ADAPTIVE SECOND-ORDER SOLVERS FOR FAST STOCHASTIC DIFFUSION SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

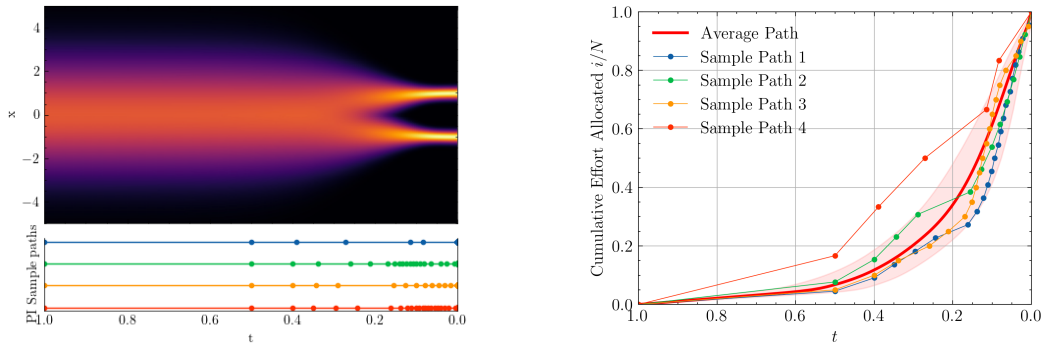
Diffusion models rely on numerical solvers requiring time-discretization, which has a large influence on the tradeoff between sampling cost and quality. However, the computational difficulty of the reverse process varies along the sampling trajectory and across data distributions, making the choice of discretization important. We adapt proportional-integral (PI) step-size control to diffusion, using our diffusion noise-normalised error estimator. Unlike existing adaptive methods in diffusion that respond only to the current error, the PI solver also incorporates the previous error, yielding smoother step adaptation. We further show that these per-sample trajectories exhibit shared structure and can be aggregated into a fixed schedule that retains much of the benefit of adaptive sampling. We evaluate both approaches on natural-image and language datasets, in terms of quality, measured by FID at a matched number of neural network evaluations (NFE), comparing them with widely used stochastic solvers and schedules. For images, our average discretization outperforms the commonly used EDM schedule in terms of sample quality when used with the stochastic Heun sampler, and with the EDM-churn sampler at low NFE. Additionally, our PI adaptive solver obtains better FID than most stochastic and adaptive baselines, although it does not beat the EDM-churn sampler at low NFE. Moreover, we find our solver outperforms both the EDM and the entropy schedule on language diffusion at low-to-medium NFE in terms of perplexity, with the drawback of lower token entropy. Lastly, we find that the benefit of per-sample adaptivity is problem-dependent. It is highly beneficial in 1D toy examples, while only marginal for image and language data, where the average schedule sometimes even outperforms the per-sample adaptive solver. Code is available at https://anonymous.4open.science/r/adaptive-generative-diffusion-solvers-4EE5/.

1 INTRODUCTION

Diffusion models generate high-quality images (Song et al., 2021; Karras et al., 2022; Song et al., 2022; Rombach et al., 2022), videos (Singer et al., 2022), and audio (Kong et al., 2021). However, diffusion sampling is slow, as it is a process requiring a large number of function evaluations (NFE) of a neural network (Song et al., 2021; Karras et al., 2022; Jolicoeur-Martineau et al., 2021). Song et al. (2021) show that the process of diffusion can be treated as solving either an ordinary or stochastic differential equation (ODE/SDE). Recent work (Li & Chen, 2024) identified critical windows within the integration interval during which many of the generative decisions are made, a phenomenon arising from symmetry-breaking bifurcations in the reverse denoising process (Raya & Ambrogioni, 2023; Ambrogioni, 2025). This suggests that numerical effort should not be distributed uniformly across the diffusion interval, but rather, more effort should be allocated to the parts of the interval where dynamics are difficult (Figure 1a), which could improve the NFE-quality tradeoff.

Many widely used schedules are handcrafted and calibrated to specific modalities (Song et al., 2021; Karras et al., 2022; Song et al., 2022; Ho et al., 2020). Recent work by Stancevic et al. (2025) discussed constructing these schedules in a data-driven manner by enforcing a linear progression in entropy. However, the authors find that their schedules are effective only for first-order samplers, with performance degrading when used with second-order methods. This limits the applicability of such schedules, as second-order samplers have been shown to improve the NFE–quality trade-off in diffusion models (Karras et al., 2022).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) (b)

Figure 1: The difficulty of generative diffusion differs across time, requiring different step-sizes at different time-points, which our PI solver determines (a). These per-sample paths share structure and can then be aggregated into an average path usable for sampling (b).

Thus, we require a method of generating schedules in a data-driven manner that also functions for second-order samplers. Numerical analysis literature offers an alternative that satisfies both conditions: adaptive solvers (Lamba & Seaman, 2006; Lehn et al., 2002; Kelly & Lord, 2022; Ilie et al., 2015; Mauthner, 1998; Hairer et al., 1993), which adapt step-size based on the local difficulty, allowing smaller step-sizes along more difficult regions. Additionally, these are generally agnostic to solver-order, allowing them to generate second-order schedules. This allows the schedule to differ on a per-sample basis, and removes the need for handcrafting. Application of these adaptive solvers to diffusion remains limited, though not unexplored (Karras et al., 2022; Jolicoeur-Martineau et al., 2021). Jolicoeur-Martineau et al. (2021) devise an adaptive SDE solver specifically designed for diffusion, adapting step-size based on the current local-error.

We instead adapt stochastic proportional-integral (PI) control for step-size adaptation (Ilie et al., 2015), which incorporates the previous local error alongside the current one, giving smoother step-size adaptation than integral-only control (Gustafsson et al., 1988). We adapt its error estimator to normalise by diffusion noise, substantially improving its performance in diffusion sampling. We compare our method to existing diffusion sampler-schedule combinations in terms of NFE-quality tradeoff. Additionally, we show that the per-sample trajectories have shared structure, which can be exploited by aggregating the trajectories into a single static schedule, usable as any other handcrafted schedule (an example of this is shown in Figure 1b). We evaluate its performance and compare it with the per-sample adaptive solver to determine if per-sample adaptivity is beneficial.

2 Related Works

Diffusion Samplers Many improvements to diffusion sampling without retraining have been made since diffusion models were introduced (Sohl-Dickstein et al., 2015; Ho et al., 2020). Song et al. (2021) introduce the Predictor-Corrector sampler, alternating predictor and corrector steps. Zhao et al. (2023) devise a unified predictor-corrector framework called UniPC. A different line of work looks into higher-order solvers. Karras et al. (2022) propose a second-order solver using Heun's method, finding that it sits optimally at the NFE-quality tradeoff, and introduce a stochastic sampler that temporarily increases noise at each step before denoising (churn). Lu et al. (2022; 2025) develop an efficient higher-order solver, using the insight that the drift part of the DEs can be analytically solved. To summarize, extensive work has gone into improving diffusion samplers and improving the sampling technique can substantially improve sample quality at low NFE. Unlike these works, this paper focuses on time-step selection using adaptive step-size control for second-order samplers, rather than the specific numerical integration method used.

Diffusion Discretization Determining the optimal discretizations for the probability flow ODE and reverse SDE has been an active topic of research after Song et al. (2021) introduced the SDE framework and two handcrafted discretizations. Karras et al. (2022) devise a new way of parametriz-

ing the probability flow ODE and handcraft a new discretization (EDM). Besides handcrafting, there are methods that obtain discretizations in a principled way. Sabour et al. (2024) introduce Align Your Steps, a technique that is grounded in stochastic calculus, obtaining optimized discretizations based on the solver used and the dataset. Chen et al. (2024) similarly optimize the time discretization, using gradient descent. Stancevic et al. (2025) discretize time in a way that is linearly spaced in terms of entropy, which can be reliably estimated from the score matching loss. However, the performance of their schedule drops for second-order samplers. Moreover, there are techniques that aim to optimize time-discretizations in an empirical way. Xia et al. (2024) train a simple model that predicts the next time step based on a sample $\mathbf{x}_t$ and a time t. Huang et al. (2026) take a reinforcement learning approach, treating time-step scheduling as a control problem. They use this approach to determine a data-driven time discretization, which they show can be interpolated and extrapolated across different NFE. However, they only show their technique to work for first-order solvers. In contrast to these approaches, this work does not directly optimize the discretization based on a predefined objective. Instead, discretizations are determined per sample and per step, using a second-order adaptive solver with PI control, and these are aggregated into one static discretization.

Adaptive SDE Samplers A substantial amount of work in the field of numerical analysis has gone into improving SDE sampling with adaptive solvers, as many different fields benefit from this. Mauthner (1998) develop a technique that uses an embedded Stochastic Runge-Kutta method to determine the local error, using that error to control the step-size. Ilie et al. (2015) expand on this approach, introducing an adaptive solver using PI control specifically designed for multidimensional SDEs, taking inspiration from Gustafsson et al. (1988) and adding a component incorporating the previous error. This approach forms the basis of the adaptive solver we apply to generative diffusion. The closest work in diffusion literature is by Jolicoeur-Martineau et al. (2021). They devise an adaptive stochastic solver and apply it to diffusion, using the difference between the first- and second-order estimates as an error metric. Their work differs from this work in the step-size controller used, only using a component proportional to the current error, not considering previous error.

3 BACKGROUND

A diffusion process progressively adds noise to a sample, transforming an initial distribution into another distribution over time. More formally, let a diffusion process run on the time interval $[t_0, T]$. Let $\mathbf{x} \in \mathbb{R}^d$ be a sample from the data distribution $p_t(\mathbf{x})$ at time t. At $t = t_0$, $p_0(\mathbf{x})$ represents the target distribution, e.g. the distribution of all images in a dataset. At $t = T$, $p_T(\mathbf{x})$ represents a simple distribution from which can be sampled, which is generally chosen to be a Gaussian. Diffusion modeling uses two separate but related diffusion processes. The forward diffusion process transforms data from $p_0(\mathbf{x})$ to $p_T(\mathbf{x})$ and is required for training. The reverse diffusion process does the opposite and is used for inference. These two diffusion processes can be modeled by SDEs (Song et al., 2021).

The forward diffusion process progressively adds noise to a sample $\mathbf{x}$ at $t = t_0$, until the sample is fully corrupted at $t = T$ (Song et al., 2021). This process is governed by the following SDE in Itô form: $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$. With $\mathbf{f}(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ the drift function, $g(t) : \mathbb{R} \to \mathbb{R}$ the diffusion function, and $\mathbf{w}(t)$ the standard Wiener process. To perform inference, the reverse diffusion process is used. This process progressively removes noise from a sample, eventually obtaining a sample from the target distribution. The reverse diffusion process is governed by the reverse time SDE (Song et al., 2021; Anderson, 1982):

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\tilde{\mathbf{w}} \tag{1}$$

Here, $\tilde{\mathbf{w}}(t)$ is the reverse Wiener process and $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is known as the score. The score can be interpreted as a vector field that points towards the modes of the distribution. The score is generally given by a trained neural network $s_\theta(\mathbf{x}, t)$ trained via denoising score matching (Song et al., 2021).

While samples from the data distribution can be obtained with the reverse SDE (1), this process can be slow (Karras et al., 2022; Song et al., 2022). However, the process can also be modeled by an ODE, known as the probability flow ODE (Song et al., 2021):

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t \tag{2}$$

The trajectories of the ODE have the same marginal probability densities as the reverse SDE.

Numerical Integration As the reverse SDE (1) generally does not have analytical solutions, they have to be numerically approximated by discretizing continuous time into time points $t_i|_{i=0}^{N}$, and computing a time step $h_i = t_{i+1} - t_i$. This time step is used to numerically integrate the differential equation, computing $\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i + \Delta\mathbf{x}_i$, where $\Delta\mathbf{x}_i(\mathbf{x}_i, h_i)$ is based on the to-be-solved SDE and the numerical integration scheme. Several integration schemes exist, such as Euler-Maruyama (Kloeden & Platen, 1992) and stochastic Heun's method (Rümelin, 1982). Additionally, different ways of determining the discretization exist. Traditionally, this is picked by hand.

4 PI Controlled Adaptive Solver

Our PI diffusion solver uses the PI control method introduced in Ilie et al. (2015). The solver consists of an error estimator, a step controller, optional added steps on the probability flow ODE, and we discuss how the solver handles batches. The algorithm is written for an $\mathbf{x}$ that is not part of a batch, but was extended to work with batches for the experiments (details in Appendix B). For brevity, we omit the ODE steps taken from this algorithm. The full algorithm can be found in Algorithm 1.

Error Estimator We adapt step-size based on the local error, which is given by the difference between the exact and estimated solution. However, for the reverse SDE (1), no exact solution is known in most diffusion contexts. Therefore, we estimate the error instead. Ilie et al. (2015) use the extrapolated error to estimate the local error, given by the difference between the second-order estimate x'' and the first-order estimate x'. Our solver uses stochastic Heun's method (Rümelin, 1982) to determine x''. We adapt the error estimator by Ilie et al. (2015), adding a tolerance term relative to the diffusion noise, yielding better performance for variance-exploding diffusion. For more information on these changes, see Appendix B.

$$\epsilon(\mathbf{x}_i, h_i) = \sqrt{\frac{1}{d}\sum_{k=1}^{d}\left(\frac{\mathbf{x}'_{i+1,k} - \mathbf{x}''_{i+1,k}}{\tau_{abs} + \tau_{rel}\sigma(t)}\right)^2} \tag{3}$$

Where $\epsilon(\mathbf{x}_i, h_i) \in \mathbb{R}_{>0}$ is the local extrapolated error, $\tau_{abs} \in \mathbb{R}_{>0}$ a user-defined tolerance parameter which scales the error by a constant, $\tau_{rel} \in \mathbb{R}_{>0}$ the relative tolerance, which scales the error based on the data. Specifically, τ_{rel} scales by the current noise level $\sigma(t)$.

Note that Ilie et al. (2015) also use a rejection mechanism. However, we found that rejected steps generally led to a worse NFE-quality tradeoff, as compute was wasted on a rejected step. Therefore, for the final experiments, step-rejection was disabled.

Step Controller The PI solver adapts the step size based on the error. For this, the PI controller by Ilie et al. (2015) is used. The PI controller not only uses the current error, but also uses the error of the previous step. They define the PI controller as:

$$h_{i+1} = h_i \min\left(\alpha_{max}, \max\left[\alpha_{min}, \left(\frac{\alpha}{\epsilon(\mathbf{x}_i, h_i)}\right)^{k_i+k_p}\left(\frac{\epsilon(\mathbf{x}_{i-1}, h_{i-1})}{\alpha}\right)^{k_p}\right]\right) \tag{4}$$

With $0 < \alpha \leq 1$ a safety factor, generally set to 0.9. α_{min} and α_{max} are also safety factors, which set a minimum and maximum value the step-size can increase in a single iteration. k_i is the integral gain, and k_p is the proportional gain. Ilie et al. (2015) suggested two settings for k_i and k_p, namely $k_i = 0.3, k_p = 0.1$ and $k_i = 0.101$ and $k_p = 0.009$. After some testing, it was found that the setting $k_i = 0.3$ and $k_p = 0.1$ performed well (see Appendix C for more details).

This equation implicitly introduces two more hyperparameters. The first of these is the starting step size h_0. This hyperparameter is dependent on all other hyperparameters, and the values used can be found in Appendix C. The second of these being the initial previous error $\epsilon(\mathbf{x}_0, h_0)$. We set $\epsilon(\mathbf{x}_0, h_0) = \alpha$, as that cancels out the influence of the second term in the first iteration.

ODE Steps Early experiments showed that at low NFE, the solver produces grainy images. This was likely caused by noise being introduced by stochasticity that cannot be fully removed at low step counts, which Karras et al. (2022) also observe with their stochasticity injection method. They run the last few steps without stochasticity to avoid this problem. Therefore, we sample the last few steps at low noise levels using the probability flow ODE (2) with a Heun sampler and a predetermined schedule.

Algorithm 1 The PI solver altered to function well on the reverse diffusion process.

procedure PISOLVER(τ_{abs}, τ_{rel}, T, t_{ode}, t_0, h_0, $\sigma(t)$, $\mathbf{f}(\mathbf{x}, t)$, $g(t)$, $s_\theta(\mathbf{x}, t)$,$k_i = 0.3$, $k_p = 0.1$, $\alpha = 0.9$, $\alpha_{min} = 0.2$, $\alpha_{max} = 5$)
  sample $x \sim \mathcal{N}(0, \sigma(T)^2\mathbf{I})$
  $h \leftarrow -|h_0|$ ▷ Ensure steps are taken in the reverse direction
  $t \leftarrow T$, $\epsilon_0 \leftarrow \alpha$, $i \leftarrow 1$
  while $t > t_{ode}$ **do**
    sample $\Delta w \sim \mathcal{N}(0, |h|\mathbf{I})$
    $\Delta\mathbf{x}' \leftarrow [\mathbf{f}(\mathbf{x}, t) - g(t)^2 s_\theta(\mathbf{x}, t)]h + g(t)\Delta w$ ▷ Euler-Maruyama step
    $\mathbf{x}' \leftarrow \mathbf{x} + \Delta\mathbf{x}$
    $\Delta\mathbf{x}'' \leftarrow [\mathbf{f}(\mathbf{x}', t + h) - g(t + h)^2 s_\theta(\mathbf{x}', t + h)]h + g(t + h)\Delta w$ ▷ Second-order step
    $\mathbf{x}'' \leftarrow \mathbf{x} + \frac{1}{2}(\Delta\mathbf{x}' + \Delta\mathbf{x}'')$ ▷ Heun
    $\epsilon_i \leftarrow \sqrt{\frac{1}{d}\sum_{k\leftarrow 1}^{d}\left(\frac{\mathbf{x}'_{i+1,k} - \mathbf{x}''_{i+1,k}}{\tau_{abs} + \tau_{rel}\sigma(t)}\right)^2}$ ▷ Extrapolated error (3)
    $\mathbf{x} \leftarrow \mathbf{x}''$ ▷ Use second order estimate to update $\mathbf{x}$
    $t \leftarrow t + h$
    $h \leftarrow h \min\left(\alpha_{max}, \max\left[\alpha_{min}, \left(\frac{\alpha}{\epsilon_i}\right)^{k_i + k_p}\left(\frac{\epsilon_{i-1}}{\alpha}\right)^{k_p}\right]\right)$ ▷ Update step (4)
    $h \leftarrow \max(h, t_{ode} - t)$ ▷ Ensure t is never less than t_{ode}
    $i \leftarrow i + 1$
  end while
  return $\mathbf{x}$
end procedure

5 Obtaining a Static Schedule

In this section, we detail how we can combine individual length-varying trajectories into a single static trajectory. As individual discretizations differ in length, aggregating over the individual discretizations cannot be done directly. Linear interpolation was applied to solve this problem, as it can be used to take samples from each discretization at exactly the same fraction of the total steps taken. Formally, linear interpolation was applied to the time points $t_{i,j}|_{i=0}^{N_j}$ for each discretization j, using $\frac{i}{N_j}$ to have a common scale from 0 to 1 for each discretization, which indicates the total fraction of steps of the discretization have been taken at that point. This creates new, interpolated paths $\hat{t}_{i,j}|_{i=0}^{\hat{N}}$, where $\hat{N}$ is the number of steps in each interpolated path. These new paths can be averaged for each different total fraction along the discretization over each discretization, obtaining the average discretization:

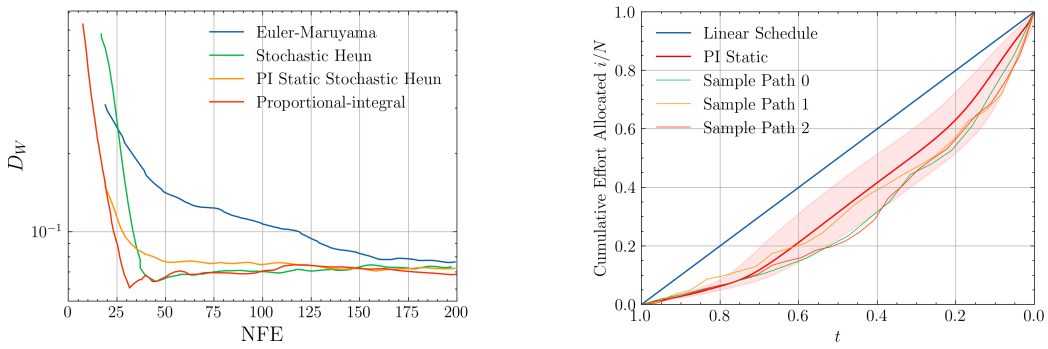
$$\bar{t}_i = \frac{1}{M}\sum_{j=1}^{M}\hat{t}_{i,j} \quad \text{for } 0 \leq i < \hat{N} \tag{5}$$

With M the total number of paths and $\bar{t}_i$ the average time at the i'th point in the discretization. The obtained discretization represents the average location of each evaluation point of the PI solver, conceptually representing the amount of effort the PI solver allocates to each time point in the integration interval. The larger the amount of time between each point on the discretization, the less effort the PI solver allocates to that interval.

6 Experiments

6.1 1D Multimodal Gaussian Diffusion

We evaluated our solver on a one-dimensional multi-modal Gaussian, comparing it to stochastic Heun's method (Rümelin, 1982) and Euler-Maruyama (Kloeden & Platen, 1992), both using equidistant steps. Additionally, we ran stochastic Heun's method with a static schedule generated by the PI solver using the method described in section 5. The experiment was performed on the variance-preserving SDE (Song et al., 2021), using an analytical expression of the score function.



(a) (b)

Figure 2: The Wasserstein distance against the NFE for different SDE solvers applied to diffusion on a mixture of Gaussians (a). The cumulative computational effort allocated ($\frac{i}{N}$) plotted against diffusion time t at $\tau_{abs} \approx 0.02656$ applied to diffusion on a Gaussian mixture (b).

We measure the Wasserstein distance at varying NFE. For the PI solver, $\tau_{rel} = 0$ was used, and τ_{abs} was used to control NFE. For details on the experimental implementation, see Appendix D, and for the hyperparameters, see Appendix C.

Figure 2a shows that the PI solver outperforms the other solvers by a large margin at low NFE. When NFE increases, the benefit over stochastic Heun's method with a linear schedule decreases. This can be explained by both samplers reaching the minimum Wasserstein distance, with PI converging more quickly. Stochastic Heun's with a PI static time schedule performs well at very low NFE ranges, but performance quickly degrades. Figure 2b shows how our solver divides computational effort over the time interval. Note that there is substantial variance in the individual allocation. This likely explains the poor performance of the stochastic Heun with PI static configuration, and suggests that per-sample adaptivity is beneficial for this problem.

6.2 FFHQ and ImageNet

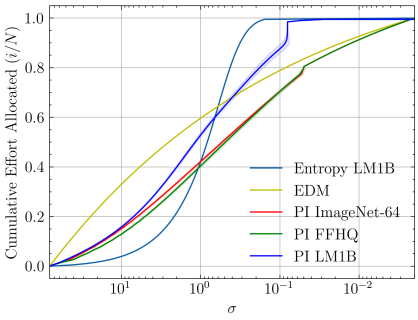
We evaluated the PI solver and the schedules it generates on FFHQ (Karras et al., 2019) and ImageNet 64x64 (Russakovsky et al., 2015). We compare it to the Euler-Maruyama (Kloeden & Platen, 1992), stochastic Heun (Rümelin, 1982), EDM-churn (Karras et al., 2022), and the Gotta Go Fast (named after their paper) (Jolicoeur-Martineau et al., 2021) solvers. Notably, for each of these solvers, we run the steps from t_{ode} onwards on the probability flow ODE. The SDE parametrization detailed in Appendix A was used. We used pretrained models from EDM and EDM2 (Karras et al., 2022; 2024b), using the unconditional VE model for FFHQ and the ImageNet-64 EDM2-XL for ImageNet-64. The FID of the generated samples is evaluated at 49, 75, and 99 NFE, taking three separate samples of 50000 images to compute the mean and standard deviation. Note that for the adaptive samplers, τ_{rel} was altered to bring the average NFE to a value that rounds to the target NFE. For details on the process of obtaining hyperparameters, refer to Appendix C, and for the experimental setup, refer to Appendix D. Due to computational constraints, we prioritized other configurations over second-order sampler runs for the entropy schedule on ImageNet, as Stancevic et al. (2025) mention the entropic schedule has deteriorated performance for second-order samplers.

The results of this experiment can be found in Table 1, with some samples found in Figure 3b, and additional samples found in Appendix E. For FFHQ, the PI static schedule clearly outperforms the EDM schedule across all NFE. This difference is less clear for ImageNet-64, although at 49 NFE, our schedule does perform better. Notably, our solver outperforms Gotta Go Fast by 2-5 times at 49 and 75 NFE. Additionally, for all the solvers except for EDM-churn, and in some cases stochastic Heun with our PI static schedule, our solver outperforms them in all cases for ImageNet-64, and some cases for FFHQ. Lastly, there is generally only a slight difference between the PI solver and stochastic Heun with a PI Static schedule, implying there is little per-sample benefit.

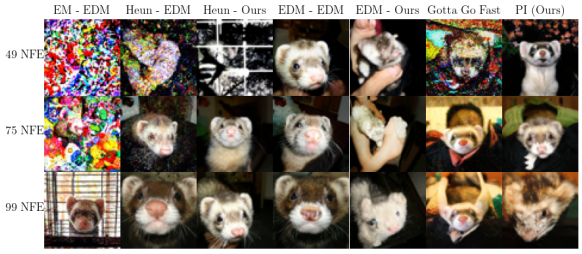
The gap between the EDM-churn solver and the pure SDE solver configurations (Euler-Maruyama, stochastic Heun, PI, Gotta Go Fast), can be explained by the sampling mechanisms. The EDM-churn

Table 1: The mean FID (↓) and standard deviation of a number of solver-schedule combinations at different NFE for ImageNet-64 and FFHQ.

Solver	Schedule	FFHQ			ImageNet 64x64		
		49 NFE	75 NFE	99 NFE	49 NFE	75 NFE	99 NFE
Euler-Marayuma	EDM	318.96±0.07	171.55±0.27	91.83±0.17	187.66±0.23	101.48±0.40	58.23±0.15
	Entropy	226.70±0.20	81.33±0.20	43.98±0.20	90.54±0.31	36.55±0.06	18.65±0.03
Stochastic Heun	EDM	274.33±0.23	23.55±0.14	3.58±0.05	167.75±0.23	20.98±0.26	2.71±0.02
	Entropy	348.81±0.15	3.26±0.02	3.54±0.05	N/A	N/A	N/A
	PI Static (ours)	137.69±0.21	3.26±0.06	3.22±0.07	66.93±1.10	**1.48**±0.02	**1.59**±0.02
EDM-churn	EDM	3.26±0.02	2.97±0.03	2.85±0.03	1.58±0.04	1.58±0.02	1.62±0.04
	Entropy	2.76±0.02	2.79±0.06	2.90±0.02	N/A	N/A	N/A
	PI Static (ours)	**2.67**±0.02	**2.66**±0.02	**2.81**±0.05	**1.49**±0.03	1.56±0.02	1.65±0.04
Gotta Go Fast	Adaptive	261.84±0.10	7.73±0.03	3.10±0.03	181.46±5.85	7.05±0.08	2.50±0.11
PI (ours)	Adaptive	83.95±0.27	3.32±0.06	3.14±0.05	37.27±0.27	**1.48**±0.00	1.61±0.01



(a)



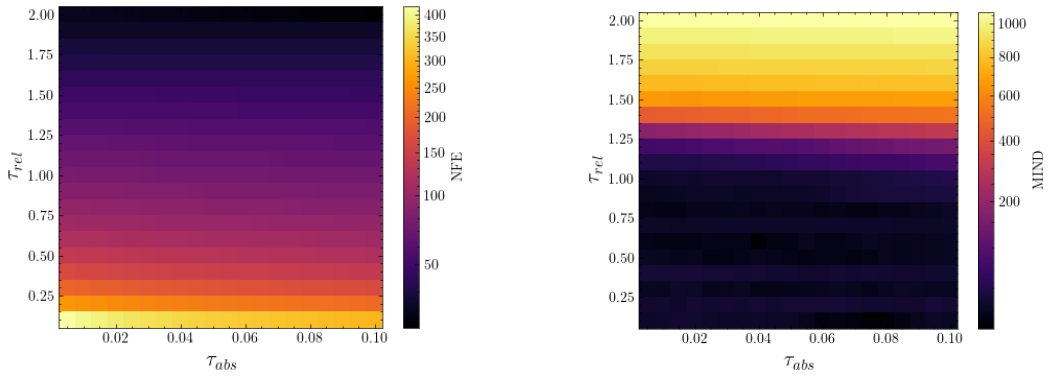
(b)

Figure 3: The cumulative effort allocation ($\frac{i}{N}$), plotted against diffusion noise σ, arrived at by the PI solver at 75 NFE for ImageNet-64 and FFHQ, and 127 NFE for LM1B, compared to the EDM schedule and LM1B entropic schedule (a). Comparison of ImageNet-64 samples generated using different solver-schedule combinations at 49, 75, and 99 NFE, with the same random seed (b).

solver samples from the probability flow ODE (2), adding noise before the denoising. This noise has different magnitude and is added every second-order step, so once per two NFE. In contrast, the SDE configurations sample from equation 1, where noise is added every step, after denoising. This introduces more total noise, causing the solvers to require more steps to fully converge to high-quality images. The stochastic Heun results confirm this, as both use a Heun's method for sampling, only differing in the way noise is added.

The average cumulative computational effort allocation of the PI static configuration for both datasets can be found in Figure 3a (refer to subsection 6.3 for comparison of the image schedules with language schedules). Firstly, it is clear that most individual discretizations lie very close to the average, as is indicated by the almost unnoticeably small error bars. Furthermore, the average discretizations deviate greatly from the EDM. Primarily, the PI discretizations take larger steps at high noise levels, while taking smaller steps than EDM at low noise levels. The low per-sample variance likely explains why there is only a slight difference between stochastic Heun with a PI static schedule and the PI solver, and only in ImageNet-64.

We performed a grid search over τ_{rel} and τ_{abs} on FFHQ, to determine the relation between those values, NFE, and image quality (see Appendix C for details). We measured image quality with the MIND metric (Berthet et al., 2026), as it is more reliable at low sample size. The grid search indicates that both sampling speed and quality are largely independent of τ_{abs} when τ_{abs} is small, supporting the claim that τ_{rel} is the primary method of selecting a point on the NFE-FID tradeoff. Additionally, MIND seems to be fairly consistent in the $\tau_{rel} < 1$ regime. This is likely caused by MIND not being sensitive enough at low sample sizes.



(a) (b)

Figure 4: The NFE (a) and the image quality measured by MIND (b) as a function of τ_{rel} and τ_{abs} measured by generating 64 ImageNet-64 images at each combination of τ_{rel}, τ_{abs}.

Table 2: The perplexity (↓) and unigram entropy (↑) of solver-schedule combination at different NFE for LM1B.

Sampler	Schedule	63 NFE		127 NFE		255 NFE	
		Perplexity	Entropy	Perplexity	Entropy	Perplexity	Entropy
DDIM	Entropic	128.26±1.62	4.35±0.00	87.03±0.28	4.34±0.01	**60.02**±0.83	4.31±0.00
	EDM	233.26±3.13	**4.36**±0.01	159.14±3.36	**4.35**±0.00	109.00±0.31	**4.34**±0.01
	PI Static (ours)	165.56±0.64	4.35±0.00	121.67±0.68	4.34±0.01	83.46±0.77	4.33±0.01
EDM-churn	Entropic	152.85±1.37	4.33±0.00	115.95±0.97	4.33±0.00	84.84±0.66	4.31±0.01
	EDM	214.76±0.29	4.33±0.01	156.30±2.05	4.33±0.01	116.49±1.59	4.32±0.01
	PI Static (ours)	121.87±0.85	4.33±0.01	95.18±0.75	4.32±0.00	69.87±0.18	4.30±0.01
PI	Adaptive	**107.04**±1.11	4.26±0.01	**83.75**±0.18	4.27±0.00	62.33±0.35	4.27±0.00

6.3 Bitstream Language Diffusion

To determine how our solver transfers between modalities, we applied the PI solver to language modeling using the Continuous Bitstream Diffusion framework (Batzolis et al., 2026), which models each token as a series of analogue bits, on which continuous diffusion is applied. We compare it to Batzolis et al. (2026)'s stochastic DDIM configurations and EDM with churn (Karras et al., 2022), both using the EDM, entropic, and PI static schedules. We use the pretrained models and evaluation pipeline provided by Batzolis et al. (2026). We measure the generative perplexity and unigram token entropy at varying NFE on the LM1B dataset (Chelba et al., 2013), generating 1024 strings of 128 tokens. For details on the hyperparameters, refer to Appendix C, and for experimental details, refer to Appendix D. The results of this experiment can be found in Table 2. Notably, our solver outperforms the EDM-churn solver in terms of perplexity at all NFE. Additionally, at 63 and 127 NFE, our solver outperforms all configurations in terms of perplexity. This improvement in perplexity does come at the cost of a lower unigram entropy, implying our samples have slightly less variance. Our PI static schedule performs better than the entropic schedule for the second-order EDM sampler, but performs worse than the entropic schedule for the first-order DDIM sampler. Additionally, it always outperforms the handcrafted EDM schedule. Interestingly, per-sample adaptivity seems to be relevant for this dataset.

Figure 3a shows the PI static schedule generated on LM1B. Note that it is substantially different from both the EDM schedule, and the schedules for image generation. It does more closely resemble the entropy schedule specifically computed for this dataset, and allocates the most effort along the center of the SDE trajectory.

Table 3: PI solver ablations on FFHQ, measuring FID (↓) of the PI solver at different NFE with varying components changed or removed.

Configuration	Components	49 NFE	75 NFE	99 NFE
Config A (base)	PI Control + Mixed Noise Tolerance	**84.09**	**3.36**	**3.11**
Config B	I Control + Mixed Noise Tolerance	125.81	4.04	3.53
Config C	PI Control + Mixed Absolute Tolerance	149.23	66.7	49.77
Config D	PI Control + Non-Mixed Tolerance	357.64	319.14	298.59

6.4 Ablations

PI Solver Ablations We ran ablations to determine the impact of the changes made to the PI solver compared to the original design by (Ilie et al., 2015) and the Gotta Go Fast solver (Jolicoeur-Martineau et al., 2021). These were run on FFHQ. The results of this can be found in Table 3. Notably, configuration A, used in the other experiments, outperforms all the other configurations. Additionally, both of the alternate tolerance options (configurations C and D) perform substantially worse than our final solver. Lastly, there is some performance gain of PI control over integral-only control, although not as large as for the other alterations. Details and hyperparameters used to obtain these ablations can be found in Appendix D.

Schedule Across Modalities To show that schedules are dataset-specific, we compare the schedules generated on FFHQ with those on LM1B. We do this comparison on both datasets. The results of this can be found in Table 4. We observe that the native schedules always obtain better performance than the foreign ones. This benefit is highlighted at the low NFE range, with the native FFHQ schedule obtaining a $3\times$ improvement over the LM1B schedule. For implementation details, refer to Appendix D.

Table 4: A PI schedule generated by the PI solver on FFHQ (images) compared to one generated on LM1B (language), evaluated in terms of FID (↓) at different NFE on FFHQ (a) and in terms of generative perplexity at different NFE on LM1B (b).

(a)

Solver	Schedule	FFHQ		
		49 NFE	75 NFE	99 NFE
Stochastic Heun	PI FFHQ (Native)	137.69	3.26	3.22
	PI LM1B (Foreign)	388.45	337.05	3.43
EDM-churn	PI FFHQ (Native)	**2.67**	**2.66**	**2.81**
	PI LM1B (Foreign)	2.83	2.83	2.98

(b)

Solver	Schedule	LM1B		
		63 NFE	127 NFE	255 NFE
Stochastic DDIM	PI LM1B (Native)	165.56	121.67	83.46
	PI FFHQ (Foreign)	184.29	125.92	87.73
EDM-churn	PI LM1B (Native)	**121.87**	**95.18**	**69.87**
	PI FFHQ (Foreign)	127.79	99.35	74.35

7 Conclusion

We adapted and applied a proportional-integral (PI) solver from the numerical analysis literature (Ilie et al., 2015) to generative diffusion, introduced an error estimator based on the diffusion noise level, and developed a technique to aggregate the PI solver's individual adaptive step trajectories into a static discretization. Our solver can be used to generate second-order compatible schedules in a data-driven manner, and these improve on the existing handcrafted EDM schedule in several of our evaluated settings.

Limitations While we did evaluate the influence of τ_{rel} and τ_{abs} on the NFE-quality tradeoff, no full search over the hyperparameter space was conducted. Moreover, all image and language experiments were conducted using VE models. The 1D experiments were done with a VP SDE and show a significant benefit of adaptivity. It is unclear if this is a property of the VP SDE or of the specific dataset. Additionally, these experiments were only conducted for stochastic sampling. Preliminary testing indicates the PI solver can also be applied to deterministic sampling, but it is left to future work to investigate if it outperforms existing sampler-schedule combinations. This could also allow the solver to discover the final ODE steps by itself, instead of requiring an existing static schedule for the last steps.

8 AI USE STATEMENT

In this work, we have used generative AI to provide feedback on the research methodology, asking it to specifically highlight potentially important additional ablations. Moreover, we used it to provide feedback on the manuscript in terms of writing style and consistency. Lastly, we used AI to help with identifying the cause of bugs in the code. Generative AI was not used for any other purposes in this paper. All of the feedback provided by LLMs was first manually checked for correctness and usefulness before applying the feedback. No AI-generated code ended up in the final codebase, so AI-generated code did not have to be checked for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

9 REPRODUCIBILITY STATEMENT

To make the paper reproducible, we provide a link to the github repository used to obtain the results of this paper in the abstract. Additionally, section 4 details the PI solver we adapted, with Appendix B providing additional detail. Furthermore, Appendix C provides detail on how the hyperparameters were obtained from scratch, with Appendix D providing the detail required to reconstruct the results of this paper without requiring access to the codebase.

REFERENCES

Luca Ambrogioni. The statistical thermodynamics of generative diffusion models: Phase transitions, symmetry breaking, and critical instability. *Entropy*, 27:291, 2025.

Brian D. O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12:313–326, 1982.

Georgios Batzolis, Mark Girolami, and Luca Ambrogioni. Cobit: Language modeling with bitstream diffusion. *arXiv preprint arXiv:2605.07013*, 2026.

Quentin Berthet, Yu-Han Wu, Clement Crepy, Romuald Elie, Klaus Greff, and Michael Eli Sander. Mind: Monge inception distance for generative models evaluation. *arXiv preprint arXiv:2605.06797*, 2026.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Yuzhu Chen, Fengxiang He, Shi Fu, Xinmei Tian, and Dacheng Tao. Adaptive time-stepping schedules for diffusion models. In *UAI*, pp. 685–697, 2024.

Kjell Gustafsson, Michael Lundh, and Gustaf Söderlind. A pi stepsize control for the numerical solution of ordinary differential equations. *BIT Numerical Mathematics*, 28:270–287, 1988.

E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I*. Springer Berlin Heidelberg, 1993.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, 2020.

Yilie Huang, Wenpin Tang, and Xunyu Zhou. Art for diffusion sampling: A reinforcement learning approach to timestep schedule. *arXiv preprint arXiv:2601.18681*, 2026.

Silvana Ilie, Kenneth R. Jackson, and Wayne H. Enright. Adaptive time-stepping for the strong numerical solution of stochastic differential equations. *Numerical Algorithms*, 68:791–812, 2015.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2019.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Advances in Neural Information Processing Systems*, 2022.

Tero Karras, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. Guiding a diffusion model with a bad version of itself. *Advances in Neural Information Processing Systems*, 2024a.

Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

Cónall Kelly and Gabriel J. Lord. Adaptive Euler methods for stochastic systems with non-globally Lipschitz coefficients. *Numerical Algorithms*, 89:721–747, 2022.

Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer Berlin Heidelberg, 1992.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. DiffWave: A Versatile Diffusion Model for Audio Synthesis. In *International Conference on Learning Representations*, 2021.

H. Lamba and T. Seaman. Mean-square stability properties of an adaptive time-stepping SDE solver. *Journal of Computational and Applied Mathematics*, 194:245–254, 2006.

J. Lehn, A. Rößler, and O. Schein. Adaptive schemes for the numerical solution of SDEs—a comparison. *Journal of Computational and Applied Mathematics*, 138:297–308, 2002.

Marvin Li and Sitan Chen. Critical windows: non-asymptotic theory for feature emergence in diffusion models. *arXiv preprint arXiv:2403.01633*, 2024.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in neural information processing systems*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 22: 730–751, 2025.

Susanne Mauthner. Step size control in the numerical solution of stochastic differential equations. *Journal of Computational and Applied Mathematics*, 100:93–109, 1998.

Gabriel Raya and Luca Ambrogioni. Spontaneous symmetry breaking in generative diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

W. Rümelin. Numerical Treatment of Stochastic Differential Equations. *SIAM Journal on Numerical Analysis*, 19:604–613, 1982.

Amirmojtaba Sabour, Sanja Fidler, and Karsten Kreis. Align your steps: Optimizing sampling schedules in diffusion models. *arXiv preprint arXiv:2404.14507*, 2024.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*, 2022.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021.

Dejan Stancevic, Florian Handke, and Luca Ambrogioni. Entropic time schedulers for generative diffusion models. In *Advances in Neural Information Processing Systems*, 2025.

Mengfei Xia, Yujun Shen, Changsong Lei, Yu Zhou, Deli Zhao, Ran Yi, Wenping Wang, and Yong-Jin Liu. Towards more accurate diffusion model acceleration with a timestep tuner. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

A Reparametrizing the EDM Probability Flow ODE

To be able to use the EDM schedule using the formulation of the reverse SDE (1), the parametrization Karras et al. (2022) provide needs to be rewritten to yield a drift and diffusion compatible with the forward SDE.

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w} \tag{6}$$

With $\mathbf{f}(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ the drift function, $g(t) : \mathbb{R} \to \mathbb{R}$ the diffusion function, and $\mathbf{w}(t)$ the standard Wiener process.

Karras et al. (2022) define the probability flow ODE:

$$\mathrm{d}\mathbf{x} = \left[\frac{\dot{s}(t)}{s(t)}\mathbf{x} - s(t)^2\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}}\log p_t\left(\frac{\mathbf{x}}{s(t)}\right)\right]\mathrm{d}t \tag{7}$$

With $s(t) \in \mathbb{R} \to \mathbb{R}$ a scaling function, $\sigma(t) \in \mathbb{R} \to \mathbb{R}$ a function indicating the noise at a time t. This is an alternate formulation of equation 2, assuming $s(t) = 1$:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}}\log p_t(\mathbf{x})]\mathrm{d}t$$

With $\nabla_{\mathbf{x}}\log p_t(\mathbf{x})$ the score. To obtain expressions for the drift and diffusion to plug in the forward SDE, one can set:

$$\mathbf{f}(\mathbf{x}, t) = \frac{\dot{s}(t)}{s(t)}\mathbf{x} \quad \text{and} \quad \frac{1}{2}g(t)^2 = s(t)^2\dot{\sigma}(t)\sigma(t) \tag{8}$$

Solving for $g(t)$ yields:

$$g(t) = s(t)\sqrt{2\dot{\sigma}(t)\sigma(t)} \tag{9}$$

Karras et al. (2022) define for the EDM framework the parameters $s(t) = 1$ and $\sigma(t) = t$. The time derivatives of these are

$$\dot{s}(t) = 0 \quad \text{and} \quad \dot{\sigma}(t) = 1$$

These can be used to obtain the drift and diffusion of the EDM parametrization:

$$\mathbf{f}(\mathbf{x}, t) = 0 \quad \text{and} \quad g(t) = \sqrt{2t} \tag{10}$$

Which can then be filled into the forward SDE to obtain the EDM SDE:

$$\mathrm{d}\mathbf{x} = \sqrt{2t}\mathrm{d}\mathbf{w} \tag{11}$$

This SDE can then be used together with the variance-exploding denoising networks provided by EDM and EDM2 (Karras et al., 2022; 2024b) by using the derived values for the drift and diffusion.

B Solver Details

B.1 Error Estimator

The error estimator given by equation 3 differs from the one defined by Ilie et al. (2015). Here, we detail how we arrived at our error estimator. Ilie et al. (2015) define the error as:

$$\epsilon(\mathbf{x}_i, h_i) = \sqrt{\frac{1}{d}\sum_{k=1}^{d}\left(\frac{\mathbf{x}'_{i+1,k} - \mathbf{x}''_{i+1,k}}{\tau_{abs}}\right)^2} \tag{12}$$

Where $\epsilon(\mathbf{x}_i, h_i) \in \mathbb{R}_{>0}$ the local extrapolated error, and $\tau_{abs} \in \mathbb{R}_{>0}$ a user-defined tolerance parameter which scales the error.

We implemented this error estimator, which performed well for the 1D experiments, which use the Variance Preserving (VP) SDE. However, when we applied this to the image datasets, for which the models were trained using the Variance Exploding (VE) SDE instead, this error estimator struggled to achieve low NFE, and low NFE results were of low quality (see Table 3).

To solve this problem, a relative tolerance parameter was introduced. This is a common parameter in ODE literature (Hairer et al., 1993), that takes into account the scale of the differential equation. This is specifically relevant when the scale of the problem varies largely, which is exactly the case in the VE formulation of diffusion. Generally, the scale of the problem is accounted for by multiplying the relative tolerance by the absolute value of each dimension. The mixed absolute tolerance is then defined as:

$$\tau(\mathbf{x}') = \tau_{abs} + \tau_{rel}|\mathbf{x}'| \tag{13}$$

Where the absolute value of each element in $\mathbf{x}$ is calculated elementwise, and $\tau_{abs} \in \mathbb{R}_{>0}$ is the absolute tolerance, which is equivalent to the tolerance in equation 3. τ_{rel} is the relative tolerance.

In the diffusion process, the noise at each time point is a known quantity, which also indicates the scale of each dimension. Therefore, it was hypothesised that instead of normalizing each dimension by its absolute value, we could normalise by the noise level. This yields the equation for the mixed noise tolerance:

$$\tau(t) = \tau_{abs} + \tau_{rel}\sigma(t) \tag{14}$$

Where $\sigma(t) \in \mathbb{R}_{>0}$ indicates the noise at time t. Ablations showed that the mixed noise tolerance generally required less NFE for similar sample quality compared to the absolute noise tolerance and the non-mixed tolerance. Therefore, the mixed noise tolerance is used in all experiments.

B.2 Step Rejection

Ilie et al. (2015) also define a step-rejection mechanism in their paper. They define that the error should satisfy the following condition:

$$\epsilon(\mathbf{x}_i, h_i) \leq 1 \tag{15}$$

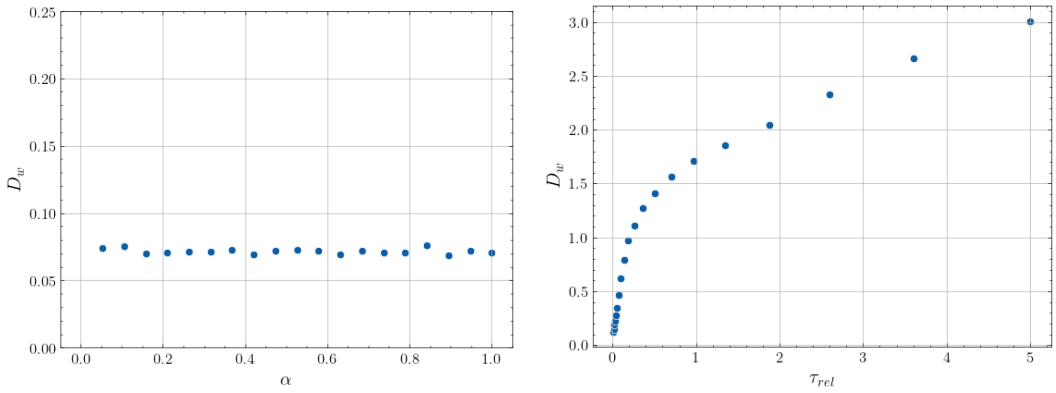
If this condition is not satisfied, step h_i is rejected, and a new step h_{i+1} is tried. As the tolerance directly scales the error, setting a smaller tolerance effectively sets the error threshold higher, making steps more likely to be rejected, while a larger tolerance has the opposite effect; it makes the solver more permissive and less likely to reject steps.

However, we found that rejected steps generally led to wasted NFE. The best results were obtained when the rejection rate was 0, and disabling the rejection mechanism led to the same or better quality results. Furthermore, for the 1D experiments, disabling step rejection improved the results. Therefore, we decided to disable step rejection for the final experiments.

B.3 Batches

To efficiently make use of the GPU, most diffusion implementations apply the reverse diffusion process in batches. As the solver is intended to have different step-sizes for each element of the batch, certain elements of the batch can finish sampling ($t = t_0$) before others. This is handled by each element in the batch having its own independent time. To ensure that these elements are not unnecessarily passed into the score network, masking was used to remove finished elements from the batch, which was inspired by the approach of Jolicoeur-Martineau et al. (2021).

C HYPERPARAMETER SELECTION



(a) Varying α

(b) Varying τ_{rel}

(c) Varying k_i

(d) Varying k_p

Figure 5: The 1-dimensional sensitivity of the α (a), τ_{rel} (b), k_i (c), and k_p (d) hyperparameters of the PI solver with regards to the minimum Wasserstein distance of the Wasserstein distance - NFE tradeoff, evaluated on a 1D multimodal Gaussian with varying means, variances, and weights.

C.1 1D VALIDATION

In this subsection, we discuss how we determined the hyperparameters we started from in subsequent experiments, and assess influence of hyperparameters on the NFE-quality tradeoff. To assess this influence, each of the hyperparameters α, τ_{rel}, h_0, k_p, and k_i were varied, while keeping the other hyperparameters constant. The setup described in Appendix D was used for this. Note that no grid search over these hyperparameters was performed, instead we performed a 1D sensitivity analysis. With the variation of τ_{abs} in addition to the other hyperparameters, doing a full evaluation over these hyperparameters would have a time complexity of $\mathcal{O}(n^6)$. With a decent resolution for each hyperparameter, this grid search was assessed to be infeasible. However, the one-dimensional sensitivity of each hyperparameter still gives information about the stability of the PI solver. The minimum Wasserstein Distance as a function of either α, τ_{rel}, k_p, and k_i, on the 1D multimodal Gaussian with varying means, variances, and weights can be seen in Figure 5. For details on the default settings for all hyperparameters that were not varied, and for details on the Gaussian, refer to Appendix D. Note that h_0 is absent here, as its minimum error plot did not reflect the NFE - Wasserstein distance tradeoff as well as the others. This will be further discussed later on in this subsection.

It can be seen in Figure 5 that the solver is insensitive to α. The Wasserstein distance barely varies between different α. The main thing that α influences is that NFE becomes more sensitive to τ_{abs} at lower α. However, the Wasserstein Distance - NFE tradeoff remains the same, implying α does not influence the tradeoff. There is some influence of τ_{rel}. Mainly, larger values of τ_{rel} increase the

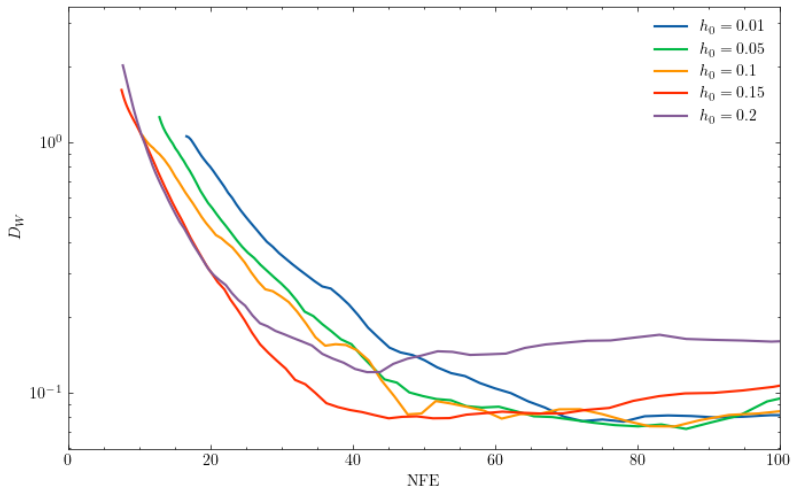


Figure 6: Influence of varying h_0 on the NFE - Wasserstein distance trade-off of the PI solver on a 1D multimodal Gaussian with varying means, variances, and weights.

Wasserstein distance. This is likely caused by the scale of the variance preserving SDE not varying enough to get good use of scaling the error by the noise level, and therefore, τ_{rel} primarily acts as a direct addition to τ_{abs}, increasing the lowest Wasserstein Distance purely by having the tolerance be consistently larger. Therefore, for this 1D experiment, τ_{rel} is best kept at 0. The controller seems fairly insensitive to k_i. Mainly, very low k_i ($k_i < 0.05$) seems to be detrimental for performance. Larger values do not have much influence. Finally, the controller seems to be insensitive to k_p. Therefore, the literature values of $k_i = 0.3$, $k_p = 0.1$, $\alpha = 0.9$ seem to be decent settings for the PI solver on 1D Gaussians. Furthermore, τ_{rel} should be kept at 0 for this experiment.

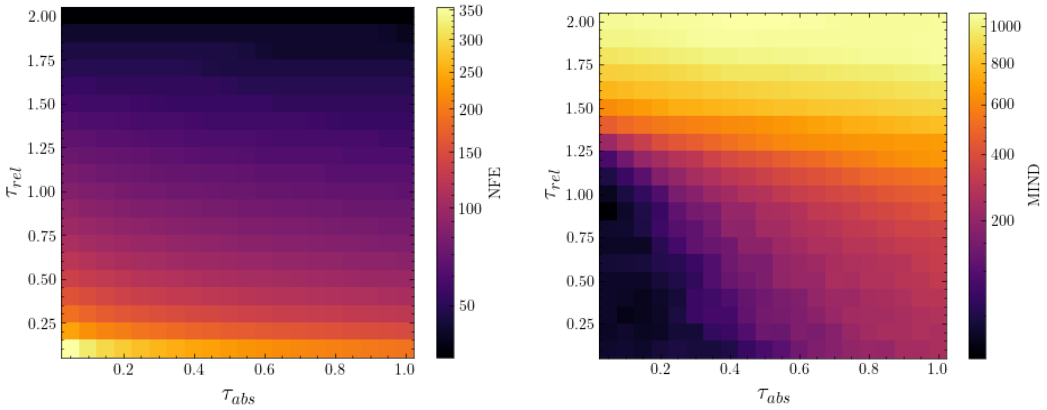
The influence of h_0 on the NFE - Wasserstein distance tradeoff cannot be easily reflected in plots just showing the minimum Wasserstein distance in the entire tradeoff. Mainly, choosing h_0 correctly has a large influence on how many steps the solver needs to converge to its optimal Wasserstein distance. To show this influence, consider Figure 6. In the figure, it can be seen that at low h_0, the Wasserstein distance reduces more slowly as a function of NFE, compared to at higher h_0. However, if h_0 is too large, which is the case at $h_0 = 0.2$, the minimum Wasserstein distance the solver reaches is larger than it can be if h_0 is chosen more optimally. Additionally, it can be seen that with $h_0 = 0.15$ the Wasserstein distance reduces quickly with NFE, while performance stabilizes at the same Wasserstein distance as the smaller h_0's. Therefore, $h_0 = 0.15$ was chosen for the 1D experiments. Lastly, note that the influence of h_0 on the tradeoff is significant. It is therefore an important parameter to determine a good value for.

C.2 Image Experiments

C.2.1 PI Solver

After concluding testing of the PI solver on 1D multimodal Gaussians, it was applied to ImageNet-64. Initially, all hyperparameters were transferred from the 1D experiment to the ImageNet-64 experiment. However, it was quickly apparent that τ_{abs}, τ_{rel}, and h_0 needed significant tweaking. The solver did work with the initial hyperparameters. It just took 750 NFE to generate an image. The main goal of hyperparameter optimization was thus to bring NFE down to a number more reasonable for modern diffusion, between 50 and 100.

An initial grid search over τ_{abs} and τ_{rel} was performed, keeping $h_0 = 0.15$, $\alpha = 0.9$, $k_i = 0.3$, and $k_p = 0.1$. At each point in the grid, 64 samples were generated using the experiment setup described in Appendix D. For these 64 samples, the average NFE was recorded, and MIND (Berthet et al., 2026) was measured as an error metric, as it is more accurate than FID at low sample size. This was done on a 20×20 grid. Step rejection was also enabled, via the method described in Appendix B. The initial grid was chosen with $\tau_{abs} \in [0.05, 1]$ and $\tau_{rel} \in [0.1, 2]$, with points linearly spaced.



(a) NFE as a function of τ_{abs}, τ_{rel}.

(b) MIND as a function of τ_{abs}, τ_{rel}.

Figure 7: The results of the initial grid-search over the absolute tolerance τ_{abs} and relative tolerance τ_{rel} of the PI solver, measuring NFE and the MIND error metric.

This grid was chosen as it spans a decently large space of the hyperparameters, and each of the four corners of the grid were stable and produced images of decent quality, with each corner of the grid converging with NFE less than 1000. The results of the grid search can be found in Figure 7.

In Figure 7, it can be seen that when $\tau_{abs} + \tau_{rel}$ are sufficiently large together, image quality quickly degrades, although NFE reduces as well when this becomes larger. Additionally, as τ_{abs} increases, NFE goes down slightly, but this comes at the cost of an increase in MIND. Only at the low range of τ_{abs} does MIND become low. MIND does not go much lower than 50, but this is likely caused by the sample size being too small to yield MIND values that accurately represent the sample quality when the sample quality is sufficient. As NFE seemed to increase less while allowing larger τ_{rel} and therefore less NFE at lower τ_{abs}, a second grid search was performed with $\tau_{abs} \in [0.005, 0.1]$ and $\tau_{rel} \in [0.1, 2]$, of which the results are presented in Figure 4. In this figure, τ_{abs} seems to have very little impact on the NFE-MIND tradeoff, although lower seems slightly better. Therefore, we chose $\tau_{abs} = 0.005$.

With $\tau_{abs} = 0.005$, the main knob for changing NFE is the combination of h_0 and τ_{rel}. h_0 is tuned in tandem with τ_{rel}. The value used for h_0 was empirically determined after selecting the other hyperparameters, picking the largest value of h_0 that led to results of high quality. Choosing h_0 to high generally led to a sharp decline in quality. Tuning τ_{rel} was done using manual binary search, with the goal of setting the solver at a specific NFE, so it can be easily compared with the other solvers. An initial value of τ_{rel} was picked based on the tolerance grid plot Figure 4. From there, the relative tolerance combined with h_0 was tuned until the desired NFE was reached by increasing τ_{rel} if NFE was too large, while decreasing if NFE was too low. An average NFE that rounds to the desired NFE when rounded to the nearest integer was taken to be sufficient.

Lastly, t_{ode} and H_{ode} need to be determined. These values are based on similar ones used by EDM (Karras et al., 2022). In their stochastic sampler, they found that running purely on the probability flow ODE from $t = 0.05$ was optimal for ImageNet-64. Therefore, t_{ode} was set to 0.05. ODE steps were taken using the EDM schedule equation 20, as that is generally regarded as a good schedule for image data. The number of ODE steps taken was based on the percentage of steps the EDM schedule would take from $t = 0.05$ until convergence. This was found to be approximately 20% of total steps. Therefore, the NFE used by the Heun sampler using the EDM discretization was set to 20% of the total NFE. As Heun steps take 2 NFE each, this means that 49 NFE gets 5 ODE steps, 75 NFE 7.5 (which is rounded down to 7), and 99 NFE 10. Please note that these NFE were considered in the total NFE measured for the PI solver.

C.2.2 Gotta Go Fast Solver

The Gotta Go Fast solver by Jolicoeur-Martineau et al. (2021) did not work out of the box on ImageNet-64 using the EDM SDE equation 11 and checkpoint. Therefore, the hyperparameters of

this solver needed to be tuned as well. This search started with using the default parameters used in the paper, $r = 0.9$, $\alpha = 0.9$ (this parameter is named θ in their paper, but corresponds to α of the PI solver), $\tau_{abs} = 0.0078$ (named ϵ_{abs} in the paper, but corresponds to τ_{abs}, $\tau_{rel} = 0.5$ (named ϵ_{rel} in the paper, but corresponds to τ_{rel}), and $h_0 = 0.01$. These parameters showed significant instability, causing large oscillations in step size and local error.

The oscillatory behavior seemed to largely be caused by too much sensitivity to the error for step size control. As step size is determined by $h = \min(t, \alpha h \epsilon^{-r})$, with ϵ the local error, r was tweaked to a much smaller value (0.1). This instantly improved stability, and increasing r, even slightly, reintroduced some instability. However, the controller was still slightly unstable. By tweaking the other hyperparameters systematically to find which had the greatest impact on instability, it was found that the controller was quite sensitive to α as well. Lowering it slightly to 0.7 improved stability. τ_{abs} was kept unaltered, as it seemed to be working mostly fine. The same methods as used to determine τ_{rel} and h_0 as used for the PI solver were applied to Gotta Go Fast solver. Note that the τ_{rel} required on this dataset to get low NFE was significantly larger than that of the original paper.

Some tweaks were made to the solver to improve stability and quality of the solver samples. The solver was adapted to also take ODE steps at the end of the trajectory, as that puts the solver on more even ground with the PI solver.

C.3 Bitstream Diffusion

The method we used to determine the hyperparameters for the PI solver was similar to the one we used for the image experiments. We took the hyperparameters from the image experiments as a baseline and went from there. When tuning the hyperparameters, we noticed the solver sampled at very low NFE for the τ_{rel} values used in the image experiments. After more tuning, we found there to be a sort of tipping point. At low values of τ_{rel}, the solver would quickly start needing an infinite number of steps to converge, and depending on the value of τ_{rel}, this point occurred at a certain t. In the end, τ_{rel} could still be tuned accurately enough to have the NFE be rounded to the target NFE.

D Experiment Details

The solvers were implemented in a new codebase, taking inspiration from the implementations by Song et al. (2021) and Karras et al. (2022). Addionally, the code by Batzolis et al. (2026) was integrated into the codebase, and extended to allow use of the PI solver. All experiments were performed on a cluster with 10 Quadro RTX 6000 GPUs using PyTorch.

D.1 1D Multimodal Gaussian Diffusion

D.1.1 Experiment Setup

The 1D experiment was executed on a mixture of Gaussians with varying weights, means, and variance, given by:

$$P(x) \sim \frac{1}{\sum_{j=1} \pi_j} \sum_{i=1}^{N} \pi_i \mathcal{N}(\mu_i, \sigma_i^2) \tag{16}$$

With π_i being the weight of the i'th Gaussian, μ_i being the mean of the i'th Gaussian, and σ_i being the standard deviation of the i'th Gaussian. The specific parameters used for each Gaussian can be found in Table 5. Note that N was set to 10. The choices made were semi-random. The standard deviations were chosen such that both shallow and sharp peaks were found in the distribution. The means were chosen such that some parts of the distribution blend slightly, while some peaks have more space in between them. Finally, it was ensured that N modes were found in the distribution, thus, no modes were fully blended.

An analytical expression for the score was used. A small value of 10^{-13} was added to the score, as otherwise, score values could get too small in areas outside of the distribution, causing them to become NaN values.

Table 5: Parametrization of the varying Gaussian.

	1	2	3	4	5	6	7	8	9	10
π	1	3	1	3	1	2	1	3	2	1
μ	-12	-10	-8	-5	0	4	9	13	15	17
σ	0.5	1.5	0.2	2	0.3	1.5	0.1	1	0.2	0.1

Table 6: Hyperparameters used for the PI solver for 1D Gaussians using the VP SDE.

Parameter	Value
τ_r	0
α	0.9
α_{min}	0.2
α_{max}	5
h_0	0.15
t_{ode}	0
N	0
k_i	0.3
k_p	0.1

The experiment was run using the variance-preserving SDE, which Song et al. (2021) define as:

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w} \tag{17}$$

Where we set $\beta(t)$ to the following linear function:

$$\beta(t) = \beta_{min} + t(\beta_{max} - \beta_{min}) \tag{18}$$

With $\beta_{min} = 0.1$ and $\beta_{max} = 20$, as is the default setting for the VP SDE (Song et al., 2021).

The Euler-Maruyama and Heun solvers used fully linear discretizations. The PI static Heun solver uses a discretization generated by a PI solver. The same PI discretization is used for each NFE, chosen as the discretization at the PI adaptive solver's lowest point, set by changing τ_{abs}. τ_{abs} was set to 0.02656. Synthetic data were created by taking 100000 samples from the multimodal Gaussian. Then, we sampled from the distribution obtained from equation 17 at $t = 1$, yielding a sample distribution very close to $\mathcal{N}(0, 1)$. These were then passed into the solvers for each different number of steps (for non-adaptive solvers) or tolerance (for adaptive solvers). After obtaining the estimate from a solver, the Wasserstein distance was computed with respect to the samples before noise was added. This was recorded for each solver at each different number of steps or tolerance.

D.1.2 Hyperparameters

The Euler-Maruyama and Heun configurations were evaluated from 10 to 200 NFE with a resolution of 100 points. The PI solver was evaluated from $\tau_{abs} = 0.001$ to $\tau_{abs} = 1$. Note that the tolerance was exponentially spaced to give approximately equal resolution to values below $\tau_{abs} = 1$ and above $\tau_{abs} = 1$, as otherwise, the experiment would be quite skewed toward higher tolerance. The full hyperparameters used for the PI solver can be found in Table 6.

D.2 Image Experiments

D.2.1 Image Generation

The image generation process is heavily based on the implementation of EDM2 (Karras et al., 2024b;a), but adapted to work with various solvers. The checkpoint edm2-img64-xl-0671088-0.040.pkl was used for the ImageNet-64 experiments, while the edm-ffhq-64x64-uncond-ve.pkl was used for FFHQ. These checkpoints are denoising networks that predict the denoised image at $t = 0$. This is not equivalent to a score network, which is required for equation 1 to be applied. Therefore,

the denoiser output $D(\mathbf{x}, t)$ is transformed to the score using (Karras et al., 2022):

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = \frac{D(\mathbf{x}, t) - \mathbf{x}}{t^2} \tag{19}$$

This can then be directly input into the reverse time SDE.

We used a batch size of 512. For each solver-NFE combination, the same random seed was used. Every new sample increments this random seed by one, changing the starting noise of that image and changing the noise added when solving the SDE. Conditional generation was done, and each sample used a label randomly generated with that sample's seed. This means that each solver generates images with the exact same starting noise, label, and adds the same random noise.

For the Euler-Maruyama, stochastic Heun (EDM discretization and PI Static discretization), Gotta Go Fast, and PI solver, the drift and diffusion of equation 11 were used. The EDM solver does not directly sample from equation 1. Instead, it uses random churn added at each step. This was not expressed in the form of an SDE. The sampler implemented in the EDM2 repository (Karras et al., 2024b;a)[1] was directly used, only adapted to allow different discretizations to be passed into the solver. The PI Static EDM condition also uses the EDM sampler, but with a discretization generated by the PI solver.

We retrieved the entropy schedule from the github repository of the paper by Stancevic et al. (2025), matching the entropic schedule to the dataset. We used the interpolation method found in their repository to downsample the 128-step entropy schedule to the correct number of steps taken.

D.2.2 Metric Calculation

Each metric was calculated using 50000 generated images with all real images as a reference. We generated 3 samples of 50000 images at different seeds to allow for expressing uncertainty. No post-processing, except for the decoder that decodes the image to RGB space, was applied to the generated samples. The pipeline for calculating metrics was based on the implementation of EDM2 (Karras et al., 2024b;a). The pipeline was verified to work by comparing the FID results with those of the EDM2 pipeline, which yielded identical results.

NFE for adaptive solvers was computed by, at every time the reverse SDE was evaluated, incrementing total NFE by the batch size. This ensures that the second-order steps actually count for two NFE, and that elements that are already finished in an adaptive solver, which are then removed from the batch by masking, do not count for NFE. Finally, the total NFE was divided by the total number of samples to obtain the NFE per sample. For the non-adaptive solvers, NFE was set by setting the number of evaluation steps. For second-order solvers, this was determined using $n = \lfloor \frac{\text{NFE}+1}{2} \rfloor$, which ensures that the number of steps set matches the number of score evaluations done. Specifically, the second-order solvers execute $n - 1$ second-order steps, and then one first-order step. This does imply that the second-order solvers cannot be evaluated at an even number of NFE, hence, all NFE chosen are odd.

D.2.3 Hyperparameters

We used the EDM schedule (Karras et al., 2022) as a baseline schedule, which is given by:

$$t_i = \left(\sigma_{max}^{\frac{1}{\rho}} + \frac{i}{N-1} \left[\sigma_{min}^{\frac{1}{\rho}} - \sigma_{max}^{\frac{1}{\rho}} \right] \right)^{\rho} \tag{20}$$

Where σ_{max} indicates the highest noise level, and σ_{min} the lowest noise level. We set $\rho = 7$, $\sigma_{max} = 80$, and $\sigma_{min} = 0.002$, as are the hyperparameters used by (Karras et al., 2022).

For both stochastic Heun and Euler-Maruyama, ODE steps were taken at the end of the trajectory from $t_{ode} = 0.05$. These ODE steps were taken by switching the SDE to ODE when $t < t_{ode}$. For the EDM configuration, the same discretization as for EM was used (the one given by Equation 20). The sampler used is the one devised by Karras et al. (2022). To introduce stochasticity, the churn mechanism of the solver was used, with $S_{churn} = 40$, $S_{tmin} = 0.05$, $S_{tmax} = 50$, and $S_{noise} = 1.003$, which are the optimal settings found by Karras et al. (2022) for ImageNet-64. These were

[1] Repository found here

Table 7: Hyperparameters used for image generation of the PI solver, and the adaptive solver devised by Jolicoeur-Martineau et al. (2021) (Gotta Go Fast)

Parameter	PI solver			Gotta Go Fast		
	49 NFE	75 NFE	99 NFE	49 NFE	75 NFE	99 NFE
τ_{abs}	0.005	0.005	0.005	0.0078	0.0078	0.0078
$\tau_{rel,\text{ImageNet-64}}$	1.09	0.87	0.72	25.5	17	13.7
$\tau_{rel,\text{FFHQ}}$	1.15	0.92	0.72	25.4	17.15	13.7
α	0.9	0.9	0.9	0.7	0.7	0.7
α_{min}	0.2	0.2	0.2	N/A	N/A	N/A
α_{max}	5	5	5	N/A	N/A	N/A
$h_{0,\text{ImageNet-64}}$	45	40	35	25	25	20
$h_{0,\text{FFHQ}}$	40	35	35	25	25	20
t_{ode}	0.05	0.05	0.05	0.05	0.05	0.05
N	5	7	10	5	7	10
k_i	0.3	0.3	0.3	N/A	N/A	N/A
k_p	0.1	0.1	0.1	N/A	N/A	N/A
r	N/A	N/A	N/A	0.1	0.1	0.1

also used for FFHQ. Whenever the PI discretization is used, we match the NFE from the run used to generate the discretization with the run using the discretization. The hyperparameters used for the PI solver and for the solver by Jolicoeur-Martineau et al. (2021) can be found in Table 7. Note that for FFHQ and ImageNet-64, mostly the same hyperparameters were used. Only τ_{rel} and h_0 were slightly altered between the datasets. How these hyperparameters were obtained is explained in Appendix C.

D.3 Bitstream Diffusion

D.3.1 Experiment Details

The Bitstream diffusion experiments were executed using the exact framework introduced by Batzolis et al. (2026), using their repository[2]. We did not change anything already existing in their repository, only extended it to be able to use our solver. We implemented an interface between their repository and ours to allow the code in their repository to use our PI solver. To allow the solver to produce high-quality samples, we also added support for self-conditioning to our solver. Note that no other alterations to the solver were necessary for functioning on this dataset. We sample from the reverse SDE that Batzolis et al. (2026) introduce, which is approximately equal to how EDM samplers add churn. This SDE is given by:

$$\mathrm{d}\mathbf{x} = (1 + \lambda(t))t\nabla_x \log_t(\mathbf{x})\mathrm{d}t + \sqrt{2\lambda(t)t}\mathrm{d}\tilde{\mathbf{w}} \tag{21}$$

With $\lambda(t) : \mathbb{R} \to \mathbb{R}_{\geq 0}$ a function determining the amount of noise added. When $\lambda(t) = 0$, this is equivalent to the probability flow ODE (2). Note that we substitute t for $\sigma(t)$ for consistency, as Batzolis et al. (2026) define $\sigma(t) = t$. The additional drift and injected variance then approximately match EDMs churn mechanism when:

$$\lambda(t_i) \approx \frac{\gamma_i t_i}{|h_i|} \tag{22}$$

With γ_i being the EDM-churn parameter indicating the amount of noise added in a step, defined by Karras et al. (2022):

$$\gamma_i = \begin{cases} \min(S_{churn}/N, \sqrt{2} - 1) & t_i \in (S_{tmin}, S_{tmax}) \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

Where S_{churn} is a hyperparameter defining the amount of noise added, and S_{tmin}, S_{tmax} define the minimum and maximum t between which noise is applied.

[2]Found at https://github.com/GBATZOLIS/BitstreamDiffusion

Table 8: Hyperparameters for text generation using the PI solver.

Parameter	PI Solver		
	63 NFE	127 NFE	255 NFE
τ_a	0.005	0.005	0.005
τ_r	0.0888	0.1017	0.0896
α	0.9	0.9	0.9
α_{min}	0.2	0.2	0.2
α_{max}	5	5	5
h_{start}	35	20	20
k_i	0.3	0.3	0.3
k_p	0.1	0.1	0.1

Table 9: PI solver ablations hyperparameters.

Configuration	Variable Name	49 NFE	75 NFE	99 NFE
Config C	τ_{rel}	91	89	67
Config D	τ_{abs}	8.5	5.3	4.5

D.3.2 HYPERPARAMETERS

For parameterizing the EDM-churn parameter, we set $S_{tmin} = 0.08$, $S_{tmax} \approx 76.756$, as those are the values Batzolis et al. (2026) set for their samplers. We also set S_{churn} to the value they use, although they choose it to give a constant γ of 0.185 when $t_i \in (S_{tmin}, S_{tmax})$, regardless of the number of steps N. We set N to the number of steps taken, so equal to $\lfloor \frac{\text{NFE}+1}{2} \rfloor$ for second-order samplers. The hyperparameters we used for the PI sampler can be found in Table 8, with the method of how these were obtained described in Appendix C. Note that the 63 NFE configuration has lower τ_{rel} than the other higher NFE configurations. This is caused by a larger h_0 requiring a smaller τ_{rel} to counterbalance it, but does improve results for 63 NFE. For generating the PI static schedules, we ran the PI sampler for 1024 samples at the NFE the schedule is used at, using the hyperparameters matching that NFE.

D.4 ABLATIONS

D.4.1 PI SOLVER ABLATIONS

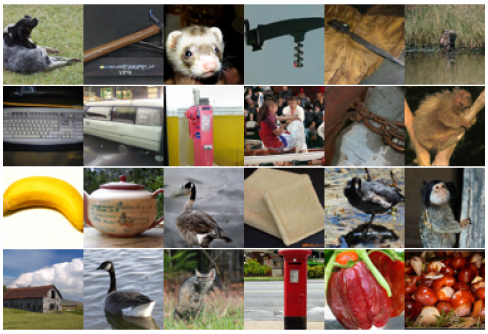
For all PI solver ablations, the setup and hyperparameters described in subsection D.2 was used, specifically using the FFHQ checkpoint. We evaluated four configurations, the first of these four, configuration A, being the PI solver detailed in section 4, with the hyperparameters found in Table 7. Configuration B is identical to configuration A, except that it sets $k_p = 0$, disabling the proportional component of equation 4, therefore ignoring the contribution of the previous error. Configuration C is another alteration of configuration A. This time, to mixed absolute tolerance, as given by equation 13 was used. This did require alteration of the relative tolerance as well. The used values can be found in Table 9. Lastly, configuration D uses the error estimator given by equation 12, which eliminates the relative tolerance and only leaves τ_{abs}. This did have to be tuned to allow for the target NFE to be reached. The used τ_{abs} can also be found in Table 9.

D.4.2 SCHEDULE ACROSS MODALITIES

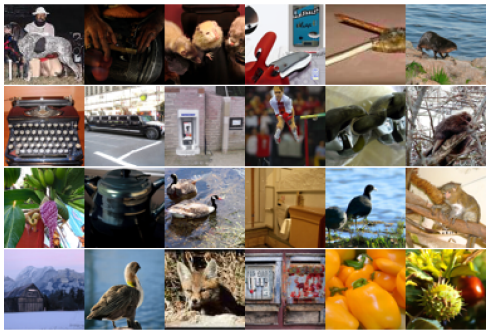
For the foreign FFHQ schedule, we used the LM1B PI static schedule generated at 255 NFE, using the exact setup described in subsection D.3 to generate the schedule. For generating images, we used the setup described in subsection D.2. For the foreign schedule on LM1B, we used the FFHQ PI static schedule generated at 99 NFE, using the setup described in subsection D.2 to generate the schedule, and using the setup described in subsection D.3 to generate the samples.

E SAMPLES

E.1 IMAGENET-64



(a) EDM-churn sampler with EDM schedule.



(b) EDM-churn sampler with PI static schedule.



(c) Gotta Go Fast adaptive sampler.



(d) Proportional-Integral (ours) adaptive sampler.



(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.

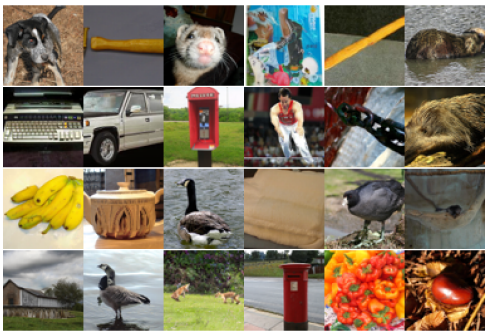


(g) Euler-Maruyama with EDM schedule.



(h) Euler-Maruyama with entropic schedule.

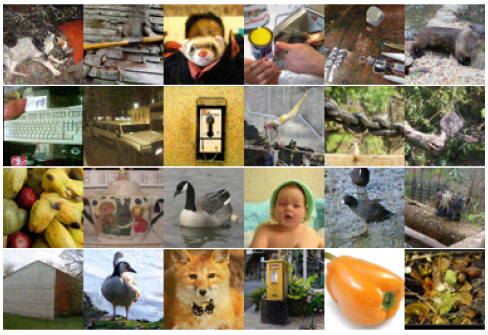
Figure 8: ImageNet-64 Images generated using different SDE solvers using the EDM2-XXL FID optimized checkpoint at 49 function evaluations



(a) EDM-churn sampler with EDM schedule.



(b) EDM-churn sampler with PI static schedule.



(c) Gotta Go Fast adaptive sampler.



(d) Proportional-Integral (ours) adaptive sampler.



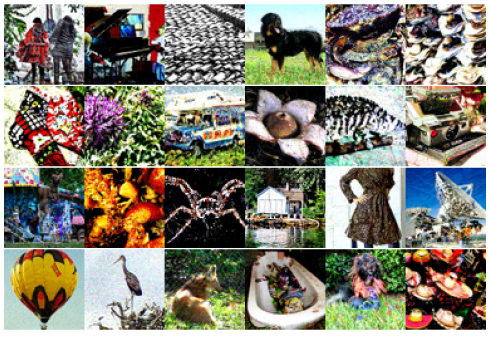
(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.

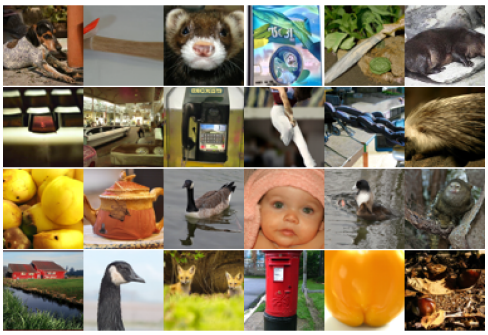


(g) Euler-Maruyama with EDM schedule.

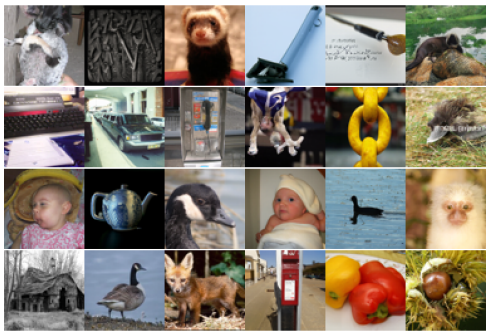


(h) Euler-Maruyama with entropic schedule.

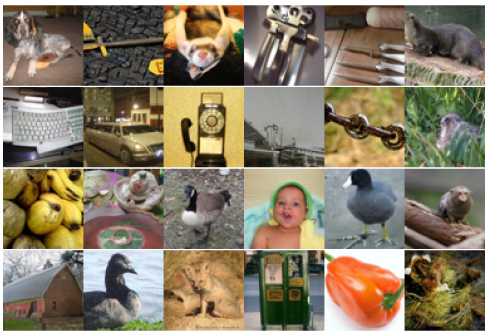
Figure 9: ImageNet-64 Images generated using different SDE solvers using the EDM2-XXL FID optimized checkpoint at 75 function evaluations



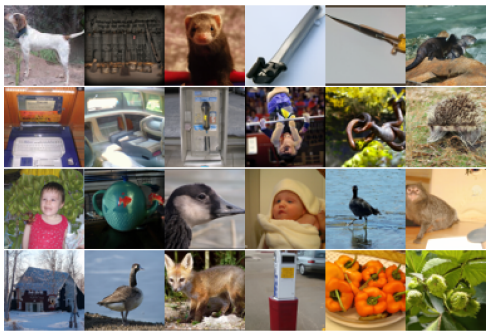
(a) EDM-churn sampler with EDM schedule.



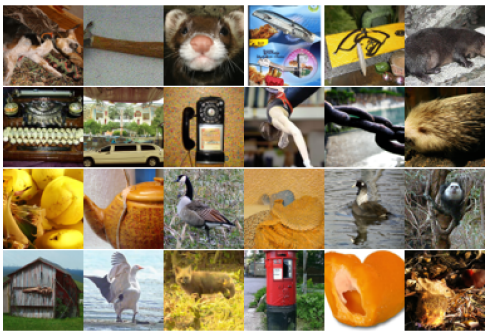
(b) EDM-churn sampler with PI static schedule.



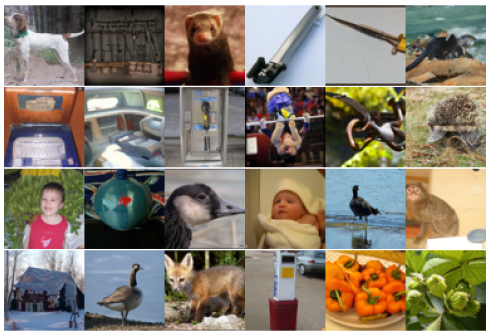
(c) Gotta Go Fast adaptive sampler.



(d) Proportional-Integral (ours) adaptive sampler.



(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.



(g) Euler-Maruyama with EDM schedule.



(h) Euler-Maruyama with entropic schedule.

Figure 10: ImageNet-64 Images generated using different SDE solvers using the EDM2-XXL FID optimized checkpoint at 99 function evaluations

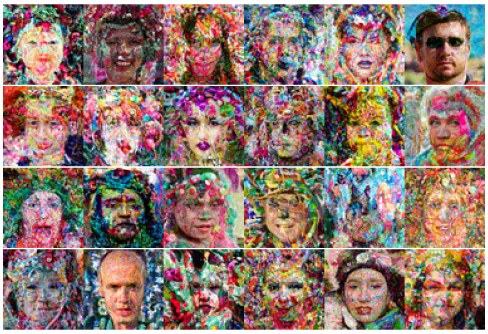
E.2 FFHQ



(a) EDM-churn sampler with EDM schedule.



(b) EDM-churn sampler with PI static schedule.



(c) Gotta Go Fast adaptive sampler.



(d) Proportional-Integral (ours) adaptive sampler.



(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.



(g) Euler-Maruyama with EDM schedule.



(h) Euler-Maruyama with entropic schedule.

Figure 11: FFHQ Images generated using different SDE solvers using the EDM-FFHQ-64x64 unconditional VP checkpoint at 49 function evaluations



(a) EDM-churn sampler with EDM schedule.



(b) EDM-churn sampler with PI static schedule.



(c) Gotta Go Fast adaptive sampler.



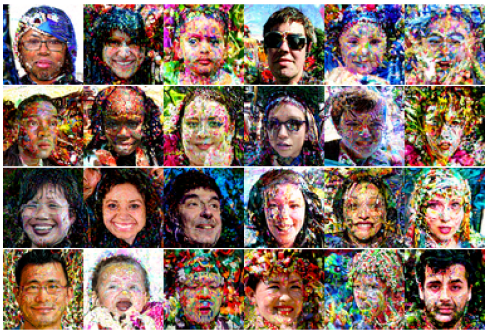
(d) Proportional-Integral (ours) adaptive sampler.



(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.



(g) Euler-Maruyama with EDM schedule.



(h) Euler-Maruyama with entropic schedule.

Figure 12: FFHQ Images generated using different SDE solvers using the EDM-FFHQ-64x64 unconditional VP checkpoint at 75 function evaluations



(a) EDM-churn sampler with EDM schedule.



(b) EDM-churn sampler with PI static schedule.



(c) Gotta Go Fast adaptive sampler.



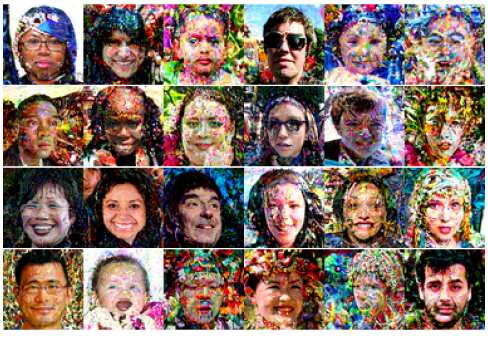
(d) Proportional-Integral (ours) adaptive sampler.



(e) Stochastic Heun with EDM schedule.



(f) Stochastic Heun with PI static schedule.



(g) Euler-Maruyama with EDM schedule.



(h) Euler-Maruyama with entropic schedule.

Figure 13: FFHQ Images generated using different SDE solvers using the EDM-FFHQ-64x64 unconditional VP checkpoint at 99 function evaluations