Elucidating the Design Space of Diffusion-Based Generative Models

Tero Karras
NVIDIA
tkarras@nvidia.com

Miika Aittala
NVIDIA
maittala@nvidia.com

Timo Aila
NVIDIA
taila@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Abstract

We argue that the theory and practice of diffusion-based generative models are currently unnecessarily convoluted and seek to remedy the situation by presenting a design space that clearly separates the concrete design choices. This lets us identify several changes to both the sampling and training processes, as well as preconditioning of the score networks. Together, our improvements yield new state-of-the-art FID of 1.79 for CIFAR-10 in a class-conditional setting and 1.97 in an unconditional setting, with much faster sampling (35 network evaluations per image) than prior designs. To further demonstrate their modular nature, we show that our design changes dramatically improve both the efficiency and quality obtainable with pre-trained score networks from previous work, including improving the FID of a previously trained ImageNet-64 model from 2.07 to near-SOTA 1.55, and after re-training with our proposed improvements to a new SOTA of 1.36.

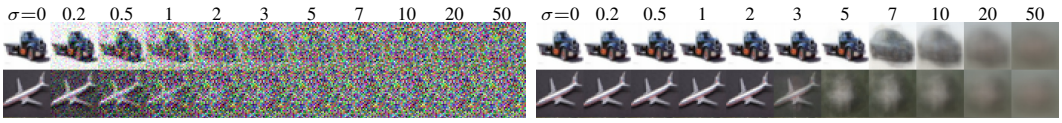
1 Introduction

Diffusion-based generative models [46] have emerged as a powerful new framework for neural image synthesis, in both unconditional [16, 37, 49] and conditional [17, 36, 37, 39, 40, 42, 43, 49] settings, even surpassing the quality of GANs [13] in certain situations [9]. They are also rapidly finding use in other domains such as audio [28, 38] and video [19] generation, image segmentation [4, 57] and language translation [35]. As such, there is great interest in applying these models and improving them further in terms of image/distribution quality, training cost, and generation speed.

The literature on these models is dense on theory, and derivations of sampling schedule, training dynamics, noise level parameterization, etc., tend to be based as directly as possible on theoretical frameworks, which ensures that the models are on a solid theoretical footing. However, this approach has a danger of obscuring the available design space — a proposed model may appear as a tightly coupled package where no individual component can be modified without breaking the entire system.

As our first contribution, we take a look at the theory behind these models from a practical standpoint, focusing more on the "tangible" objects and algorithms that appear in the training and sampling phases, and less on the statistical processes from which they might be derived. The goal is to obtain better insights into how these components are linked together and what degrees of freedom are available in the design of the overall system. We focus on the broad class of models where a neural network is used to model the score [22] of a noise level dependent marginal distribution of the training data corrupted by Gaussian noise. Thus, our work is in the context of *denoising score matching* [54].

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) Noisy images drawn from $p(\boldsymbol{x};\sigma)$ (b) Ideal denoiser outputs $D(\boldsymbol{x};\sigma)$

Figure 1: Denoising score matching on CIFAR-10. **(a)** Images from the training set corrupted with varying levels of additive Gaussian noise. High levels of noise lead to oversaturated colors; we normalize the images for cleaner visualization. **(b)** Optimal denoising result from minimizing Eq. 2 analytically (see Appendix B.3). With increasing noise level, the result approaches dataset mean.

Our second set of contributions concerns the sampling processes used to synthesize images using diffusion models. We identify the best-performing time discretization for sampling, apply a higher-order Runge–Kutta method for the sampling process, evaluate different sampler schedules, and analyze the usefulness of stochasticity in the sampling process. The result of these improvements is a significant drop in the number of sampling steps required during synthesis, and the improved sampler can be used as a drop-in replacement with several widely used diffusions models [37, 49].

The third set of contributions focuses on the training of the score-modeling neural network. While we continue to rely on the commonly used network architectures (DDPM [16], NCSN [48]), we provide the first principled analysis of the preconditioning of the networks' inputs, outputs, and loss functions in a diffusion model setting and derive best practices for improving the training dynamics. We also suggest an improved distribution of noise levels during training, and note that non-leaking augmentation [25] — typically used with GANs — is beneficial for diffusion models as well.

Taken together, our contributions enable significant improvements in result quality, e.g., leading to record FIDs of 1.79 for CIFAR-10 [29] and 1.36 for ImageNet [8] in 64×64 resolution. With all key ingredients of the design space explicitly tabulated, we believe that our approach will allow easier innovation on the individual components, and thus enable more extensive and targeted exploration of the design space of diffusion models. Our implementation and pre-trained models are available at `https://github.com/NVlabs/edm`

2 Expressing diffusion models in a common framework

Let us denote the data distribution by $p_{\text{data}}(\boldsymbol{x})$, with standard deviation σ_{data}, and consider the family of mollified distributions $p(\boldsymbol{x};\sigma)$ obtained by adding i.i.d. Gaussian noise of standard deviation σ to the data. For $\sigma_{\text{max}} \gg \sigma_{\text{data}}$, $p(\boldsymbol{x};\sigma_{\text{max}})$ is practically indistinguishable from pure Gaussian noise. The idea of diffusion models is to randomly sample a noise image $\boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{0}, \sigma_{\text{max}}^2 \mathbf{I})$, and sequentially denoise it into images $\boldsymbol{x}_i$ with noise levels $\sigma_0 = \sigma_{\text{max}} > \sigma_1 > \cdots > \sigma_N = 0$ so that at each noise level $\boldsymbol{x}_i \sim p(\boldsymbol{x}_i;\sigma_i)$. The endpoint $\boldsymbol{x}_N$ of this process is thus distributed according to the data.

Song et al. [49] present a stochastic differential equation (SDE) that maintains the desired distribution p as sample $\boldsymbol{x}$ evolves over time. This allows the above process to be implemented using a stochastic solver that both removes and adds noise at each iteration. They also give a corresponding "probability flow" ordinary differential equation (ODE) where the only source of randomness is the initial noise image $\boldsymbol{x}_0$. Contrary to the usual order of treatment, we begin by examining the ODE, as it offers a fruitful setting for analyzing sampling trajectories and their discretizations. The insights carry over to stochastic sampling, which we reintroduce as a generalization in Section 4.

ODE formulation. A probability flow ODE [49] continuously increases or reduces noise level of the image when moving forward or backward in time, respectively. To specify the ODE, we must first choose a schedule $\sigma(t)$ that defines the desired noise level at time t. For example, setting $\sigma(t) \propto \sqrt{t}$ is mathematically natural, as it corresponds to constant-speed heat diffusion [12]. However, we will show in Section 3 that the choice of schedule has major practical implications and should not be made on the basis of theoretical convenience.

The defining characteristic of the probability flow ODE is that evolving a sample $\boldsymbol{x}_a \sim p\big(\boldsymbol{x}_a;\sigma(t_a)\big)$ from time t_a to t_b (either forward or backward in time) yields a sample $\boldsymbol{x}_b \sim p\big(\boldsymbol{x}_b;\sigma(t_b)\big)$. Following previous work [49], this requirement is satisfied (see Appendix B.1 and B.2) by

$$\mathrm{d}\boldsymbol{x} = -\dot{\sigma}(t)\,\sigma(t)\,\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x};\sigma(t)\big)\,\mathrm{d}t, \tag{1}$$

Table 1: Specific design choices employed by different model families. N is the number of ODE solver iterations that we wish to execute during sampling. The corresponding sequence of time steps is $\{t_0, t_1, \ldots, t_N\}$, where $t_N = 0$. If the model was originally trained for specific choices of N and $\{t_i\}$, the originals are denoted by M and $\{u_j\}$, respectively. The denoiser is defined as $D_\theta(\boldsymbol{x};\sigma) = c_{\text{skip}}(\sigma)\boldsymbol{x} + c_{\text{out}}(\sigma)F_\theta\big(c_{\text{in}}(\sigma)\boldsymbol{x}; c_{\text{noise}}(\sigma)\big)$; F_θ represents the raw neural network layers.

		VP [49]	VE [49]	iDDPM [37] + DDIM [47]	Ours ("EDM")
Sampling (Section 3)					
ODE solver		Euler	Euler	Euler	2nd order Heun
Time steps	$t_{i<N}$	$1 + \frac{i}{N-1}(\epsilon_{\text{s}} - 1)$	$\sigma_{\text{max}}^2 \left(\sigma_{\text{min}}^2/\sigma_{\text{max}}^2\right)^{\frac{i}{N-1}}$	$u_{\lfloor j_0 + \frac{M-1-j_0}{N-1} i + \frac{1}{2} \rfloor}$, where $u_M = 0$, $u_{j-1} = \sqrt{\frac{u_j^2+1}{\max(\bar{\alpha}_{j-1}/\bar{\alpha}_j, C_1)} - 1}$	$\left(\sigma_{\text{max}}^{\frac{1}{\rho}} + \frac{i}{N-1}(\sigma_{\text{min}}^{\frac{1}{\rho}} - \sigma_{\text{max}}^{\frac{1}{\rho}})\right)^\rho$
Schedule	$\sigma(t)$	$\sqrt{e^{\frac{1}{2}\beta_{\text{d}} t^2 + \beta_{\text{min}} t} - 1}$	$\sqrt{t}$	t	t
Scaling	$s(t)$	$1/\sqrt{e^{\frac{1}{2}\beta_{\text{d}} t^2 + \beta_{\text{min}} t}}$	1	1	1
Network and preconditioning (Section 5)					
Architecture of F_θ		DDPM++	NCSN++	DDPM	(any)
Skip scaling	$c_{\text{skip}}(\sigma)$	1	1	1	$\sigma_{\text{data}}^2/\left(\sigma^2 + \sigma_{\text{data}}^2\right)$
Output scaling	$c_{\text{out}}(\sigma)$	$-\sigma$	σ	$-\sigma$	$\sigma \cdot \sigma_{\text{data}}/\sqrt{\sigma_{\text{data}}^2 + \sigma^2}$
Input scaling	$c_{\text{in}}(\sigma)$	$1/\sqrt{\sigma^2+1}$	1	$1/\sqrt{\sigma^2+1}$	$1/\sqrt{\sigma^2 + \sigma_{\text{data}}^2}$
Noise cond.	$c_{\text{noise}}(\sigma)$	$(M-1)\,\sigma^{-1}(\sigma)$	$\ln(\frac{1}{2}\sigma)$	$M - 1 - \arg\min_j \lvert u_j - \sigma \rvert$	$\frac{1}{4}\ln(\sigma)$
Training (Section 5)					
Noise distribution		$\sigma^{-1}(\sigma) \sim \mathcal{U}(\epsilon_{\text{t}}, 1)$	$\ln(\sigma) \sim \mathcal{U}(\ln(\sigma_{\text{min}}), \ln(\sigma_{\text{max}}))$	$\sigma = u_j,\ \ j \sim \mathcal{U}\{0, M-1\}$	$\ln(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$
Loss weighting	$\lambda(\sigma)$	$1/\sigma^2$	$1/\sigma^2$	$1/\sigma^2$ (note: *)	$\left(\sigma^2 + \sigma_{\text{data}}^2\right)/(\sigma \cdot \sigma_{\text{data}})^2$
Parameters		$\beta_{\text{d}} = 19.9, \beta_{\text{min}} = 0.1$ $\epsilon_{\text{s}} = 10^{-3}, \epsilon_{\text{t}} = 10^{-5}$ $M = 1000$	$\sigma_{\text{min}} = 0.02$ $\sigma_{\text{max}} = 100$	$\bar{\alpha}_j = \sin^2(\frac{\pi}{2}\frac{j}{M(C_2+1)})$ $C_1 = 0.001, C_2 = 0.008$ $M = 1000, j_0 = 8^\dagger$	$\sigma_{\text{min}} = 0.002, \sigma_{\text{max}} = 80$ $\sigma_{\text{data}} = 0.5, \rho = 7$ $P_{\text{mean}} = -1.2, P_{\text{std}} = 1.2$

* iDDPM also employs a second loss term L_{vlb} † In our tests, $j_0 = 8$ yielded better FID than $j_0 = 0$ used by iDDPM

where the dot denotes a time derivative. $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma)$ is the *score function* [22], a vector field that points towards higher density of data at a given noise level. Intuitively, an infinitesimal forward step of this ODE nudges the sample away from the data, at a rate that depends on the change in noise level. Equivalently, a backward step nudges the sample towards the data distribution.

Denoising score matching. The score function has the remarkable property that it does not depend on the generally intractable normalization constant of the underlying density function $p(\boldsymbol{x};\sigma)$ [22], and thus can be much easier to evaluate. Specifically, if $D(\boldsymbol{x};\sigma)$ is a denoiser function that minimizes the expected L_2 denoising error for samples drawn from p_{data} separately for every σ, i.e.,

$$\mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}} \mathbb{E}_{\boldsymbol{n}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})} \|D(\boldsymbol{y}+\boldsymbol{n};\sigma) - \boldsymbol{y}\|_2^2, \text{ then } \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma) = \big(D(\boldsymbol{x};\sigma) - \boldsymbol{x}\big)/\sigma^2, \quad (2,3)$$

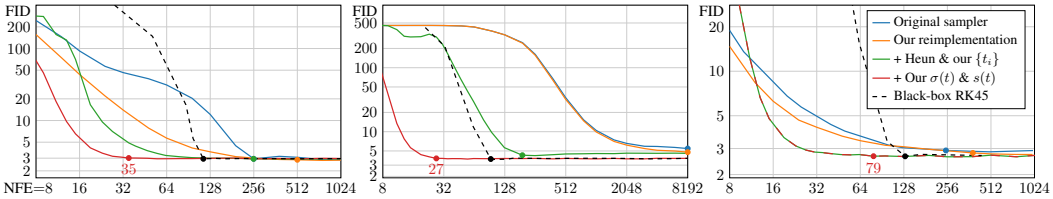
where $\boldsymbol{y}$ is a training image and $\boldsymbol{n}$ is noise. In this light, the score function isolates the noise component from the signal in $\boldsymbol{x}$, and Eq. 1 amplifies (or diminishes) it over time. Figure 1 illustrates the behavior of ideal D in practice. The key observation in diffusion models is that $D(\boldsymbol{x};\sigma)$ can be implemented as a neural network $D_\theta(\boldsymbol{x};\sigma)$ trained according to Eq. 2. Note that D_θ may include additional pre- and post-processing steps, such as scaling $\boldsymbol{x}$ to an appropriate dynamic range; we will return to such *preconditioning* in Section 5.

Time-dependent signal scaling. Some methods (see Appendix C.1) introduce an additional scale schedule $s(t)$ and consider $\boldsymbol{x} = s(t)\hat{\boldsymbol{x}}$ to be a scaled version of the original, non-scaled variable $\hat{\boldsymbol{x}}$. This changes the time-dependent probability density, and consequently also the ODE solution trajectories. The resulting ODE is a generalization of Eq. 1:

$$\mathrm{d}\boldsymbol{x} = \left[\frac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - s(t)^2\,\dot{\sigma}(t)\,\sigma(t)\,\nabla_{\boldsymbol{x}} \log p\left(\frac{\boldsymbol{x}}{s(t)}; \sigma(t)\right)\right]\,\mathrm{d}t. \quad (4)$$

Note that we explicitly undo the scaling of $\boldsymbol{x}$ when evaluating the score function to keep the definition of $p(\boldsymbol{x};\sigma)$ independent of $s(t)$.

Solution by discretization. The ODE to be solved is obtained by substituting Eq. 3 into Eq. 4 to define the point-wise gradient, and the solution can be found by numerical integration, i.e., taking



(a) Uncond. CIFAR-10, VP ODE (b) Uncond. CIFAR-10, VE ODE (c) Class-cond. ImageNet-64, DDIM

Figure 2: Comparison of deterministic sampling methods using three pre-trained models. For each curve, the dot indicates the lowest NFE whose FID is within 3% of the lowest observed FID.

finite steps over discrete time intervals. This requires choosing both the integration scheme (e.g., Euler or a variant of Runge–Kutta), as well as the discrete sampling times $\{t_0, t_1, \ldots, t_N\}$. Many prior works rely on Euler's method, but we show in Section 3 that a 2nd order solver offers a better computational tradeoff. For brevity, we do not provide a separate pseudocode for Euler's method applied to our ODE here, but it can be extracted from Algorithm 1 by omitting lines 6–8.

Putting it together. Table 1 presents formulas for reproducing deterministic variants of three earlier methods in our framework. These methods were chosen because they are widely used and achieve state-of-the-art performance, but also because they were derived from different theoretical foundations. Some of our formulas appear quite different from the original papers as indirection and recursion have been removed; see Appendix C for details. The main purpose of this reframing is to bring into light all the independent components that often appear tangled together in previous work. In our framework, there are no implicit dependencies between the components — any choices (within reason) for the individual formulas will, in principle, lead to a functioning model. In other words, changing one component does not necessitate changes elsewhere in order to, e.g., maintain the property that the model converges to the data in the limit. In practice, some choices and combinations will of course work better than others.

3 Improvements to deterministic sampling

Improving the output quality and/or decreasing the computational cost of sampling are common topics in diffusion model research (e.g., [10, 24, 31, 32, 33, 37, 44, 53, 55, 56, 59]). Our hypothesis is that the choices related to the sampling process are largely independent of the other components, such as network architecture and training details. In other words, the training procedure of D_θ should not dictate $\sigma(t)$, $s(t)$, and $\{t_i\}$, nor vice versa; from the viewpoint of the sampler, D_θ is simply a black box [55, 56]. We test this by evaluating different samplers on three *pre-trained* models, each representing a different theoretical framework and model family. We first measure baseline results for these models using their original sampler implementations, and then bring these samplers into our unified framework using the formulas in Table 1, followed by our improvements. This allows us to evaluate different practical choices and propose general improvements to the sampling process that are applicable to all models.

We evaluate the "DDPM++ cont. (VP)" and "NCSN++ cont. (VE)" models by Song et al. [49] trained on unconditional CIFAR-10 [29] at 32×32, corresponding to the variance preserving (VP) and variance exploding (VE) formulations [49], originally inspired by DDPM [16] and SMLD [48]. We also evaluate the "ADM (dropout)" model by Dhariwal and Nichol [9] trained on class-conditional ImageNet [8] at 64×64, corresponding to the improved DDPM (iDDPM) formulation [37]. This model was trained using a discrete set of $M = 1000$ noise levels. Further details are given in Appendix C.

We evaluate the result quality in terms of Fréchet inception distance (FID) [15] computed between 50,000 generated images and all available real images. Figure 2 shows FID as a function of neural function evaluations (NFE), i.e., how many times D_θ is evaluated to produce a single image. Given that the sampling process is dominated entirely by the cost of D_θ, improvements in NFE translate directly to sampling speed. The original deterministic samplers are shown in blue, and the reimplementations of these methods in our unified framework (orange) yield similar but consistently better results. The differences are explained by certain oversights in the original implementations as well as our more careful treatment of discrete noise levels in the case of DDIM; see Appendix C. Note that our reimplementations are fully specified by Algorithm 1 and Table 1, even though the original codebases are structured very differently from each other.

Algorithm 1 Deterministic sampling using Heun's 2nd order method with arbitrary $\sigma(t)$ and $s(t)$.

procedure HEUNSAMPLER($D_\theta(\boldsymbol{x};\sigma),\ \sigma(t),\ s(t),\ t_{i\in\{0,\ldots,N\}}$)
  sample $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ \sigma^2(t_0)\, s^2(t_0)\, \mathbf{I}\big)$ ▷ Generate initial sample at t_0
  for $i \in \{0,\ldots,N-1\}$ **do** ▷ Solve Eq. 4 over N time steps
    $\boldsymbol{d}_i \leftarrow \left(\frac{\dot\sigma(t_i)}{\sigma(t_i)} + \frac{\dot s(t_i)}{s(t_i)}\right)\boldsymbol{x}_i - \frac{\dot\sigma(t_i)s(t_i)}{\sigma(t_i)} D_\theta\left(\frac{\boldsymbol{x}_i}{s(t_i)};\sigma(t_i)\right)$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at t_i
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\boldsymbol{d}_i$ ▷ Take Euler step from t_i to t_{i+1}
    if $\sigma(t_{i+1}) \neq 0$ **then** ▷ Apply 2nd order correction unless σ goes to zero
      $\boldsymbol{d}'_i \leftarrow \left(\frac{\dot\sigma(t_{i+1})}{\sigma(t_{i+1})} + \frac{\dot s(t_{i+1})}{s(t_{i+1})}\right)\boldsymbol{x}_{i+1} - \frac{\dot\sigma(t_{i+1})s(t_{i+1})}{\sigma(t_{i+1})} D_\theta\left(\frac{\boldsymbol{x}_{i+1}}{s(t_{i+1})};\sigma(t_{i+1})\right)$ ▷ Eval. $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at t_{i+1}
      $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\big(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}'_i\big)$ ▷ Explicit trapezoidal rule at t_{i+1}
  return $\boldsymbol{x}_N$ ▷ Return noise-free sample at t_N

Discretization and higher-order integrators. Solving an ODE numerically is necessarily an approximation of following the true solution trajectory. At each step, the solver introduces *truncation error* that accumulates over the course of N steps. The local error generally scales superlinearly with respect to step size, and thus increasing N improves the accuracy of the solution.

The commonly used Euler's method is a first order ODE solver with $\mathcal{O}(h^2)$ local error with respect to step size h. Higher-order Runge–Kutta methods [50] scale more favorably but require multiple evaluations of D_θ per step. Linear multistep methods have also been recently proposed for sampling diffusion models [31, 59]. Through extensive tests, we have found Heun's 2nd order method [2] (a.k.a. improved Euler, trapezoidal rule) — previously explored in the context of diffusion models by Jolicoeur-Martineau et al. [24] — to provide an excellent tradeoff between truncation error and NFE. As illustrated in Algorithm 1, it introduces an additional correction step for $\boldsymbol{x}_{i+1}$ to account for change in $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ between t_i and t_{i+1}. This correction leads to $\mathcal{O}(h^3)$ local error at the cost of one additional evaluation of D_θ per step. Note that stepping to $\sigma = 0$ would result in a division by zero, so we revert to Euler's method in this case. We discuss the general family of 2nd order solvers in Appendix D.2.

The time steps $\{t_i\}$ determine how the step sizes and thus truncation errors are distributed between different noise levels. We provide a detailed analysis in Appendix D.1, concluding that the step size should decrease monotonically with decreasing σ and it does not need to vary on a per-sample basis. We adopt a parameterized scheme where the time steps are defined according to a sequence of noise levels $\{\sigma_i\}$, i.e., $t_i = \sigma^{-1}(\sigma_i)$. We set $\sigma_{i<N} = (Ai+B)^\rho$ and select the constants A and B so that $\sigma_0 = \sigma_{\text{max}}$ and $\sigma_{N-1} = \sigma_{\text{min}}$, which gives

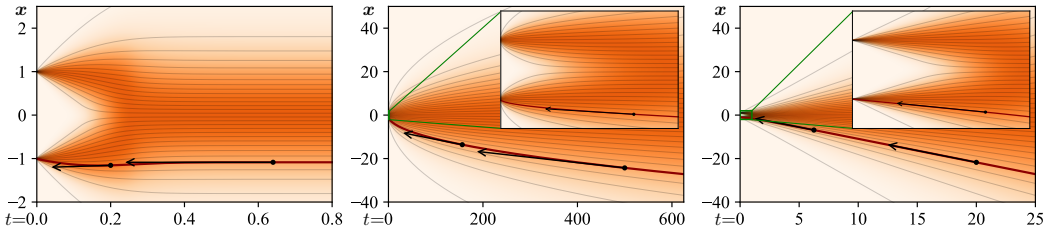
$$\sigma_{i<N} = \big(\sigma_{\text{max}}{}^{\frac{1}{\rho}} + \tfrac{i}{N-1}(\sigma_{\text{min}}{}^{\frac{1}{\rho}} - \sigma_{\text{max}}{}^{\frac{1}{\rho}})\big)^\rho \quad \text{and} \quad \sigma_N = 0. \tag{5}$$

Here ρ controls how much the steps near σ_{min} are shortened at the expense of longer steps near σ_{max}. Our analysis in Appendix D.1 shows that setting $\rho = 3$ nearly equalizes the truncation error at each step, but that ρ in range of 5 to 10 performs much better for sampling images. This suggests that errors near σ_{min} have a large impact. We set $\rho = 7$ for the remainder of this paper.

Results for Heun's method and Eq. 5 are shown as the green curves in Figure 2. We observe consistent improvement in all cases: Heun's method reaches the same FID as Euler's method with considerably lower NFE.

Trajectory curvature and noise schedule. The shape of the ODE solution trajectories is defined by functions $\sigma(t)$ and $s(t)$. The choice of these functions offers a way to reduce the truncation errors discussed above, as their magnitude can be expected to scale proportional to the curvature of $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$. We argue that the best choice for these functions is $\sigma(t) = t$ and $s(t) = 1$, which is also the choice made in DDIM [47]. With this choice, the ODE of Eq. 4 simplifies to $\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \big(\boldsymbol{x} - D(\boldsymbol{x};t)\big)/t$ and σ and t become interchangeable.

An immediate consequence is that at any $\boldsymbol{x}$ and t, a single Euler step to $t = 0$ yields the denoised image $D_\theta(\boldsymbol{x};t)$. The tangent of the solution trajectory therefore always points towards the denoiser output. This can be expected to change only slowly with the noise level, which corresponds to largely linear solution trajectories. The 1D ODE sketch of Figure 3c supports this intuition; the solution trajectories approach linear at both large and small noise levels, and have substantial curvature in only a small region in between. The same effect can be seen with real data in Figure 1b, where the



(a) Variance preserving ODE [49] (b) Variance exploding ODE [49] (c) DDIM [47] / Our ODE

Figure 3: A sketch of ODE curvature in 1D where p_{data} is two Dirac peaks at $\boldsymbol{x} = \pm 1$. Horizontal t axis is chosen to show $\sigma \in [0, 25]$ in each plot, with insets showing $\sigma \in [0, 1]$ near the data. Example local gradients are shown with black arrows. **(a)** Variance preserving ODE of Song et al. [49] has solution trajectories that flatten out to horizontal lines at large σ. Local gradients start pointing towards data only at small σ. **(b)** Variance exploding variant has extreme curvature near data and the solution trajectories are curved everywhere. **(c)** With the schedule used by DDIM [47] and us, as σ increases the solution trajectories approach straight lines that point towards the mean of data. As $\sigma \to 0$, the trajectories become linear and point towards the data manifold.

change between different denoiser targets occurs in a relatively narrow σ range. With the advocated schedule, this corresponds to high ODE curvature being limited to this same range.

The effect of setting $\sigma(t) = t$ and $s(t) = 1$ is shown as the red curves in Figure 2. As DDIM already employs these same choices, the red curve is identical to the green one for ImageNet-64. However, VP and VE benefit considerably from switching away from their original schedules.

Discussion. The choices that we made in this section to improve deterministic sampling are summarized in the *Sampling* part of Table 1. Together, they reduce the NFE needed to reach high-quality results by a large factor: 7.3× for VP, 300× for VE, and 3.2× for DDIM, corresponding to the highlighted NFE values in Figure 2. In practice, we can generate 26.3 high-quality CIFAR-10 images per second on a single NVIDIA V100. The consistency of improvements corroborates our hypothesis that the sampling process is orthogonal to how each model was originally trained. As further validation, we show results for the adaptive RK45 method [11] using our schedule as the dashed black curves in Figure 2; the cost of this sophisticated ODE solver outweighs its benefits.

4 Stochastic sampling

Deterministic sampling offers many benefits, e.g., the ability to turn real images into their corresponding latent representations by inverting the ODE. However, it tends to lead to worse output quality [47, 49] than stochastic sampling that injects fresh noise into the image in each step. Given that ODEs and SDEs recover the same distributions in theory, what exactly is the role of stochasticity?

Background. The SDEs of Song et al. [49] can be generalized [20, 58] as a sum of the probability flow ODE of Eq. 1 and a time-varying *Langevin diffusion* SDE [14] (see Appendix B.5):

$$\mathrm{d}\boldsymbol{x}_{\pm} = \underbrace{-\dot{\sigma}(t)\sigma(t)\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}; \sigma(t)\big)\,\mathrm{d}t}_{\text{probability flow ODE (Eq. 1)}} \pm \underbrace{\underbrace{\beta(t)\sigma(t)^2\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}; \sigma(t)\big)\,\mathrm{d}t}_{\text{deterministic noise decay}} + \underbrace{\sqrt{2\beta(t)}\sigma(t)\,\mathrm{d}\omega_t}_{\text{noise injection}}}_{\text{Langevin diffusion SDE}}, \quad (6)$$

where ω_t is the standard Wiener process. $\mathrm{d}\boldsymbol{x}_+$ and $\mathrm{d}\boldsymbol{x}_-$ are now separate SDEs for moving forward and backward in time, related by the time reversal formula of Anderson [1]. The Langevin term can further be seen as a combination of a deterministic score-based denoising term and a stochastic noise injection term, whose net noise level contributions cancel out. As such, $\beta(t)$ effectively expresses the relative rate at which existing noise is replaced with new noise. The SDEs of Song et al. [49] are recovered with the choice $\beta(t) = \dot{\sigma}(t)/\sigma(t)$, whereby the score vanishes from the forward SDE.

This perspective reveals why stochasticity is helpful in practice: The implicit Langevin diffusion drives the sample towards the desired marginal distribution at a given time, actively correcting for any errors made in earlier sampling steps. On the other hand, approximating the Langevin term

Algorithm 2 Our stochastic sampler with $\sigma(t) = t$ and $s(t) = 1$.

1: **procedure** STOCHASTICSAMPLER($D_\theta(\boldsymbol{x};\sigma),\ t_{i\in\{0,\ldots,N\}},\ \gamma_{i\in\{0,\ldots,N-1\}},\ S_{\text{noise}}$)
2: **sample** $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ t_0^2\,\mathbf{I}\big)$
3: **for** $i \in \{0,\ldots,N-1\}$ **do** ▷ $\gamma_i = \begin{cases} \min\left(\frac{S_{\text{churn}}}{N}, \sqrt{2}-1\right) & \text{if } t_i \in [S_{\text{tmin}}, S_{\text{tmax}}] \\ 0 & \text{otherwise} \end{cases}$
4: **sample** $\boldsymbol{\epsilon}_i \sim \mathcal{N}\big(\mathbf{0},\ S_{\text{noise}}^2\,\mathbf{I}\big)$
5: $\hat{t}_i \leftarrow t_i + \gamma_i t_i$ ▷ Select temporarily increased noise level $\hat{t}_i$
6: $\hat{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i + \sqrt{\hat{t}_i^2 - t_i^2}\,\boldsymbol{\epsilon}_i$ ▷ Add new noise to move from t_i to $\hat{t}_i$
7: $\boldsymbol{d}_i \leftarrow \big(\hat{\boldsymbol{x}}_i - D_\theta(\hat{\boldsymbol{x}}_i;\hat{t}_i)\big)/\hat{t}_i$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $\hat{t}_i$
8: $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\boldsymbol{d}_i$ ▷ Take Euler step from $\hat{t}_i$ to t_{i+1}
9: **if** $t_{i+1} \neq 0$ **then**
10: $\boldsymbol{d}_i' \leftarrow \big(\boldsymbol{x}_{i+1} - D_\theta(\boldsymbol{x}_{i+1};t_{i+1})\big)/t_{i+1}$ ▷ Apply 2nd order correction
11: $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\big(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}_i'\big)$
12: **return** $\boldsymbol{x}_N$

with discrete SDE solver steps introduces error in itself. Previous results [3, 24, 47, 49] suggest that non-zero $\beta(t)$ is helpful, but as far as we can tell, the implicit choice for $\beta(t)$ in Song et al. [49] enjoys no special properties. Hence, the optimal amount of stochasticity should be determined empirically.

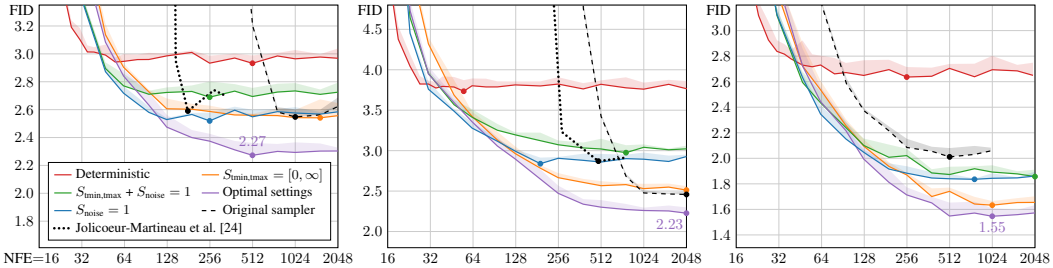
Our stochastic sampler. We propose a stochastic sampler that combines our 2nd order deterministic ODE integrator with explicit Langevin-like "churn" of adding and removing noise. A pseudocode is given in Algorithm 2. At each step i, given the sample $\boldsymbol{x}_i$ at noise level t_i $(= \sigma(t_i))$, we perform two sub-steps. First, we add noise to the sample according to a factor $\gamma_i \geq 0$ to reach a higher noise level $\hat{t}_i = t_i + \gamma_i t_i$. Second, from the increased-noise sample $\hat{\boldsymbol{x}}_i$, we solve the ODE backward from $\hat{t}_i$ to t_{i+1} with a single step. This yields a sample $\boldsymbol{x}_{i+1}$ with noise level t_{i+1}, and the iteration continues. We stress that this is not a general-purpose SDE solver, but a sampling procedure tailored for the specific problem. Its correctness stems from the alternation of two sub-steps that each maintain the correct distribution (up to truncation error in the ODE step). The predictor-corrector sampler of Song et al. [49] has a conceptually similar structure to ours.

To analyze the main difference between our method and Euler–Maruyama, we first note a subtle discrepancy in the latter when discretizing Eq. 6. One can interpret Euler–Maruyama as first adding noise and then performing an ODE step, not from the intermediate state after noise injection, but assuming that $\boldsymbol{x}$ and σ remained at the initial state at the beginning of the iteration step. In our method, the parameters used to evaluate D_θ on line 7 of Algorithm 2 correspond to the state after noise injection, whereas an Euler–Maruyama -like method would use $\boldsymbol{x}_i; t_i$ instead of $\hat{\boldsymbol{x}}_i; \hat{t}_i$. In the limit of Δ_t approaching zero there may be no difference between these choices, but the distinction appears to become significant when pursuing low NFE with large steps.

Practical considerations. Increasing the amount of stochasticity is effective in correcting errors made by earlier sampling steps, but it has its own drawbacks. We have observed (see Appendix E.1) that excessive Langevin-like addition and removal of noise results in gradual loss of detail in the generated images with all datasets and denoiser networks. There is also a drift toward oversaturated colors at very low and high noise levels. We suspect that practical denoisers induce a slightly non-conservative vector field in Eq. 3, violating the premises of Langevin diffusion and causing these detrimental effects. Notably, our experiments with analytical denoisers (such as the one in Figure 1b) have not shown such degradation.

If the degradation is caused by flaws in $D_\theta(\boldsymbol{x};\sigma)$, they can only be remedied using heuristic means during sampling. We address the drift toward oversaturated colors by only enabling stochasticity within a specific range of noise levels $t_i \in [S_{\text{tmin}}, S_{\text{tmax}}]$. For these noise levels, we define $\gamma_i = S_{\text{churn}}/N$, where S_{churn} controls the overall amount of stochasticity. We further clamp γ_i to never introduce more new noise than what is already present in the image. Finally, we have found that the loss of detail can be partially counteracted by setting S_{noise} slightly above 1 to inflate the standard deviation for the newly added noise. This suggests that a major component of the hypothesized non-conservativity of $D_\theta(\boldsymbol{x};\sigma)$ is a tendency to remove slightly too much noise — most likely due to regression toward the mean that can be expected to happen with any L_2-trained denoiser [30].

Evaluation. Figure 4 shows that our stochastic sampler outperforms previous samplers [24, 37, 49] by a significant margin, especially at low step counts. Jolicoeur-Martineau et al. [24] use a standard



(a) Uncond. CIFAR-10, VP (b) Uncond. CIFAR-10, VE (c) Class-cond. ImageNet-64

Figure 4: Evaluation of our stochastic sampler (Algorithm 2). The purple curve corresponds to optimal choices for $\{S_{\text{churn}}, S_{\text{tmin}}, S_{\text{tmax}}, S_{\text{noise}}\}$; orange, blue, and green correspond to disabling the effects of $S_{\text{tmin,tmax}}$ and/or S_{noise}. The red curves show reference results for our deterministic sampler (Algorithm 1), equivalent to setting $S_{\text{churn}} = 0$. The dashed black curves correspond to the original stochastic samplers from previous work: Euler–Maruyama [49] for VP, predictor-corrector [49] for VE, and iDDPM [37] for ImageNet-64. The dots indicate lowest observed FID.

higher-order adaptive SDE solver [41] and its performance is a good baseline for such solvers in general. Our sampler has been tailored to the use case by, e.g., performing noise injection and ODE step sequentially, and it is not adaptive. It is an open question if adaptive solvers can be a net win over a well-tuned fixed schedule in sampling diffusion models.

Through sampler improvements alone, we are able to bring the ImageNet-64 model that originally achieved FID 2.07 [9] to 1.55 that is very close to the state-of-the-art; previously, FID 1.48 has been reported for cascaded diffusion [17], 1.55 for classifier-free guidance [18], and 1.52 for StyleGAN-XL [45]. While our results showcase the potential gains achievable through sampler improvements, they also highlight the main shortcoming of stochasticity: For best results, one must make several heuristic choices — either implicit or explicit — that depend on the specific model. Indeed, we had to find the optimal values of $\{S_{\text{churn}}, S_{\text{tmin}}, S_{\text{tmax}}, S_{\text{noise}}\}$ on a case-by-case basis using grid search (Appendix E.2). This raises a general concern that using stochastic sampling as the primary means of evaluating model improvements may inadvertently end up influencing the design choices related to model architecture and training.

5 Preconditioning and training

There are various known good practices for training neural networks in a supervised fashion. For example, it is advisable to keep input and output signal magnitudes fixed to, e.g., unit variance, and to avoid large variation in gradient magnitudes on a per-sample basis [5, 21]. Training a neural network to model D directly would be far from ideal — for example, as the input $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{n}$ is a combination of clean signal $\boldsymbol{y}$ and noise $\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, its magnitude varies immensely depending on noise level σ. For this reason, the common practice is to not represent D_θ as a neural network directly, but instead train a different network F_θ from which D_θ is derived.

Previous methods [37, 47, 49] address the input scaling via a σ-dependent normalization factor and attempt to precondition the output by training F_θ to predict $\boldsymbol{n}$ scaled to unit variance, from which the signal is then reconstructed via $D_\theta(\boldsymbol{x}; \sigma) = \boldsymbol{x} - \sigma F_\theta(\cdot)$. This has the drawback that at large σ, the network needs to fine-tune its output carefully to cancel out the existing noise $\boldsymbol{n}$ exactly and give the output at the correct scale; note that any errors made by the network are amplified by a factor of σ. In this situation, it would seem much easier to predict the expected output $D(\boldsymbol{x}; \sigma)$ directly. In the same spirit as previous parameterizations that adaptively mix signal and noise (e.g., [10, 44, 53]), we propose to precondition the neural network with a σ-dependent skip connection that allows it to estimate either $\boldsymbol{y}$ or $\boldsymbol{n}$, or something in between. We thus write D_θ in the following form:

$$D_\theta(\boldsymbol{x}; \sigma) = c_{\text{skip}}(\sigma)\, \boldsymbol{x} + c_{\text{out}}(\sigma)\, F_\theta\big(c_{\text{in}}(\sigma)\, \boldsymbol{x};\ c_{\text{noise}}(\sigma)\big), \tag{7}$$

where F_θ is the neural network to be trained, $c_{\text{skip}}(\sigma)$ modulates the skip connection, $c_{\text{in}}(\sigma)$ and $c_{\text{out}}(\sigma)$ scale the input and output magnitudes, and $c_{\text{noise}}(\sigma)$ maps noise level σ into a conditioning input for F_θ. Taking a weighted expectation of Eq. 2 over the noise levels gives the overall training loss $\mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\left[\lambda(\sigma)\, \|D(\boldsymbol{y} + \boldsymbol{n}; \sigma) - \boldsymbol{y}\|_2^2\right]$, where $\sigma \sim p_{\text{train}}$, $\boldsymbol{y} \sim p_{\text{data}}$, and $\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The probability of sampling a given noise level σ is given by $p_{\text{train}}(\sigma)$ and the corresponding weight is given by

Table 2: Evaluation of our training improvements. The starting point (config A) is VP & VE using our **deterministic** sampler. At the end (configs E,F), VP & VE only differ in the architecture of F_θ.

	CIFAR-10 [29] at 32×32				FFHQ [27] 64×64		AFHQv2 [7] 64×64	
	Conditional		Unconditional		Unconditional		Unconditional	
Training configuration	VP	VE	VP	VE	VP	VE	VP	VE
A Baseline [49] (*pre-trained)	2.48	3.11	3.01*	3.77*	3.39	25.95	2.58	18.52
B + Adjust hyperparameters	2.18	2.48	2.51	2.94	3.13	22.53	2.43	23.12
C + Redistribute capacity	2.08	2.52	2.31	2.83	2.78	41.62	2.54	15.04
D + Our preconditioning	2.09	2.64	2.29	3.10	2.94	3.39	2.79	3.81
E + Our loss function	1.88	1.86	2.05	1.99	2.60	2.81	2.29	2.28
F + Non-leaky augmentation	**1.79**	**1.79**	**1.97**	**1.98**	**2.39**	**2.53**	**1.96**	**2.16**
NFE	35	35	35	35	79	79	79	79

$\lambda(\sigma)$. We can equivalently express this loss with respect to the raw network output F_θ in Eq. 7:

$$\mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\underbrace{\lambda(\sigma)\,c_{\text{out}}(\sigma)^2}_{\text{effective weight}}\,\big\|\,\underbrace{F_\theta\big(c_{\text{in}}(\sigma)\cdot(\boldsymbol{y}+\boldsymbol{n});c_{\text{noise}}(\sigma)\big)}_{\text{network output}}-\underbrace{\tfrac{1}{c_{\text{out}}(\sigma)}\big(\boldsymbol{y}-c_{\text{skip}}(\sigma)\cdot(\boldsymbol{y}+\boldsymbol{n})\big)}_{\text{effective training target}}\,\big\|_2^2\Big]. \quad (8)$$

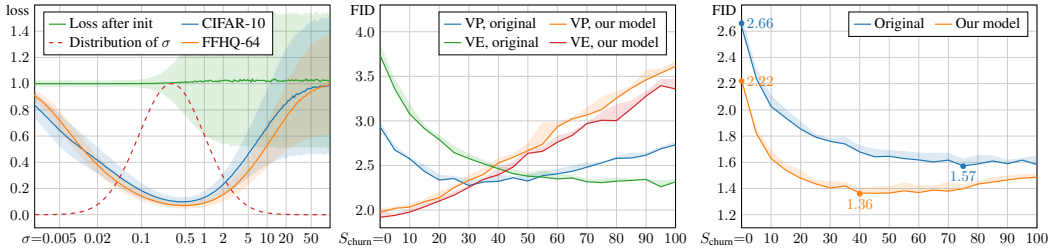
This form reveals the effective training target of F_θ, allowing us to determine suitable choices for the preconditioning functions from first principles. As detailed in Appendix B.6, we derive our choices shown in Table 1 by requiring network inputs and training targets to have unit variance (c_{in}, c_{out}), and amplifying errors in F_θ as little as possible (c_{skip}). The formula for c_{noise} is chosen empirically.

Table 2 shows FID for a series of training setups, evaluated using our deterministic sampler from Section 3. We start with the baseline training setup of Song et al. [49], which differs considerably between the VP and VE cases; we provide separate results for each (config A). To obtain a more meaningful point of comparison, we re-adjust the basic hyperparameters (config B) and improve the expressive power of the model (config C) by removing the lowest-resolution layers and doubling the capacity of the highest-resolution layers instead; see Appendix F.3 for further details. We then replace the original choices of $\{c_{\text{in}}, c_{\text{out}}, c_{\text{noise}}, c_{\text{skip}}\}$ with our preconditioning (config D), which keeps the results largely unchanged — except for VE that improves considerably at 64×64 resolution. Instead of improving FID per se, the main benefit of our preconditioning is that it makes the training more robust, enabling us to turn our focus on redesigning the loss function without adverse effects.

Loss weighting and sampling. Eq. 8 shows that training F_θ as preconditioned in Eq. 7 incurs an effective per-sample loss weight of $\lambda(\sigma)c_{\text{out}}(\sigma)^2$. To balance the effective loss weights, we set $\lambda(\sigma) = 1/c_{\text{out}}(\sigma)^2$, which also equalizes the initial training loss over the entire σ range as shown in Figure 5a (green curve). Finally, we need to select $p_{\text{train}}(\sigma)$, i.e., how to choose noise levels during training. Inspecting the per-σ loss after training (blue and orange curves) reveals that a significant reduction is possible only at intermediate noise levels; at very low levels, it is both difficult and irrelevant to discern the vanishingly small noise component, whereas at high levels the training targets are always dissimilar from the correct answer that approaches dataset average. Therefore, we target the training efforts to the relevant range using a simple log-normal distribution for $p_{\text{train}}(\sigma)$ as detailed in Table 1 and illustrated in Figure 5a (red curve).

Table 2 shows that our proposed p_{train} and λ (config E) lead to a dramatic improvement in FID in all cases when used in conjunction with our preconditioning (config D). In concurrent work, Choi et al. [6] propose a similar scheme to prioritize noise levels that are most relevant w.r.t. forming the perceptually recognizable content of the image. However, they only consider the choice of λ in isolation, which results in a smaller overall improvement.

Augmentation regularization. To prevent potential overfitting that often plagues diffusion models with smaller datasets, we borrow an augmentation pipeline from the GAN literature [25]. The pipeline consists of various geometric transformations (see Appendix F.2) that we apply to a training image prior to adding noise. To prevent the augmentations from leaking to the generated images, we provide the augmentation parameters as a conditioning input to F_θ; during inference we set the them to zero to guarantee that only non-augmented images are generated. Table 2 shows that data augmentation provides a consistent improvement (config F) that yields new state-of-the-art FIDs of 1.79 and 1.97 for conditional and unconditional CIFAR-10, beating the previous records of 1.85 [45] and 2.10 [53].



(a) Loss & noise distribution (b) Stochasticity on CIFAR-10 (c) Stochasticity on ImageNet-64

Figure 5: **(a)** Observed initial (green) and final loss per noise level, representative of the the 32×32 (blue) and 64×64 (orange) models considered in this paper. The shaded regions represent the standard deviation over 10k random samples. Our proposed training sample density is shown by the dashed red curve. **(b)** Effect of S_{churn} on unconditional CIFAR-10 with 256 steps (NFE = 511). For the original training setup of Song et al. [49], stochastic sampling is highly beneficial (blue, green), while deterministic sampling ($S_{\text{churn}} = 0$) leads to relatively poor FID. For our training setup, the situation is reversed (orange, red); stochastic sampling is not only unnecessary but harmful. **(c)** Effect of S_{churn} on class-conditional ImageNet-64 with 256 steps (NFE = 511). In this more challenging scenario, stochastic sampling turns out to be useful again. Our training setup improves the results for both deterministic and stochastic sampling.

Stochastic sampling revisited. Interestingly, the relevance of stochastic sampling appears to diminish as the model itself improves, as shown in Figure 5b,c. When using our training setup in CIFAR-10 (Figure 5b), the best results were obtained with deterministic sampling, and any amount of stochastic sampling was detrimental.

ImageNet-64. As a final experiment, we trained a class-conditional ImageNet-64 model from scratch using our proposed training improvements. This model achieved a new state-of-the-art FID of 1.36 compared to the previous record of 1.48 [17]. We used the ADM architecture [9] with no changes, and trained it using our config E with minimal tuning; see Appendix F.3 for details. We did not find overfitting to be a concern, and thus chose to not employ augmentation regularization. As shown in Figure 5c, the optimal amount of stochastic sampling was much lower than with the pre-trained model, but unlike with CIFAR-10, stochastic sampling was clearly better than deterministic sampling. This suggests that more diverse datasets continue to benefit from stochastic sampling.

6 Conclusions

Our approach of putting diffusion models to a common framework exposes a modular design. This allows a targeted investigation of individual components, potentially helping to better cover the viable design space. In our tests this let us simply replace the samplers in various earlier models, drastically improving the results. For example, in ImageNet-64 our sampler turned an average model (FID 2.07) to a challenger (1.55) for the previous SOTA model (1.48) [17], and with training improvements achieved SOTA FID of 1.36. We also obtained new state-of-the-art results on CIFAR-10 while using only 35 model evaluations, deterministic sampling, and a small network. The current high-resolution diffusion models rely either on separate super-resolution steps [17, 36, 40], subspace projection [23], very large networks [9, 49], or hybrid approaches [39, 42, 53] — we believe that our contributions are orthogonal to these extensions. That said, many of our parameter values may need to be re-adjusted for higher resolution datasets. Furthermore, we feel that the precise interaction between stochastic sampling and the training objective remains an interesting question for future work.

Societal impact. Our advances in sample quality can potentially amplify negative societal effects when used in a large-scale system like DALL·E 2, including types of disinformation or emphasizing sterotypes and harmful biases [34]. The training and sampling of diffusion models needs a lot of electricity; our project consumed ∼250MWh on an in-house cluster of NVIDIA V100s.

Acknowledgments. We thank Jaakko Lehtinen, Ming-Yu Liu, Tuomas Kynkäänniemi, Axel Sauer, Arash Vahdat, and Janne Hellsten for discussions and comments, and Tero Kuosmanen, Samuel Klenberg, and Janne Hellsten for maintaining our compute infrastructure.

References

[1] B. D. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[2] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.

[3] F. Bao, C. Li, J. Zhu, and B. Zhang. Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *Proc. ICLR*, 2022.

[4] D. Baranchuk, A. Voynov, I. Rubachev, V. Khrulkov, and A. Babenko. Label-efficient semantic segmentation with diffusion models. In *Proc. ICLR*, 2022.

[5] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[6] J. Choi, J. Lee, C. Shin, S. Kim, H. Kim, and S. Yoon. Perception prioritized training of diffusion models. In *Proc. CVPR*, 2022.

[7] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha. StarGAN v2: Diverse image synthesis for multiple domains. In *Proc. CVPR*, 2020.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009.

[9] P. Dhariwal and A. Q. Nichol. Diffusion models beat GANs on image synthesis. In *Proc. NeurIPS*, 2021.

[10] T. Dockhorn, A. Vahdat, and K. Kreis. Score-based generative modeling with critically-damped Langevin diffusion. In *Proc. ICLR*, 2022.

[11] 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.

[12] J. B. J. Fourier, G. Darboux, et al. *Théorie analytique de la chaleur*, volume 504. Didot Paris, 1822.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. In *Proc. NIPS*, 2014.

[14] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(4):549–581, 1994.

[15] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Proc. NIPS*, 2017.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Proc. NeurIPS*, 2020.

[17] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23, 2022.

[18] J. Ho and T. Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[19] J. Ho, T. Salimans, A. A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. In *Proc. ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022.

[20] C.-W. Huang, J. H. Lim, and A. C. Courville. A variational perspective on diffusion-based generative models and score matching. In *Proc. NeurIPS*, 2021.

[21] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao. Normalization techniques in training DNNs: Methodology, analysis and application. *CoRR*, abs/2009.12836, 2020.

[22] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[23] B. Jing, G. Corso, R. Berlinghieri, and T. Jaakkola. Subspace diffusion generative models. In *Proc. ECCV*, 2022.

[24] A. Jolicoeur-Martineau, K. Li, R. Piché-Taillefer, T. Kachman, and I. Mitliagkas. Gotta go fast when generating data with score-based models. *CoRR*, abs/2105.14080, 2021.

[25] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. Training generative adversarial networks with limited data. In *Proc. NeurIPS*, 2020.

[26] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021.

[27] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2018.

[28] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro. DiffWave: A versatile diffusion model for audio synthesis. In *Proc. ICLR*, 2021.

[29] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[30] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. Noise2Noise: Learning image restoration without clean data. In *Proc. ICML*, 2018.

[31] L. Liu, Y. Ren, Z. Lin, and Z. Zhao. Pseudo numerical methods for diffusion models on manifolds. In *Proc. ICLR*, 2022.

[32] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Proc. NeurIPS*, 2022.

[33] E. Luhman and T. Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *CoRR*, abs/2101.02388, 2021.

[34] P. Mishkin, L. Ahmad, M. Brundage, G. Krueger, and G. Sastry. DALL·E 2 preview – risks and limitations. *OpenAI*, 2022.

[35] E. Nachmani and S. Dovrat. Zero-shot translation using diffusion models. *CoRR*, abs/2111.01471, 2021.

[36] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. In *Proc. ICML*, 2022.

[37] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *Proc. ICML*, volume 139, pages 8162–8171, 2021.

[38] V. Popov, I. Vovk, V. Gogoryan, T. Sadekova, and M. Kudinov. Grad-TTS: A diffusion probabilistic model for text-to-speech. In *Proc. ICML*, volume 139, pages 8599–8608, 2021.

[39] K. Preechakul, N. Chatthee, S. Wizadwongsa, and S. Suwajanakorn. Diffusion autoencoders: Toward a meaningful and decodable representation. In *Proc. CVPR*, 2022.

[40] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with CLIP latents. Technical report, OpenAI, 2022.

[41] A. J. Roberts. Modify the improved Euler scheme to integrate stochastic differential equations. *CoRR*, abs/1210.0933, 2012.

[42] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022.

[43] C. Saharia, W. Chan, H. Chang, C. A. Lee, J. Ho, T. Salimans, D. J. Fleet, and M. Norouzi. Palette: Image-to-image diffusion models. In *Proc. SIGGRAPH*, 2022.

[44] T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models. In *Proc. ICLR*, 2022.

[45] A. Sauer, K. Schwarz, and A. Geiger. StyleGAN-XL: Scaling StyleGAN to large diverse datasets. In *Proc. SIGGRAPH*, 2022.

[46] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proc. ICML*, pages 2256–2265, 2015.

[47] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *Proc. ICLR*, 2021.

[48] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. In *Proc. NeurIPS*, 2019.

[49] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *Proc. ICLR*, 2021.

[50] E. Süli and D. F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. In *Proc. CVPR*, 2016.

[52] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proc. NeurIPS*, 2020.

[53] A. Vahdat, K. Kreis, and J. Kautz. Score-based generative modeling in latent space. In *Proc. NeurIPS*, 2021.

[54] P. Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

[55] D. Watson, W. Chan, J. Ho, and M. Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality. In *Proc. ICLR*, 2022.

[56] D. Watson, J. Ho, M. Norouzi, and W. Chan. Learning to efficiently sample from diffusion probabilistic models. *CoRR*, abs/2106.03802, 2021.

[57] J. Wolleb, R. Sandkühler, F. Bieder, P. Valmaggia, and P. C. Cattin. Diffusion models for implicit image segmentation ensembles. In *Medical Imaging with Deep Learning*, 2022.

[58] Q. Zhang and Y. Chen. Diffusion normalizing flow. In *Proc. NeurIPS*, 2021.

[59] Q. Zhang and Y. Chen. Fast sampling of diffusion models with exponential integrator. *CoRR*, abs/2204.13902, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 6. The main limitations of the analysis relate to the set of tested datasets and their limited resolution.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [No] We follow common application-specific assumptions about the probability distributions, functions and other components, but do not exhaustively specify them, or consider pathological corner cases.
   (b) Did you include complete proofs of all theoretical results? [No] Our equations and algorithms build on previously known results, and highlight their practical aspects through mostly readily verifiable algebraic manipulations (Appendix B). We do not explicitly present all details of the derivations, and assume that the previous results are sufficiently rigorously proven in the respective literature.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our implementation and pre-trained models are available at `https://github.com/NVlabs/edm`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix F.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Shaded regions in Figures 4 and 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Section 6.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Appendix F.5.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendices

A Additional results

Figure 6 presents generated images for class-conditional ImageNet-64 [8] using the pre-trained ADM model by Dhariwal and Nichol [9]. The original DDIM [47] and iDDPM [37] samplers are compared to ours in both deterministic and stochastic settings (Sections 3 and 4). Figure 7 shows the corresponding results that we obtain by training the model from scratch using our improved training configuration (Section 5).

The original samplers and training configurations by Song et al. [49] are compared to ours in Figures 8 and 9 (unconditional CIFAR-10 [29]), Figure 10 (class-conditional CIFAR-10), and Figure 11 (FFHQ [27] and AFHQv2 [7]). For ease of comparison, the same latent codes $\boldsymbol{x}_0$ are used for each dataset/scenario across different training configurations and ODE choices. Figure 12 shows generated image quality with various NFE when using deterministic sampling.

Tables 3 and 4 summarize the numerical results on deterministic and stochastic sampling methods in various datasets, previously shown as functions of NFE in Figures 2 and 4.

B Derivation of formulas

B.1 Original ODE / SDE formulation from previous work

Song et al. [49] define their forward SDE (Eq. 5 in [49]) as

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\omega_t, \tag{9}$$

where ω_t is the standard Wiener process and $\boldsymbol{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ and $g(\cdot) : \mathbb{R} \to \mathbb{R}$ are the drift and diffusion coefficients, respectively, where d is the dimensionality of the dataset. These coefficients are selected differently for the variance preserving (VP) and variance exploding (VE) formulations, and $\boldsymbol{f}(\cdot)$ is always of the form $\boldsymbol{f}(\boldsymbol{x}, t) = f(t)\,\boldsymbol{x}$, where $f(\cdot) : \mathbb{R} \to \mathbb{R}$. Thus, the SDE can be equivalently written as

$$\mathrm{d}\boldsymbol{x} = f(t)\,\boldsymbol{x}\,\mathrm{d}t + g(t)\,\mathrm{d}\omega_t. \tag{10}$$

The perturbation kernels of this SDE (Eq. 29 in [49]) have the general form

$$p_{0t}\big(\boldsymbol{x}(t) \mid \boldsymbol{x}(0)\big) = \mathcal{N}\big(\boldsymbol{x}(t);\ s(t)\,\boldsymbol{x}(0),\ s(t)^2\,\sigma(t)^2\,\mathbf{I}\big), \tag{11}$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the probability density function of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ evaluated at $\boldsymbol{x}$,

$$s(t) = \exp\left(\int_0^t f(\xi)\,\mathrm{d}\xi\right), \quad \text{and} \quad \sigma(t) = \sqrt{\int_0^t \frac{g(\xi)^2}{s(\xi)^2}\,\mathrm{d}\xi}. \tag{12}$$

The marginal distribution $p_t(\boldsymbol{x})$ is obtained by integrating the perturbation kernels over $\boldsymbol{x}(0)$:

$$p_t(\boldsymbol{x}) = \int_{\mathbb{R}^d} p_{0t}(\boldsymbol{x} \mid \boldsymbol{x}_0)\,p_{\text{data}}(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0. \tag{13}$$

Song et al. [49] define the probability flow ODE (Eq. 13 in [49]) so that it obeys this same $p_t(\boldsymbol{x})$:

$$\mathrm{d}\boldsymbol{x} = \left[f(t)\,\boldsymbol{x} - \tfrac{1}{2}\,g(t)^2\,\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})\right]\,\mathrm{d}t. \tag{14}$$

B.2 Our ODE formulation (Eq. 1 and Eq. 4)

The original ODE formulation (Eq. 14) is built around the functions f and g that correspond directly to specific terms that appear in the formula; the properties of the marginal distribution (Eq. 12) can only be derived indirectly based on these functions. However, f and g are of little practical interest in themselves, whereas the marginal distributions are of utmost importance in terms of training the model in the first place, bootstrapping the sampling process, and understanding how the ODE behaves in practice. Given that the idea of the probability flow ODE is to match a particular set of marginal

Figure 6: Results for different samplers on class-conditional ImageNet [8] at 64×64 resolution, using the pre-trained model by Dhariwal and Nichol [9]. The cases correspond to dots in Figures 2c and 4c.

Deterministic, Our sampler & training configuration

FID 2.23 NFE **79**

Stochastic, Our sampler & training configuration

FID **1.36** NFE 511

Figure 7: Results for our training configuration on class-conditional ImageNet [8] at 64×64 resolution, using our deterministic and stochastic samplers.

Figure 8: Results for different samplers on unconditional CIFAR-10 [29] at 32×32 resolution, using the pre-trained models by Song et al. [49]. The cases correspond to dots in Figures 2a,b and 4a,b.

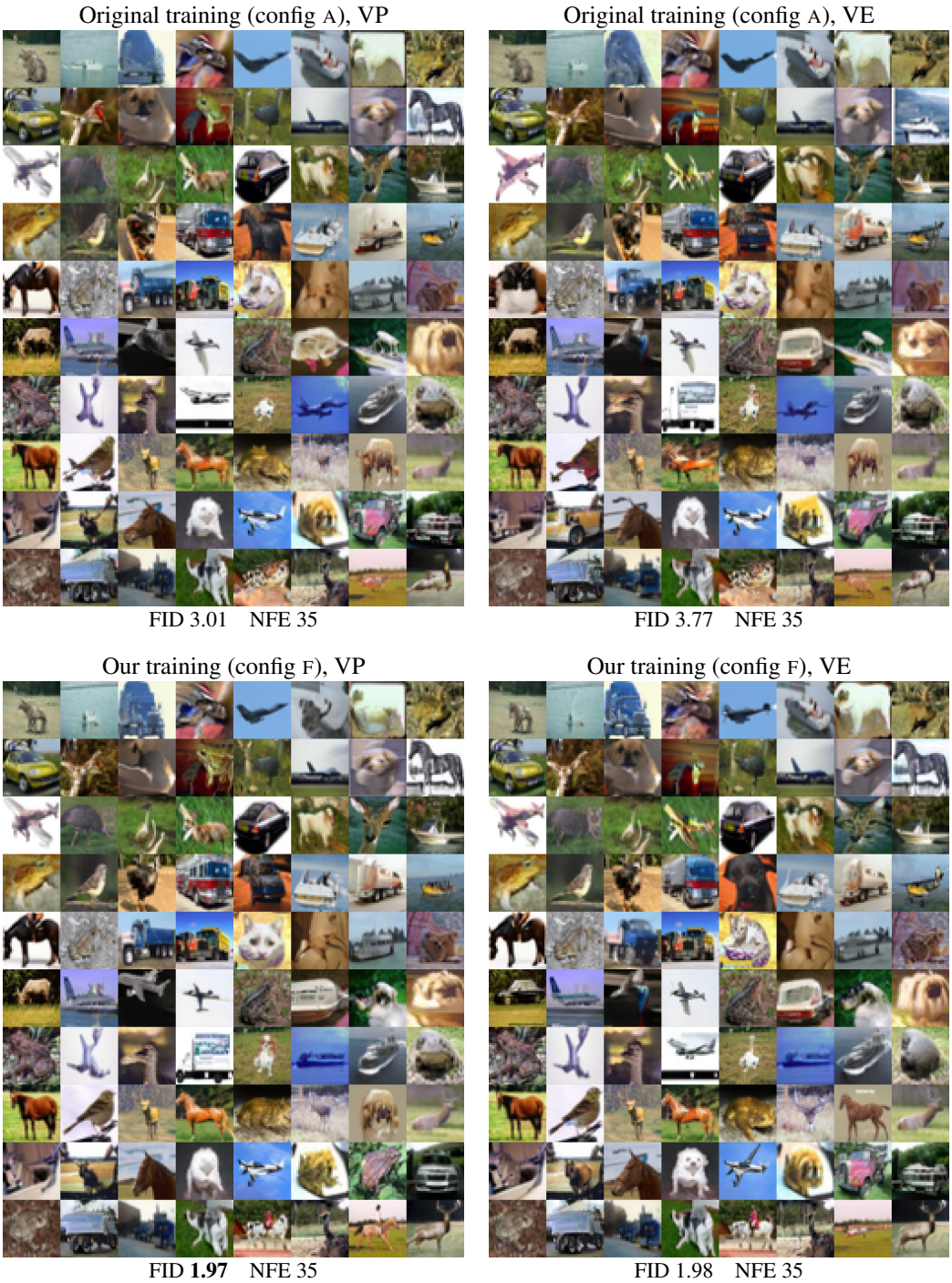


Figure 9: Results for different training configurations on unconditional CIFAR-10 [29] at 32×32 resolution, using our deterministic sampler with the same set of latent codes ($\boldsymbol{x}_0$) in each case.

Figure 10: Results for different training configurations on class-conditional CIFAR-10 [29] at 32×32 resolution, using our deterministic sampler with the same set of latent codes ($\boldsymbol{x}_0$) in each case.

Figure 11: Results for different training configurations on FFHQ [27] and AFHQv2 [7] at 64×64 resolution, using our deterministic sampler with the same set of latent codes ($\boldsymbol{x}_0$) in each case.

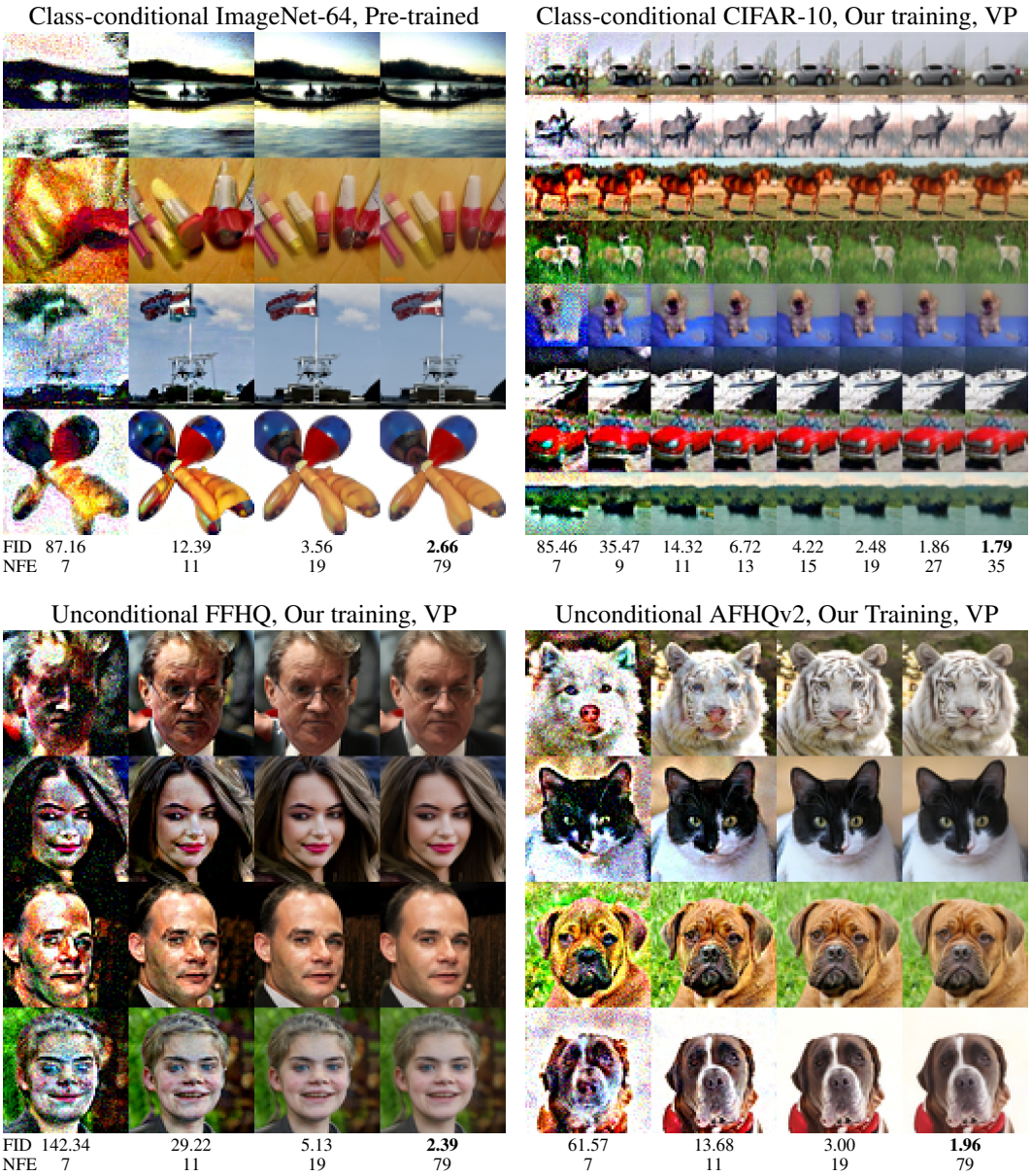


Figure 12: Image quality and FID as a function of NFE using our deterministic sampler. At 32×32 resolution, reasonable image quality is reached around NFE = 13, but FID keeps improving until NFE = 35. At 64×64 resolution, reasonable image quality is reached around NFE = 19, but FID keeps improving until NFE = 79.

Table 3: Evaluation of our improvements to deterministic sampling. The values correspond to the curves shown in Figure 2. We summarize each curve with two key values: the lowest observed FID for any NFE ("FID"), and the lowest NFE whose FID is within 3% of the lowest FID ("NFE"). The values marked with "–" are identical to the ones above them, because our sampler uses the same $\sigma(t)$ and $s(t)$ as DDIM.

	Unconditional CIFAR-10 at 32×32				Class-conditional ImageNet-64	
	VP		VE			
Sampling method	FID ↓	NFE ↓	FID ↓	NFE ↓	FID ↓	NFE ↓
Original sampler [49, 9]	2.85	256	5.45	8192	2.85	250
Our Algorithm 1	**2.79**	512	4.78	8192	2.73	384
+ Heun & our t_i	2.88	255	4.23	191	**2.64**	**79**
+ Our $\sigma(t)$ & $s(t)$	2.93	**35**	3.73	**27**	–	–
Black-box RK45	2.94	115	**3.69**	93	2.66	131

Table 4: Evaluation and ablations of our improvements to stochastic sampling. The values correspond to the curves shown in Figure 4.

	Unconditional CIFAR-10 at 32×32				Class-conditional ImageNet-64	
	VP		VE			
Sampling method	FID ↓	NFE ↓	FID ↓	NFE ↓	FID ↓	NFE ↓
Deterministic baseline (Alg. 1)	2.93	**35**	3.73	**27**	2.64	**79**
Alg. 2, $S_{\text{tmin,tmax}} = [0, \infty]$, $S_{\text{noise}} = 1$	2.69	95	2.97	383	1.86	383
Alg. 2, $S_{\text{tmin,tmax}} = [0, \infty]$	2.54	127	2.51	511	1.63	767
Alg. 2, $S_{\text{noise}} = 1$	2.52	95	2.84	191	1.84	255
Alg. 2, Optimal settings	**2.27**	383	**2.23**	767	**1.55**	511
Previous work [49, 9]	2.55	768	2.46	1024	2.01	384

distributions, it makes sense to treat the marginal distributions as first-class citizens and define the ODE directly based on $\sigma(t)$ and $s(t)$, eliminating the need for $f(t)$ and $g(t)$.

Let us start by expressing the marginal distribution of Eq. 13 in closed form:

$$
\begin{aligned}
p_t(\boldsymbol{x}) &= \int_{\mathbb{R}^d} p_{0t}(\boldsymbol{x} \mid \boldsymbol{x}_0)\, p_{\text{data}}(\boldsymbol{x}_0)\, \mathrm{d}\boldsymbol{x}_0 && (15)\\
&= \int_{\mathbb{R}^d} p_{\text{data}}(\boldsymbol{x}_0) \left[\mathcal{N}\big(\boldsymbol{x};\ s(t)\,\boldsymbol{x}_0,\ s(t)^2\,\sigma(t)^2\,\mathbf{I}\big)\right] \mathrm{d}\boldsymbol{x}_0 && (16)\\
&= \int_{\mathbb{R}^d} p_{\text{data}}(\boldsymbol{x}_0) \left[s(t)^{-d}\,\mathcal{N}\big(\boldsymbol{x}/s(t);\ \boldsymbol{x}_0,\ \sigma(t)^2\,\mathbf{I}\big)\right] \mathrm{d}\boldsymbol{x}_0 && (17)\\
&= s(t)^{-d} \int_{\mathbb{R}^d} p_{\text{data}}(\boldsymbol{x}_0)\, \mathcal{N}\big(\boldsymbol{x}/s(t);\ \boldsymbol{x}_0,\ \sigma(t)^2\,\mathbf{I}\big)\, \mathrm{d}\boldsymbol{x}_0 && (18)\\
&= s(t)^{-d} \left[p_{\text{data}} * \mathcal{N}\big(\mathbf{0},\ \sigma(t)^2\,\mathbf{I}\big)\right]\big(\boldsymbol{x}/s(t)\big), && (19)
\end{aligned}
$$

where $p_a * p_b$ denotes the convolution of probability density functions p_a and p_b. The expression inside the brackets corresponds to a mollified version of p_{data} obtained by adding i.i.d. Gaussian noise to the samples. Let us denote this distribution by $p(\boldsymbol{x};\sigma)$:

$$
p(\boldsymbol{x};\sigma) = p_{\text{data}} * \mathcal{N}\big(\mathbf{0},\ \sigma(t)^2\,\mathbf{I}\big) \quad \text{and} \quad p_t(\boldsymbol{x}) = s(t)^{-d}\, p\big(\boldsymbol{x}/s(t);\sigma(t)\big). \tag{20}
$$

We can now express the probability flow ODE (Eq. 14) using $p(\boldsymbol{x};\sigma)$ instead of $p_t(\boldsymbol{x})$:

$$
\begin{aligned}
\mathrm{d}\boldsymbol{x} &= \big[f(t)\boldsymbol{x} - \tfrac{1}{2}\, g(t)^2\, \nabla_{\boldsymbol{x}} \log\big[p_t(\boldsymbol{x})\big]\big]\ \mathrm{d}t && (21)\\
&= \big[f(t)\boldsymbol{x} - \tfrac{1}{2}\, g(t)^2\, \nabla_{\boldsymbol{x}} \log\big[s(t)^{-d}\, p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\big]\big]\ \mathrm{d}t && (22)\\
&= \big[f(t)\boldsymbol{x} - \tfrac{1}{2}\, g(t)^2\, \big[\nabla_{\boldsymbol{x}} \log s(t)^{-d} + \nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\big]\big]\ \mathrm{d}t && (23)\\
&= \big[f(t)\boldsymbol{x} - \tfrac{1}{2}\, g(t)^2\, \nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\big]\ \mathrm{d}t. && (24)
\end{aligned}
$$

Next, let us rewrite $f(t)$ in terms of $s(t)$ based on Eq. 12:

$$\exp\left(\int_0^t f(\xi)\,\mathrm{d}\xi\right) = s(t) \tag{25}$$

$$\int_0^t f(\xi)\,\mathrm{d}\xi = \log s(t) \tag{26}$$

$$\mathrm{d}\left[\int_0^t f(\xi)\,\mathrm{d}\xi\right]/\mathrm{d}t = \mathrm{d}\big[\log s(t)\big]/\mathrm{d}t \tag{27}$$

$$f(t) = \dot{s}(t)/s(t). \tag{28}$$

Similarly, we can also rewrite $g(t)$ in terms of $\sigma(t)$:

$$\sqrt{\int_0^t \frac{g(\xi)^2}{s(\xi)^2}\,\mathrm{d}\xi} = \sigma(t) \tag{29}$$

$$\int_0^t \frac{g(\xi)^2}{s(\xi)^2}\,\mathrm{d}\xi = \sigma(t)^2 \tag{30}$$

$$\mathrm{d}\left[\int_0^t \frac{g(\xi)^2}{s(\xi)^2}\,\mathrm{d}\xi\right]/\mathrm{d}t = \mathrm{d}\big[\sigma(t)^2\big]/\mathrm{d}t \tag{31}$$

$$g(t)^2/s(t)^2 = 2\,\dot{\sigma}(t)\,\sigma(t) \tag{32}$$

$$g(t)/s(t) = \sqrt{2\,\dot{\sigma}(t)\,\sigma(t)} \tag{33}$$

$$g(t) = s(t)\,\sqrt{2\,\dot{\sigma}(t)\,\sigma(t)}. \tag{34}$$

Finally, substitute f (Eq. 28) and g (Eq. 34) into the ODE of Eq. 24:

$$\mathrm{d}\boldsymbol{x} = \big[[f(t)]\,\boldsymbol{x} - \tfrac{1}{2}\,[g(t)]^2\,\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\big]\;\mathrm{d}t \tag{35}$$

$$= \left[\big[\dot{s}(t)/s(t)\big]\,\boldsymbol{x} - \tfrac{1}{2}\left[s(t)\sqrt{2\,\dot{\sigma}(t)\,\sigma(t)}\right]^2\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\right]\mathrm{d}t \tag{36}$$

$$= \left[\big[\dot{s}(t)/s(t)\big]\,\boldsymbol{x} - \tfrac{1}{2}\left[2\,s(t)^2\,\dot{\sigma}(t)\,\sigma(t)\right]\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\right]\mathrm{d}t \tag{37}$$

$$= \left[\frac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - s(t)^2\,\dot{\sigma}(t)\,\sigma(t)\,\nabla_{\boldsymbol{x}}\log p\left(\frac{\boldsymbol{x}}{s(t)};\sigma(t)\right)\right]\;\mathrm{d}t. \tag{38}$$

Thus we have obtained Eq. 4 in the main paper, and Eq. 1 is recovered by setting $s(t) = 1$:

$$\mathrm{d}\boldsymbol{x} = -\dot{\sigma}(t)\,\sigma(t)\,\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x};\sigma(t)\big)\;\mathrm{d}t. \tag{39}$$

Our formulation (Eq. 4) highlights the fact that every realization of the probability flow ODE is simply a reparameterization of the same canonical ODE; changing $\sigma(t)$ corresponds to reparameterizing t, whereas changing $s(t)$ corresponds to reparameterizing $\boldsymbol{x}$.

B.3 Denoising score matching (Eq. 2 and Eq. 3)

For the sake of completeness, we derive the connection between score matching and denoising for a finite dataset. For a more general treatment and further background on the topic, see Hyvärinen [22] and Vincent [54].

Let us assume that our training set consists of a finite number of samples $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_Y\}$. This implies $p_{\text{data}}(\boldsymbol{x})$ is represented by a mixture of Dirac delta distributions:

$$p_{\text{data}}(\boldsymbol{x}) = \frac{1}{Y}\sum_{i=1}^{Y}\delta\big(\boldsymbol{x} - \boldsymbol{y}_i\big), \tag{40}$$

which allows us to also express $p(\boldsymbol{x};\sigma)$ in closed form based on Eq. 20:

$$
\begin{aligned}
p(\boldsymbol{x};\sigma) &= p_{\text{data}} * \mathcal{N}\big(\mathbf{0},\ \sigma(t)^2\,\mathbf{I}\big) && (41)\\
&= \int_{\mathbb{R}^d} p_{\text{data}}(\boldsymbol{x}_0)\,\mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{x}_0,\ \sigma^2\,\mathbf{I}\big)\,\mathrm{d}\boldsymbol{x}_0 && (42)\\
&= \int_{\mathbb{R}^d} \left[\frac{1}{Y}\sum_{i=1}^{Y}\delta\big(\boldsymbol{x}_0-\boldsymbol{y}_i\big)\right]\mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{x}_0,\ \sigma^2\,\mathbf{I}\big)\,\mathrm{d}\boldsymbol{x}_0 && (43)\\
&= \frac{1}{Y}\sum_{i=1}^{Y}\int_{\mathbb{R}^d}\mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{x}_0,\ \sigma^2\,\mathbf{I}\big)\,\delta\big(\boldsymbol{x}_0-\boldsymbol{y}_i\big)\,\mathrm{d}\boldsymbol{x}_0 && (44)\\
&= \frac{1}{Y}\sum_{i=1}^{Y}\mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big). && (45)
\end{aligned}
$$

Let us now consider the denoising score matching loss of Eq. 2. By expanding the expectations, we can rewrite the formula as an integral over the noisy samples $\boldsymbol{x}$:

$$
\begin{aligned}
\mathcal{L}(D;\sigma) &= \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\,\mathbb{E}_{\boldsymbol{n}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})}\,\big\|D(\boldsymbol{y}+\boldsymbol{n};\sigma)-\boldsymbol{y}\big\|_2^2 && (46)\\
&= \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\,\mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{y},\sigma^2\mathbf{I})}\,\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}\big\|_2^2 && (47)\\
&= \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\int_{\mathbb{R}^d}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y},\ \sigma^2\,\mathbf{I})\,\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}\big\|_2^2\,\mathrm{d}\boldsymbol{x} && (48)\\
&= \frac{1}{Y}\sum_{i=1}^{Y}\int_{\mathbb{R}^d}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}_i\big\|_2^2\,\mathrm{d}\boldsymbol{x} && (49)\\
&= \int_{\mathbb{R}^d}\underbrace{\frac{1}{Y}\sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}_i\big\|_2^2}_{=:\ \mathcal{L}(D;\boldsymbol{x},\sigma)}\,\mathrm{d}\boldsymbol{x}. && (50)
\end{aligned}
$$

Eq. 50 means that we can minimize $\mathcal{L}(D;\sigma)$ by minimizing $\mathcal{L}(D;\boldsymbol{x},\sigma)$ independently for each $\boldsymbol{x}$:

$$
D(\boldsymbol{x};\sigma) = \arg\min\nolimits_{D(\boldsymbol{x};\sigma)}\,\mathcal{L}(D;\boldsymbol{x},\sigma). \qquad (51)
$$

This is a convex optimization problem; its solution is uniquely identified by setting the gradient w.r.t. $D(\boldsymbol{x};\sigma)$ to zero:

$$
\begin{aligned}
\mathbf{0} &= \nabla_{D(\boldsymbol{x};\sigma)}\Big[\mathcal{L}(D;\boldsymbol{x},\sigma)\Big] && (52)\\
\mathbf{0} &= \nabla_{D(\boldsymbol{x};\sigma)}\left[\frac{1}{Y}\sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}_i\big\|_2^2\right] && (53)\\
\mathbf{0} &= \sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\nabla_{D(\boldsymbol{x};\sigma)}\Big[\big\|D(\boldsymbol{x};\sigma)-\boldsymbol{y}_i\big\|_2^2\Big] && (54)\\
\mathbf{0} &= \sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\Big[2\,D(\boldsymbol{x};\sigma)-2\,\boldsymbol{y}_i\Big] && (55)\\
\mathbf{0} &= \left[\sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\right]D(\boldsymbol{x};\sigma)-\sum_{i=1}^{Y}\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\boldsymbol{y}_i && (56)\\
D(\boldsymbol{x};\sigma) &= \frac{\sum_i\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})\,\boldsymbol{y}_i}{\sum_i\mathcal{N}(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I})}, && (57)
\end{aligned}
$$

which gives a closed-form solution for the ideal denoiser $D(\boldsymbol{x};\sigma)$. Note that Eq. 57 is feasible to compute in practice for small datasets — we show the results for CIFAR-10 in Figure 1b.

Next, let us consider the score of the distribution $p(\boldsymbol{x};\sigma)$ defined in Eq. 45:

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma) = \frac{\nabla_{\boldsymbol{x}} p(\boldsymbol{x};\sigma)}{p(\boldsymbol{x};\sigma)} \tag{58}$$

$$= \frac{\nabla_{\boldsymbol{x}} \Big[\frac{1}{Y}\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\Big]}{\Big[\frac{1}{Y}\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\Big]} \tag{59}$$

$$= \frac{\sum_i \nabla_{\boldsymbol{x}} \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)}{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)}. \tag{60}$$

We can simplify the numerator of Eq. 60 further:

$$\nabla_{\boldsymbol{x}} \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big) = \nabla_{\boldsymbol{x}}\left[\big(2\pi\sigma^2\big)^{-\frac{d}{2}}\ \exp\frac{\|\boldsymbol{x}-\boldsymbol{y}_i\|_2^2}{-2\,\sigma^2}\right] \tag{61}$$

$$= \big(2\pi\sigma^2\big)^{-\frac{d}{2}}\ \nabla_{\boldsymbol{x}}\left[\exp\frac{\|\boldsymbol{x}-\boldsymbol{y}_i\|_2^2}{-2\,\sigma^2}\right] \tag{62}$$

$$= \left[\big(2\pi\sigma^2\big)^{-\frac{d}{2}}\exp\frac{\|\boldsymbol{x}-\boldsymbol{y}_i\|_2^2}{-2\,\sigma^2}\right]\nabla_{\boldsymbol{x}}\left[\frac{\|\boldsymbol{x}-\boldsymbol{y}_i\|_2^2}{-2\,\sigma^2}\right] \tag{63}$$

$$= \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\ \nabla_{\boldsymbol{x}}\left[\frac{\|\boldsymbol{x}-\boldsymbol{y}_i\|_2^2}{-2\,\sigma^2}\right] \tag{64}$$

$$= \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\left[\frac{\boldsymbol{y}_i-\boldsymbol{x}}{\sigma^2}\right]. \tag{65}$$

Let us substitute the result back to Eq. 60:

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma) = \frac{\sum_i \nabla_{\boldsymbol{x}} \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)}{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)} \tag{66}$$

$$= \frac{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\left[\frac{\boldsymbol{y}_i-\boldsymbol{x}}{\sigma^2}\right]}{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)} \tag{67}$$

$$= \left(\frac{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)\boldsymbol{y}_i}{\sum_i \mathcal{N}\big(\boldsymbol{x};\ \boldsymbol{y}_i,\ \sigma^2\,\mathbf{I}\big)} - \boldsymbol{x}\right)/\sigma^2. \tag{68}$$

Notice that the fraction in Eq. 68 is identical to Eq. 57. We can thus equivalently write Eq. 68 as

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma) = \big(D(\boldsymbol{x};\ \sigma) - \boldsymbol{x}\big)/\sigma^2, \tag{69}$$

which matches Eq. 3 in the main paper.

B.4 Evaluating our ODE in practice (Algorithm 1)

Let us consider $\boldsymbol{x}$ to be a scaled version of an original, non-scaled variable $\hat{\boldsymbol{x}}$ and substitute $\boldsymbol{x} = s(t)\,\hat{\boldsymbol{x}}$ into the score term that appears in our scaled ODE (Eq. 4):

$$\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}/s(t);\sigma(t)\big) \tag{70}$$

$$= \nabla_{[s(t)\hat{\boldsymbol{x}}]} \log p\big([s(t)\ \hat{\boldsymbol{x}}]/s(t);\sigma(t)\big) \tag{71}$$

$$= \nabla_{s(t)\hat{\boldsymbol{x}}} \log p\big(\hat{\boldsymbol{x}};\sigma(t)\big) \tag{72}$$

$$= \tfrac{1}{s(t)}\nabla_{\hat{\boldsymbol{x}}} \log p\big(\hat{\boldsymbol{x}};\sigma(t)\big). \tag{73}$$

We can further rewrite this with respect to $D(\cdot)$ using Eq. 3:

$$\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}/s(t);\sigma(t)\big) = \tfrac{1}{s(t)\sigma(t)^2}\Big(D\big(\hat{\boldsymbol{x}};\sigma(t)\big) - \hat{\boldsymbol{x}}\Big). \tag{74}$$

Let us now substitute Eq. 74 into Eq. 4, approximating the ideal denoiser $D(\cdot)$ with our trained model $D_\theta(\cdot)$:

$$\begin{aligned} \mathrm{d}\boldsymbol{x} &= \left[\dot{s}(t)\,\boldsymbol{x}/s(t) - s(t)^2\,\dot{\sigma}(t)\,\sigma(t)\,\left[\tfrac{1}{s(t)\sigma(t)^2}\Big(D_\theta\big(\hat{\boldsymbol{x}};\sigma(t)\big) - \hat{\boldsymbol{x}}\Big)\right]\right]\,\mathrm{d}t && (75)\\ &= \left[\tfrac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - \tfrac{\dot{\sigma}(t)s(t)}{\sigma(t)}\Big(D_\theta\big(\hat{\boldsymbol{x}};\sigma(t)\big) - \hat{\boldsymbol{x}}\Big)\right]\,\mathrm{d}t. && (76)\end{aligned}$$

Finally, backsubstitute $\hat{\boldsymbol{x}} = \boldsymbol{x}/s(t)$:

$$\begin{aligned} \mathrm{d}\boldsymbol{x} &= \left[\tfrac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - \tfrac{\dot{\sigma}(t)s(t)}{\sigma(t)}\Big(D_\theta\big([\hat{\boldsymbol{x}}];\sigma(t)\big) - [\hat{\boldsymbol{x}}]\Big)\right]\,\mathrm{d}t && (77)\\ &= \left[\tfrac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - \tfrac{\dot{\sigma}(t)s(t)}{\sigma(t)}\Big(D_\theta\big([\boldsymbol{x}/s(t)];\sigma(t)\big) - [\boldsymbol{x}/s(t)]\Big)\right]\,\mathrm{d}t && (78)\\ &= \left[\tfrac{\dot{s}(t)}{s(t)}\,\boldsymbol{x} - \tfrac{\dot{\sigma}(t)s(t)}{\sigma(t)}D_\theta\big(\boldsymbol{x}/s(t);\sigma(t)\big) + \tfrac{\dot{\sigma}(t)}{\sigma(t)}\,\boldsymbol{x}\right]\,\mathrm{d}t && (79)\\ &= \left[\Big(\tfrac{\dot{\sigma}(t)}{\sigma(t)} + \tfrac{\dot{s}(t)}{s(t)}\Big)\,\boldsymbol{x} - \tfrac{\dot{\sigma}(t)s(t)}{\sigma(t)}D_\theta\big(\boldsymbol{x}/s(t);\sigma(t)\big)\right]\,\mathrm{d}t. && (80)\end{aligned}$$

We can equivalenty write Eq. 80 as

$$\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \left(\frac{\dot{\sigma}(t)}{\sigma(t)} + \frac{\dot{s}(t)}{s(t)}\right)\boldsymbol{x} - \frac{\dot{\sigma}(t)s(t)}{\sigma(t)}D_\theta\left(\frac{\boldsymbol{x}}{s(t)};\sigma(t)\right), \tag{81}$$

matching lines 4 and 7 of Algorithm 1.

B.5 Our SDE formulation (Eq. 6)

We derive the SDE of Eq. 6 by the following strategy:

- The desired marginal densities $p\big(\boldsymbol{x};\sigma(t)\big)$ are convolutions of the data density p_{data} and an isotropic Gaussian density with standard deviation $\sigma(t)$ (see Eq. 20). Hence, considered as a function of the time t, the density evolves according to a heat diffusion PDE with time-varying diffusivity. As a first step, we find this PDE.
- We then use the Fokker–Planck equation to recover a family of SDEs for which the density evolves according to this PDE. Eq. 6 is obtained from a suitable parametrization of this family.

B.5.1 Generating the marginals by heat diffusion

We consider the time evolution of a probability density $q(\boldsymbol{x},t)$. Our goal is to find a PDE whose solution with the initial value $q(\boldsymbol{x},0) := p_{\text{data}}(\boldsymbol{x})$ is $q(\boldsymbol{x},t) = p\big(\boldsymbol{x},\sigma(t)\big)$. That is, the PDE should reproduce the marginals we postulate in Eq. 20.

The desired marginals are convolutions of p_{data} with isotropic normal distributions of time-varying standard deviation $\sigma(t)$, and as such, can be generated by the heat equation with time-varying diffusivity $\kappa(t)$. The situation is most conveniently analyzed in the Fourier domain, where the marginal densities are simply pointwise products of a Gaussian function and the transformed data density. To find the diffusivity that induces the correct standard deviations, we first write down the heat equation PDE:

$$\frac{\partial q(\boldsymbol{x},t)}{\partial t} = \kappa(t)\Delta_{\boldsymbol{x}} q(\boldsymbol{x},t). \tag{82}$$

The Fourier transformed counterpart of Eq. 82, where the transform is taken along the $\boldsymbol{x}$-dimension, is given by

$$\frac{\partial \hat{q}(\boldsymbol{\nu},t)}{\partial t} = -\kappa(t)|\boldsymbol{\nu}|^2\hat{q}(\boldsymbol{\nu},t). \tag{83}$$

The target solution $q(\boldsymbol{x},t)$ and its Fourier transform $\hat{q}(\boldsymbol{\nu},t)$ are given by Eq. 20:

$$\begin{aligned} q(\boldsymbol{x},t) &= p\big(\boldsymbol{x};\sigma(t)\big) = p_{\text{data}}(\boldsymbol{x}) * \mathcal{N}\big(\mathbf{0},\ \sigma(t)^2\,\mathbf{I}\big) && (84)\\ \hat{q}(\boldsymbol{\nu},t) &= \hat{p}_{\text{data}}(\boldsymbol{\nu})\ \exp\Big(-\tfrac{1}{2}\,|\boldsymbol{\nu}|^2\,\sigma(t)^2\Big). && (85)\end{aligned}$$

Differentiating the target solution along the time axis, we have

$$\frac{\partial \hat{q}(\boldsymbol{\nu}, t)}{\partial t} = -\dot{\sigma}(t)\sigma(t)\, |\boldsymbol{\nu}|^2\, \hat{p}_{\text{data}}(\boldsymbol{\nu})\, \exp\left(-\tfrac{1}{2}\, |\boldsymbol{\nu}|^2\, \sigma(t)^2\right) \tag{86}$$

$$= -\dot{\sigma}(t)\sigma(t)\, |\boldsymbol{\nu}|^2\, \hat{q}(\boldsymbol{\nu}, t). \tag{87}$$

Eqs. 83 and 87 share the same left hand side. Equating them allows us to solve for $\kappa(t)$ that generates the desired evolution:

$$-\kappa(t)|\boldsymbol{\nu}|^2 \hat{q}(\boldsymbol{\nu}, t) = -\dot{\sigma}(t)\sigma(t)\, |\boldsymbol{\nu}|^2\, \hat{q}(\boldsymbol{\nu}, t) \tag{88}$$

$$\kappa(t) = \dot{\sigma}(t)\sigma(t). \tag{89}$$

To summarize, the desired marginal densities corresponding to noise levels $\sigma(t)$ are generated by the PDE

$$\frac{\partial q(\boldsymbol{x}, t)}{\partial t} = \dot{\sigma}(t)\sigma(t)\Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) \tag{90}$$

from the initial density $q(\boldsymbol{x}, 0) = p_{\text{data}}(\boldsymbol{x})$.

B.5.2 Derivation of our SDE

Given an SDE

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)\, \mathrm{d}t \;+\; \boldsymbol{g}(\boldsymbol{x}, t)\, \mathrm{d}\omega_t, \tag{91}$$

the Fokker–Planck PDE describes the time evolution of its solution probability density $r(\boldsymbol{x}, t)$ as

$$\frac{\partial r(\boldsymbol{x}, t)}{\partial t} = -\nabla_{\boldsymbol{x}} \cdot \big(\boldsymbol{f}(\boldsymbol{x}, t)\, r(\boldsymbol{x}, t)\big) + \tfrac{1}{2}\nabla_{\boldsymbol{x}}\nabla_{\boldsymbol{x}} : \big(\mathbf{D}(\boldsymbol{x}, t)\, r(\boldsymbol{x}, t)\big), \tag{92}$$

where $\mathbf{D}_{ij} = \sum_k \boldsymbol{g}_{ik}\boldsymbol{g}_{jk}$ is the *diffusion tensor*. We consider the special case $\boldsymbol{g}(\boldsymbol{x}, t) = g(t)\, \mathbf{I}$ of $\boldsymbol{x}$-independent white noise addition, whereby the equation simplifies to

$$\frac{\partial r(\boldsymbol{x}, t)}{\partial t} = -\nabla_{\boldsymbol{x}} \cdot \big(\boldsymbol{f}(\boldsymbol{x}, t)\, r(\boldsymbol{x}, t)\big) + \tfrac{1}{2}\, g(t)^2\, \Delta_{\boldsymbol{x}} r(\boldsymbol{x}, t). \tag{93}$$

We are seeking an SDE whose solution density is described by the PDE in Eq. 90. Setting $r(\boldsymbol{x}, t) = q(\boldsymbol{x}, t)$ and equating Eqs. 93 and 90, we find the sufficient condition that the SDE must satisfy

$$-\nabla_{\boldsymbol{x}} \cdot \big(\boldsymbol{f}(\boldsymbol{x}, t)\, q(\boldsymbol{x}, t)\big) + \tfrac{1}{2}\, g(t)^2\, \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) = \dot{\sigma}(t)\, \sigma(t)\, \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) \tag{94}$$

$$\nabla_{\boldsymbol{x}} \cdot \big(\boldsymbol{f}(\boldsymbol{x}, t)\, q(\boldsymbol{x}, t)\big) = \left(\tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t)\right) \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t). \tag{95}$$

Any choice of functions $\boldsymbol{f}(\boldsymbol{x}, t)$ and $g(t)$ satisfying this equation constitute a sought after SDE. Let us now find a specific family of such solutions. The key idea is given by the identity $\nabla_{\boldsymbol{x}} \cdot \nabla_{\boldsymbol{x}} = \Delta_{\boldsymbol{x}}$. Indeed, if we set $\boldsymbol{f}(\boldsymbol{x}, t)\, q(\boldsymbol{x}, t) = \upsilon(t)\, \nabla_{\boldsymbol{x}} q(\boldsymbol{x}, t)$ for any choice of $\upsilon(t)$, the term $\Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t)$ appears on both sides and cancels out:

$$\nabla_{\boldsymbol{x}} \cdot \big(\upsilon(t)\, \nabla_{\boldsymbol{x}} q(\boldsymbol{x}, t)\big) = \left(\tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t)\right) \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) \tag{96}$$

$$\upsilon(t)\, \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) = \left(\tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t)\right) \Delta_{\boldsymbol{x}} q(\boldsymbol{x}, t) \tag{97}$$

$$\upsilon(t) = \tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t). \tag{98}$$

The stated $\boldsymbol{f}(\boldsymbol{x}, t)$ is in fact proportional to the score function, as the formula matches the gradient of the logarithm of the density:

$$\boldsymbol{f}(\boldsymbol{x}, t) = \upsilon(t)\, \frac{\nabla_{\boldsymbol{x}} q(\boldsymbol{x}, t)}{q(\boldsymbol{x}, t)} \tag{99}$$

$$= \upsilon(t)\, \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x}, t) \tag{100}$$

$$= \left(\tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t)\right) \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x}, t). \tag{101}$$

Substituting this back into Eq. 91 and writing $p(\boldsymbol{x};\sigma(t))$ in place of $q(\boldsymbol{x},t)$, we recover a family of SDEs whose solution densities have the desired marginals with noise levels $\sigma(t)$ for any choice of $g(t)$:

$$\mathrm{d}\boldsymbol{x} = \Big(\tfrac{1}{2}\, g(t)^2 - \dot{\sigma}(t)\, \sigma(t)\Big)\, \nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x};\sigma(t)\big)\, \mathrm{d}t \;+\; g(t)\, \mathrm{d}\omega_t. \tag{102}$$

The free parameter $g(t)$ effectively specifies the rate of noise replacement at any given time instance. The special case choice of $g(t) = 0$ corresponds to the probability flow ODE. The parametrization by $g(t)$ is not particularly intuitive, however. To obtain a more interpretable parametrization, we set $g(t) = \sqrt{2\,\beta(t)}\,\sigma(t)$, which yields the (forward) SDE of Eq. 6 in the main paper:

$$\mathrm{d}\boldsymbol{x}_+ = -\dot{\sigma}(t)\sigma(t)\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x};\sigma(t)\big)\, \mathrm{d}t \;+\; \beta(t)\sigma(t)^2\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x};\sigma(t)\big)\, \mathrm{d}t + \sqrt{2\beta(t)}\sigma(t)\, \mathrm{d}\omega_t. \tag{103}$$

The noise replacement is now proportional to the standard deviation $\sigma(t)$ of the noise, with the proportionality factor $\beta(t)$. Indeed, expanding the score function in the middle term according to Eq. 3 yields $\beta(t)\,\big[D\big(\boldsymbol{x};\sigma(t)\big) - \boldsymbol{x}\big]\,\mathrm{d}t$, which changes $\boldsymbol{x}$ proportionally to the negative noise component; the stochastic term injects new noise at the same rate. Intuitively, scaling the magnitude of Langevin exploration according to the current noise standard deviation is a reasonable baseline, as the data manifold is effectively "spread out" by this amount due to the blurring of the density.

The *reverse* SDE used in denoising diffusion is simply obtained by applying the time reversal formula of Anderson [1] (as stated in Eq. 6 of Song et al. [49]) on Eq. 103; the entire effect of the reversal is a change of sign in the middle term.

The scaled generalization of the SDE can be derived using a similar approach as with the ODE previously. As such, the derivation is omitted here.

B.6 Our preconditioning and training (Eq. 8)

Following Eq. 2, the denoising score matching loss for a given denoiser D_θ on a given noise level σ is given by

$$\mathcal{L}(D_\theta;\sigma) = \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\, \mathbb{E}_{\boldsymbol{n}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\mathbf{I})}\, \big\|D_\theta(\boldsymbol{y}+\boldsymbol{n};\sigma) - \boldsymbol{y}\big\|_2^2. \tag{104}$$

We obtain overall training loss by taking a weighted expectation of $\mathcal{L}(D_\theta;\sigma)$ over the noise levels:

$$\begin{aligned}
\mathcal{L}(D_\theta) &= \mathbb{E}_{\sigma\sim p_{\text{train}}}\big[\lambda(\sigma)\, \mathcal{L}(D_\theta;\sigma)\big] && (105)\\
&= \mathbb{E}_{\sigma\sim p_{\text{train}}}\Big[\lambda(\sigma)\, \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\, \mathbb{E}_{\boldsymbol{n}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\mathbf{I})}\, \big\|D_\theta(\boldsymbol{y}+\boldsymbol{n};\sigma) - \boldsymbol{y}\big\|_2^2\Big] && (106)\\
&= \mathbb{E}_{\sigma\sim p_{\text{train}}}\, \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\, \mathbb{E}_{\boldsymbol{n}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\mathbf{I})}\Big[\lambda(\sigma)\, \big\|D_\theta(\boldsymbol{y}+\boldsymbol{n};\sigma) - \boldsymbol{y}\big\|_2^2\Big] && (107)\\
&= \mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\lambda(\sigma)\, \big\|D_\theta(\boldsymbol{y}+\boldsymbol{n};\sigma) - \boldsymbol{y}\big\|_2^2\Big], && (108)
\end{aligned}$$

where the noise levels are distributed according to $\sigma \sim p_{\text{train}}$ and weighted by $\lambda(\sigma)$.

Using our definition of $D_\theta(\cdot)$ from Eq. 7, we can further rewrite $\mathcal{L}(D_\theta)$ as

$$\begin{aligned}
& \mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\lambda(\sigma)\big\|c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n}) + c_{\text{out}}(\sigma)F_\theta\big(c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n}); c_{\text{noise}}(\sigma)\big) - \boldsymbol{y}\big\|_2^2\Big] && (109)\\
=\;& \mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\lambda(\sigma)\big\|c_{\text{out}}(\sigma)F_\theta\big(c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n}); c_{\text{noise}}(\sigma)\big) - \big(\boldsymbol{y} - c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\big)\big\|_2^2\Big] && (110)\\
=\;& \mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\lambda(\sigma)c_{\text{out}}(\sigma)^2\big\|F_\theta\big(c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n}); c_{\text{noise}}(\sigma)\big) - \tfrac{1}{c_{\text{out}}(\sigma)}\big(\boldsymbol{y} - c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\big)\big\|_2^2\Big] && (111)\\
=\;& \mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[w(\sigma)\, \big\|F_\theta\big(c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n}); c_{\text{noise}}(\sigma)\big) - F_{\text{target}}(\boldsymbol{y},\boldsymbol{n};\sigma)\big\|_2^2\Big], && (112)
\end{aligned}$$

which matches Eq. 8 and corresponds to traditional supervised training of F_θ using standard L_2 loss with effective weight $w(\cdot)$ and target $F_{\text{target}}(\cdot)$ given by

$$w(\sigma) = \lambda(\sigma)\, c_{\text{out}}(\sigma)^2 \quad \text{and} \quad F_{\text{target}}(\boldsymbol{y},\boldsymbol{n};\sigma) = \tfrac{1}{c_{\text{out}}(\sigma)}\big(\boldsymbol{y} - c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\big), \tag{113}$$

We can now derive formulas for $c_{\text{in}}(\sigma)$, $c_{\text{out}}(\sigma)$, $c_{\text{skip}}(\sigma)$, and $\lambda(\sigma)$ from first principles, shown in the "Ours" column of Table 1.

First, we require the training inputs of $F_\theta(\cdot)$ to have unit variance:

$$\begin{aligned}
\mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\right] &= 1 && (114)\\
c_{\text{in}}(\sigma)^2\ \mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[\boldsymbol{y}+\boldsymbol{n}\right] &= 1 && (115)\\
c_{\text{in}}(\sigma)^2\left(\sigma_{\text{data}}^2+\sigma^2\right) &= 1 && (116)\\
c_{\text{in}}(\sigma) &= 1\big/\sqrt{\sigma^2+\sigma_{\text{data}}^2}. && (117)
\end{aligned}$$

Second, we require the effective training target F_{target} to have unit variance:

$$\begin{aligned}
\mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[F_{\text{target}}(\boldsymbol{y},\boldsymbol{n};\sigma)\right] &= 1 && (118)\\
\mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[\tfrac{1}{c_{\text{out}}(\sigma)}\big(\boldsymbol{y}-c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\big)\right] &= 1 && (119)\\
\tfrac{1}{c_{\text{out}}(\sigma)^2}\,\mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[\boldsymbol{y}-c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\right] &= 1 && (120)\\
c_{\text{out}}(\sigma)^2 &= \mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[\boldsymbol{y}-c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})\right] && (121)\\
c_{\text{out}}(\sigma)^2 &= \mathrm{Var}_{\boldsymbol{y},\boldsymbol{n}}\left[\big(1-c_{\text{skip}}(\sigma)\big)\ \boldsymbol{y}+c_{\text{skip}}(\sigma)\ \boldsymbol{n}\right] && (122)\\
c_{\text{out}}(\sigma)^2 &= \big(1-c_{\text{skip}}(\sigma)\big)^2\ \sigma_{\text{data}}^2+c_{\text{skip}}(\sigma)^2\ \sigma^2. && (123)
\end{aligned}$$

Third, we select $c_{\text{skip}}(\sigma)$ to minimize $c_{\text{out}}(\sigma)$, so that the errors of F_θ are amplified as little as possible:

$$c_{\text{skip}}(\sigma)=\arg\min_{c_{\text{skip}}(\sigma)} c_{\text{out}}(\sigma). \tag{124}$$

Since $c_{\text{out}}(\sigma)\geq 0$, we can equivalently write

$$c_{\text{skip}}(\sigma)=\arg\min_{c_{\text{skip}}(\sigma)} c_{\text{out}}(\sigma)^2. \tag{125}$$

This is a convex optimization problem; its solution is uniquely identified by setting the derivative w.r.t. $c_{\text{skip}}(\sigma)$ to zero:

$$\begin{aligned}
0 &= \mathrm{d}\big[c_{\text{out}}(\sigma)^2\big]/\mathrm{d}c_{\text{skip}}(\sigma) && (126)\\
0 &= \mathrm{d}\Big[\big(1-c_{\text{skip}}(\sigma)\big)^2\ \sigma_{\text{data}}^2+c_{\text{skip}}(\sigma)^2\ \sigma^2\Big]/\mathrm{d}c_{\text{skip}}(\sigma) && (127)\\
0 &= \sigma_{\text{data}}^2\ \mathrm{d}\Big[\big(1-c_{\text{skip}}(\sigma)\big)^2\Big]/\mathrm{d}c_{\text{skip}}(\sigma)+\sigma^2\ \mathrm{d}\big[c_{\text{skip}}(\sigma)^2\big]/\mathrm{d}c_{\text{skip}}(\sigma) && (128)\\
0 &= \sigma_{\text{data}}^2\ \left[2\ c_{\text{skip}}(\sigma)-2\right]+\sigma^2\ \left[2\ c_{\text{skip}}(\sigma)\right] && (129)\\
0 &= \left(\sigma^2+\sigma_{\text{data}}^2\right)\ c_{\text{skip}}(\sigma)-\sigma_{\text{data}}^2 && (130)\\
c_{\text{skip}}(\sigma) &= \sigma_{\text{data}}^2\big/\left(\sigma^2+\sigma_{\text{data}}^2\right). && (131)
\end{aligned}$$

We can now substitute Eq. 131 into Eq. 123 to complete the formula for $c_{\text{out}}(\sigma)$:

$$\begin{aligned}
c_{\text{out}}(\sigma)^2 &= \big(1-\left[c_{\text{skip}}(\sigma)\right]\big)^2\ \sigma_{\text{data}}^2+\left[c_{\text{skip}}(\sigma)\right]^2\ \sigma^2 && (132)\\
c_{\text{out}}(\sigma)^2 &= \left(1-\left[\frac{\sigma_{\text{data}}^2}{\sigma^2+\sigma_{\text{data}}^2}\right]\right)^2\ \sigma_{\text{data}}^2+\left[\frac{\sigma_{\text{data}}^2}{\sigma^2+\sigma_{\text{data}}^2}\right]^2\ \sigma^2 && (133)\\
c_{\text{out}}(\sigma)^2 &= \left[\frac{\sigma^2\ \sigma_{\text{data}}}{\sigma^2+\sigma_{\text{data}}^2}\right]^2+\left[\frac{\sigma_{\text{data}}^2\ \sigma}{\sigma^2+\sigma_{\text{data}}^2}\right]^2 && (134)\\
c_{\text{out}}(\sigma)^2 &= \frac{\left(\sigma^2\ \sigma_{\text{data}}\right)^2+\left(\sigma_{\text{data}}^2\ \sigma\right)^2}{\left(\sigma^2+\sigma_{\text{data}}^2\right)^2} && (135)\\
c_{\text{out}}(\sigma)^2 &= \frac{(\sigma\cdot\sigma_{\text{data}})^2\ \left(\sigma^2+\sigma_{\text{data}}^2\right)}{\left(\sigma^2+\sigma_{\text{data}}^2\right)^2} && (136)\\
c_{\text{out}}(\sigma)^2 &= \frac{(\sigma\cdot\sigma_{\text{data}})^2}{\sigma^2+\sigma_{\text{data}}^2} && (137)\\
c_{\text{out}}(\sigma) &= \sigma\cdot\sigma_{\text{data}}\big/\sqrt{\sigma^2+\sigma_{\text{data}}^2}. && (138)
\end{aligned}$$

Fourth, we require the effective weight $w(\sigma)$ to be uniform across noise levels:

$$
\begin{aligned}
w(\sigma) &= 1 && (139)\\
\lambda(\sigma)\, c_{\text{out}}(\sigma)^2 &= 1 && (140)\\
\lambda(\sigma) &= 1/c_{\text{out}}(\sigma)^2 && (141)\\
\lambda(\sigma) &= 1\Big/\left[\frac{\sigma\cdot\sigma_{\text{data}}}{\sqrt{\sigma^2+\sigma_{\text{data}}^2}}\right]^2 && (142)\\
\lambda(\sigma) &= 1\Big/\left[\frac{(\sigma\cdot\sigma_{\text{data}})^2}{\sigma^2+\sigma_{\text{data}}^2}\right] && (143)\\
\lambda(\sigma) &= \left(\sigma^2+\sigma_{\text{data}}^2\right)/(\sigma\cdot\sigma_{\text{data}})^2. && (144)
\end{aligned}
$$

We follow previous work and initialize the output layer weights to zero. Consequently, upon initialization $F_\theta(\cdot)=0$ and the expected value of the loss at each noise level is 1. This can be seen by substituting the choices of $\lambda(\sigma)$ and $c_{\text{skip}}(\sigma)$ into Eq. 109, considered at a fixed σ:

$$
\begin{aligned}
& \mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\left[\lambda(\sigma)\left\|c_{\text{skip}}(\sigma)(\boldsymbol{y}+\boldsymbol{n})+c_{\text{out}}(\sigma)F_\theta\big(c_{\text{in}}(\sigma)(\boldsymbol{y}+\boldsymbol{n});c_{\text{noise}}(\sigma)\big)-\boldsymbol{y}\right\|_2^2\right] && (145)\\
= \;& \mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\left[\frac{\sigma^2+\sigma_{\text{data}}^2}{(\sigma\cdot\sigma_{\text{data}})^2}\left\|\frac{\sigma_{\text{data}}^2}{\sigma^2+\sigma_{\text{data}}^2}(\boldsymbol{y}+\boldsymbol{n})-\boldsymbol{y}\right\|_2^2\right] && (146)\\
= \;& \mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\left[\frac{\sigma^2+\sigma_{\text{data}}^2}{(\sigma\cdot\sigma_{\text{data}})^2}\left\|\frac{\sigma_{\text{data}}^2\boldsymbol{n}-\sigma^2\boldsymbol{y}}{\sigma^2+\sigma_{\text{data}}^2}\right\|_2^2\right] && (147)\\
= \;& \mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\left[\frac{1}{\sigma^2+\sigma_{\text{data}}^2}\left\|\frac{\sigma_{\text{data}}}{\sigma}\boldsymbol{n}-\frac{\sigma}{\sigma_{\text{data}}}\boldsymbol{y}\right\|_2^2\right] && (148)\\
= \;& \frac{1}{\sigma^2+\sigma_{\text{data}}^2}\mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\left[\frac{\sigma_{\text{data}}^2}{\sigma^2}\langle\boldsymbol{n},\boldsymbol{n}\rangle+\frac{\sigma^2}{\sigma_{\text{data}}^2}\langle\boldsymbol{y},\boldsymbol{y}\rangle-2\langle\boldsymbol{y},\boldsymbol{n}\rangle\right] && (149)\\
= \;& \frac{1}{\sigma^2+\sigma_{\text{data}}^2}\left[\frac{\sigma_{\text{data}}^2}{\sigma^2}\underbrace{\mathrm{Var}(\boldsymbol{n})}_{=\sigma^2}+\frac{\sigma^2}{\sigma_{\text{data}}^2}\underbrace{\mathrm{Var}(\boldsymbol{y})}_{=\sigma_{\text{data}}^2}-2\underbrace{\mathrm{Cov}(\boldsymbol{y},\boldsymbol{n})}_{=0}\right] && (150)\\
= \;& 1 && (151)
\end{aligned}
$$

C Reframing previous methods in our framework

In this section, we derive the formulas shown in Table 1 for previous methods, discuss the corresponding original samplers and pre-trained models, and detail the practical considerations associated with using them in our framework.

In practice, the original implementations of these methods differ considerably in terms of the definitions of model inputs and outputs, dynamic range of image data, scaling of $\boldsymbol{x}$, and interpretation of σ. We eliminate this variation by standardizing on a unified setup where the model always matches our definition of F_θ, image data is always represented in the continuous range $[-1,1]$, and the details of $\boldsymbol{x}$ and σ are always in agreement with Eq. 4.

We minimize the accumulation of floating point round-off errors by always executing Algorithms 1 and 2 at double precision (`float64`). However, we still execute the network $F_\theta(\cdot)$ at single precision (`float32`) to minimize runtime and remain faithful to previous work in terms of network architecture.

C.1 Variance preserving formulation

C.1.1 VP sampling

Song et al. [49] define the VP SDE (Eq. 32 in [49]) as

$$
\mathrm{d}\boldsymbol{x} = -\tfrac{1}{2}\left(\beta_{\text{min}}+t\left(\beta_{\text{max}}-\beta_{\text{min}}\right)\right)\boldsymbol{x}\,\mathrm{d}t+\sqrt{\beta_{\text{min}}+t\left(\beta_{\text{max}}-\beta_{\text{min}}\right)}\,\mathrm{d}\omega_t, \qquad (152)
$$

which matches Eq. 10 with the following choices for f and g:

$$f(t) = -\tfrac{1}{2}\,\beta(t), \quad g(t) = \sqrt{\beta(t)}, \quad \text{and} \quad \beta(t) = \big(\beta_{\text{max}} - \beta_{\text{min}}\big)\,t + \beta_{\text{min}}. \tag{153}$$

Let $\alpha(t)$ denote the integral of $\beta(t)$:

$$\begin{aligned}
\alpha(t) &= \int_0^t \beta(\xi)\,\mathrm{d}\xi && (154)\\
&= \int_0^t \Big[\big(\beta_{\text{max}} - \beta_{\text{min}}\big)\,\xi + \beta_{\text{min}}\Big]\,\mathrm{d}\xi && (155)\\
&= \tfrac{1}{2}\,\big(\beta_{\text{max}} - \beta_{\text{min}}\big)\,t^2 + \beta_{\text{min}}\,t && (156)\\
&= \tfrac{1}{2}\,\beta_{\text{d}}\,t^2 + \beta_{\text{min}}\,t, && (157)
\end{aligned}$$

where $\beta_{\text{d}} = \beta_{\text{max}} - \beta_{\text{min}}$. We can now obtain the formula for $\sigma(t)$ by substituting Eq. 153 into Eq. 12:

$$\begin{aligned}
\sigma(t) &= \sqrt{\int_0^t \frac{\big[g(\xi)\big]^2}{\big[s(\xi)\big]^2}\,\mathrm{d}\xi} && (158)\\
&= \sqrt{\int_0^t \frac{\big[\sqrt{\beta(\xi)}\big]^2}{\big[1/\sqrt{e^{\alpha(\xi)}}\big]^2}\,\mathrm{d}\xi} && (159)\\
&= \sqrt{\int_0^t \frac{\beta(\xi)}{1/e^{\alpha(\xi)}}\,\mathrm{d}\xi} && (160)\\
&= \sqrt{\int_0^t \dot{\alpha}(\xi)\,e^{\alpha(\xi)}\,\mathrm{d}\xi} && (161)\\
&= \sqrt{e^{\alpha(t)} - e^{\alpha(0)}} && (162)\\
&= \sqrt{e^{\frac{1}{2}\beta_{\text{d}}t^2 + \beta_{\text{min}}t} - 1}, && (163)
\end{aligned}$$

which matches the "Schedule" row of Table 1. Similarly for $s(t)$:

$$\begin{aligned}
s(t) &= \exp\left(\int_0^t \big[f(\xi)\big]\,\mathrm{d}\xi\right) && (164)\\
&= \exp\left(\int_0^t \big[-\tfrac{1}{2}\,\beta(\xi)\big]\,\mathrm{d}\xi\right) && (165)\\
&= \exp\left(-\tfrac{1}{2}\left[\int_0^t \beta(\xi)\,\mathrm{d}\xi\right]\right) && (166)\\
&= \exp\big(-\tfrac{1}{2}\,\alpha(t)\big) && (167)\\
&= 1/\sqrt{e^{\alpha(t)}} && (168)\\
&= 1/\sqrt{e^{\frac{1}{2}\beta_{\text{d}}t^2 + \beta_{\text{min}}t}}, && (169)
\end{aligned}$$

which matches the "Scaling" row of Table 1. We can equivalently write Eq. 169 in a slightly simpler form by utilizing Eq. 163:

$$s(t) = 1/\sqrt{\sigma(t)^2 + 1}. \tag{170}$$

Song et al. [49] choose to distribute the sampling time steps $\{t_0, \ldots, t_{N-1}\}$ at uniform intervals within $[\epsilon_{\text{s}}, 1]$. This corresponds to setting

$$t_{i<N} = 1 + \tfrac{i}{N-1}(\epsilon_{\text{s}} - 1), \tag{171}$$

which matches the "Time steps" row of Table 1.

Finally, Song et al. [49] set $\beta_{\text{min}} = 0.1$, $\beta_{\text{max}} = 20$, and $\epsilon_{\text{s}} = 10^{-3}$ (Appendix C in [49]), and choose to represent images in the range $[-1, 1]$. These choices are readily compatible with our formulation and are reflected by the "Parameters" section of Table 1.

C.1.2 VP preconditioning

In the VP case, Song et al. [49] approximate the score of $p_t(\boldsymbol{x})$ of Eq. 13 as[1]

$$\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) \;\approx\; \underbrace{-\tfrac{1}{\bar{\sigma}(t)}\, F_\theta\big(\boldsymbol{x};\; (M{-}1)t\big)}_{\mathrm{score}(\boldsymbol{x};F_\theta,t)}, \tag{172}$$

where $M = 1000$, F_θ denotes the network, and $\bar{\sigma}(t)$ corresponds to the standard deviation of the perturbation kernel of Eq. 11.

Let us expand the definitions of $p_t(\boldsymbol{x})$ and $\bar{\sigma}(t)$ from Eqs. 20 and 11, respectively, and substitute $\boldsymbol{x} = s(t)\hat{\boldsymbol{x}}$ to obtain the corresponding formula with respect to the non-scaled variable $\hat{\boldsymbol{x}}$:

$$\nabla_{\boldsymbol{x}} \log\big[p\big(\boldsymbol{x}/s(t);\sigma(t)\big)\big] \;\approx\; -\tfrac{1}{[s(t)\sigma(t)]}\, F_\theta\big(\boldsymbol{x};\; (M{-}1)t\big) \tag{173}$$

$$\nabla_{[s(t)\hat{\boldsymbol{x}}]} \log p\big([s(t)\;\hat{\boldsymbol{x}}]/s(t);\sigma(t)\big) \;\approx\; -\tfrac{1}{s(t)\sigma(t)}\, F_\theta\big([s(t)\;\hat{\boldsymbol{x}}];\; (M{-}1)t\big) \tag{174}$$

$$\tfrac{1}{s(t)}\nabla_{\hat{\boldsymbol{x}}} \log p\big(\hat{\boldsymbol{x}};\sigma(t)\big) \;\approx\; -\tfrac{1}{s(t)\sigma(t)}\, F_\theta\big(s(t)\;\hat{\boldsymbol{x}};\; (M{-}1)t\big) \tag{175}$$

$$\nabla_{\hat{\boldsymbol{x}}} \log p\big(\hat{\boldsymbol{x}};\sigma(t)\big) \;\approx\; -\tfrac{1}{\sigma(t)}\, F_\theta\big(s(t)\;\hat{\boldsymbol{x}};\; (M{-}1)t\big). \tag{176}$$

We can now replace the left-hand side with Eq. 3 and expand the definition of $s(t)$ from Eq. 170:

$$\Big[\Big(D\big(\hat{\boldsymbol{x}};\sigma(t)\big) - \hat{\boldsymbol{x}}\Big)/\sigma(t)^2\Big] \;\approx\; -\tfrac{1}{\sigma(t)}\, F_\theta\big(s(t)\;\hat{\boldsymbol{x}};\; (M{-}1)t\big) \tag{177}$$

$$D\big(\hat{\boldsymbol{x}};\sigma(t)\big) \;\approx\; \hat{\boldsymbol{x}} - \sigma(t)\, F_\theta\big(s(t)\;\hat{\boldsymbol{x}};\; (M{-}1)t\big) \tag{178}$$

$$D\big(\hat{\boldsymbol{x}};\sigma(t)\big) \;\approx\; \hat{\boldsymbol{x}} - \sigma(t)\, F_\theta\Big(\Big[\tfrac{1}{\sqrt{\sigma(t)^2+1}}\Big]\,\hat{\boldsymbol{x}};\; (M{-}1)t\Big), \tag{179}$$

which can be further expressed in terms of σ by replacing $\sigma(t) \to \sigma$ and $t \to \sigma^{-1}(\sigma)$:

$$D(\hat{\boldsymbol{x}};\sigma) \;\approx\; \hat{\boldsymbol{x}} - \sigma\, F_\theta\Big(\tfrac{1}{\sqrt{\sigma^2+1}}\,\hat{\boldsymbol{x}};\; (M{-}1)\,\sigma^{-1}(\sigma)\Big). \tag{180}$$

We adopt the right-hand side of Eq. 180 as the definition of D_θ, obtaining

$$D_\theta(\hat{\boldsymbol{x}};\sigma) = \underbrace{1\cdot}_{c_{\text{skip}}}\hat{\boldsymbol{x}}\; \underbrace{-\,\sigma}_{c_{\text{out}}} \cdot F_\theta\Big(\underbrace{\tfrac{1}{\sqrt{\sigma^2+1}}}_{c_{\text{in}}}\cdot\hat{\boldsymbol{x}};\; \underbrace{(M{-}1)\,\sigma^{-1}(\sigma)}_{c_{\text{noise}}}\Big), \tag{181}$$

where c_{skip}, c_{out}, c_{in}, and c_{noise} match the "Network and preconditioning" section of Table 1.

C.1.3 VP training

Song et al. [49] define their training loss as[2]

$$\mathbb{E}_{t\sim\mathcal{U}(\epsilon_t,1),\boldsymbol{y}\sim p_{\text{data}},\bar{\boldsymbol{n}}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\Big[\big\|\bar{\sigma}(t)\;\mathrm{score}\big(s(t)\;\boldsymbol{y} + \bar{\sigma}(t)\;\bar{\boldsymbol{n}};\; F_\theta, t\big) + \bar{\boldsymbol{n}}\big\|_2^2\Big], \tag{182}$$

where the definition of $\mathrm{score}(\cdot)$ is the same as in Eq. 172. Let us simplify the formula by substituting $\bar{\sigma}(t) = s(t)\sigma(t)$ and $\bar{\boldsymbol{n}} = \boldsymbol{n}/\sigma(t)$, where $\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \sigma(t)^2\mathbf{I})$:

$$\mathbb{E}_{t,\boldsymbol{y},\bar{\boldsymbol{n}}}\Big[\big\|s(t)\sigma(t)\;\mathrm{score}\big(s(t)\;\boldsymbol{y} + [s(t)\sigma(t)]\;\bar{\boldsymbol{n}};\; F_\theta, t\big) + \bar{\boldsymbol{n}}\big\|_2^2\Big] \tag{183}$$

$$= \;\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\big\|s(t)\sigma(t)\;\mathrm{score}\big(s(t)\;\boldsymbol{y} + s(t)\sigma(t)\;[\boldsymbol{n}/\sigma(t)];\; F_\theta, t\big) + [\boldsymbol{n}/\sigma(t)]\big\|_2^2\Big] \tag{184}$$

$$= \;\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\big\|s(t)\sigma(t)\;\mathrm{score}\big(s(t)\;(\boldsymbol{y}+\boldsymbol{n});\; F_\theta, t\big) + \boldsymbol{n}/\sigma(t)\big\|_2^2\Big]. \tag{185}$$

We can express $\mathrm{score}(\cdot)$ in terms of $D_\theta(\cdot)$ by combining Eqs. 172, 170, and 74:

$$\mathrm{score}\big(s(t)\;\boldsymbol{x}; F_\theta, t\big) \;=\; \tfrac{1}{s(t)\sigma(t)^2}\Big(D_\theta\big(\boldsymbol{x};\sigma(t)\big) - \boldsymbol{x}\Big). \tag{186}$$

[1] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/models/utils.py#L144

[2] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/losses.py#L73

Substituting this back into Eq. 185 gives

$$\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\Big\|s(t)\sigma(t)\,\Big[\tfrac{1}{s(t)\sigma(t)^2}\Big(D_\theta\big(\boldsymbol{y}+\boldsymbol{n};\sigma(t)\big)-(\boldsymbol{y}+\boldsymbol{n})\Big)\Big]+\tfrac{1}{\sigma(t)}\,\boldsymbol{n}\Big\|_2^2\Big] \tag{187}$$

$$= \mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\Big\|\tfrac{1}{\sigma(t)}\Big(D_\theta\big(\boldsymbol{y}+\boldsymbol{n};\sigma(t)\big)-(\boldsymbol{y}+\boldsymbol{n})\Big)+\tfrac{1}{\sigma(t)}\,\boldsymbol{n}\Big\|_2^2\Big] \tag{188}$$

$$= \mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\tfrac{1}{\sigma(t)^2}\,\big\|D_\theta\big(\boldsymbol{y}+\boldsymbol{n};\sigma(t)\big)-\boldsymbol{y}\big\|_2^2\Big]. \tag{189}$$

We can further express this in terms of σ by replacing $\sigma(t)\to\sigma$ and $t\to\sigma^{-1}(\sigma)$:

$$\underbrace{\mathbb{E}_{\sigma^{-1}(\sigma)\sim\mathcal{U}(\epsilon_\mathrm{t},1)}}_{p_\text{train}}\mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\Big[\underbrace{\tfrac{1}{\sigma^2}}_{\lambda}\,\big\|D_\theta\big(\boldsymbol{y}+\boldsymbol{n};\sigma\big)-\boldsymbol{y}\big\|_2^2\Big], \tag{190}$$

which matches Eq. 108 with the choices for p_train and λ shown in the "Training" section of Table 1.

C.1.4 VP practical considerations

The pre-trained VP model that we use on CIFAR-10 corresponds to the "DDPM++ cont. (VP)" checkpoint[3] provided by Song et al. [49]. It contains a total of 62 million trainable parameters and supports a continuous range of noise levels $\sigma\in\big[\sigma(\epsilon_\mathrm{t}),\sigma(1)\big]\approx[0.001,152]$, i.e., wider than our preferred sampling range $[0.002,80]$. We import the model directly as $F_\theta(\cdot)$ and run Algorithms 1 and 2 using the definitions in Table 1.

In Figure 2a, the differences between the original sampler (blue) and our reimplementation (orange) are explained by oversights in the implementation of Song et al. [49], also noted by Jolicoeur-Martineau et al. [24] (Appendix D in [24]). First, the original sampler employs an incorrect multiplier[4] in the Euler step: it multiplies $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ by $-1/N$ instead of $(\epsilon_\mathrm{s}-1)/(N-1)$. Second, it either overshoots or undershoots on the last step by going from $t_{N-1}=\epsilon_\mathrm{s}$ to $t_N=\epsilon_\mathrm{s}-1/N$, where $t_N<0$ when $N<1000$. In practice, this means that the generated images contain noticeable noise that becomes quite severe with, e.g., $N=128$. Our formulation avoids these issues, because the step sizes in Algorithm 1 are computed consistently from $\{t_i\}$ and $t_N=0$.

C.2 Variance exploding formulation

C.2.1 VE sampling in theory

Song et al. [49] define the VE SDE (Eq. 30 in [49]) as

$$\mathrm{d}\boldsymbol{x}=\sigma_\text{min}\left(\frac{\sigma_\text{max}}{\sigma_\text{min}}\right)^t\sqrt{2\log\frac{\sigma_\text{max}}{\sigma_\text{min}}}\;\mathrm{d}\omega_t, \tag{191}$$

which matches Eq. 10 with

$$f(t)=0,\quad g(t)=\sigma_\text{min}\sqrt{2\log\sigma_\mathrm{d}}\;\sigma_\mathrm{d}^t,\quad\text{and}\quad \sigma_\mathrm{d}=\sigma_\text{max}/\sigma_\text{min}. \tag{192}$$

The VE formulation does not employ scaling, which can be easily seen from Eq. 12:

$$s(t)=\exp\left(\int_0^t\big[f(\xi)\big]\,\mathrm{d}\xi\right)=\exp\left(\int_0^t\big[0\big]\,\mathrm{d}\xi\right)=\exp(0)=1. \tag{193}$$

[3] vp/cifar10_ddpmpp_continuous/checkpoint_8.pth, https://drive.google.com/drive/folders/1xYjVMx10N9ivQQBIsEoXEeu9nvSGTBrC

[4] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/sampling.py#L182

Substituting Eq. 192 into Eq. 12 suggests the following form for $\sigma(t)$:

$$\begin{aligned}
\sigma(t) &= \sqrt{\int_0^t \frac{\big[g(\xi)\big]^2}{\big[s(\xi)\big]^2}\,\mathrm{d}\xi} && (194)\\
&= \sqrt{\int_0^t \frac{\big[\sigma_{\text{min}}\sqrt{2\log\sigma_{\text{d}}}\;\sigma_{\text{d}}^{\xi}\big]^2}{\big[1\big]^2}\,\mathrm{d}\xi} && (195)\\
&= \sqrt{\int_0^t \sigma_{\text{min}}^2\,\big[2\log\sigma_{\text{d}}\big]\,\big[\sigma_{\text{d}}^{2\xi}\big]\,\mathrm{d}\xi} && (196)\\
&= \sigma_{\text{min}}\sqrt{\int_0^t \Big[\log\big(\sigma_{\text{d}}^2\big)\Big]\,\Big[\big(\sigma_{\text{d}}^2\big)^{\xi}\Big]\,\mathrm{d}\xi} && (197)\\
&= \sigma_{\text{min}}\sqrt{\big(\sigma_{\text{d}}^2\big)^t - \big(\sigma_{\text{d}}^2\big)^0} && (198)\\
&= \sigma_{\text{min}}\sqrt{\sigma_{\text{d}}^{2t} - 1}. && (199)
\end{aligned}$$

Eq. 199 is consistent with the perturbation kernel reported by Song et al. (Eq. 29 in [49]). However, we note that this does not fulfill their intended definition of $\sigma(t) = \sigma_{\text{min}}\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)^t$ (Appendix C in [49]).

C.2.2 VE sampling in practice

The original implementation[5] of Song et al. [49] uses reverse diffusion predictor[6] to integrate discretized reverse probability flow[7] of discretized VE SDE[8]. Put together, these yield the following update rule for $\boldsymbol{x}_{i+1}$:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \tfrac{1}{2}\left(\bar{\sigma}_i^2 - \bar{\sigma}_{i+1}^2\right)\nabla_{\boldsymbol{x}}\log\bar{p}_i(\boldsymbol{x}), \tag{200}$$

where

$$\bar{\sigma}_{i<N} = \sigma_{\text{min}}\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)^{1-i/(N-1)} \quad \text{and} \quad \bar{\sigma}_N = 0. \tag{201}$$

Interestingly, Eq. 200 is identical to the Euler iteration of our ODE with the following choices:

$$s(t) = 1, \quad \sigma(t) = \sqrt{t}, \quad \text{and} \quad t_i = \bar{\sigma}_i^2. \tag{202}$$

These formulas match the "Sampling" section of Table 1, and their correctness can be verified by substituting them into line 5 of Algorithm 1:

$$\begin{aligned}
\boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + (t_{i+1} - t_i)\,\boldsymbol{d}_i && (203)\\
&= \boldsymbol{x}_i + (t_{i+1} - t_i)\left[\left(\frac{\dot{\sigma}(t)}{\sigma(t)} + \frac{\dot{s}(t)}{s(t)}\right)\boldsymbol{x} - \frac{\dot{\sigma}(t)s(t)}{\sigma(t)}D\left(\frac{\boldsymbol{x}}{s(t)};\sigma(t)\right)\right] && (204)\\
&= \boldsymbol{x}_i + (t_{i+1} - t_i)\left[\frac{\dot{\sigma}(t)}{\sigma(t)}\,\boldsymbol{x} - \frac{\dot{\sigma}(t)}{\sigma(t)}\,D\big(\boldsymbol{x};\sigma(t)\big)\right] && (205)\\
&= \boldsymbol{x}_i - (t_{i+1} - t_i)\,\dot{\sigma}(t)\,\sigma(t)\left[\Big(D\big(\boldsymbol{x};\sigma(t)\big) - \boldsymbol{x}\Big)/\sigma(t)^2\right] && (206)\\
&= \boldsymbol{x}_i - (t_{i+1} - t_i)\,\dot{\sigma}(t)\,\sigma(t)\,\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x};\sigma(t)\big) && (207)\\
&= \boldsymbol{x}_i - (t_{i+1} - t_i)\left[\tfrac{1}{2\sqrt{t}}\right]\left[\sqrt{t}\right]\,\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x};\sigma(t)\big) && (208)\\
&= \boldsymbol{x}_i + \tfrac{1}{2}\,(t_i - t_{i+1})\,\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x};\sigma(t)\big) && (209)\\
&= \boldsymbol{x}_i + \tfrac{1}{2}\left(\bar{\sigma}_i^2 - \bar{\sigma}_{i+1}^2\right)\nabla_{\boldsymbol{x}}\log p\big(\boldsymbol{x};\sigma(t)\big), && (210)
\end{aligned}$$

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

[6] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/sampling.py#L191

[7] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/sde_lib.py#L102

[8] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/sde_lib.py#L246

which is made identical to Eq. 200 by the choice $\bar{p}_i(\boldsymbol{x}) = p\big(\boldsymbol{x}; \sigma(t_i)\big)$.

Finally, Song et al. [49] set $\sigma_{\text{min}} = 0.01$ and $\sigma_{\text{max}} = 50$ for CIFAR-10 (Appendix C in [49]), and choose to represent their images in the range $[0, 1]$ to match previous SMLD models. Since our standardized range $[-1, 1]$ is twice as large, we must multiply σ_{min} and σ_{max} by $2\times$ to compensate. The "Parameters" section of Table 1 reflects these adjusted values.

C.2.3 VE preconditioning

In the VE case, Song et al. [49] approximate the score of $p_t(\boldsymbol{x})$ of Eq. 13 directly as[9]

$$\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) \;\approx\; \bar{F}_\theta\big(\boldsymbol{x}; \sigma(t)\big), \tag{211}$$

where the network $\bar{F}_\theta$ is designed to include additional pre-[10] and[11] postprocessing[12] steps:

$$\bar{F}_\theta\big(\boldsymbol{x}; \sigma\big) \;=\; \tfrac{1}{\sigma}\, F_\theta\big(2\boldsymbol{x}{-}1; \log(\sigma)\big). \tag{212}$$

For consistency, we handle the pre- and postprocessing using $\{c_{\text{skip}}, c_{\text{out}}, c_{\text{in}}, c_{\text{noise}}\}$ as opposed to baking them into the network itself.

We cannot use Eqs. 211 and 212 directly in our framework, however, because they assume that the images are represented in range $[0, 1]$. In order to use $[-1, 1]$ instead, we replace $p_t(\boldsymbol{x}) \to p_t(2\boldsymbol{x}{-}1)$, $\boldsymbol{x} \to \frac{1}{2}\boldsymbol{x} + \frac{1}{2}$ and $\sigma \to \frac{1}{2}\sigma$:

$$\nabla_{[\frac{1}{2}\boldsymbol{x}+\frac{1}{2}]} \log p_t\big(2\big[\tfrac{1}{2}\boldsymbol{x} + \tfrac{1}{2}\big]{-}1\big) \;\approx\; \tfrac{1}{[\frac{1}{2}\sigma]}\, F_\theta\big(2\big[\tfrac{1}{2}\boldsymbol{x} + \tfrac{1}{2}\big]{-}1; \log\big[\tfrac{1}{2}\sigma\big]\big) \tag{213}$$

$$2\,\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) \;\approx\; \tfrac{2}{\sigma}\, F_\theta\Big(\boldsymbol{x}; \log\big(\tfrac{1}{2}\sigma\big)\Big) \tag{214}$$

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}; \sigma) \;\approx\; \tfrac{1}{\sigma}\, F_\theta\Big(\boldsymbol{x}; \log\big(\tfrac{1}{2}\sigma\big)\Big). \tag{215}$$

We can now express the model in terms of $D_\theta(\cdot)$ by replacing the left-hand side of Eq. 215 with Eq. 3:

$$\Big(D_\theta\big(\boldsymbol{x}; \sigma\big) - \boldsymbol{x}\Big)/\sigma^2 \;=\; \tfrac{1}{\sigma}\, F_\theta\Big(\boldsymbol{x}; \log\big(\tfrac{1}{2}\sigma\big)\Big) \tag{216}$$

$$D_\theta\big(\boldsymbol{x}; \sigma\big) \;=\; \underbrace{1\,\cdot}_{c_{\text{skip}}}\boldsymbol{x} + \underbrace{\sigma\,\cdot}_{c_{\text{out}}} F_\theta\Big(\underbrace{1\,\cdot}_{c_{\text{in}}}\boldsymbol{x};\; \underbrace{\log\big(\tfrac{1}{2}\sigma\big)}_{c_{\text{noise}}}\Big), \tag{217}$$

where c_{skip}, c_{out}, c_{in}, and c_{noise} match the "Network and preconditioning" section of Table 1.

C.2.4 VE training

Song et al. [49] define their training loss similarly for VP and VE, so we can reuse Eq. 185 by borrowing the definition of $\text{score}(\cdot)$ from Eq. 216:

$$\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\big\|s(t)\sigma(t)\;\text{score}\big(s(t)\;(\boldsymbol{y}+\boldsymbol{n});\; F_\theta, t\big) + \boldsymbol{n}/\sigma(t)\big\|_2^2\Big] \tag{218}$$

$$= \;\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\big\|\sigma(t)\;\text{score}\big(\boldsymbol{y}+\boldsymbol{n};\; F_\theta, t\big) + \boldsymbol{n}/\sigma(t)\big\|_2^2\Big] \tag{219}$$

$$= \;\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\big\|\sigma(t)\;\Big[\Big(D_\theta\big(\boldsymbol{y}+\boldsymbol{n}; \sigma(t)\big) - (\boldsymbol{y}+\boldsymbol{n})\Big)/\sigma(t)^2\Big] + \boldsymbol{n}/\sigma(t)\big\|_2^2\Big] \tag{220}$$

$$= \;\mathbb{E}_{t,\boldsymbol{y},\boldsymbol{n}}\Big[\tfrac{1}{\sigma(t)^2}\,\big\|D_\theta\big(\boldsymbol{y}+\boldsymbol{n}; \sigma(t)\big) - \boldsymbol{y}\big\|_2^2\Big]. \tag{221}$$

For VE training, the original implementation[13] defines $\sigma(t) = \sigma_{\text{min}}\,\big(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\big)^t$. We can thus rewrite Eq. 221 as

$$\underbrace{\mathbb{E}_{\ln(\sigma)\sim\mathcal{U}(\ln(\sigma_{\text{min}}),\ln(\sigma_{\text{max}}))}}_{p_{\text{train}}}\,\mathbb{E}_{\boldsymbol{y},\boldsymbol{n}}\Big[\underbrace{\tfrac{1}{\sigma^2}}_{\lambda}\,\big\|D_\theta\big(\boldsymbol{y}+\boldsymbol{n}; \sigma\big) - \boldsymbol{y}\big\|_2^2\Big], \tag{222}$$

which matches Eq. 108 with the choices for p_{train} and λ shown in the "Training" section of Table 1.

[9] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/models/utils.py#L163
[10] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/models/ncsnpp.py#L239
[11] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/models/ncsnpp.py#L261
[12] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/models/ncsnpp.py#L379
[13] https://github.com/yang-song/score_sde_pytorch/blob/1618ddea340f3e4a2ed7852a0694a809775cf8d0/sde_lib.py#L234

C.2.5 VE practical considerations

The pre-trained VE model that we use on CIFAR-10 corresponds to the "NCSN++ cont. (VE)" checkpoint[14] provided by Song et al. [49]. It contains a total of 63 million trainable parameters and supports a continuous range of noise levels $\sigma \in \big[\sigma(\epsilon_{\text{t}}), \sigma(1)\big] \approx [0.02, 100]$. This is narrower than our preferred sampling range $[0.002, 80]$, so we set $\sigma_{\text{min}} = 0.02$ in all related experiments. Note that this limitation is lifted by our training improvements in config E, so we revert back to using $\sigma_{\text{min}} = 0.002$ with configs E and F in Table 2. When importing the model, we remove the pre- and postprocessing steps shown in Eq. 212 to stay consistent with the definition of $F_\theta(\cdot)$ in Eq. 217. With these changes, we can run Algorithms 1 and 2 using the definitions in Table 1.

In Figure 2b, the differences between the original sampler (blue) and our reimplementation (orange) are explained by floating point round-off errors that the original implementation suffers from at high step counts. Our results are more accurate in these cases because we represent $\boldsymbol{x}_i$ at double precision in Algorithm 1.

C.3 Improved DDPM and DDIM

C.3.1 DDIM ODE formulation

Song et al. [47] make the observation that their deterministic DDIM sampler can be expressed as Euler integration of the following ODE (Eq. 14 in [47]):

$$\mathrm{d}\boldsymbol{x}(t) = \epsilon_\theta^{(t)}\left(\frac{\boldsymbol{x}(t)}{\sqrt{\sigma(t)^2+1}}\right)\,\mathrm{d}\sigma(t), \tag{223}$$

where $\boldsymbol{x}(t)$ is a scaled version of the iterate that appears in their discrete update formula (Eq. 10 in [47]) and ϵ_θ is a model trained to predict the normalized noise vector, i.e., $\epsilon_\theta^{(t)}\big(\boldsymbol{x}(t)/\sqrt{\sigma(t)^2+1}\big) \approx \boldsymbol{n}(t)/\sigma(t)$ for $\boldsymbol{x}(t) = \boldsymbol{y}(t) + \boldsymbol{n}(t)$. In our formulation, D_θ is trained to approximate the clean signal, i.e., $D_\theta\big(\boldsymbol{x}(t); \sigma(t)\big) \approx \boldsymbol{y}$, so we can reinterpret ϵ_θ in terms of D_θ as follows:

$$\begin{aligned}
\boldsymbol{n}(t) &= \boldsymbol{x}(t) - \boldsymbol{y}(t) && (224)\\
\big[\boldsymbol{n}(t)/\sigma(t)\big] &= \big(\boldsymbol{x}(t) - \big[\boldsymbol{y}(t)\big]\big)/\sigma(t) && (225)\\
\epsilon_\theta^{(t)}\big(\boldsymbol{x}(t)/\sqrt{\sigma(t)^2+1}\big) &= \big(\boldsymbol{x}(t) - D_\theta\big(\boldsymbol{x}(t); \sigma(t)\big)\big)/\sigma(t). && (226)
\end{aligned}$$

Assuming ideal $\epsilon(\cdot)$ and $D(\cdot)$ in L_2 sense, we can further simplify the above formula using Eq. 3:

$$\begin{aligned}
\epsilon^{(t)}\big(\boldsymbol{x}(t)/\sqrt{\sigma(t)^2+1}\big) &= \big(\boldsymbol{x}(t) - D\big(\boldsymbol{x}(t); \sigma(t)\big)\big)/\sigma(t) && (227)\\
&= -\sigma(t)\,\Big[\Big(D\big(\boldsymbol{x}(t); \sigma(t)\big) - \boldsymbol{x}(t)\Big)/\sigma(t)^2\Big] && (228)\\
&= -\sigma(t)\,\nabla_{\boldsymbol{x}(t)} \log p\big(\boldsymbol{x}(t); \sigma(t)\big). && (229)
\end{aligned}$$

Substituting Eq. 229 back into Eq. 223 gives

$$\mathrm{d}\boldsymbol{x}(t) = -\sigma(t)\,\nabla_{\boldsymbol{x}(t)} \log p\big(\boldsymbol{x}(t); \sigma(t)\big)\,\mathrm{d}\sigma(t), \tag{230}$$

which we can further simplify by setting $\sigma(t) = t$:

$$\mathrm{d}\boldsymbol{x} = -t\,\nabla_{\boldsymbol{x}} \log p\big(\boldsymbol{x}; \sigma(t)\big)\,\mathrm{d}t. \tag{231}$$

This matches our Eq. 4 with $s(t) = 1$ and $\sigma(t) = t$, reflected by the "Sampling" section of Table 1.

C.3.2 iDDPM time step discretization

The original DDPM formulation of Ho et al. [16] defines the forward process (Eq. 2 in [16]) as a Markov chain that gradually adds Gaussian noise to $\bar{\boldsymbol{x}}_0 \sim p_{\text{data}}$ according to a discrete variance schedule $\{\beta_1, \ldots, \beta_T\}$:

$$q(\bar{\boldsymbol{x}}_t \mid \bar{\boldsymbol{x}}_{t-1}) = \mathcal{N}\big(\bar{\boldsymbol{x}}_t;\ \sqrt{1-\beta_t}\,\bar{\boldsymbol{x}}_{t-1},\ \beta_t\,\mathbf{I}\big). \tag{232}$$

[14] ve/cifar10_ncsnpp_continuous/checkpoint_24.pth, https://drive.google.com/drive/folders/1b0gy_LLgO_DaQBgoWXwlVnL_rcAUgREh

The corresponding transition probability from $\bar{\boldsymbol{x}}_0$ to $\bar{\boldsymbol{x}}_t$ (Eq. 4 in [16]) is given by

$$q(\bar{\boldsymbol{x}}_t \mid \bar{\boldsymbol{x}}_0) = \mathcal{N}\big(\bar{\boldsymbol{x}}_t;\ \sqrt{\bar{\alpha}_t}\,\bar{\boldsymbol{x}}_0,\ (1-\bar{\alpha}_t)\,\mathbf{I}\big), \quad \text{where} \quad \bar{\alpha}_t = \prod_{s=1}^{t}\,(1-\beta_s). \tag{233}$$

Ho et al. [16] define $\{\beta_t\}$ based on a linear schedule and then calculate the corresponding $\{\bar{\alpha}_t\}$ from Eq. 233. Alternatively, one can also define $\{\bar{\alpha}_t\}$ first and then solve for $\{\beta_t\}$:

$$\bar{\alpha}_t = \prod_{s=1}^{t}\,(1-\beta_s) \tag{234}$$

$$\bar{\alpha}_t = \bar{\alpha}_{t-1}\,(1-\beta_t) \tag{235}$$

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}. \tag{236}$$

The improved DDPM formulation of Nichol and Dhariwal [37] employs a cosine schedule for $\bar{\alpha}_t$ (Eq. 17 in [37]), defined as

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad \text{where} \quad f(t) = \cos^2\left(\frac{t/T+s}{1+s}\cdot\frac{\pi}{2}\right), \tag{237}$$

where $s = 0.008$. In their implementation[15], however, Nichol et al. leave out the division by $f(0)$ and simply define[16]

$$\bar{\alpha}_t = \cos^2\left(\frac{t/T+s}{1+s}\cdot\frac{\pi}{2}\right). \tag{238}$$

To prevent singularities near $t = T$, they also clamp β_t to 0.999. We can express the clamping in terms of $\bar{\alpha}_t$ by utilizing Eq. 233 and Eq. 234:

$$\bar{\alpha}'_t = \prod_{s=1}^{t}\,\big(1-[\beta'_s]\big) \tag{239}$$

$$= \prod_{s=1}^{t}\,\Big(1-\min\big([\beta_s],\ 0.999\big)\Big) \tag{240}$$

$$= \prod_{s=1}^{t}\,\left(1-\min\left(1-\frac{\bar{\alpha}_s}{\bar{\alpha}_{s-1}},\ 0.999\right)\right) \tag{241}$$

$$= \prod_{s=1}^{t}\,\max\left(\frac{\bar{\alpha}_s}{\bar{\alpha}_{s-1}},\ 0.001\right). \tag{242}$$

Let us now reinterpret the above formulas in our unified framework. Recall from Table 1 that we denote the original iDDPM sampling steps by $\{u_j\}$ in the order of descending noise level $\sigma(u_j)$, where $j \in \{0,\ldots,M\}$. To harmonize the notation of Eq. 233, Eq. 238, and Eq. 239, we thus have to replace $T \longrightarrow M$ and $t \longrightarrow M-j$:

$$q(\bar{\boldsymbol{x}}_j \mid \bar{\boldsymbol{x}}_M) = \mathcal{N}\big(\bar{\boldsymbol{x}}_j;\ \sqrt{\bar{\alpha}'_j}\,\bar{\boldsymbol{x}}_M,\ (1-\bar{\alpha}'_j)\,\mathbf{I}\big), \tag{243}$$

$$\bar{\alpha}_j = \cos^2\left(\frac{(M-j)/M+C_2}{1+C_2}\cdot\frac{\pi}{2}\right), \quad \text{and} \tag{244}$$

$$\bar{\alpha}'_j = \prod_{s=M-1}^{j}\,\max\left(\frac{\bar{\alpha}_j}{\bar{\alpha}_{j+1}},\ C_1\right) = \bar{\alpha}'_{j+1}\,\max\left(\frac{\bar{\alpha}_j}{\bar{\alpha}_{j+1}},\ C_1\right), \tag{245}$$

where the constants are $C_1 = 0.001$ and $C_2 = 0.008$.

[15] `https://github.com/openai/improved-diffusion`

[16] `https://github.com/openai/improved-diffusion/blob/783b6740edb79fdb7d063250db2c51cc9545dcd1/improved_diffusion/gaussian_diffusion.py#L39`

We can further simplify Eq. 244:

$$\bar{\alpha}_j = \cos^2\left(\frac{(M-j)/M + C_2}{1+C_2} \cdot \frac{\pi}{2}\right) \tag{246}$$

$$= \cos^2\left(\frac{\pi}{2}\,\frac{(1+C_2) - j/M}{1+C_2}\right) \tag{247}$$

$$= \cos^2\left(\frac{\pi}{2} - \frac{\pi}{2}\,\frac{j}{M(1+C_2)}\right) \tag{248}$$

$$= \sin^2\left(\frac{\pi}{2}\,\frac{j}{M(1+C_2)}\right), \tag{249}$$

giving the formula shown in the "Parameters" section of Table 1.

To harmonize the definitions of $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$, we must match the perturbation kernel of Eq. 11 with the transition probability of Eq. 243 for each time step $t = u_j$:

$$p_{0t}\big(\boldsymbol{x}(u_j) \mid \boldsymbol{x}(0)\big) = q(\bar{\boldsymbol{x}}_j \mid \bar{\boldsymbol{x}}_M) \tag{250}$$

$$\mathcal{N}\big(\boldsymbol{x}(u_j);\ s(t)\,\boldsymbol{x}(0),\ s(u_j)^2\,\sigma(u_j)^2\,\mathbf{I}\big) = \mathcal{N}\left(\bar{\boldsymbol{x}}_j;\ \sqrt{\bar{\alpha}'_j}\,\bar{\boldsymbol{x}}_M,\ \big(1-\bar{\alpha}'_j\big)\,\mathbf{I}\right). \tag{251}$$

Substituting $s(t) = 1$ and $\sigma(t) = t$ from Appendix C.3.1, as well as $\bar{\boldsymbol{x}}_M = \boldsymbol{x}(0)$:

$$\mathcal{N}\big(\boldsymbol{x}(u_j);\ \boldsymbol{x}(0),\ u_j^2\,\mathbf{I}\big) = \mathcal{N}\left(\bar{\boldsymbol{x}}_j;\ \sqrt{\bar{\alpha}'_j}\,\boldsymbol{x}(0),\ \big(1-\bar{\alpha}'_j\big)\,\mathbf{I}\right). \tag{252}$$

We can match the means of these two distributions by defining $\bar{\boldsymbol{x}}_j = \sqrt{\bar{\alpha}'_j}\,\boldsymbol{x}(u_j)$:

$$\mathcal{N}\big(\boldsymbol{x}(u_j);\ \boldsymbol{x}(0),\ u_j^2\,\mathbf{I}\big) = \mathcal{N}\left(\sqrt{\bar{\alpha}'_j}\,\boldsymbol{x}(u_j);\ \sqrt{\bar{\alpha}'_j}\,\boldsymbol{x}(0),\ \big(1-\bar{\alpha}'_j\big)\,\mathbf{I}\right) \tag{253}$$

$$= \mathcal{N}\left(\boldsymbol{x}(u_j);\ \boldsymbol{x}(0),\ \frac{1-\bar{\alpha}'_j}{\bar{\alpha}'_j}\,\mathbf{I}\right). \tag{254}$$

Matching the variances and solving for $\bar{\alpha}'_j$ gives

$$u_j^2 = (1-\bar{\alpha}'_j)\ /\ \bar{\alpha}'_j \tag{255}$$

$$u_j^2\,\bar{\alpha}'_j = 1-\bar{\alpha}'_j \tag{256}$$

$$u_j^2\,\bar{\alpha}'_j + \bar{\alpha}'_j = 1 \tag{257}$$

$$(u_j^2+1)\,\bar{\alpha}'_j = 1 \tag{258}$$

$$\bar{\alpha}'_j = 1\ /\ (u_j^2+1). \tag{259}$$

Finally, we can expand the left-hand side using Eq. 245 and solve for u_{j-1}:

$$\bar{\alpha}'_{j+1}\ \max(\bar{\alpha}_j/\bar{\alpha}_{j+1},\ C_1) = 1\ /\ (u_j^2+1) \tag{260}$$

$$\bar{\alpha}'_j\ \max(\bar{\alpha}_{j-1}/\bar{\alpha}_j,\ C_1) = 1\ /\ (u_{j-1}^2+1) \tag{261}$$

$$\big[1\ /\ (u_j^2+1)\big]\ \max(\bar{\alpha}_{j-1}/\bar{\alpha}_j,\ C_1) = 1\ /\ (u_{j-1}^2+1) \tag{262}$$

$$\max(\bar{\alpha}_{j-1}/\bar{\alpha}_j,\ C_1)\ (u_{j-1}^2+1) = u_j^2+1 \tag{263}$$

$$u_{j-1}^2+1 = (u_j^2+1)\ /\ \max(\bar{\alpha}_{j-1}/\bar{\alpha}_j,\ C_1) \tag{264}$$

$$u_{j-1} = \sqrt{\frac{u_j^2+1}{\max(\bar{\alpha}_{j-1}/\bar{\alpha}_j,\ C_1)} - 1}, \tag{265}$$

giving a recurrence formula for $\{u_j\}$, bootstrapped by $u_M = 0$, that matches the "Time steps" row of Table 1.

C.3.3 iDDPM preconditioning and training

We can solve $D_\theta(\cdot)$ from Eq. 227 by substituting $\sigma(t) = t$ from Appendix C.3.1:

$$\epsilon_\theta^{(j)}\left(\boldsymbol{x}/\sqrt{\sigma^2+1}\right) = \big(\boldsymbol{x} - D_\theta(\boldsymbol{x};\sigma)\big)/\sigma \tag{266}$$

$$D_\theta(\boldsymbol{x};\sigma) = \boldsymbol{x} - \sigma\,\epsilon_\theta^{(j)}\left(\boldsymbol{x}/\sqrt{\sigma^2+1}\right). \tag{267}$$

We choose to define $F_\theta(\cdot;j) = \epsilon_\theta^{(j)}(\cdot)$ and solve j from σ by finding the nearest u_j:

$$D_\theta(\boldsymbol{x};\sigma) = \underbrace{1\cdot}_{c_{\text{skip}}}\boldsymbol{x}\ \underbrace{-\sigma}_{c_{\text{out}}}\cdot F_\theta\Big(\underbrace{\tfrac{1}{\sqrt{\sigma^2+1}}}_{c_{\text{in}}}\cdot\boldsymbol{x};\ \underbrace{\arg\min_j|u_j-\sigma|}_{c_{\text{noise}}}\Big), \tag{268}$$

where c_{skip}, c_{out}, c_{in}, and c_{noise} match the "Network and preconditioning" section of Table 1.

Note that Eq. 268 is identical to the VP preconditioning formula in Eq. 181. Furthermore, Nichol and Dhariwal [37] define their main training loss L_{simple} (Eq. 14 in [37]) the same way as Song et al. [49], with σ drawn uniformly from $\{u_j\}$. Thus, we can reuse Eq. 190 with $\sigma = u_j$, $j \sim \mathcal{U}(0, M-1)$, and $\lambda(\sigma) = 1/\sigma^2$, matching the "Training" section of Table 1. In addition to L_{simple}, Nichol and Dhariwal [37] also employ a secondary loss term L_{vlb}; we refer the reader to Section 3.1 in [37] for details.

C.3.4 iDDPM practical considerations

The pre-trained iDDPM model that we use on ImageNet-64 corresponds to the "ADM (dropout)" checkpoint[17] provided by Dhariwal and Nichol [9]. It contains 296 million trainable parameters and supports a discrete set of $M = 1000$ noise levels $\sigma \in \{u_j\} \approx \{20291, 642, 321, 214, 160, 128, 106, 92, 80, 71, \ldots, 0.0064\}$. The fact that we can only evaluate F_θ these specific choices of σ presents three practical challenges:

1. In the context of DDIM, we must choose how to resample $\{u_j\}$ to yield $\{t_i\}$ for $N \neq M$. Song et al. [47] employ a simple resampling scheme where $t_i = u_{k\cdot i}$ for resampling factor $k \in \mathbb{Z}^+$. This scheme, however, requires that $1000 \equiv 0 \pmod N$, which limits the possible choices for N considerably. Nichol and Dhariwal [37], on the other hand, employ a more flexible scheme where $t_i = u_j$ with $j = \lfloor (M-1)/(N-1)\cdot i \rfloor$. We note, however, that in practice the values of $u_{j<8}$ are considerably larger than our preferred $\sigma_{\text{max}} = 80$. We choose to skip these values by defining $j = \lfloor j_0 + (M-1-j_0)/(N-1)\cdot i \rfloor$ with $j_0 = 8$, matching the "Time steps" row in Table 1. In Figure 2c, the differences between the original sampler (blue) and our reimplementation (orange) are explained by this choice.
2. In the context of our time step discretization (Eq. 5), we must ensure that $\sigma_i \in \{u_j\}$. We accomplish this by rounding each σ_i to its nearest supported counterpart, i.e., $\sigma_i \leftarrow u_{\arg\min_j|u_j-\sigma_i|}$, and setting $\sigma_{\text{min}} = 0.0064 \approx u_{N-1}$. This is sufficient, because Algorithm 1 only evaluates $D_\theta(\cdot;\sigma)$ with $\sigma \in \{\sigma_{i<N}\}$.
3. In the context of our stochastic sampler, we must ensure that $\hat{t}_i \in \{u_j\}$. We accomplish this by replacing line 5 of Algorithm 2 with $\hat{t}_i \leftarrow u_{\arg\min_j|u_j-(t_i+\gamma_i t_i)|}$.

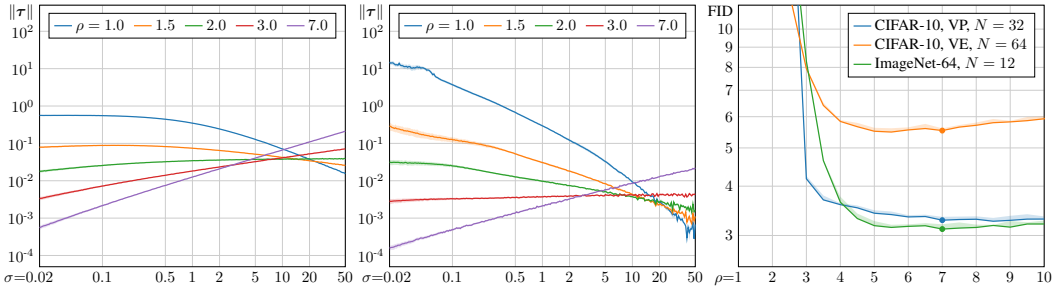
With these changes, we are able to import the pre-trained model directly as $F_\theta(\cdot)$ and run Algorithms 1 and 2 using the definitions in Table 1. Note that the model outputs both $\epsilon_\theta(\cdot)$ and $\Sigma_\theta(\cdot)$, as described in Section 3.1 of [37]; we use only the former and ignore the latter.

D Further analysis of deterministic sampling

D.1 Truncation error analysis and choice of discretization parameters

As discussed in Section 3, the fundamental reason why diffusion models tend to require a large number of sampling steps is that any numerical ODE solver is necessarily an approximation; the

[17] `https://openaipublic.blob.core.windows.net/diffusion/jul-2021/64x64_diffusion.pt`



(a) Truncation error, VE + Euler (b) Truncation error, VE + Heun (c) FID as a function of ρ

Figure 13: **(a)** Local truncation error (y-axis) at different noise levels (x-axis) using Euler's method with the VE-based CIFAR-10 model. Each curve corresponds to a different time step discretization, defined for $N = 64$ and a specific choice for the polynomial exponent ρ. The values represent the root mean square error (RMSE) between one Euler iteration and a sequence of multiple smaller Euler iterations, representing the ground truth. The shaded regions, barely visible at low σ, represent standard deviation over different latents $\boldsymbol{x}_0$. **(b)** Corresponding error curves for Heun's 2nd order method (Algorithm 1). **(c)** FID (y-axis) as a function of the polynomial exponent (x-axis) for different models, measured using Heun's 2nd order method. The shaded regions indicate the range of variation between the lowest and highest observed FID, and the dots indicate the value of ρ that we use in all other experiments.

larger the steps, the farther away we drift from the true solution at each step. Specifically, given the value of $\boldsymbol{x}_{i-1}$ at time step $i-1$, the solver approximates the true $\boldsymbol{x}_i^*$ as $\boldsymbol{x}_i$, resulting in local truncation error $\boldsymbol{\tau}_i = \boldsymbol{x}_i^* - \boldsymbol{x}_i$. The local errors get accumulated over the N steps, ultimately leading to global truncation error $\boldsymbol{e}_N$.

Euler's method is a first order ODE solver, meaning that $\boldsymbol{\tau}_i = \mathcal{O}\left(h_i^2\right)$ for any sufficiently smooth $\boldsymbol{x}(t)$, where $h_i = |t_i - t_{i-1}|$ is the local step size [50]. In other words, there exist some C and H such that $||\boldsymbol{\tau}_i|| < Ch_i^2$ for every $h_i < H$, i.e., halving h_i reduces $\boldsymbol{\tau}_i$ by 4×. Furthermore, if we assume that D_θ is Lipschitz continuous — which is true for all network architectures considered in this paper — the global truncation error is bounded by $||\boldsymbol{e}_N|| \leq E \max_i ||\boldsymbol{\tau}_i||$, where the value of E depends on N, t_0, t_N, and the Lipschitz constant [50]. Thus, reducing the global error for given N, which in turn enables reducing N itself, boils down to choosing the solver and $\{t_i\}$ so that $\max_i ||\boldsymbol{\tau}_i||$ is minimized.

To gain insight on how the local truncation error behaves in practice, we measure the values of $\boldsymbol{\tau}_i$ over different noise levels using the VE-based CIFAR-10 model. For a given noise level, we set $t_i = \sigma^{-1}(\sigma_i)$ and choose some $t_{i-1} > t_i$ depending on the case. We then sample $\boldsymbol{x}_{i-1}$ from $p(\boldsymbol{x}; \sigma_{i-1})$ and estimate the true $\boldsymbol{x}_i^*$ by performing 200 Euler steps over uniformly selected subintervals between t_{i-1} and t. Finally, we plot the mean and standard deviation of the root mean square error (RMSE), i.e., $||\boldsymbol{\tau}_i||/\sqrt{\dim \boldsymbol{\tau}}$, as a function of σ_i, averaged over 200 random samples of $\boldsymbol{x}_{i-1}$. Results for Euler's method are shown in Figure 13a, where the blue curve corresponds to uniform step size $h_\sigma = 1.25$ with respect to σ, i.e., $\sigma_{i-1} = \sigma_i + h_\sigma$ and $t_{i-1} = \sigma^{-1}(\sigma_{i-1})$. We see that the error is very large (RMSE ≈ 0.56) for low noise levels ($\sigma_i \leq 0.5$) and considerably smaller for high noise levels. This is in line with the common intuition that, in order to reduce $\boldsymbol{e}_N$, the step size should be decreased monotonically with decreasing σ. Each curve is surrounded by a shaded region that indicates standard deviation, barely visible at low values of σ. This indicates that $\boldsymbol{\tau}_i$ is nearly constant with respect to $\boldsymbol{x}_{i-1}$, and thus there would be no benefit in varying $\{t_i\}$ schedule on a per-sample basis.

A convenient way to vary the local step size depending on the noise level is to define $\{\sigma_i\}$ as a linear resampling of some monotonically increasing, unbounded warp function $w(z)$. In other words, $\sigma_{i<N} = w(Ai + B)$ and $\sigma_N = 0$, where constants A and B are selected so that $\sigma_0 = \sigma_{\text{max}}$ and $\sigma_{N-1} = \sigma_{\text{min}}$. In practice, we set $\sigma_{\text{min}} = \max(\sigma_{\text{lo}}, 0.002)$ and $\sigma_{\text{max}} = \min(\sigma_{\text{hi}}, 80)$, where σ_{lo} and σ_{hi} are the lowest and highest noise levels supported by a given model, respectively; we have found these choices to perform reasonably well in practice. Now, to balance $\boldsymbol{\tau}_i$ between low and high noise levels, we can, for example, use a polynomial warp function $w(z) = z^\rho$ parameterized by the

exponent ρ. This choice leads to the following formula for $\{\sigma_i\}$:

$$\sigma_{i<N} = \left(\sigma_{\max}{}^{\frac{1}{\rho}} + \frac{i}{N-1}\left(\sigma_{\min}{}^{\frac{1}{\rho}} - \sigma_{\max}{}^{\frac{1}{\rho}}\right)\right)^{\rho}, \sigma_N = 0, \tag{269}$$

which reduces to uniform discretization when $\rho = 1$ and gives more and more emphasis to low noise levels as ρ increases.[18]

Based on the value of σ_i, we can now compute $\sigma_{i-1} = \left(\sigma_i^{1/\rho} - A\right)^{\rho}$, which enables us to visualize $\boldsymbol{\tau}_i$ for different choices of ρ in Figure 13a. We see that increasing ρ reduces the error for low noise levels ($\sigma < 10$) while increasing it for high noise levels ($\sigma > 10$). Approximate balance is achieved at $\rho = 2$, but RMSE remains relatively high (~ 0.03), meaning that Euler's method drifts away from the correct result by several ULPs at each step. While the error could be reduced by increasing N, we would ideally like the RMSE to be well below 0.01 even with low step counts.

Heun's method introduces an additional correction step for $\boldsymbol{x}_{i+1}$ to account for the fact that $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ may change between t_i and t_{i+1}; Euler's method assumes it to be constant. The correction leads to cubic convergence of the local truncation error, i.e., $\boldsymbol{\tau}_i = \mathcal{O}\left(h_i^3\right)$, at the cost of one additional evaluation of D_θ per step. We discuss the general family of Heun-like schemes later in Appendix D.2. Figure 13b shows local truncation error for Heun's method using the same setup as Figure 13a. We see that the differences in $||\boldsymbol{\tau}_i||$ are generally more pronounced, which is to be expected given the quadratic vs. cubic convergence of the two methods. Cases where Euler's method has low RMSE tend to have even lower RMSE with Heun's method, and vice versa for cases with high RMSE. Most remarkably, the red curve shows almost constant RMSE $\in [0.0030, 0.0045]$. This means that the combination of Eq. 269 and Heun's method is, in fact, very close to optimal with $\rho = 3$.

Thus far, we have only considered the raw numerical error, i.e., component-wise deviation from the true result in RGB space. The raw numerical error is relevant for certain use cases, e.g., image manipulation where the ODE is first evaluated in the direction of increasing t and then back to $t = 0$ again — in this case, $||\boldsymbol{e}_N||$ directly tells us how much the original image degrades in the process and we can use $\rho = 3$ to minimize it. Considering the generation of novel images from scratch, however, it is reasonable to expect different noise levels to introduce different kinds of errors that may not necessarily be on equal footing considering their perceptual importance. We investigate this in Figure 13c, where we plot FID as a function of ρ for different models and different choices of N. Note that the ImageNet-64 model was only trained for a discrete set of noise levels; in order to use it with Eq. 269, we round each t_i to its nearest supported counterpart, i.e., $t_i' = u_{\arg\min_j |u_j - t_i|}$.

From the plot, we can see that even though $\rho = 3$ leads to relatively good FID, it can be reduced further by choosing $\rho > 3$. This corresponds to intentionally introducing error at high noise levels to reduce it at low noise levels, which makes intuitive sense because the value of $\sigma_{\max}$ is somewhat arbitrary to begin with — increasing $\sigma_{\max}$ can have a large impact on $||\boldsymbol{e}_N||$, but it does not affect the resulting image distribution nearly as much. In general, we have found $\rho = 7$ to perform reasonably well in all cases, and use this value in all other experiments.

D.2 General family of 2nd order Runge–Kutta variants

Heun's method illustrated in Algorithm 1 belongs to a family of explicit two-stage 2nd order Runge–Kutta methods, each having the same computational cost. A common parameterization [50] of this family is,

$$\boldsymbol{d}_i = f(\boldsymbol{x}_i; t_i) \;\; ; \;\; \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + h\Big[\Big(1 - \tfrac{1}{2\alpha}\Big)\boldsymbol{d}_i + \tfrac{1}{2\alpha} f(\boldsymbol{x}_i + \alpha h \boldsymbol{d}_i; t_i + \alpha h)\Big], \tag{270}$$

where $h = t_{i+1} - t_i$ and α is a parameter that controls where the additional gradient is evaluated and how much it influences the step taken. Setting $\alpha = 1$ corresponds to Heun's method, and $\alpha = \frac{1}{2}$ and $\alpha = \frac{2}{3}$ yield so-called midpoint and Ralston methods, respectively. All these variants differ in the kind of approximation error they incur due to the geometry of the underlying function f.

To establish the optimal α in our use case, we ran a separate series of experiments. According to the results, it appears that $\alpha = 1$ is very close to being optimal. Nonetheless, the experimentally

[18] In the limit, Eq. 269 reduces to the same geometric sequence employed by original VE ODE when $\rho \to \infty$. Thus, our discretization can be seen as a parametric generalization of the one proposed by Song et al. [49].

Algorithm 3 Deterministic sampling using general 2nd order Runge–Kutta, $\sigma(t) = t$ and $s(t) = 1$.

1: **procedure** ALPHASAMPLER($D_\theta(\boldsymbol{x};\sigma),\ t_{i\in\{0,\ldots,N\}},\ \alpha$)
2: **sample** $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ t_0^2\,\mathbf{I}\big)$
3: **for** $i \in \{0,\ldots,N-1\}$ **do**
4: $h_i \leftarrow t_{i+1} - t_i$ ▷ Step length
5: $\boldsymbol{d}_i \leftarrow \big(\boldsymbol{x}_i - D_\theta(\boldsymbol{x}_i;t_i)\big)/t_i$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $(\boldsymbol{x},\ t_i)$
6: $(\boldsymbol{x}'_i,\ t'_i) \leftarrow (\boldsymbol{x}_i + \alpha h \boldsymbol{d}_i,\ t_i + \alpha h)$ ▷ Additional evaluation point
7: **if** $t'_i \neq 0$ **then**
8: $\boldsymbol{d}'_i \leftarrow \big(\boldsymbol{x}'_i - D_\theta(\boldsymbol{x}'_i;t'_i)\big)/t'_i$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $(\boldsymbol{x}'_i,\ t'_i)$
9: $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + h\Big[\Big(1 - \frac{1}{2\alpha}\Big)\boldsymbol{d}_i + \frac{1}{2\alpha}\boldsymbol{d}'_i\Big]$ ▷ Second order step from t_i to t_{i+1}
10: **else**
11: $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + h\boldsymbol{d}_i$ ▷ Euler step from t_i to t_{i+1}
12: **return** $\boldsymbol{x}_N$

best choice was $\alpha = 1.1$ that performed slightly better, even though values greater than one are theoretically hard to justify as they overshoot the target t_{i+1}. As we have no good explanation for this observation and cannot tell if it holds in general, we chose not to make α a new hyperparameter and instead fixed it to 1, corresponding exactly to Heun's method. Further analysis is left as future work, including the possibility of having α vary during sampling.

An additional benefit of setting $\alpha = 1$ is that it makes it possible to use pre-trained neural networks $D_\theta(\boldsymbol{x};\sigma)$ that have been trained only for specific values of σ. This is because a Heun step evaluates the additional gradient at exactly t_{i+1} unlike the other 2nd order variants. Hence it is sufficient to ensure that each t_i corresponds to a value of σ that the network was trained for.

Algorithm 3 shows the pseudocode for a general 2nd order solver parameterized by α. For clarity, the pseudocode assumes the specific choices of $\sigma(t) = t$ and $s(t) = 1$ that we advocate in Section 3. Note that the fallback to Euler step (line 11) can occur only when $\alpha \geq 1$.

E Further results with stochastic sampling

E.1 Image degradation due to excessive stochastic iteration

Figure 14 illustrates the image degradation caused by excessive Langevin iteration (Section 4, "Practical considerations"). These images are generated by doing a specified number of iterations at a fixed noise level σ so that at each iteration an equal amount of noise is added and removed. In theory, Langevin dynamics should bring the distribution towards the ideal distribution $p(\boldsymbol{x};\sigma)$ but as noted in Section 4, this holds only if the denoiser $D_\theta(\boldsymbol{x};\sigma)$ induces a conservative vector field in Eq. 3.

As seen in the figure, it is clear that the image distribution suffers from repeated iteration in all cases, although the exact failure mode depends on dataset and noise level. For low noise levels (below 0.2 or so), the images tend to oversaturate starting at 2k iterations and become fully corrupted after that. Our heuristic of setting $S_{\text{tmin}} > 0$ is designed to prevent stochastic sampling altogether at very low noise levels to avoid this effect.

For high noise levels, we can see that iterating without the standard deviation correction, i.e., when $S_{\text{noise}} = 1.000$, the images tend to become more abstract and devoid of color at high iteration counts; this is especially visible in the 10k column of CIFAR-10 where the images become mostly black and white with no discernible backgrounds. Our heuristic inflation of standard deviation by setting $S_{\text{noise}} > 1$ counteracts this tendency efficiently, as seen in the corresponding images on the right hand side of the figure. Notably, this still does not fix the oversaturation and corruption at low noise levels, suggesting multiple sources for the detrimental effects of excessive iteration. Further research will be required to better understand the root causes of these observed effects.

Figure 15 presents the output quality of our stochastic sampler in terms of FID as a function of S_{churn} at fixed NFE, using pre-trained networks of Song et al. [49] and Dhariwal and Nichol [9]. Generally, for each case and combination of our heuristic corrections, there is an optimal amount of stochasticity after which the results start to degrade. It can also be seen that regardless of the value of S_{churn}, the

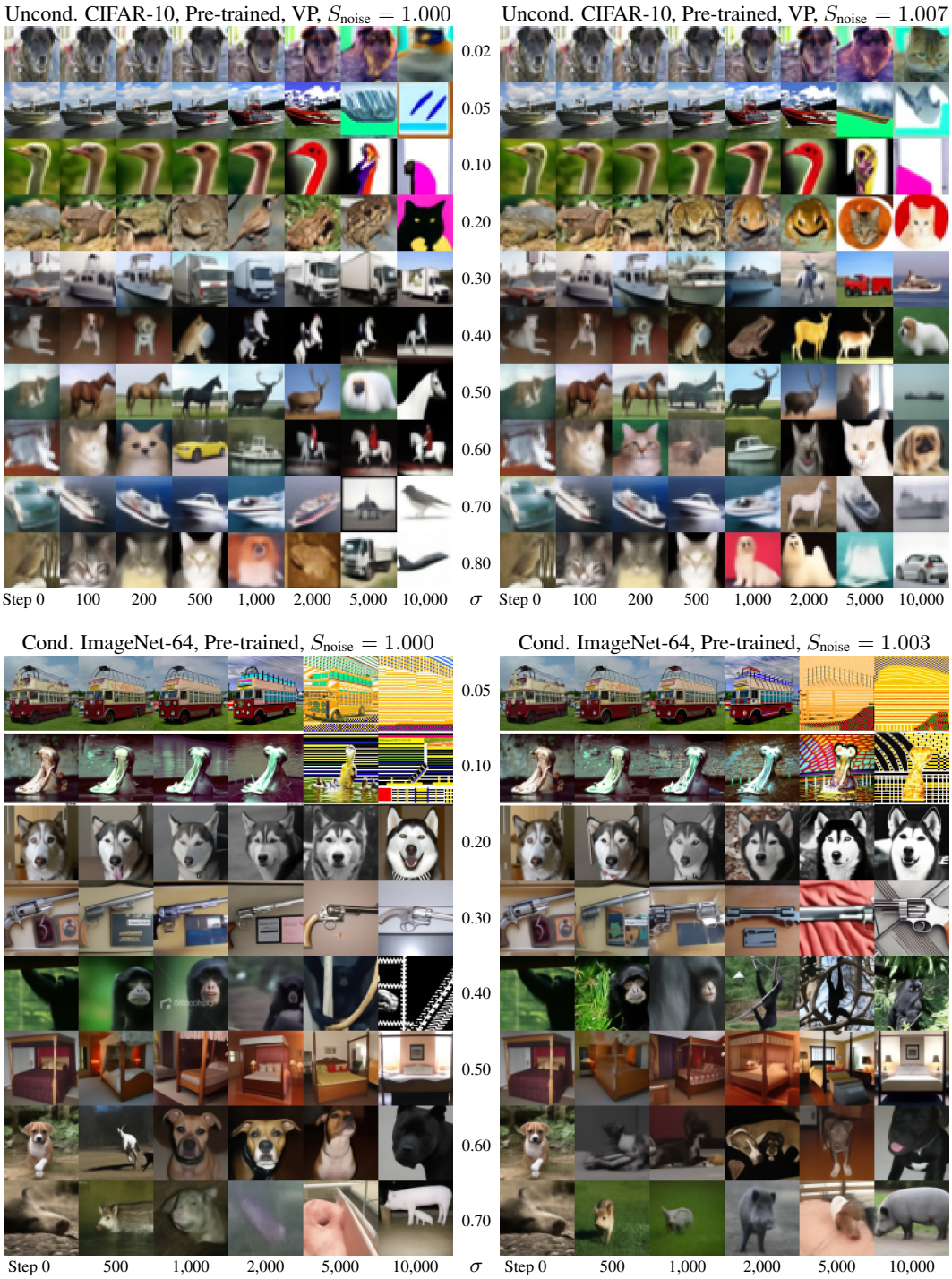
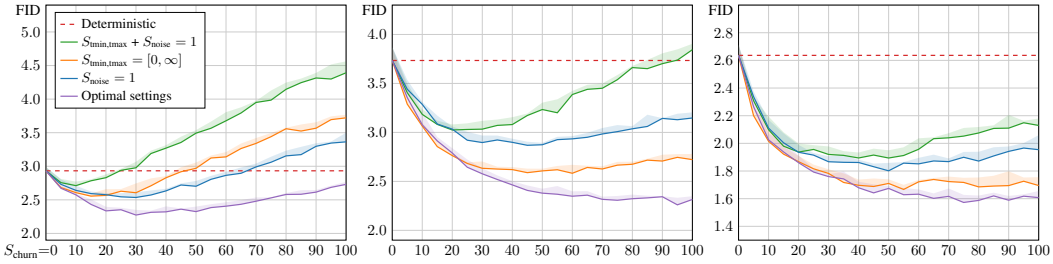


Figure 14: Gradual image degradation with repeated addition and removal of noise. We start with a random image drawn from $p(\boldsymbol{x};\sigma)$ (first column) and run Algorithm 2 for a certain number of steps (remaining columns) with fixed $\gamma_i = \sqrt{2} - 1$. Each row corresponds to a specific choice of σ (indicated in the middle) that we keep fixed throughout the entire process. We visualize the results after running them through the denoiser, i.e., $D_\theta(\boldsymbol{x}_i;\sigma)$.



(a) Uncond. CIFAR-10, VP (b) Uncond. CIFAR-10, VE (c) Class-cond. ImageNet-64

Figure 15: Ablations of our stochastic sampler (Algorithm 2) parameters using pre-trained networks of Song et al. [49] and Dhariwal and Nichol [9]. Each curve shows FID (y-axis) as a function of S_{churn} (x-axis) for $N = 256$ steps (NFE = 511). The dashed red lines correspond to our deterministic sampler (Algorithm 1), equivalent to setting $S_{\text{churn}} = 0$. The purple curves correspond to optimal choices for $\{S_{\text{tmin}}, S_{\text{tmax}}, S_{\text{noise}}\}$, found separately for each case using grid search. Orange, blue, and green correspond to disabling the effects of $S_{\text{tmin,tmax}}$ and/or S_{noise}. The shaded regions indicate the range of variation between the lowest and highest observed FID.

Table 5: Parameters used for the stochastic sampling experiments in Section 4.

Parameter	CIFAR-10		ImageNet		Grid search
	VP	VE	Pre-trained	Our model	
S_{churn}	30	80	80	40	0, 10, 20, 30, ..., 70, 80, 90, 100
S_{tmin}	0.01	0.05	0.05	0.05	0, 0.005, 0.01, 0.02, ..., 1, 2, 5, 10
S_{tmax}	1	1	50	50	0.2, 0.5, 1, 2, ..., 10, 20, 50, 80
S_{noise}	1.007	1.007	1.003	1.003	1.000, 1.001, ..., 1.009, 1.010

best results are obtained by enabling all corrections, although whether S_{noise} or $S_{\text{tmin,tmax}}$ is more important depends on the case.

E.2 Stochastic sampling parameters

Table 5 lists the values for S_{churn}, S_{tmin}, S_{tmax}, and S_{noise} that we used in our stochastic sampling experiments. These were determined with a grid search over the combinations listed in the rightmost column. It can be seen that the optimal parameters depend on the case; better understanding of the degradation phenomena will hopefully give rise to more direct ways of handling the problem in the future.

F Implementation details

We implemented our techniques in a newly written codebase, based loosely on the original implementations by Song et al.[19] [49], Dhariwal and Nichol[20] [9], and Karras et al.[21] [26]. We performed extensive testing to verify that our implementation produced exactly the same results as previous work, including samplers, pre-trained models, network architectures, training configurations, and evaluation. We ran all experiments using PyTorch 1.10.0, CUDA 11.4, and CuDNN 8.2.0 on NVIDIA DGX-1's with 8 Tesla V100 GPUs each.

Our implementation and pre-trained models are available at `https://github.com/NVlabs/edm`

[19] `https://github.com/yang-song/score_sde_pytorch`
[20] `https://github.com/openai/guided-diffusion`
[21] `https://github.com/NVlabs/stylegan3`

Table 6: Our augmentation pipeline. Each training image undergoes a combined geometric transformation based on 8 random parameters that receive non-zero values with a certain probability. The model is conditioned with an additional 9-dimensional input vector derived from these parameters.

Augmentation	Transformation	Parameters	Prob.	Conditioning	Constants
x-flip	$\textsc{Scale2D}\big(1-2a_0,\ 1\big)$	$a_0 \sim \mathcal{U}\{0,1\}$	100%	a_0	$A_{\text{prob}} = 12\%$
y-flip	$\textsc{Scale2D}\big(1,\ 1-2a_1\big)$	$a_1 \sim \mathcal{U}\{0,1\}$	A_{prob}	a_1	or 15%
Scaling	$\textsc{Scale2D}\big((A_{\text{scale}})^{a_2},$ $(A_{\text{scale}})^{a_2}\big)$	$a_2 \sim \mathcal{N}(0,1)$	A_{prob}	a_2	$A_{\text{scale}} = 2^{0.2}$
Rotation	$\textsc{Rotate2D}\big(-a_3\big)$	$a_3 \sim \mathcal{U}(-\pi,\pi)$	A_{prob}	$\cos a_3 - 1$ $\sin a_3$	
Anisotropy	$\textsc{Rotate2D}\big(a_4\big)$ $\textsc{Scale2D}\big((A_{\text{aniso}})^{a_5},$ $1/(A_{\text{aniso}})^{a_5}\big)$ $\textsc{Rotate2D}\big(-a_4\big)$	$a_4 \sim \mathcal{U}(-\pi,\pi)$ $a_5 \sim \mathcal{N}(0,1)$	A_{prob}	$a_5\ \cos a_4$ $a_5\ \sin a_4$	$A_{\text{aniso}} = 2^{0.2}$
Translation	$\textsc{Translate2D}\big((A_{\text{trans}})a_6,$ $(A_{\text{trans}})a_7\big)$	$a_6 \sim \mathcal{N}(0,1)$ $a_7 \sim \mathcal{N}(0,1)$	A_{prob}	a_6 a_7	$A_{\text{trans}} = 1/8$

F.1 FID calculation

We calculate FID [15] between 50,000 generated images and all available real images, without any augmentation such as x-flips. We use the pre-trained Inception-v3 model provided with StyleGAN3[22] [26] that is, in turn, a direct PyTorch translation of the original TensorFlow-based model[23]. We have verified that our FID implementation produces identical results compared to Dhariwal and Nichol [9] and Karras et al. [26]. To reduce the impact of random variation, typically in the order of $\pm 2\%$, we compute FID three times in each experiment and report the minimum. We also highlight the difference between the highest and lowest achieved FID in Figures 4, 5b, 13c, and 15.

F.2 Augmentation regularization

In Section 5, we propose to combat overfitting of D_θ using conditional augmentation. We build our augmentation pipeline around the same concepts that were originally proposed by Karras et al. [25] in the context of GANs. In practice, we employ a set of 6 geometric transformations; we have found other types of augmentations, such as color corruption and image-space filtering, to be consistently harmful for diffusion-based models.

The details of our augmentation pipeline are shown in Table 6. We apply the augmentations independently to each training image $\boldsymbol{y} \sim p_{\text{data}}$ prior to adding the noise $\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. First, we determine whether to enable or disable each augmentation based on a weighted coin toss. The probability of enabling a given augmentation ("Prob." column) is fixed to 12% for CIFAR-10 and 15% for FFHQ and AFHQv2, except for x-flips that are always enabled. We then draw 8 random parameters from their corresponding distributions ("Parameters" column); if a given augmentation is disabled, we override the associated parameters with zero. Based on these, we construct a homogeneous 2D transformation matrix based on the parameters ("Transformation" column). This transformation is applied to the image using the implementation of [25] that employs $2\times$ supersampled high-quality Wavelet filters. Finally, we construct a 9-dimensional conditioning input vector ("Conditioning" column) and feed it to the denoiser network, in addition to the image and noise level inputs.

The role of the conditioning input is to present the network with a set of auxiliary tasks; in addition to the main task of modeling $p(\boldsymbol{x};\sigma)$, we effectively ask the network to also model an infinite set of distributions $p(\boldsymbol{x};\sigma,\boldsymbol{a})$ for each possible choice of the augmentation parameters $\boldsymbol{a}$. These auxiliary tasks provide the network with a large variety of unique training samples, preventing it from

[22] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/metrics/inception-2015-12-05.pkl

[23] http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz

Table 7: Hyperparameters used for the training runs in Section 5.

Hyperparameter	CIFAR-10		FFHQ & AFHQv2		ImagetNet
	Baseline	Ours	Baseline	Ours	Ours
Number of GPUs	4	8	4	8	32
Duration (Mimg)	200	200	200	200	2500
Minibatch size	128	512	128	256	4096
Gradient clipping	✓	–	✓	–	–
Mixed-precision (FP16)	–	–	–	–	✓
Learning rate $\times 10^4$	2	10	2	2	1
LR ramp-up (Mimg)	0.64	10	0.64	10	10
EMA half-life (Mimg)	0.89 / 0.9 (VP / VE)	0.5	0.89 / 0.9 (VP / VE)	0.5	50
Dropout probability	10%	13%	10%	5% / 25% (FFHQ / AFHQ)	10%
Channel multiplier	128	128	128	128	192
Channels per resolution	1-2-2-2	2-2-2	1-1-2-2-2	1-2-2-2	1-2-3-4
Dataset x-flips	✓	–	✓	–	–
Augment probability	–	12%	–	15%	–

overfitting to any individual sample. Still, the auxiliary tasks appear to be beneficial for the main task; we speculate that this is because the denoising operation itself is similar for every choice of $\boldsymbol{a}$.

We have designed the conditioning input so that zero corresponds to the case where no augmentations were applied. During sampling, we simply set $\boldsymbol{a} = \mathbf{0}$ to obtain results consistent with the main task. We have not observed any leakage between the auxiliary tasks and the main task; the generated images exhibit no traces of out-of-domain geometric transformations even with $A_{\text{prob}} = 100\%$. In practice, this means that we are free to choose the constants $\{A_{\text{prob}}, A_{\text{scale}}, A_{\text{aniso}}, A_{\text{trans}}\}$ any way we like as long as the results improve. Horizontal flips serve as an interesting example. Most of the prior work augments the training set with random x-flips, which is beneficial for most datasets but has the downside that any text or logos may appear mirrored in the generated images. With our non-leaky augmentations, we get the same benefits without the downsides by executing the x-flip augmentation with 100% probability. Thus, we rely exclusively on our augmentation scheme and disable dataset x-flips to ensure that the generated images stay true to the original distribution.

F.3 Training configurations

Table 7 shows the exact set of hyperparameters that we used in our training experiments reported in Section 5. We will first detail the configurations used with CIFAR-10, FFHQ, and AFHQv2, and then discuss the training of our improved ImageNet model.

Config A of Table 2 ("Baseline") corresponds to the original setup of Song et al. [49] for the two cases (VP and VE), and config F ("Ours") corresponds to our improved setup. We trained each model until a total of 200 million images had been drawn from the training set, abbreviated as "200 Mimg" in Table 7; this corresponds to a total of ∼400,000 training iterations using a batch size of 512. We saved a snapshot of the model every 2.5 million images and reported results for the snapshot that achieved the lowest FID according to our deterministic sampler with NFE = 35 or NFE = 79, depending on the resolution.

In config B, we re-adjust the basic hyperparameters to enable faster training and obtain a more meaningful point of comparison. Specifically, we increase the parallelism from 4 to 8 GPUs and batch size from 128 to 512 or 256, depending on the resolution. We also disable gradient clipping, i.e., forcing $\|\mathrm{d}\mathcal{L}(D_\theta)/\mathrm{d}\theta\|_2 \leq 1$, that we found to provide no benefit in practice. Furthermore, we increase the learning rate from 0.0002 to 0.001 for CIFAR-10, ramping it up during the first 10 million images, and standardize the half-life of the exponential moving average of θ to 0.5 million images. Finally, we adjust the dropout probability for each dataset as shown in Table 7 via a full grid search at 1% increments. Our total training time is approximately 2 days for CIFAR-10 at 32×32 resolution and 4 days for FFHQ and AFHQv2 at 64×64 resolution.

Table 8: Details of the network architectures used in this paper.

Parameter	DDPM++ (VP)	NCSN++ (VE)	ADM (ImageNet)
Resampling filter	Box	Bilinear	Box
Noise embedding	Positional	Fourier	Positional
Skip connections in encoder	–	Residual	–
Skip connections in decoder	–	–	–
Residual blocks per resolution	4	4	3
Attention resolutions	{16}	{16}	{32, 16, 8}
Attention heads	1	1	6-9-12
Attention blocks in encoder	4	4	9
Attention blocks in decoder	2	2	13

In config C, we improve the expressive power of the model by removing the 4×4 layers and doubling the capacity of the 16×16 layers instead; we found the former to mainly contribute to overfitting, whereas the latter were critical for obtaining high-quality results. The original models of Song et al. [49] employ 128 channels at 64×64 (where applicable) and 32×32, and 256 channels at 16×16, 8×8, and 4×4. We change these numbers to 128 channels at 64×64 (where applicable), and 256 channels at 32×32, 16×16, and 8×8. We abbreviate these counts in Table 7 as multiples of 128, listed from the highest resolution to the lowest. In practice, this rebalancing reduces the total number of trainable parameters slightly, resulting in ∼56 million parameters for each model at 32×32 resolution and ∼62 million parameters at 64×64 resolution.

In config D, we replace the original preconditioning with our improved formulas ("Network and preconditioning" section in Table 1). In config E, we do the same for the noise distribution and loss weighting ("Training" section in Table 1). Finally, in config F, we enable augmentation regularization as discussed in Appendix F.2. The other hyperparameters remain the same as in config C.

With ImageNet-64, it is necessary to train considerably longer compared to the other datasets in order to reach state-of-the-art results. To reduce the training time, we employed 32 NVIDIA Ampere GPUs (4 nodes) with a batch size of 4096 (128 per GPU) and utilized the high-performance Tensor Cores via mixed-precision FP16/FP32 training. In practice, we store the trainable parameters as FP32 but cast them to FP16 when evaluating F_θ, except for the embedding and self-attention layers, where we found the limited exponent range of FP16 to occasionally lead to stability issues. We trained the model for two weeks, corresponding to ∼2500 million images drawn from the training set and ∼600,000 training iterations, using learning rate 0.0001, exponential moving average of 50 million images, and the same model architecture and dropout probability as Dhariwal and Nichol [9]. We did not find overfitting to be a concern, and thus chose to not employ augmentation regularization.

F.4 Network architectures

As a result of our training improvements, the VP and VE cases become otherwise identical in config F except for the network architecture; VP employs the DDPM++ architecture while VE employs NCSN++, both of which were originally proposed by Song et al. [49]. These architectures correspond to relatively straightforward variations of the same U-net backbone with three differences, as illustrated in Table 8. First, DDPM++ employs box filter $[1, 1]$ for the upsampling and downsampling layers whereas NCSN++ employs bilinear filter $[1, 3, 3, 1]$. Second, DDPM++ inherits its positional encoding scheme for the noise level directly from DDPM [16] whereas NCSN++ replaces it with random Fourier features [52]. Third, NCSN++ incorporates additional residual skip connections from the input image to each block in the encoder, as explained in Appendix H of [49] ("progressive growing architectures").

For class conditioning and augmentation regularization, we extend the original DDPM++ and NCSN++ arhictectures by introducing two optional conditioning inputs alongside the noise level input. We represent class labels as one-hot encoded vectors that we first scale by $\sqrt{C}$, where C is the total number of classes, and then feed through a fully-connected layer. For the augmentation parameters, we feed the conditioning inputs of Appendix F.2 through a fully-connected layer as-is.

We then combine the resulting feature vectors with the original noise level conditioning vector through elementwise addition.

For class-conditional ImageNet-64, we use the ADM architecture of Dhariwal and Nichol [9] with no changes. The model has a total of ∼296 million trainable parameters. As detailed in Tables 7 and 8, the most notable differences to DDPM++ include the use of a slightly shallower model (3 residual blocks per resolution instead of 4) with considerably more channels (e.g., 768 in the lowest resolution instead of 256), more self-attention layers interspersed throughout the network (22 instead of 6), and the use of multi-head attention (e.g., 12 heads in the lowest resolution). We feel that the precise impact of architectural choices remains an interesting question for future work.

F.5 Licenses

Datasets:

- CIFAR-10 [29]: MIT license
- FFHQ [27]: Creative Commons BY-NC-SA 4.0 license
- AFHQv2 [7]: Creative Commons BY-NC 4.0 license
- ImageNet [8]: The license status is unclear

Pre-trained models:

- CIFAR-10 models by Song et al. [49]: Apache V2.0 license
- ImageNet-64 model by Dhariwal and Nichol [9]: MIT license
- Inception-v3 model by Szegedy et al. [51]: Apache V2.0 license