it can be beneficial to denoise the ODE solver output at the cutoff t=10^{-3}, i.e., $\mathbf{x}_0 = \frac{\mathbf{x}_t - \sigma_t \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\alpha_t}$ [135, 263]. Note that the denoising step involves a score model evaluation, and therefore "loses" a function evaluation that could otherwise be used as an additional step in the ODE solver. To this end, denoising the output of the ODE solver is left as a hyperparameter of our synthesis strategy.

Analytical First Step (AFS): Every additional neural network call becomes crucial in the low number of function evaluations (NFEs) regime. We found that we can improve the performance of GENIE and all other methods evaluated on our checkpoints by replacing the learned score with the (analytical) score of $\mathcal{N}(\mathbf{0}, \boldsymbol{I}) \approx p_{t=1}(\mathbf{x}_t)$ in the first step of the ODE solver. The "gained" function evaluation can then be used as an additional step in the ODE solver. Similarly to the denoising step mentioned above, AFS is treated as a hyperparameter of our **Synthesis Strategy**. AFS details in Section 3.4.6.2.

Accounting for Computational Overhead: GENIE has a slightly increased computational overhead compared to other solvers due to the prediction head $\boldsymbol{k}_{\psi}$. The computational overhead is increased by 1.47%, 2.83%, 14.0%, and 14.4% on CIFAR-10, ImageNet, LSUN Bedrooms, and LSUN Church-Outdoor, respectively (see also Paragraph 3.4.3.2.5). This additional overhead is always accounted for implicitly: we divide the NFEs by the computational overhead and round to the nearest integer. For example, on LSUN Bed-

rooms, we compare baselines with 10/15 NFEs to GENIE with 9/13 NFEs.

3.3.5.1 Image Generation

In Figure 3.5 we compare our method to the most competitive baselines. In particular, on the same score model checkpoints, we compare GENIE with DDIM [258], S-PNDM [176], and F-PNDM [176]. For these four methods, we only include the best result over the two hyperparameters discussed above, namely, the denoising step and AFS (see Section 3.4.6.6 for genie/tables with all results). We also include three competitive results from the literature [20, 156, 286] that use different checkpoints and sampling strategies: for each method, we include the best result for their respective set of hyperparameters. We do not compare in this figure with Knowledge Distillation [KD, 186], Progressive Distillation [PG, 244] and Denoising Diffusion GANs [DDGAN, 295] as they do not solve the generative ODE/SDE and use fundamentally different sampling approaches with drawbacks discussed in Section 3.3.4.

For NFEs $\in \{10, 15, 20, 25\}$, GENIE outperforms all baselines (on the same checkpoint) on all four datasets (see detailed results in Section 3.4.6.6 and GENIE image samples in Section 3.4.6.7). On CIFAR-10 and (conditional) ImageNet, GENIE also outperforms these baselines for NFEs=5, whereas DDIM outperforms GENIE slightly on the LSUN datasets (see genie/tables in Section 3.4.6.6). GENIE also performs better than the three additional baselines from the literature (which use different checkpoints and sampling strategies) with the exception of the Learned Sampler [LS, 286] on LSUN Bedrooms for NFEs=20. Though LS uses a learned striding schedule on LSUN Bedrooms (whereas GENIE simply uses quadratic striding), the LS's advantage is most likely due to the different checkpoint. In Table 3.1, we investigate the effect of optimizing the striding schedule, via learning (LS) or grid search (DDIM & GENIE), on CIFAR-10 and find that its significance decreases rapidly with increased NFEs (also see Section 3.4.6.6 for details). In Table 3.1, we also show additional baseline results; however, we do not include commonly-used adaptive step size solvers in Figure 3.5, as they are arguably not well-suited for this low NFE regime: for example, on the same CIFAR-10 checkpoint we use for GENIE, the adaptive SDE solver introduced in Jolicoeur-Martineau et al. [134] obtains an FID of 82.4 at 48 NFEs. Also on the same checkpoint, the adaptive Runge–Kutta 4(5) [77] method applied to the ProbabilityFlow ODE achieves an FID of 13.1 at 38 NFEs (solver tolerances set to 10^{-2}).

The results in Figure 3.5 suggest that higher-order gradient information, as used in GENIE, can be efficiently leveraged for image synthesis. Despite using small prediction heads our distillation seems to be sufficiently accurate: for reference, replacing the distillation

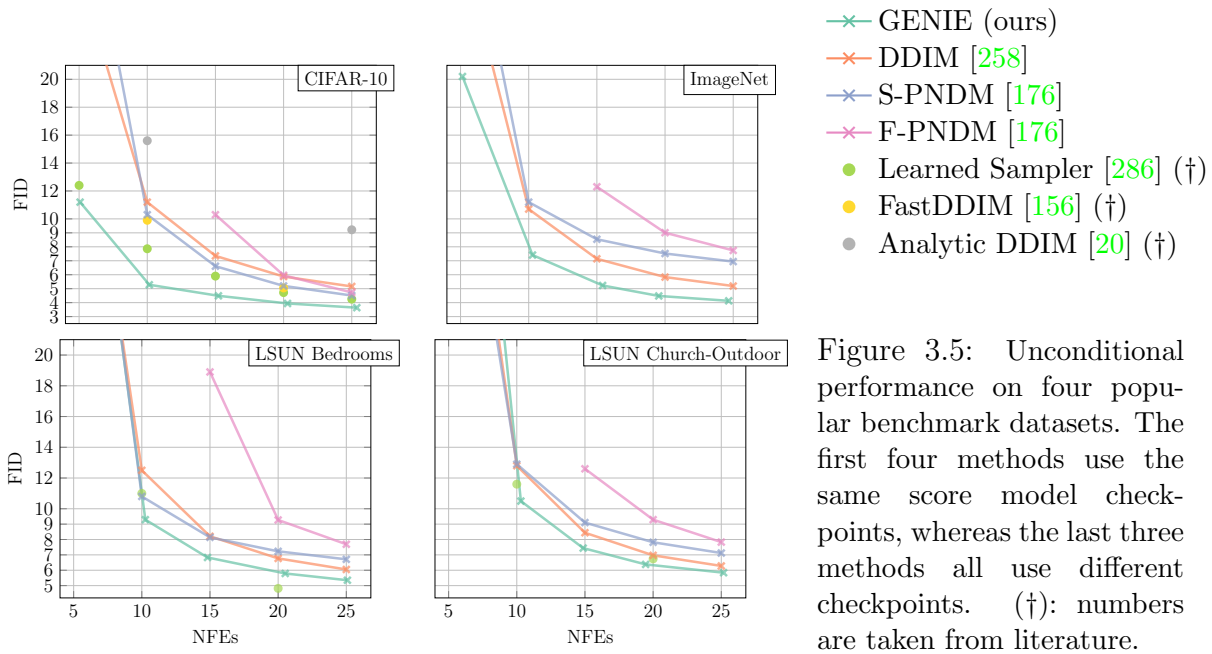


Figure 3.5: Unconditional performance on four popular benchmark datasets. The first four methods use the same score model checkpoints, whereas the last three methods all use different checkpoints. (†): numbers are taken from literature.

heads with the derivatives computed via AD, we obtain FIDs of 9.22, 4.11, 3.54, 3.46 using 10, 20, 30, and 40 NFEs, respectively (NFEs adjusted assuming an additional computational overhead of 100%). As discussed in Section 3.3.3, linear multistep methods such as S-PNDM [176] and F-PNDM [176] can be considered (finite difference) approximations to TTMs as used in GENIE. These approximations can be inaccurate for large timesteps, which potentially explains their inferior performance when compared to GENIE. When compared to DDIM, the superior performance of GENIE seems to become less significant for large NFE: this is in line with the theory, as higher-order gradients contribute less for smaller step sizes (see the GENIE scheme in Equation (3.16)). Approaches such as FastDDIM [156] and AnalyticDDIM [20], which adapt variances and discretizations of discrete-time DDMs, are useful; however, GENIE suggests that rigorous higher-order ODE solvers leveraging the continuous-time DDM formalism are still more powerful. To the best of our knowledge, the only methods that outperform GENIE abandon this ODE or SDE formulation entirely and train NFE-specific models [244, 295] which are optimized for the single use-case of image synthesis.

3.3.5.2 Guidance and Encoding

As discussed in Section 3.3.4, one major drawback of approaches such as KD [186], PG [244] and DDGAN [295] is that they abandon the ODE/SDE formalism, and cannot easily use methods such as classifier(-free) guidance [108, 263] or perform image encoding. However,

Method	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours) (∗)	**11.2**	**5.28**	**4.49**	**3.94**	**3.64**
GENIE (ours)	13.9	5.97	**4.49**	**3.94**	3.67
DDIM [258] (∗)	27.6	11.2	7.35	5.87	5.16
DDIM [258]	29.7	11.2	7.35	5.87	5.16
S-PNDM [176]	35.9	10.3	6.61	5.20	4.51
F-PNDM [176]	N/A	N/A	10.3	5.96	4.73
Euler–Maruyama	325	230	164	112	80.3
FastDDIM [156] (†)	-	9.90	-	5.05	-
Learned Sampler [286] (†/ ∗)	12.4	7.86	5.90	4.72	4.25
Learned Sampler [286] (†)	14.3	8.15	5.94	4.89	4.47
Analytic DDIM [20] (†)	-	14.0	-	-	5.71
CLD-SGM [75]	334	306	236	162	106
VESDE-PC [263]	461	461	461	461	462

Table 3.1: Unconditional CIFAR-10 generative performance (measured in FID). Methods above the middle line use the same score model checkpoint; methods below all use different ones. (†): numbers are taken from literature. (∗): methods either learn an optimal striding schedule (Learned Sampler) or do a small grid search over striding schedules (DDIM & GENIE); also see Section 3.4.6.6

these techniques can play an important role in synthesizing photorealistic images from DDMs [66, 211, 212, 230], as well as for image editing tasks [197, 230].

Classifier-Free Guidance [108]: We replace the unconditional diffusion model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ with $\hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c, w) = (1+w)\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c) - w\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ in the DDIM ODE (cf Equation (3.13)), where $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c)$ is a conditional model and $w > 1.0$ is the "guidance scale". GENIE then requires the derivative

$$d_{\gamma_t}\hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c, w) = (1+w)d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c) - wd_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{3.23}$$

for guidance. Hence, we need to distill $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, c)$ and $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, for which we could also share parameters [108]. We compare GENIE with DDIM on ImageNet in Figure 3.6. GENIE clearly outperforms DDIM, in particular for few NFEs, and GENIE also synthesizes high-quality images (see Figure 3.7).

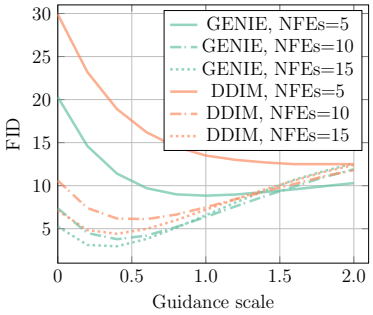


Figure 3.6: Sample quality as a function of guidance scale on ImageNet.

Image Encoding: We can use GENIE also to solve the generative ODE in reverse to encode given images. Therefore, we compare GENIE to DDIM on the "encode-decode" task, analyzing reconstructions for different NFEs (used twice for encoding and decoding): We find that GENIE reconstructs images much more accurately (see Figure 3.8). For more details on this experiment as well as the guidance experiment above, see Section 3.4.6.4 and Section 3.4.6.3, respectively. We also show latent space interpolations for both GENIE and DDIM in Section 3.4.6.5.

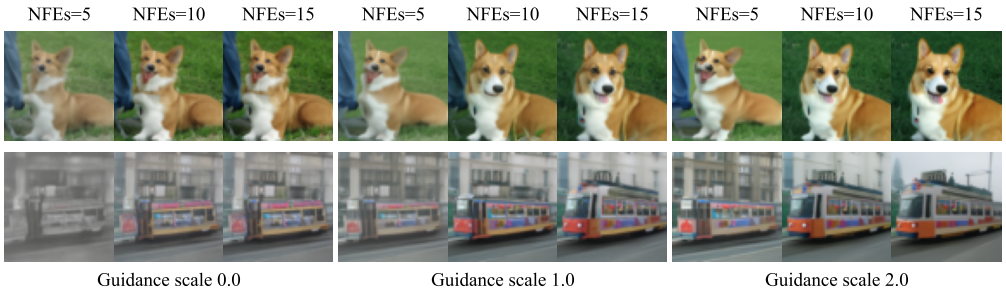


Figure 3.7: Classifier-free guidance for the ImageNet classes Pembroke Welsh Corgi (263) and Streetcar (829).

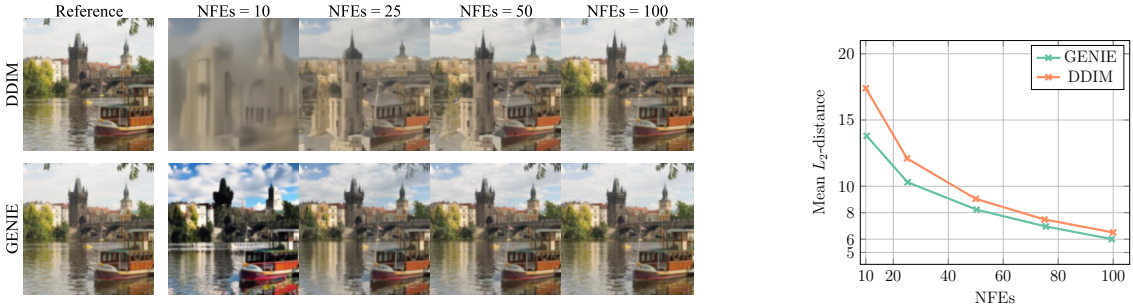


Figure 3.8: Encoding and subsequent decoding on LSUN Church-Outdoor. *Left:* Visual reconstruction. *Right:* L_2-distance to reference in Inception feature space [268], averaged over 100 images.

3.3.5.3 Ablation Studies

We perform ablation studies over architecture and training objective for the prediction heads used in GENIE: In Table 3.2, "No mixed" refers to learning $d_{\gamma_t}\boldsymbol{\varepsilon_\theta}$ directly as single network output without mixed network parameterization; "No weighting" refers to setting $g_\mathrm{d}(t) = 1$ in Equation (3.22); "Standard" uses both the mixed network parameterization and the weighting function $g_\mathrm{d}(t) = \gamma_t^2$. We can see that having both the mixed network parametrization and the weighting function is clearly beneficial. We also tested deeper networks in the prediction heads: for "Bigger model" we increased the number of residual blocks from one to two. The performance is roughly on par with "Standard", and we therefore opted for the smaller head due to the lower computational overhead.

Table 3.2: CIFAR-10 ablation studies (measured in FID).

Ablation	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
Standard	13.9	6.04	4.49	3.94	3.67
No mixed	14.7	6.32	4.82	4.31	4.10
No weighting	14.8	7.45	5.89	5.17	4.80
Bigger model	13.7	5.58	4.46	4.05	3.77

3.3.5.4 Upsampling

Table 3.3: Cats (upsampler) generative performance (measured in FID).

Method	NFEs=5	NFEs=10	NFEs=15
GENIE (ours)	**5.53**	**4.90**	**4.83**
DDIM [258]	9.47	6.64	5.85
S-PNDM [176]	14.6	11.0	8.83
F-PNDM [176]	N/A	N/A	11.7

Cascaded diffusion model pipelines [110] and DDM-based super-resolution [240] have become crucial ingredients in DDMs for large-scale image generation [242]. Hence, we also explore the applicability of GENIE in this setting. We train a 128×128 base model as well as a $128 \times 128 \rightarrow 512 \times 512$ diffusion upsampler [110, 240] on Cats. In Table 3.3, we compare the generative performance of GENIE to other fast samplers for the upsampler (in isolation). We find that GENIE performs very well on this task: with only five NFEs GENIE outperforms all other methods at NFEs=15. We show upsampled samples for GENIE with NFEs=5 in Figure 3.9. For more quantitative and qualitative results, we refer to Section 3.4.6.6 and Section 3.4.6.7, respectively. Training and inference details for the score model and the GENIE prediction head, for both base model and upsampler, can be found in Section 3.4.3.

3.3.6 Conclusions

We introduced GENIE, a higher-order ODE solver for DDMs. GENIE improves upon the commonly used DDIM solver by capturing the local curvature of its ODE's gradient field, which allows for larger step sizes when solving the ODE. We further propose to distill the required higher-order derivatives into a small prediction head—which we can efficiently call during inference—on top of the first-order score network. A limitation of GENIE is that it is still slightly slower than approaches that abandon the differential equation framework of DDMs altogether, which, however, comes at the considerable cost of preventing applications such as guided sampling. To overcome this limitation, future work could leverage even higher-order gradients to accelerate sampling from DDMs even further (also see Section 3.4.7.2).

Broader Impact. Fast synthesis from DDMs, the goal of GENIE, can potentially make DDMs an attractive method for promising interactive generative modeling applications, such as digital content creation or real-time audio synthesis, and also reduce DDMs' environmental footprint by decreasing the computational load during inference. Although we validate GENIE on image synthesis, it could also be utilized for other tasks, which makes its broader societal impact application-dependent. In that context, it is important that practitioners apply an abundance of caution to mitigate impacts given generative modeling can also be used for malicious purposes, discussed for instance in Mirsky and Lee [201], Nguyen et al. [210], Vaccari and Chadwick [278].

Figure 3.9: High-resolution images generated with the $128 \times 128 \rightarrow 512 \times 512$ GENIE upsampler using only five neural network calls. For the two images at the top, the upsampler is conditioned on test images from the Cats dataset. For the two images at the bottom, the upsampler is conditioned on samples from the 128×128 GENIE base model (generated using 25 NFEs); an upsampler neural network evaluation is roughly four times as expensive as a base model evaluation.

3.4 Appendix

3.4.1 DDIM ODE

The DDIM ODE has previously been shown [244, 258] to be a re-parameterization of the Probability Flow ODE [263]. In this subsection, we show an alternative presentation to the ones given in Song et al. [258] and Salimans and Ho [244]. We start from the Probability Flow ODE for variance-preserving continuous-time DDMs [263], i.e.,

$$d\mathbf{x}_t = -\tfrac{1}{2}\beta_t \left[\mathbf{x}_t + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt, \tag{3.24}$$

where $\beta_t = -\frac{d}{dt}\log \alpha_t^2$ and $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ is the *score function*. Replacing the unknown score function with a learned score model $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, we obtain the approximate Probability Flow ODE

$$d\mathbf{x}_t = -\tfrac{1}{2}\beta_t \left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right] dt. \tag{3.25}$$

Let us now define $\gamma_t = \sqrt{\frac{1-\alpha_t^2}{\alpha_t^2}}$ and $\bar{\mathbf{x}}_t = \mathbf{x}_t\sqrt{1+\gamma_t^2}$, and take the (total) derivative of $\bar{\mathbf{x}}_t$ with respect to γ_t:

$$\frac{d\bar{\mathbf{x}}_t}{d\gamma_t} = \frac{\partial \bar{\mathbf{x}}_t}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial \bar{\mathbf{x}}_t}{\partial \gamma_t} \tag{3.26}$$

$$= \sqrt{1+\gamma_t^2}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\gamma_t}{\sqrt{1+\gamma_t^2}}\mathbf{x}_t. \tag{3.27}$$

The derivative $\frac{d\mathbf{x}_t}{d\gamma_t}$ can be computed as follows

$$\frac{d\mathbf{x}_t}{d\gamma_t} = \frac{d\mathbf{x}_t}{dt}\frac{dt}{d\gamma_t} \quad \text{(by chain rule)} \tag{3.28}$$

$$= -\frac{1}{2}\beta_t \left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]\frac{dt}{d\gamma_t} \quad \text{(inserting Equation (3.25))} \tag{3.29}$$

$$= \frac{1}{2}\frac{d\log\alpha_t^2}{dt}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]\frac{dt}{d\gamma_t} \quad \text{(by definition of } \beta_t) \tag{3.30}$$

$$= \frac{1}{2}\frac{d\log\alpha_t^2}{d\gamma_t}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right] \quad \text{(by chain rule)} \tag{3.31}$$

$$= \frac{1}{2}\frac{d\log\alpha_t^2}{d\alpha_t^2}\frac{d\alpha_t^2}{d\gamma_t}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right] \quad \text{(by chain rule)} \tag{3.32}$$

$$= \frac{1}{2}\frac{1}{\alpha_t^2}\frac{d\alpha_t^2}{d\gamma_t}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]. \tag{3.33}$$

We can write α_t^2 as a function of γ_t, i.e., $\alpha_t^2 = (\gamma_t^2 + 1)^{-1}$, and therefore

$$\frac{d\alpha_t^2}{d\gamma_t} = -\frac{2\gamma_t}{(\gamma_t^2 + 1)^2}. \tag{3.34}$$

Inserting Equation (3.34) into Equation (3.33), we obtain

$$\frac{d\mathbf{x}_t}{d\gamma_t} = -\frac{\gamma_t}{\gamma_t^2 + 1}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]. \tag{3.35}$$

Lastly, inserting Equation (3.35) into Equation (3.27), we have

$$\frac{d\bar{\mathbf{x}}_t}{d\gamma_t} = -\frac{\gamma_t}{\sqrt{\gamma_t^2 + 1}}\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \tag{3.36}$$

Letting $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$, where $\sigma_t = \sqrt{1 - \alpha_t^2} = \frac{\gamma_t}{\sqrt{\gamma_t^2+1}}$, denote a particular parameterization of the score model, we obtain the approximate generative DDIM ODE as

$$\frac{d\bar{\mathbf{x}}_t}{d\gamma_t} = \frac{\gamma_t}{\sqrt{\gamma_t^2 + 1}}\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t} \tag{3.37}$$

$$= \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{3.38}$$

3.4.2 Synthesis from Denoising Diffusion Models via Truncated Taylor Methods

In this work, we propose Higher-Order Denoising Diffusion Solvers (GENIE). GENIE is based on the *truncated Taylor method* (TTM) [154]. As outlined in Section 3.3.3, the p-th TTM is simply the *p-th order Taylor polynomial* applied to an ODE. For example, for the general $\frac{d\mathbf{y}}{dt} = \boldsymbol{f}(\mathbf{y}, t)$, the p-th TTM reads as

$$\mathbf{y}_{t_{n+1}} = \mathbf{y}_{t_n} + h_n \frac{d\mathbf{y}}{dt}|_{(\mathbf{y}_{t_n}, t_n)} + \cdots + \frac{1}{p!}h_n^p \frac{d^p\mathbf{y}}{dt^p}|_{(\mathbf{y}_{t_n}, t_n)}, \tag{3.39}$$

where $h_n = t_{n+1} - t_n$. To generate samples from denoising diffusion models, we can, for example, apply the second TTM to the (approximate) Probability Flow ODE or the (approximate) DDIM ODE, resulting in the following respective schemes:

$$\mathbf{x}_{t_{n+1}} = \mathbf{x}_{t_n} + (t_{n+1} - t_n)\boldsymbol{f}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2}(t_{n+1} - t_n)^2 \frac{d\boldsymbol{f}}{dt}|_{(\mathbf{x}_{t_n}, t_n)}, \tag{3.40}$$

where $\boldsymbol{f}(\mathbf{x}_t, t) = -\frac{1}{2}\beta(t)\left[\mathbf{x}_t - \frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\sigma_t}\right]$, and

$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + (\gamma_{t_{n+1}} - \gamma_{t_n})\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2}(\gamma_{t_{n+1}} - \gamma_{t_n})^2 \frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t}|_{(\mathbf{x}_{t_n}, t_n)}. \tag{3.41}$$

In this work, we generate samples from DDMs using the scheme in Equation (3.41). We distill the derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}} \coloneqq \frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t}$ into a small neural network $\boldsymbol{k}_{\boldsymbol{\psi}}$. For training, $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ is computed via automatic differentiation, however, during inference, we can efficiently query the trained network $\boldsymbol{k}_{\boldsymbol{\psi}}$.

3.4.2.1 Theoretical Bounds for the Truncated Taylor Method

Consider the p-TTM for a general ODE $\frac{d\mathbf{y}}{dt} = \boldsymbol{f}(\mathbf{y}, t)$:

$$\mathbf{y}_{t_{n+1}} = \mathbf{y}_{t_n} + h_n \frac{d\mathbf{y}}{dt}|_{(\mathbf{y}_{t_n}, t_n)} + \cdots + \frac{1}{p!} h_n^p \frac{d^p\mathbf{y}}{dt^p}|_{(\mathbf{y}_{t_n}, t_n)}. \tag{3.42}$$

We represent, the exact solution $\mathbf{y}(t_{n+1})$ using the $(p+2)$-th Taylor expansion

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h_n \frac{d\mathbf{y}}{dt}|_{(\mathbf{y}_{t_n}, t_n)} + \cdots + \frac{1}{p!} h_n^p \frac{d^p\mathbf{y}}{dt^p}|_{(\mathbf{y}_{t_n}, t_n)} + \frac{1}{(p+1)!} h_n^{p+1} \frac{d^{p+1}\mathbf{y}}{dt^{p+1}}|_{(\mathbf{y}_{t_n}, t_n)} + \mathcal{O}(h_n^{p+2}). \tag{3.43}$$

The local truncation error (LTE) introduced by the p-th TTM is given by the difference between the two equations above

$$\|\mathbf{y}_{t_{n+1}} - \mathbf{y}(t_{n+1})\| = \|\frac{1}{(p+1)!} h_n^{p+1} \frac{d^{p+1}\mathbf{y}}{dt^{p+1}}|_{(\mathbf{y}_{t_n}, t_n)} + \mathcal{O}(h_n^{p+2})\|. \tag{3.44}$$

For small h_n, the LTE is proportional to h_n^{p+1}. Consequently, using higher orders p implies lower errors, as h_n usually is a small time step.

In conclusion, this demonstrates that it is preferable to use higher-order methods with lower errors when aiming to accurately solve ODEs like the Probability Flow ODE or the DDIM ODE of diffusion models.

3.4.2.2 Approximate Higher-Order Derivatives via the "Ideal Derivative Trick"

Tachibana et al. [269] sample from DDMs using (an approximation to) a higher-order Itô-Taylor method [154]. In their scheme, they approximate higher-order score functions with

the "ideal derivative trick", essentially assuming simple single-point ($\mathbf{x}_0$) data distributions, for which higher-order score functions can be computed analytically (more formally, their approximation corresponds to ignoring the expectation over the full data distribution when learning the score function. They assume that for any $\mathbf{x}_t$, there is a single unique $\mathbf{x}_0$ from the input data to be predicted with the score model). In that case, further assuming the score model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is learnt perfectly (i.e., it perfectly predicts the noise that was used to generate $\mathbf{x}_t$ from $\mathbf{x}_0$), one has

$$\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t}. \tag{3.45}$$

This expression can now be used to analytically calculate approximate spatial and time derivatives (also see App. F.1 and App. F.2 in Tachibana et al. [269]):

$$\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \approx \frac{\partial}{\partial \mathbf{x}_t}\left(\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t}\right) = \frac{1}{\sigma_t}\boldsymbol{I}, \tag{3.46}$$

and

$$\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} \approx \frac{\partial}{\partial t}\left(\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t}\right) = -\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t^2}\frac{d\sigma_t}{dt} - \frac{\mathbf{x}_0}{\sigma_t}\frac{d\alpha_t}{dt}. \tag{3.47}$$

Rearranging Equation (3.45), we have

$$\mathbf{x}_0 \approx \frac{\mathbf{x}_t - \sigma_t \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\alpha_t}. \tag{3.48}$$

Inserting this expression, Equation (3.47) becomes

$$\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} \approx \frac{\frac{d \log \alpha_t^2}{dt}}{2\sigma_t}\left(\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t} - \mathbf{x}_t\right). \tag{3.49}$$

We will now proceed to show that the "ideal derivative trick", i.e. using the approximations in Equations (3.46) and (3.49), results in $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}} = \mathbf{0}$.

As in Section 3.3.3, the total derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ is composed as

$$d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{dt}{d\gamma_t}. \tag{3.50}$$

Inserting the "ideal derivative trick", the above becomes

$$d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \frac{1}{\sigma_t}\left(\frac{1}{2}\frac{1}{\alpha_t^2}\frac{d\alpha_t^2}{d\gamma_t}\left[\mathbf{x}_t - \frac{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}{\sigma_t}\right]\right) + \left(\frac{\frac{d \log \alpha_t^2}{dt}}{2\sigma_t}\left(\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t} - \mathbf{x}_t\right)\right)\frac{dt}{d\gamma_t}, \tag{3.51}$$

where we have inserted Equation (3.33) for $\frac{d\mathbf{x}_t}{d\gamma_t}$ and used the usual parameterization $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\sigma_t}$. Using $\frac{d \log \alpha_t^2}{dt} = \frac{1}{\alpha_t^2}\frac{d\alpha_t^2}{dt}$ and $\frac{d\alpha_t^2}{dt}\frac{dt}{d\gamma_t} = \frac{d\alpha_t^2}{d\gamma_t}$, we can see that the right-hand side of Equation (3.51) is $\mathbf{0}$. Hence, applying the second TTM to the DDIM ODE and using the "ideal derivative trick" is equivalent to the first TTM (Euler's method) applied to the DDIM ODE. We believe that this is potentially a reason why the DDIM solver [258], Euler's method applied to the DDIM ODE, shows such great empirical performance: it can be interpreted as an approximate ("ideal derivative trick") second order ODE solver. On the other hand, our derivation also implies that the "ideal derivative trick" used in the second TTM for the DDIM ODE does not actually provide any benefit over the standard DDIM solver, because all additional second-order terms vanish. Hence, to improve upon regular DDIM, the "ideal derivative trick" is insufficient and we need to learn the higher-order score terms more accurately without such coarse approximations, as we do in our work.

Furthermore, it is interesting to show that we do not obtain the same cancellation effect when applying the "ideal derivative trick" to the Probability Flow ODE in Equation (3.25): Let $\boldsymbol{f}(\mathbf{x}_t, t) = -\frac{1}{2}\beta(t)\left[\mathbf{x}_t - \frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\sigma_t}\right]$ (right-hand side of Probability Flow ODE), then

$$\frac{d\boldsymbol{f}}{dt}|_{(\mathbf{x}_t,t)} = \frac{\beta'(t)}{\beta(t)}\boldsymbol{f}(\mathbf{x}_t, t) - \frac{1}{2}\beta(t)\frac{d}{dt}\left[\mathbf{x}_t - \frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}\right] \tag{3.52}$$

$$= \left[\frac{\beta'(t)}{\beta(t)} - \frac{1}{2}\beta(t)\right]\boldsymbol{f}(\mathbf{x}_t, t) + \frac{1}{2}\beta(t)\left(\frac{\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{dt}}{\sigma_t} - \sigma_t^{-2}\frac{d\sigma_t}{dt}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right), \tag{3.53}$$

where $\beta'(t) \coloneqq \frac{d\beta(t)}{dt}$. Using the "ideal derivative trick", we have $\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{dt} = d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}\, d_t\gamma_t \approx \mathbf{0}$, and therefore the above becomes

$$\frac{d\boldsymbol{f}}{dt}|_{(\mathbf{x}_t,t)} \approx \left[\frac{\beta'(t)}{\beta(t)} - \frac{1}{2}\beta(t)\right]\boldsymbol{f}(\mathbf{x}_t, t) - \frac{\beta(t)}{2\sigma_t^2}\frac{d\sigma_t}{dt}\boldsymbol{\varepsilon}(\mathbf{x}_t, t). \tag{3.54}$$

The derivative $\frac{d\sigma_t}{dt}$ can be computed as follows

$$\frac{d\sigma_t}{dt} = \frac{1}{2\sigma_t}\frac{d\sigma_t^2}{dt} \tag{3.55}$$

$$= \frac{1}{2\sigma_t}\frac{d}{dt}\left(1 - e^{-\int_0^t \beta(t')\,dt'}\right) \tag{3.56}$$

$$= \frac{\beta(t)e^{-\int_0^t \beta(t')\,dt'}}{2\sigma_t}. \tag{3.57}$$

Putting everything back together, we have

$$\frac{d\boldsymbol{f}}{dt}|_{(\mathbf{x}_t,t)} = \left[\frac{\beta'(t)}{2\sigma_t} + \frac{\beta^2(t)}{4\sigma_t} - \frac{\beta^2(t)e^{-\int_0^t \beta(t')\,dt'}}{4\sigma_t^3}\right]\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t) + \left[-\frac{\beta'(t)}{2} + \frac{\beta^2(t)}{4}\right]\mathbf{x}_t, \quad (3.58)$$

which is clearly not $\mathbf{0}$ for all $\mathbf{x}_t$ and t. Hence, in contrast to the DDIM ODE, applying Euler's method to the Probability Flow ODE does not lead to an approximate (in the sense of the "ideal derivative trick") second order ODE solver.

Note that very related observations have been made in the concurrent works Karras et al. [141] and Zhang et al. [315]. These works notice that when the data distribution consist only of a single data point or a spherical Gaussian distribution, then the solution trajectories of the generative DDIM ODE are straight lines. In fact, this exactly corresponds to our observation that in such a setting we have $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}} = \mathbf{0}$, as shown above in the analysis of the "ideal derivatives approximation". Note in that context that our above derivation considers the "single data point" distribution assumption, but also applies to the setting where the data is a spherical normal distribution (only σ_t would be different, which would not affect the derivation).

3.4.2.3 3rd TTM Applied to the DDIM ODE

As promised in Section 3.3.3, we show here how to apply the third TTM to the DDIM ODE, resulting in the following scheme:

$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + h_n\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n},t_n) + \frac{1}{2}h_n^2\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t}|_{(\mathbf{x}_{t_n},t_n)} + \frac{1}{6}h_n^3\frac{d^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t^2}|_{(\mathbf{x}_{t_n},t_n)}, \quad (3.59)$$

where $h_n = (\gamma_{t_{n+1}} - \gamma_{t_n})$. In the remainder of this subsection, we derive a computable formula for $\frac{d^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t^2}$, only containing partial derivatives.

Using the chain rule, we have

$$\frac{d^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t^2}|_{(\mathbf{x}_t,t)} = \frac{\partial d_\gamma\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial d_\gamma\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial t}\frac{dt}{d\gamma_t}, \quad (3.60)$$

where, using Equation (3.50),

$$\frac{\partial d_\gamma\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t} = \frac{\partial^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}^2}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t}\left(\frac{1}{\sqrt{\gamma_t^2+1}}\frac{\partial\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial\mathbf{x}_t} - \frac{\gamma_t}{1+\gamma_t^2}\boldsymbol{I}\right) + \frac{\partial^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\partial t\partial\mathbf{x}_t}\frac{dt}{d\gamma_t}, \quad (3.61)$$

and

$$\frac{\partial d_\gamma \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} = \frac{\partial}{\partial t}\left(\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t}\right) + \frac{\partial}{\partial t}\left(\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{dt}{d\gamma_t}\right). \tag{3.62}$$

The remaining terms in Equation (3.62) can be computed as

$$\frac{\partial}{\partial t}\left(\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{dt}{d\gamma_t}\right) = \frac{\partial^2 \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t^2}\frac{dt}{d\gamma_t} + \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{d\left(\frac{dt}{d\gamma_t}\right)}{dt}, \tag{3.63}$$

and

$$\frac{\partial}{\partial t}\left(\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t}\right) = \frac{\partial^2 \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t\,\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{d\gamma_t} + \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{\partial\left(\frac{d\mathbf{x}_t}{d\gamma_t}\right)}{\partial t} \tag{3.64}$$

where, inserting Equation (3.35) for $\frac{d\mathbf{x}_t}{d\gamma_t}$ as well as using the usual parameterization $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$,

$$\frac{\partial\left(\frac{d\mathbf{x}_t}{d\gamma_t}\right)}{\partial t} = \frac{\partial}{\partial t}\left(-\frac{\gamma_t}{\gamma_t^2+1}\left[\mathbf{x}_t - \frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}\right]\right) \tag{3.65}$$

$$= \frac{\partial\left(\frac{1}{\sqrt{\gamma_t^2+1}}\right)}{\partial t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \frac{1}{\sqrt{\gamma_t^2+1}}\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} - \frac{\partial\left(\frac{\gamma_t}{1+\gamma_t^2}\right)}{\partial t}\mathbf{x}_t \quad \left(\sigma_t = \frac{\gamma_t}{\sqrt{\gamma_t^2+1}}\right) \tag{3.66}$$

$$= \left(-\frac{\gamma_t}{(\gamma_t^2+1)^{3/2}}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \frac{\gamma_t^2-1}{(\gamma_t^2+1)^2}\mathbf{x}_t\right)\frac{d\gamma_t}{dt} + \frac{1}{\sqrt{\gamma_t^2+1}}\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}. \tag{3.67}$$

We now have a formula for $\frac{d^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t^2}$ containing only partial derivatives, and therefore we can compute $\frac{d^2\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}}{d\gamma_t^2}$ using automatic differentiation. Note that we could follow the same procedure to compute even higher derivatives of $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$.

We repeat the 2D toy distribution single step error experiment from Section 3.3.3 (see also Figure 3.2 (*top*) and Section 3.4.5 for details). As expected, in Figure 3.10 we can clearly see that the third TTM improves upon the second TTM.

In Figure 3.11, we compare the second TTM to the third TTM applied to the DDIM

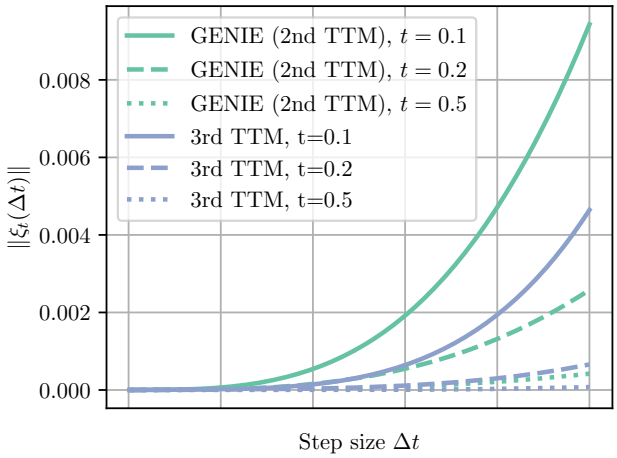


Figure 3.10: Single step error using analytical score function. See also Figure 3.2 (*top*).

ODE on CIFAR-10. Both for the second and the third TTM, we compute all partial derivatives using automatic differentiation (without distillation). It appears that for using 15 or less steps in the ODE solver, the second TTM performs better than the third TTM. We believe that this could potentially be due to our score model $s_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ not being accurate enough, in contrast to the above 2D toy distribution experiment, where we have access to the analytical score function. Furthermore, note that when we train $s_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ via score matching, we never regularize (higher-order) derivatives of the neural network, and therefore there is no incentive for them to be well-behaved. It would be interesting to see if, besides having more accurate score models, regularization techniques such as spectral regularization [203] could potentially alleviate this issue. Also the higher-order score matching techniques derived by Meng et al. [195] could help to learn higher-order derivates of the score functions more accurately. We leave this exploration to future work.

3.4.2.4 GENIE is Consistent and Principled

GENIE is a consistent and principled approach to developing a higher-order ODE solver for sampling from diffusion models: GENIE's design consists of two parts: (1) We are building on the second Truncated Taylor Method (TTM), which is a well-studied ODE solver (see Kloeden and Platen [154]) with provable local and global truncation errors (see also Section 3.4.2.1). Therefore, if during inference we had access to the ground truth second-order ODE derivatives, which are required for the second TTM, GENIE would simply correspond to the *exact* second TTM.

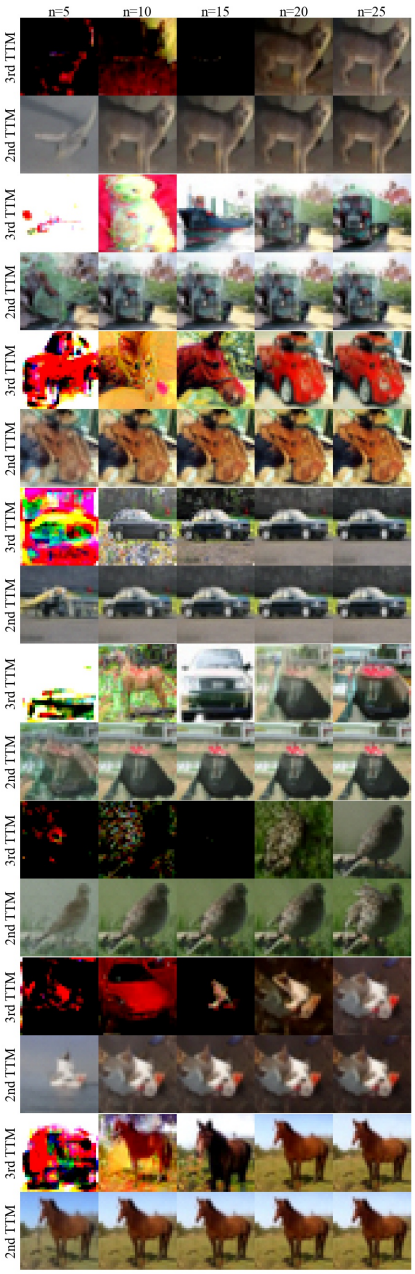


Figure 3.11: Qualitative comparison of the second and the third TTMs applied to the DDIM ODE on CIFAR-10 (all necessary derivatives calculated with automatic differentiation). The number of steps in the ODE solver is denoted as n.

(2) In principle, we could calculate the exact second-order derivatives *during inference* using automatic differentiation. However, this is too slow for competitive sampling speeds, as it requires additional backward passes through the first-order score network. Therefore, in practice, we use the learned prediction heads $\mathbf{k}_\psi(\mathbf{x}_t, t)$.

Consequently, if $\mathbf{k}_\psi(\mathbf{x}_t, t)$ modeled the ground truth second-order derivatives exactly, i.e. $\mathbf{k}_\psi(\mathbf{x}_t, t) = d_{\gamma_t}\epsilon_\theta(\mathbf{x}_t, t)$ for all $\mathbf{x}_t$ and t, we would obtain a rigorous second-order solver based on the TTM, following (1) above.

In practice, distillation will not be perfect. However, given the above analysis, optimizing a neural network $\mathbf{k}_\psi(\mathbf{x}_t, t)$ towards $d_{\gamma_t}\epsilon_\theta(\mathbf{x}_t, t)$ is well motivated and theoretically grounded. In particular, *during training* we are calculating *exact* ODE gradients using automatic differentiation on the first-order score model as distillation targets. Therefore, in the limit of infinite neural network capacity and perfect optimization, we could in theory minimize our distillation objective function (Equation (3.22)) perfectly and obtain $\mathbf{k}_\psi(\mathbf{x}_t, t) = d_{\gamma_t}\epsilon_\theta(\mathbf{x}_t, t)$.

Also recall that regular denoising score matching itself, on which all diffusion models rely, follows the exact same argument. In particular, denoising score matching also minimizes a "simple" (weighted) L_2-loss between a trainable score model $\mathbf{s}_\theta(\mathbf{x}_t, t)$ and the spatial derivative of the log-perturbation kernel, i.e., $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$. From this perspective, denoising score matching itself also simply tries to "distill" (spatial) derivatives into a model. If we perfectly optimized the denoising score matching objective, we would obtain a diffusion model that models the data distribution exactly, but in practice, similar to GENIE, we never achieve that due to imperfect optimization and finite-capacity neural networks. Nevertheless, denoising score matching similarly is a well-defined and principled method, precisely because of that theoretical limit in which the distribution can be reproduced exactly.

We would also like to point out that other, established higher-order methods for diffusion model sampling with the generative ODE, such as linear multistep methods [176], make approximations, too, which can be worse in fact. In particular, multistep methods *always* approximate higher-order derivatives in the TTM using finite differences which is crude for large step sizes, as can be seen in Fig. 3 (*bottom*). From this perspective, if our distillation is sufficiently accurate, GENIE can be expected to be more accurate than such multistep methods.

3.4.3 Model and Implementation Details

3.4.3.1 Score Models

We train *variance-preserving* DDMs [263] for which $\sigma_t^2 = 1 - \alpha_t^2$. We follow Song et al. [263] and set $\beta(t) = 0.1 + 19.9t$; note that $\alpha_t = e^{-\frac{1}{2}\int_0^t \beta(t')\,dt'}$. All score models are parameterized as either $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$ ($\boldsymbol{\varepsilon}$-prediction) or $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \coloneqq -\frac{\alpha_t \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \sigma_t \mathbf{x}_t}{\sigma_t}$ ($\mathbf{v}$-prediction), where $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ and $\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ are U-Nets [236]. The $\boldsymbol{\varepsilon}$-prediction model is trained using the following score matching objective [109]

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\|_2^2\right], \quad \mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\varepsilon}. \tag{3.68}$$

The $\mathbf{v}$-prediction model is trained using the following score matching objective [244]

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[\|\tfrac{\boldsymbol{\varepsilon} - \sigma_t \mathbf{x}_t}{\alpha_t} - \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\|_2^2\right], \quad \mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\varepsilon}, \tag{3.69}$$

which is referred to as "SNR+1" weighting [244]. The neural network $\mathbf{v}_{\boldsymbol{\theta}}$ is now effectively tasked with predicting $\mathbf{v} \coloneqq \alpha_t \boldsymbol{\varepsilon} - \sigma_t \mathbf{x}_0$.

CIFAR-10: On this dataset, we do not train our own score model, but rather use a checkpoint[1] provided by Song et al. [263]. The model is based on the DDPM++ architecture introduced in Song et al. [263] and predicts $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$.

LSUN Bedrooms and LSUN Church-Outdoor: Both datasets use exactly the same model structure. The model structure is based on the DDPM architecture introduced in Ho et al. [109] and predicts $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$.

ImageNet: This model is based on the architecture introduced in Dhariwal and Nichol [66]. We make a small change to the architecture and replace its sinusoidal time embedding by a Gaussian Fourier projection time embedding [263]. The model is class-conditional and we follow Dhariwal and Nichol [66] and simply add the class embedding to the (Gaussian Fourier projection) time embedding. The model predicts $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$.

Cats (Base): This model is based on the architecture introduced in Dhariwal and Nichol [66]. We make a small change to the architecture and replace its sinusoidal time embedding

[1] The checkpoint can be found at https://drive.google.com/file/d/16_-Ahc6ImZV5ClUc0vM5Iivf8OJ1VSif/view?usp=sharing.

Table 3.4: Model hyperparameters and training details. The CIFAR-10 model is taken from Song et al. [263]; all other models are trained by ourselves.

Hyperparameter	CIFAR-10	LSUN Bedrooms	LSUN Church-Outdoor	ImageNet	Cats (Base)	Cats (Upsampler)
Model						
Data dimensionality (in pixels)	32	128	128	64	128	512
Residual blocks per resolution	8	2	2	3	2	2
Attention resolutions	16	16	16	8	(8, 16)	(8, 16)
Base channels	128	128	128	192	96	192
Channel multipliers	1,2,3,4	1,1,2,2,4,4,4	1,1,2,2,4,4,4	1,2,3,4	1,2,2,3,3	1,1,2,2,3,3,4
EMA rate	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
# of head channels	N/A	N/A	N/A	64	64	64
# of parameters	107M	148M	148M	283M	200M	80.2M
Base architecture	DDPM++ [263]	DDPM [109]	DDPM [109]	[66]	[66]	[66]
Prediction	$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$	$\mathbf{v}$	$\mathbf{v}$
Training						
# of iterations	400k	300k	300k	400k	400k	150k
# of learning rate warmup iterations	100k	100k	100k	100k	100k	100k
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Mixed precision training	✗	✓	✓	✓	✓	✓
Learning rate	10^{-4}	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	10^{-4}	10^{-4}
Gradient norm clipping	1.0	1.0	1.0	1.0	1.0	1.0
Dropout	0.1	0.0	0.0	0.1	0.1	0.1
Batch size	128	256	256	1024	128	64
t_{cutoff}	10^{-5}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}

by a Gaussian Fourier projection time embedding [263]. The model predicts $\mathbf{v}_{\boldsymbol{\theta}}$.

Cats (Upsampler): This model is based on the architecture introduced in Dhariwal and Nichol [66]. We make a small change to the architecture and replace its sinusoidal time embedding by a Gaussian Fourier projection time embedding [263]. The upsampler is conditioned on noisy upscaled lower-resolution images, which are concatenated to the regular channels that form the synthesized outputs of the diffusion model. Therefore, we expand the number of input channels from three to six. We use augmentation conditioning [242] to noise the lower-resolution image. In particular, we upscale $\alpha_{t'}\mathbf{x}_{\text{low}} + \sigma_{t'}\mathbf{z}$, where $\mathbf{x}_{\text{low}}$ is the clean lower-resolution image. During training t' is sampled from $\mathcal{U}[t_{\text{cutoff}}, 1]$. During inference, t' is a hyper-parameter which we set to 0.1 for all experiments.

We use two-independent Gaussian Fourier projection embeddings for t and t' and concatenate them before feeding them into the layers of the U-Net.

Model Hyperparameters and Training Details: All model hyperparameters and training details can be found in Table 3.4.

3.4.3.2 Prediction Heads

We model the derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ using a small prediction head $\boldsymbol{k}_{\boldsymbol{\psi}}$ on top of the first-order score model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$. In particular, we provide the last feature layer from the $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ network together with its time embedding as well as $\mathbf{x}_t$ and the output of $\boldsymbol{\varepsilon}(\mathbf{x}_t, t)$ to the prediction head (see Figure 3.4 for a visualization). We found modeling $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ to be effective even for our Cats models that learn to predict $\mathbf{v} = \alpha_t\boldsymbol{\varepsilon} - \sigma_t\mathbf{x}_0$ rather than $\boldsymbol{\varepsilon}$. Directly learning $d_{\gamma_t}\mathbf{v}_{\boldsymbol{\theta}}$ and adapting the mixed network parameterization (see Paragraph 3.4.3.2.3) could potentially improve results further. We leave this exploration to future work.

We provide additional details on our architecture next.

3.4.3.2.1 Model Architecture The architecture of our prediction heads is based on (modified) BigGAN residual blocks [30, 263]. To minimize computational overhead, we only use a single residual block.

In particular, we concatenate the last feature layer with $\mathbf{x}_t$ as well as $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ and feed it into a convolutional layer. For the upsampler, we also condition on the noisy up-scaled lower resolution image. We experimented with normalizing the feature layer before concatenation. The output of the convolutional layer as well as the time embedding are then fed to the residual block. Similar to U-Nets used in score models, we normalize the output of the residual block and apply an activation function. Lastly, the signal is fed to another convolutional layer that brings the number of channels to a desired value (in our case nine, three for each $\boldsymbol{k}_{\boldsymbol{\psi}}^{(i)}$, $i \in \{1, 2, 3\}$, in Equation (3.73)).

All model hyperparameters can be found in Table 3.5. We also include the additional computational overhead induced by the prediction heads in Table 3.5; see Paragraph 3.4.3.2.5 for details on how we measured the overhead.

3.4.3.2.2 Training Details We train for 50k iterations using Adam [148]. We experimented with two base learning rates: 10^{-4} and $5 \cdot 10^{-5}$. We furthermore tried two "optimization setups": (linearly) warming up the learning rate in the first 10k iterations (score models are often trained by warming up the learning rate in the first 100k iterations) or, following Salimans and Ho [244], linearly decaying the learning rate to 0 in the entire 50k iterations of training; we respectively refer to these two setups as "warmup" and

Table 3.5: Model hyperparameters and training details for the prediction heads.

Hyperparameter	CIFAR-10	LSUN Bedrooms	LSUN Church-Outdoor	ImageNet	Cats (Base)	Cats (Upsampler)
Model						
Data dimensionality	32	128	128	64	128	512
EMA rate	0	0	0	0	0	0
Number of channels	128	128	128	196	196	92
# of parameters	526k	526k	526k	1.17M	1.17M	302k
Normalize $\mathbf{x}_{\text{embed}}$	✗	✗	✓	✗	✗	✗
Training						
# of iterations	20k	40k	35k	15k	20k	20k
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Optimization setup	Decay	Warmup	Warmup	Warmup	Warmup	Warmup
Mixed precision training	✗	✗	✗	✗	✗	✗
Learning rate	$5 \cdot 10^{-5}$	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Gradient norm clipping	1.0	1.0	1.0	1.0	1.0	1.0
Dropout	0.0	0.0	0.0	0.0	0.0	0.0
Batch size	128	256	256	256	64	16
t_{cutoff}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Inference						
Add. comp. overhead	1.47%	14.0%	14.4%	2.83%	7.55%	13.3%

"decay". We measure the FID every 5k iterations and use the best checkpoint.

Note that we have to compute the Jacobian-vector products in Equation (3.19) via automatic differentiation during training. We repeatedly found that computing the derivative $\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}$ via automatic differentiation leads to numerical instability (`NaN`) for small t when using mixed precision training. For simplicity, we turned off mixed precision training altogether. However, training performance could have been optimized by only turning off mixed precision training for the derivative $\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}$.

All training details can be found in Table 3.5.

3.4.3.2.3 Mixed Network Parameterization Our mixed network parameterization is derived from a simple single data point assumption, i.e., $p_t(\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t; \mathbf{0}, \sigma_t^2 \boldsymbol{I})$. This assumption leads to $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \frac{\mathbf{x}_t}{\sigma_t}$ which we can plug into the three terms of Equation (3.19):

$$\frac{1}{\sqrt{\gamma_t^2 + 1}} \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \frac{1}{\sqrt{\gamma_t^2 + 1}} \frac{\mathbf{x}_t}{\sigma_t^2} = \frac{1}{\gamma_t} \frac{\mathbf{x}_t}{\sigma_t}, \tag{3.70}$$

and

$$-\frac{\gamma_t}{1 + \gamma_t^2} \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \mathbf{x}_t \approx -\frac{\gamma_t}{\sigma_t \left(1 + \gamma_t^2\right)} \mathbf{x}_t = -\frac{\gamma_t}{1 + \gamma_t^2} \frac{\mathbf{x}_t}{\sigma_t}, \tag{3.71}$$

and finally

$$\frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{dt}{d\gamma_t} \approx -\frac{\mathbf{x}_t}{\sigma_t^2}\frac{d\sigma_t}{dt}\frac{dt}{d\gamma_t} = -\frac{\gamma_t^2+1}{\gamma_t^2}\mathbf{x}_t\frac{1}{(\gamma_t^2+1)^{3/2}} = -\frac{1}{\gamma_t(1+\gamma_t^2)}\frac{\mathbf{x}_t}{\sigma_t}, \tag{3.72}$$

where we have used $\sigma_t = \frac{\gamma_t}{\sqrt{\gamma_t^2+1}}$. This derivation therefore implies the following mixed network parameterization

$$\boldsymbol{k}_{\boldsymbol{\psi}} = -\frac{1}{\gamma_t}\boldsymbol{k}_{\boldsymbol{\psi}}^{(1)} + \frac{\gamma_t}{1+\gamma_t^2}\boldsymbol{k}_{\boldsymbol{\psi}}^{(2)} + \frac{1}{\gamma_t(1+\gamma_t^2)}\boldsymbol{k}_{\boldsymbol{\psi}}^{(3)} \approx d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}, \tag{3.73}$$

where $\boldsymbol{k}_{\boldsymbol{\psi}}^{(i)}(\mathbf{x}_t, t)$, $i \in \{1, 2, 3\}$, are different output channels of the neural network (i.e. the additional head on top of the $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$ network). To provide additional intuition, we basically replaced the $-\frac{\mathbf{x}_t}{\sigma_t}$ terms in Equations (3.70) to (3.72) by neural networks. However, we know that for approximately Normal data $\frac{\mathbf{x}_t}{\sigma_t} \approx \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, where $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ predicts "noise" values $\boldsymbol{\varepsilon}$ that were drawn from a standard Normal distribution and are therefore varying on a well-behaved scale. Consequently, up to the Normal data assumption, we can also expect our prediction heads $\boldsymbol{k}_{\boldsymbol{\psi}}^{(i)}(\mathbf{x}_t, t)$ in the parameterization in Equation (3.73) to predict well-behaved output values, which should make training stable. This *mixed network parameterization* approach is inspired by the mixed score parameterization from Vahdat et al. [280] and Dockhorn et al. [75].

3.4.3.2.4 Pseudocode In this subsection, we provide pseudocode for training our prediction heads $\boldsymbol{k}_{\boldsymbol{\psi}}$ and using them for sampling with GENIE. In Algorithm 2, the analytical $\frac{dt}{d\gamma_t}$ is an implicit hyperparameter of the DDM as it depends on α_t. For our choice of $\alpha_t = e^{-\frac{1}{2}\int_0^t 0.1+19.9t'\,dt'}$ (see Section 3.4.3.1), we have

$$\frac{dt}{d\gamma_t} = \frac{\frac{2\gamma_t}{19.9\left(\gamma_t^2+1\right)}}{\sqrt{\left(\frac{0.1}{19.9}\right)^2 + \frac{2\log\left(\gamma_t^2+1\right)}{19.9}}}, \tag{3.74}$$

where $\gamma_t = \sqrt{\frac{1-\alpha_t^2}{\alpha_t^2}}$.

In Line 0, we are free to use any time discretization $t_0 = 1.0 > t_1 > \cdots > t_N = t_{\text{cutoff}}$. When referring to "linear striding" in this work, we mean the time discretization $t_n = 1.0 - (1.0 - t_{\text{cutoff}})\frac{n}{N}$. When referring to "quadratic striding" in this work, we mean the time discretization $t_n = \left(1.0 - (1.0 - \sqrt{t_{\text{cutoff}}})\frac{n}{N}\right)^2$.

Algorithm 2 Training prediction heads $\boldsymbol{k}_{\boldsymbol{\psi}}$

Input: Score model $\boldsymbol{s}_{\boldsymbol{\theta}} \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$, number of training iterations N.
Output: Trained prediction head $\boldsymbol{k}_{\boldsymbol{\psi}}$.

for $n = 1$ **to** N **do**
  Sample $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$, $t \sim \mathcal{U}[t_{\text{cutoff}}, 1], \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  Set $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\varepsilon}$
  Compute $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$
  Compute the spatial JVP $\text{JVP}_\text{s} = \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t} \left(\frac{1}{\sqrt{\gamma_t^2 + 1}} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \frac{\gamma_t}{1+\gamma_t^2} \mathbf{x}_t \right)$ via AD
  Compute the temporal JVP $\text{JVP}_\text{t} = \frac{\partial \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t} \frac{dt}{d\gamma_t}$ via AD ($\frac{dt}{d\gamma_t}$ can be computed analytically)
  Compute $\boldsymbol{k}_{\boldsymbol{\psi}}(\mathbf{x}_t, t)$ using the mixed parameterization in Equation (3.73)
  Update weights $\boldsymbol{\psi}$ to minimize $\gamma_t^2 \| \boldsymbol{k}_{\boldsymbol{\psi}}(\mathbf{x}_t, t) - d_{\gamma_t} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \|_2^2$, where $d_{\gamma_t} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \text{JVP}_\text{s} - \text{JVP}_\text{t}$
end for

Algorithm 3 GENIE sampling

Input: Score model $\boldsymbol{s}_{\boldsymbol{\theta}} \coloneqq -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\sigma_t}$, prediction head $\boldsymbol{k}_{\boldsymbol{\psi}}$, number of sampler steps N, time discretization $\{t_n\}_{n=0}^N$.
Output: Generated GENIE output sample $\mathbf{y}$.
Sample $\mathbf{x}_{t_0} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
Set $\bar{\mathbf{x}}_{t_0} = \sqrt{1 + \gamma_{t_0}^2} \mathbf{x}_{t_0}$ ▷ *Note that* $\bar{\mathbf{x}}_{t_n} = \sqrt{1 + \gamma_{t_n}^2} \mathbf{x}_{t_n}$ *for all* t_n
for $n = 0$ **to** $N - 1$ **do**
  if AFS and $n = 0$ **then**
    $\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + (\gamma_{t_{n+1}} - \gamma_{t_n}) \mathbf{x}_{t_n}$
  else
    $\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + (\gamma_{t_{n+1}} - \gamma_{t_n}) \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2} (\gamma_{t_{n+1}} - \gamma_{t_n})^2 \boldsymbol{k}_{\boldsymbol{\psi}}(\mathbf{x}_{t_n}, t_n)$
  end if
  $\mathbf{x}_{t_{n+1}} = \frac{\bar{\mathbf{x}}_{t_{n+1}}}{\sqrt{1 + \gamma_{t_{n+1}}^2}}$
end for
if Denoising **then**
  $\mathbf{y} = \frac{\mathbf{x}_{t_N} - \sigma_{t_N} \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_N}, t_N)}{\alpha_{t_N}}$
else
  $\mathbf{y} = \mathbf{x}_{t_N}$
end if

3.4.3.2.5 Measuring Computational Overhead Our prediction heads induce a slight computational overhead since their forward pass has to occur after the forward pass of the score model. We measure the overhead as follows: first, we measure the inference time of the score model itself. We do five forward passes to "warm-up" the model and then subsequently synchronize via `torch.cuda.synchronize()`. We then measure the total wall-clock time of 50 forward passes. We then repeat this process using a combined forward pass: first the score model and subsequently the prediction head. We choose the batch size to (almost) fill the entire GPU memory. In particular we chose batch sizes of 512, 128, 128, 64, 64, and 8, for CIFAR-10, LSUN Bedrooms, LSUN Church-Outdoor, ImageNet, Cats (base), and Cats (upsampler), respectively. The computational overhead for each model is reported in Table 3.5. This measurement was carried out on a single NVIDIA 3080 Ti GPU.

3.4.4 Learning Higher-Order Gradients without Automatic Differentiation and Distillation

In this work, we learn the derivative $d_{\gamma_t}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}$, which includes a spatial and a temporal Jacobian-vector product, by distillation based on automatic differentiation (AD). We now derive an alternative learning objective for the spatial Jacobian-vector product (JVP) which does not require any AD. We start with the following (conditional) expectation

$$\mathbb{E}\left[\alpha_t^2\mathbf{x}_0\mathbf{x}_t^\top - \alpha_t\left[\mathbf{x}_0\mathbf{x}_t^\top + \mathbf{x}_t\mathbf{x}_0^\top\right] \mid \mathbf{x}_t, t\right] = -\mathbf{x}_t\mathbf{x}_t^\top + \sigma_t^4\boldsymbol{S}_2(\mathbf{x}_t, t) + \sigma_t^4\boldsymbol{s}_1(\mathbf{x}_t, t)\boldsymbol{s}_1(\mathbf{x}_t, t)^\top + \sigma_t^2\boldsymbol{I}, \tag{3.75}$$

where $\boldsymbol{s}_1(\mathbf{x}_t, t) \coloneqq \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)$ and $\boldsymbol{S}_2(\mathbf{x}_t, t) \coloneqq \nabla_{\mathbf{x}_t}^\top\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)$. The above formula is derived in Meng et al. [Theorem 1, 195]. Adding $\mathbf{x}_t\mathbf{x}_t^\top$ to Equation (3.75) and subsequently dividing by σ_t^2, we have

$$\mathbb{E}\left[\frac{\alpha_t^2}{\sigma_t^2}\mathbf{x}_0\mathbf{x}_t^\top - \frac{\alpha_t}{\sigma_t^2}\left[\mathbf{x}_0\mathbf{x}_t^\top + \mathbf{x}_t\mathbf{x}_0^\top\right] + \frac{1}{\sigma_t^2}\mathbf{x}_t\mathbf{x}_t^\top \mid \mathbf{x}_t, t\right] = \sigma_t^2\boldsymbol{S}_2(\mathbf{x}_t, t) + \sigma_t^2\boldsymbol{s}_1(\mathbf{x}_t, t)\boldsymbol{s}_1(\mathbf{x}_t, t)^\top + \boldsymbol{I}, \tag{3.76}$$

where we could pull the $\frac{1}{\sigma_t^2}\mathbf{x}_t\mathbf{x}_t^\top$ term into the expectation because it is conditioned on t and $\mathbf{x}_t$. Using $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}$, we can rewrite the above as

$$\mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top \mid \mathbf{x}_t, t\right] = \sigma_t^2\boldsymbol{S}_2(\mathbf{x}_t, t) + \sigma_t^2\boldsymbol{s}_1(\mathbf{x}_t, t)\boldsymbol{s}_1(\mathbf{x}_t, t)^\top + \boldsymbol{I}. \tag{3.77}$$

For an arbitrary $\boldsymbol{v} \coloneqq \boldsymbol{v}(\mathbf{x}_t, t)$, we then have

$$\mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\boldsymbol{v} \mid \mathbf{x}_t, t\right] = \sigma_t^2\boldsymbol{S}_2(\mathbf{x}_t, t)\boldsymbol{v} + \sigma_t^2\boldsymbol{s}_1(\mathbf{x}_t, t)\boldsymbol{s}_1(\mathbf{x}_t, t)^\top\boldsymbol{v} + \boldsymbol{v}. \tag{3.78}$$

Therefore, we can develop a score matching-like learning objective for the (general) spatial JVP $\boldsymbol{o}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \boldsymbol{S}_2(\mathbf{x}_t, t)\boldsymbol{v}$ as

$$\mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[g_{\text{no-ad}}(t)\|\boldsymbol{o}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)+\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)^\top\boldsymbol{v}+\frac{1}{\sigma_t^2}\boldsymbol{v}-\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\boldsymbol{v}\|_2^2\right], \tag{3.79}$$

for some weighting function $g_{\text{no-ad}}(t)$. Setting $\boldsymbol{v}(\mathbf{x}_t, t) = -\sigma_t\left(\frac{1}{\sqrt{\gamma_t^2+1}}\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t) - \frac{\gamma_t}{1+\gamma_t^2}\mathbf{x}_t\right)$, would recover the spatial JVP needed for the computation of $d_{\gamma_t}\boldsymbol{\varepsilon}$. In the initial phase of this project, we briefly experimented with learning the spatial JVP using this approach; however, we found that our distillation approach worked significantly better.

3.4.5 Toy Experiments

For all toy experiments in Section 3.3.3, we consider the following ground truth distribution:

$$p_0(\mathbf{x}_0) = \frac{1}{8}\sum_{i=1}^{8} p_0^{(i)}(\mathbf{x}_0), \tag{3.80}$$

where

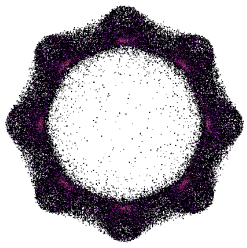
$$p_0^{(i)}(\mathbf{x}_0) = \frac{1}{8}\sum_{j=1}^{8}\mathcal{N}(\mathbf{x}_0, s_1\boldsymbol{\mu}_i + s_1 s_2\boldsymbol{\mu}_j, \sigma^2\boldsymbol{I}). \tag{3.81}$$

We set $\sigma = 10^{-2}$, $s_1 = 0.9$, $s_2 = 0.2$, and

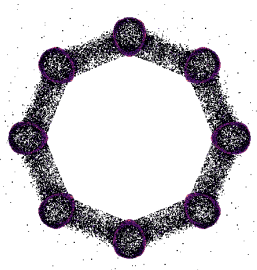
$$\boldsymbol{\mu}_1 = \begin{pmatrix}1\\0\end{pmatrix}, \quad \boldsymbol{\mu}_2 = \begin{pmatrix}-1\\0\end{pmatrix}, \quad \boldsymbol{\mu}_3 = \begin{pmatrix}0\\1\end{pmatrix}, \quad \boldsymbol{\mu}_4 = \begin{pmatrix}0\\-1\end{pmatrix}$$
$$\boldsymbol{\mu}_5 = \begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}, \quad \boldsymbol{\mu}_6 = \begin{pmatrix}\frac{1}{\sqrt{2}}\\-\frac{1}{\sqrt{2}}\end{pmatrix}, \quad \boldsymbol{\mu}_7 = \begin{pmatrix}-\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}, \quad \boldsymbol{\mu}_8 = \begin{pmatrix}-\frac{1}{\sqrt{2}}\\-\frac{1}{\sqrt{2}}\end{pmatrix}.$$

The ground truth distribution is visualized in Figure 3.3a. Note that we can compute the score functions (and all its derivatives) analytically for Gaussian mixture distributions.

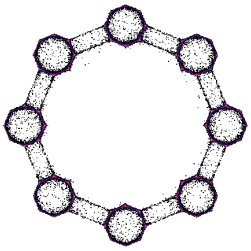
In Figure 3.3, we compared DDIM to GENIE for sampling using the analytical score function of the ground truth distribution with 25 solver steps. In Figure 3.12, we repeated this experiment for 5, 10, 15, and 20 solver steps. We found that in particular for $n = 10$ both solvers generate samples in interesting patterns.



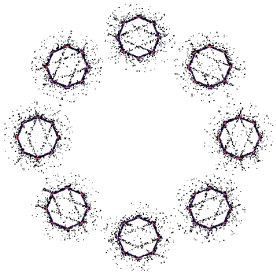
(a) DDIM, $n = 5$



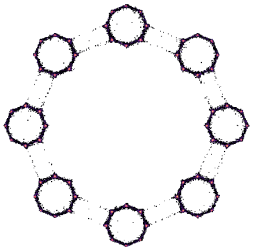
(b) GENIE, $n = 5$



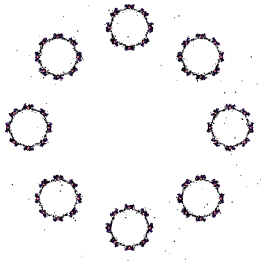
(c) DDIM, $n = 10$



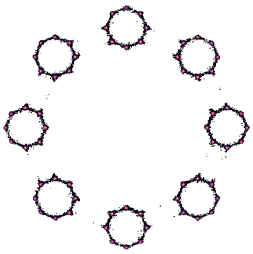
(d) GENIE, $n = 10$



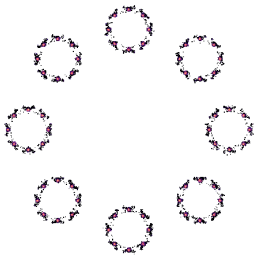
(e) DDIM, $n = 15$



(f) GENIE, $n = 15$



(g) DDIM, $n = 20$



(h) GENIE, $n = 20$

Figure 3.12: Modeling a complex 2D toy distribution: Samples are generated with DDIM and GENIE with n solver steps using the analytical score function of the ground truth distribution (visualized in Figure 3.3a). Zoom in for details.

3.4.6 Image Experiments

3.4.6.1 Evaluation Metrics, Baselines, and Datasets

Metrics: We quantitatively measure sample quality via Fréchet Inception Distance [FID, 105]. It is common practice to use 50k samples from the training set for reference statistics. We follow this practice for all datasets except for ImageNet and Cats. For ImageNet, we follow Dhariwal and Nichol [66] and use the entire training set for reference statistics. For the small Cats dataset, we use the training as well as the validation set for reference statistics.

Baselines: We run baseline experiments using two publicly available repositories. The `score_sde_pytorch` repository is licensed according to the Apache License 2.0; see also their license file here. The `CLD-SGM` repository is licensed according to the NVIDIA Source Code License; see also their license file here.

Datasets: We link here the websites of the datasets used in this experiment: CIFAR-10, LSUN datasets, ImageNet, and AFHQv2.

3.4.6.2 Analytical First Step (AFS)

The forward process of DDMs generally converges to an analytical distribution. This analytical distribution is then used to sample from DDMs, defining the initial condition for the generative ODE/SDE. For example, for variance-preserving DDMs, we have $p_1(\mathbf{x}_1) \approx \mathcal{N}(\mathbf{x}_1; \mathbf{0}, \boldsymbol{I})$.

In this work, we try to minimize the computational complexity of sampling from DDMs, and therefore operate in a low NFE regime. In this regime, every additional function evaluation makes a significant difference. We therefore experimented with replacing the learned score with the (analytical score) of $\mathcal{N}(\mathbf{0}, \boldsymbol{I}) \approx p_1(\mathbf{x}_1)$ in the first step of the ODE solver. This "gained" function evaluation can then be used as an additional step in the ODE solver later.

In particular, we have

$$\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_1, 1) \approx \mathbf{x}_1, \tag{3.82}$$

and $\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_1,1)}{d\gamma_1} \approx \mathbf{0}$ as shown below:

$$\frac{d\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_1,1)}{d\gamma_1} \approx \frac{d\mathbf{x}_t}{d\gamma_t}|_{t=1} \tag{3.83}$$

$$= -\frac{\gamma_t}{\gamma_t^2+1}\left[\mathbf{x}_t + \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)\right]|_{t=1} \quad \text{(using Equation (3.35))} \tag{3.84}$$

$$\approx \mathbf{0} \quad \text{(using normal assumption } \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t) \approx -\mathbf{x}_t) \tag{3.85}$$

Given this, the AFS step becomes identical to the Euler update that uses the Normal score function for $\mathbf{x}_1$. This step is shown in the pseudocode in Line 0.

3.4.6.3 Classifier-Free Guidance

As discussed in Section 3.3.5.2, to guide diffusion sampling towards particular classes, we replace $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)$ with

$$\hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t,t,c,w) = (1+w)\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t,c) - w\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t), \tag{3.86}$$

where $w > 1.0$ is the "guidance scale", in the DDIM ODE. We experiment with classifier-free guidance on ImageNet. In Equation (3.86) we re-use the conditional ImageNet score model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t,c)$ trained before (see Section 3.4.3.1 for details), and train an additional unconditional ImageNet score model $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)$ using the exact same setup (and simply setting the class embedding to zero). We also re-use the conditional prediction head trained on top of the conditional ImageNet score model and train an additional prediction head for the unconditional model. Note that for both the score models as well as the prediction heads, we could share parameters between the models to reduce computational complexity [108]. The modified GENIE scheme for classifier-free guidance is then given as

$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + (\gamma_{t_{n+1}} - \gamma_{t_n})\hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n},t_n,c,w) + \frac{1}{2}(\gamma_{t_{n+1}} - \gamma_{t_n})^2\hat{\boldsymbol{k}}_{\boldsymbol{\psi}}(\mathbf{x}_{t_n},t_n,c,w), \tag{3.87}$$

where

$$\hat{\boldsymbol{k}}_{\boldsymbol{\psi}}(\mathbf{x}_{t_n},t_n,c,w) = (1+w)\boldsymbol{k}_{\boldsymbol{\psi}}(\mathbf{x}_{t_n},t_n,c) - w\boldsymbol{k}_{\boldsymbol{\psi}}(\mathbf{x}_{t_n},t_n). \tag{3.88}$$

3.4.6.4 Encoding

To encode a data point $\mathbf{x}_0$ into latent space, we first "diffuse" the data point to $t = 10^{-3}$, i.e., $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$. We subsequently simulate the generative ODE (backwards) from $t = 10^{-3}$ to $t = 1$, obtaining the latent point $\mathbf{x}_1$.

To decode a latent point $\mathbf{x}_1$, we simulate the generative ODE (forwards) from $t = 1.0$ to $t = 10^{-3}$. We then denoise the data point, i.e., $\mathbf{x}_0 = \frac{\mathbf{x}_t - \sigma_t \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\alpha_t}$. Note that denoising is generally optional to sample from DDMs; however, for our encoding-decoding experiment we always used denoising in the decoding part to match the inital "diffusion" in the encoding part.

3.4.6.5 Latent Space Interpolation

We can use encoding to perform latent space interpolation of two data points $\mathbf{x}_0^{(0)}$ and $\mathbf{x}_0^{(1)}$. We first encode both data points, following the encoding setup from Section 3.4.6.4, and obtain $\mathbf{x}_1^{(0)}$ and $\mathbf{x}_1^{(1)}$, respectively. We then perform spherical interpolation of the latent codes:

$$\mathbf{x}_1^{(b)} = \mathbf{x}_1^{(0)} \sqrt{1 - \mathrm{b}} + \mathbf{x}_2^{(1)} \sqrt{\mathrm{b}}, \quad \mathrm{b} \in [0, 1]. \tag{3.89}$$

Subsequently, we decode the latent code $\mathbf{x}_1^{(b)}$ following the decoding setup from Section 3.4.6.4. In Figure 3.13, we show latent space interpolations for LSUN Church-Outdoor and LSUN Bedrooms.

3.4.6.6 Extended Quantitative Results

In this subsection, we show additional quantitative results not presented in the main paper. In particular, we show results for all four hyperparameter combinations (binary choice of AFS and binary choice of denoising) for methods evaluated by ourselves. For these methods (i.e., GENIE, DDIM, S-PNDM, F-PNDM, Euler–Maruyama), we follow the **Synthesis Strategy** outlined in Section 3.3.5, with the exception that we use linear striding instead of quadratic striding for S-PNDM [176] and F-PNDM [176]. To apply quadratic striding to these two methods, one would have to derive the Adams–Bashforth methods for non-constant step sizes which is beyond the scope of our work.

Results can be found in Tables 3.8 to 3.13. As expected, AFS can considerably improve results for almost all methods, in particular for NFEs ≤ 15. Denoising, on the other hand, is more important for larger NFEs. For our Cats models, we initially found that denoising hurts performance, and therefore did not further test it in all settings.

Recall Scores. We quantify the sample diversity of GENIE and other fast samplers

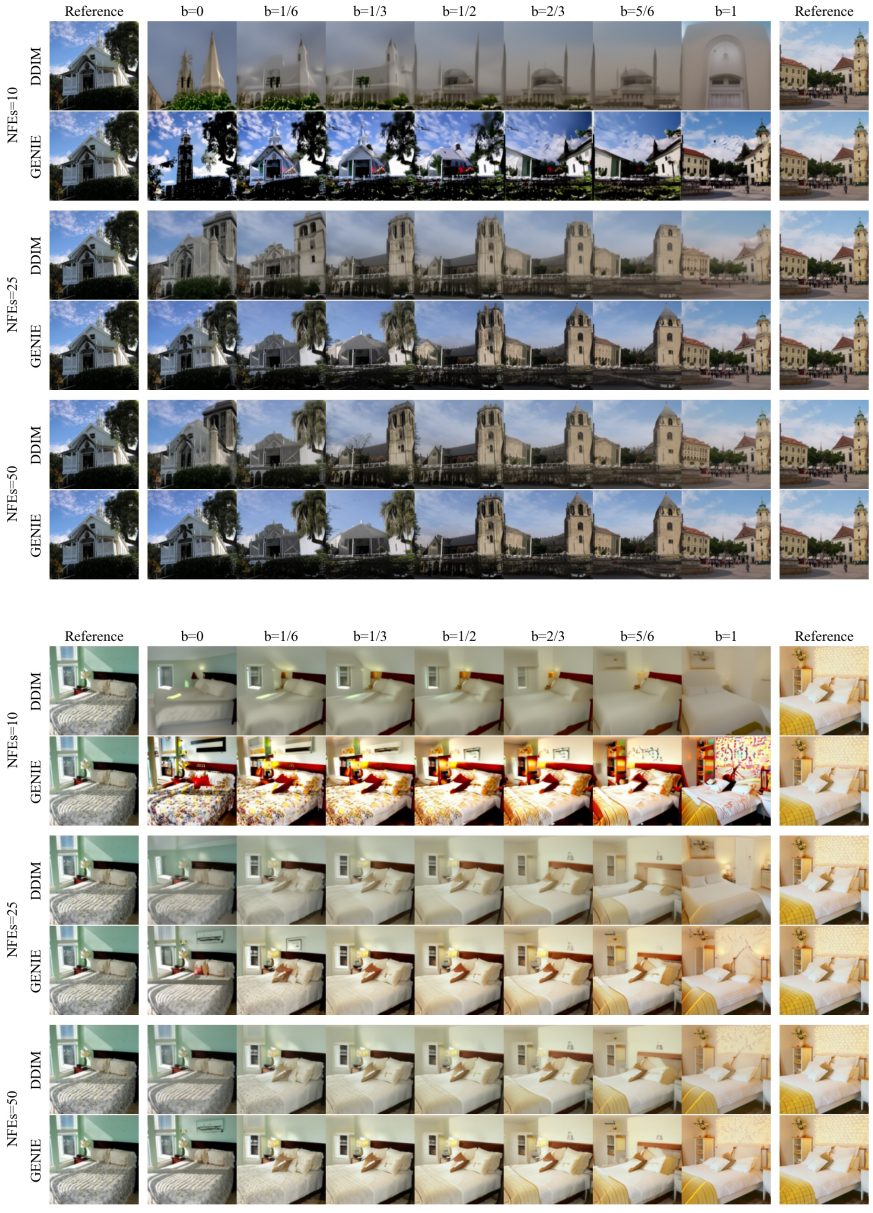


Figure 3.13: Latent space interpolations for LSUN Church-Outdoor (Top) and LSUN Bedrooms (Bottom). Note that $b = 0$ and $b = 1$ correspond to the decodings of the encoded reference images. Since this encode-decode loop is itself not perfect, the references are not perfectly reproduced at $b = 0$ and $b = 1$.

Table 3.6: Unconditional CIFAR-10 generative performance, measured in Recall (higher values are better). All methods use the same score model checkpoint.

Method	AFS	Denoising	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	✗	0.28	0.48	0.54	0.56	0.56
	✗	✓	0.21	0.45	0.52	0.56	0.57
	✓	✗	0.27	0.47	0.53	0.56	0.56
	✓	✓	0.19	0.46	0.53	0.55	0.56
DDIM [258]	✗	✗	0.10	0.27	0.38	0.43	0.46
	✗	✓	0.07	0.24	0.35	0.42	0.46
	✓	✗	0.08	0.27	0.38	0.43	0.46
	✓	✓	0.04	0.24	0.36	0.42	0.45
S-PNDM [176]	✗	✗	0.06	0.30	0.43	0.49	0.52
	✗	✓	0.02	0.25	0.39	0.46	0.50
	✓	✗	0.11	0.33	0.45	0.50	0.53
	✓	✓	0.06	0.29	0.41	0.47	0.51
F-PNDM [176]	✗	✗	N/A	N/A	0.55	0.57	0.58
	✗	✓	N/A	N/A	0.52	0.56	0.57
	✓	✗	N/A	N/A	0.55	0.58	0.59
	✓	✓	N/A	N/A	0.54	0.56	0.57
Euler–Maruyama	✗	✗	0.00	0.00	0.00	0.02	0.08
	✗	✓	0.00	0.00	0.00	0.03	0.06
	✓	✗	0.00	0.00	0.00	0.03	0.09
	✓	✓	0.00	0.00	0.00	0.03	0.09

using the recall score [243]. In particular, we follow DDGAN [295] and use the improved recall score [162]; results on CIFAR-10 can be found in Table 3.6. As expected, we can see that for all methods recall scores suffer as the NFEs decrease. Compared to the baselines, GENIE achieves excellent recall scores, being on par with F-PNDM for NFE$\geq$ 15. However, F-PNDM cannot be run for NFE$\leq$10 (due to its additional Runge–Kutta warm-up iterations). Overall, these results confirm that GENIE offers strong sample diversity when compared to other common samplers using the same score model checkpoint.

Striding Schedule Grid Search. As discussed in Section 3.3.5 the fixed quadratic striding schedule (for choosing the times t for evaluating the model during synthesis under fixed NFE budgets) used in GENIE may be sub-optimal, in particular for small NFEs. To explore this, we did a small grid search over three different striding schedules. As described in Paragraph 3.4.3.2.4, the quadratic striding schedule can be written as $t_n =$

Table 3.7: Unconditional CIFAR-10 generative performance (measured in FID) using our GENIE and DDIM [258] with different striding schedules using exponents $\rho \in \{1.5, 2.0, 2.5\}$.

Method	ρ	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE	1.5	**11.2**	**5.28**	5.03	4.35	3.97
	2.0	13.9	5.97	**4.49**	**3.94**	3.67
	2.5	17.8	7.19	4.57	**3.94**	**3.64**
DDIM	1.5	**27.6**	13.5	8.97	7.20	6.15
	2.0	29.7	**11.2**	**7.35**	**5.87**	**5.16**
	2.5	33.2	13.4	8.28	6.36	5.39

$\left(1.0 - (1.0 - \sqrt{t_{\text{cutoff}}})\frac{n}{N}\right)^2$, and easily be generalized to

$$t_n = \left(1.0 - (1.0 - t_{\text{cutoff}}^{1/\rho})\frac{n}{N}\right)^{\rho}, \rho > 1. \tag{3.90}$$

In particular, besides the quadratic schedule $\rho = 2$, we also tested the two additional values $\rho = 1.5$ and $\rho = 2.5$. We tested these schedules on GENIE as well as DDIM [258]; note that the other two comptetive baselines, S-PNDM [176] and F-PNDM [176], rely on linear striding, and therefore a grid search is not applicable. We show results for GENIE and DDIM in Table 3.7; for each combination of solver and NFE we applied the best synthesis strategy (whether or not we use denoising and/or the analytical first step) of quadratic striding ($\rho = 2.0$) also to $\rho = 1.5$ and $\rho = 2.5$. As can be seen in the table, $\rho = 1.5$ improves for both DDIM and GENIE for NFE=5 (over the quadratic schedule $\rho = 2$), whereas larger ρ are preferred for larger NFE. The improvement of GENIE from 13.9 to 11.2 FID for NFE=5 is significant.

Discretization Errors of GENIE compared to other Fast Samplers. We compute discretization errors, in particular local and global truncation errors, of GENIE and compare to existing faster solvers. We are using the CIFAR-10 model. We initially sample 100 latent vectors $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and then, starting from those latent vectors, synthesize 100 approximate ground truth trajectories (GTTs) using DDIM with 1k NFEs (for that many steps, the discretization error is negligible; hence, we can treat this as a pseudo ground truth).

We then synthesize 100 sample trajectories for DDIM [258], S-PNDM [176], F-PNDM [176], and GENIE (for NFEs=$\{5, 10, 15, 20, 25\}$, similar to the main experiments) using the same

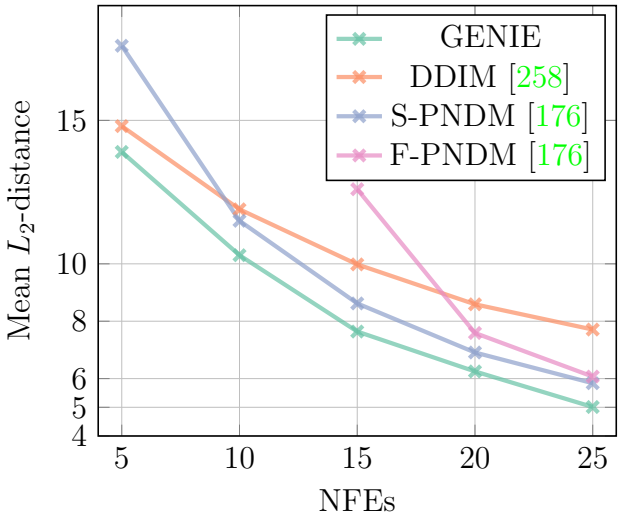


Figure 3.14: **Global Truncation Error:** L_2-distance of generated outputs by the fast samplers to the (approximate) ground truth (computed using DDIM with 1k NFEs) in Inception feature space [268]. Results are averaged over 100 samples.

latent vectors as starting points that were used to generate the GTTs. DDIM, S-PNDM, and F-PNDM are training-free methods that can be run on the exact same score model, which also our GENIE relies on. Thereby, we are able to isolate discretization errors from errors in the learnt score function. We then compute the average L_2-distance (in Inception feature space [268]) between the output image of the fast samplers and the "output" of the pseudo GTT. As can be seen in Figure 3.14, GENIE outperforms the three other methods on all NFEs.

Comparing the local truncation error (LTE) of different higher-order solvers can unfortunately not be done in a fair manner. Similar to DDIM, GENIE only needs the current value and a single NFE to predict the next step. In contrast, multistep methods rely on a history of predictions and Runge–Kutta methods rely on multiple NFEs to predict the next step. Thus, we can only fairly compare the LTE of GENIE to the LTE of DDIM. In particular, we compute LTEs at three starting times $t \in \{0.1, 0.2, .5\}$ (similar to what we did in Figure 3.2). For each t, we then compare one step predictions for different step sizes Δt against the ground truth trajectory (L_2-distance in data space averaged over 100 predictions; since we are not operating directly in image space at these intermediate t, using inception feature would not make sense here). As expected, we can see in Figure 3.15 that GENIE has smaller LTE than DDIM for all starting times t.

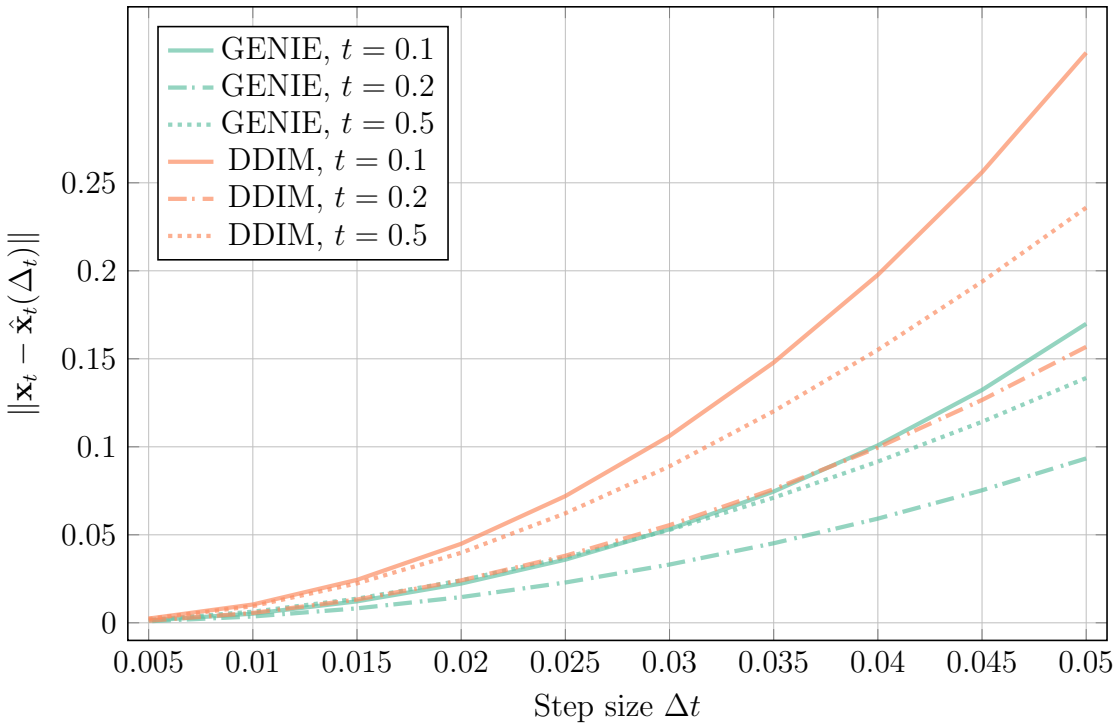


Figure 3.15: **Local Truncation Error:** Single step (local discretization) error, measured in L_2-distance to (approximate) ground truth (computed using DDIM with 1k NFEs) in data space and averaged over 100 samples, for GENIE and DDIM for three starting time points $t \in \{0.1, 0.2, 0.5\}$ (this is, the t from which a small step with size Δt is taken).

Table 3.8: Unconditional CIFAR-10 generative performance (measured in FID). Methods above the middle line use the same score model checkpoint; methods below all use different ones. (†): numbers are taken from literature. This table is an extension of Table 3.1.

Method	AFS	Denoising	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	✗	15.4	**5.97**	4.70	4.30	4.10
	✗	✓	23.5	6.91	4.74	4.02	3.72
	✓	✗	13.9	6.04	4.76	4.33	4.18
	✓	✓	17.9	6.27	**4.49**	**3.94**	**3.67**
DDIM [258]	✗	✗	30.1	11.6	7.56	6.00	5.27
	✗	✓	37.9	13.9	8.76	6.77	5.76
	✓	✗	29.7	11.2	7.35	5.87	5.16
	✓	✓	35.2	12.8	8.17	6.39	5.49
S-PNDM [176]	✗	✗	60.2	12.1	7.16	5.48	4.62
	✗	✓	101	17.2	10.8	8.74	7.62
	✓	✗	35.9	10.3	6.61	5.20	4.51
	✓	✓	56.8	14.9	10.2	8.37	7.35
F-PNDM [176]	✗	✗	N/A	N/A	12.1	6.58	4.89
	✗	✓	N/A	N/A	19.5	10.6	8.43
	✓	✗	N/A	N/A	10.3	5.96	4.73
	✓	✓	N/A	N/A	15.7	10.9	8.52
Euler–Maruyama	✗	✗	364	236	178	121	85.0
	✗	✓	391	235	191	129	89.9
	✓	✗	325	230	164	112	80.3
	✓	✓	364	235	176	120	83.6
FastDDIM [156] (†)	✗	✓	-	9.90	-	5.05	-
Learned Sampler [286] (†)	✗	✓	**12.4**	7.86	5.90	4.72	4.25
Analytic DDIM (LS) [20] (†)	✗	✓	-	14.0	-	-	5.71
CLD-SGM [75]	✗	✗	334	306	236	162	106
VESDE-PC [263]	✗	✓	461	461	461	461	462

Table 3.9: Conditional ImageNet generative performance (measured in FID).

Method	AFS	Denoising	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	✗	23.4	8.35	6.13	5.36	5.00
	✗	✓	35.4	7.59	**5.23**	**4.48**	**4.13**
	✓	✗	21.6	8.92	6.59	5.73	5.27
	✓	✓	**20.2**	**7.41**	5.36	4.68	4.27
DDIM [258]	✗	✗	39.0	14.5	9.47	7.57	6.64
	✗	✓	39.8	11.1	7.17	5.83	5.19
	✓	✗	37.4	14.7	9.73	7.86	6.92
	✓	✓	30.0	10.7	7.14	5.93	5.35
S-PNDM [176]	✗	✗	57.9	15.2	10.0	8.12	7.20
	✗	✓	60.6	12.2	8.69	7.59	6.94
	✓	✗	39.0	13.7	9.75	8.08	7.22
	✓	✓	35.5	11.2	8.54	7.52	6.94
F-PNDM [176]	✗	✗	N/A	N/A	13.9	9.45	7.87
	✗	✓	N/A	N/A	14.5	9.45	8.05
	✓	✗	N/A	N/A	12.5	9.01	7.74
	✓	✓	N/A	N/A	12.3	9.26	7.86

3.4.6.7 Extended Qualitative Results

In this subsection, we show additional qualitative comparisons of DDIM and GENIE on LSUN Church-Outdoor (Figure 3.16), ImageNet (Figure 3.17), and Cats (upsampler conditioned on test set images) (Figure 3.18 and Figure 3.19). In all genie/figures, we can see that samples generated with GENIE generally exhibit finer details as well as sharper contrast and are less blurry compared to standard DDIM.

In Figure 3.20 and Figure 3.21, we show additional high-resolution images generated with the GENIE Cats upsampler using base model samples and test set samples, respectively.

3.4.6.8 Computational Resources

The total amount of compute used in this research project is roughly 163k GPU hours. We used an in-house GPU cluster of V100 NVIDIA GPUs.

Table 3.10: Unconditional LSUN Bedrooms generative performance (measured in FID). Methods above the middle line use the same score model checkpoint; Learned Sampler uses a different one. (†): numbers are taken from literature.

Method	AFS	Denoising	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	✗	74.1	17.1	13.3	11.6	11.1
	✗	✓	115	11.4	7.18	5.80	**5.35**
	✓	✗	55.9	18.4	14.1	12.3	11.6
	✓	✓	47.3	**9.29**	**6.83**	5.79	5.40
DDIM [258]	✗	✗	69.6	27.1	19.0	15.8	14.2
	✗	✓	81.0	16.3	9.18	7.12	6.20
	✓	✗	62.1	27.1	19.3	16.3	14.6
	✓	✓	42.5	12.5	8.21	6.77	6.05
S-PNDM [176]	✗	✗	70.4	22.1	15.7	13.5	12.4
	✗	✓	88.9	12.2	8.40	7.33	6.80
	✓	✗	48.0	20.2	15.2	13.4	12.4
	✓	✓	45.0	10.8	8.14	7.23	6.71
F-PNDM [176]	✗	✗	N/A	N/A	36.1	18.5	14.6
	✗	✓	N/A	N/A	26.8	9.85	7.86
	✓	✗	N/A	N/A	29.4	17.5	14.3
	✓	✓	N/A	N/A	18.9	9.27	7.69
Learned Sampler [286] (†)	✗	✓	**29.2**	11.0	-	**4.82**	-

Table 3.11: Unconditional LSUN Church-Outdoor generative performance (measured in FID). Methods above the middle line use the same score model checkpoint; Learned Sampler uses a different one. (†): numbers are taken from literature.

Method	AFS	Denoising	NFEs=5	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	✗	97.2	25.4	15.9	11.6	9.57
	✗	✓	147	13.7	11.7	8.52	7.28
	✓	✗	47.8	13.6	10.6	9.17	8.28
	✓	✓	60.3	**10.5**	**7.44**	**6.38**	**5.84**
DDIM [258]	✗	✗	81.5	28.5	16.7	11.9	9.9
	✗	✓	110	25.3	11.5	8.53	7.35
	✓	✗	44.0	17.4	12.5	10.2	9.07
	✓	✓	45.8	12.8	8.44	6.97	6.28
S-PNDM [176]	✗	✗	59.4	18.7	13.3	11.4	10.4
	✗	✓	87.5	14.8	9.54	7.98	7.21
	✓	✗	40.7	17.0	12.8	11.2	10.3
	✓	✓	48.8	12.9	9.10	7.82	7.12
F-PNDM [176]	✗	✗	N/A	N/A	15.5	12.0	10.6
	✗	✓	N/A	N/A	15.7	9.78	7.99
	✓	✗	N/A	N/A	15.2	11.8	10.4
	✓	✓	N/A	N/A	12.6	9.29	7.83
Learned Sampler [286] (†)	✗	✓	**30.2**	11.6	-	6.74	-

Table 3.12: Cats (base model) generative performance (measured in FID).

Method	AFS	NFEs=10	NFEs=15	NFEs=20	NFEs=25
GENIE (ours)	✗	**12.2**	**8.74**	**7.40**	6.84
	✓	13.3	9.07	7.76	**6.76**
DDIM [258]	✗	12.7	9.89	8.66	7.98
	✓	13.6	10.0	8.73	7.87
S-PNDM [176]	✗	12.8	11.6	10.8	10.4
	✓	12.5	11.3	10.7	10.2
F-PNDM [176]	✗	N/A	12.8	10.4	10.6
	✓	N/A	11.8	10.4	10.3

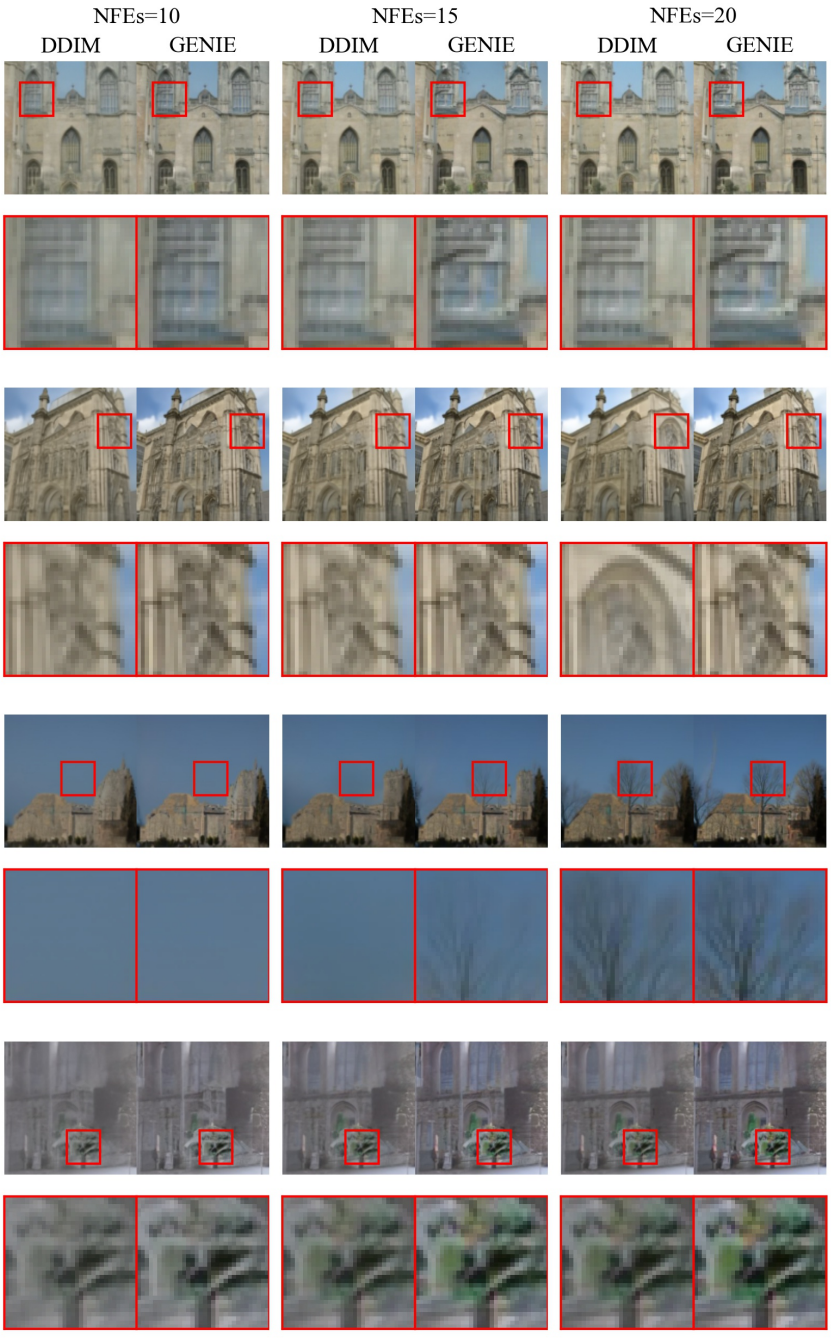


Figure 3.16: Additional samples on LSUN Church-Outdoor with zoom-in on details. GENIE often results in sharper and higher contrast samples compared to DDIM.

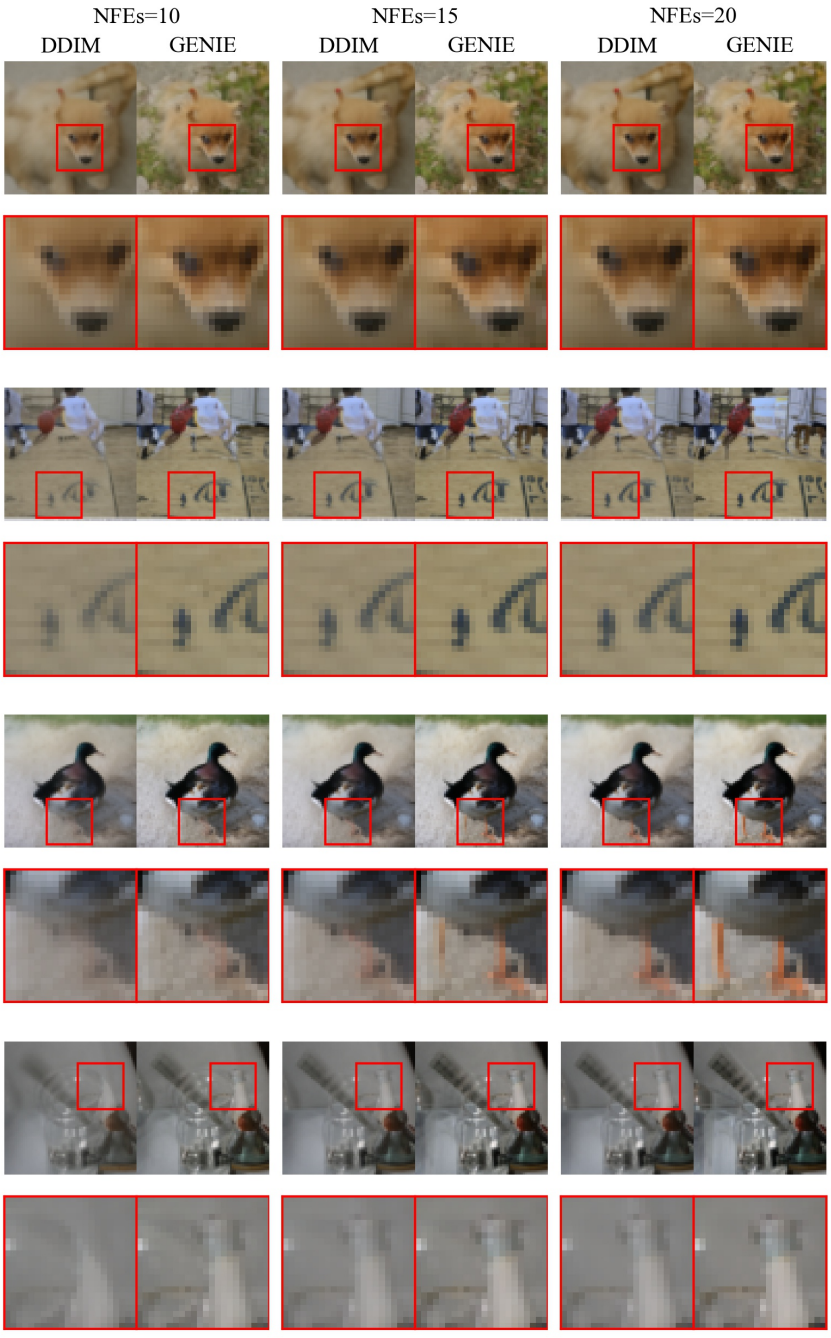


Figure 3.17: Additional samples on ImageNet with zoom-in on details. GENIE often results in sharper and higher contrast samples compared to DDIM.

Figure 3.18: Additional samples on Cats with zoom-in on details. GENIE often results in sharper and higher contrast samples compared to DDIM.

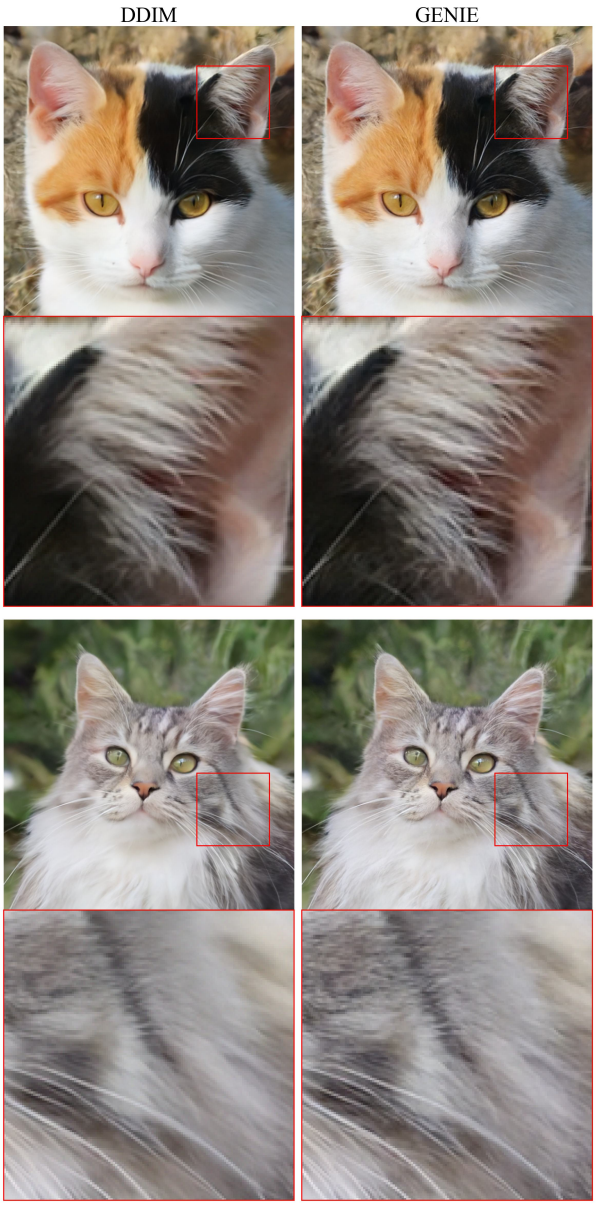


Figure 3.19: Additional samples on Cats with zoom-in on details. GENIE often results in sharper and higher contrast samples compared to DDIM.

Figure 3.20: End-to-end samples on Cats. The GENIE base model uses 25 function evaluations and the GENIE upsampler only uses five function evaluations. An upsampler evaluation is roughly four times as expensive as a base model evaluation.

Figure 3.21: Upsampling 128×128 test set images using the GENIE upsampler with only five function evaluations.

Table 3.13: Cats (upsampler) generative performance (measured in FID).

Method	AFS	NFEs=5	NFEs=10	NFEs=15
GENIE (ours)	✗	7.03	4.93	**4.83**
	✓	**5.53**	**4.90**	4.91
DDIM [258]	✗	11.3	7.16	5.99
	✓	9.47	6.64	5.85
S-PNDM [176]	✗	16.7	12.1	8.83
	✓	14.6	11.0	9.01
F-PNDM [176]	✗	N/A	N/A	12.9
	✓	N/A	N/A	11.7

3.4.7 Miscellaneous

3.4.7.1 Connection to Bao et al. [19]

The concurrent Bao et al. [19] learn covariance matrices for diffusion model sampling using prediction heads somewhat similar to the ones in GENIE. Specifically, both Bao et al. [19] and GENIE use small prediction heads that operate on top of the large first-order score predictor. However, we would like to stress multiple differences: (i) Bao et al. [19] learn the DDM's sampling covariance matrices, while we learn higher-order ODE gradients. More generally, Bao et al. [19] rely on stochastic diffusion model sampling, while we use the ODE formulation. (ii) Most importantly, in our case we can resort to directly learning the low-dimensional JVPs without low-rank or diagonal matrix approximations or other assumptions. Similar techniques are not directly applicable in Bao et al. [19]'s setting. In detail, this is because in their case the relevant matrices (obtained after Cholesky or another applicable decomposition of the covariance) do not act on regular vectors but random noise variables. In other words, instead of using a deterministic JVP predictor (which takes $\mathbf{x}_t$ and t as inputs), as in GENIE, Bao et al. [19] would require to model an entire distribution for each $\mathbf{x}_t$ and t without explicitly forming high-dimensional Cholesky decomposition-based matrices, if they wanted to do something somewhat analogous to GENIE's novel JVP-based approach. As a consequence, Bao et al. [19] take another route to keeping the dimensionality of the additional network outputs manageable in practice. In particular, they resort to assuming a diagonal covariance matrix in their experiments. By directly learning JVPs, we never have to rely on such potentially limiting assumptions. (iii) Experimentally, Bao et al. [19] also consider fast sampling with few neural network calls.

However, GENIE generally outperforms them (see, for example, their CIFAR10 results in their Table 2 for 10 and 25 NFE). This might indeed be due to the assumptions made by Bao et al. [19], which we avoid. Furthermore, their stochastic vs. our deterministic sampling may play a role, too.

3.4.7.2 Combining GENIE with Progressive Distillation

We speculate that GENIE could potentially be combined with Progressive Distillation [244]: In every distillation stage of [244], one could quickly train a small GENIE prediction head to model higher-order ODE gradients. This would then allow for larger and/or more accurate steps, whose results represent the distillation target (teacher) in the progressive distillation protocol. This may also reduce the number of required distillation stages. Overall, this could potentially speed up the cumbersome stage-wise distillation and maybe also lead to an accuracy and performance improvement. In particular, we could replace the DDIM predictions in Algorithm 2 of [244] with improved GENIE predictions.

Note that this approach would not be possible with multistep methods as proposed by Liu et al. [176]. Such techniques could not be used here, because they require the history of previous predictions, which are not available in the progressive distillation training scheme.

We leave exploration of this direction to future work.

3.5 Epilogue

3.5.1 Additional Related Work

Since the publication of Dockhorn et al. [74], there has been a plethora of work on accelerated DM solvers [61, 141, 169, 183, 184, 291]. Furthermore, there have been considerable efforts to distill the *entire* iterative sampling process of a DM into a neural network. In *progressive distillation* [244] an Euler-like sampler with N steps is distilled into an Euler-like sampler with $N/2$ steps. This procedure can be performed iteratively, bringing the number of evaluations for sampling as low as one. While progressive distillation was proposed around the same time as GENIE, it originally came with a set of limitations (see the paper). Recently, progressive distillation has been extended to classifier-free guidance [196] and to ODE solvers beyond Euler's method [24]. Other distillation methods that distill the entire iterative sampling process of a DMs have been proposed as well [180, 186, 265, 318].

Chapter 4

Differentially Private Diffusion Models

4.1 Differentially Private Generative Modeling: What and Why?

As we have seen in the last chapters, DMs have the ability to learn complex data distributions, such as those found in natural images. However, as with any (generative) model, training DMs on sensitive data raises concerns about privacy. For example, an adversary may be able to recover training images of deep classifiers using gradients of the networks [303] or reproduce training text sequences from large transformers [39]. Generative models may even overfit directly, generating data indistinguishable from the data they have been trained on. In fact, overfitting and privacy-leakage of generative models are more relevant than ever, considering recent works that train powerful photo-realistic image generators on large-scale web-scraped data. Recently, it has been shown that large text-to-image DMs may almost perfectly *memorize* (portions of the) training data [40]. Memorization of training data is particularly problematic in privacy-sensitive domains, such as medical imaging. One potential method to provably avoid memorization in generative modeling is to train models with strict *differential privacy* (DP) [83, 84] guarantees.

DP is a rigorous mathematical definition of privacy applied to statistical queries; in machine learning these queries generally correspond to the training of a neural network using sensitive training data. Informally, training is said to be differentially private, if, given the trained weights of the network, an adversary cannot tell with certainty whether a particular

data point was part of the training data. A popular algorithm for DP training of neural networks is *differentially private stochastic gradient descent* (DP-SGD) [1]. DP-SGD preserves privacy by clipping and noising the parameter gradients during training. This leads to an inevitable trade-off between privacy and utility; for instance, small clipping constants and large noise injection result in very private models that may be of little practical use. In the past, DP-SGD has, for example, been employed to train generative adversarial networks [88, 273, 296], which are particularly susceptible to privacy-leakage [288].

4.2 Preface

This section presents the paper "Differentially Private Diffusion Models". In this work, we train DMs with strict DP guarantees using DP-SGD [1]. We motivate why we believe DMs to be better suited for DP-SGD training than one-shot generation models such as generative adversarial networks. We study the DM parameterization, training settings and sampling methods in detail, and optimize them for the DP setup. We propose *noise multiplicity*, a modification of DP-SGD tailored to the training of DMs. Experimentally, we significantly surpass the state-of-the-art in DP synthesis on widely-studied image modeling benchmarks. The paper was initially put on arXiv in October 2022, and then submitted to the journal *Transactions on Machine Learning Research* in April 2023.

Contributions: I was the sole first author of this work. Tianshi Cao, Arash Vahdat and Karsten Kreis were co-authors, and Karsten supervised the project. I implemented all code and ran all experiments, and wrote the majority of the paper. I proposed "noise multiplicity", the main modification of DP-SGD used to train DPDMs.

Reproducibility: The code and models for this work have been open-sourced. See `https://github.com/nv-tlabs/DPDM` for instructions to reproduce our results.

Abstract: While modern machine learning models rely on increasingly large training datasets, data is often limited in privacy-sensitive domains. Generative models trained with differential privacy (DP) on sensitive data can sidestep this challenge, providing access to synthetic data instead. We build on the recent success of diffusion models (DMs) and introduce *Differentially Private Diffusion Models* (DPDMs), which enforce privacy using differentially private stochastic gradient descent (DP-SGD). We investigate the DM parameterization and the sampling algorithm, which turn out to be crucial ingredients in DPDMs, and propose *noise multiplicity*, a powerful modification of DP-SGD tailored to the training

of DMs. We validate our novel DPDMs on image generation benchmarks and achieve state-of-the-art performance in all experiments. Moreover, on standard benchmarks, classifiers trained on DPDM-generated synthetic data perform on par with task-specific DP-SGD-trained classifiers, which has not been demonstrated before for DP generative models. Project page and code: https://nv-tlabs.github.io/DPDM.

4.3 Main Paper

4.3.1 Introduction

Modern deep learning usually requires significant amounts of training data. However, sourcing large datasets in privacy-sensitive domains is often difficult. To circumvent this challenge, generative models trained on sensitive data can provide access to large synthetic data instead, which can be used flexibly to train downstream models. Unfortunately, typical overparameterized neural networks have been shown to provide little to no privacy to the data they have been trained on. For example, an adversary may be able to recover training images of deep classifiers using gradients of the networks [303] or reproduce training text sequences from large transformers [39]. Generative models may even overfit directly, generating data indistinguishable from the data they have been trained on. In fact, overfitting and privacy-leakage of generative models are more relevant than ever, considering recent works that train powerful photo-realistic image generators on large-scale Internet-scraped data [16, 230, 235, 241].

To protect the privacy of training data, one may train their model using differential privacy (DP). DP is a rigorous privacy framework that applies to statistical queries [83, 84]. In our case, this query corresponds to the training of a neural network using sensitive data. Differentially private stochastic gradient descent (DP-SGD) [1] is the workhorse of DP training of neural networks. It preserves privacy by clipping and noising the parameter gradients during training. This leads to an inevitable trade-off between privacy and utility; for instance, small clipping constants and large noise injection result in very private models that may be of little practical use.

DP-SGD has, for example, been employed to train generative adversarial networks (GANs) [88, 273, 296], which are particularly susceptible to privacy-leakage [288]. However, while GANs in the non-private setting can synthesize photo-realistic images [30, 138–140], their application in the private setting is challenging. GANs are difficult to optimize [11, 198] and prone to mode collapse; both phenomena may be amplified during DP-SGD training.

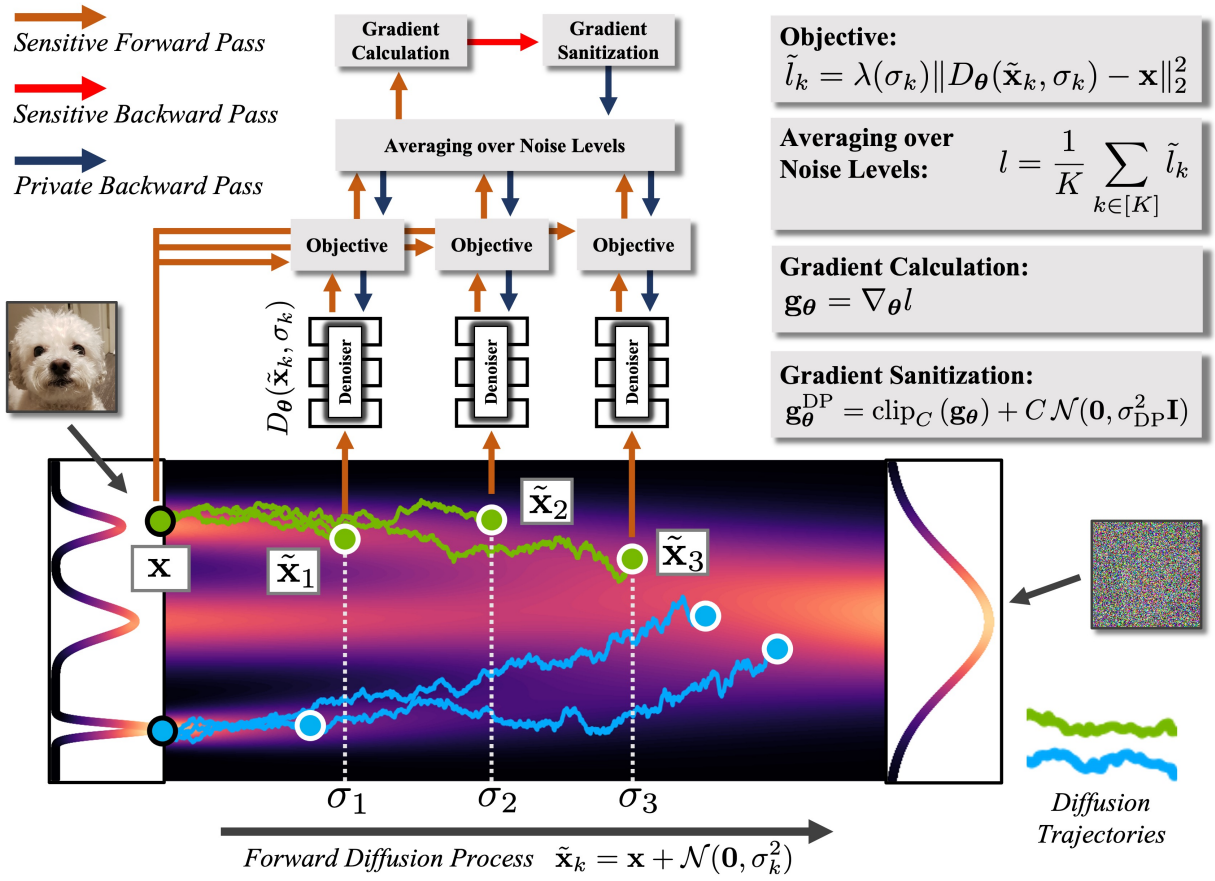


Figure 4.1: Information flow during training in our *Differentially Private Diffusion Model* (DPDM) for a single training sample in **green** (*i.e.* batchsize B=1, another sample shown in **blue**). We rely on DP-SGD to guarantee privacy and use *noise multiplicity*; here, K=3. The diffusion is visualized for a one-dim. toy distribution (marginal probabilities in **purple**); our main experiments use high-dim. images. Note that for brevity in the visualization we dropped the index i, which indicates the minibatch element in Equations (4.6) and (4.7).

Recently, Diffusion Models (DMs) have emerged as a powerful class of generative models [109, 257, 263], demonstrating outstanding performance in image synthesis [66, 110, 212, 230, 235, 241]. In DMs, a diffusion process gradually perturbs the data towards random noise, while a deep neural network learns to denoise. DMs stand out not only by high synthesis quality, but also sample diversity, and a simple and robust training objective. This makes them arguably well suited for training under DP perturbations. Moreover, generation in DMs corresponds to an iterative denoising process, breaking the difficult generation task into many small denoising steps that are individually simpler than the one-shot synthesis task performed by GANs and other traditional methods. In particular, the denoising neural network that is learnt in DMs and applied repeatedly at each synthesis step is less complex and smoother than the generator networks of one-shot methods, as we validate in experiments on toy data (synthetically generated mixture of 2D Gaussians). Therefore, training of the denoising neural network is arguably less sensitive to gradient clipping and noise injection required for DP.

Based on these observations, we propose *Differentially Private Diffusion Models* (DPDMs), DMs trained with rigorous DP guarantees based on DP-SGD. We thoroughly study the DM parameterization and sampling algorithm, and tailor them to the DP setting. We find that the stochasticity in DM sampling, which is empirically known to be error-correcting [141], can be particularly helpful in DP-SGD training to obtain satisfactory perceptual quality. We also propose *noise multiplicity*, where a single training data sample is re-used for training at multiple perturbation levels along the diffusion process (see Figure 4.1). This simple yet powerful modification of the DM training objective improves learning at no additional privacy cost. We validate DPDMs on standard DP image generation tasks, and achieve state-of-the-art performance by large margins, both in terms of perceptual quality and performance of downstream classifiers trained on synthetically generated data from our models. For example, on MNIST we improve the state-of-the-art FID from 56.2 to 23.4 and downstream classification accuracy from 81.5% to 95.3% for the privacy setting DP-(ε=1, δ=10^{-5}). We also find that classifiers trained on DPDM-generated synthetic data perform on par with task-specific DP-classifiers trained on real data, which has not been demonstrated before for DP generative models.

We make the following contributions: **(i)** We carefully motivate training DMs with DP-SGD and introduce DPDMs, the first DMs trained under DP guarantees. **(ii)** We study DPDM parameterization, training setting and sampling in detail, and optimize it for the DP setup. **(iii)** We propose *noise multiplicity* to efficiently boost DPDM performance. **(iv)** Experimentally, we significantly surpass the state-of-the-art in DP synthesis on widely-studied image modeling benchmarks. **(v)** We demonstrate for the first time that classifiers

trained on DPDM-generated data perform on par with task-specific DP-trained discriminative models. This implies a very high utility of the synthetic data generated by DPDMs, delivering on the promise of DP generative models as an effective data sharing medium. Finally, we hope that our work has implications for the literature on DMs, which are now routinely trained on ultra large-scale datasets of diverse origins.

4.3.2 Background

4.3.2.1 Diffusion Models

We consider continuous-time DMs [263] and follow the presentation of Karras et al. [141]. Let $p_{\text{data}}(\mathbf{x})$ denote the data distribution and $p(\mathbf{x};\sigma)$ the distribution obtained by adding i.i.d. σ^2-variance Gaussian noise to the data distribution. For sufficiently large $\sigma_{\max}$, $p(\mathbf{x};\sigma^2_{\max})$ is almost indistinguishable from $\sigma^2_{\max}$-variance Gaussian noise. Capitalizing on this observation, DMs sample (high variance) Gaussian noise $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \sigma^2_{\max})$ and sequentially denoise $\mathbf{x}_0$ into $\mathbf{x}_i \sim p(\mathbf{x}_i;\sigma_i)$, $i \in [0, ..., M]$, with $\sigma_i < \sigma_{i-1}$ ($\sigma_0 = \sigma_{\max}$). If $\sigma_M = 0$, then $\mathbf{x}_0$ is distributed according to the data.

Sampling. In practice, the sequential denoising is often implemented through the simulation of the *Probability Flow* ordinary differential equation (ODE) [263]

$$d\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt, \tag{4.1}$$

where $\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma)$ is the *score function* [123]. The schedule $\sigma(t) : [0,1] \to \mathbb{R}_+$ is user-specified and $\dot{\sigma}(t)$ denotes the time derivative of $\sigma(t)$. Alternatively, we may also sample from a stochastic differential equation (SDE) [141, 263]:

$$d\mathbf{x} = \underbrace{-\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt}_{\text{Probability Flow ODE; see Equation (4.1)}} \underbrace{-\beta(t)\sigma^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt + \sqrt{2\beta(t)}\sigma(t)\, d\omega_t}_{\text{Langevin diffusion component}}, \tag{4.2}$$

where $d\omega_t$ is the standard Wiener process. In principle, given initial samples $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \sigma^2_{\max})$, simulating either Probability Flow ODE or SDE produces samples from the same distribution. In practice, though, neither ODE nor SDE can be simulated exactly: Firstly, any numerical solver inevitably introduces discretization errors. Secondly, the score function is only accessible through a model $s_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$ that needs to be learned; replacing the score function with an imperfect model also introduces an error. Empirically, the ODE formulation has been used frequently to develop fast solvers [74, 176, 183, 258, 313],

whereas the SDE formulation often leads to higher quality samples (while requiring more steps) [141]. One possible explanation for the latter observation is that the Langevin diffusion component in the SDE at any time during the synthesis process actively drives the process towards the desired marginal distribution $p(\mathbf{x};\sigma)$, whereas errors accumulate in the ODE formulation, even when using many synthesis steps. In fact, it has been shown that as the score model $s_{\boldsymbol{\theta}}$ improves, the performance boost that can be obtained by an SDE solver diminishes [141]. Finally, note that we are using classifier-free guidance [108] to perform class-conditional sampling in this work. For details on classifier-free guidance and the numerical solvers for Equation (4.1) and Equation (4.2), we refer to Section 4.4.3.3.

Training. DM training reduces to learning the score model $s_{\boldsymbol{\theta}}$. The model can, for example, be parameterized as $\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma) \approx s_{\boldsymbol{\theta}} = (D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - \mathbf{x})/\sigma^2$ [141], where $D_{\boldsymbol{\theta}}$ is a learnable *denoiser* that, given a noisy data point $\mathbf{x} + \mathbf{n}$, $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$, $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2)$ and conditioned on the noise level σ, tries to predict the clean $\mathbf{x}$. The denoiser $D_{\boldsymbol{\theta}}$ can be trained by minimizing an L_2-loss

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x}),(\sigma,\mathbf{n})\sim p(\sigma,\mathbf{n})} \left[\lambda_\sigma \| D_{\boldsymbol{\theta}}(\mathbf{x} + \mathbf{n}, \sigma) - \mathbf{x} \|_2^2 \right], \tag{4.3}$$

where $p(\sigma, \mathbf{n}) = p(\sigma)\mathcal{N}(\mathbf{n}; \mathbf{0}, \sigma^2)$ and $\lambda_\sigma \colon \mathbb{R}_+ \to \mathbb{R}_+$ is a weighting function. Previous works proposed various denoiser models $D_{\boldsymbol{\theta}}$, noise distributions $p(\sigma)$, and weightings λ_σ. We refer to the triplet $(D_{\boldsymbol{\theta}}, p, \lambda)$ as the DM *config*. Here, we consider four such configs: *variance preserving* (VP) [263], *variance exploding* (VE) [263], **v**-prediction [244], and EDM Karras et al. [141]; Section 4.4.3.1 for details.

4.3.2.2 Differential Privacy

DP is a rigorous mathematical definition of privacy applied to statistical queries; in our work the queries correspond to the training of a neural network using sensitive training data. Informally, training is said to be DP, if, given the trained weights $\boldsymbol{\theta}$ of the network, an adversary cannot tell with certainty whether a particular data point was part of the training data. This degree of certainty is controlled by two positive parameters ε and δ: training becomes more private as ε and δ decrease. Note, however, that there is an inherent trade-off between utility and privacy: very private models may be of little to no practical use. To guarantee a sufficient amount of privacy, as a rule of thumb, δ should not be larger than $1/N$, where N is number of training points $\{\mathbf{x}_i\}_{i=1}^N$, and ε should be a small constant. More formally, we refer to (ε, δ)-DP defined as follows [83]:

Definition 4.3.1. (Differential Privacy) A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ε, δ)-DP if for any two datasets $d, d' \in \mathcal{D}$ differing by at

most one entry, and for any subset of outputs $S \subseteq \mathcal{R}$ it holds that

$$\mathbf{Pr}\left[\mathcal{M}(d) \in S\right] \leq e^{\varepsilon}\mathbf{Pr}\left[\mathcal{M}(d') \in S\right] + \delta. \tag{4.4}$$

DP-SGD. We require a DP algorithm that trains a neural network using sensitive data. The workhorse for this particular task is differentially private stochastic gradient descent (DP-SGD) [1]. DP-SGD is a modification of SGD for which per-sample-gradients are clipped and noise is added to the clipped gradients; the DP-SGD parameter updates are defined as follows

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{B}\left(\sum\nolimits_{i\in\mathbb{B}} \texttt{clip}_C\left(\nabla_{\boldsymbol{\theta}} l_i(\boldsymbol{\theta})\right) + C\mathbf{z}\right), \tag{4.5}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{\text{DP}}\boldsymbol{I})$, $\mathbb{B}$ is a B-sized subset of $\{1, \ldots, N\}$ drawn uniformly at random, l_i is the loss function for data point $\mathbf{x}_i$, η is the learning rate, and the clipping function is $\texttt{clip}_C(\mathbf{g}) = \min\left\{1, C/\|\mathbf{g}\|_2\right\}\mathbf{g}$. DP-SGD can be adapted to other first-order optimizers, such as Adam [194].

Privacy Accounting. According to the *Gaussian mechanism* [84], a single DP-SGD update (Equation (4.5)) satisfies (ε, δ)-DP if $\sigma^2_{\text{DP}} > 2\log(1.25/\delta)C^2/\varepsilon^2$. Privacy accounting methods can be used to *compose* the privacy cost of multiple DP-SGD training updates and to determine the variance σ^2_{DP} needed to satisfy (ε, δ)-DP for a particular number of DP-SGD updates with clipping constant C and subsampling rate B/N. Also see Section 4.4.1.

4.3.3 Differentially Private Diffusion Models

We propose DPDMs, DMs trained with rigorous DP guarantees based on DP-SGD [1]. DP-SGD is a well-established method to train DP neural networks and our intention is not to re-invent DP-SGD; instead, the novelty in this work lies in the combination of DMs with DP-SGD, modifications of DP-SGD specifically tailored to the DP training of DMs, as well as the study of design choices and training recipes that greatly influence the performance of DPDMs. Combinations of DP-SGD with GANs have been widely studied [26, 88, 273, 296], motivating a similar line of research for DMs. To the best of our knowledge, we are the first to explore DP training of DMs. In Section 4.3.3.1, we discuss the motivation for using DMs for DP generative modeling. In Section 4.3.3.2, we then discuss training and methodological details as well as DM design choices, and we prove that DPDMs satisfy DP.

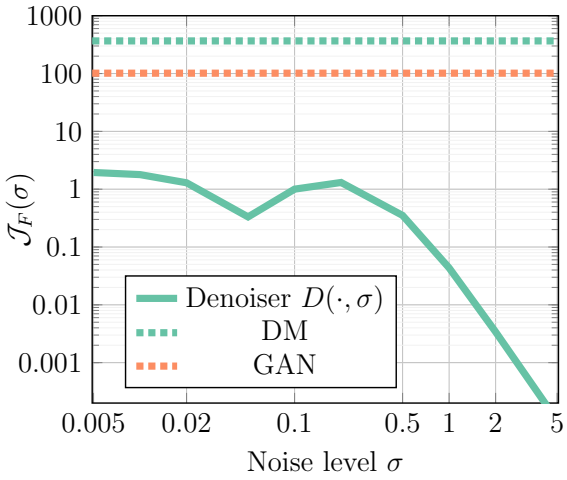


Figure 4.2: Frobenius norm of the Jacobian $\mathcal{J}_F(\sigma)$ of the denoiser $D(\cdot, \sigma)$ and constant Frobenius norms of the Jacobians $\mathcal{J}_F$ of the sampling functions defined by the DM and a GAN. Section 4.4.5 for experiment details.

4.3.3.1 Motivation

(i) Objective function. GANs have so far been the workhorse of DP generative modeling (see Section 4.3.4), even though they are generally difficult to optimize [11, 198] due to their adversarial training and propensity to mode collapse. Both phenomena may be amplified during DP-SGD training. DMs, in contrast, have been shown to produce outputs as good or even better than GANs' [66], while being trained with a very simple regression-like L_2-loss (Equation (4.3)), which makes them robust and scalable in practice. DMs are therefore arguably also well-suited for DP-SGD-based training and offer better stability under gradient clipping and noising than adversarial training frameworks.

(ii) Sequential denoising. In GANs and most other traditional generative modeling approaches, the generator directly learns the sampling function, i.e., the mapping of latents to synthesized samples, end-to-end. In contrast, the sampling function in DMs is defined through a sequential denoising process, breaking the difficult generation task into many small denoising steps which are individually less complex than the one-shot synthesis task performed by, for instance, a GAN generator. The denoiser neural network, the learnable component in DMs that is evaluated once per denoising step, is therefore simpler and smoother than the one-shot generator networks of other methods. We fit both a DM and a GAN to a two-dimensional toy distribution (mixture of Gaussians, see Section 4.4.5) and empirically verify that the denoiser D is indeed significantly less complex

(quantified by the Frobenius norm of the Jacobian) than the generator learnt by the GAN and also than the end-to-end multi-step synthesis process (Probability Flow ODE) of the DM (see Figure 4.2; we calculate denoiser $\mathcal{J}_F(\sigma)$ at varying noise levels σ). Generally, more complex functions require larger neural networks and are more difficult to learn. In DP-SGD training we only have a limited number of training iterations available until the privacy budget is depleted. Consequently, the fact that DMs require less complexity out of their neural networks than typical one-shot generation methods, while still being able to represent expressive generative models due to the iterative synthesis process, makes them likely well-suited for DP generative modeling with DP-SGD.

(iii) Stochastic diffusion model sampling. As discussed in Section 4.3.2.1, generating samples from DMs with stochastic sampling can perform better than deterministic sampling when the score model is not learned well. Since we replace gradient estimates in DP-SGD training with biased large variance estimators, we cannot expect a perfectly accurate score model. In Section 4.3.5.2, we empirically show that stochastic sampling can in fact boost perceptual synthesis quality in DPDMs as measured by FID.

4.3.3.2 Training Details, Design Choices, Privacy

The clipping and noising of the gradient estimates in DP-SGD (Equation (4.5)) pose a major challenge for efficient optimization. Blindly reducing the added noise could be fatal, as it decreases the number of training iterations allowed within a certain (ε, δ)-DP budget. Furthermore, as discussed the L_2-norm of the noise added in DP-SGD scales linearly to the number of parameters. Consequently, settings that work well for non-private DMs, such as relatively small batch sizes, a large number of training iterations, and heavily overparameterized models, may not work well for DPDMs. Below, we discuss how we propose to adjust DPDMs for successful DP-SGD training.

Noise multiplicity. Recall that the DM objective in Equation (4.3) involves three expectations. As usual, the expectation with respect to the data distribution $p_{\text{data}}(\mathbf{x})$ is approximated using mini-batching. For non-private DMs, the expectations over σ and $\mathbf{n}$ are generally approximated using a single Monte Carlo sample $(\sigma_i, \mathbf{n}_i) \sim p(\sigma)\mathcal{N}(\mathbf{0}, \sigma^2)$ per data point $\mathbf{x}_i$, resulting in the loss for training sample i

$$l_i = \lambda(\sigma_i)\|D_{\boldsymbol{\theta}}(\mathbf{x}_i + \mathbf{n}_i, \sigma_i) - \mathbf{x}_i\|_2^2. \tag{4.6}$$

The estimator l_i is very noisy in practice. Non-private DMs counteract this by training for a large number of iterations in combination with an exponential moving average (EMA) of the trainable parameters $\boldsymbol{\theta}$ [260]. When training DMs with DP-SGD, we incur a privacy cost for each iteration, and therefore prefer a small number of iterations. Furthermore, since the per-example gradient clipping as well as the noise injection induce additional variance, we would like our objective function to be less noisy than in the non-DP case. We achieve this by estimating the expectation over σ and $\mathbf{n}$ using an average over K noise samples, $\{(\sigma_{ik}, \mathbf{n}_{ik})\}_{k=1}^{K} \sim p(\sigma)\mathcal{N}(\mathbf{0}, \sigma^2)$ for each data point $\mathbf{x}_i$, replacing the non-private DM objective l_i in Equation (4.6) with

$$\tilde{l}_i = \frac{1}{K} \sum_{k=1}^{K} \lambda(\sigma_{ik}) \| D_{\boldsymbol{\theta}}(\mathbf{x}_i + \mathbf{n}_{ik}, \sigma_{ik}) - \mathbf{x}_i \|_2^2. \tag{4.7}$$

Importantly, we show that this modification comes at *no* additional privacy cost (also see Section 4.4.1). We call this simple yet powerful modification of the DM objective, which is tailored to the DP setup, *noise multiplicity*.

Theorem 1. *The variance of the DM objective (Equation (4.7)) decreases with increased noise multiplicity K as $1/K$.*

Proof in Section 4.4.4. Intuitively, the key is that we first create a relatively accurate low-variance gradient estimate by averaging over multiple noise samples before performing gradient sanitization in the backward pass via clipping and noising. This averaging process increases computational cost, but provides better utility at the same privacy budget, which is the main bottleneck in DP generative modeling; see Section 4.4.4.3 for further discussion.

We empirically showcase in Section 4.3.5.2 that the variance reduction induced by noise multiplicity is a key factor in training strong DPDMs. In Figure 4.3, we show that the reduction of variance in the DM objective also empirically leads to lower variance gradient estimates (see Section 4.4.4 for experiment details). The noise multiplicity mechanism is also highlighted in Figure 4.1: the figure describes the information flow during training for a single training sample (i.e., batch size $B = 1$). Note that noise multiplicity is loosely inspired by *augmentation multiplicity* [64], a technique where multiple augmentations per image are used to train *classifiers* with DP-SGD. In contrast to augmentation multiplicity, our novel noise multiplicity is carefully designed specifically for DPDMs and comes with theoretical proofs on its variance reduction. The reader may find a more detailed discussion on the difference between noise multiplicity and data multiplicity (for DPDMs) in Section 4.4.4.4.

Neural networks sizes. Current DMs are heavily overparameterized: For example, the

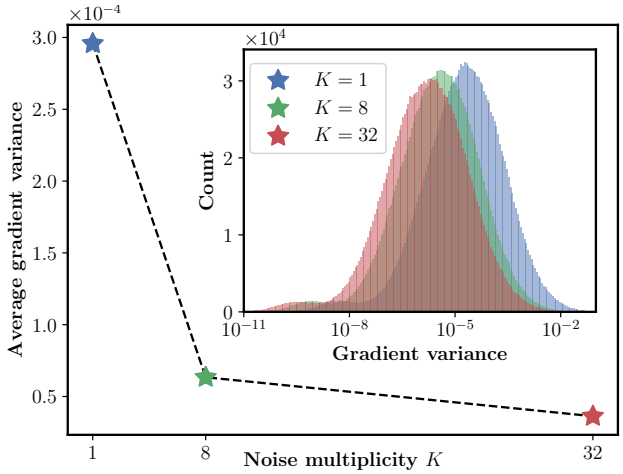


Figure 4.3: Increasing K in *noise multiplicity* leads to significant variance reduction of parameter gradient estimates during training (note logarithmic axis in inset). Enlarged version in Figure 4.6.

current state-of-the-art image generation model (in terms of perceptual quality) on CIFAR-10 uses more than 100M parameters, despite the dataset consisting of only 50k training points [141]. The per-example clipping operation of DP-SGD requires the computation of the loss gradient on each training example $\nabla_{\boldsymbol{\theta}}\tilde{l}_i$, rather than the minibatch gradient. In theory, this increases the memory footprint by at least $O(B)$; however, in common DP frameworks, such as Opacus [306], which we use, the peak memory requirement is $\mathcal{O}(B^2)$ compared to non-private training (recent methods such as *ghost clipping* [32] require less memory, but are not widely implemented) On top of that, DP-SGD generally already relies on a significantly increased batch size, when compared to non-private training, to improve the privacy-utility trade-off. As a result, we train very small neural networks for DPDMs, when compared to their non-DP counterparts: our models on MNIST/Fashion-MNIST and CelebA have 1.75M and 1.80M parameters, respectively. Furthermore, we found smaller models to perform better across our experiments which may be due to the L_2-norm of the noise added in our DP-SGD update scaling linearly with the number of parameters. This is in contrast to recent works in supervised DP learning, which show that larger models may perform better than smaller models [9, 64, 171, 172].

Diffusion model config. In addition to network size, we found the choice of DM config, i.e., denoiser parameterization $D_{\boldsymbol{\theta}}$, weighting function $\lambda(\sigma)$, and noise distribution $p(\sigma)$, to be important. In particular the latter is crucial to obtain strong results with DPDMs. In Figure 4.4, we visualize the noise distributions of the four configs under consideration. We follow Karras et al. [141] and plot the distribution $p(\log \sigma)$ over the log-noise level.

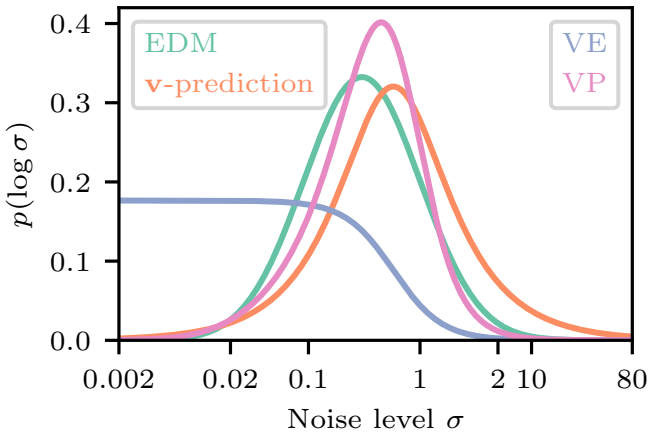


Figure 4.4: Noise level sampling for four DM configs: VP and VE [263], **v**-prediction [244], and EDM [141]; see Section 4.4.3.1.

Especially for high privacy settings (small ε), we found it important to use distributions that give sufficiently much weight to larger σ, such as the distribution of **v**-prediction [244]. It is known that at large σ the DM learns the global, coarse structure of the data, i.e., the low frequency content in the data (images, in our case). Learning global structure reasonably well is crucial to form visually coherent images that can also be used to train downstream models. This is relatively easy to achieve in the non-DP setting, due to the heavily smoothed diffused distribution at these high noise level. At high privacy levels, however, even training at such high noise levels can be challenging due to DP-SGD's gradient clipping and noising. We hypothesize that this is why it is beneficial to give relatively more weight to high noise levels when training in the DP setting. In Section 4.3.5.2, we empirically demonstrate the importance of the right choice of the DM config.

DP-SGD settings. Following De et al. [64] we use very large batch sizes: 4096 on MNIST/Fashion-MNIST and 2048 on CelebA. Similar to previous works [64, 161, 172], we found that small clipping constants C work better than larger clipping norms; in particular, we found $C = 1$ to work well across our experiments. Decreasing C even further had little effect; in contrast, increasing C significantly worsened performance. Similar to non-private DMs, we use an EMA of the learnable parameters $\boldsymbol{\theta}$. Incidentally, this has recently been reported to also have a positive effect on DP-SGD training of classifiers by De et al. [64].

Privacy. We formulate privacy protection under the Rényi Differential Privacy (RDP) [199] framework (see Definition 4.4.1), which can be converted to (ϵ, δ)-DP. For an algorithm for

Algorithm 4 DPDM Training

Input: Private data set $d = \{\mathbf{x}_j\}_{j=1}^{N}$, subsampling rate B/N, DP noise scale σ_{DP}, clipping constant C, sampling function *Poisson Sample* (Algorithm 5), denoiser $D_{\boldsymbol{\theta}}$ with initial parameters $\boldsymbol{\theta}$, noise distribution $p(\sigma)$, learning rate η, total steps T, noise multiplicity K, *Adam* [148] optimizer

Output: Trained parameters $\boldsymbol{\theta}$

for $t = 1$ **to** T **do**

  $\mathbb{B} \sim$ *Poisson Sample*$(N, B/N)$

  for $i \in \mathbb{B}$ **do**

    $\{(\sigma_{ik}, \mathrm{n}_{ik})\}_{k=1}^{K} \sim p(\sigma)\mathcal{N}(\mathbf{0}, \sigma^2)$

    $\tilde{l}_i = \frac{1}{K}\sum_{k=1}^{K} \lambda(\sigma_{ik}) \| D_{\boldsymbol{\theta}}(\mathbf{x}_i + \mathbf{n}_{ik}, \sigma_{ik}) - \mathbf{x}_i \|_2^2$

  end for

  $G_{batch} = \frac{1}{B}\sum_{i \in \mathbb{B}} \texttt{clip}_C \left(\nabla_{\boldsymbol{\theta}} \tilde{l}_i \right)$

  $\tilde{G}_{batch} = G_{batch} + (C/B)\mathbf{z}, \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{DP}}^2)$

  $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta * Adam(\tilde{G}_{batch})$

end for

DPDM training with noise multiplicity see Algorithm 4. For the sake of completeness we also formally prove the DP of DPDMs (DP of releasing sanitized training gradients $\tilde{G}_{batch}$):

Theorem 2. *For noise magnitude σ_{DP}, releasing $\tilde{G}_{batch}$ in Algorithm 4 satisfies $(\alpha, \alpha/2\sigma_{\text{DP}}^2)$-RDP.*

The proof can be found in Section 4.4.1. Note that the strength of DP protection is independent of the noise multiplicity, as discussed above. In practice, we construct mini-batches by *Poisson Sampling* (See Algorithm 5) the training dataset for privacy amplification via sub-sampling [200], and compute the overall privacy cost of training DPDM via RDP composition [199]. Tighter privacy bounds, such as the one developed in Gopi et al. [98], may lead to better results but are not widely implemented (not in Opacus [306], the DP-SGD library we use).

4.3.4 Related Work

In the DP generative learning literature, several works [45, 88, 273, 296] have explored applying DP-SGD [1] to GANs, while others [182, 284, 304] train GANs under the PATE [220] framework, which distills private teacher models (discriminators) into a public student (generator) model. Apart from GANs, Acs et al. [2] train variational autoencoders on

DP-sanitized data clusters, and Cao et al. [38] use the Sinkhorn divergence and DP-SGD.

DP-MERF [102] was the first work to perform one-shot privatization on the data, followed by non-private learning. It uses differentially private random Fourier features to construct a Maximum Mean Discrepancy loss, which is then minimized by a generative model. PEARL [173] instead minimizes an empirical characteristic function, also based on Fourier features. DP-MEPF [103] extends DP-MERF to the mixed public-private setting with pre-trained feature extractors. While these approaches are efficient in the high-privacy/small dataset regime, they are limited in expressivity by the data statistics that can be extracted during one-shot privatization. As a result, the performance of these methods does not scale well in the low-privacy/large dataset regime.

In our experimental comparisons, we excluded [270] and [46] due to concerns regarding their privacy guarantees. The privacy analysis of [270] relies on the Wishart mechanism, which has been retracted due to privacy leakage [248]. [46] attempt to train a score-based model while guaranteeing differential privacy through a data-dependent randomized response mechanism. In Section 4.4.2, we prove why their proposed mechanism leaks privacy, and further discuss other sources of privacy leakage.

Our DPDM relies on DP-SGD [1] to enforce DP guarantees. DP-SGD has also been used to train DP classifiers [78, 161, 274]. Recently, De et al. [64] demonstrated how to train very large discriminative models with DP-SGD and proposed augmentation multiplicity, which is related to our noise multiplicity, as discussed in Section 4.3.3.2. Furthrmore, DP-SGD has been utilized to train and fine-tune large language models [9, 172, 307], to protect sensitive training data in the medical domain [17, 320, 321], and to obscure geo-spatial location information [309].

Our work builds on DMs and score-based generative models [109, 257, 263]. DMs have been used prominently for image synthesis [16, 66, 110, 212, 230, 235, 241] and other image modeling tasks [142, 168, 197, 239, 240, 249]. They have also found applications in other areas, for instance in audio and speech generation [48, 130, 157], video generation [27, 111, 112, 252] and 3D synthesis [147, 188, 310, 319]. Methodologically, DMs have been adapted, for example, for fast sampling [74, 75, 134, 244, 258, 286, 295] and maximum likelihood training [151, 262, 280]. To the best of our knowledge, we are the first to train DMs under differential privacy guarantees.

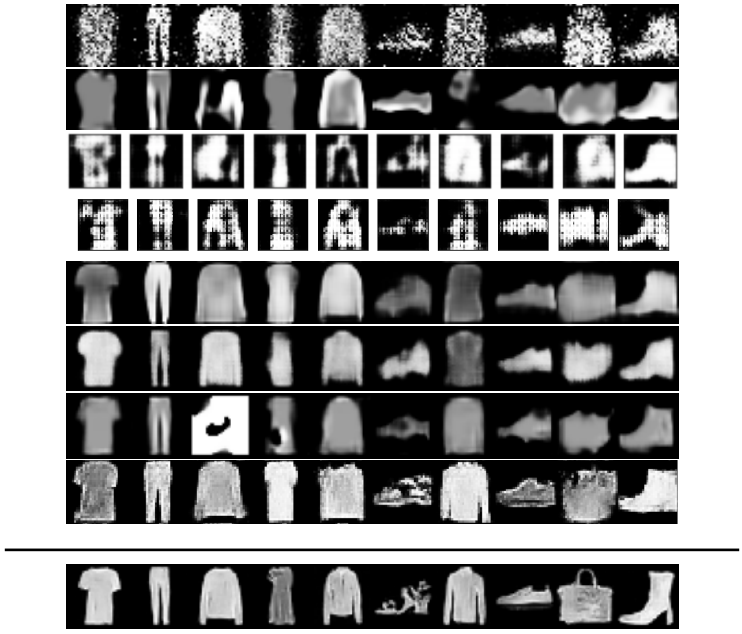


Figure 4.5: Fashion-MNIST images generated by, from top to bottom, DP-CGAN [273], DP-MERF [103], Datalens [284], G-PATE [182], GS-WGAN [45], DP-Sinkhorn [38], PEARL [173], DPGANr [26] (all above bar), and our DPDM (below bar) using the privacy budget ε=10. See Section 4.4.6.5 for more samples.

4.3.5 Experiments

In this subsection, we present results of DPDM on standard image synthesis benchmarks. Importantly, note that all models are private *by construction* through training with DP-SGD. The privacy guarantee is given by the (ε, δ) parameters of DP-SGD, clearly stated for each experiment below.

Datasets. We focus on image synthesis and use MNIST [164], Fashion-MNIST [293] (28x28), and CelebA [181] (downsampled to 32x32). These datasets are standard benchmarks in the DP generative modeling literature. In Section 4.4.7, we consider more challenging datasets and provide initial results.

Architectures. We implement the neural networks of DPDMs using the DDPM++ architecture [263]. See Section 4.4.3.2 for details.

Evaluation. We measure sample quality via Fréchet Inception Distance (FID) [105].

Table 4.1: Class-conditional DP image generation performance (MNIST & Fashion-MNIST). For PEARL [173], we train models and compute metrics ourselves (Section 4.4.6.1). All other results taken from the literature. DP-MEPF (†) uses additional public data for training (only included for completeness).

Method	DP-ε	MNIST				Fashion-MNIST			
		FID	Acc (%)			FID	Acc (%)		
			Log Reg	MLP	CNN		Log Reg	MLP	CNN
DPDM (FID) (*ours*)	0.2	**61.9**	65.3	65.8	71.9	**78.4**	53.6	55.3	57.0
DPDM (Acc) (*ours*)	0.2	104	**81.0**	**81.7**	**86.3**	128	**70.4**	**71.3**	**72.3**
PEARL [173]	0.2	133	76.2	77.1	77.6	160	70.0	70.8	68.0
DPDM (FID) (*ours*)	1	**23.4**	83.8	87.0	93.4	**37.8**	71.5	71.7	73.6
DPDM (Acc) (*ours*)	1	35.5	**86.7**	**91.6**	**95.3**	51.4	**76.3**	**76.9**	**79.4**
PEARL [173]	1	121	76.0	79.6	78.2	109	74.4	74.0	68.3
DPGANr [26]	1	56.2	-	-	80.1	121.8	-	-	68.0
DP-HP [281]	1	-	-	-	81.5	-	-	-	72.3
DPDM (FID) (*ours*)	10	**5.01**	90.5	94.6	97.3	**18.6**	80.4	81.1	84.9
DPDM (Acc) (*ours*)	10	6.65	**90.8**	**94.8**	**98.1**	19.1	**81.1**	**83.0**	**86.2**
PEARL [173]	10	116	76.5	78.3	78.8	102	72.6	73.2	64.9
DPGANr [26]	10	13.0	-	-	95.0	56.8	-	-	74.8
DP-Sinkhorn [38]	10	48.4	82.8	82.7	83.2	128.3	75.1	74.6	71.1
G-PATE [182]	10	150.62	-	-	80.92	171.90	-	-	69.34
DP-CGAN [273]	10	179.2	60	60	63	243.8	51	50	46
DataLens [284]	10	173.5	-	-	80.66	167.7	-	-	70.61
DP-MERF [102]	10	116.3	79.4	78.3	82.1	132.6	75.5	74.5	75.4
GS-WGAN [45]	10	61.3	79	79	80	131.3	68	65	65
DP-MEPF (ϕ_1) [103] (†)	0.2	-	72.1	77.1	-	-	71.7	69.0	-
DP-MEPF (ϕ_1, ϕ_2) [103] (†)	0.2	-	75.8	79.9	-	-	72.5	70.4	-
DP-MEPF (ϕ_1) [103] (†)	1	-	79.0	87.5	-	-	76.2	75.0	-
DP-MEPF (ϕ_1, ϕ_2) [103] (†)	1	-	82.5	89.3	-	-	75.4	74.7	-
DP-MEPF (ϕ_1) [103] (†)	10	-	80.8	88.8	-	-	75.5	75.5	-
DP-MEPF (ϕ_1, ϕ_2) [103] (†)	10	-	83.4	89.8	-	-	75.7	76.0	-

Table 4.2: Class prediction accuracy on real test data. DP-SGD: Classifiers trained directly with DP-SGD and real training data. DPDM: Classifiers trained non-privately on synthesized data from DP-SGD-trained DPDMs (using 60,000 samples, following [38]).

DP-ε	MNIST						Fashion-MNIST					
	Log Reg		MLP		CNN		Log Reg		MLP		CNN	
	DP-SGD	DPDM	DP-SGD	DPDM	DP-SGD	DPDM	DP-SGD	DPDM	DP-SGD	DPDM	DP-SGD	DPDM
0.2	**83.8**	81.0	**82.0**	81.7	69.9	**86.3**	**74.8**	70.4	**73.9**	71.3	59.5	**72.3**
1	**89.1**	86.7	89.6	**91.6**	88.2	**95.3**	**79.6**	76.3	**79.6**	76.9	70.5	**79.4**
10	**91.6**	90.8	92.9	**94.8**	96.4	**98.1**	**83.3**	81.1	**83.9**	83.0	77.1	**86.2**

Table 4.3: DM config ablation on MNIST for ε=0.2. See Table 4.12 for extended results.

DM config	FID	CNN-Acc (%)
VP [263]	197	24.2
VE [263]	171	13.9
v-prediction [244]	**97.8**	**84.4**
EDM [141]	119	49.2

On MNIST and Fashion-MNIST, we also assess utility of class-labeled generated data by training classifiers on synthesized samples and compute class prediction accuracy on real data. As is standard practice, we consider logistic regression (Log Reg), MLP, and CNN classifiers; see Section 4.4.6.1 for details.

Sampling. We sample from DPDM using (stochastic) DDIM [263] and the Churn sampler introduced in [141]. See Section 4.4.3.3 for details.

Privacy implementation: We implement DPDMs in PyTorch [222] and use Opacus [306], a DP-SGD library in PyTorch, for training and privacy accounting. We use δ=10^{-5} for MNIST and Fashion-MNIST, and δ=10^{-6} for CelebA. These values are standard [38] and chosen such that δ is smaller than the reciprocal of the number of training images. Similar to existing DP generative modeling work, we do not account for the (small) privacy cost of hyperparameter tuning. However, training and sampling is very robust with regards to hyperparameters, which makes DPDMs an ideal candidate for real privacy-critical situations; see Section 4.4.3.4.

4.3.5.1 Main Results

Class-conditional gray scale image generation. For MNIST and Fashion-MNIST, we train models for three privacy settings: ε=$\{0.2, 1, 10\}$ (Table 4.1). Informally, the three settings provide high, moderate, and low amounts of privacy, respectively. The DPDMs use the **v**-prediction DM config [244] for ε=0.2 and the EDM config [141] for ε=$\{1, 10\}$; see Section 4.3.5.2. We use the Churn sampler [141]: the two settings (FID) and (Acc) are based on the same DM, differing only in sampler setting; see Table 4.14 and Table 4.15 for all sampler settings.

DPDMs outperform all other existing models for all privacy settings and all metrics by large margins (see Table 4.1). Interestingly, DPDM also outperforms DP-MEPF [103], a method which is trained on additional public data, in 22 out of 24 setups. Generated samples for ε=10 are shown in Figure 4.5. Visually, DPDM's samples appear to be of significantly higher quality than the baselines'.

Comparison to DP-SGD-trained classifiers. Is it better to train a task-specific private classifier with DP-SGD directly, or can a non-private classifier trained on DPDM's synthethized data perform as well on downstream tasks? To answer this question, we train private classifiers with DP-SGD on real (training) data and compare them to our classifiers learnt using DPDM-synthesized data (details in Section 4.4.6.3). For a fair comparison, we are using the same architectures that we have already been using in our main experiments to quantify downstream classification accuracy (results in Table 4.2; we test on real (test) data). While direct DP-SGD training on real data outperforms the DPDM downstream classifier for logistic regression in all six setups (in line with empirical findings that it is easier to train classifiers with few parameters than large ones with DP-SGD [274]), CNN classifiers trained on DPDM's synthetic data generally outperform DP-SGD-trained classifiers. These results imply a very high utility of the synthetic data generated by DPDMs, demonstrating that DPDMs can potentially be used as an effective, privacy-preserving data sharing medium in practice. In fact, this approach is beneficial over training task-specific models with DP-SGD, because a user can generate as much data from DPDMs as they desire for various downstream applications without further privacy implications. To the best of our knowledge, it has not been demonstrated before in the DP generative modeling literature that image data generated by DP generative models can be used to train discriminative models on-par with directly DP-SGD-trained task-specific models.

Unconditional color image generation. On CelebA, we train models for ε=$\{1, 10\}$ (Table 4.4). The two DPDMs use the EDM config [141] as well as the Churn sampler;

Table 4.4: Unconditional CelebA generative performance. G-PATE and DataLens (†) use $\delta = 10^{-5}$ (less privacy) and model images at 64x64 resolution.

Method	DP-ε	FID
DPDM (*ours*)	1	71.8
DPDM (*ours*)	10	**21.1**
DP-Sinkhorn [38]	10	189.5
DP-MERF [102]	10	274.0
G-PATE [182] (†)	10	305.92
DataLens [284] (†)	10	320.8

Table 4.5: Noise multiplicity ablation on MNIST for ε=1. See Table 4.11 for extended results.

K	FID	CNN-Acc (%)
1	76.9	91.7
2	60.1	93.1
4	57.1	92.8
8	44.8	94.1
16	36.9	94.2
32	34.8	94.4

see Table 4.14. For ε=10, DPDM again outperforms existing methods by a significant margin. DPDM's synthesized images (see Figure 4.16) appear much more diverse and vivid than the baselines' samples.

4.3.5.2 Ablation Studies

Noise multiplicity. Table 4.5 shows results for DPDMs trained with different noise multiplicity K (using the **v**-prediction DM config) [244]. As expected, increasing K leads to a general trend of improving performance; however, the metrics start to plateau at around K=32.

Diffusion model config. We train DPDMs with different DM configs (see Section 4.4.3.1). VP- and VE-based models [263] perform poorly for all settings, while for ε=0.2 **v**-prediction significantly outperforms the EDM config on MNIST (Table 4.3). On Fashion-MNIST, the advantage is less significant (extended Table 4.12). For ε=$\{1, 10\}$, the EDM config performs better than **v**-prediction. Note that the denoiser parameterization for these configs is almost identical and their main difference is the noise distribution $p(\sigma)$ (Figure 4.4). As discussed in Section 4.3.3.2, oversampling large noise levels σ is expected to be especially important for the large privacy setting (small ε), which is validated by our ablation.

Table 4.6: Sampler comparison on MNIST (see Table 4.13 for results on Fashion-MNIST). We compare the Churn sampler [141] to DDIM [258].

Sampler	DP-ε	FID	Acc (%)		
			Log Reg	MLP	CNN
Churn (FID)	0.2	**61.9**	65.3	65.8	71.9
Churn (Acc)	0.2	104	81.0	81.7	**86.3**
Stochastic DDIM	0.2	97.8	80.2	81.3	84.4
Deterministic DDIM	0.2	120	**81.3**	**82.1**	84.8
Churn (FID)	1	**23.4**	83.8	87.0	93.4
Churn (Acc)	1	35.5	**86.7**	91.6	**95.3**
Stochastic DDIM	1	34.2	86.2	90.1	94.9
Deterministic DDIM	1	50.4	85.7	**91.8**	94.9
Churn (FID)	10	**5.01**	90.5	94.6	97.3
Churn (Acc)	10	6.65	**90.8**	94.8	**98.1**
Stochastic DDIM	10	6.13	90.4	94.6	97.5
Deterministic DDIM	10	10.9	90.5	**95.2**	97.7

Sampling. Table 4.6 shows results for different samplers: deterministic and stochastic DDIM [258] as well as the Churn sampler (tuned for high FID scores and downstream accuracy); see Section 4.4.3.3 for details on the samplers. Stochastic sampling is crucial to obtain good perceptual quality, as measured by FID (see poor performance of deterministic DDIM), while it is less important for downstream accuracy. We hypothesize that FID better captures image details that require a sufficiently accurate synthesis process. As discussed in Sections 4.3.2.1 and 4.3.3.1, stochastic sampling can help with that and therefore is particularly important in DP-SGD-trained DMs. We also observe that the advantage of the Churn sampler compared to stochastic DDIM becomes less significant as ε increases. Moreover, in particular for ε=0.2 the FID-adjusted Churn sampler performs poorly on downstream accuracy. This is arguably because its settings sacrifice sample diversity, which downstream accuracy usually benefits from, in favor of synthesis quality (also see samples in Section 4.4.6.5).

4.3.6 Conclusions

We propose *Differentially Private Diffusion Models* (DPDMs), which use DP-SGD to enforce DP guarantees. DMs are strong candidates for DP generative learning due to their robust training objective and intrinsically less complex denoising neural networks. To re-

duce the gradient variance during training, we introduce noise multiplicity and find that DPDMs achieve state-of-the-art performance in common DP image generation benchmarks. Furthermore, downstream classifiers trained with DPDM-generated synthetic data perform on-par with task-specific discriminative models trained with DP-SGD directly. Note that despite their state-of-the-art results, DPDMs are based on a straightforward idea, this is, to carefully combine DMs with DP-SGD (leveraging the novel noise multiplicity). This "simplicity" is a crucial advantage, as it makes DPDMs a potentially powerful tool that can be easily adopted by DP practitioners. Based on our promising results, we conclude that DMs are an ideal generative modeling framework for DP generative learning. Moreover, we believe that advancing DM-based DP generative modeling is a pressing topic, considering the extremely fast progress of DM-based large-scale photo-realistic image generation systems [16, 230, 235, 241]. As future directions we envision applying our DPDM approach during training of such large image generation DMs, as well as applying DPDMs to other types of data. Furthermore, it may be interesting to pre-train our DPDMs with public data that is not subject to privacy constraints, similar to Harder et al. [103], which may boost performance. Also see Section 4.4.8 for further discussion on ethics, reproducibility, limitations and more future work.

4.4 Appendix

4.4.1 Differential Privacy and Proof of Theorem 2

In this subsection, we provide a short proof that the gradients released by the Gaussian mechanism in DPDM are DP. By DP, we are specifically refering to the (ε, δ)-DP as defined in Definition 4.3.1, which approximates (ε)-DP. For completeness, we state the definition of Rényi Differential Privacy (RDP) [199]:

Definition 4.4.1. (Rényi Differential Privacy) A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (α, ϵ)-RDP if for any adjacent $d, d' \in \mathcal{D}$:

$$D_\alpha(\mathcal{M}(d)|\mathcal{M}(d')) \leq \epsilon, \tag{4.8}$$

where D_α is the Rényi divergence of order α.

Gaussian mechanism can provide RDP according to the following theorem:

Theorem 3. (RDP Gaussian mechanism [199]) For query function f with Sensitivity $S = \max_{d,d} ||f(d) - f(d')||_2$, the mechanism that releases $f(d) + \mathcal{N}(0, \sigma^2_{\text{DP}})$ satisfies $(\alpha, \alpha S^2/(2\sigma^2))$-RDP.

Note that any $\mathcal{M}$ that satisfies (α, ϵ)-RDP also satisfies $(\epsilon + \frac{\log 1/\delta}{\alpha - 1}, \delta)$-DP.

We slightly deviate from the notation used in the main text to make the dependency of variables on input data explicit. Recall from the main text that the per-data point loss is computed as an average over K noise samples:

$$\tilde{l}_i = \frac{1}{K} \sum_{k=1}^{K} \lambda(\sigma_{ik}) \| D_{\boldsymbol{\theta}}(\mathbf{x}_i + \mathbf{n}_{ik}, \sigma_{ik}) - \mathbf{x}_i \|_2^2, \text{ where } \{(\sigma_{ik}, \mathbf{n}_{ik})\}_{k=1}^{K} \sim p(\sigma) \mathcal{N}\left(\mathbf{0}, \sigma^2\right). \quad (4.9)$$

In each iteration of Algorithm 4, we are given a (random) set of indices $\mathbb{B}$ of expected size B with no repeated indices, from which we construct a mini-batch $\{\mathbf{x}_i\}_{i \in \mathbb{B}}$. In our implementation (which is based on Yousefpour et al. [306]) of the Gaussian mechanism for gradient sanitization, we compute the gradient of l_i and apply clipping with norm C, and then divide the clipped gradients by the expected batch size B to obtain the batched gradient G_{batch}:

$$G_{batch}(\{\mathbf{x}_i\}_{i \in \mathbb{B}}) = \frac{1}{B} \sum_{i \in \mathbb{B}} \texttt{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i) \right). \quad (4.10)$$

Finally, Gaussian noise $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{DP}}^2)$ is added to G_{batch} and released as the response $\tilde{G}_{batch}$:

$$\tilde{G}_{batch}(\{\mathbf{x}_i\}_{i \in \mathbb{B}}) = G_{batch}(\{\mathbf{x}_i\}_{i \in \mathbb{B}}) + \frac{C}{B} \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{DP}}^2 \boldsymbol{I}) \quad (4.11)$$

Now, we can restate Theorem 2 as follows with our modified notation:

Theorem 4. For noise magnitude σ_{DP}, dataset $d = \{\mathbf{x}_i\}_{i=1}^{N}$, and set of (non-repeating) indices $\mathbb{B}$, releasing $\tilde{G}_{batch}(\{\mathbf{x}_i\}_{i \in \mathbb{B}})$ satisfies $(\alpha, \alpha / 2\sigma_{\text{DP}}^2)$-RDP.

Proof. Without loss of generality, consider two neighboring datasets $d = \{\mathbf{x}_i\}_{i=1}^{N}$ and $d' = d \cup \mathbf{x}'$, $\mathbf{x}' \notin d$, and mini-batches $\{\mathbf{x}_i\}_{i \in \mathbb{B}}$ and $\mathbf{x}' \cup \{\mathbf{x}_i\}_{i \in \mathbb{B}}$, where the counter-factual set/batch has one additional entry $\mathbf{x}'$. We can bound the difference of their gradients in L_2-norm as:

$$\begin{aligned}
&\| G_{batch}(\{\mathbf{x}_i\}_{i \in \mathbb{B}}) - G_{batch}(\mathbf{x}' \cup \{\mathbf{x}_i\}_{i \in \mathbb{B}}) \|_2 \\
&= \left\| \frac{1}{B} \sum_{i \in \mathbb{B}} \text{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i) \right) - \left(\frac{1}{B} \text{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}') \right) + \frac{1}{B} \sum_{i \in \mathbb{B}} \text{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i) \right) \right) \right\|_2 \\
&= \left\| -\frac{1}{B} \text{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}') \right) \right\|_2 \\
&= \frac{1}{B} \left\| \text{clip}_C \left(\nabla_{\boldsymbol{\theta}} l(\mathbf{x}') \right) \right\|_2 \leq \frac{C}{B}.
\end{aligned}$$

We thus have *sensitivity* $S(G_{batch}) = \frac{C}{B}$. Furthermore, since $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{\text{DP}})$, $(C/B)\mathbf{z} \sim \mathcal{N}(\mathbf{0}, (C/B)^2\sigma^2_{\text{DP}})$. Following standard arguments, releasing $\tilde{G}_{batch}(\{\mathbf{x}_i\}_{i\in\mathbb{B}}) = G_{batch}(\{\mathbf{x}_i\}_{i\in\mathbb{B}}) + (C/B)\mathbf{z}$ satisfies $(\alpha, \alpha/2\sigma^2_{\text{DP}})$-RDP [199]. □

In practice, we construct mini-batches by sampling the training dataset for privacy amplification via Poisson Sampling [200], and compute the overall privacy cost of training DPDM via RDP composition [199]. We use these processes as implemented in Opacus [306].

For completeness, we also include the Poisson Sampling algorithm in Algorithm 5.

Algorithm 5 Poisson Sampling

Input : Index range N, subsampling rate q
Output: Random batch of indices $\mathbb{B}$ (of expected size B)
$\mathrm{c} = \{c_i\}_{i=1}^N \sim \texttt{Bernoulli}(q)$
$\mathbb{B} = \{j : j \in \{1, \ldots, N\}, \mathrm{c}_j = 1\}$

4.4.2 DPGEN Analysis

In this subsection, we provide a detailed analysis of the privacy guarantees provided in DPGEN [46].

As a brief overview, Chen et al. [46] proposes to learn an energy function $q_\vartheta(\mathbf{x})$ by optimizing the following objective ([46], Eq. 7):

$$l(\theta; \sigma) = \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{N}(\mathbf{x},\sigma^2)}\left[\left|\left|\frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} - \nabla_{\mathbf{x}} \log q_\vartheta(\mathbf{x})\right|\right|^2\right].$$

In practice, the first expectation is replaced by averaging over examples in a private training set $d = \{x_i : x_i \in Y, i \in 1, \ldots, m\}$, and $\frac{\tilde{x}-x}{\sigma^2}$ is replaced by $d_i^r = (\tilde{x}_i - x_i^r)/\sigma_i^2$ for each i in $[1, m]$ (not to be confused with d which denotes the dataset in the DP context), where x_i^r is the query response produced by a data-dependent randomized response mechanism.

We believe that there are three errors in DPGEN that renders the privacy guarantee in DPGEN false. We formally prove the first error in the following subsection, and state the other two errors which are factual but not mathematical. The three errors are:

- The randomized response mechanism employed in DPGEN has a output space that is only supported (has non-zero probability) on combinations of its input *private*

dataset. ϵ-differential privacy cannot be achieved as outcomes with non-zero probability[1] can have zero probability when the input dataset is changed by one element. Furthermore, adversaries observing the output can immediately deduce elements of the private dataset.

- The k-nearest neighbor filtering used by DPGEN to reduce the number of candidates for the randomized response mechanism is a function of the private data. The likelihood of the k-selected set varies with the noisy image $\tilde{x}$ (line 20 of algorithm 1 in DPGEN), and is not correctly accounted for in DPGEN.
- The objective function used to train the denoising network in DPGEN depends on both the ground-truth denoising direction and a noisy image provided to the denoising network. The noisy image is dependent on the training data, and hence leaks privacy. The privacy cost incurred by using this noisy image is not accounted for in DPGEN.

To prove the first error, we begin with re-iterating the formal definition of differential privacy (DP):

Definition 4.4.2. (ϵ-Differential Privacy) A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{I}$ with domain $\mathcal{D}$ and image $\mathcal{I}$ satisfies (ε)-DP if for any two adjacent inputs $d, d' \in \mathcal{D}$ differing by at most one entry, and for any subset of outputs $S \subseteq \mathcal{I}$ it holds that

$$\mathbf{Pr}\left[\mathcal{M}(d) \in S\right] \leq e^{\varepsilon}\mathbf{Pr}\left[\mathcal{M}(d') \in S\right]. \tag{4.12}$$

The randomized response (RR) mechanism is a fundamental privacy mechanism in differential privacy. A key assumption required in the RR mechanism is that the choices of random response are not dependent on private information, such that when a respondent draws their response randomly from the possible choices, no private information is given. More formally, we give the following definition for randomized response over multiple choices[2]:

Definition 4.4.3. Given a fixed response set Y of size k. Let $d = \{x_i : x_i \in Y, i \in 1, \ldots, m\}$ be an input dataset. Define "randomized response" mechanism $\mathcal{RR}$ as:

$$\mathcal{RR}(d) = \{G(x_i)\}_{i \in [1,m]} \tag{4.13}$$

where,

$$G(x_i) = \begin{cases} x_i, \text{ with probability } \frac{e^{\epsilon}}{e^{\epsilon}+k-1} \\ x_i' \in Y \setminus x_i, \text{ with probability } \frac{1}{e^{\epsilon}+k-1} \end{cases}. \tag{4.14}$$

[1] probability over randomness in the privacy mechanism

[2] This mechanism is analogous to the coin flipping mechanism, where the participant first flip a biased coin to determine whether they'll answer truthfully or lie with probability of lying $\frac{k}{e^{\epsilon}+k-1}$, and if they were to lie, they then roll a fair k dice to determine the response.

A classical result is that the mechanism $\mathcal{RR}$ satisfies ϵ-DP [84].

DPGEN considers datasets of the form $d = \{x_i : x_i \in \mathbb{R}^n, i \in 1, \ldots, m\}$. It claims to guarantee differential privacy by applying a stochastic function H to each element of the dataset defined as follows (Eq. 8 of [46]):

$$\Pr[H(\tilde{x}_i) = w] = \begin{cases} \frac{e^\epsilon}{e^\epsilon + k - 1}, & w = x_i \\ \frac{1}{e^\epsilon + k - 1}, & w = x_i' \in \mathrm{X} \setminus x_i \end{cases},$$

where $\mathrm{X} = \{x_j : max(\tilde{x}_i - x_j)/\sigma_j \leq \beta, x_j \in d\}$ (max is over the dimensions of $\tilde{x}_i - x_j$), $|\mathrm{X}| = k \geq 2$, and $\tilde{x}_i = x_i + z_i$, $z_i \sim \mathcal{N}(0, \sigma^2 I)$. We first note that H is not only a function of $\tilde{x}_i$ but also $\mathrm{X} \cup x_i$, since its image is determined by $\mathrm{X} \cup x_i$. That is, changes in X will alter the possible outputs of H, independently from the value of $\tilde{x}_i$. We make this dependency explicit in our formulation here-forth. This distinction is important as it determines the set of possible outcomes that we need to consider for in the privacy analysis. The authors also noted that z_i is added for training with the denoising objective, not for privacy, so this added Gaussian noise is not essential to the privacy analysis. Furthermore, since k (or equivalently β) is a hyperparameter that can be tuned, we consider the simpler case where $k = m$, i.e. $\mathrm{X} = d$, as done in the appendix (Eq. 9) by the authors. Thereby we define the privacy mechanism utilized in DPGEN as follows:

Definition 4.4.4. Let $d = \{x_i : x_i \in \mathbb{R}^n, i \in 1, \ldots, m\}$ be an input dataset. Define "data dependent randomized response" $\mathcal{M}$ as:

$$\mathcal{M}(d) = \{H(x_i, d)\}_{i \in [1,m]} \tag{4.15}$$

where,

$$H(x_i, d) = \begin{cases} x_i, \text{ with probability } \frac{e^\epsilon}{e^\epsilon + m - 1} \\ x_i' \in d \setminus x_i, \text{ with probability } \frac{1}{e^\epsilon + m - 1} \end{cases}. \tag{4.16}$$

Since the image of $H(x_i, d)$ is d, $\mathcal{M}(d)$ is only supported on d^m.[3] In other words, the image of $\mathcal{M}$ is data dependent, and any outcome O (which are sets of $\mathbb{R}^n$ tensors, of cardinality m) that include elements which are not in d would have a probability of zero to be the outcome of $\mathcal{M}(d)$, i.e. if there exists $z \in O$ and $z \notin d$, then $\Pr[\mathcal{M}(d) = O] = 0$.

To construct our counter-example, we start with considering two neighboring datasets:

[3] We mean dataset-exponentiation in the sense of repeated cartesian products between sets, i.e. $d^2 = d \otimes d$

the training data $\mathrm{d} = \{x_i : x_i \in \mathbb{R}^n, i \in 1, \ldots, m\}$, and a counter-factual dataset $\mathrm{d}' = \{x'_1 : x'_1 \in \mathbb{R}^n, x_i : x_i \in \mathbb{R}^n, i \in 2, \ldots, m\}$, differing in their first element ($x_1 \neq x'_1$). Importantly, since differential privacy requires that the likelihoods of outputs to be similar for all valid pairs of neighboring datasets, we are free to assume that elements of d are unique, i.e. no two rows of d are identical.

Another requirement of differential privacy is that the likelihood of any subsets of outputs must be similar, hence we are free to choose any valid response for the counter-example. Thus, letting O denote the outcome of $\mathcal{M}(\mathrm{d})$, we choose $O = d = \{x_1, \ldots, x_m\}$. Clearly, by Definition 0.3, this is a plausible outcome of $\mathcal{M}(\mathrm{d})$ as it is in the support d^m. However, O is not in the support of $\mathcal{M}(\mathrm{d}')$ since the first element x_1 is not in the image of $H(\cdot, \mathrm{d}')$; that is $\Pr[H(x, \mathrm{d}') = x_1] = 0$ for all $x \in \mathrm{d}'$. Privacy protection is violated since any adversary observing O can immediately deduce the participation of x_1 in the data release as opposed to any counterfactual data x'_1.

More formally, consider response set $T = \{O\} \subset \mathrm{d}^m$, and d^m is the image of $\mathcal{M}(\mathrm{d})$, we have

$$\Pr[\mathcal{M}(d) \in T] = \Pr[\mathcal{M}(d) = O] \tag{4.17}$$

$$= \Pr[H(x_1) = x_1] \prod_{i=2}^{m} \Pr[H(x_i) = x_i] \quad \text{(independent dice rolls)} \tag{4.18}$$

$$= \frac{e^\epsilon}{e^\epsilon + m - 1} \prod_{i=2}^{m} \Pr[H(x_i) = x_i] \quad \text{(apply definition 4.4.4)} \tag{4.19}$$

$$> 0 \prod_{i=2}^{m} \Pr[H(x_i) = x_i] \tag{4.20}$$

$$= \Pr[H(x'_1) = x_1] \prod_{i=2}^{m} \Pr[H(x_i) = x_i] \tag{4.21}$$

$$= \Pr[\mathcal{M}(d') = O] = \Pr[\mathcal{M}(d') \in T]. \tag{4.22}$$

We can observe that $\Pr[\mathcal{M}(d') \in T] = 0$, as shown in line 9. Clearly, this result violates ϵ-DP for all ϵ, which requires $\Pr[\mathcal{M}(d) \in T] \leq e^\epsilon \Pr[\mathcal{M}(d') \in T]$.

In essence, by using private data to form the response set, we make the image of the privacy mechanism data-dependent. This in turn leaks privacy, since an adversary can

immediately rule-out all counter-factual datasets that do not include every element of the response O, as these counter-factuals now have likelihood 0. To fix this privacy leak, one could determine a response set a-priori, and use the $\mathcal{RR}$ mechanism in Definition 4.4.3 to privately release data. This modification may not be feasible in practice, since constructing a response set of finite size (k) suitable for images is non-trivial. Hence, we believe that it would require fundamental modifications to DPGEN to achieve differential privacy.

Regarding error 2, we point out that in the paragraph following Eq. 8 in DPGEN, X is defined as the set of k points in d that are closest to $\tilde{x}_i$ when weighted by σ_j. This means that the membership of X is dependent on the value of $\tilde{x}_i$. Thus, any counter-factual input x_i' and $\tilde{x}_i'$ with a different set of k nearest neighbors could have many possible outcomes with 0 likelihood under the true input. In essence, this is a more extreme form of data-dependent randomized response where the response set is dependent on both d and x_i.

Regarding error 3, the loss objective in DPGEN includes the term $\nabla_x \log q_\theta(\tilde{x})$ (Eq. 7 of DPGEN, $l = \frac{1}{2}\mathbb{E}_{p(x)}E_{\tilde{x}\sim N(x,\sigma^2)}\left[||\frac{\tilde{x}-x}{\sigma^2} - \nabla_x \log q_\theta(\tilde{x})||^2\right]$), and $\tilde{x}$ is also a function of the private data that is yet to be accounted for at all in the privacy analysis of DPGEN. Hence, one would need to further modify the learning algorithm in DPGEN, such that the inputs to the score model are either processed through an additional privacy mechanism, or sampled randomly without dependence on private data.

Regarding justifying the premise that DPGEN implements the data-dependent randomized response mechanism, we have verified that the privacy mechanism implemented in the repository of DPGEN (https://github.com/chiamuyu/DPGEN[4]) is indeed data-dependent:

In line 30 of `losses/dsm.py`:

```
sample_ix = random.choices(range(k), weights=weight)[0]
```

randomly selects an index in the range of $[0, k-1]$, which is then used in line 46,

```
sample_buff.append(samples[sample_ix]),
```

to index the private training data and assigned to the output of

```
sample_buff.
```

Values of this variable are then accessed on line 85 to calculate the $\frac{\tilde{x}-x^r}{\sigma^2}$ (as x^r) term in the objective function ([46], Eq. 7).

[4]In particular, we refer to the code at commit: 1f684b9b8898bef010838c6a29c030c07d4a5f87.

Table 4.7: Four popular DM configs from the literature.

	VP [263]	VE [263]	v-prediction [244]	EDM [141]
Network and preconditioning				
Skip scaling $c_{\text{skip}}(\sigma)$	1	1	$1/(\sigma^2+1)$	$\sigma_{\text{data}}^2/(\sigma^2+\sigma_{\text{data}}^2)$
Output scaling $c_{\text{out}}(\sigma)$	$-\sigma$	σ	$\sigma/\sqrt{1+\sigma^2}$	$\sigma\cdot\sigma_{\text{data}}/\sqrt{\sigma_{\text{data}}^2+\sigma^2}$
Input scaling $c_{\text{in}}(\sigma)$	$1/\sqrt{\sigma^2+1}$	1	$1/\sqrt{\sigma^2+1^2}$	$1/\sqrt{\sigma^2+\sigma_{\text{data}}^2}$
Noise cond. $c_{\text{noise}}(\sigma)$	$(M-1)\,t$	$\ln(\frac{1}{2}\sigma)$	t	$\frac{1}{4}\ln(\sigma)$
Training				
Noise distribution	$t\sim\mathcal{U}(\epsilon_t,1)$	$\ln(\sigma)\sim\mathcal{U}(\ln(\sigma_{\min}),\ln(\sigma_{\max}))$	$t\sim\mathcal{U}(\epsilon_{\min},\epsilon_{\max})$	$\ln(\sigma)\sim\mathcal{N}(P_{\text{mean}},P_{\text{std}}^2)$
Loss weighting $\lambda(\sigma)$	$1/\sigma^2$	$1/\sigma^2$	$(\sigma^2+1)/\sigma^2$ ("SNR+1" weighting)	$(\sigma^2+\sigma_{\text{data}}^2)/(\sigma\cdot\sigma_{\text{data}})^2$
Parameters	$\beta_d=19.9,\beta_{\min}=0.1$	$\sigma_{\min}=0.002$	$\epsilon_{\min}=\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{-13}}}$	$P_{\text{mean}}=-1.2,\ P_{\text{std}}=1.2$
	$\epsilon_t=10^{-5},M=1000$	$\sigma_{\max}=80$	$\epsilon_{\max}=\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{9}}}$	$\sigma_{\text{data}}=\sqrt{\frac{1}{3}}$
	$\sigma(t)=\sqrt{e^{\frac{1}{2}\beta_d t^2+\beta_{\min}t}-1}$		$\sigma(t)=\sqrt{\cos^{-2}(\pi t/2)-1}$	

4.4.3 Model and Implementation Details

4.4.3.1 Diffusion Model Configs

As discussed in Section 4.3.2, previous works proposed various denoiser models $D_{\boldsymbol{\theta}}$, noise distributions $p(\sigma)$, and weighting functions $\lambda(\sigma)$. We refer to the triplet $(D_{\boldsymbol{\theta}},p,\lambda)$ as DM config. In this work, we consider four such configs: *variance preserving* (VP) [263], *variance exploding* (VE) [263], **v**-prediction [244], and EDM Karras et al. [141]. The triplet for each of these configs can be found in Table 4.7. Note, that we use the parameterization of the denoiser model $D_{\boldsymbol{\theta}}$ from [141]

$$D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)=c_{\text{skip}}(\sigma)\mathbf{x}+c_{\text{out}}(\sigma)F_{\boldsymbol{\theta}}(c_{\text{in}}(\sigma)\mathbf{x};c_{\text{noise}}(\sigma)), \tag{4.23}$$

where $F_{\boldsymbol{\theta}}$ is the raw neural network. To accommodate for our particular sampler setting (we require to learn the denoiser model for $\sigma\in[0.002,80]$; see Section 4.4.3.3) we slightly modified the parameters of VE and **v**-prediction. For VE, we changed $\sigma_{\min}$ and $\sigma_{\max}$ from 0.02 to 0.002 and from 100 to 80, respectively. For **v**-prediction, we changed $\epsilon_{\min}$ and $\epsilon_{\max}$ from $\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{-20}}}$ to $\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{-13}}}$ and $\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{20}}}$ to $\frac{2}{\pi}\arccos\frac{1}{\sqrt{1+e^{9}}}$, respectively. Furthermore, we cannot base our EDM models on the true (training) data standard deviation σ_{data} as releasing this information would result in a privacy cost. Instead, we set σ_{data} to the standard deviation of a uniform distribution between -1 and 1, assuming no prior information on the modeled image data.

4.4.3.1.1 Noise Level Visualization In the following, we provide details on how exactly the noise distributions of the four configs are visualized in Figure 4.4. The reason we want to plot these noise distributions is to understand how the different configs assign weight to different noise levels σ during training through sampling some σ's more and others less. However, to be able to make a meaningful conclusion, we also need to take into account the loss weighting $\lambda(\sigma)$.

Therefore, we consider the effective "importance-weighted" distributions $p(\sigma)\frac{\lambda(\sigma)}{\lambda_{\text{EDM}}(\sigma)}$, where we use the loss weighting from the EDM config as reference weighting.

The $\frac{\lambda(\sigma)}{\lambda_{\text{EDM}}(\sigma)}$ weightings for VP, VE, $\mathbf{v}$-prediction, and EDM are then, $\sigma_{\text{data}}^2/(\sigma^2+\sigma_{\text{data}}^2)$, $\sigma_{\text{data}}^2/(\sigma^2+\sigma_{\text{data}}^2)$, $\sigma_{\text{data}}^2(\sigma^2+1)/(\sigma^2+\sigma_{\text{data}}^2)$, and 1, respectively. Figure 4.4 then visualizes the "importance-weighted" distributions in log-σ space, following Karras et al. [141] (that way, the final visualized log-σ distribution of EDM remains a normal distribution $\mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$).

4.4.3.2 Model Architecture

We focus on image synthesis and implement the neural network backbone of DPDMs using the DDPM++ architecture [263]. For class-conditional generation, we add a learned class-embedding to the σ-embedding as is common practice [66]. All model hyperparameters and training details can be found in Table 4.8.

4.4.3.3 Sampling from Diffusion Models

Let us recall the differential equations we can use to generate samples from DMs:

$$\text{ODE: } d\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt, \tag{4.24}$$

$$\text{SDE: } d\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt - \beta(t)\sigma^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt + \sqrt{2\beta(t)}\sigma(t)\, d\omega_t. \tag{4.25}$$

Before choosing a numerical sampler, we first need to define a sampling schedule. In this work, we follow Karras et al. [141] and use the schedule

$$\sigma_i = \left(\sigma_{\max}^{1/\rho} + \frac{i}{M-1}(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho})\right)^{\rho}, i \in \{0,\dots,M-1\}, \tag{4.26}$$

Table 4.8: Model hyperparameters and training details.

Hyperparameter	MNIST & Fashion-MNIST	CelebA
Model		
Data dimensionality (in pixels)	28	32
Residual blocks per resolution	2	2
Attention resolution(s)	7	8,16
Base channels	32	32
Channel multipliers	1,2,2	1,2,2
EMA rate	0.999	0.999
# of parameters	1.75M	1.80M
Base architecture	DDPM++ [263]	DDPM++ [263]
Training		
# of epochs	300	300
Optimizer	Adam [148]	Adam [148]
Learning rate	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
Batch size	4096	2048
Dropout	0	0
Clipping constant C	1	1
DP-δ	10^{-5}	10^{-6}

with ρ=7.0, $\sigma_{\max}$=80 and $\sigma_{\min}$=0.002. We consider two solvers: the (stochastic ($\eta = 1$)/deterministic ($\eta = 0$)) DDIM solver [258] as well as the stochastic Churn solver introduced in [141], for pseudocode see Algorithm 6 and Algorithm 7, respectively. Both implementations can readily be combined with classifier-free guidance, which is described in Paragraph 4.4.3.3.1, in which case the denoiser $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$ may be replaced by $D^w_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{y})$, where the guidance scale w is a hyperparameter. Note that the Churn sampler has four additional hyperparameters which should be tuned empirically [141]. If not stated otherwise, we set M=1000 for the Churn sampler and the stochastic DDIM sampler, and M=50 for the deterministic DDIM sampler.

Algorithm 6 DDIM sampler [258]

Input: Denoiser $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$, Schedule $\{\sigma_i\}_{i\in\{0,\dots,M-1\}}$
Output: Sample $\mathbf{x}_M$
Sample $\mathbf{x}_0 \sim \mathcal{N}\left(\mathbf{0}, \sigma_0^2 \boldsymbol{I}\right)$
for $n = 0$ **to** $M-2$ **do**
  Evaluate denoiser $\mathbf{d}_n = D_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i)$
  if Stochastic DDIM **then**
    $\mathbf{x}_{n+1} = \mathbf{x}_n + 2\frac{\sigma_{n+1}-\sigma_n}{\sigma_n}(\mathbf{x}_n - \mathbf{d}_n) + \sqrt{2(\sigma_n - \sigma_{n+1})\sigma_n}\mathbf{z}_n, \quad \mathbf{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  else if Deterministic DDIM **then**
    $\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\sigma_{n+1}-\sigma_n}{\sigma_n}(\mathbf{x}_n - \mathbf{d}_n)$
  end if
end for
Return $\mathbf{x}_M = D(\mathbf{x}_{N-1}, \sigma_{M-1})$

Algorithm 7 Churn sampler [141]

Input: Denoiser $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$, Schedule $\{\sigma_i\}_{i\in\{0,\dots,M-1\}}$, S_{noise}, S_{churn}, S_{min}, S_{max}
Output: Sample $\mathbf{x}_M$
Set $\sigma_M = 0$
Sample $\mathbf{x}_0 \sim \mathcal{N}\left(\mathbf{0}, \sigma_0^2 \boldsymbol{I}\right)$
for $n = 0$ **to** $M-1$ **do**
  if $\sigma_i \in [S_{\text{min}}, S_{\text{max}}]$ **then**
    $\gamma_i = \min(\frac{S_{\text{churn}}}{M}, \sqrt{2}-1)$
  else
    $\gamma_i = 0$
  end if
  Increase noise level $\widetilde{\sigma}_n = (1+\gamma_n)\sigma_n$
  Sample $\mathbf{z}_n \sim \mathcal{N}\left(\mathbf{0}, S_{\text{noise}}^2 \boldsymbol{I}\right)$ and set $\widetilde{\mathbf{x}}_n = \mathbf{x}_n + \sqrt{\widetilde{\sigma}_n^2 - \sigma_n^2}\mathbf{z}_n$
  Evaluate denoiser $\mathbf{d}_n = D_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}_n, \widetilde{\sigma}_n)$ and set $\mathbf{f}_n = \frac{\widetilde{\mathbf{x}}_n - \mathbf{d}_n}{\widetilde{\sigma}_n}$
  $\mathbf{x}_{n+1} = \widetilde{\mathbf{x}}_M + (\sigma_{n+1} - \widetilde{\sigma}_n)\mathbf{f}_n$
  if $\sigma_{n+1} \neq 0$ **then**
    Evaluate denoiser $\mathbf{d}'_n = D_{\boldsymbol{\theta}}(\mathbf{x}_{n+1}, \sigma_{n+1})$ and set $\mathbf{f}'_n = \frac{\mathbf{x}_{n+1} - \mathbf{d}'_n}{\sigma_{n+1}}$
    Apply second order correction: $\mathbf{x}_{n+1} = \widetilde{\mathbf{x}}_n + \frac{1}{2}(\sigma_{n+1} - \widetilde{\sigma}_n)(\mathbf{f}_n + \mathbf{f}'_n)$
  end if
end for
Return $\mathbf{x}_M$

4.4.3.3.1 Guidance Classifier guidance [66, 263] is a technique to guide the diffusion sampling process towards a particular conditioning signal $\mathbf{y}$ using gradients, with respect to $\mathbf{x}$, of a pre-trained, noise-conditional classifier $p(\mathbf{y}|\mathbf{x},\sigma)$. Classifier-free guidance [108], in contrast, avoids training additional classifiers by mixing denoising predictions of an unconditional and a conditional model, according to a *guidance scale* w, by replacing $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$ in the score parameterization $s_{\boldsymbol{\theta}} = (D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - \mathbf{x})/\sigma^2$ with

$$D^w_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{y}) = (1-w)D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) + wD_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{y}). \tag{4.27}$$

$D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$ and $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{y})$ can be trained jointly; to train $D_{\boldsymbol{\theta}}(\mathbf{x};\sigma)$ the conditioning signal $\mathbf{y}$ is discarded at random and replaced by a *null token* [108]. Increased guidance scales w tend to drive samples deeper into the model's modes defined by $\mathbf{y}$ at the cost of sample diversity.

Table 4.9: DP noise σ_{DP} used for all our experiments.

ε	MNIST	Fashion-MNIST	CelebA
0.2	82.5	82.5	N/A
1	18.28125	18.28125	8.82812
10	2.48779	2.48779	1.30371

4.4.3.4 Hyperparameters of Differentially Private Diffusion Models

Tuning hyperparameters for DP models generally induces a privacy cost which should be accounted for [219]. Similar to existing works [64], we neglect the (small) privacy cost associated with hyperparameter tuning. Nonetheless, in this subsection we want to point out that our hyperparameters show consistent trends across different settings. As a result, we believe our models need little to no hyperparameter tuning in similar settings to the ones considered in this work.

Model. We use the DDPM++ [263] architecture for all models in this work. Across all three datasets (MNIST, Fashion-MNIST, and CelebA) we found the EDM [141] config to perform best for ε=$\{1, 10\}$. On MNIST and Fashion-MNIST, we use the $\mathbf{v}$-prediction config for $\varepsilon = 0.2$ (not applicable to CelebA).

DP-SGD training. In all settings, we use 300 epochs and clipping constant C=1. We use batch size B=4096 for MNIST and Fashion-MNIST and decrease the batch size of CelebA to B=2048 for the sole purpose of fitting the entire batch into GPU memory. The DP noise σ_{DP} values for each setup can be found in Table 4.9

DM Sampling. We experiment with different DM solvers in this work. We found the DDIM sampler [258] (in particular the stochastic version), which does not have any hyperparameters (without guidance), to perform well across all settings. Using the Churn sampler [141], we could improve perceptual quality (measured in FID), however, out of the five (four without guidance) hyperparameters, we only found two (one without guidance) to improve results significantly. We show results for all samplers in Section 4.4.6.5.

4.4.4 Variance Reduction via Noise Multiplicity

As discussed in Section 4.3.3.2, we introduce *noise multiplicity* to reduce gradient variance.

4.4.4.1 Proof of Theorem 1

Theorem. *The variance of the DM objective (Equation (4.7)) decreases with increased noise multiplicity K as $1/K$.*

Proof. The DM objective in Equation (4.7) is a Monte Carlo estimator of the true intractable L_2-loss in Equation (4.3), using one data sample $\mathbf{x}_i \sim p_{\text{data}}$ and K noise-level-noise tuples $\{(\sigma_{ik}, \mathbf{n}_{ik})\}_{k=1}^{K} \sim p(\sigma)\mathcal{N}(\mathbf{0}, \sigma^2)$. Replacing expectations with Monte Carlo estimates is a common practice to ensure numerical tractability. For a generic function r over distribution $p(\mathbf{k})$, we have $\mathbb{E}_{p(\mathbf{k})}[r(\mathbf{k})] \approx \frac{1}{K}\sum_{i=1}^{K} r(\mathbf{k}_i)$, where $\{\mathbf{k}_i\}_{i=1}^{K} \sim p(\mathbf{k})$ (Monte Carlo estimator for expectation of function r with respect to distribution p). The Monte Carlo estimate is a noisy unbiased estimator of the expectation $\mathbb{E}_{p(\mathbf{k})}[r(\mathbf{k})]$ with variance $\frac{1}{K}\text{Var}_p[r]$, where $\text{Var}_p[r]$ is the variance of r itself. This is a well-known fact; see for example Chapter 2 of the excellent book by Owen [217]. This proves that the variance of the DM objective in Equation (4.7) decreases with increased noise multiplicity K as $1/K$. □

4.4.4.2 Variance Reduction Experiment

In this subsection, we empirically show how the reduced variance of the DM objective from noise multiplicity leads to reduced gradient variance during training. In particular, we set $\mathbf{x}_i$ to a randomly sampled MNIST image and set the denoiser $D_{\boldsymbol{\theta}}$ to our trained model on MNIST. We then compute gradients for different noise multiplicities K. We resample the noise values 1k times (for each K) to estimate the variance of the gradient for each parameter. In Figure 4.6, we show the histogram over gradient variance as well as the average gradient variance (averaged over all parameters in the model). Note that the variance of each gradient is a random variable itself (which is estimated using 1k Monte Carlo samples). We find that an increased K leads to significantly reduced training parameter gradient variance.

4.4.4.3 Computational Cost of Noise Multiplicity

In terms of computational cost, noise multiplicity is expensive and likely not useful for non-DP DMs. The computational cost increases linearly with K, as the denoiser needs to run K times. Furthermore, in theory, noise multiplicity increases the memory footprint by at least $\mathcal{O}(K)$; however, in common DP frameworks, such as Opacus [306], which we use, the peak memory requirement is $\mathcal{O}(K^2)$ compared to non-private training. Recent methods such as *ghost clipping* [32] require less memory, but are currently not widely implemented.

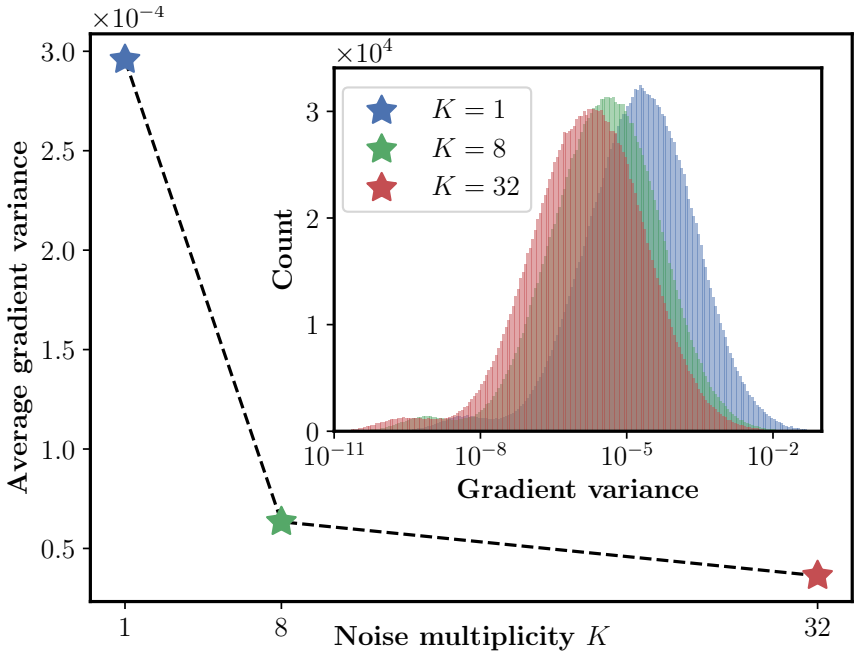


Figure 4.6: Variance reduction via noise multiplicity. Increasing K in *noise multiplicity* leads to significant variance reduction of parameter gradient estimates during training (note logarithmic axis in inset). This is an enlarged version of Figure 4.3.

That being said, in DP generative modeling, and DP machine learning more generally, computational cost is hardly ever the bottleneck; the main bottleneck is the privacy restriction. The privacy restriction implies only a finite number of training iterations and we need to use that budget of training iterations in the most efficient way (this is, using training gradients that suffer from as little noise as possible). Our numerical experiments clearly show that noise multiplicity is a technique to shift the privacy-utility trade-off, effectively getting better utility at the same privacy budget, using additional computational cost.

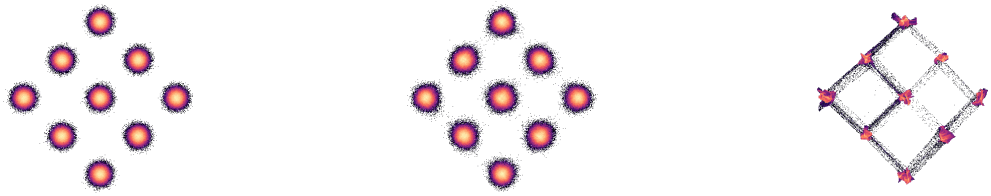
4.4.4.4 On the Difference between Noise Multiplicity and Augmentation Multiplicity

Augmentation multiplicity [64] is a technique where multiple augmentations per image are used to train classifiers with DP-SGD. Image augmentations have also been shown to be potentially helpful in data-limited (image) generative modeling, for example, for autoregressive models [137] and DMs [141]. In stark contrast to discriminative modeling where the data distribution p_{data} can simply be replaced by the *augmented data distribution* and the neural backbone can be left as is, in generative modeling both the loss function and the neural backbone need to be adapted. For example, for DMs [141], the standard DM loss (Equation (4.3)) is formally replaced by

$$\mathbb{E}_{\tilde{\mathbf{x}}\sim p_{\text{data}}(\tilde{\mathbf{x}}),c\sim p_{\text{aug}}(c),\mathbf{x}\sim p_{\text{augdata}}(\mathbf{x}|\tilde{\mathbf{x}},c),(\sigma,\mathbf{n})\sim p(\sigma,\mathbf{n})}\left[\lambda_\sigma \|D_{\boldsymbol{\theta}}(\mathbf{x}+\mathbf{n},\sigma,c)-\mathbf{x}\|_2^2\right], \tag{4.28}$$

where $p_{\text{aug}}(c)$ is the distribution over augmentation choices c (for example, cropping at certain coordinates or other image transformations or perturbations), and $p_{\text{augdata}}(\mathbf{x} \mid \tilde{\mathbf{x}}, c)$ is the distribution over augmented images $\mathbf{x}$ given the original dataset images $\tilde{\mathbf{x}}$ and the augmentation c. Importantly, note that the neural backbone $D_{\boldsymbol{\theta}}$ also needs to be conditioned on the augmentation choice c as, at inference time, we generally only want to generate "clean images" with $c(\mathbf{x}) = \mathbf{x}$ (no augmentation).

While noise multiplicity provably reduces the variance of the standard diffusion loss (see Theorem 1), augmentation multiplicity, that is, averaging over multiple augmentations for a given clean image $\mathbf{x}$, only provably reduces the variance of the augmented diffusion loss, which is by definition more noisy due to the additional expectations. Furthermore, it is not obvious how minimizing the augmented diffusion loss relates to minimizing the true diffusion loss. In contrast to noise multiplicity, augmentation multiplicity *does not* provably reduce the variance of the original diffusion loss; rather, it is a data augmentation



(a) Data p_{data} (b) Samples from DM. (c) Samples from GAN.

Figure 4.7: Mixture of Gaussians: data distribution and (1M) samples from a DM as well as a GAN. Our visualization is based on the log-histogram, which shows single data points as black dots.

technique for enriching training data.

Pointing out the orthogonality of the two ideas again, note that noise multiplicity is still applicable for the above modified augmented diffusion loss objective. Furthermore, we would like to point out that noise multiplicity is applicable for DPDMs in any domain, beyond images; in contrast, data augmentations need to be handcrafted and may not readily available in all fields.

4.4.5 Toy Experiments

In this subsection, we describe the details of the toy experiment from paragraph **(ii) Sequential denoising** in Section 4.3.3.1. For this experiment, we consider a two-dimensional simple Gaussian mixture model of the form

$$p_{\text{data}}(\mathbf{x}) = \sum_{k=1}^{9} \frac{1}{9} p^{(k)}(\mathbf{x}), \tag{4.29}$$

where $p^{(k)}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k; \sigma_0^2)$ and

$$\begin{aligned}
\boldsymbol{\mu}_1 &= \begin{pmatrix} -a \\ 0 \end{pmatrix}, & \boldsymbol{\mu}_2 &= \begin{pmatrix} -a/2 \\ a/2 \end{pmatrix}, & \boldsymbol{\mu}_3 &= \begin{pmatrix} 0 \\ a \end{pmatrix}, \\
\boldsymbol{\mu}_4 &= \begin{pmatrix} -a/2 \\ -a/2 \end{pmatrix}, & \boldsymbol{\mu}_5 &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, & \boldsymbol{\mu}_6 &= \begin{pmatrix} a/2 \\ a/2 \end{pmatrix}, \\
\boldsymbol{\mu}_7 &= \begin{pmatrix} 0 \\ -a \end{pmatrix}, & \boldsymbol{\mu}_8 &= \begin{pmatrix} a/2 \\ -a/2 \end{pmatrix}, & \boldsymbol{\mu}_9 &= \begin{pmatrix} a \\ 0 \end{pmatrix},
\end{aligned}$$

where $\sigma_0 = 1/25$ and $a = 1/\sqrt{2}$. The data distribution is visualized in Figure 4.7a.

Fitting. Initially, we fitted a DM as well as a GAN to the mixture of Gaussians. The neural networks of the DM and the GAN generator use similar ResNet architectures with 267k and 264k (1.1% smaller) parameters, respectively (see Section 4.4.5.1 for training details). The fitted distributions are visualized in Figure 4.7. In this experiment, we use deterministic DDIM (Algorithm 6) [258], a numerical solver for the Probability Flow ODE (Equation (4.1)) [263], with 100 neural function evaluations (DDIM-100) as the end-to-end multi-step synthesis process for the DM. Even though our visualization shows that the DM clearly fits the distribution better (Figure 4.7), the GAN does not do bad either. Note that our visualization is based on the log-histogram of the sampling distributions, and therefore puts significant emphasis on single data point outliers.

We provide a second method to assess the fitting: In particular, we measure the percentage of points (out of 1M samples) that are within a h-standard deviation vicinity of any of the nine modes. A point $\mathbf{x}$ is said to be within a h-standard deviation vicinity of the mode $\boldsymbol{\mu}_k$ if $\|\mathbf{x} - \boldsymbol{\mu}_k\| < h\sigma_0$. We present results for this metric in Table 4.10 for h=$\{1, 2, 3, 4, 5, 6\}$. Note that any mode is at least 12.5 standard deviations separated to the next mode, and therefore no point can be in the h-standard deviation vicinity of more than two modes for $h \leq 6$.

The results in Table 4.10 indicate that the GAN is slightly too sharp, that is, it puts too many points within the 1- and 2-standard deviation vicinity of modes. Moreover, for larger h, the result in Table 4.10 suggests that the samples in Figure 4.7c that appear to "connect" the GAN's modes are heavily overemphasized—these samples actually represent less than 1% of the total samples; 99.3% of samples are within a 4-standard deviation vicinity of a mode while modes are at least 12.5 standard deviations separated.

Complexity. Now that we have ensured that both the GAN as well as the DM fit the target distribution reasonably well, we can measure the complexity of the DM denoiser D, the generator defined by the GAN, as well as the end-to-end multi-step synthesis process (DDIM-100) of the DM. In particular, we measure the complexity of these functions using the Frobenius norm of the Jacobian [75]. In particular, we define

$$\mathcal{J}_F(\sigma) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x},\sigma)} \|\nabla_{\mathbf{x}} D_{\boldsymbol{\theta}}(\mathbf{x}, \sigma)\|_F^2. \tag{4.30}$$

Note that the convolution of a mixture of Gaussian with i.i.d. Gaussian noise is simply

Table 4.10: h-standard deviation vicinity metric as defined in the paragraph **Fitting** of Section 4.4.5.

h	Data	DM	GAN
1	39.4	37.2	56.8
2	86.5	83.3	95.3
3	98.9	97.7	98.9
4	100	99.8	99.3
5	100	100	99.6
6	100	100	99.9

the sum of the convolution of the mixture components, i.e.,

$$p(\mathbf{x};\sigma) = \left(p_{\text{data}} * \mathcal{N}\left(\mathbf{0}, \sigma^2\right)\right)(\mathbf{x}) \tag{4.31}$$

$$= \sum_{k=1}^{9} \frac{1}{9} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k; \sigma_0^2 + \sigma^2). \tag{4.32}$$

We then compare $\mathcal{J}_F(\sigma)$ with the complexity of the GAN generator (S_1) and the end-to-end synthesis process of the DM (S_2). In particular, we define

$$\mathcal{J}_F = \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \|\nabla_{\mathbf{x}} S_i(\mathbf{x})\|_F^2, \quad i \in \{1, 2\}. \tag{4.33}$$

We want to clarify that for S_2 we do not have to backpropagate through an ODE but rather through its discretization, i.e., deterministic DDIM with 100 function evaluations (Algorithm 6), since that is how we define the end-to-end multi-step synthesis process of the DM in this experiment. Furthermore, we chose the latent space of the GAN to be two-dimensional such that $\nabla_{\mathbf{x}} S_i(\mathbf{x}) \in \mathbb{R}^{2\times 2}$ for both the GAN and the DM; this ensures a fair comparison. The final complexities are visualized in Figure 4.2.

4.4.5.1 Training Details

DM training. Training the diffusion model is very simple. We use the EDM config and train for 50k iterations (with batch size B=256) using Adam with learning rate $3 \cdot 10^{-4}$. We use an EMA rate of 0.999.

GAN training. Training GANs on two-dimensional mixture of Gaussians is notoriously difficult (see, for example, Sec. 5.1 in [302]). We experimented with several setups and

found the following to perform well: We train for 50k iterations (with batch size B=256) using Adam with learning rate $3 \cdot 10^{-4}$ and (β_1=0.0, $\beta_2 = 0.9$) for both the generator and the discriminator. Following Yazıcı et al. [302], we use EMA (rate of 0.999 as in the DM). We found it crucial to make the discriminator bigger than the generator; in particular, we use twice as many hidden layers in the discriminator's ResNet. Furthermore, we use `ReLU` and `LeakyReLU` for the generator and the discriminator, respectively.

4.4.6 Image Experiments

4.4.6.1 Evaluation Metrics, Baselines, and Datasets

Metrics. We measure sample quality via Fréchet Inception Distance (FID) [105]. We follow the DP generation literature and use 60k generated samples. The particular Inception-v3 model used for FID computation is taken from Karras et al. [140][5]. On MNIST and Fashion-MNIST, we follow the standard procedure of repeating the channel dimension three times before feeding images into the Inception-v3 model.

On MNIST and Fashion-MNIST, we additionally assess the utility of generated data by training classifiers on synthesized samples and compute class prediction accuracy on real data. Similar to previous works, we consider three classifiers: logistic regression (Log Reg), MLP, and CNN classifiers. The model architectures are taken from the `DP-Sinkhorn` repository [38].

For downstream classifier training, we follow the DP generation literature and use 60k synthesized samples. We follow Cao et al. [38] and split the 60k samples into a training set (90%) and a validation set (remaining 10%). We train all models for 50 epochs, using Adam with learning rate $3 \cdot 10^{-4}$. We regularly save checkpoints during training and use the checkpoint that achieves the best accuracy on the validation split for final evaluation. Final evaluation is performed on real, non-synthetic data. We train all models for 50 epochs, using Adam with learning rate $3 \cdot 10^{-4}$.

Note that we chose to only use 60k synthesized samples to follow prior work and therefore be able to compare to baselines in a fair manner. That being said, during this project we did explore training classifiers with more samples but did not find any significant improvements in downstream accuracy. We hypothesize that 60k samples are enough to accurately

[5] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/metrics/inception-2015-12-05.pkl

represent the underlying learned distribution by the DPDM and to train good classifiers on MNIST/FashionMNIST. We believe that a more detailed study on how many samples are needed to get a certain accuracy is an interesting avenue for future work.

Baselines. We run baseline experiments for PEARL [173]. In particular, we train models for ε=$\{0.2, 1, 10\}$ on MNIST and Fashion-MNIST. We confirmed that our models match the performance reported in their paper. In fact, our models perform slightly better (in terms of the LeNet-FID metric Liew et al. [173] uses). We then follow the same evaluation setup (see **Metrics** above) as for our DPDMs. Most importantly, we use the standard Inception network-based FID calculation, similarly as most works in the (DP) image generative modeling literature.

Datasets. We use three datasets in our main experiments: MNIST [164], Fashion-MNIST [293] and CelebA [181]. Furthermore, we provide initial results on CIFAR-10 [160] and ImageNet [65] (as well as CelebA on a higher resolution); see Section 4.4.7.

We would like to point out that these dataset may contain multiple images per identity (e.g. person, animal, etc.), whereas our method, as well as all other baselines in this work, considers the per-image privacy guarantee. For an identity with k images in the dataset, a model with (ε, δ) per-image DP affords $(k\varepsilon, ke^{(k-1)\varepsilon}\delta)$-DP to the individual according to the Group Privacy theorem [84]. We leave a more rigorous study of DPDMs with Group Privacy to future research and note that these datasets currently simply serve as benchmarks in the community. Nonetheless, we believe that it is important to point out that these datasets do not necessarily serve as a realistic test bed for per-image DP generative models in privacy critical applications.

4.4.6.2 Computational Resources

For all experiments, we use an in-house GPU cluster of V100 NVIDIA GPUs. On eight GPUs, models on MNIST and Fashion-MNIST trained for roughly one day and models on CelebA for roughly four days. We tried to maximize performance by using a large number of epochs, which results in a good privacy-utility trade-off, as well as high noise multiplicity; this results in relatively high training time (when compared to existing DP generative models). Using a smaller noise multiplicity K decreases computation, although generally at the cost of model performance; see also Section 4.4.4.3.

Table 4.11: Noise multiplicity ablation on MNIST and Fashion-MNIST.

K	MNIST				Fashion-MNIST			
	FID	Acc (%)			FID	Acc (%)		
		Log Reg	MLP	CNN		Log Reg	MLP	CNN
1	76.9	84.2	87.5	91.7	72.5	76.0	76.3	75.9
2	60.1	84.8	88.3	93.1	61.4	76.7	77.0	77.4
4	57.1	85.2	88.0	92.8	61.1	76.7	77.2	77.0
8	44.8	86.2	89.2	94.1	58.2	75.2	76.3	77.4
16	36.9	86.0	89.8	94.2	58.5	77.0	77.4	78.8
32	34.8	86.8	90.1	94.4	57.7	76.4	77.0	77.1

4.4.6.3 Training DP-SGD Classifiers

We train classifiers on MNIST and Fashion-MNIST using DP-SGD directly. We follow the setup used for training DPDMs, in particular, batchsize $B = 4096$, 300 epochs and clipping constant $C = 1$. Recently, De et al. [64] found EMA to be helpful in training image classifiers: we follow this suggestion and use an EMA rate of 0.999 (same rate as used for training DPDMs).

4.4.6.4 Extended Quantitative Results

In this subsection, we show additional quantitative results not presented in the main paper. In particular, we present extended results for all ablation experiments.

4.4.6.4.1 Noise Multiplicity In the main paper, we present noise multiplicity ablation results on MNIST with ε=1 (Table 4.5). All results for MNIST and Fashion-MNIST on all three privacy settings (ε=$\{0.2, 1, 10\}$) can be found in Table 4.11.

4.4.6.4.2 Diffusion Model Config In the main paper, we present DM config ablation results on MNIST with ε=0.2 (Table 4.5). All results for MNIST and Fashion-MNIST on all three privacy settings (ε=$\{0.2, 1, 10\}$) can be found in Table 4.12.

Table 4.12: DM config ablation.

Method	DP-ε	MNIST				Fashion-MNIST			
		FID	Acc (%)			FID	Acc (%)		
			Log Reg	MLP	CNN		Log Reg	MLP	CNN
VP [263]	0.2	197	23.1	25.5	24.2	146	49.7	51.6	51.7
VE [263]	0.2	171	17.9	15.4	13.9	178	22.2	27.9	49.4
V-prediction [244]	0.2	97.8	80.2	81.3	84.4	115	71.3	70.9	71.8
EDM [141]	0.2	119	62.4	67.3	49.2	93.5	64.7	65.9	66.6
VP [263]	1	82.2	59.4	69.3	72.6	73.4	68.3	70.4	72.7
VE [263]	1	165	17.9	20.5	26.0	156	30.7	36.0	49.8
V-prediction [244]	1	34.8	86.8	90.1	94.4	57.7	76.4	77.0	77.1
EDM [141]	1	34.2	86.2	90.1	94.9	47.1	77.4	78.0	79.4
VP [263]	10	12.3	88.8	94.1	97.0	22.3	81.2	81.6	84.5
VE [263]	10	88.6	48.0	56.9	63.8	83.2	69.0	70.4	75.4
V-prediction [244]	10	7.65	90.4	94.4	97.7	23.1	82.0	83.7	85.5
EDM [141]	10	6.13	90.4	94.6	97.5	17.4	82.6	84.1	86.2

4.4.6.4.3 Diffusion Sampler Grid Search and Ablation Churn sampler grid search. We run a small grid search for the hyperparameters of the Churn sampler (together with the guidance weight w for classifier-free guidance). For MNIST and Fashion-MNIST on ε=0.2 we run a two-stage grid search. Using S_{min}=0.05, S_{max}=50, and S_{noise}=1, which we found to be sensible starting values, we ran an initial grid search over the guidance scale w=$\{0, 0.125, 0.25, 0.5, 1.0, 2.0\}$ and S_{churn}=$\{0, 5, 10, 25, 50, 100, 150, 200\}$, which we found to be the two most critical hyperparameters of the Churn sampler. Afterwards, we ran a second grid search over S_{noise}=$\{1, 1.005\}$, S_{min}=$\{0.01, 0.02, 0.05, 0.1, 0.2\}$, and S_{max}=$\{10, 50, 80\}$ using the best (w, S_{churn}) setting for each of the two models. For MNIST and Fashion-MNIST on ε=$\{1, 10\}$, we ran a single full grid search over the guidance scale w=$\{0, 0.25, 0.5, 1.0, 2.0\}$, S_{churn}=$\{10, 25, 50, 100\}$, and S_{min}=$\{0.025, 0.05, 0.1, 0.2\}$ while setting S_{noise}=1. For CelebA, on both ε=1 and ε=10, we also ran a single full grid search over S_{churn}=$\{50, 100, 150, 200\}$, and S_{min}=$\{0.005, 0.05\}$ while setting S_{noise}=1. The best settings for FID metric and downstream CNN accuracy can be found in Table 4.14 and Table 4.15, respectively.

Throughout all experiments we found two consistent trends that are listed in the following:

- If optimizing for FID, set S_{churn} relatively high and S_{min} relatively small. Increase S_{churn} and decrease S_{min} as ε is decreased.

Table 4.13: Diffusion sampler comparison. We compare the Churn sampler [141] to stochastic and determistic DDIM [258].

Sampler	DP-ε	MNIST				Fashion-MNIST			
		FID	Acc (%)			FID	Acc (%)		
			Log Reg	MLP	CNN		Log Reg	MLP	CNN
Churn (FID)	0.2	**61.9**	65.3	65.8	71.9	**78.4**	53.6	55.3	57.0
Churn (Acc)	0.2	104	81.0	81.7	**86.3**	128	70.4	71.3	**72.3**
Stochastic DDIM	0.2	97.8	80.2	81.3	84.4	115	71.3	70.9	71.8
Deterministic DDIM	0.2	120	**81.3**	**82.1**	84.8	132	**71.5**	**71.6**	71.8
Churn (FID)	1	**23.4**	83.8	87.0	93.4	**37.8**	71.5	71.7	73.6
Churn (Acc)	1	35.5	**86.7**	91.6	**95.3**	51.4	76.3	76.9	**79.4**
Stochastic DDIM	1	34.2	86.2	90.1	94.9	47.1	77.4	78.0	**79.4**
Deterministic DDIM	1	50.4	85.7	**91.8**	94.9	60.6	**77.5**	**78.2**	78.9
Churn (FID)	10	**5.01**	90.5	94.6	97.3	18.6	80.4	81.1	84.9
Churn (Acc)	10	6.65	**90.8**	94.8	**98.1**	19.1	81.1	83.0	**86.2**
Stochastic DDIM	10	6.13	90.4	94.6	97.5	**17.4**	**82.6**	**84.1**	**86.2**
Deterministic DDIM	10	10.9	90.5	**95.2**	97.7	19.7	81.9	83.9	**86.2**

- If optimizing for downstream accuracy, set S_{churn} relatively small and S_{min} relatively high.

Sampling ablation. In the main paper, we present a sampler ablation for MNIST (Table 4.6). Results for Fashion-MNIST (as well as) MNIST can be found in Table 4.13.

4.4.6.4.4 Distribution Matching Analysis We perform a distribution matching analysis on MNIST using the CNN classifier, that is, computing per-class downstream accuracies at different privacy levels. In Table 4.16, we can see that with increased privacy (lower ε) classification performance degrades roughly similar for most digits, implying that our DPDMs learn a well-balanced distribution and cover all modes of the data distribution faithfully even under strong privacy. The only result that stands out to us is that class 1 appears significantly easier than all other classes for ε=0.2. However, that may potentially be due to the class 1 in MNIST being a "line" which may be easy to classify by a CNN even if training data is noisy.

Table 4.14: Best Churn sampler settings for FID metric.

Parameter	MNIST			Fashion-MNIST			CelebA	
	ε=0.2	ε=1	ε=10	ε=0.2	ε=1	ε=10	ε=1	ε=10
w	1	0	0.25	2	1	0.25	N/A	N/A
S_{churn}	200	100	50	150	50	25	200	50
S_{min}	0.01	0.05	0.05	0.02	0.025	0.2	0.005	0.005
S_{max}	50	50	50	10	50	50	50	50
S_{noise}	1	1	1	1	1	1	1	1

Table 4.15: Best Churn sampler settings for downstream CNN accuracy.

Parameter	MNIST			Fashion-MNIST		
	ε=0.2	ε=1	ε=10	ε=0.2	ε=1	ε=10
w	0.125	0	0	0.125	0	0
S_{churn}	10	10	10	5	10	10
S_{min}	0.2	0.1	0.025	0.02	0.025	0.1
S_{max}	10	50	50	80	50	50
S_{noise}	1.005	1	1	1.005	1	1

Table 4.16: Distribution matching analysis for MNIST using downstream CNN accuracy.

Class	0	1	2	3	4	5	6	7	8	9
ε=10	98.5	98.7	98.7	98.4	99.1	99.0	98.2	97.7	98.1	96.1
ε=1	95.4	98.9	96.7	96.1	96.1	97.3	96.6	91.3	92.3	93.2
ε=0.2	81.9	96.0	80.2	82.3	87.3	85.2	90.7	86.9	83.2	83.5

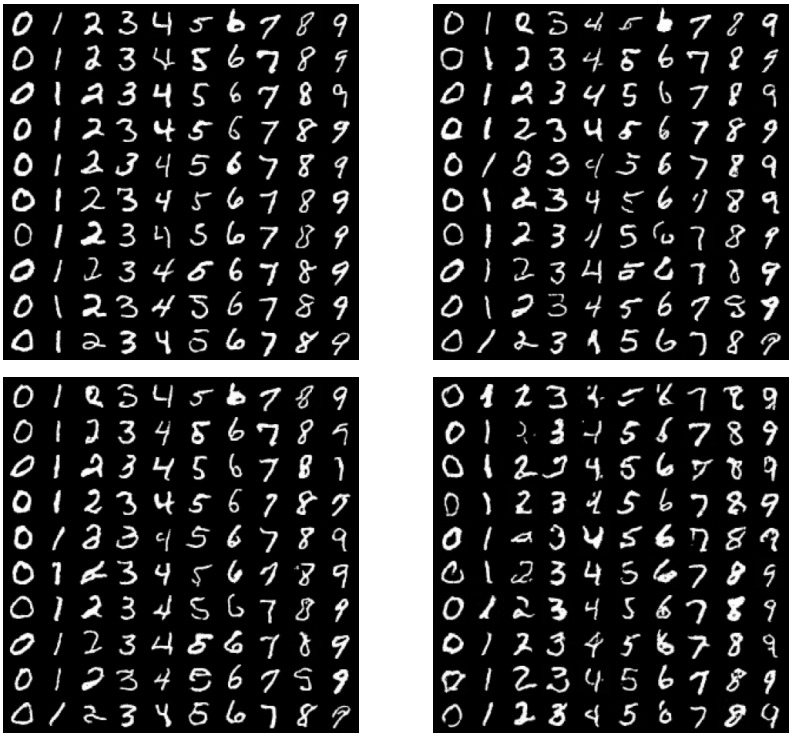


Figure 4.8: Additional images generated by DPDM on MNIST for ε=10 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

4.4.6.5 Extended Qualitative Results

In this subsection, we show additional generated samples by our DPDMs. On MNIST, see Figure 4.8, Figure 4.9, and Figure 4.10 for ε=10, ε=1, and ε=0.2, respectively. On Fashion-MNIST, see Figure 4.11, Figure 4.12, and Figure 4.13 for ε=10, ε=1, and ε=0.2, respectively. On CelebA, see Figure 4.14 and Figure 4.15 for ε=10 and ε=1, respectively. For a visual comparison of our CelebA samples to other works in the literature, see Figure 4.16.

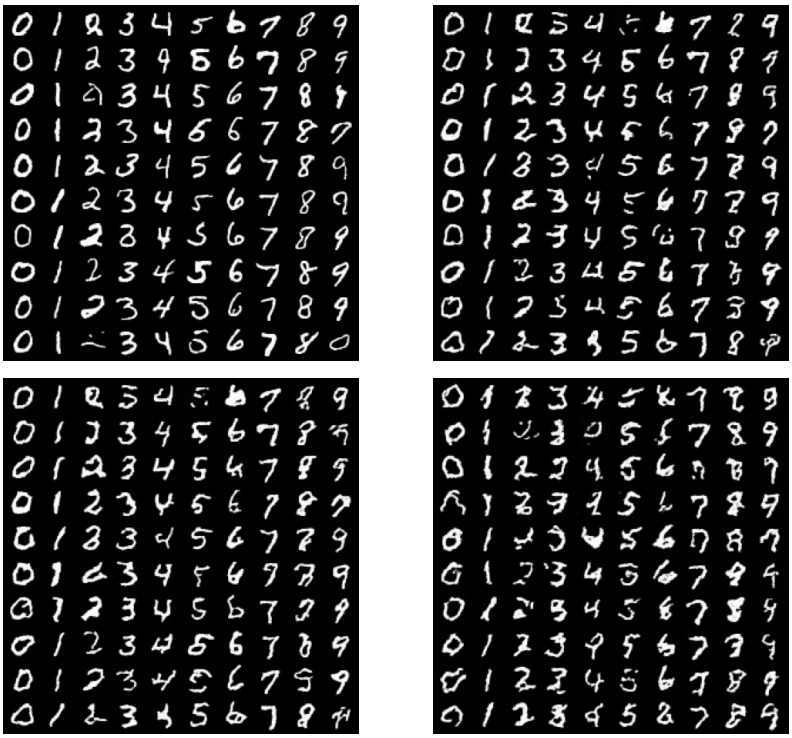


Figure 4.9: Additional images generated by DPDM on MNIST for ε=1 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

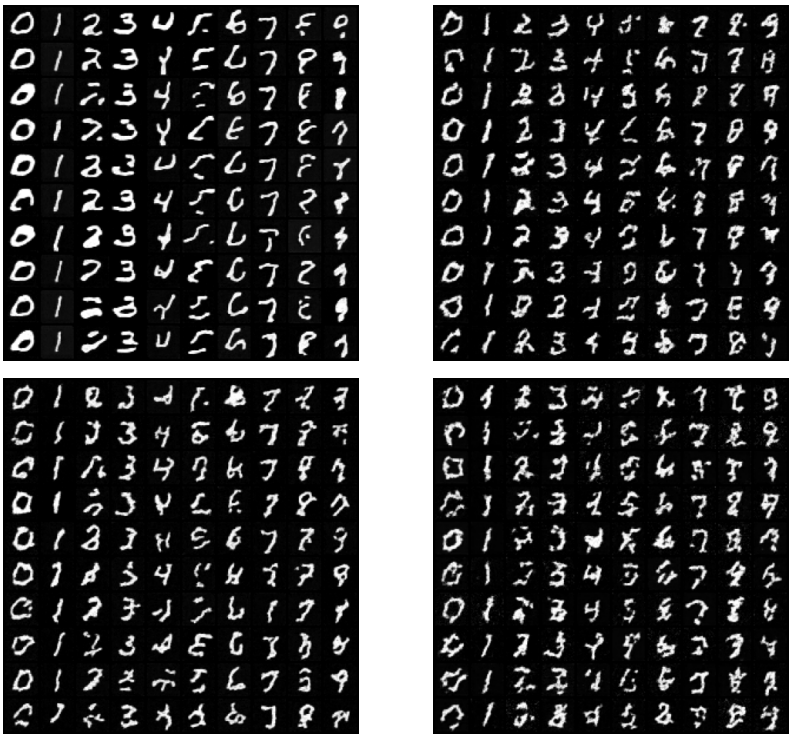


Figure 4.10: Additional images generated by DPDM on MNIST for ε=0.2 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

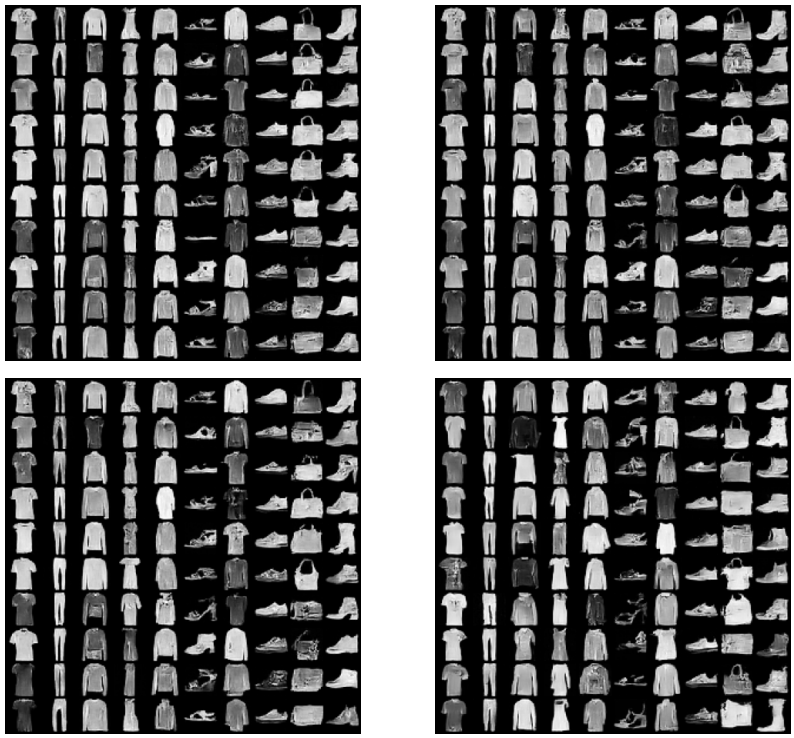


Figure 4.11: Additional images generated by DPDM on Fashion-MNIST for ε=10 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

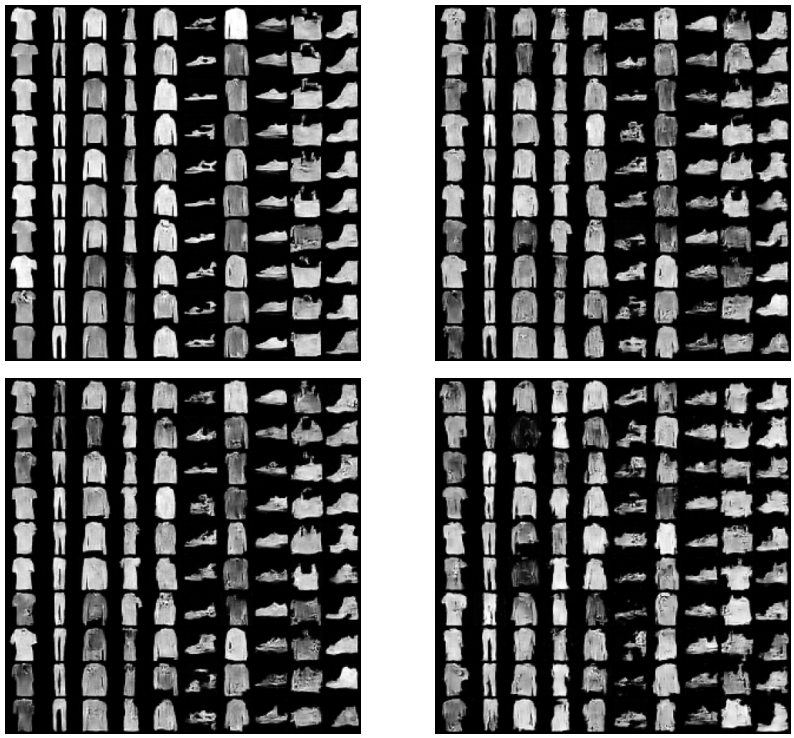


Figure 4.12: Additional images generated by DPDM on Fashion-MNIST for ε=1 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

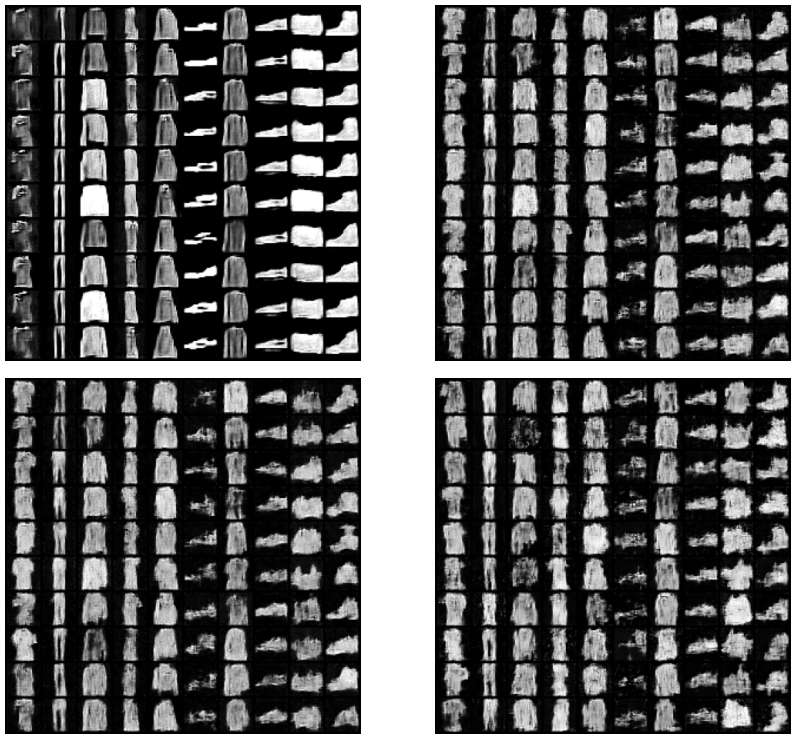


Figure 4.13: Additional images generated by DPDM on Fashion-MNIST for ε=0.2 using Churn (FID) (*top left*), Churn (Acc) (*top right*), stochastic DDIM (*bottom left*), and deterministic DDIM (*bottom right*).

Figure 4.14: Additional images generated by DPDM on CelebA for ε=10 using Churn (*top*), stochastic DDIM (*middle*), and deterministic DDIM (*bottom*).

Figure 4.15: Additional images generated by DPDM on CelebA for ε=1 using Churn (*top*), stochastic DDIM (*middle*), and deterministic DDIM (*bottom*).

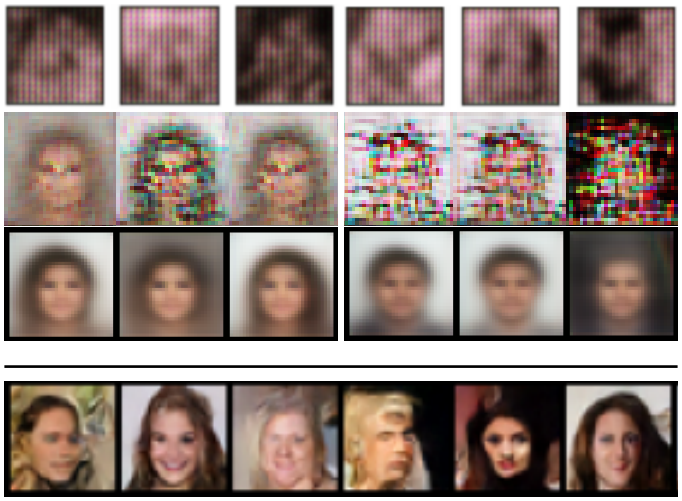


Figure 4.16: CelebA images generated by **DataLens** (*1st row*), **DP-MEPF** (*2nd row*), **DP-Sinkhorn** (*3rd row*), and our **DPDM** (*4th row*) for DP-ε=10.

4.4.7 Additional Experiments on More Challenging Problems

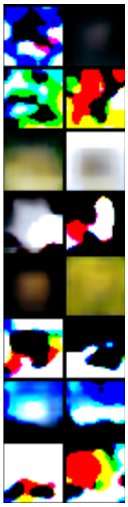
4.4.7.1 Diverse Datasets

We provide results for additional experiments on challenging diverse datasets, namely, CIFAR-10 [160] and ImageNet [65] (resolution 32x32), both in the class-conditional setting similar to our other experiments on MNIST and Fashion-MNIST. To the best of our knowledge, we are the first to attempt pure DP image generation on ImagenNet.

For both experiments, we use the same neural network architecture as for CelebA (32x32) in the main paper; see model hyperparameters in Table 4.8. On CIFAR-10, we train for 500 epochs using noise multiplicity $K = 32$ under the privacy setting ($\varepsilon = 10, \delta = 10^{-5}$). In ImageNet, we train for 100 epochs using noise multiplicity $K = 8$ under the privacy setting ($\varepsilon = 10, \delta = 7 \cdot 10^{-7}$); training for longer (or using larger K) was not possible on ImageNet due to its sheer size. We achieve FIDs of 97.7 and 61.3 for CIFAR-10 and ImageNet, respectively. No previous works reported FID scores on these datasets and for these privacy settings, but we hope that our scores can serve as reference points for future work. In Figure 4.17, we show samples for both datasets from our DPDMs and visually compare to an existing DP generative modeling work on CIFAR-10, DP-MERF [102]. Our DPDMs cannot learn clear objects; however, overall image/pixel statistics seem to be captured correctly. In contrast, the DP-MERF baseline collapses entirely. We are not

(a) DPDM (*ours*) (ImageNet)

(b) DPDM (*ours*) (CIFAR-10)



(c)

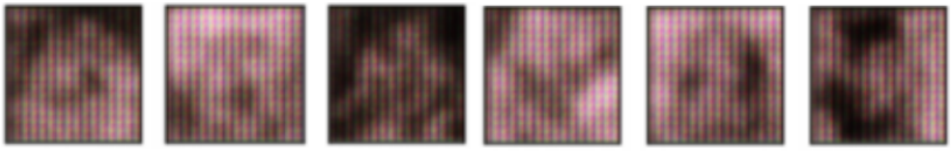
Figure 4.17: Additional experiments on challenging diverse datasets. Samples from our DPDM on ImageNet and CIFAR-10, as well as CIFAR-10 samples from DP-MERF [102] in (c).

aware of any other works tackling these tasks. Hence, we believe that DPDMs represent a major step forward.

4.4.7.2 Higher Resolution

We provide results for additional experiments on CelebA at higher resolution (64x64). To accommodate the higher resolution, we added an additional upsampling/downsampling layer to the U-Net, which results in roughly a 11% increase in the number of parameters, from 1.80M to 2.00M parameters. The only row that changed in the CelebA model hyperparameter table (Table 4.8) is the one about the channel multipliers. It is adapted from (1,2,2) to (1,2,2,2). We train for 300 epochs using $K = 8$ under the privacy setting ($\varepsilon = 10, \delta = 10^{-6}$). We achieve an FID of 78.3 (again, for reference; no previous works reported quantitative results on this task). In Figure 4.18, we show samples and visually compare to existing DP generative modeling work on CelebA at 64x64 resolution. Although the faces generated by our DPDM are somewhat distorted, the model overall is able to clearly generate face-like structures. In contrast, DataLens generates incoherent very low quality outputs. No other existing works tried generating 64x64 CelebA images with rigorous DP guarantees, to the best of our knowledge. Also this experiment implies

(a) DPDM (*ours*)



(b) DataLens [284]

Figure 4.18: Additional experiments on CelebA at higher resolution (64x64). Samples from our method and DataLens [284].

that DPDMs can be considered a major step forward for DP generative modeling.

4.4.8 Ethics, Reproducibility, Limitations and Future Work

Our work improves the state-of-the-art in differentially private generative modeling and we validate our proposed DPDMs on image synthesis benchmarks. Generative modeling of images has promising applications, for example for digital content creation and artistic expression [15], but it can in principle also be used for malicious purposes [201, 210, 278]. However, differentially private image generation methods, including our DPDM, are currently not able to produce photo-realistic content, which makes such abuse unlikely.

As discussed in Section 4.3.1, a severe issue in modern generative models is that they can easily overfit to the data distribution, thereby closely reproducing training samples and leaking privacy of the training data. Our DPDMs aim to rigorously address such problems via the well-established DP framework and fundamentally protect the privacy of the training data and prevent overfitting to individual data samples. This is especially important when training generative models on diverse and privacy-sensitive data. Therefore, DPDMs can potentially act as an effective medium for data sharing without needing to worry about data privacy, which we hope will benefit the broader machine learning community. Note, however, that although DPDM provides privacy protection in generative

learning, information about individuals cannot be eliminated entirely, as no useful model can be learned under DP-(ε=0, δ=0). This should be communicated clearly to dataset participants.

An important question for future work is scaling DPDMs to *(a)* larger datasets and *(b)* more complicated, potentially higher-resolution image datasets. Regarding *(a)*, recall that an increased number of data points leads to less noise injection during DP-SGD training (while keeping all other parameters fixed). Therefore, we believe that DPDMs should scale well with respect to larger datasets; see, for example, our results on ImageNet, a large dataset, which are to some degree better than the results on CIFAR-10, a relatively small dataset (Section 4.4.7). Regarding *(b)*, however, scaling to more complicated, potentially higher-resolution datasets is challenging if the number of data points is kept fixed. Higher-resolution data requires larger neural networks, but this comes with more parameters, which can be problematic during DP training (see Section 4.3.3.2). Using parameter-efficient architectures for diffusion models may be promising; for instance, see concurrent work by [126]. Generally, we believe that scaling both *(a)* and *(b)* are interesting avenues for future work.

To aid reproducibility of the results and methods presented in our paper, we made source code to reproduce all quantitative and qualitative results of the paper publicly available, including detailed instructions. Moreover, all training details and hyperparameters are already described in detail in this appendix, in particular in Section 4.4.3.

4.5 Epilogue

4.5.1 Public Pre-training of Differentially Private Diffusion Models

Pre-training on public datasets is a common technique to improve the performance of DP models [103, 307]. Recently, Ghalebikesabi et al. [95] propose public pre-training of DMs to improve the performance of DPDMs. In particular, they perform public pre-training on ImageNet and private DPDM fine-tuning on CIFAR-10 and Camelyon17 [155]. Compared to from-scratch DPDM training, Ghalebikesabi et al. [95] adapt the time distribution $p(t)$: they argue that the pre-trained DM transfers well to new tasks for very small and very large noise levels. In contrast, the "medium" noise levels differ considerably between datasets. Therefore, they adjust $p(t)$ and "focus" on intermediate times ($t \in [0.3, 0.6]$, with $T = 1.0$).

Furthermore, Ghalebikesabi et al. [95] extend our *noise multiplicity* (Equation (7) of the paper) by also averaging the per-sample DM loss over traditional augmentation strategies such as *random cropping* and *horizontal flipping*.

Chapter 5

Distilling the Knowledge in Diffusion Models

5.1 Taming Large Models with Knowledge Distillation

Recent text-to-image DMs have attracted much attention, being able to generate high-resolution photorealistic samples [230, 235, 242]. This powerful performance, however, requires large neural network backbones with billions of parameters. Networks of this size generally require a large amount of GPU memory and they are slow at inference time, making it difficult to deploy them in real-time or on resource-limited devices.

In Chapter 3, we have discussed a multitude of methods that accelerate the inference of DMs, however, none of those methods address the issue of requiring a large amount of GPU memory. In discriminative modeling, *knowledge distillation* is a widely used method that addresses this particular issue: For example, the knowledge of a (teacher) classifier can be distilled into a smaller (student) model by training the student on the (soft) teacher predictions rather than the (hard) labels of the data [106]. Knowledge distillation has been widely used in discriminative modeling [25, 101, 106, 271, 297], but only rarely in generative modeling [3]. One potential reason for this may be that the objective functions of popular one-shot generative models such as generative adversarial networks and normalizing flow do not naturally extend to knowledge distillation.

5.2 Preface

This section presents the workshop paper "Distilling the Knowledge in Diffusion Models". In this work, we distill the knowledge of a large pre-trained DM into a small DM using an (approximate) score matching objective:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0),\, t \sim p(t),\, \mathbf{n} \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)} \left[\lambda(t) \| D_{\boldsymbol{\theta}}(\alpha_t \mathbf{x}_0 + \sigma_t \mathbf{n}, t) - D_{\boldsymbol{\phi}}(\alpha_t \mathbf{x}_0 + \sigma_t \mathbf{n}, t) \|_2^2 \right], \quad (5.1)$$

where $D_{\boldsymbol{\phi}}$ is the large teacher model and $D_{\boldsymbol{\theta}}$ the smaller student model. Similar to knowledge distillation in classifiers, the "hard" predictions $\mathbf{x}_0$ in the DM objective (Equation (1.27)) are replaced by the "soft" teacher predictions $D_{\boldsymbol{\phi}}(\alpha_t \mathbf{x}_0 + \sigma_t \mathbf{n}, t)$. Furthermore, the above distillation objective is a natural modification of score matching [123], replacing the true data score by the teacher score model. The paper was accepted and presented at the CVPR 2023 workshop for "Generative Models for Computer Vision" in June 2023.

Contributions: I was the sole first author of this work. Robin Rombach, Andreas Blattmann, and Yaoliang Yu were co-authors, and Yaoliang supervised the project. The knowledge distillation idea was conceived jointly between myself and Yaoliang. I implemented all code and ran all experiments on CIFAR-10. Throughout the project, I contacted Robin and Andreas to apply the method on Stable Diffusion [235]. I wrote the paper myself.

Differences in notation: For simplicity, we only consider DMs that use the perturbation kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_0, \sigma_t^2 \boldsymbol{I}_d)$.

Abstract: Large-scale diffusion models have achieved unprecedented results in (conditional) image synthesis, however, they generally require a large amount of GPU memory and are slow at inference time. To overcome this limitation, we propose to distill the knowledge of pre-trained (teacher) diffusion models into smaller student diffusion models via an approximate score matching objective. For classifier-free guided generation on CIFAR-10, our student model achieves a FID-5K of 8.03 using 273G flops. In comparison, the larger teacher model only achieves a FID-5K of 294 using 424G flops. We present initial experiments on distilling the knowledge of Stable Diffusion, a large scale text-to-image diffusion model, and discuss several promising future directions.

(a) s: 129G flops, 5 steps (b) s: 273G flops, 10 steps (c) t: 424G flops, 3 steps (d) t: 764G flops, 5 steps (e) t: 1612G flops, 10 steps

Figure 5.1: Guided image generation on CIFAR-10 (guidance strength w=0.5) with student model s and teacher model t. The student model needs considerably less flops to achieve similar performance.

5.3 Main Paper

5.3.1 Introduction

Diffusion models (DMs) achieve both state-of-the-art synthesis quality and sample diversity using an iterative sampling process. In computer vision DMs have been used for (conditional) image [66, 109, 110, 211, 230, 235] and (conditional) video [112, 252, 301] synthesis, super-resolution [168, 240], deblurring [142, 290], image editing and inpainting [185, 197, 235, 239], conditional and semantic image generation [22, 56, 179, 218, 228], image-to-image translation [239, 249, 267], inverse problems in medical imaging [59, 60, 117, 187, 224, 264, 299], and differentially private image synthesis [73, 95].

In particular large-scale text-to-image DMs have recently gained a lot of attention, being able to synthesize high-resolution photorealistic images [230, 235, 242]. To achieve this powerful performance these DMs rely on neural network backbones with billions of parameters [230, 242]. Networks of this size require a large amount of GPU memory and they are slow at inference time, making it difficult to deploy them in real-time or on resource-limited devices.

The issue of slow inference has, for example, been addressed by the development of faster DM samplers [74, 75, 183, 184, 258]. Another promising approach to tackle this issue is to distill the entire iterative sampling process of a (*teacher*) DM into a *student* (sampling) model [24, 196, 244, 265]; we refer to this approach as *sampling distillation*. To accelerate training, the student network is initialized from the teacher, and therefore student

and teacher have the same number of parameters. After distillation, the student model can synthesize samples using only a few network evaluations rather than tens of network evaluations needed for the teacher model. Sampling distillation reduces the inference time while keeping the required GPU memory constant.

In this work, we instead propose to distill the knowledge (rather than the iterative sampling process) of a teacher DM into a *smaller* student DM, that is, the student should learn to match the predictions of the teacher for *any input*. Compared to sampling distillation, reducing the network size of a DM results in less GPU memory as well as faster inference: though the student model may still require tens of evaluations for sampling, each evaluation is significantly faster. Knowledge distillation (KD) [106] has been widely used in discriminative modeling [25, 101, 106, 271, 297], but only rarely in generative modeling [3]. We propose a robust approximate score matching objective to perform KD.

We thoroughly evaluate our proposed method on CIFAR-10 [160] and find that we can drastically decrease the required inference time of students model compared to their teachers; see Figure 5.1. Furthermore, we show early results on distilling Stable Diffusion [235], a large text-to-image *latent* DM; see Section 5.3.5. We envision that our framework, which can potentially be combined with orthogonal ideas such as fast DM samplers and sampling distillation, paves the way towards fast and high-resolution synthesis of DMs on resource limited devices.

5.3.2 Background

We consider continuous-time DMs [263] and follow the presentation of Karras et al. [141]. Let $p_{\text{data}}(\mathbf{x}_0)$ denote the data distribution and $p(\mathbf{x};\sigma)$ be the distribution obtained by adding i.i.d. σ^2-variance Gaussian noise to the data distribution. For sufficiently large $\sigma_{\max}$, $p(\mathbf{x};\sigma^2_{\max})$ is almost indistinguishable from $\sigma^2_{\max}$-variance Gaussian noise. Capitalizing on this observation, DMs sample high variance Gaussian noise $\mathbf{x}_M \sim \mathcal{N}(\mathbf{0}, \sigma^2_{\max})$ and sequentially denoise $\mathbf{x}_M$ into $\mathbf{x}_i \sim p(\mathbf{x}_i;\sigma_i)$, $i \in \{0,\dots,M\}$, with $\sigma_i < \sigma_{i+1}$ and $\sigma_M = \sigma_{\max}$. Assuming the DM is accurate, if $\sigma_0 = 0$ then the resulting $\mathbf{x}_0$ is distributed according to the data.

Sampling. In practice, this iterative denoising process explained above can be implemented through the numerical simulation of the *Probability Flow* ordinary differential

equation (ODE) [263]

$$d\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma(t))\, dt, \tag{5.2}$$

where $\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma)$ is the *score function* [123]. The schedule $\sigma(t)\colon [0, 1] \to \mathbb{R}_+$ is user-specified and $\dot{\sigma}(t)$ denotes the time derivative of $\sigma(t)$. Alternatively, we may also numerically simulate a stochastic differential equation (SDE) [141, 263]:

$$d\mathbf{x} = \underbrace{-\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma(t))\, dt}_{\text{Probability Flow ODE; see Equation (5.2)}} \underbrace{-\beta(t)\sigma^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma(t))\, dt + \sqrt{2\beta(t)}\sigma(t)\, d\omega_t}_{\text{Langevin diffusion component}}, \tag{5.3}$$

where $d\omega_t$ is the standard Wiener process. In principle, simulating either the Probability Flow ODE or the SDE above results in samples from the same distribution.

Training. DM training reduces to learning a model $\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}; \sigma)$ for the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma)$. The model can, for example, be parameterized as $\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma) \approx s_{\boldsymbol{\theta}}(\mathbf{x}; \sigma) = (D_{\boldsymbol{\theta}}(\mathbf{x}; \sigma) - \mathbf{x})/\sigma^2$ [141], where $D_{\boldsymbol{\theta}}$ is a learnable *denoiser* that, given a noisy data point $\mathbf{x}_0 + \mathbf{n}$, $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$, $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_d)$, and conditioned on the noise level σ, tries to predict the clean $\mathbf{x}_0$. The denoiser $D_{\boldsymbol{\theta}}$ (or equivalently the score model) can be trained via *denoising score matching* (DSM)

$$\mathbb{E}_{\substack{(\mathbf{x}_0, \mathbf{c}) \sim p_{\text{data}}(\mathbf{x}_0, \mathbf{c}), \\ (\sigma, \mathbf{n}) \sim p(\sigma, \mathbf{n})}} \left[\lambda_\sigma \| D_{\boldsymbol{\theta}}(\mathbf{x}_0 + \mathbf{n}; \sigma, \mathbf{c}) - \mathbf{x}_0 \|_2^2 \right], \tag{5.4}$$

where $p(\sigma, \mathbf{n}) = p(\sigma)\mathcal{N}(\mathbf{n}; \mathbf{0}, \sigma^2)$, $p(\sigma)$ is a distribution over noise levels σ, $\lambda_\sigma\colon \mathbb{R}_+ \to \mathbb{R}_+$ is a weighting function, and $\mathbf{c}$ is a conditioning signal, e.g., a class label or a text prompt. For unconditional modeling, $\mathbf{c}$ may simply be ignored.

Classifier-free guidance. Classifier-free guidance [108] is a technique to guide the iterative sampling process of a DM towards a particular conditioning signal $\mathbf{c}$ by mixing the predictions of a conditional and an unconditional model

$$D^w(\mathbf{x}; \sigma, \mathbf{c}) = (1 + w)D(\mathbf{x}; \sigma, \mathbf{c}) - wD(\mathbf{x}; \sigma), \tag{5.5}$$

where $w \geq 0$ is the *guidance strength*. In practice, the unconditional model can be trained jointly alongside the conditional model in a single network by randomly replacing the conditional signal $\mathbf{c}$ with a (learnable) null embedding in Equation (5.4), e.g., 10% of the time [108]. Classifier-free guidance is widely used to improve the sampling quality, at the cost of reduced diversity, of text-to-image DMs [211, 235].

5.3.3 Method

We propose to distill the knowledge of a large DM, with *frozen* parameters $\boldsymbol{\phi}$, into a small student DM, with parameters $\boldsymbol{\theta}$, via an (approximate) *score matching* (SM) loss

$$\mathbb{E}\left[\lambda_\sigma \| D_{\boldsymbol{\theta}}(\mathbf{x}_0 + \mathbf{n}; \sigma, \mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x}_0 + \mathbf{n}; \sigma, \mathbf{c}) \|_2^2\right], \tag{5.6}$$

where the expectation is over the same distributions as in Equation (5.4). The loss in Equation (5.6) is consistent: zero loss implies that the knowledge of the teacher has been perfectly distilled into the student model (assuming full support of $p_{\text{data}}(\mathbf{x}_0)$ and $p(\sigma)$). Furthermore, Equation (5.6) becomes standard score matching [123] as the teacher score model $s_{\boldsymbol{\phi}}(\mathbf{x}; \sigma) = (D_{\boldsymbol{\phi}}(\mathbf{x}; \sigma) - \mathbf{x})/\sigma^2$ approaches the true score function $\nabla_{\mathbf{x}} \log p(\mathbf{x}; \sigma)$.

Guided distillation. To distill the knowledge of a jointly trained (un)conditional DM for classifier-free guidance, we can randomly replace the conditioning signal $\mathbf{c}$ with the null embedding in Equation (5.6). Note, however, that during inference we still need to evaluate the student model twice to compute Equation (5.5). To accelerate inference even further, we may follow Meng et al. [196] and directly distill the guidance computation (for an interval $[w_{\min}, w_{\max}]$) jointly with the knowledge of the teacher model, i.e.,

$$\mathbb{E}_{\substack{(\mathbf{x}_0,\mathbf{c})\sim p_{\text{data}}(\mathbf{x}_0,\mathbf{c}),\\ (\sigma,\mathbf{n})\sim p(\sigma,\mathbf{n}),\\ w\sim\mathcal{U}[w_{\min},w_{\max}]}} \left[\lambda_\sigma \| D_{\boldsymbol{\theta}}(\mathbf{x}; \sigma, \mathbf{c}, w) - D^w_{\boldsymbol{\phi}}(\mathbf{x}; \sigma, \mathbf{c}) \|_2^2\right], \tag{5.7}$$

where $D^w_{\boldsymbol{\phi}}$ is computed via Equation (5.5) and $\mathbf{x} = \mathbf{x}_0 + \mathbf{n}$. Note that the student model is now additionally conditioned on the guidance strength w.

5.3.4 Experiments

We focus our efforts on a thorough evaluation on CIFAR-10 [160]. Student and teacher DMs are implemented using the DDPM++ architecture [263]. The teacher model uses 128 base channels while we use a variety of student models ranging from 32 to 128 base channels. We generate samples from our DMs using the deterministic Heun sampler proposed in Karras et al. [141] and we measure the sample quality via Fréchet Inception Distance (FID) [105] using 5k synthesized samples and all training samples. All experiment and training details can be found in Section 5.4.1.1

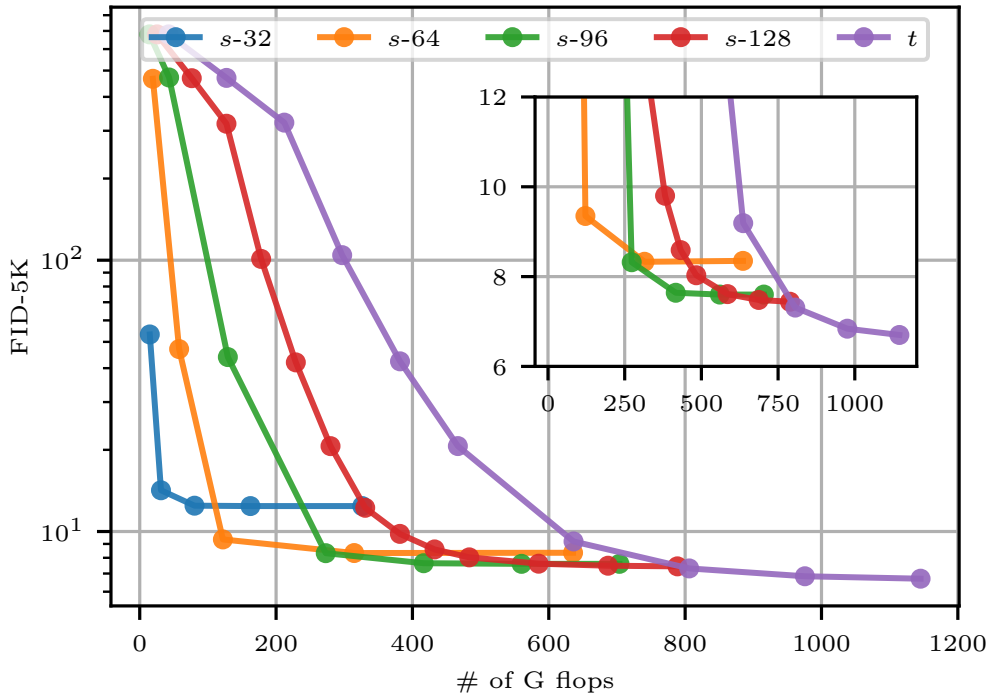


Figure 5.2: Unconditional image generation on CIFAR-10. Teacher model t and a variety of student models s with different number of base channels indicated in the legend. Linear y-axis as inset.

Table 5.1: Parameters and number of flops (per single forward pass) of the unconditional teacher model t and student models s.

Model	s-32	s-64	s-96	s-128	t
# of M parameters	2.2	8.8	19.8	35.1	55.7
# of G flops	1.64	6.42	14.36	25.44	42.42
# of residual blocks	2	2	2	2	4
# of base channels	32	64	96	128	128

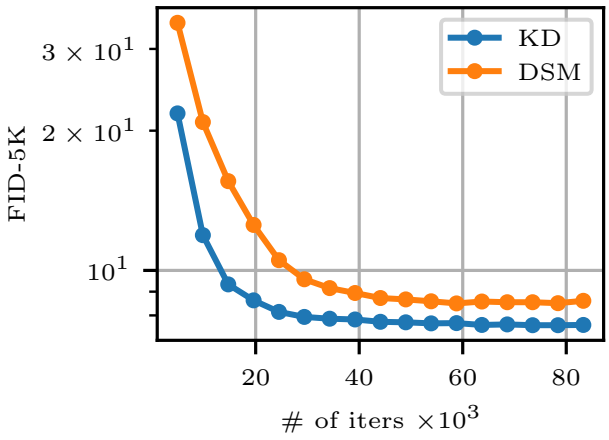


Figure 5.3: Distilling a large teacher DM into a smaller model (KD) leads to faster convergence and overall better performance than standard DSM training (for the smaller model).

5.3.4.1 Unconditional Distillation

We compare an unconditional teacher model to a variety of student models; see Table 5.1. We compute FID-5k for each model using a variety of solver steps; see Figure 5.2 for results. For fair comparison, the x-axis shows the accumulated number of G flops rather than the number of solver steps. We can see that there exist fixed budgets of G flops for which each of the four student model performs best, e.g., s-32 at 50 G Flops and s-96 at 400 G Flops, etc. Therefore, given GPU memory or inference time constraints, the size of the teacher model can be tuned to optimize performance. Overall, the gap between the larger student models (with 96 and 128 base channels) and the teacher model are reasonable, i.e., less than one FID-5K.

We also compare the training speed of our s-96 model to a standard DM (trained with Equation (5.4)) with the same neural network backbone; see Figure 5.3. The student model needs less iterations for convergence and overall converges to a better FID-5K value (7.63 vs 8.60). Note that during each iteration of KD we also need to do a forward pass through the larger teacher model; to reduce additional training time cost, the forward passes of teacher and student models may be parallelized.

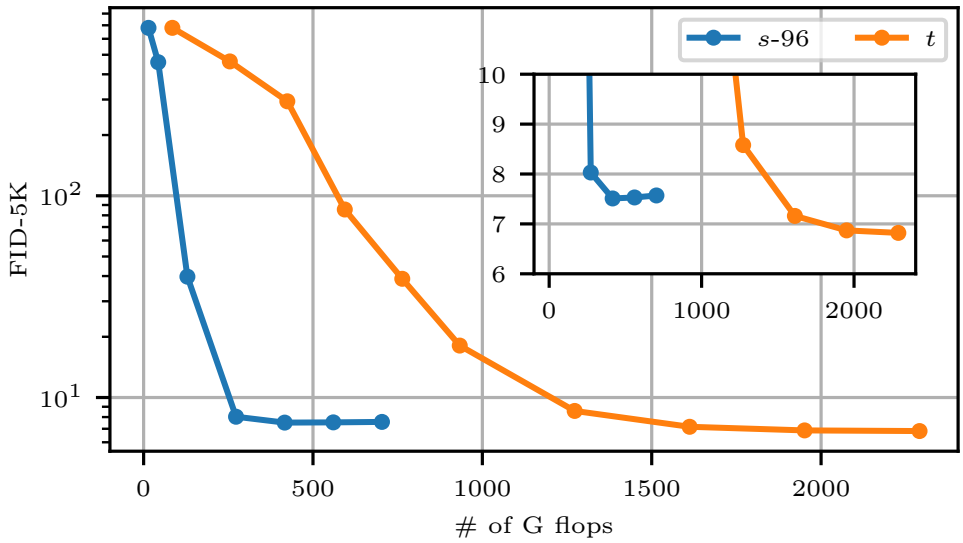


Figure 5.4: Guided image generation on CIFAR-10 (guidance strength w=0.25). Teacher model t and student model s. The student is conditioned on the guidance scale w whereas the teacher model needs to evaluate both a conditional and an unconditional DM per step. Linear y-axis as inset.

5.3.4.2 Guided Distillation

We additionally train a guided student model (96 base channels) with KD according to Equation (5.5) where we set w_{min}=0.0 and w_{max}=3.0. In Figure 5.4, we compare the student model to the teacher model for guidance strength w=0.25. The discrepancy of the performance at small number of G flops between the student and the teacher is even more striking than in the unconditional case, likely due to the simultaneous KD and guidance distillation. Overall, the gap between the student and the teacher model is reasonable, less than 0.75 FID-5K. Samples for both the student and teacher models can be found in Figure 5.1.

5.3.5 Future Directions

We have shown that the size of the neural network backbone in DMs can be drastically reduced with our KD approach resulting in faster inference, while keeping overall performance drops to a reasonable level. We envision our method as a promising tool for relevant and novel applications in generative modeling, e.g., text-to-image synthesis on edge devices.

(a) DSM: 7742G flops, 50 steps

(b) s: 7742G flops, 50 steps

(c) t: 8145G flops, 12 steps

(d) t: 33936G flops, 50 steps

Figure 5.5: Samples generated for the DSM baseline, the KD student model s and the original Stable Diffusion (teacher) model t. Prompt: "A beautiful castle, matte painting.".

In future work, we are planning to expand this work into the following directions:

Combining KD with sampling distillation. Distilling the sampling process of DMs [24, 196, 244, 265] allows for high-quality synthesis of large-scale models in several seconds. Sampling distillation is orthogonal to our KD approach, and combining these two ideas is a promising future research direction. An interesting question may be the order of distillation: Should we first distill the sampling process or the network? Or could we potentially do both distillations jointly?

Mixed training. In this work, we only considered pure distillation, however, it has been shown to be helpful to combine KD with standard training in discriminative models [106]. One approach for mixed training may be a linear combination of Equation (5.4) and Equation (5.6). As we show in Section 5.4.2, this mixed training approach is equivalent to performing distillation with an additional term

$$2\alpha\lambda_\sigma(D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma))^\top(D_{\boldsymbol{\phi}}(\mathbf{x};\sigma) - \mathbf{x}_0), \tag{5.8}$$

where $\alpha \in [0, 1]$, inside the expectation of Equation (5.6).

Better initialization. Fine-tuning large-scale DMs has been shown to be highly effective: for example, fine-tuning text-to-image DMs for, say, 100 to 1000 iterations on a small dataset of a few images results in highly editable personalized text-to-image models [90, 237]. Similarly, student models in sampling distillation are generally initialized from the teacher model, which allows for fine-tuning and results in faster convergence. In contrast, our student architectures are smaller, and therefore we cannot directly make use of the teacher for initialization. Future work could explore better initialization methods that may improve the training speed of our KD approach.

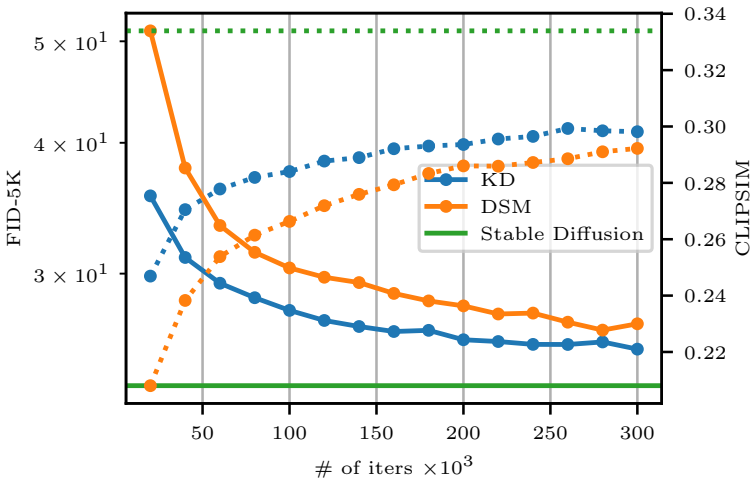


Figure 5.6: Initial experiments on Stable Diffusion show that distilling the network of large scale text-to-image DMs into smaller student models leads to better performance and faster convergence compared to standard DSM training. The gap of the student to the teacher is, however, still significant and needs to be addressed in future work. The solid and dotted lines represent *zero-shot* FID-5K and CLIPSIM on COCO [175], respectively.

Applying KD to larger models. In this work, we thoroughly study KD of DMs for CIFAR-10. An obvious future direction is to scale our approach to larger models: To this end, we perform a preliminary study on Stable Diffusion [235], distilling its latent DM into a network of less than a quarter of the original size (from 866M to 200M parameters). Compared to training a standard DM of the same size with DSM (Equation (5.4)), we find that the KD student converges faster and to a better value; see Figure 5.6. This is a promising result which may indicate that our results on CIFAR-10 transfer to large-scale DMs. Compared to our CIFAR-10 results, however, we found that there is still a substantial gap compared to the teacher model; see also samples in Figure 5.5. Experiment details can be found in Section 5.4.1.2. In future work, we are planning to thoroughly evaluate our KD approach to large-scale (latent) DMs.

5.4 Appendix

5.4.1 Experiment Details

5.4.1.1 CIFAR-10

Our teacher models for CIFAR-10 are taken from Karras et al. [141]; one conditional[1] and one unconditional model[2]. The networks are based on the DDPM++ architecture [263]. The teacher and student models have four and two residual blocks, respectively. The teacher model has 128 base channels while we train multiple student models ranging from 32 to 128 base channels. All models are trained for 100k iterations, using a batch size of 512, to ensure convergence. We use Adam with learning rate 1×10^{-3} and an exponential moving average half-life of 50M images, following Karras et al. [141]. For KD, we do not use any dropout while the DSM baseline in Figure 5.3 uses a dropout probability of 10% to prevent over-fitting. All student models (and the DSM baseline) use the same network preconditioning, noise distribution $p(\sigma)$ and loss weighting λ_σ as the teacher model; see the last column of Table 1 in Karras et al. [141].

5.4.1.2 Stable Diffusion

Our teacher model is Stable Diffusion [235] fine-tuned to **v**-parameterization [244]. The student model (and the DSM baseline) uses the same architecture as Stable Diffusion, however, the number of base channels is reduced from 360 to 192 and the transformer block at the highest resolution is removed. We use AdamW with learning rate 3×10^{-4} and batch size 512. The exponential moving average half life, the noise distribution $p(\sigma)$, and the loss weighting λ_σ for both the student model and the DSM baseline are the same as used in the original Stable Diffusion model.

5.4.2 Mixed Training Derivation

In Section 5.3.5, we propose the following mixed training objective

$$\mathbb{E}[\lambda_\sigma \left((1-\alpha)\|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c})\|_2^2 + \alpha\|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - \mathbf{x}_0\|_2^2\right)], \tag{5.9}$$

[1] https://nvlabs-fi-cdn.nvidia.com/edm/pretrained/edm-cifar10-32x32-uncond-vp.pkl
[2] https://nvlabs-fi-cdn.nvidia.com/edm/pretrained/edm-cifar10-32x32-uncond-ve.pkl

where $\mathbf{x} = \mathbf{x}_0 + \mathbf{n}$. Let us add and subtract the teacher model $D_{\boldsymbol{\phi}}$ to the second norm

$$\|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - \mathbf{x}_0\|_2^2 \tag{5.10}$$
$$= \|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c}) + D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c}) - \mathbf{x}_0\|_2^2 \tag{5.11}$$
$$= \|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c})\|_2^2 + 2(D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma))^\top (D_{\boldsymbol{\phi}}(\mathbf{x};\sigma) - \mathbf{x}_0) + \|D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c}) - \mathbf{x}_0\|_2^2 \tag{5.12}$$
$$= \|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c})\|_2^2 + 2(D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma))^\top (D_{\boldsymbol{\phi}}(\mathbf{x};\sigma) - \mathbf{x}_0) + \text{const.} \tag{5.13}$$

Note that the last term in the above equation is a constant with respect to the learnable parameters $\boldsymbol{\theta}$. Plugging the above into Equation (5.9), we have

$$\mathbb{E}[\lambda_\sigma \left(\|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma,\mathbf{c})\|_2^2 + 2\alpha(D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma))^\top (D_{\boldsymbol{\phi}}(\mathbf{x};\sigma) - \mathbf{x}_0)\right)] + \text{const.} \tag{5.14}$$

This shows that mixed training is equivalent to plain distillation with a regularization term which uses the clean data $\mathbf{x}_0$.

Alternatively, we may similarly add and subtract the clean data $\mathbf{x}_0$ to the first norm in Equation (5.9) which results in

$$\mathbb{E}[\lambda_\sigma \left(\|D_{\boldsymbol{\theta}}(\mathbf{x};\sigma,\mathbf{c}) - \mathbf{x}_0\|_2^2 + 2(1-\alpha)(D_{\boldsymbol{\theta}}(\mathbf{x};\sigma) - \mathbf{x}_0)^\top (\mathbf{x}_0 - D_{\boldsymbol{\phi}}(\mathbf{x};\sigma))\right)] + \text{const}, \tag{5.15}$$

which shows that mixed training is also equivalent to standard DM training (DSM in Equation (5.4)) with an additional regularization term involving the teacher network $D_{\boldsymbol{\phi}}$.

Chapter 6

Conclusion

In this thesis, we use DMs to model the data generating processes commonly encountered in artificial intelligence, focusing on natural images. While the primary objective of generative models is to generate samples that closely resemble those generated by the data generating process, there may be (several) secondary objectives which are important to practitioners. For example, in this thesis, we introduce several methods that accelerate the sampling process of DMs. Furthermore, we introduce one method that privatizes DMs, that is, prevents revealing training data through (samples from) the trained network.

In Chapter 2 we investigate the underlying diffusion process of DMs and introduce a new DM based on critically-damped Langevin dynamics. In Chapter 3 we show how reparameterizing the differential equations arising in DMs may naturally lead to faster numerical schemes. We also discuss several numerical solvers and the pitfall of those based on finite differences in the large step size limit. To circumvent this issue, we then introduce an efficient neural higher-order solver that learns higher-order derivatives. We also discuss recent methods that distill the sampling process of DMs altogether. In Chapter 5 we propose to distill the knowledge of large pre-trained DMs into smaller student models using an approximate score matching objective. In particular, the small student model is regressed towards predictions of the teacher DM rather than the clean data as is done in standard DM training. The student models are fast at inference time and may also be deployed on GPUs with significantly less memory than was required for the original teacher model.

In the future, our work on accelerated sampling could be extended in several directions. All three methods introduced in this thesis are orthogonal, and may therefore be mixed and matched. For example, we may distill the knowledge of DMs that utilize critically-damped

Langevin dynamics as their diffusion process, and then use the neural higher-order solver introduced in Chapter 3 to generate samples. Furthermore, combining the knowledge distillation technique from Chapter 5 with methods that distill the entire sampling process of DMs (see Section 3.5.1) may result in very fast and efficient samplers. Not restricted to the methods introduced in this thesis, we believe that the full potential of accelerated sampling can be unlocked by customizing samplers to their respective domains. For example, solvers for DMs modeling video data may not only exploit the spatial redundancy of individual frames but also the temporal redundancy. We believe that the ultimate goal should be instant generation across all domains, potentially on resource-limited devices.

To privatize DMs, we build on well-established techniques from the differential privacy literature. In particular, we enforce privacy in DMs by training the neural network backbone with differentially private stochastic gradient descent. Overfitting and privacy-leakage of generative models are more relevant than ever, considering recent works that train powerful photo-realistic image generators on large-scale web-scraped data and the recent adaption of DMs in domains with naturally more sensitive data, such as medical imaging.

Differentially private diffusion models with moderate privacy guarantees are able to generate high quality small-resolution images. However, non-private DMs are currently still significantly better at generating high-resolution images. To attempt closing this gap, future research may carefully pre-train DMs on public data and only fine-tune DMs on sensitive data for a small number of iterations as has been proposed in some recent work. Additionally, we may consider more parameter-efficient neural network backbones which are naturally more private given a fixed number of training iterations. More efficient methods for enforcing differential privacy guarantess in DMs and other generative models beyond differentially private stochastic gradient descent may also be developed.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016. Pages: 152, 153, 158, 164, and 165

[2] Gergely Acs, Luca Melis, Claude Castelluccia, and Emiliano De Cristofaro. Differentially Private Mixture of Generative Neural Networks. *IEEE Transactions on Knowledge and Data Engineering*, 31(6):1109–1121, 2018. Page: 164

[3] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. Compressing GANs using Knowledge Distillation. *arXiv:1902.00159*, 2019. Pages: 210 and 213

[4] Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Éric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. GSNs: generative stochastic networks. *Information and Inference: A Journal of the IMA*, 5(2):210–249, 03 2016. ISSN 2049-8764. Page: 29

[5] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models. *arXiv:2208.09399*, 2022. Page: 15

[6] Hans C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of Chemical Physics*, 72(4):2384–2393, 1980. Page: 29

[7] Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982. Pages: 6, 22, and 97

[8] Jyoti Aneja, Alexander Schwing, Jan Kautz, and Arash Vahdat. A Contrastive Learning Approach for Training Variational Autoencoder Priors. In *Neural Information Processing Systems*, 2021. Page: 32

[9] Rohan Anil, Badih Ghazi, Vineet Gupta, Ravi Kumar, and Pasin Manurangsi. Large-Scale Differentially Private BERT. *arXiv:2108.01624*, 2021. Pages: 162 and 165

[10] Abdul Fatir Ansari, Ming Liang Ang, and Harold Soh. Refining Deep Generative Models via Discriminator Gradient Flow. In *International Conference on Learning Representations*, 2021. Page: 32

[11] Martin Arjovsky and Leon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. In *International Conference on Learning Representations*, 2017. Pages: 153 and 159

[12] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. Pages: 15, 31, and 32

[13] Omri Avrahami, Dani Lischinski, and Ohad Fried. Blended Diffusion for Text-driven Editing of Natural Images. *arXiv:2111.14818*, 2021. Page: 95

[14] Yasaman Bahri, Jonathan Kadmon, Jeffrey Pennington, Sam S. Schoenholz, Jascha Sohl-Dickstein, and Surya Ganguli. Statistical Mechanics of Deep Learning. *Annual Review of Condensed Matter Physics*, 11:501–528, 2020. Page: 29

[15] J. Bailey. The tools of generative art, from flash to neural networks. *Art in America*, 2020. Pages: 35 and 207

[16] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers. *arXiv:2211.01324*, 2022. Pages: 12, 153, 165, and 172

[17] Irene Balelli, Santiago Silva, and Marco Lorenzi. A Differentially Private Probabilistic Framework for Modeling the Variability Across Federated Datasets of Heterogeneous Multi-View Observations. *Journal of Machine Learning for Biomedical Imaging*, 2022. Page: 165

[18] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise. *arXiv:2208.09392*, 2022. Page: 90

[19] Fan Bao, Chongxuan Li, Jiacheng Sun, Jun Zhu, and Bo Zhang. Estimating the Optimal Covariance with Imperfect Mean in Diffusion Probabilistic Models. In *International Conference on Machine Learning*, pages 1555–1584, 2022. Pages: xiii, 104, 149, and 150

[20] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-DPM: an Analytic Estimate of the Optimal Reverse Variance in Diffusion Probabilistic Models. In *International Conference on Learning Representations*, 2022. Pages: 95, 103, 106, 107, 108, and 139

[21] Dmitry Baranchuk, Andrey Voynov, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Label-Efficient Semantic Segmentation with Diffusion Models. In *International Conference on Learning Representations*, 2022. Pages: 15 and 101

[22] Georgios Batzolis, Jan Stanczuk, Carola-Bibiane Schönlieb, and Christian Etmann. Conditional Image Generation with Score-Based Diffusion Models. *arXiv:2111.13606*, 2021. Pages: 95 and 212

[23] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep Generative Stochastic Networks Trainable by Backprop. In *International Conference on Machine Learning*. PMLR, 2014. Page: 29

[24] David Berthelot, Arnaud Autef, Jierui Lin, Dian Ang Yap, Shuangfei Zhai, Siyuan Hu, Daniel Zheng, Walter Talbot, and Eric Gu. TRACT: Denoising Diffusion Models with Transitive Closure Time-Distillation. *arXiv:2303.04248*, 2023. Pages: 150, 212, and 219

[25] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge Distillation: A Good Teacher is Patient and Consistent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10925–10934, 2022. Pages: 210 and 213

[26] Alex Bie, Gautam Kamath, and Guojun Zhang. Private GANs, Revisited. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*, 2022. Pages: xxi, 158, 166, and 167

[27] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models. In *Conference on Computer Vision and Pattern Recognition 2023*, 2023. Pages: 13, 16, and 165

[28] Florian Bordes, Sina Honari, and Pascal Vincent. Learning to Generate Samples from Noise through Infusion Training. In *5th International Conference on Learning Representations, ICLR*, 2017. Page: 29

[29] Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. Riemannian Score-Based Generative Modelling. *arXiv:2202.02763*, 2022. Page: 95

[30] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *International Conference on Learning Representations*, 2019. Pages: 105, 124, and 153

[31] Tim Brooks, Aleksander Holynski, and Alexei A Efros. InstructPix2Pix: Learning to Follow Image Editing Instructions. *arXiv:2211.09800*, 2022. Page: 12

[32] Zhiqi Bu, Jialin Mao, and Shiyun Xu. Scalable and Efficient Training of Large Convolutional Neural Networks with Differential Privacy. *Advances in Neural Information Processing Systems*, 35:38305–38318, 2022. Pages: 162 and 185

[33] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance Weighted Autoencoders. *arXiv:1509.00519*, 2015. Pages: 30 and 48

[34] Giovanni Bussi and Michele Parrinello. Accurate sampling using Langevin dynamics. *Phys. Rev. E*, 75:056707, 2007. Pages: 20, 29, and 56

[35] Giovanni Bussi, Davide Donadio, and Michele Parrinello. Canonical sampling through velocity rescaling. *The Journal of Chemical Physics*, 126(1):014101, 2007. Pages: 29 and 90

[36] John Charles Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2016. Pages: 100 and 103

[37] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning Gradient Fields for Shape Generation. In *Proceedings of the European Conference on Computer Vision*, 2020. Pages: 15 and 95

[38] Tianshi Cao, Alex Bie, Arash Vahdat, Sanja Fidler, and Karsten Kreis. Don't Generate Me: Training Differentially Private Generative Models with Sinkhorn Divergence. *Advances in Neural Information Processing Systems*, 34:12480–12492, 2021. Pages: xxi, xxvi, 165, 166, 167, 168, 170, and 191

[39] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting Training Data from Large Language Models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021. Pages: 151 and 153

[40] Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting Training Data from Diffusion Models. *arXiv:2301.13188*, 2023. Page: 151

[41] Michele Ceriotti, Giovanni Bussi, and Michele Parrinello. Langevin Equation with Colored Noise for Constant-Temperature Molecular Dynamics Simulations. *Physical Review Letters*, 102:020601, 2009. Page: 29

[42] Michele Ceriotti, Michele Parrinello, Thomas E. Markland, and David E. Manolopoulos. Efficient stochastic thermostatting of path integral molecular dynamics. *The Journal of Chemical Physics*, 133(12):124104, 2010. Pages: 29 and 90

[43] B Chandra and Rajesh Kumar Sharma. Adaptive Noise Schedule for Denoising Autoencoder. In *International Conference on Neural Information Processing*, pages 535–542. Springer, 2014. Page: 11

[44] Y. F. Chang and George Corliss. ATOMFT: Solving ODEs and DAEs using Taylor series. *Computers & Mathematics with Applications*, 28(10-12):209–233, 1994. Page: 104

[45] Dingfan Chen, Tribhuvanesh Orekondy, and Mario Fritz. GS-WGAN: A Gradient-Sanitized Approach for Learning Differentially Private Generators. *Advances in Neural Information Processing Systems*, 33:12673–12684, 2020. Pages: xxi, 164, 166, and 167

[46] Jia-Wei Chen, Chia-Mu Yu, Ching-Chia Kao, Tzai-Wei Pang, and Chun-Shien Lu. DPGEN: Differentially Private Generative Energy-Guided Network for Natural Image Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8387–8396, June 2022. Pages: 165, 174, 176, and 178

[47] Jianfei Chen, Cheng Lu, Biqi Chenli, Jun Zhu, and Tian Tian. VFlow: More Expressive Generative Flows with Variational Data Augmentation. In *International Conference on Machine Learning*, pages 1660–1669. PMLR, 2020. Pages: 30 and 48

[48] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. WaveGrad: Estimating Gradients for Waveform Generation. In *International Conference on Learning Representations*, 2021. Pages: 15, 20, 95, 103, and 165

[49] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, Najim Dehak, and William Chan. WaveGrad 2: Iterative Refinement for Text-to-Speech Synthesis. *arXiv:2106.09660*, 2021. Pages: 15 and 95

[50] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *Advances in Neural Information Processing Systems*, 2018. Pages: 7, 8, 11, 12, 48, 97, 99, and 104

[51] Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual Flows for Invertible Generative Modeling. In *Advances in Neural Information Processing Systems*, 2019. Page: 32

[52] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*. PMLR, 2014. Page: 30

[53] Ting Chen. On the Importance of Noise Scheduling for Diffusion Models. *arXiv:2301.10972*, 2023. Page: 8

[54] Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog Bits: Generating Discrete Data using Diffusion Models with Self-Conditioning. *arXiv:2208.04202*, 2022. Page: 15

[55] Xiang Cheng, Niladri S Chatterji, Peter L Bartlett, and Michael I Jordan. Underdamped Langevin MCMC: A non-asymptotic analysis. In *Conference on Learning Theory*, pages 300–323. PMLR, 2018. Page: 18

[56] Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon, and Sungroh Yoon. ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14347–14356, 2021. Pages: 95 and 212

[57] Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. Perception Prioritized Training of Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11472–11481, 2022. Page: 101

[58] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. StarGAN v2: Diverse Image Synthesis for Multiple Domains. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8188–8197, 2020. Page: 105

[59] Hyungjin Chung and Jong Chul Ye. Score-based diffusion models for accelerated MRI. *arXiv:2110.05243*, 2021. Pages: 12, 95, and 212

[60] Hyungjin Chung, Byeongsu Sim, and Jong Chul Ye. Come-Closer-Diffuse-Faster: Accelerating Conditional Diffusion Models for Inverse Problems through Stochastic Contraction. *arXiv:2112.05146*, 2021. Pages: 95 and 212

[61] Hyungjin Chung, Suhyeon Lee, and Jong Chul Ye. Fast Diffusion Sampler for Inverse Problems by Geometric Decomposition. *arXiv:2303.05754*, 2023. Page: 150

[62] George Corliss and YF Chang. Solving ordinary differential equations using Taylor series. *ACM Transactions on Mathematical Software (TOMS)*, 8(2):114–144, 1982. Page: 104

[63] Giannis Daras, Mauricio Delbracio, Hossein Talebi, Alexandros G Dimakis, and Peyman Milanfar. Soft Diffusion: Score Matching for General Corruptions. *arXiv:2209.05442*, 2022. Page: 90

[64] Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking High-Accuracy Differentially Private Image Classification through Scale. *arXiv:2204.13650*, 2022. Pages: 161, 162, 163, 165, 184, 187, and 193

[65] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. Pages: 105, 192, and 205

[66] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. In *Neural Information Processing Systems*, 2021. Pages: 13, 20, 95, 96, 104, 105, 108, 122, 123, 131, 155, 159, 165, 180, 183, and 212

[67] Sander Dieleman. Diffusion models are autoencoders, 2022. Page: 11

[68] Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. Continuous diffusion for categorical data. *arXiv:2211.15089*, 2022. Page: 15

[69] Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D Skeel, and Hartmut Neven. Bayesian Sampling Using Stochastic Gradient Thermostats. In *Advances in Neural Information Processing Systems*, 2014. Page: 30

[70] Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Taylor-Lagrange Neural Ordinary Differential Equations: Toward Fast Training and Evaluation of Neural ODEs. *arXiv:2201.05715*, 2022. Page: 104

[71] Tim Dockhorn, James A Ritchie, Yaoliang Yu, and Iain Murray. Density Deconvolution with Normalizing Flows. *arXiv:2006.09396*, 2020. Page: 16

[72] Tim Dockhorn, Yaoliang Yu, Eyyüb Sari, Mahdi Zolnouri, and Vahid Partovi Nia. Demystifying and Generalizing BinaryConnect. *Advances in Neural Information Processing Systems*, 34:13202–13216, 2021. Page: 16

[73] Tim Dockhorn, Tianshi Cao, Arash Vahdat, and Karsten Kreis. Differentially Private Diffusion Models. *arXiv:2210.09929*, 2022. Pages: 15 and 212

[74] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. GENIE: Higher-Order Denoising Diffusion Solvers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. Pages: 15, 91, 150, 156, 165, and 212

[75] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-Based Generative Modeling with Critically-Damped Langevin Diffusion. In *International Conference on Learning Representations*, 2022. Pages: 8, 15, 17, 95, 102, 104, 108, 126, 139, 165, 189, and 212

[76] Tim Dockhorn, Robin Rombach, Andreas Blattmann, and Yaoliang Yu. Distilling the Knowledge in Diffusion Models. In *CVPR Workshop on Generative Models for Computer Vision 2023*, 2023. Page: 16

[77] J. R. Dormand and P. J. Prince. A family of embedded Runge–Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. Pages: 48, 68, 103, and 106

[78] Friedrich Dörmann, Osvald Frisk, Lars Nørvang Andersen, and Christian Fischer Pedersen. Not All Noise is Accounted Equally: How Differentially Private Learning Benefits from Large Sampling Rates. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2021. Page: 165

[79] Arnaud Doucet, Will Grathwohl, Alexander G Matthews, and Heiko Strathmann. Score-Based Diffusion meets Annealed Importance Sampling. *Advances in Neural Information Processing Systems*, 35:21482–21494, 2022. Page: 90

[80] Yilun Du and Igor Mordatch. Implicit Generation and Modeling with Energy Based Models. In *Advances in Neural Information Processing Systems*, pages 3608–3618, 2019. Pages: 32 and 39

[81] Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987. Page: 20

[82] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you should treat it like one. *arXiv:2201.12204*, 2022. Pages: 15 and 95

[83] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006. Pages: 151, 153, and 157

[84] Cynthia Dwork, Aaron Roth, et al. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. Pages: 151, 153, 158, 176, and 192

[85] Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. Structure and Content-Guided Video Synthesis with Diffusion Models. *arXiv:2302.03011*, 2023. Page: 12

[86] W Feller. On the Theory of Stochastic Processes, with Particular Reference to Applications. In *First Berkeley Symposium on Mathematical Statistics and Probability*, pages 403–432, 1949. Page: 9

[87] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization. In *International Conference on Machine Learning*, pages 3154–3164. PMLR, 2020. Page: 104

[88] Lorenzo Frigerio, Anderson Santana de Oliveira, Laurent Gomez, and Patrick Duverger. Differentially Private Generative Adversarial Networks for Time Series, Continuous, and Discrete Open Data. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 151–164. Springer, 2019. Pages: 152, 153, 158, and 164

[89] Chie Furusawa, Shinya Kitaoka, Michael Li, and Yuri Odagiri. Generative Probabilistic Image Colorization. *arXiv:2109.14518*, 2021. Page: 20

[90] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion. *arXiv:2208.01618*, 2022. Pages: 14 and 219

[91] Rinon Gal, Moab Arar, Yuval Atzmon, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. Encoder-based Domain Tuning for Fast Personalization of Text-to-Image Models . *arXiv:2302.12228*, 2023. Pages: 12 and 14

[92] Ruiqi Gao, Yang Song, Ben Poole, Ying Nian Wu, and Diederik P Kingma. Learning Energy-Based Models by Diffusion Recovery Likelihood. In *International Conference on Learning Representations*, 2021. Pages: 12 and 32

[93] Krzysztof Geras and Charles Sutton. Scheduled denoising autoencoders. In *International Conference on Learning Representations*, 2015. Page: 11

[94] Krzysztof J Geras and Charles Sutton. Composite Denoising Autoencoders. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 681–696. Springer, 2016. Page: 11

[95] Sahra Ghalebikesabi, Leonard Berrada, Sven Gowal, Ira Ktena, Robert Stanforth, Jamie Hayes, Soham De, Samuel L Smith, Olivia Wiles, and Borja Balle. Differentially Private Diffusion Models Generate Useful Synthetic Images. *arXiv:2302.13861*, 2023. Pages: 208, 209, and 212

[96] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. AutoGAN: Neural Architecture Search for Generative Adversarial Networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019. Page: 32

[97] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Advances in neural information processing systems*, 27, 2014. Page: 35

[98] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical Composition of Differential Privacy. *Advances in Neural Information Processing Systems*, 34:11631–11642, 2021. Page: 164

[99] Anirudh Goyal, Nan Rosemary Ke, Surya Ganguli, and Yoshua Bengio. Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017. Page: 29

[100] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *International Conference on Learning Representations*, 2019. Pages: 7, 8, 11, 12, 30, 48, 97, 99, and 104

[101] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020. Pages: 210 and 213

[102] Frederik Harder, Kamil Adamczewski, and Mijung Park. DP-MERF: Differentially Private Mean Embeddings with RandomFeatures for Practical Privacy-preserving Data Generation. In *International Conference on Artificial Intelligence and Statistics*, pages 1819–1827. PMLR, 2021. Pages: xxii, 165, 167, 170, 205, and 206

[103] Fredrik Harder, Milad Jalali Asadabadi, Danica J Sutherland, and Mijung Park. Differentially Private Data Generation Needs Better Features. *arXiv:2205.12900*, 2022. Pages: xxi, 165, 166, 167, 169, 172, and 208

[104] Ulrich G Haussmann and Etienne Pardoux. Time Reversal of Diffusions. *The Annals of Probability*, pages 1188–1205, 1986. Pages: 6, 22, and 97

[105] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. Pages: 30, 105, 131, 166, 191, and 215

[106] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531*, 2(7), 2015. Pages: 210, 213, and 219

[107] Geoffrey E Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural computation*, 14(8):1771–1800, 2002. Page: 2

[108] Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. Pages: 13, 96, 104, 107, 108, 132, 157, 183, and 214

[109] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, 2020. Pages: 9, 10, 20, 26, 27, 29, 32, 39, 45, 75, 91, 95, 96, 98, 102, 103, 105, 122, 123, 155, 165, and 212

[110] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded Diffusion Models for High Fidelity Image Generation. *arXiv:2106.15282*, 2021. Pages: 12, 14, 20, 95, 110, 155, 165, and 212

[111] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen Video: High Definition Video Generation with Diffusion Models. *arXiv:2210.02303*, 2022. Pages: 13 and 165

[112] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video Diffusion Models. *arXiv:2204.03458*, 2022. Pages: 12, 13, 95, 165, and 212

[113] Emiel Hoogeboom and Tim Salimans. Blurring Diffusion Models. *arXiv:2209.05557*, 2022. Page: 90

[114] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant Diffusion for Molecule Generation in 3D. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022. Page: 15

[115] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. *arXiv:2301.11093*, 2023. Page: 8

[116] William G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Physical Review A*, 31:1695–1697, 1985. Page: 29

[117] Dewei Hu, Yuankai K. Tao, and Ipek Oguz. Unsupervised Denoising of Retinal OCT with Diffusion Probabilistic Model. *arXiv:2201.11760*, 2022. Pages: 12, 95, and 212

[118] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685*, 2021. Page: 14

[119] Chin-Wei Huang, Laurent Dinh, and Aaron Courville. Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models. *arXiv:2002.07101*, 2020. Page: 30

[120] Chin-Wei Huang, Jae Hyun Lim, and Aaron Courville. A Variational Perspective on Diffusion-Based Generative Models and Score Matching. In *Neural Information Processing Systems*, 2021. Pages: 8, 30, 95, and 98

[121] Philippe H. Hünenberger. *Thermostat Algorithms for Molecular Dynamics Simulations*, volume 173 of *Advanced Computer Simulation. Advances in Polymer Science*. Springer, Berlin, Heidelberg, 2005. Page: 29

[122] Michael F Hutchinson. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989. Pages: 4 and 8

[123] Aapo Hyvärinen. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6:695–709, 2005. ISSN 1532-4435. Pages: 2, 4, 26, 156, 211, 214, and 215

[124] John Ingraham, Max Baranov, Zak Costello, Vincent Frappier, Ahmed Ismail, Shan Tie, Wujie Wang, Vincent Xue, Fritz Obermeyer, Andrew Beam, et al. Illuminating protein space with a programmable generative model. *bioRxiv*, pages 2022–12, 2022. Page: 15

[125] Allan Jabri, David Fleet, and Ting Chen. Scalable Adaptive Computation for Iterative Generation. *arXiv:2212.11972*, 2022. Page: 13

[126] Allan Jabri, David Fleet, and Ting Chen. Scalable Adaptive Computation for Iterative Generation. *arXiv:2212.1197*, 2023. Page: 208

[127] Christopher Jarzynski. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E*, 56:5018–5035, 1997. Page: 29

[128] Christopher Jarzynski. Nonequilibrium Equality for Free Energy Differences. *Physical Review Letters*, 78:2690–2693, 1997.

[129] Christopher Jarzynski. Equalities and Inequalities: Irreversibility and the Second Law of Thermodynamics at the Nanoscale. *Annual Review of Condensed Matter Physics*, 2(1):329–351, 2011. Page: 29

[130] Myeonghun Jeong, Hyeongju Kim, Sung Jun Cheon, Byoung Jin Choi, and Nam Soo Kim. Diff-TTS: A Denoising Diffusion Model for Text-to-Speech. *arXiv:2104.01409*, 2021. Pages: 15, 20, 95, and 165

[131] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up. *arXiv:2102.07074*, 2021. Page: 32

[132] Bowen Jing, Gabriele Corso, Renato Berlinghieri, and Tommi Jaakkola. Subspace Diffusion Generative Models. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIII*, pages 274–289. Springer, 2022. Page: 90

[133] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations. *arXiv:2202.02514*, 2022. Pages: 15 and 95

[134] Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta Go Fast When Generating Data with Score-Based Models. *arXiv:2105.14080*, 2021. Pages: xxiv, 27, 29, 31, 32, 33, 47, 60, 65, 68, 95, 103, 106, and 165

[135] Alexia Jolicoeur-Martineau, Rémi Piché-Taillefer, Ioannis Mitliagkas, and Remi Tachet des Combes. Adversarial score matching and improved sampling for image generation. In *International Conference on Learning Representations*, 2021. Pages: 32, 67, 68, and 105

[136] Heewoo Jun, Rewon Child, Mark Chen, John Schulman, Aditya Ramesh, Alec Radford, and Ilya Sutskever. Distribution Augmentation for Generative Modeling. In *International Conference on Machine Learning*, pages 5006–5019. PMLR, 2020. Page: 32

[137] Heewoo Jun, Rewon Child, Mark Chen, John Schulman, Aditya Ramesh, Alec Radford, and Ilya Sutskever. Distribution Augmentation for Generative Modeling. In *International Conference on Machine Learning*, pages 5006–5019. PMLR, 2020. Page: 187

[138] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training Generative Adversarial Networks with Limited Data. In *Neural Information Processing Systems*, 2020. Pages: 32 and 153

[139] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Bision and Pattern Recognition*, pages 8110–8119, 2020.

[140] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-Free Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, 34:852–863, 2021. Pages: 153 and 191

[141] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. *arXiv:2206.00364*, 2022. Pages: xxi, xxvi, 6, 8, 9, 90, 92, 104, 117, 150, 155, 156, 157, 162, 163, 168, 169, 171, 179, 180, 182, 183, 184, 187, 194, 195, 213, 214, 215, and 221

[142] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising Diffusion Restoration Models. *arXiv:2201.11793*, 2022. Pages: 95, 165, and 212

[143] Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-Based Real Image Editing with Diffusion Models. *arXiv:2210.09276*, 2022. Page: 12

[144] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning Differential Equations that are Easy to Solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020. Page: 104

[145] Levon Khachatryan, Andranik Movsisyan, Vahram Tadevosyan, Roberto Henschel, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Text2Video-Zero: Text-to-Image Diffusion Models are Zero-Shot Video Generators. *arXiv:2303.13439*, 2023. Page: 12

[146] Dongjun Kim, Seungjae Shin, Kyungwoo Song, Wanmo Kang, and Il-Chul Moon. Score Matching Model for Unbounded Data Score. *arXiv:2106.05527*, 2021. Pages: 22, 25, 26, 32, and 34

[147] Seung Wook Kim, Bradley Brown, Kangxue Yin, Karsten Kreis, Katja Schwarz, Daiqing Li, Robin Rombach, Antonio Torralba, and Sanja Fidler. NeuralField-LDM: Scene Generation with Hierarchical Latent Diffusion Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. Page: 165

[148] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015. Pages: 30, 124, 164, and 181

[149] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114*, 2013. Pages: 2 and 10

[150] Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational Diffusion Models. *arXiv:2107.00630*, 2021. Pages: 20, 30, and 32

[151] Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational Diffusion Models. In *Advances in Neural Information Processing Systems*, 2021. Pages: 6, 95, 97, 98, and 165

[152] Durk P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018. Page: 32

[153] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by Simulated Annealing. *science*, 220(4598):671–680, 1983. Page: 5

[154] Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, Berlin, 1992. Pages: 3, 27, 29, 60, 95, 98, 100, 103, 104, 113, 114, and 119

[155] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. WILDS: A Benchmark of in-the-Wild Distribution Shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021. Page: 208

[156] Zhifeng Kong and Wei Ping. On Fast Sampling of Diffusion Probabilistic Models. *arXiv:2106.00132*, 2021. Pages: 29, 30, 32, 67, 95, 103, 106, 107, 108, and 139

[157] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. DiffWave: A Versatile Diffusion Model for Audio Synthesis. In *International Conference on Learning Representations*, 2021. Pages: 15, 20, 95, and 165

[158] Karsten Kreis, Kurt Kremer, Raffaello Potestio, and Mark E. Tuckerman. From classical to quantum and back: Hamiltonian adaptive resolution path integral, ring polymer, and centroid molecular dynamics. *The Journal of Chemical Physics*, 147 (24):244104, 2017. Pages: 29 and 90

[159] Karsten Kreis, Tim Dockhorn, Zihao Li, and Ellen Zhong. Latent Space Diffusion Models of Cryo-EM Structures. *arXiv:2211.14169*, 2022. Pages: 15 and 16

[160] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009. Pages: 105, 192, 205, 213, and 215

[161] Alexey Kurakin, Steve Chien, Shuang Song, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. Toward Training at ImageNet Scale with Differential Privacy. *arXiv:2201.12328*, 2022. Pages: 163 and 165

[162] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved Precision and Recall Metric for Assessing Generative Models. *Advances in Neural Information Processing Systems*, 32, 2019. Page: 135

[163] Max WY Lam, Jun Wang, Rongjie Huang, Dan Su, and Dong Yu. Bilateral Denoising Diffusion Models. *arXiv:2108.11514*, 2021. Page: 104

[164] Yann LeCun, Corinna Cortes, and Chris Burges. MNIST handwritten digit database, 2010. Pages: 166 and 192

[165] Benedict Leimkuhler and Charles Matthews. Rational Construction of Stochastic Numerical Methods for Molecular Sampling. *Applied Mathematics Research eXpress*, 2013(1):34–56, 2013. Pages: 20, 28, 29, 54, 56, and 90

[166] Benedict Leimkuhler and Charles Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics. Springer, 2015. Pages: 28, 29, 42, 54, 56, and 61

[167] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005. Pages: 27, 47, and 61

[168] Haoying Li, Yifan Yang, Meng Chang, Huajun Feng, Zhihai Xu, Qi Li, and Yueting Chen. SRDiff: Single Image Super-Resolution with Diffusion Probabilistic Models. *arXiv:2104.14951*, 2021. Pages: 20, 95, 165, and 212

[169] Shengmeng Li, Luping Liu, Zenghao Chai, Runnan Li, and Xu Tan. ERA-Solver: Error-Robust Adams Solver for Fast Sampling of Diffusion Probabilistic Models. *arXiv:2301.12935*, 2023. Page: 150

[170] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B Hashimoto. Diffusion-LM Improves Controllable Text Generation. *arXiv:2205.14217*, 2022. Page: 15

[171] Xuechen Li, Daogao Liu, Tatsunori B Hashimoto, Huseyin A Inan, Janardhan Kulkarni, Yin-Tat Lee, and Abhradeep Guha Thakurta. When Does Differentially Private Learning Not Suffer in High Dimensions? *Advances in Neural Information Processing Systems*, 35:28616–28630, 2022. Page: 162

[172] Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large Language Models Can Be Strong Differentially Private Learners. In *International Conference on Learning Representations*, 2022. Pages: 162, 163, and 165

[173] Seng Pei Liew, Tsubasa Takahashi, and Michihiko Ueno. PEARL: Data Synthesis via Private Embeddings and Adversarial Reconstruction Learning. In *International Conference on Learning Representations*, 2022. Pages: xxi, xxv, 165, 166, 167, and 192

[174] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3D: High-Resolution Text-to-3D Content Creation. *arXiv:2211.10440*, 2022. Page: 15

[175] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft Coco: Common Objects in Context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. Pages: xxiii and 220

[176] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo Numerical Methods for Diffusion Models on Manifolds. In *International Conference on Learning Representations*, 2022. Pages: 93, 95, 100, 103, 106, 107, 108, 110, 121, 133, 135, 136, 137, 139, 140, 141, 142, 149, 150, and 156

[177] Shaoteng Liu, Yuechen Zhang, Wenbo Li, Zhe Lin, and Jiaya Jia. Video-P2P: Video Editing with Cross-attention Control. *arXiv:2303.04761*, 2023. Page: 12

[178] Songxiang Liu, Dan Su, and Dong Yu. DiffGAN-TTS: High-Fidelity and Efficient Text-to-Speech with Denoising Diffusion GANs. *arXiv:2201.11972*, 2022. Pages: 15 and 95

[179] Xihui Liu, Dong Huk Park, Samaneh Azadi, Gong Zhang, Arman Chopikyan, Yuxiao Hu, Humphrey Shi, Anna Rohrbach, and Trevor Darrell. More Control for Free! Image Synthesis with Semantic Diffusion Guidance. *arXiv:2112.05744*, 2021. Pages: 95 and 212

[180] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. *arXiv:2209.03003*, 2022. Page: 150

[181] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738, 2015. Pages: 166 and 192

[182] Yunhui Long, Suxin Lin, Zhuolin Yang, Carl A Gunter, Han Liu, and Bo Li. Scalable differentially private data generation via private aggregation of teacher ensembles. 2019. Pages: xxi, 164, 166, 167, and 170

[183] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps. *arXiv:2206.00927*, 2022. Pages: 93, 104, 150, 156, and 212

[184] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models. *arXiv:2211.01095*, 2022. Pages: 93, 150, and 212

[185] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using Denoising Diffusion Probabilistic Models. *arXiv:2201.09865*, 2022. Pages: 12, 95, and 212

[186] Eric Luhman and Troy Luhman. Knowledge Distillation in Iterative Generative Models for Improved Sampling Speed. *arXiv:2101.02388*, 2021. Pages: 32, 96, 103, 106, 107, and 150

[187] Guanxiong Luo, Martin Heide, and Martin Uecker. MRI Reconstruction via Data Driven Markov Chain with Joint Uncertainty Estimation. *arXiv:2202.01479*, 2022. Pages: 12, 95, and 212

[188] Shitong Luo and Wei Hu. Diffusion Probabilistic Models for 3D Point Cloud Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2837–2845, 2021. Pages: 15, 20, 95, and 165

[189] Siwei Lyu. Interpretation and Generalization of Score Matching. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 359–366, Arlington, Virginia, USA, 2009. AUAI Press. Page: 29

[190] Zhaoyang Lyu, Zhifeng Kong, Xudong XU, Liang Pan, and Dahua Lin. A Conditional Point Diffusion-Refinement Paradigm for 3D Point Cloud Completion. In *International Conference on Learning Representations*, 2022. Pages: 15 and 95

[191] Yi-An Ma, Niladri Chatterji, Xiang Cheng, Nicolas Flammarion, Peter Bartlett, and Michael I Jordan. Is There an Analog of Nesterov Acceleration for MCMC? *arXiv:1902.00996*, 2019. Page: 30

[192] Glenn J. Martyna, Michael L. Klein, and Mark Tuckerman. Nosé–Hoover chains: The canonical ensemble via continuous dynamics. *The Journal of Chemical Physics*, 97(4):2635–2643, 1992. Page: 29

[193] Martin W McCall. *Classical Mechanics: From Newton to Einstein: A Modern Introduction, 2nd Edition*. Wiley, Hoboken, N.J., 2010. Pages: 24 and 36

[194] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. A General Approach to Adding Differential Privacy to Iterative Training Procedures. *arXiv:1812.06210*, 2018. Page: 158

[195] Chenlin Meng, Yang Song, Wenzhe Li, and Stefano Ermon. Estimating High Order Gradients of the Data Distribution by Denoising. *Advances in Neural Information Processing Systems*, 34, 2021. Pages: 93, 102, 103, 119, and 128

[196] Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On Distillation of Guided Diffusion Models. *arXiv:2210.03142*, 2022. Pages: 150, 212, 215, and 219

[197] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations. In *International Conference on Learning Representations*, 2022. Pages: 12, 13, 20, 95, 108, 165, and 212

[198] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which Training Methods for GANs do actually Converge? In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2018. Pages: 153 and 159

[199] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017. Pages: 163, 164, 172, and 174

[200] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi Differential Privacy of the Sampled Gaussian Mechanism. *arXiv:1908.10530*, 2019. Pages: 164 and 174

[201] Yisroel Mirsky and Wenke Lee. The Creation and Detection of Deepfakes: A Survey. *ACM Comput. Surv.*, 54(1), 2021. Pages: 35, 110, and 207

[202] Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021. URL `https://archives.ismir.net/ismir2021/paper/000058.pdf`. Page: 20

[203] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2018. Pages: 32 and 119

[204] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video Diffusion Models are General Video Editors. *arXiv:2302.01329*, 2023. Page: 12

[205] Wenlong Mou, Yi-An Ma, Martin J. Wainwright, Peter L. Bartlett, and Michael I. Jordan. High-Order Langevin Diffusion Yields an Accelerated MCMC Algorithm. *Journal of Machine Learning Research*, 22(42):1–41, 2021. Page: 30

[206] Javier R. Movellan. Contrastive Divergence in Gaussian Diffusions. *Neural Computation*, 20(9):2238–2252, 2008. Page: 29

[207] Eliya Nachmani, Robin San Roman, and Lior Wolf. Non Gaussian Denoising Diffusion Models. *arXiv:2106.07582*, 2021. Pages: 29 and 104

[208] Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 2001. Pages: 5, 29, and 90

[209] Radford M. Neal. MCMC Using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162, 2011. Pages: 18, 20, 23, 27, 30, 47, 50, 61, and 70

[210] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Cuong M. Nguyen, Dung Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep Learning for Deepfakes Creation and Detection: A Survey. *arXiv:1909.11573*, 2021. Pages: 35, 110, and 207

[211] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models. *arXiv:2112.10741*, 2021. Pages: 12, 13, 95, 108, 212, and 214

[212] Alexander Quinn Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. Pages: 20, 31, 32, 95, 103, 108, 155, and 165

[213] Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. Diffusion Models for Adversarial Purification. In *International Conference on Machine Learning*. PMLR, 2022. Page: 95

[214] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, 2020. Page: 95

[215] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of Chemical Physics*, 81(1):511–519, 1984. Page: 29

[216] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. *International Conference on Machine Learning*, 2016. Page: 32

[217] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013. Page: 185

[218] Kushagra Pandey, Avideep Mukherjee, Piyush Rai, and Abhishek Kumar. DiffuseVAE: Efficient, Controllable and High-Fidelity Generation from Low-Dimensional Latents. *arXiv:2201.00308*, 2022. Pages: 95 and 212

[219] Nicolas Papernot and Thomas Steinke. Hyperparameter Tuning with Renyi Differential Privacy. In *International Conference on Learning Representations*, 2022. Page: 184

[220] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable Private Learning with PATE. In *International Conference on Learning Representations*, 2018. Page: 164

[221] Gaurav Parmar, Dacheng Li, Kwonjoon Lee, and Zhuowen Tu. Dual Contradistinctive Generative Autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 823–832, 2021. Page: 32

[222] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 2019. Page: 168

[223] William Peebles and Saining Xie. Scalable Diffusion Models with Transformers. *arXiv:2212.09748*, 2022. Page: 13

[224] Cheng Peng, Pengfei Guo, S. Kevin Zhou, Vishal Patel, and Rama Chellappa. Towards performant and reliable undersampled MR reconstruction via diffusion model sampling. *arXiv:2203.04292*, 2022. Pages: 95 and 212

[225] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. Page: 30

[226] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D Diffusion. *arXiv:2209.14988*, 2022. Page: 15

[227] Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Sadekova, and Mikhail Kudinov. Grad-TTS: A Diffusion Probabilistic Model for Text-to-Speech. In *International Conference on Machine Learning*. PMLR, 2021. Pages: 15 and 95

[228] Konpat Preechakul, Nattanat Chatthee, Suttisak Wizadwongsa, and Supasorn Suwajanakorn. Diffusion Autoencoders: Toward a Meaningful and Decodable Representation. *arXiv:2111.15640*, 2021. Pages: 95 and 212

[229] Chenyang Qi, Xiaodong Cun, Yong Zhang, Chenyang Lei, Xintao Wang, Ying Shan, and Qifeng Chen. FateZero: Fusing Attentions for Zero-shot Text-based Video Editing. *arXiv:2303.09535*, 2023. Page: 12

[230] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv:2204.06125*, 2022. Pages: 12, 95, 104, 108, 153, 155, 165, 172, 210, and 212

[231] Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015. Page: 2

[232] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*, pages 1278–1286. PMLR, 2014. Pages: 2 and 10

[233] Severi Rissanen, Markus Heinonen, and Arno Solin. Generative Modelling With Inverse Heat Dissipation. *arXiv:2206.13397*, 2022. Page: 90

[234] A. J. Roberts. Modify the Improved Euler scheme to integrate stochastic differential equations. *arXiv:1210.0933*, 2012. Page: 29

[235] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. *arXiv:2112.10752*, 2021. Pages: 12, 13, 95, 153, 155, 165, 172, 210, 211, 212, 213, 214, 220, and 221

[236] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015. Pages: 13, 101, and 122

[237] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation. *arXiv:2208.12242*, 2022. Pages: 12, 14, and 219

[238] Hshmat Sahak, Daniel Watson, Chitwan Saharia, and David Fleet. Denoising Diffusion Probabilistic Models for Robust Image Super-Resolution in the Wild. *arXiv:2302.07864*, 2023. Page: 12

[239] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Palette: Image-to-Image Diffusion Models. *arXiv:2111.05826*, 2021. Pages: 12, 95, 165, and 212

[240] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image Super-Resolution via Iterative Refinement. *arXiv:2104.07636*, 2021. Pages: 12, 20, 95, 110, 165, and 212

[241] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv:2205.11487*, 2022. Pages: 153, 155, 165, and 172

[242] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic Text-to-Image Diffusion Models with

Deep Language Understanding. *arXiv:2205.11487*, 2022. Pages: 12, 110, 123, 210, and 212

[243] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing Generative Models via Precision and Recall. *Advances in Neural Information Processing Systems*, 31, 2018. Page: 135

[244] Tim Salimans and Jonathan Ho. Progressive Distillation for Fast Sampling of Diffusion Models. In *International Conference on Learning Representations*, 2022. Pages: xxi, 8, 91, 96, 97, 103, 106, 107, 112, 122, 124, 150, 157, 163, 165, 168, 169, 170, 179, 194, 212, 219, and 221

[245] Robin San-Roman, Eliya Nachmani, and Lior Wolf. Noise Estimation for Generative Diffusion Models. *arXiv:2104.02600*, 2021. Page: 29

[246] Pedro Sanchez and Sotirios A. Tsaftaris. Diffusion Causal Models for Counterfactual Estimation. In *First Conference on Causal Learning and Reasoning*, 2022. Page: 95

[247] Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*, volume 10. Cambridge University Press, 2019. Pages: 22, 23, 40, and 55

[248] Anand Sarwate. Retraction for Symmetric Matrix Perturbation for Differentially-Private Principal Component Analysis, 2017. Page: 165

[249] Hiroshi Sasaki, Chris G. Willcocks, and Toby P. Breckon. UNIT-DDPM: UNpaired Image Translation with Denoising Diffusion Probabilistic Models. *arXiv:2104.05358*, 2021. Pages: 12, 20, 95, 165, and 212

[250] Xiaocheng Shang, Zhanxing Zhu, Benedict Leimkuhler, and Amos J Storkey. Covariance-Controlled Adaptive Langevin Thermostat for Large-Scale Bayesian Sampling. In *Advances in Neural Information Processing Systems*, 2015. Page: 30

[251] Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning gradient fields for molecular conformation generation. In *International Conference on Machine Learning*. PMLR, 2021. Pages: 15 and 95

[252] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-A-Video: Text-to-Video Generation without Text-Video Data. *arXiv:2209.14792*, 2022. Pages: 12, 165, and 212

[253] Uriel Singer, Shelly Sheynin, Adam Polyak, Oron Ashual, Iurii Makarov, Filippos Kokkinos, Naman Goyal, Andrea Vedaldi, Devi Parikh, Justin Johnson, et al. Text-To-4D Dynamic Scene Generation. *arXiv:2301.11280*, 2023. Page: 15

[254] Raghav Singhal, Mark Goldstein, and Rajesh Ranganath. Where to Diffuse, How to Diffuse, and How to Get Back: Automated Learning for Multivariate Diffusions. *arXiv:2302.07261*, 2023. Page: 90

[255] Abhishek Sinha, Jiaming Song, Chenlin Meng, and Stefano Ermon. D2C: Diffusion-Denoising Models for Few-shot Conditional Generation. *arXiv:2106.06819*, 2021. Page: 20

[256] Jascha Sohl-Dickstein, Peter Battaglino, and Michael R. DeWeese. Minimum Probability Flow Learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011. Page: 29

[257] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *International Conference on Machine Learning*. PMLR, 2015. Pages: 9, 20, 29, 91, 96, 103, 155, and 165

[258] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*, 2021. Pages: xix, xxv, xxvi, 29, 30, 32, 65, 67, 75, 91, 95, 96, 97, 98, 103, 105, 106, 107, 108, 110, 112, 116, 135, 136, 137, 139, 140, 141, 142, 149, 156, 165, 171, 182, 184, 189, 195, and 212

[259] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, 2019. Pages: 5, 29, and 32

[260] Yang Song and Stefano Ermon. Improved Techniques for Training Score-Based Generative Models. *Advances in Neural Information Processing Systems*, 33:12438–12448, 2020. Page: 161

[261] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced Score Matching: A Scalable Approach to Density and Score Estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020. Pages: 4 and 8

[262] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum Likelihood Training of Score-Based Diffusion Models. In *Neural Information Processing Systems*, 2021. Pages: 8, 20, 22, 24, 27, 30, 32, 42, 45, 47, 48, 72, 95, 97, 98, 104, and 165

[263] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021. Pages: xxi, xxiv, xxv, 6, 7, 13, 17, 20, 22, 24, 26, 27, 29, 30, 31, 32, 33, 39, 47, 48, 60, 63, 65, 68, 69, 74, 75, 91, 95, 96, 97, 99, 102, 103, 104, 105, 107, 108, 112, 122, 123, 124, 139, 155, 156, 157, 163, 165, 166, 168, 170, 179, 180, 181, 183, 184, 189, 194, 213, 214, 215, and 221

[264] Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. Solving Inverse Problems in Medical Imaging with Score-Based Generative Models. In *International Conference on Learning Representations*, 2022. Pages: 12, 95, and 212

[265] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. *arXiv:2303.01469*, 2023. Pages: 150, 212, and 219

[266] Gilbert Strang. On the Construction and Comparison of Difference Schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968. Pages: 28 and 56

[267] Xuan Su, Jiaming Song, Chenlin Meng, and Stefano Ermon. Dual Diffusion Implicit Bridges for Image-to-Image Translation. *arXiv:2203.08382*, 2022. Pages: 12, 95, and 212

[268] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. Pages: xix, xx, 109, and 137

[269] Hideyuki Tachibana, Mocho Go, Muneyoshi Inahara, Yotaro Katayama, and Yotaro Watanabe. Itô-Taylor Sampling Scheme for Denoising Diffusion Probabilistic Models using Ideal Derivatives. *arXiv:2112.13339*, 2021. Pages: 95, 103, 114, and 115

[270] Shun Takagi, Tsubasa Takahashi, Yang Cao, and Masatoshi Yoshikawa. P3GM: Private High-Dimensional Data Release via Privacy Preserving Phased Generative Model. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 169–180. IEEE, 2021. Page: 165

[271] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in Neural Information Processing Systems*, 30, 2017. Pages: 210 and 213

[272] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation. In *Advances in Neural Information Processing Systems*, 2021. Pages: 15 and 95

[273] Reihaneh Torkzadehmahani, Peter Kairouz, and Benedict Paten. DP-CGAN: Differentially Private Synthetic Data and Label Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. Pages: xxi, 152, 153, 158, 164, 166, and 167

[274] Florian Tramer and Dan Boneh. Differentially Private Learning Needs Better Features (or Much More Data). In *International Conference on Learning Representations*, 2021. Pages: 165 and 169

[275] H. F. Trotter. On the Product of Semi-Groups of Operators. *Proceedings of the American Mathematical Society*, 10:545–551, 1959. Pages: 28 and 56

[276] M. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, 97(3):1990–2001, 1992. Pages: 29, 56, and 90

[277] Mark E. Tuckerman. *Statistical Mechanics: Theory and Molecular Simulation*. Oxford University Press, New York, 2010. Pages: 20, 23, 28, 29, 42, 47, 54, 56, and 61

[278] Cristian Vaccari and Andrew Chadwick. Deepfakes and Disinformation: Exploring the Impact of Synthetic Political Video on Deception, Uncertainty, and Trust in News. *Social Media + Society*, 6(1):2056305120903408, 2020. Pages: 35, 110, and 207

[279] Arash Vahdat and Jan Kautz. NVAE: A Deep Hierarchical Variational Autoencoder. In *Neural Information Processing Systems*, 2020. Pages: 11 and 32

[280] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based Generative Modeling in Latent Space. In *Neural Information Processing Systems*, 2021. Pages: 8, 20, 22, 26, 27, 29, 30, 31, 32, 34, 45, 48, 68, 69, 70, 95, 98, 102, 104, 126, and 165

[281] Margarita Vinaroz, Mohammad-Amin Charusaie, Frederik Harder, Kamil Adamczewski, and Mi Jung Park. Hermite Polynomial Features for Private Data Generation. In *International Conference on Machine Learning*, pages 22300–22324. PMLR, 2022. Page: 167

[282] Pascal Vincent. A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. Pages: 4, 22, 26, 50, 51, and 98

[283] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *International Conference on Machine learning*, pages 1096–1103. PMLR, 2008. Page: 11

[284] Boxin Wang, Fan Wu, Yunhui Long, Luka Rimanic, Ce Zhang, and Bo Li. DataLens: Scalable Privacy Preserving Training via Gradient Compression and Aggregation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2146–2168, 2021. Pages: xxi, xxii, 164, 166, 167, 170, and 207

[285] Daniel Watson, Jonathan Ho, Mohammad Norouzi, and William Chan. Learning to Efficiently Sample from Diffusion Probabilistic Models. *arXiv:2106.03802*, 2021. Pages: 29, 30, 67, and 103

[286] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning Fast Samplers for Diffusion Models by Differentiating Through Sample Quality. In *International Conference on Learning Representations*, 2022. Pages: 95, 103, 104, 106, 107, 108, 139, 141, 142, and 165

[287] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, pages 2022–12, 2022. Page: 15

[288] Ryan Webster, Julien Rabin, Loic Simon, and Frederic Jurie. This Person (Probably) Exists. Identity Membership Attacks Against GAN Generated Faces. *arXiv:2107.06018*, 2021. Pages: 152 and 153

[289] Li Wenliang, Danica J Sutherland, Heiko Strathmann, and Arthur Gretton. Learning deep kernels for exponential family densities. In *International Conference on Machine Learning*, pages 6737–6746. PMLR, 2019. Page: 5

[290] Jay Whang, Mauricio Delbracio, Hossein Talebi, Chitwan Saharia, Alexandros G. Dimakis, and Peyman Milanfar. Deblurring via Stochastic Refinement. *arXiv:2112.02475*, 2021. Pages: 95 and 212

[291] Suttisak Wizadwongsa and Supasorn Suwajanakorn. Accelerating Guided Diffusion Sampling with Splitting Numerical Methods. *arXiv:2301.11558*, 2023. Pages: 90 and 150

[292] Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Weixian Lei, Yuchao Gu, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation. *arXiv preprint arXiv:2212.11565*, 2022. Page: 12

[293] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747*, 2017. Pages: 166 and 192

[294] Zhisheng Xiao, Karsten Kreis, Jan Kautz, and Arash Vahdat. VAEBM: A Symbiosis between Variational Autoencoders and Energy-based Models. In *International Conference on Learning Representations*, 2021. Pages: 32 and 39

[295] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the Generative Learning Trilemma with Denoising Diffusion GANs. In *International Conference on Learning Representations*, 2022. Pages: 96, 103, 106, 107, 135, and 165

[296] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially Private Generative Adversarial Network. *arXiv:1802.06739*, 2018. Pages: 152, 153, 158, and 164

[297] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-Training With Noisy Student Improves ImageNet Classification. In *Proceedings of the IEEE/CVF Conference on Computer Bision and Pattern Recognition*, pages 10687–10698, 2020. Pages: 210 and 213

[298] Tian Xie, Xiang Fu, Octavian-Eugen Ganea, Regina Barzilay, and Tommi S. Jaakkola. Crystal Diffusion Variational Autoencoder for Periodic Material Generation. In *International Conference on Learning Representations*, 2022. Pages: 15 and 95

[299] Yutong Xie and Quanzheng Li. Measurement-conditioned Denoising Diffusion Probabilistic Model for Under-sampled Medical Image Reconstruction. *arXiv:2203.03623*, 2022. Pages: 12, 95, and 212

[300] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation. In *International Conference on Learning Representations*, 2022. Pages: 15 and 95

[301] Ruihan Yang, Prakhar Srivastava, and Stephan Mandt. Diffusion Probabilistic Modeling for Video Generation. *arXiv:2203.09481*, 2022. Pages: 95 and 212

[302] Yasin Yazıcı, Chuan-Sheng Foo, Stefan Winkler, Kim-Hui Yap, Georgios Piliouras, and Vijay Chandrasekhar. The Unusual Effectiveness of Averaging in GAN Training. In *International Conference on Learning Representations*, 2019. Pages: 190 and 191

[303] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See Through Gradients: Image Batch Recovery via GradInversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16337–16346, 2021. Pages: 151 and 153

[304] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees. In *International Conference on Learning Representations*, 2019. Page: 164

[305] Jongmin Yoon, Sung Ju Hwang, and Juho Lee. Adversarial Purification with Score-based Generative Models. In *International Conference on Machine Learning*. PMLR, 2021. Page: 95

[306] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-Friendly Differential Privacy Library in PyTorch. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021. Pages: 162, 164, 168, 173, 174, and 185

[307] Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, Sergey Yekhanin, and Huishuai Zhang. Differentially Private Fine-tuning of Language Models. In *International Conference on Learning Representations*, 2022. Pages: 165 and 208

[308] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. *arXiv:1506.03365*, 2015. Page: 105

[309] Sepanta Zeighami, Ritesh Ahuja, Gabriel Ghinita, and Cyrus Shahabi. A neural database for differentially private spatial range queries. *Proceedings of the VLDB Endowment*, 15(5):1066–1078, 2022. Page: 165

[310] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. LION: Latent Point Diffusion Models for 3D Shape Generation. In *Advances in Neural Information Processing Systems*, 2022. Page: 165

[311] Qianjun Zhang and Lei Zhang. Convolutional adaptive denoising autoencoders for hierarchical feature extraction. *Frontiers of Computer Science*, 12(6):1140–1148, 2018. Page: 11

[312] Qinsheng Zhang and Yongxin Chen. Fast Sampling of Diffusion Models with Exponential Integrator. *arXiv:2204.13902*, 2022. Pages: 91, 92, and 93

[313] Qinsheng Zhang and Yongxin Chen. Fast Sampling of Diffusion Models with Exponential Integrator. *arXiv:2204.13902*, 2022. Page: 156

[314] Qinsheng Zhang and Yongxin Chen. Fast Sampling of Diffusion Models with Exponential Integrator. In *International Conference on Learning Representations*, 2023. Page: 104

[315] Qinsheng Zhang, Molei Tao, and Yongxin Chen. gDDIM: Generalized denoising diffusion implicit models. *arXiv:2206.05564*, 2022. Pages: 90, 104, and 117

[316] Richard Zhang. Making Convolutional Networks Shift-Invariant Again. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7324–7334. PMLR, 09–15 Jun 2019. Page: 66

[317] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable Augmentation for Data-Efficient GAN Training. *Advances in Neural Information Processing Systems*, 33, 2020. Page: 32

[318] Hongkai Zheng, Weili Nie, Arash Vahdat, Kamyar Azizzadenesheli, and Anima Anandkumar. Fast Sampling of Diffusion Models via Operator Learning. *arXiv:2211.13449*, 2022. Page: 150

[319] Linqi Zhou, Yilun Du, and Jiajun Wu. 3D Shape Generation and Completion Through Point-Voxel Diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021. Pages: 15, 20, 95, and 165

[320] Alexander Ziller, Dmitrii Usynin, Rickmer Braren, Marcus Makowski, Daniel Rueckert, and Georgios Kaissis. Medical imaging deep learning with differential privacy. *Scientific Reports*, 11(1):1–8, 2021. Page: 165

[321] Alexander Ziller, Dmitrii Usynin, Nicolas Remerscheid, Moritz Knolle, Marcus Makowski, Rickmer Braren, Daniel Rueckert, and Georgios Kaissis. Differentially private federated deep learning for multi-site medical image segmentation. *arXiv:2107.02586*, 2021. Page: 165