DUAL-SOLVER: A GENERALIZED ODE SOLVER FOR DIFFUSION MODELS WITH DUAL PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion models deliver state-of-the-art image quality. However, sampling is costly at inference time because it requires many model evaluations (number of function evaluations, NFEs). To reduce NFEs, classical ODE multistep methods have been adopted. Yet differences in the choice of prediction type (noise/data/velocity) and integration domain (half log-SNR/noise-to-signal ratio) lead to different outcomes. We introduce Dual-Solver, which generalizes multistep samplers by introducing learnable parameters that continuously (i) interpolate among prediction types, (ii) select the integration domain, and (iii) adjust the residual terms. It maintains the traditional predictor-corrector structure and guarantees second-order local accuracy. These parameters are learned with a classification-based objective using a frozen pretrained classifier (e.g., ViT or CLIP). On ImageNet class-conditional generation (DiT, GM-DiT) and text-to-image (SANA, PixArt-α), Dual-Solver improves FID and CLIP scores in the low-NFE regime ($3 \leq$ NFE ≤ 9) across backbones.

1 INTRODUCTION

Generative modeling aims to learn a data distribution and draw new samples that resemble real data. Classic approaches include autoregressive models that factorize likelihoods over pixels or tokens (Van den Oord et al., 2016; Salimans et al., 2017), variational auto-encoders that optimize an evidence lower bound (Kingma & Welling, 2013; Vahdat & Kautz, 2020), flow-based models that construct exact, invertible density maps (Dinh et al., 2014; Kingma & Dhariwal, 2018), and generative adversarial networks that learn via a discriminator–generator (Goodfellow et al., 2020; Arjovsky et al., 2017). Diffusion models have emerged as a modern family in this landscape: the seminal work of (Sohl-Dickstein et al., 2015) have introduced diffusion probabilistic modeling with a forward noising process and a learned reverse process trained by minimizing a KL-divergence objective. Subsequent reformulations (Ho et al., 2020) have streamlined training and inference with simple denoising objectives (e.g., noise or data prediction), leading to state-of-the-art fidelity and robust scaling. Today, diffusion models drive progress across modalities, including images (Dhariwal & Nichol, 2021; Rombach et al., 2022), audio (Kong et al., 2020; Liu et al., 2023), and video (Ho et al., 2022; Kong et al., 2024).

Diffusion models generate samples by advancing a Markov chain in the latent domain and repeatedly evaluating a neural network at each step, making inference cost scale with the number of function evaluations (NFEs). Leveraging the probability–flow formulation, which casts sampling as an ordinary differential equation (Song et al., 2021b), a large literature has pursued ODE-based acceleration. Along one axis, classical numerical methods—singlestep Runge–Kutta (Runge, 1895) and multistep Adams–Bashforth (Bashforth & Adams, 1883)—provide off-the-shelf accuracy–NFE trade-offs for a given evaluation budget (Butcher, 2016). Along a second axis, diffusion-dedicated solvers exploit the structure of the denoising dynamics: they approximate noise/data predictions with low-order Taylor expansions or Lagrange interpolation and derive closed-form updates (Lu et al., 2022a;b; Qinsheng & Chen, 2023; Zhao et al., 2023; Xue et al., 2024). Lastly, there are learned solvers that learn the timestep schedule and other sampling-related parameters (Zhou et al., 2024; Shaul et al., 2023; 2024; Wang et al., 2025). Because these parameters depend on the backbone and the dataset, such solvers are typically confined to a fixed backbone and specific settings ((e.g., a chosen NFE and CFG (Ho & Salimans, 2021)). Training also incurs substantial preparation overhead, as it requires many teacher trajectories or final samples generated at high NFE. Nevertheless,

(a) DPM-Solver++ (Lu et al., 2022b) (b) Dual-Solver (Ours) (c) BNS-Solver (Shaul et al., 2024) (d) DS-Solver (Wang et al., 2025)

Figure 1: **Sampling results.** SANA (Xie et al., 2024), NFE=3, CFG=4.5. See Fig. 21 for further results.

compared to the classical methods and dedicated solvers discussed above, learned solvers deliver substantially better sample quality (e.g., lower FID) and are therefore an active area of research.

We introduce Dual-Solver, a learned solver for diffusion models with three types of learnable parameters:

- a prediction parameter γ that interpolates among noise, data, and velocity prediction types (Sec. 3.1);
- a domain change parameter τ that interpolates between the log and linear domains (Sec. 3.2);
- a residual parameter κ that adjusts the residual term while preserving second-order accuracy (Sec. 3.3).

We further propose a classification-based learning strategy that yields high-fidelity images even in the low-NFE regime (Section 5.2). Unlike regression-based learning, which typically requires many target samples at high NFE, our approach requires no target samples. Solver parameters are learned using either pretrained image classifiers (He et al., 2016; Dosovitskiy et al., 2020; Howard et al., 2017) or the zero-shot classifier (Radford et al., 2021). For $3 \leq \text{NFE} \leq 9$, it outperforms prior state-of-the-art solvers (Section 6).

2 Preliminaries

2.1 Diffusion Models

Training process. Diffusion models (Ho et al., 2020; Song et al., 2021b) train a backbone network as follows. We sample a clean sample $\boldsymbol{x}_0$ from the data distribution and noise $\boldsymbol{\epsilon}$ from the standard normal, then linearly combine them with weights α_t and σ_t:

$$\boldsymbol{x}_t \;=\; \alpha_t\, \boldsymbol{x}_0 \;+\; \sigma_t\, \boldsymbol{\epsilon}, \qquad \text{where } \boldsymbol{x}_0 \sim p_{\text{data}},\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}),\; 0 \leq t \leq T. \tag{1}$$

Here, α_t and σ_t are set by a predefined schedule. Common choices include variance-preserving (VP), with $\alpha_t^2 + \sigma_t^2 = 1$ (Sohl-Dickstein et al., 2015; Ho et al., 2020); variance-exploding (VE), with $\alpha_t = 1,\ \sigma_t \geq 0$ (Song & Ermon, 2019; 2020); and optimal transport (OT), with $\alpha_t = 1 - t,\ \sigma_t = t,\ T = 1$ (Lipman et al., 2022). The backbone is trained to take a noisy sample $\boldsymbol{x}_t$ and the time t as input and to predict one of the following: the noise $\boldsymbol{\epsilon}$, the clean sample $\boldsymbol{x}_0$ (Ho et al., 2020), or the velocity $\boldsymbol{v}_t = \dot{\alpha}_t \boldsymbol{x}_0 + \dot{\sigma}_t \boldsymbol{\epsilon}$ (Lipman et al., 2022). By convention, the backbone parameters are denoted θ, and the predictions are written as $\boldsymbol{\epsilon}_\theta$ (noise), $\boldsymbol{x}_\theta$ (data), and $\boldsymbol{v}_\theta$ (velocity).

Sampling process. The dynamics in Eq. 1 can be written as the following stochastic differential equation (SDE):

$$d\boldsymbol{x}_t \;=\; f_t\, \boldsymbol{x}_t\, dt \;+\; g_t\, d\boldsymbol{w}_t, \qquad f_t \;=\; \frac{d \log \alpha_t}{dt}, \quad g_t^2 \;=\; \frac{d\sigma_t^2}{dt} \;-\; 2\, \frac{d \log \alpha_t}{dt}\, \sigma_t^2, \tag{2}$$

where $\boldsymbol{w}_t$ is a standard Wiener process. For any time t, a probability-flow ODE that shares the same marginal distribution as the SDE has been proposed (Song et al., 2021b):

$$\frac{d\boldsymbol{x}_t}{dt} = f_t\,\boldsymbol{x}_t - \frac{1}{2}\,g_t^2\,\nabla_{\boldsymbol{x}}\log q_t(\boldsymbol{x}_t), \quad \text{where } \nabla_{\boldsymbol{x}}\log q_t(\boldsymbol{x}_t) = -\frac{\mathbb{E}[\boldsymbol{\epsilon}\mid \boldsymbol{x}_t]}{\sigma_t} \approx -\frac{\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)}{\sigma_t}. \tag{3}$$

Given this ODE, one can perform sampling using classical numerical methods beyond Euler, such as singlestep Runge–Kutta (Runge, 1895) and multistep Adams–Bashforth schemes (Bashforth & Adams, 1883). Moreover, several works split the right-hand side into linear and non-linear parts, and evaluate the non-linear term using finite-difference approximations (Lu et al., 2022a;b; Zhao et al., 2023) or via Lagrange interpolation (Qinsheng & Chen, 2023; Xue et al., 2024).

2.2 Prediction Types

Depending on the form of a model's output, we distinguish three prediction types: *noise*, *data*, and *velocity*. The diffusion model of Sohl-Dickstein et al. (2015) was parameterized to predict the mean and covariance matrix of the next sample's distribution. Following Ho et al. (2020), it has become standard to train networks that predict either the additive noise or the clean data. In parallel, the flow matching literature (Lipman et al., 2022) proposes models that output a vector field; in this paper, we refer to this as *velocity* prediction. Although these outputs differ, they are mutually convertible through simple transformations. In particular, one can obtain the desired prediction from any other via

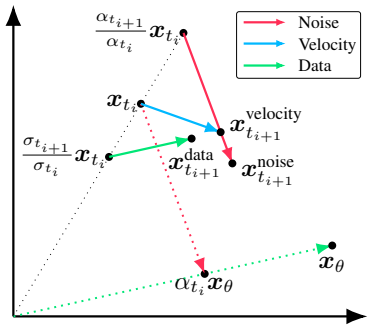


Figure 2: Euler updates for noise, velocity, and data predictions.

$$\boldsymbol{x}_\theta(\boldsymbol{x}_t,t) = \frac{\boldsymbol{x}_t - \sigma_t\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)}{\alpha_t}, \quad \boldsymbol{v}_\theta(\boldsymbol{x}_t,t) = \frac{d\alpha_t}{dt}\,\boldsymbol{x}_\theta(\boldsymbol{x}_t,t) + \frac{d\sigma_t}{dt}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t). \tag{4}$$

Depending on which of the three predictions is used, we derive Eq. 3 in different forms. The resulting integral expressions are summarized in Table 1.

Discretization discrepancy. We ask a simple question: do the predictions yield the same update? In continuous time, yes; in discrete time, no. Fig. 2 illustrates this discrepancy in two dimensions. For example, with a left-point first-order Euler rule in the λ-domain (half log-SNR), the noise-prediction update is $\boldsymbol{x}^{\text{noise}}_{t_{i+1}} = \frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\big[\boldsymbol{x}_{t_i} - \sigma_{t_i}\,\Delta\lambda_{t_i}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_i},t_i)\big]$. If we take the Euler update for the data prediction and rewrite it using $\boldsymbol{x}_\theta = (\boldsymbol{x}_{t_i} - \sigma_{t_i}\boldsymbol{\epsilon}_\theta)/\alpha_{t_i}$, we obtain $\boldsymbol{x}^{\text{data}}_{t_{i+1}} = \frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\big[(1{+}\Delta\lambda_{t_i})e^{-\Delta\lambda_{t_i}}\,\boldsymbol{x}_{t_i} - e^{-\Delta\lambda_{t_i}}\sigma_{t_i}\,\Delta\lambda_{t_i}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_i},t_i)\big]$. Since $e^{-\Delta\lambda_{t_i}} = 1{-}\Delta\lambda_{t_i}{+}\frac{1}{2}\Delta\lambda_{t_i}^2{+}\cdots$, a discrepancy appears at order $O((\Delta\lambda_{t_i})^2)$. This naturally raises the question of which update is preferable in practice.

	Differential form	**Integral form on** $[t_i, t_{i+1}]$
Noise $\boldsymbol{\epsilon}_\theta$	$\frac{d\boldsymbol{x}_t}{dt} = \frac{d\log\alpha_t}{dt}\,\boldsymbol{x}_t - \sigma_t\,\frac{d\lambda_t}{dt}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)$	$\boldsymbol{x}_{t_{i+1}} = \frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\,\boldsymbol{x}_{t_i} - \alpha_{t_{i+1}}\int_{t_i}^{t_{i+1}} \frac{\sigma_t}{\alpha_t}\,\frac{d\lambda_t}{dt}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)\,dt$
Velocity $\boldsymbol{v}_\theta$	$\frac{d\boldsymbol{x}_t}{dt} = \frac{d\alpha_t}{dt}\,\boldsymbol{x}_\theta(\boldsymbol{x}_t,t) + \frac{d\sigma_t}{dt}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)$	$\boldsymbol{x}_{t_{i+1}} = \boldsymbol{x}_{t_i} + \int_{t_i}^{t_{i+1}} \big[\frac{d\alpha_t}{dt}\,\boldsymbol{x}_\theta(\boldsymbol{x}_t,t) + \frac{d\sigma_t}{dt}\,\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)\big]\,dt$
Data $\boldsymbol{x}_\theta$	$\frac{d\boldsymbol{x}_t}{dt} = \frac{d\log\sigma_t}{dt}\,\boldsymbol{x}_t + \alpha_t\,\frac{d\lambda_t}{dt}\,\boldsymbol{x}_\theta(\boldsymbol{x}_t,t)$	$\boldsymbol{x}_{t_{i+1}} = \frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\,\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}\int_{t_i}^{t_{i+1}} \frac{\alpha_t}{\sigma_t}\,\frac{d\lambda_t}{dt}\,\boldsymbol{x}_\theta(\boldsymbol{x}_t,t)\,dt$

Table 1: Differential and integral forms for noise, data, and velocity predictions. (α_t: signal rate, σ_t: noise rate, λ_t: $\log\alpha_t/\sigma_t$)

3 DUAL-SOLVER

3.1 DUAL PREDICTION WITH PARAMETER γ

We propose a *dual prediction* scheme that uses both $\boldsymbol{x}_\theta$ and $\boldsymbol{\epsilon}_\theta$ together. We call it *dual* because it treats $\boldsymbol{x}_\theta$ and $\boldsymbol{\epsilon}_\theta$ separately, unlike velocity prediction, which bundles them into $\boldsymbol{v}_\theta$. Moreover, we introduce the following integral formulation parameterized by γ, which interpolates between the integral forms of noise, velocity, and data prediction.

Integral form of dual prediction.

$$\boldsymbol{x}_{t_{i+1}} = \begin{cases} \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma} \left[\int_{t_i}^{t_{i+1}} \frac{d}{dt}\big(\alpha_t\, \sigma_t^{-\gamma}\big)\; \boldsymbol{x}_\theta(t)\, dt \;+\; \int_{t_i}^{t_{i+1}} \frac{d}{dt}\Big(\sigma_t^{1-\gamma}\Big)\; \boldsymbol{\epsilon}_\theta(t)\, dt\right], \; \gamma \geq 0, \\ \left(\frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma} \boldsymbol{x}_{t_i} \;+\; \alpha_{t_{i+1}}^{-\gamma} \left[\int_{t_i}^{t_{i+1}} \frac{d}{dt}\Big(\alpha_t^{1+\gamma}\Big)\; \boldsymbol{x}_\theta(t)\, dt \;+\; \int_{t_i}^{t_{i+1}} \frac{d}{dt}(\sigma_t\, \alpha_t^{\gamma})\; \boldsymbol{\epsilon}_\theta(t)\, dt\right], \; \gamma < 0. \end{cases} \tag{5}$$

Here, we write $\boldsymbol{x}_\theta(t) \coloneqq \boldsymbol{x}_\theta(\boldsymbol{x}_t, t)$ and $\boldsymbol{\epsilon}_\theta(t) \coloneqq \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$.

- When $\gamma = -1$, the integral reduces to the noise-prediction form (Table 1).
- When $\gamma = 0$, the integral reduces to the velocity-prediction form (Table 1).
- When $\gamma = 1$, the integral reduces to the data-prediction form (Table 1).

From Eq. 3, we derive the differential form of dual prediction and, by integrating, obtain the integral form given above; the full derivation is provided in Appendix B.1. For brevity, in Secs. 3.2 and 3.3 we consider only $\gamma \geq 0$; the case $\gamma < 0$ follows by the same steps.

3.2 LOG-LINEAR DOMAIN CHANGE WITH PARAMETER τ

Domain change. Let $L : (0, \infty) \to \mathcal{I}$ be a C^1 diffeomorphism onto an interval $\mathcal{I} \subseteq \mathbb{R}$. Applying a change of variables to the two integrals in Eq. 5, define

$$u(t; \tau_u) \;=\; L\big(\alpha_t\, \sigma_t^{-\gamma}; \tau_u\big), \qquad v(t; \tau_v) \;=\; L\big(\sigma_t^{1-\gamma}; \tau_v\big).$$

Here, τ_u and τ_v parametrize L in the u- and v-integrals, respectively. Denote $u_i = u(t_i),\; u_{i+1} = u(t_{i+1})$ and $v_i = v(t_i),\; v_{i+1} = v(t_{i+1})$. By the chain rule, $\frac{d}{dt} L^{-1}(u) = \frac{dL^{-1}(u)}{du}\, \frac{du}{dt}$ and similarly for v. Thus, for $\gamma \geq 0$ we obtain

$$\boldsymbol{x}_{t_{i+1}} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma} \left[\int_{u_i}^{u_{i+1}} \frac{dL^{-1}(u; \tau_u)}{du}\, \boldsymbol{x}_\theta(u)\, du + \int_{v_i}^{v_{i+1}} \frac{dL^{-1}(v; \tau_v)}{dv}\, \boldsymbol{\epsilon}_\theta(v)\, dv\right]. \tag{6}$$

Here, $\boldsymbol{x}_\theta(u) \coloneqq \boldsymbol{x}_\theta(\boldsymbol{x}_{u^{-1}(u)}, u^{-1}(u))$ and $\boldsymbol{\epsilon}_\theta(v) \coloneqq \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{v^{-1}(v)}, v^{-1}(v))$. The full equation is given in Eq. 11.

Log-linear transform. Previous works (Dockhorn et al., 2022; Qinsheng & Chen, 2023; Zhou et al., 2024) adapt the linear transform $L(y) = y$ with noise prediction. Because $\frac{d}{du} L^{-1}(u) = 1$, the integrand carries no weighting factor, making it straightforward to develop approximations such as Taylor expansions and Lagrange interpolation. By contrast, the other works (Lu et al., 2022a;b; Zhao et al., 2023; Xue et al., 2024) use a logarithmic transform $L(y) = \log y$. Because $\frac{d}{du} L^{-1}(u) = e^u$, the integrand carries an exponential weight. A closed-form approximation can be obtained via an exponential integrator (Hochbruck & Ostermann, 2010) or by using Lagrange interpolation. Motivated by these works, we propose a *log–linear* transform that interpolates between the two via a scalar parameter τ:

$$L(y; \tau) \;=\; \frac{\log(1 + \tau y)}{\tau}, \quad \tau > 0. \tag{7}$$

This transform is invertible, with inverse $L^{-1}(u; \tau) \;=\; \big(e^{\tau u} - 1\big)/\tau$; its weighting factor is $\frac{\mathrm{d}}{\mathrm{d}u} L^{-1}(u; \tau) = e^{\tau u}$. Consequently, it has the following properties:

- As $\tau \to 0^+$: $\frac{\mathrm{d}}{\mathrm{d}u} L^{-1}(u; \tau) \to 1$ (no weight).
- When $\tau = 1$: $\frac{\mathrm{d}}{\mathrm{d}u} L^{-1}(u; \tau) = e^u$ (exponential weight).

We apply the log–linear transform to Eq. 6, allowing separate parameters τ_u and τ_v for the u- and v-integrals, respectively.

3.3 Second-Order Approximation with Parameter κ

On the interval $[u_i, u_{i+1}]$, we approximate $\boldsymbol{x}_\theta$ by using the second-order forward-difference approximation $\boldsymbol{x}_\theta(u) = \boldsymbol{x}_\theta(u_i) + \frac{\Delta \boldsymbol{x}_\theta(u_i)}{\Delta u_i}(u - u_i)$, where $\boldsymbol{x}_\theta(u_i) \coloneqq \boldsymbol{x}_\theta(\boldsymbol{x}_{u^{-1}(u_i)}, u^{-1}(u_i))$, $\Delta \boldsymbol{x}_\theta(u_i) \coloneqq \boldsymbol{x}_\theta(u_{i+1}) - \boldsymbol{x}_\theta(u_i)$, and $\Delta u_i \coloneqq u_{i+1} - u_i$. We also introduce a function $K(\Delta u_i; \kappa_u) = \kappa_u(\Delta u_i)^2$, an $\mathcal{O}((\Delta u_i)^2)$ term, to allow additional flexibility in the residual term while preserving second-order local accuracy. κ_u is a real scalar parameter that controls the magnitude of the residual term. Applying the same approximations to $\boldsymbol{\epsilon}_\theta$ yields the following second-order corrector $\boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}}$:

$$\boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta(u_i)\,\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta \boldsymbol{x}_\theta(u_i)}{2}\left(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\right) + \boldsymbol{\epsilon}_\theta(v_i)\,\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta \boldsymbol{\epsilon}_\theta(v_i)}{2}\left(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\right)\Big]. \tag{8}$$

Here, $\boldsymbol{x}_\theta(u_i) \coloneqq \boldsymbol{x}_\theta(\boldsymbol{x}_{u^{-1}(u_i)}, u^{-1}(u_i))$, $\Delta \boldsymbol{x}_\theta(u_i) \coloneqq \boldsymbol{x}_\theta(u_{i+1}) - \boldsymbol{x}_\theta(u_i)$, $\Delta L^{-1}(u_i;\tau_u) \coloneqq L^{-1}(u_{i+1};\tau_u) - L^{-1}(u_i;\tau_u)$, and $\Delta u_i \coloneqq u_{i+1} - u_i$; the definitions for $\boldsymbol{\epsilon}_\theta(v_i)$, $\Delta \boldsymbol{\epsilon}_\theta(v_i)$, $\Delta L^{-1}(v_i;\tau_v)$, and Δv_i are analogous. The first-order predictor and second-order predictor equations are provided in Appendix B.3. **Note.** In Eq. 8, $L^{-1}(u_i;\tau_u) = \alpha_{t_i}\sigma_{t_i}^{-\gamma}$, so there is no need to invert L explicitly. The parameter τ_u determines the scale of Δu_i, which feeds into B. The same applies to $L^{-1}(v_i;\tau_v)$.

Theorem 3.1 (Local truncation error)**.** *Assume that $\boldsymbol{x}_\theta(u)$ and $\boldsymbol{\epsilon}_\theta(v)$ are C^2 on $[u_i, u_{i+1}]$ and $[v_i, v_{i+1}]$, respectively. Let $\boldsymbol{x}_{t_{i+1}}^{\mathrm{exact}}$ denote the exact update in equation 6, and let $x_{t_{i+1}}^{\mathrm{2nd-corr.}}$ denote the second–order corrector defined in equation 8. Then we have*

$$\left\|\boldsymbol{x}_{t_{i+1}}^{\mathrm{exact}} - \boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}}\right\| = \mathcal{O}\big((\Delta u_i)^3 + (\Delta v_i)^3\big).$$

We provide the detailed proof in Appendix B.4, and by the same argument the accuracies of the first- and second-order predictors can also be shown.

4 Implementation Details

Dual-Solver performs sampling using a predictor–corrector scheme (Butcher, 2016) based on the equations developed above. We examine this sampling scheme in detail in Sec. 4.1 and then present the set of learnable parameters in Sec. 4.2.

4.1 Sampling Scheme

Alg. 1 details the sampling procedure of Dual-Solver. Sampling requires a backbone that provides both $\boldsymbol{x}_\theta$ and $\boldsymbol{\epsilon}_\theta$; when only one head is available, the other can be obtained via Eq. 4. Given M steps, we use timesteps $\{t_i\}_{i=0}^{M}$ with $t_0 = T$ and draw the initial noise $\boldsymbol{x}_{t_0} \sim \mathcal{N}(0, I)$. We also maintain a list ℓ to store previous evaluations. Empirically, a first-order predictor with a second-order corrector performs best (Sec. 6.2). At step i, the first-order predictor takes the current state $\boldsymbol{x}_{t_i}$ together with the model evaluations $\{\boldsymbol{x}_\theta(\boldsymbol{x}_{t_i}, t_i),\ \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_i}, t_i)\}$ and produces a provisional sample $\boldsymbol{x}'_{t_{i+1}}$. The second-order corrector then combines the evaluations at t_i with fresh evaluations at t_{i+1}, i.e., $\{\boldsymbol{x}_\theta(\boldsymbol{x}'_{t_{i+1}}, t_{i+1}),\ \boldsymbol{\epsilon}_\theta(\boldsymbol{x}'_{t_{i+1}}, t_{i+1})\}$, to yield the next sample $\boldsymbol{x}_{t_{i+1}}$. At the final step $i = M - 1$, the corrector is not applied. Explicit formulas for the first-order predictor and second-order corrector are given in Eqs. 13 and 21.

4.2 Learnable Parameters

For each i-th predictor/corrector step, the parameter sets are $\phi_i^{\mathrm{pred}} = \{\gamma_i^{\mathrm{pred}}, \tau_{u,i}^{\mathrm{pred}}, \tau_{v,i}^{\mathrm{pred}}, \kappa_{u,i}^{\mathrm{pred}}, \kappa_{v,i}^{\mathrm{pred}}\}$ and $\phi_i^{\mathrm{corr}} = \{\gamma_i^{\mathrm{corr}}, \tau_{u,i}^{\mathrm{corr}}, \tau_{v,i}^{\mathrm{corr}}, \kappa_{u,i}^{\mathrm{corr}}, \kappa_{v,i}^{\mathrm{corr}}\}$. Thus, each step uses $2 \times 5 = 10$ parameters. (The last step ($i = M - 1$) does not use the corrector, so it has 5 parameters.) Fig. 3 shows the learned parameters for the NFE=5 setting using a DiT (Peebles & Xie, 2023) backbone. Assuming a noise-prediction backbone (the same reasoning applies to data- or velocity-prediction) and a first-order predictor with a second-order corrector (Sec. 4.1), ϕ_i^{pred} and ϕ_i^{corr} determine the coefficients for an update that combines the current state $\boldsymbol{x}_{t_i}$ and two model

Algorithm 1 Dual-Solver predictor–corrector sampling (Sec. 4.1)

Require: Diffusion backbone with dual prediction $\{\boldsymbol{x}_\theta, \boldsymbol{\epsilon}_\theta\}$, timesteps $\{t_i\}_{i=0}^{M}$, initial noise $\boldsymbol{x}_{t_0}$, empty list ℓ, parameters ϕ

Evaluate $\{\boldsymbol{x}_\theta(\boldsymbol{x}_{t_0}, t_0), \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_0}, t_0)\}$ and add to L
for $i = 0$ **to** $M - 1$ **do**
  $\boldsymbol{x}'_{t_{i+1}} \leftarrow \mathrm{Predictor}(\boldsymbol{x}_{t_i}, \ell; \phi_i^{\mathrm{pred}})$
  if $i \geq M - 1$ **then break**
  Evaluate $\{\boldsymbol{x}_\theta(\boldsymbol{x}'_{t_{i+1}}, t_{i+1}), \boldsymbol{\epsilon}_\theta(\boldsymbol{x}'_{t_{i+1}}, t_{i+1})\}$ and add to ℓ
  $\boldsymbol{x}_{t_{i+1}} \leftarrow \mathrm{Corrector}(\boldsymbol{x}_{t_i}, \ell; \phi_i^{\mathrm{corr}})$
end for
return $\boldsymbol{x}'_{t_M}$

Algorithm 2 Hard-label classification for parameter learning (Sec. 5.2)

Require: Diffusion backbone with parameters θ, VAE decoder $\mathcal{D}$, pre-trained classifier $\mathcal{C}$, solver $\mathcal{S}$ with parameters ϕ, label dataset $\mathcal{Y}$, learning rate η

while not converged **do**
  Sample $\boldsymbol{x}_T \sim \mathcal{N}(0, I)$ ▷ initial noise
  Sample $y \sim \mathcal{Y}$ ▷ class label
  $\boldsymbol{x}_0 \leftarrow \mathcal{S}(\boldsymbol{x}_T; y, \phi, \theta)$ ▷ sampling
  $\hat{\boldsymbol{x}}_0 \leftarrow \mathcal{D}(\boldsymbol{x}_0)$ ▷ decoding
  $p \leftarrow \mathcal{C}(\hat{\boldsymbol{x}}_0)$ ▷ class probabilities
  $\mathcal{L} \leftarrow \mathrm{CrossEntropy}(p, y)$ ▷ loss
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}$ ▷ parameter update
end while

evaluations, $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}'_{t_i}, t_i)$ and $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}'_{t_{i+1}}, t_{i+1})$, to produce $\boldsymbol{x}_{t_{i+1}}$ (Appendix H). This may seem heavy, but Sec. 6.2 (B) shows it is necessary.

In addition to these parameters, we also learn the evaluation times $\{t_i\}_{i=1}^{M-1}$ (with t_0 and t_M fixed), where M denotes the number of steps. Following prior work (Tong et al., 2025; Wang et al., 2025), we employ unnormalized step variables $\{\Delta t'_i\}_{i=0}^{M-1}$ and apply a softmax over $i = 0, \ldots, M-1$ to obtain normalized step sizes $\{\Delta t_i\}_{i=0}^{M-1}$ (nonnegative and summing to one). The timesteps are obtained via a cumulative sum: $t_i = t_0 + (t_M - t_0) \sum_{k=0}^{i-1} \Delta t_k$, $i = 1, \ldots, M-1$.

5 Solver Parameter Learning

In this section, we review existing regression-based parameter learning methods, identify their limitations, and introduce a classification-based approach. Fig. 4 provides a schematic overview of all these methods. We apply them to our proposed Dual-Solver and report FID results for each method in Sec. 6.2 (C).

5.1 Regression-based Parameter Learning

In regression-based learning, a solver with trainable parameters is referred to as a student solver, while an existing fixed solver is referred to as a teacher solver, and the student is trained to imitate the behavior of the teacher running at a high NFE. Most prior works adopt *trajectory regression* (Shaul et al., 2023; Zhou et al., 2024; Wang et al., 2025), which compares the trajectories generated by the teacher and the student, or *sample regression* (Shaul et al., 2024), which compares only the final samples. Since comparisons in the trajectory or sample space often show a mismatch with visual perceptual quality, *feature regression* has been proposed (Tong et al., 2025), where the measure is computed in a feature space using metrics such as LPIPS (Zhang et al., 2018). However, all of these methods require a teacher solver and incur substantial overhead to prepare the supervision targets, and they tend to perform poorly in the very low NFE regime (e.g., NFE ≤ 5; see Table 3).

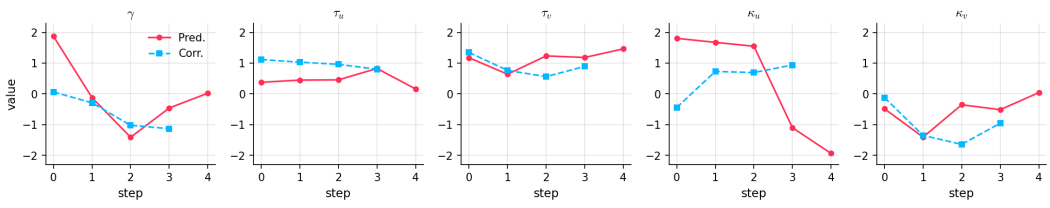


Figure 3: **Learned parameters.** Values of $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ across sampling steps, learned on DiT (Peebles & Xie, 2023) at NFE$= 5$. See Figs. 14, 15, 16, and 17 for further results.

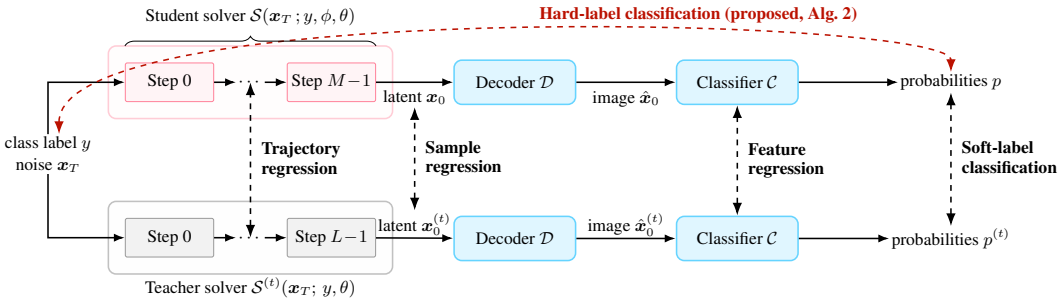


Figure 4: **Solver parameter learning methods.** It schematically illustrates trajectory, sample, and feature regression, as well as soft- and hard-label classification methods.

5.2 Classification-based Parameter Learning

Beyond feature regression, we consider *soft-label classification*, where we apply a cross-entropy loss between the classifier outputs (probabilities) of the student and the teacher. This approach yields improved results in the low-NFE regime (Table 3), but it still requires a teacher solver to generate the target probabilities.

To remove this dependency, we further propose *hard-label classification*, which compares the student's classifier outputs (probabilities) against the input class label using a cross-entropy loss, without any teacher solver. The overall training procedure is described in Alg. 2. The sample $\boldsymbol{x}_0$ generated by the solver $\mathcal{S}$ is passed through the decoder $\mathcal{D}$ and the classifier $\mathcal{C}$ to obtain class probabilities p. We then compute a cross-entropy loss between p and the class label $y \sim \mathcal{Y}$, and update the solver parameters ϕ for all time steps via backpropagation. For text-to-image tasks, the class labels are replaced with text prompts, which can be obtained from datasets such as MSCOCO 2014 (Lin et al., 2014) or MJHQ-30K (Li et al., 2024), and the cross-entropy loss is replaced with the CLIP loss (Radford et al., 2021).

Unlike regression to teacher targets, this method focuses on whether the generated samples lie on the correct side of the classifier's decision boundary, enabling more flexible training. A potential issue is a mismatch between the distribution learned by the classifier and the ground-truth data distribution, but this can be mitigated by selecting an appropriate classifier. In Sec. 6.2 (D), we analyze the relationship between the classifier's accuracy and the resulting FID scores.

6 Experiments

We benchmark Dual-Solver against two families of baselines:

- Dedicated solvers: DDIM (Song et al., 2021a), DPM-Solver++ (Lu et al., 2022b).
- Learned solvers: BNS-Solver (Shaul et al., 2024), DS-Solver (Wang et al., 2025).

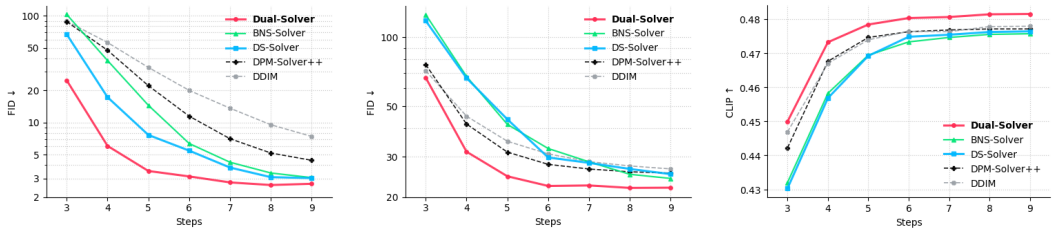
We select backbones that span diffusion and flow matching, covering ImageNet (Deng et al., 2009) conditional generation and text-to-image:

- DiT-XL/2-256×256 (Peebles & Xie, 2023): diffusion, ImageNet.
- PixArt-α XL-2-512 (Chen et al., 2023): diffusion, text-to-image.
- GM-DiT 256×256 (Chen et al., 2025): flow matching, ImageNet.
- SANA 600M-512px (Xie et al., 2024): flow matching, text-to-image.

Solver implementations are taken from official sources or reimplemented, and the backbones can be run easily via the diffusers library (von Platen et al., 2022). Further details are provided in Appendix C.

6.1 Main Quantitative Results

We evaluate quantitative performance using FID (Heusel et al., 2017) and CLIP score (Radford et al., 2021). For DiT and GM-DiT, FID is computed on 50k images uniformly sampled across the 1,000 ImageNet (Deng et al., 2009) classes. For SANA and PixArt-α, FID and CLIP are computed on the MSCOCO 2014 (Lin et al., 2014) validation set (30k image–caption pairs). The CLIP



(a) DiT, FID ↓ (Table 4). (b) PixArt-α, FID ↓ (Table 6). (c) PixArt-α, CLIP ↑ (Table 6).

(d) GM-DiT, FID ↓ (Table 5). (e) SANA, FID ↓ (Table 7). (f) SANA, CLIP ↑ (Table 7).

Figure 5: **Main quantitative results.** FID and CLIP score; evaluated on 50k (DiT/GM-DiT) and 30k (SANA/PixArt-α) samples; CFG: DiT=1.5, GM-DiT=1.4, SANA=4.5, PixArt-α=3.5.

score is computed as the cosine similarity between text and image features. As described in Sec. 5, we train Dual-Solver with a classification-based objective. For DiT and GM-DiT, MobileNetV3-Large (Howard et al., 2019) is used, and for SANA and PixArt-α, CLIP(RN101) is used. Fig. 5 plots the measured FID and CLIP scores. Across all evaluated NFEs, Dual-Solver outperforms competing solvers on both FID and CLIP for DiT, SANA, and PixArt-α. For GM-DiT, Dual-Solver underperforms at NFE 7–9; however, when trained with a trajectory regression-based objective, it surpasses the baselines at NFE 8 and 9 (Table 5).

6.2 Ablation Study

We conduct ablations to determine (A) the predictor–corrector order, (B) how much to constrain the degrees of freedom within the parameter set $\phi = \{\gamma, \tau, \kappa\}$, (C) the choice of learning method, and (D) the choice of classifier. Ablations are conducted with DiT (Peebles & Xie, 2023).

	Cross entropy (↓)			
NFE	**3**	**5**	**7**	**9**
(A) Predictor, Corrector Order				
p1	0.667	0.225	0.183	0.175
p1c2	**0.574**	**0.197**	**0.178**	**0.173**
p2	1.023	0.253	0.222	0.181
p2c2	5.009	0.317	0.203	0.191
(B) Parameters (γ, τ, κ) Setting				
$\gamma = 1$ fixed	0.816	0.223	0.182	0.176
$\gamma = 0$ fixed	0.600	0.202	0.183	0.180
$\gamma = -1$ fixed	7.871	7.676	0.238	0.196
$\tau = 1$ fixed	0.601	0.217	**0.175**	0.178
$\kappa = 0$ fixed	0.944	0.256	0.202	0.190
τ, κ shared	0.667	0.221	0.177	**0.169**
global γ	0.593	0.213	0.181	0.177
global τ	0.596	0.219	0.186	0.179
global κ	0.612	0.240	0.185	0.182
all learnable	**0.574**	**0.197**	0.178	0.173

Table 2: Ablation Study

(A) Predictor-corrector order. We ablate the predictor/corrector order: p1 (first-order predictor), p1c2 (first-order predictor + second-order corrector), p2 (second-order predictor), and p2c2 (second-order predictor + second-order corrector). The equations for each predictor and corrector are given in Eqs. 13, 17, and 21. As shown in Table 2 (A), p1c2 achieves superior performance across NFE $= 3, 5, 7, 9$. We therefore adopt p1c2 as the default configuration for Dual-Solver (Sec. 4.1).

(B) Parameters (γ, τ, κ) setting. We ablate the parameterization by fixing selected values or sharing them across parameters to reduce degrees of freedom. Choosing γ as 1, 0, or -1 recovers data, velocity, and noise prediction, respectively; $\tau = 1$ gives the log transform; and $\kappa = 0$ disables the second-order residual term. We also tie $\tau_u = \tau_v$ and $\kappa_u = \kappa_v$ to share parameters between the integrations for $\boldsymbol{x}_\theta$ and $\boldsymbol{\epsilon}_\theta$. Additionally, we consider a global setting in which each of γ, τ, and κ is shared across all sampling steps.

As shown in Table 2 (B), at low NFE (3 and 5), leaving all parameters free yields the best performance. At higher NFEs (7 and 9), configurations that fix or share certain parameters can oc-

casionally perform better. Since gaps are small and our focus is low NFE, we adopt the all-learnable setting by default.

(C) Parameter learning methods. Table 3 reports the results of applying the regression- and classification-based parameter learning methods discussed in Sec. 5 to Dual-Solver. Feature regression is implemented using LPIPS (Zhang et al., 2018), and it generally performs better than regression in the trajectory or sample space. The improvement is particularly pronounced in the very low NFE regime (NFE = 3, 5). Depending on which classifier (AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014), SqueezeNet (Iandola et al., 2016)) is used to extract features, the results vary significantly, underscoring the importance of classifier choice in feature space.

NFE	FID (↓) 3	5	7	9
Regression-based learning (Sec. 5.1)				
Sample	107.13	11.71	4.60	2.99
Trajectory	100.89	11.59	3.66	2.84
Feature (AlexNet)	47.75	7.24	3.42	2.91
Feature (VGG)	41.58	5.48	3.23	2.88
Feature (SqueezeNet)	44.00	7.07	3.31	2.79
Classification-based learning (Sec. 5.2)				
Soft-label	25.13	4.90	3.37	3.01
Hard-label	**24.91**	**3.52**	**2.75**	**2.67**

Table 3: Comparison of parameter learning methods

Classification-based methods further widen the performance gap relative to feature regression in the low NFE regime (NFE = 3, 5). In particular, the method that uses only hard labels achieves substantially superior results across all NFEs, which we attribute to the choice of an appropriate classifier; we investigate this in the next paragraph.

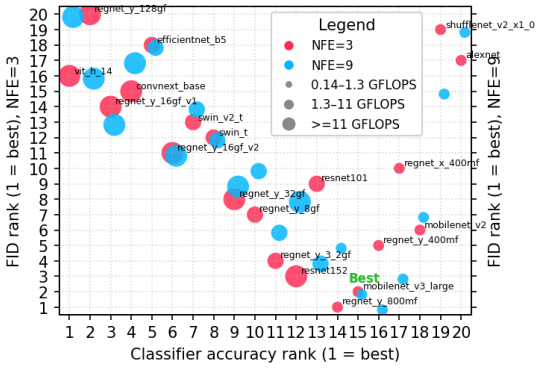


Figure 6: Top-5 accuracy vs. FID (Table 8).

(D) Classifier model selection. A natural question for classifier-based learning is which classifier to use. To investigate this, we evaluate 20 pretrained classifiers from TorchVision (maintainers & contributors, 2016). Fig. 6 plots FID (on 10k samples) versus classifier top-5 accuracy for each model, ordered by rank. Interestingly, the curve exhibits a clear V-shape: as accuracy decreases, FID initially improves, but beyond ranks 14–16 the FID degrades sharply. This suggests that neither very high nor very low classifier accuracy is optimal; a moderate level is most beneficial for FID.

Based on this observation, we choose `mobilenet_v3_large`, which consistently yields low FID at NFE = 3 and 9, as the classifier for parameter learning with both DiT (Peebles & Xie, 2023) and GM-DiT (Chen et al., 2025). A detailed analysis linking this pattern to precision and recall (Kynkäänniemi et al., 2019) is provided in Appendix D.

6.3 Parameter Interpolation across NFEs

From the learned parameters of Dual-Solver (Appendix F), we observe that their overall shapes remain similar across different NFEs. Motivated by this, we test the robustness of the learned parameters by applying them to other NFEs. To obtain the parameters for an unseen NFE, we interpolate the parameters of its two neighboring NFEs and then take their weighted average. The detailed formulas are provided in the Appendix G. Fig. 7 reports the results. *Interp. (3,5), (5,7), (7,9)* denotes that we interpolate between the parameter pairs at NFEs (3,5), (5,7), and (7,9) to obtain the parameters for the intermediate NFEs 4, 6, and 8, respectively; the other interpolation schemes are defined analogously. Although these interpolated parameters do not match the performance of parameters directly optimized for each NFE, the gaps are modest, and the resulting FID scores still outperform those of other solvers.

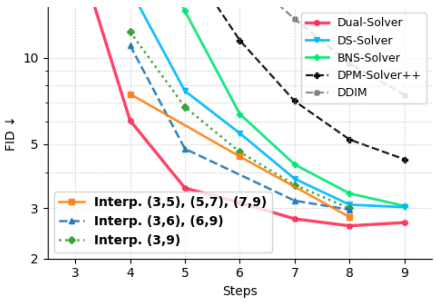


Figure 7: FID results for parameter interpolation across NFEs with a DiT (Peebles & Xie, 2023) backbone.

(a) DPM-Solver++ (Lu et al., 2022b) (b) Dual-Solver (Ours) (c) BNS-Solver (Shaul et al., 2024) (d) DS-Solver (Wang et al., 2025)

Figure 8: **Sampling results.** PixArt-α (Chen et al., 2023), NFE=5, CFG=3.5. See Fig. 22 for further results.

(a) Dual-Solver (Ours)

(b) BNS-Solver (Shaul et al., 2024)

(c) DS-Solver (Wang et al., 2025)

Figure 9: **Sampling results.** GM-DiT (Chen et al., 2025), NFE=3, CFG=1.4. See Fig. 20 for more.

6.4 Training Time

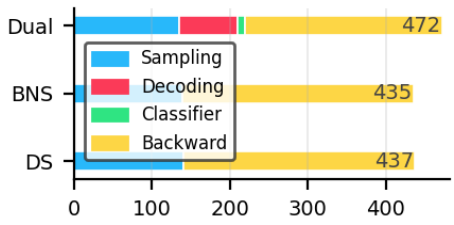


Figure 10: Training time per step (ms) (Table 10).

Fig. 10 reports per-step training time, decomposed into sampling, decoding, classifier, and backward, measured at NFE $= 5$ using GM-DiT (Chen et al., 2025) as the backbone. Dual-Solver adds decoding and classifier overheads; the latter is modest thanks to lightweight choices, as discussed in Sec. 6.2 (D). Overall, total per-step time increases by about 8% versus BNS-Solver (Shaul et al., 2024) and DS-Solver (Wang et al., 2025). See Appendix E for convergence-time details.

7 Conclusions and Limitations

This paper introduces Dual-Solver, a predictor–corrector sampler that achieves second-order numerical accuracy, featuring per-step learnable parameters (γ, τ, κ) that govern the prediction parameterization (noise/velocity/data), a change of variables, and a second-order residual adjustment. All solver parameters are optimized end-to-end with a classification-based objective using a pretrained image classifier. Across diverse diffusion and flow matching backbones, experiments show substantial improvements over competing solvers in the low-NFE regime ($3 \leq$ NFE ≤ 9), measured by both FID and CLIP score. Limitations include the absence of unconditional backbones and a lack of analysis beyond second-order accuracy; both are left for future work.

Reproducibility Statement

In Sec. 6 and Appendix C, we report detailed information regarding the backbones, solvers, training, and evaluation used in our experiments. All datasets employed, such as ImageNet (Deng et al., 2009) and MSCOCO (Lin et al., 2014), are publicly available under their respective licenses.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Francis Bashforth and John Couch Adams. *An Attempt to Test the Theories of Capillary Action by Comparing the Theoretical and Measured Forms of Drops of Fluid: With an Explanation of the Method of Integration Employed in Constructing the Tables*. Cambridge University Press, Cambridge, 1883.

John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

Hansheng Chen, Kai Zhang, Hao Tan, Zexiang Xu, Fujun Luan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. Gaussian mixture flow matching models. *arXiv preprint arXiv:2504.05304*, 2025.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv preprint arXiv:2310.00426*, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. IEEE, 2009.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Genie: Higher-order denoising diffusion solvers. *Advances in Neural Information Processing Systems*, 35:30150–30166, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 6626–6637, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:6840–6851, 2020.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022.

Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021. URL `https://doi.org/10.5281/zenodo.5143773`. If you use this software, please cite it as below.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019.

Daiqing Li, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi. Playground v2. 5: Three insights towards enhancing aesthetic quality in text-to-image generation. *arXiv preprint arXiv:2402.17245*, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022a.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022b.

TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Zhang Qinsheng and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *ICLR*, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Carl Runge. Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895. URL `http://eudml.org/doc/157756`.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Neta Shaul, Juan Perez, Ricky TQ Chen, Ali Thabet, Albert Pumarola, and Yaron Lipman. Bespoke solvers for generative flow models. *arXiv preprint arXiv:2310.19075*, 2023.

Neta Shaul, Uriel Singer, Ricky TQ Chen, Matthew Le, Ali Thabet, Albert Pumarola, and Yaron Lipman. Bespoke non-stationary solvers for fast sampling of diffusion and flow models. *arXiv preprint arXiv:2403.01329*, 2024.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021a.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in Neural Information Processing Systems*, 33, 2020.

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*, 2021b.

Vinh Tong, Trung-Dung Hoang, Anji Liu, Guy Van den Broeck, and Mathias Niepert. Learning to discretize denoising diffusion odes. In *International Conference on Learning Representations (ICLR)*, 2025.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679, 2020.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. `https://github.com/huggingface/diffusers`, 2022.

Shuai Wang, Zexian Li, Tianhui Song, Xubin Li, Tiezheng Ge, Bo Zheng, Limin Wang, et al. Differentiable solver search for fast diffusion sampling. *arXiv preprint arXiv:2505.21114*, 2025.

Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, et al. Sana: Efficient high-resolution image synthesis with linear diffusion transformers. *arXiv preprint arXiv:2410.10629*, 2024.

Shuchen Xue, Mingyang Yi, Weijian Luo, Shifeng Zhang, Jiacheng Sun, Zhenguo Li, and Zhi-Ming Ma. Sa-solver: Stochastic adams solver for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

We Zhao, Li Bai, Yong Rao, Jian Zhou, and Jun Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *arXiv preprint arXiv:2302.04867*, 2023.

Zhenyu Zhou, Defang Chen, Can Wang, and Chun Chen. Fast ode-based sampling for diffusion models in around 5 steps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7777–7786, 2024.

CONTENTS

A LLM USAGE

We disclose that Large Language Models (LLMs) were used as assistive tools for:

- verifying the consistency of mathematical derivations,
- assisting literature search and surfacing related work,
- helping with formal writing (style, grammar, typos),
- drafting figure and table scripts.

All scientific ideas, methods, and results originate from the authors, who take full responsibility for the content of this paper.

B DERIVATIONS

B.1 DERIVATION OF DUAL PREDICTION

Differential Form of Dual Prediction. The differential form corresponding to the integral form in Eq. 5 is given by

$$\boxed{\frac{d\boldsymbol{x}_t}{dt} = \beta \boldsymbol{x}_t + \alpha_t\Big(\frac{d\log\alpha_t}{dt} - \beta\Big)\boldsymbol{x}_\theta(\boldsymbol{x}_t, t) + \sigma_t\Big(\frac{d\log\sigma_t}{dt} - \beta\Big)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t).} \tag{9}$$

To derive this form, we start from the probability-flow ODE in Eq. 3:

$$\frac{d\boldsymbol{x}_t}{dt} = f_t\,\boldsymbol{x}_t - \frac{1}{2}\,g_t^2\,\nabla_{\boldsymbol{x}}\log q(\boldsymbol{x}_t),$$

$$\text{where } f_t = \frac{d\log\alpha_t}{dt}, \quad g_t^2 = \frac{d}{dt}\sigma_t^2 - 2\,f_t\,\sigma_t^2, \quad \nabla_{\boldsymbol{x}}\log q(\boldsymbol{x}_t) = -\frac{\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)}{\sigma_t}.$$

Substituting f_t, g_t^2, and $\nabla_{\boldsymbol{x}}\log q(\boldsymbol{x}_t)$ yields:

$$\begin{aligned}\frac{d\boldsymbol{x}_t}{dt} &= \frac{d\log\alpha_t}{dt}\,\boldsymbol{x}_t + \frac{1}{2\sigma_t}\left(\frac{d}{dt}\sigma_t^2 - 2\frac{d\log\alpha_t}{dt}\,\sigma_t^2\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\\ &= \frac{d\log\alpha_t}{dt}\,\boldsymbol{x}_t + \sigma_t\left(\frac{d\log\sigma_t}{dt} - \frac{d\log\alpha_t}{dt}\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t).\end{aligned}$$

Introducing an arbitrary $\beta \in \mathbb{R}$, we can rewrite:

$$\begin{aligned}\frac{d\boldsymbol{x}_t}{dt} &= \beta\boldsymbol{x}_t + \Big(\frac{d\log\alpha_t}{dt} - \beta\Big)\boldsymbol{x}_t + \sigma_t\left(\frac{d\log\sigma_t}{dt} - \frac{d\log\alpha_t}{dt}\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\\ &= \beta\boldsymbol{x}_t + \Big(\frac{d\log\alpha_t}{dt} - \beta\Big)\big(\alpha_t\boldsymbol{x}_\theta(\boldsymbol{x}_t, t) + \sigma_t\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\big) + \sigma_t\left(\frac{d\log\sigma_t}{dt} - \frac{d\log\alpha_t}{dt}\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\\ &= \beta\boldsymbol{x}_t + \alpha_t\Big(\frac{d\log\alpha_t}{dt} - \beta\Big)\boldsymbol{x}_\theta(\boldsymbol{x}_t, t) + \sigma_t\Big(\frac{d\log\sigma_t}{dt} - \beta\Big)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t).\end{aligned}$$

Thus we obtain the differential form of *dual prediction*. It is straightforward to verify that choosing $\beta = \frac{d}{dt}\log\alpha_t$ recovers the noise-prediction form, $\beta = \frac{d}{dt}\log\sigma_t$ recovers the data-prediction form, and $\beta = 0$ recovers the velocity-prediction form in Table 1.

Integral Form of Dual Prediction. We next apply the variation-of-constants method to Eq. 9 to obtain the following integral representation:

$$\boldsymbol{x}_{t_{i+1}} = \exp\Big(\textstyle\int_{t_i}^{t_{i+1}}\beta_u\,du\Big)\,\boldsymbol{x}_{t_i} \;+\; \int_{t_i}^{t_{i+1}}\exp\Big(\textstyle\int_s^{t_{i+1}}\beta_u\,du\Big)\Big[\alpha_s\Big(\frac{d\log\alpha_s}{ds} - \beta_s\Big)\,\boldsymbol{x}_\theta \;+\; \sigma_s\Big(\frac{d\log\sigma_s}{ds} - \beta_s\Big)\,\boldsymbol{\epsilon}_\theta\Big]\,ds.$$

We reparameterize β in terms of a new variable $\gamma \in \mathbb{R}$ as follows:

$$\beta(\gamma) = \begin{cases} \gamma \dfrac{d \log \sigma_t}{dt} \;=\; \gamma \dfrac{\dot{\sigma}_t}{\sigma_t}, & \gamma \geq 0, \\ -\gamma \dfrac{d \log \alpha_t}{dt} \;=\; -\gamma \dfrac{\dot{\alpha}_t}{\alpha_t}, & \gamma < 0. \end{cases}$$

Then we obtain the following γ-interpolated integral form of dual prediction:

$$\boxed{\boldsymbol{x}_{t_{i+1}} = \begin{cases} \left(\dfrac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma} \left[\displaystyle\int_{t_i}^{t_{i+1}} \frac{d}{dt}\big(\alpha_t\, \sigma_t^{-\gamma}\big)\, \boldsymbol{x}_\theta\, dt \;+\; \int_{t_i}^{t_{i+1}} \frac{d}{dt}\Big(\sigma_t^{1-\gamma}\Big)\, \boldsymbol{\epsilon}_\theta\, dt\right], & \gamma \geq 0, \\ \left(\dfrac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma} \boldsymbol{x}_{t_i} \;+\; \alpha_{t_{i+1}}^{-\gamma} \left[\displaystyle\int_{t_i}^{t_{i+1}} \frac{d}{dt}\Big(\alpha_t^{1+\gamma}\Big)\, \boldsymbol{x}_\theta\, dt \;+\; \int_{t_i}^{t_{i+1}} \frac{d}{dt}\big(\sigma_t\, \alpha_t^{\gamma}\big)\, \boldsymbol{\epsilon}_\theta\, dt\right], & \gamma < 0. \end{cases}}$$

(10)

B.2 Change of Variables

Using the transform L defined in Sec. 3.2, we define

$$u(t;\tau_u) = \begin{cases} L\big(\alpha_t\, \sigma_t^{-\gamma}; \tau_u\big), & \gamma \geq 0, \\ L\big(\alpha_t^{1+\gamma}; \tau_u\big), & \gamma < 0, \end{cases} \qquad v(t;\tau_v) = \begin{cases} L\big(\sigma_t^{1-\gamma}; \tau_v\big), & \gamma \geq 0, \\ L\big(\sigma_t\, \alpha_t^{\gamma}; \tau_v\big), & \gamma < 0. \end{cases}$$

Using these variables, applying a change of variables to the integral in Eq. 10 yields

$$\boxed{\boldsymbol{x}_{t_{i+1}} = \begin{cases} \left(\dfrac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma} \left[\displaystyle\int_{u_i}^{u_{i+1}} \frac{dL^{-1}(u;\tau_u)}{du}\, \boldsymbol{x}_\theta(u)\, du \;+\; \int_{v_i}^{v_{i+1}} \frac{dL^{-1}(v;\tau_v)}{dv}\, \boldsymbol{\epsilon}_\theta(v)\, dv\right] & (\gamma \geq 0), \\ \left(\dfrac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma} \boldsymbol{x}_{t_i} \;+\; \alpha_{t_{i+1}}^{-\gamma} \left[\displaystyle\int_{u_i}^{u_{i+1}} \frac{dL^{-1}(u;\tau_u)}{du}\, \boldsymbol{x}_\theta(u)\, du \;+\; \int_{v_i}^{v_{i+1}} \frac{dL^{-1}(v;\tau_v)}{dv}\, \boldsymbol{\epsilon}_\theta(v)\, dv\right] & (\gamma < 0). \end{cases}}$$

(11)

B.3 Derivation of Sampling Equations

B.3.1 Derivation of First-Order Predictor

First, we take a first-order approximation of $\boldsymbol{x}_\theta$ and $\boldsymbol{\epsilon}_\theta$ in Eq. 11. For the case $\gamma \geq 0$, this yields

$$\boldsymbol{x}_{t_{i+1}} = \left(\tfrac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma} \Big[\boldsymbol{x}_\theta(u_i)\, \big(\Delta L^{-1}(u_i;\tau_u) + O((\Delta u_i)^2)\big) + \boldsymbol{\epsilon}_\theta(v_i)\, \big(\Delta L^{-1}(v_i;\tau_v) + O((\Delta v_i)^2)\big)\Big] \quad (\gamma \geq 0).$$

(12)

Here, we write $\boldsymbol{x}_\theta(u_i) \coloneqq \boldsymbol{x}_\theta(\boldsymbol{x}_{u^{-1}(u_i)}, u^{-1}(u_i))$, and $\Delta L^{-1}(u_i;\tau_u) \coloneqq L^{-1}(u_{i+1};\tau_u) - L^{-1}(u_i;\tau_u)$; the definitions for $\boldsymbol{\epsilon}_\theta(v_i)$ and $\Delta L^{-1}(v_i;\tau_v)$ are analogous.

Next, while preserving first-order accuracy, we incorporate the $B(\Delta u_i;\kappa_u) = \mathcal{O}((\Delta u_i)^2)$ and $B(\Delta v_i;\kappa_v) = \mathcal{O}((\Delta v_i)^2)$ correction terms and include the $\gamma < 0$ case, which yields the following first-order predictor of Dual-Solver.

$$\boxed{\boldsymbol{x}_{t_{i+1}}^{\mathrm{1st-pred.}} = \begin{cases} \left(\dfrac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma} \Big[\boldsymbol{x}_\theta(u_i)\big(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\big) + \boldsymbol{\epsilon}_\theta(v_i)\big(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\big)\Big], & \gamma \geq 0, \\ \left(\dfrac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma} \boldsymbol{x}_{t_i} + \alpha_{t_{i+1}}^{-\gamma} \Big[\boldsymbol{x}_\theta(u_i)\big(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\big) + \boldsymbol{\epsilon}_\theta(v_i)\big(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\big)\Big], & \gamma < 0. \end{cases}}$$

(13)

B.3.2 Derivation of Second-Order Predictor

First, we approximate $\boldsymbol{x}_\theta(u)$ and $\boldsymbol{\epsilon}_\theta(v)$ near u_i and v_i by a second-order backward-difference expansion. For the case $\gamma \geq 0$, this yields

$$\begin{aligned} \boldsymbol{x}_{t_{i+1}} \approx \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma} \Bigg[& \boldsymbol{x}_\theta(u_i) \Delta L^{-1}(u_i;\tau_u) + \frac{\Delta \boldsymbol{x}_\theta(u_{i-1})}{\Delta u_{i-1}} \int_{u_i}^{u_{i+1}} (u - u_i) \frac{dL^{-1}(u;\tau_u)}{du} du \\ & + \boldsymbol{\epsilon}_\theta(v_i) \Delta L^{-1}(v_i;\tau_v) + \frac{\Delta \boldsymbol{\epsilon}_\theta(v_{i-1})}{\Delta v_{i-1}} \int_{v_i}^{v_{i+1}} (v - v_i) \frac{dL^{-1}(v;\tau_v)}{dv} dv \Bigg] \quad (\gamma \geq 0). \end{aligned}$$

(14)

Using

$$\frac{1}{\Delta u_{i-1}}\int_{u_i}^{u_{i+1}}(u-u_i)\frac{dL^{-1}(u;\tau_u)}{du}\,du = \frac{1}{2r_i^{(u)}}\left(\left.\frac{dL^{-1}(u;\tau_u)}{du}\right|_{u_i}\Delta u_i + O((\Delta u_i)^2)\right) = \frac{1}{2r_i^{(u)}}\left(\Delta L^{-1}(u_i;\tau_u) + \mathcal{O}((\Delta u_i)^2)\right) \tag{15}$$

where $r_i^{(u)} := \frac{\Delta u_{i-1}}{\Delta u_i}$ and $r_i^{(v)} := \frac{\Delta v_{i-1}}{\Delta v_i}$, we obtain

$$\boldsymbol{x}_{t_{i+1}} \approx \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma}\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta(u_i)\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta\boldsymbol{x}_\theta(u_{i-1})}{2r_i^{(u)}}\left(\Delta L^{-1}(u_i;\tau_u) + \mathcal{O}((\Delta u_i)^2)\right) + \boldsymbol{\epsilon}_\theta(v_i)\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta\boldsymbol{\epsilon}_\theta(v_{i-1})}{2r_i^{(v)}}\left(\Delta L^{-1}(v_i;\tau_v) + \mathcal{O}((\Delta v_i)^2)\right)\Big] \quad (\gamma \geq 0). \tag{16}$$

Next, while preserving second-order accuracy, we incorporate the $B(\Delta u_i;\kappa_u) = \mathcal{O}((\Delta u_i)^2)$ and $B(\Delta v_i;\kappa_v) = \mathcal{O}((\Delta v_i)^2)$ correction terms and include the $\gamma < 0$ case, which yields the following second-order predictor of Dual-Solver.

$$\boxed{\boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-pred.}} = \begin{cases} \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma}\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta\,\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta\boldsymbol{x}_\theta(u_{i-1})}{2r_i^{(u)}}\left(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\right) \\ \qquad + \boldsymbol{\epsilon}_\theta\,\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta\boldsymbol{\epsilon}_\theta(v_{i-1})}{2r_i^{(v)}}\left(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\right)\Big],\ \gamma \geq 0, \\ \left(\frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma}\boldsymbol{x}_{t_i} + \alpha_{t_{i+1}}^{-\gamma}\Big[\boldsymbol{x}_\theta\,\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta\boldsymbol{x}_\theta(u_{i-1})}{2r_i^{(u)}}\left(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\right) \\ \qquad + \boldsymbol{\epsilon}_\theta\,\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta\boldsymbol{\epsilon}_\theta(v_{i-1})}{2r_i^{(v)}}\left(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\right)\Big],\ \gamma < 0. \end{cases}} \tag{17}$$

B.3.3 Derivation of Second-Order Corrector

First, we approximate $\boldsymbol{x}_\theta(u)$ and $\boldsymbol{\epsilon}_\theta(v)$ near u_i and v_i by a forward-difference expansion. For the case $\gamma \geq 0$, this yields

$$\boldsymbol{x}_{t_{i+1}} \approx \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma}\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta(u_i)\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta\boldsymbol{x}_\theta(u_i)}{\Delta u_i}\int_{u_i}^{u_{i+1}}(u-u_i)\frac{dL^{-1}(u;\tau_u)}{du}du + \boldsymbol{\epsilon}_\theta(v_i)\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta\boldsymbol{\epsilon}_\theta(v_i)}{\Delta v_i}\int_{v_i}^{v_{i+1}}(v-v_i)\frac{dL^{-1}(v;\tau_v)}{dv}dv\Big] \quad (\gamma \geq 0). \tag{18}$$

Using

$$\frac{1}{\Delta u_i}\int_{u_i}^{u_{i+1}}(u-u_i)\frac{dL^{-1}(u;\tau_u)}{du}\,du = \frac{1}{2}\left(\left.\frac{dL^{-1}(u;\tau_u)}{du}\right|_{u_i}\Delta u_i + \mathcal{O}((\Delta u_i)^2)\right) = \frac{1}{2}\left(\Delta L^{-1}(u_i;\tau_u) + \mathcal{O}((\Delta u_i)^2)\right) \tag{19}$$

we obtain

$$\boldsymbol{x}_{t_{i+1}} \approx \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma}\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta(u_i)\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta\boldsymbol{x}_\theta(u_i)}{2}\left(\Delta L^{-1}(u_i;\tau_u) + \mathcal{O}((\Delta u_i)^2)\right) + \boldsymbol{\epsilon}_\theta(v_i)\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta\boldsymbol{\epsilon}_\theta(v_i)}{2}\left(\Delta L^{-1}(v_i;\tau_v) + \mathcal{O}((\Delta v_i)^2)\right)\Big] \quad (\gamma \geq 0). \tag{20}$$

Next, while preserving second-order accuracy, we incorporate the $B(\Delta u_i;\kappa_u) = \mathcal{O}((\Delta u_i)^2)$ and $B(\Delta v_i;\kappa_v) = \mathcal{O}((\Delta v_i)^2)$ correction terms and include the $\gamma < 0$ case, which yields the following second-order corrector of Dual-Solver.

$$\boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}} = \begin{cases} \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} \;+\; \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta\,\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta \boldsymbol{x}_\theta(u_i)}{2}\left(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\right) \\ \qquad\qquad + \boldsymbol{\epsilon}_\theta\,\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta \boldsymbol{\epsilon}_\theta(v_i)}{2}\left(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\right)\Big],\ \gamma \geq 0, \\ \left(\frac{\alpha_{t_{i+1}}}{\alpha_{t_i}}\right)^{-\gamma} \boldsymbol{x}_{t_i} \;+\; \alpha_{t_{i+1}}^{-\gamma}\Big[\boldsymbol{x}_\theta\,\Delta L^{-1}(u_i;\tau_u) + \frac{\Delta \boldsymbol{x}_\theta(u_i)}{2}\left(\Delta L^{-1}(u_i;\tau_u) + B(\Delta u_i;\kappa_u)\right) \\ \qquad\qquad + \boldsymbol{\epsilon}_\theta\,\Delta L^{-1}(v_i;\tau_v) + \frac{\Delta \boldsymbol{\epsilon}_\theta(v_i)}{2}\left(\Delta L^{-1}(v_i;\tau_v) + B(\Delta v_i;\kappa_v)\right)\Big],\ \gamma < 0. \end{cases} \tag{21}$$

B.4 Proof of Local Truncation Error

Proof. We begin by rewriting the exact solution in Eq. 6 as follows:

$$\boldsymbol{x}_{t_{i+1}} = \left(\tfrac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma}\boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big(I_x + I_\epsilon\Big),$$

with

$$I_x := \int_{u_i}^{u_{i+1}} (L^{-1})'(u)\,\boldsymbol{x}_\theta(u)\,du, \quad I_\epsilon := \int_{v_i}^{v_{i+1}} (L^{-1})'(v)\,\boldsymbol{\epsilon}_\theta(v)\,dv.$$

We then demonstrate the accuracy of the proposed second-order corrector $\boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}}$ in equation 8 by using the approximation for I_x and I_ϵ with third-order accuracy.

We take second–order Taylor expansions of $\boldsymbol{x}_\theta(u)$ and $(L^{-1})'(u)$ at u_i:

$\boldsymbol{x}_\theta(u) = \boldsymbol{x}_\theta(u_i) + \boldsymbol{x}'_\theta(u_i)s + \frac{1}{2}\,\boldsymbol{x}''_\theta(\xi_u)s^2, \quad (L^{-1})'(u) = (L^{-1})'(u_i) + (L^{-1})''(u_i)s + \frac{1}{2}\,(L^{-1})^{(3)}(\zeta_u)s^2,$

where $s = u - u_i$ and $\xi_u, \zeta_u \in (u_i, u_{i+1})$. Then the integral I_x can be represented in terms of the integral of s as

$$I_x = \int_0^{\Delta u_i} \big(A_0 + A_1 s + \mathcal{O}(s^2)\big)\,ds = A_0\,\Delta u_i + \tfrac{1}{2}A_1\,(\Delta u_i)^2 + \mathcal{O}((\Delta u_i)^3)$$

with

$$A_0 := (L^{-1})'(u_i)\boldsymbol{x}_\theta(u_i), \;\; A_1 := (L^{-1})'(u_i)\boldsymbol{x}'_\theta(u_i) + (L^{-1})''(u_i)\boldsymbol{x}_\theta(u_i),$$

and $\Delta L^{-1}(u_i) := L^{-1}(u_{i+1}) - L^{-1}(u_i)$. Furthermore, I_x can be expressed in terms of $\boldsymbol{x}_\theta(u_i)$ and its derivative as

$$I_x = \boldsymbol{x}_\theta(u_i)\underbrace{\big((L^{-1})'(u_i)\Delta u_i + \tfrac{1}{2}(L^{-1})''(u_i)(\Delta u_i)^2\big)}_{=\,\Delta L^{-1}(u_i)+\mathcal{O}((\Delta u_i)^3)} + \boldsymbol{x}'_\theta(u_i)\,(L^{-1})'(u_i)\,\tfrac{1}{2}(\Delta u_i)^2 + \mathcal{O}((\Delta u_i)^3)$$

$$= \boldsymbol{x}_\theta(u_i)\Delta L^{-1}(u_i) + \boldsymbol{x}'_\theta(u_i)\,(L^{-1})'(u_i)\,\tfrac{1}{2}(\Delta u_i)^2 + \mathcal{O}((\Delta u_i)^3).$$

From the definition of second-order corrector approximation in equation 8, let us denote the last part of the approximation with $\widetilde{I}_x$ and $\widetilde{I}_\epsilon$ as

$$\widetilde{I}_x = \boldsymbol{x}_\theta(u_i)\,\Delta L^{-1}(u_i) + \tfrac{1}{2}\Delta\boldsymbol{x}_\theta(u_i)\Big(\Delta L^{-1}(u_i) + B_x(\Delta u_i)\Big),$$

$$\widetilde{I}_\epsilon = \boldsymbol{\epsilon}_\theta(v_i)\,\Delta L^{-1}(v_i) + \tfrac{1}{2}\,\Delta\boldsymbol{\epsilon}_\theta(v_i)\Big(\Delta L^{-1}(v_i) + B_\epsilon(\Delta v_i)\Big).$$

Here, we remark that $B_x(\Delta u_i) = \mathcal{O}((\Delta u_i)^2)$ and $B_\epsilon(\Delta v_i) = \mathcal{O}((\Delta v_i)^2)$. Using the fact

$$\Delta\boldsymbol{x}_\theta(u_i) = \boldsymbol{x}_\theta(u_{i+1}) - \boldsymbol{x}_\theta(u_i) = \boldsymbol{x}'_\theta(u_i)\Delta u_i + \mathcal{O}((\Delta u_i)^2),$$

$\widetilde{I}_x$ can be rewritten as

$$\widetilde{I}_x = \boldsymbol{x}_\theta(u_i)\,\Delta L^{-1}(u_i) + \boldsymbol{x}'_\theta(u_i)\,\tfrac{1}{2}\Delta u_i\Big(\Delta L^{-1}(u_i) + B_x(\Delta u_i)\Big) + \mathcal{O}((\Delta u_i)^3).$$

Then

$$I_x - \widetilde{I}_x = \tfrac{1}{2}\boldsymbol{x}'_\theta(u_i)\Big[(L^{-1})'(u_i)(\Delta u_i)^2 - \Delta L^{-1}(u_i)\,\Delta u_i\Big] - \tfrac{1}{2}\,\boldsymbol{x}'_\theta(u_i)\,B_x(\Delta u_i)\Delta u_i + \mathcal{O}((\Delta u_i)^3).$$

Since $\Delta L^{-1}(u_i) = (L^{-1})'(u_i)\Delta u_i + \mathcal{O}((\Delta u_i)^2)$ and $B_x(\Delta u_i) = \mathcal{O}((\Delta u_i)^2)$, we conclude that $I_x - \widetilde{I}_x = \mathcal{O}((\Delta u_i)^3)$. By the same argument, $I_\epsilon - \widetilde{I}_\epsilon = \mathcal{O}((\Delta v_i)^3)$. Therefore, it follows that

$$\boldsymbol{x}_{t_{i+1}}^{\mathrm{exact}} - \boldsymbol{x}_{t_{i+1}}^{\mathrm{2nd-corr.}} = \sigma_{t_{i+1}}^{\gamma}\big[(I_x - \widetilde{I}_x) + (I_\epsilon - \widetilde{I}_\epsilon)\big] = \mathcal{O}((\Delta u_i)^3 + (\Delta v_i)^3).$$

□

C Experimental Details

C.1 Setup

Environment details We run all experiments on a single NVIDIA RTX pro 6000 (Driver 575.57.08) under Ubuntu 24.04 with Python 3.11.13, PyTorch 2.8.0, and CUDA 12.9.

Backbone Details We evaluate DiT-XL/2 (Peebles & Xie, 2023), GM-DiT (Chen et al., 2025), SANA (Xie et al., 2024), and PixArt-α (Chen et al., 2023). All models are obtained via the Hugging Face `Diffusers` (von Platen et al., 2022) pipelines and run in evaluation mode with `bfloat16`. The model identifiers are:

- DiT: `facebook/DiT-XL-2-256`
- GM-DiT: `Lakonik/gmflow_imagenet_k8_ema`
- SANA: `Efficient-Large-Model/Sana_600M_512px_diffusers`
- PixArt-α: `PixArt-alpha/PixArt-XL-2-512x512`

Solver Details The solvers used in our experiments are the diffusion-dedicated DDIM (Song et al., 2021a), the second-order multistep DPM-Solver++ (Lu et al., 2022b), and the learned solvers BNS-Solver (Shaul et al., 2024) and DS-Solver (Wang et al., 2025). We use DDIM as the first-order counterpart of DPM-Solver++ (as proposed in Lu et al. (2022a)). Implementations of DPM-Solver++ and DS-Solver are taken from their official GitHub repositories[1,2], and BNS-Solver is implemented according to the paper.

Learning Details We train with AdamW (Loshchilov & Hutter, 2017) using $\beta = (0.9, 0.999)$, $\epsilon = 1\text{e}{-8}$, and weight decay 0.01. The learning rate decays from 2×10^{-3} to 1×10^{-4} over 20k steps via cosine annealing (Loshchilov & Hutter, 2016). The batch size is fixed to 10 for all experiments. For regression-based learning, the teacher trajectory is a 200-step DDIM Song et al. (2021a) method on 1k samples.

Sampling Details We use an NFE grid of $\{3, 4, 5, 6, 7, 8, 9\}$ for all backbones. Classifier-free guidance (CFG; Ho & Salimans, 2021) is fixed per backbone as follows: DiT $= 1.5$, GM-DiT $= 1.4$, SANA $= 4.5$, and PixArt-$\alpha = 3.5$. For FID, we use the publicly released ImageNet training-set statistics (Dhariwal & Nichol, 2021), while for SANA and PixArt-α we compute the FID statistics from 30k samples drawn from the MSCOCO 2014 (Lin et al., 2014) evaluation set (Lin et al., 2014); CLIP is computed with the official RN101 variant (Radford et al., 2021). We generate $50{,}000$ images for ImageNet and $30{,}000$ images for text-to-image (MSCOCO 2014 (Lin et al., 2014) evaluation prompts).

C.2 Quantitative Results

Table 4: **DiT** (Peebles & Xie, 2023): FID (↓) vs. NFE. ImageNet generation, 50k samples.

Method	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DDIM (Song et al., 2021a)	89.33	56.33	32.91	20.06	13.64	9.55	7.42
DPM-Solver++(Lu et al., 2022b)	88.46	47.64	22.19	11.49	7.06	5.19	4.43
BNS-Solver(Shaul et al., 2024)	103.26	38.20	14.53	6.37	4.25	3.37	3.05
DS-Solver(Wang et al., 2025)	67.31	17.31	7.66	5.46	3.79	3.08	3.02
Dual-Solver (Ours)	**24.91**	**6.05**	**3.52**	**3.13**	**2.75**	**2.60**	**2.67**

[1] `https://github.com/LuChengTHU/dpm-solver`
[2] `https://github.com/MCG-NJU/NeuralSolver`

Table 5: **GM-DiT** Chen et al. (2025): FID (↓) vs. NFE. ImageNet generation, 50k samples.

Method	3	4	5	6	7	8	9
DDIM (Song et al., 2021a)	70.15	35.98	19.70	12.55	9.03	7.01	5.78
DPM-Solver++(Lu et al., 2022b)	63.24	19.53	7.85	4.74	3.65	3.15	2.89
BNS-Solver(Shaul et al., 2024)	57.88	26.64	10.31	5.02	3.43	2.66	2.44
DS-Solver(Wang et al., 2025)	34.15	11.60	5.64	3.49	**2.70**	2.48	2.41
Dual-Solver-R (Ours)	45.53	14.43	7.49	3.75	2.77	**2.44**	**2.32**
Dual-Solver-C (Ours)	**6.81**	**3.76**	**3.09**	**2.97**	2.77	2.70	2.60

R = trajectory regression-based; C = classification-based.

Table 6: **PixArt-α** (Chen et al., 2023): FID (↓) and CLIP score (↑) vs. NFE. Text-to-image on MSCOCO 2014 (Lin et al., 2014) with 30k samples.

Method	3	4	5	6	7	8	9
FID (↓)							
DDIM (Song et al., 2021a)	71.37	45.21	35.12	30.92	28.60	27.39	26.58
DPM-Solver++ (Lu et al., 2022b)	76.01	41.77	31.48	27.83	26.56	25.82	25.48
BNS-Solver (Shaul et al., 2024)	125.65	66.94	41.31	32.55	28.55	25.18	24.15
DS-Solver (Wang et al., 2025)	118.01	66.09	43.62	29.74	28.25	26.57	25.22
Dual-Solver (Ours)	**66.61**	**31.61**	**24.68**	**22.39**	**22.51**	**21.96**	**22.01**
CLIP (RN101, ↑)							
DDIM (Song et al., 2021a)	0.4469	0.4670	0.4739	0.4764	0.4763	0.4778	0.4779
DPM-Solver++ (Lu et al., 2022b)	0.4422	0.4676	0.4746	0.4763	0.4768	0.4771	0.4771
BNS-Solver (Shaul et al., 2024)	0.4320	0.4582	0.4694	0.4733	0.4746	0.4755	0.4757
DS-Solver (Wang et al., 2025)	0.4303	0.4568	0.4692	0.4748	0.4754	0.4762	0.4764
Dual-Solver (Ours)	**0.4499**	**0.4732**	**0.4784**	**0.4803**	**0.4806**	**0.4814**	**0.4815**

Table 7: **SANA** (Xie et al., 2024): FID (↓) and CLIP score (RN101, ↑) vs. NFE. Text-to-image on MSCOCO 2014 (Lin et al., 2014) with 30k samples.

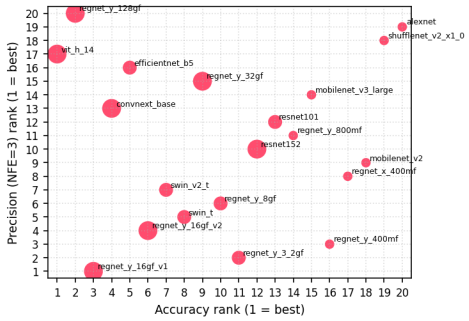
Method	3	4	5	6	7	8	9
FID (↓)							
DDIM (Song et al., 2021a)	45.05	27.72	23.93	23.06	22.99	22.96	22.97
DPM-Solver++ (Lu et al., 2022b)	45.33	26.12	22.56	22.48	22.79	22.90	23.11
BNS-Solver (Shaul et al., 2024)	48.16	26.37	21.04	20.66	20.79	21.13	21.64
DS-Solver (Wang et al., 2025)	48.65	29.15	21.66	20.65	21.43	21.80	22.27
Dual-Solver (Ours)	**21.79**	**19.40**	**18.81**	**18.52**	**19.57**	**19.43**	**19.77**
CLIP (↑)							
DDIM (Song et al., 2021a)	0.4656	0.4782	0.4813	0.4821	0.4824	0.4824	0.4824
DPM-Solver++ (Lu et al., 2022b)	0.4652	0.4779	0.4813	0.4821	0.4822	0.4820	0.4820
BNS-Solver (Shaul et al., 2024)	0.4651	0.4772	0.4800	0.4807	0.4808	0.4809	0.4808
DS-Solver (Wang et al., 2025)	0.4635	0.4765	0.4808	0.4816	0.4818	0.4818	0.4816
Dual-Solver (Ours)	**0.4795**	**0.4801**	**0.4821**	**0.4837**	**0.4843**	**0.4844**	**0.4849**

D CLASSIFIER ACCURACY VS. SAMPLE QUALITY

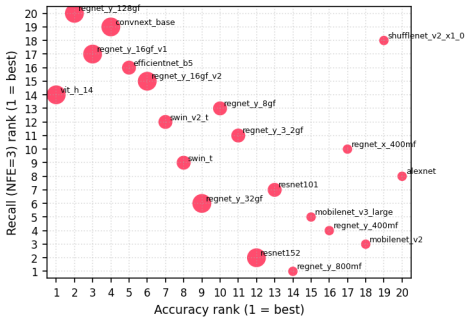
As examined in Sec. 6.2, we study the relationship between classifier accuracy and FID. In this section, Table 8 provides detailed numerical results, and Fig. 11 analyzes the trend from the perspectives of precision and recall (Kynkäänniemi et al., 2019). Table 8 reports the pretrained weights from TorchVision (maintainers & contributors, 2016) and the results of training Dual-Solver with each set of weights. Using a GM-DiT (Chen et al., 2025) backbone, we present FID, precision, and recall at NFE= 3 and 9. The best value for each metric is highlighted in bold. In Fig. 12, we show NFE=3 samples from Dual-Solver trained with the classifiers listed in Table 8, ordered by increasing FID (lower is better).

Table 8: **ImageNet per-classifier metrics.** Top-5 accuracy, FID, precision/recall at NFE=3 and 9, and GFLOPs; FID is measured on 10k samples after training Dual-Solver for each classifier.

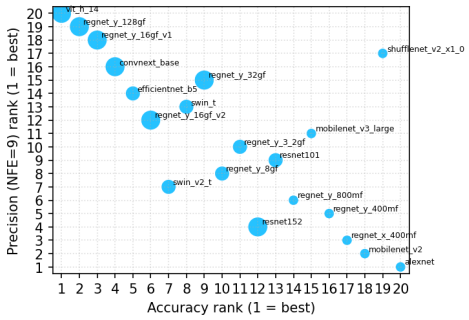
Weights[3]	Top-5 Acc. (%)	FID@3	FID@9	Precision@3	Precision@9	Recall@3	Recall@9	GFLOPS
ViT_H_14_Weights.IMAGENET1K_SWAG_E2E_V1	**98.694**	10.71	6.72	0.8376	0.8960	0.7420	0.7311	**1016.72**
RegNet_Y_128GF_Weights.IMAGENET1K_SWAG_LINEAR_V1	97.844	12.99	5.61	0.8230	0.8996	0.7327	0.7336	127.52
RegNet_Y_16GF_Weights.IMAGENET1K_SWAG_LINEAR_V1	97.244	9.97	5.32	**0.8573**	0.9007	0.7392	0.7467	15.91
ConvNeXt_Base_Weights.IMAGENET1K_V1	96.870	10.28	5.78	0.8529	0.9040	0.7359	0.7327	15.36
EfficientNet_B5_Weights.IMAGENET1K_V1	96.628	11.53	6.01	0.8379	0.9048	0.7393	0.7310	10.27
RegNet_Y_16GF_Weights.IMAGENET1K_V2	96.328	9.63	5.21	0.8559	0.9061	0.7396	0.7455	15.91
SwinV2_T_Weights.IMAGENET1K_V1	96.132	9.73	5.34	0.8555	0.9078	0.7439	0.7410	5.94
Swin_T_Weights.IMAGENET1K_V1	95.776	9.65	5.29	0.8559	0.9055	0.7448	0.7425	4.49
RegNet_Y_32GF_Weights.IMAGENET1K_V1	95.340	9.55	5.04	0.8520	0.9043	0.7464	0.7455	32.28
RegNet_Y_8GF_Weights.IMAGENET1K_V1	95.048	9.54	5.10	0.8559	0.9074	0.7438	0.7414	8.47
RegNet_Y_3_2GF_Weights.IMAGENET1K_V1	94.576	9.50	5.01	0.8564	0.9065	0.7448	0.7436	3.18
ResNet152_Weights.IMAGENET1K_V1	94.046	9.49	5.03	0.8537	0.9089	0.7493	0.7459	11.51
ResNet101_Weights.IMAGENET1K_V1	93.546	9.59	4.98	0.8529	0.9067	0.7459	0.7454	7.80
RegNet_Y_800MF_Weights.IMAGENET1K_V1	93.136	**9.40**	4.99	0.8532	0.9081	**0.7508**	0.7437	0.83
MobileNet_V3_Large_Weights.IMAGENET1K_V2	92.566	9.44	**4.87**	0.8523	0.9062	0.7467	0.7475	0.22
RegNet_Y_400MF_Weights.IMAGENET1K_V1	91.716	9.50	**4.87**	0.8560	0.9082	0.7470	**0.7489**	0.40
RegNet_X_400MF_Weights.IMAGENET1K_V1	90.950	9.60	4.93	0.8553	0.9091	0.7448	0.7410	0.41
MobileNet_V2_Weights.IMAGENET1K_V1	90.286	9.52	5.03	0.8541	0.9118	0.7471	0.7331	0.30
ShuffleNet_V2_X1_0_Weights.IMAGENET1K_V1	88.316	11.57	5.51	0.8368	0.9017	0.7388	0.7436	0.14
AlexNet_Weights.IMAGENET1K_V1	79.066	11.20	6.22	0.8300	**0.9163**	0.7455	0.7187	0.71



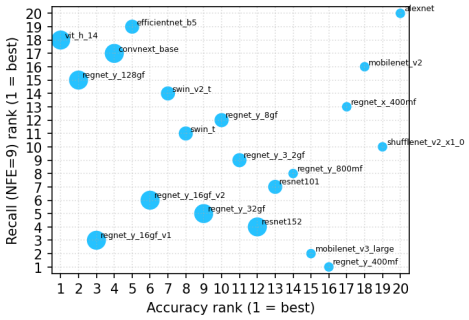
(a) Accuracy vs. Precision @ NFE= 3



(b) Accuracy vs. Recall @ NFE= 3



(c) Accuracy vs. Precision @ NFE= 9



(d) Accuracy vs. Recall @ NFE= 9

Figure 11: **Accuracy vs. precision/recall.** We select 20 TorchVision classifiers sorted by accuracy, learn the Dual-Solver for each, and report precision/recall at NFE=3 and 9 on 10k samples.

[3] https://docs.pytorch.org/vision/main/models.html

Classifier accuracy vs. precision and recall. Fig. 11 (a) and (b) plot precision and recall versus classifier accuracy at NFE= 3, respectively. The results show little relationship between accuracy and precision, while accuracy versus recall follows the V-shape seen in Fig. 6. In other words, neither very high nor very low accuracy helps recall; a moderate level yields higher recall. The pattern differs at NFE= 9. Fig. 11 (c) and (d) plot precision and recall versus classifier accuracy at NFE= 9. Unlike the NFE= 3 case, precision exhibits a strong negative correlation with accuracy, and accuracy appears largely unrelated to recall.

OpenCLIP accuracy vs. FID. Table 9 reports the FID evaluation used to select the CLIP model for learning Dual-Solver on the text-to-image task. All weights are from OpenCLIP (Ilharco et al., 2021) and are available from the official repository[4]. Using the SANA (Xie et al., 2024) backbone we trained the Dual-Solver for 20k steps. At NFE=3 and NFE=6, we generated 10k samples with MSCOCO 2014 (Lin et al., 2014) prompts and measured FID on the evaluation set. Based on these results, we chose the RN101 weights—which achieved an FID of 18.52 at NFE=6—as the model for learning the Dual-Solver in the main text-to-image experiment. Notably, this result also indicates that models with somewhat lower classification accuracy can yield lower FID.

Table 9: **OpenCLIP per-classifier metrics.** MSCOCO accuracy, FID at NFE=3 and 6, and GFLOPs; FID is measured on 30k samples after learning Dual-Solver for each classifier.

Weights	MSCOCO Acc. (%)	FID@3	FID@6	GFLOPs
ViT-H-14-378-quickgelu, dfn5b	**63.76**	23.98	23.28	**1054.05**
coca_ViT-L-14, mscoco_finetuned_laion2b_s13b_b90k	60.28	23.09	21.22	214.52
EVA02-E-14, laion2b_s4b_b115k	58.92	22.41	21.12	1007.93
convnext_xxlarge, laion2b_s34b_b82k_augreg	58.34	21.05	20.86	800.88
ViT-B-16-SigLIP-256, webli	57.24	21.03	19.87	57.84
EVA02-L-14-336, merged2b_s6b_b61k	56.05	23.15	23.28	167.50
ViT-L-14, commonpool_xl_laion_s13b_b90k	55.13	23.46	22.81	175.33
convnext_base_w, laion_aesthetic_s13b_b82k	52.38	20.97	19.86	49.38
convnext_base_w_320, laion_aesthetic_s13b_b82k_augreg	51.42	**20.69**	20.26	175.33
ViT-B-16-plus-240, laion400m_e32	49.79	21.66	21.18	64.03
ViT-B-32, laion2b_e16	47.68	23.75	23.49	14.78
ViT-B-32-quickgelu, metaclip_fullcc	46.62	22.46	21.42	14.78
RN50x16, openai	45.38	22.49	21.36	33.34
ViT-B-32, laion400m_e31	43.27	22.40	21.70	14.78
RN101, openai	40.25	21.78	**18.52**	25.50
ViT-B-16, commonpool_l_text_s1b_b8k	37.30	23.50	23.54	41.09
ViT-B-16, commonpool_l_s1b_b8k	28.55	22.94	24.73	41.09
ViT-B-32, commonpool_m_text_s128m_b4k	14.52	22.39	22.55	14.78
ViT-B-32, commonpool_s_clip_s13m_b4k	2.24	22.32	21.31	14.78
coca_ViT-B-32, mscoco_finetuned_laion2b_s13b_b90k	0.60	23.76	23.14	33.34

[4] https://github.com/mlfoundations/open_clip

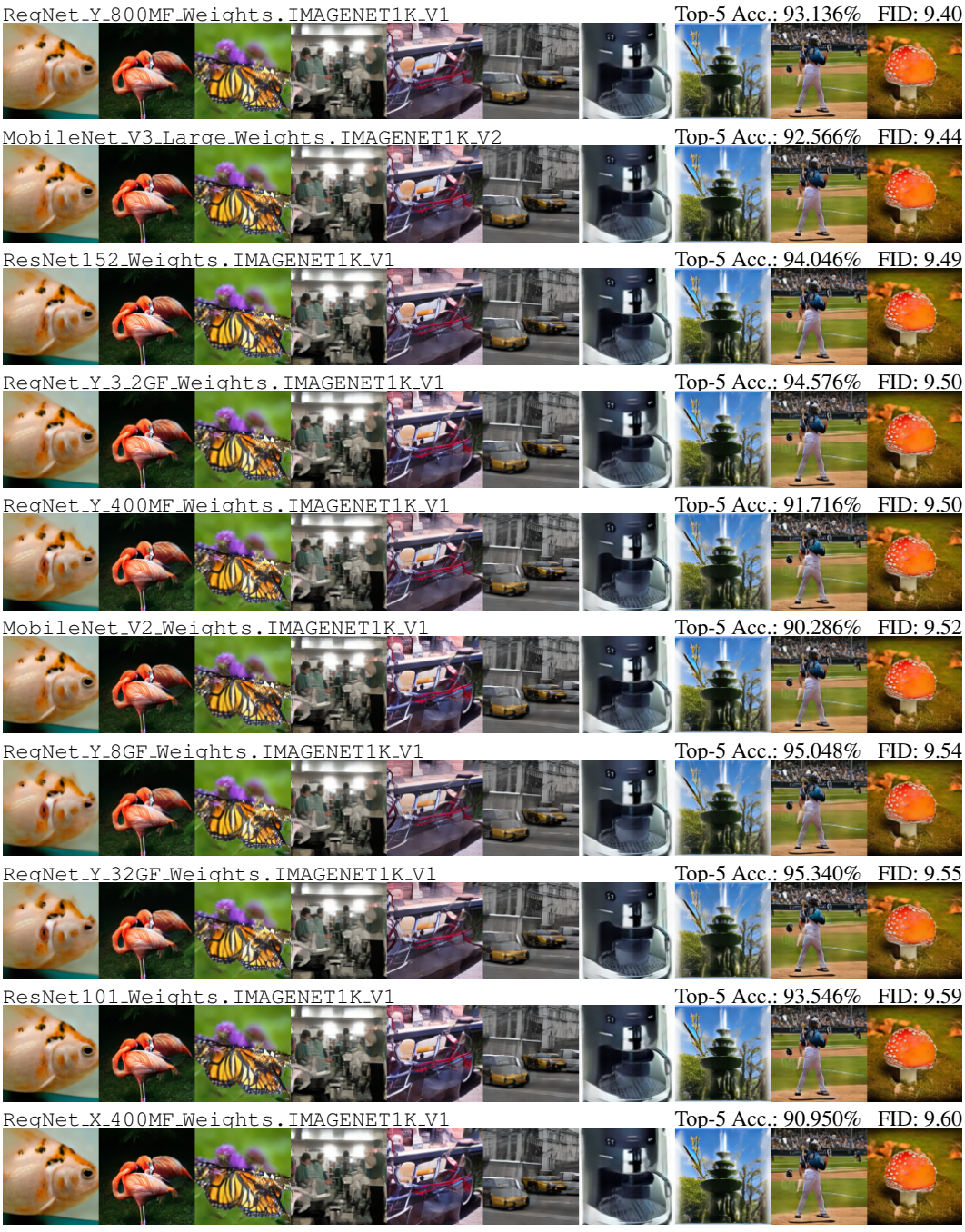


Figure 12: **Sampling results by classifier.** Classifier weights, top-5 accuracy, and FID are reported; entries are sorted by ascending FID.

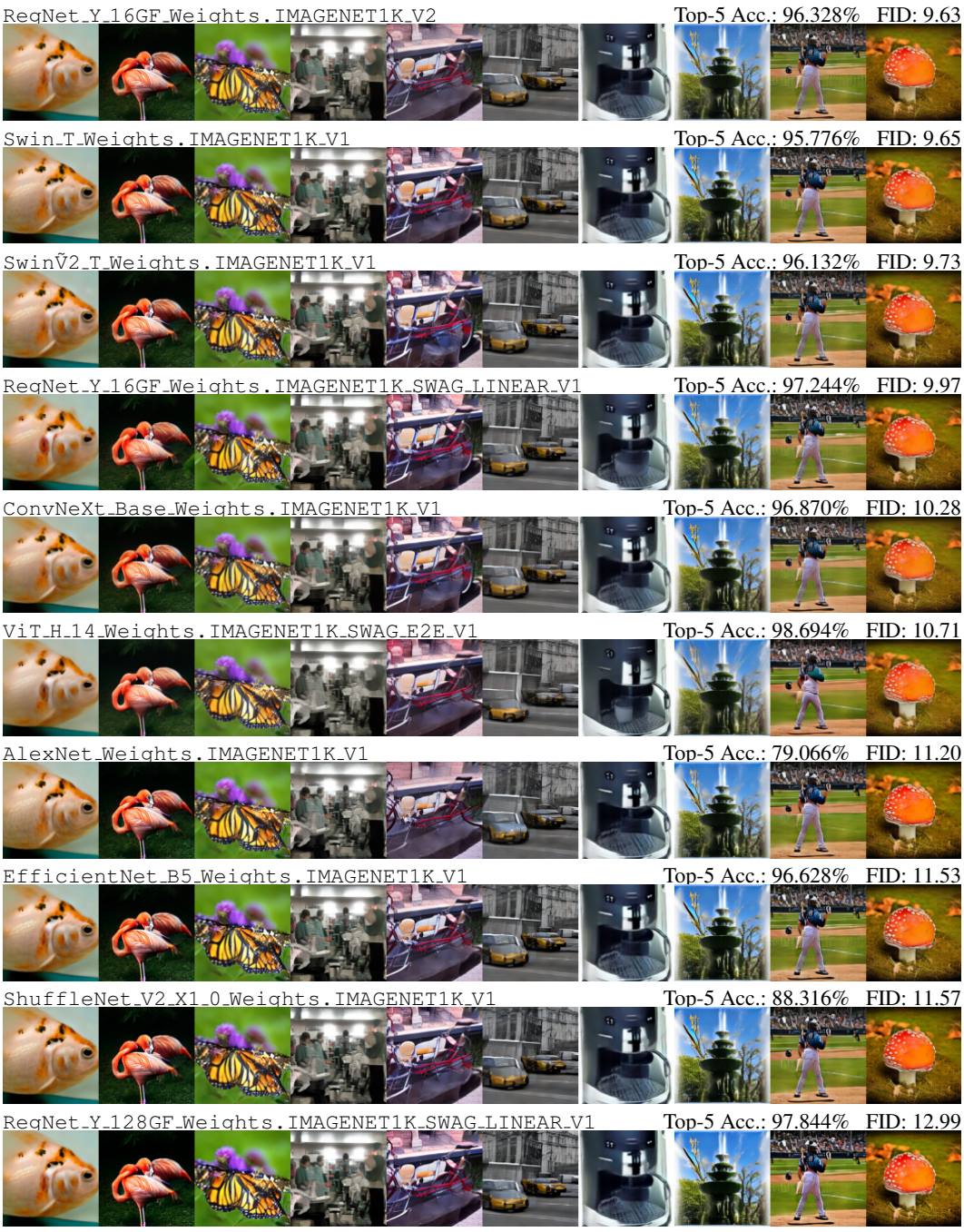
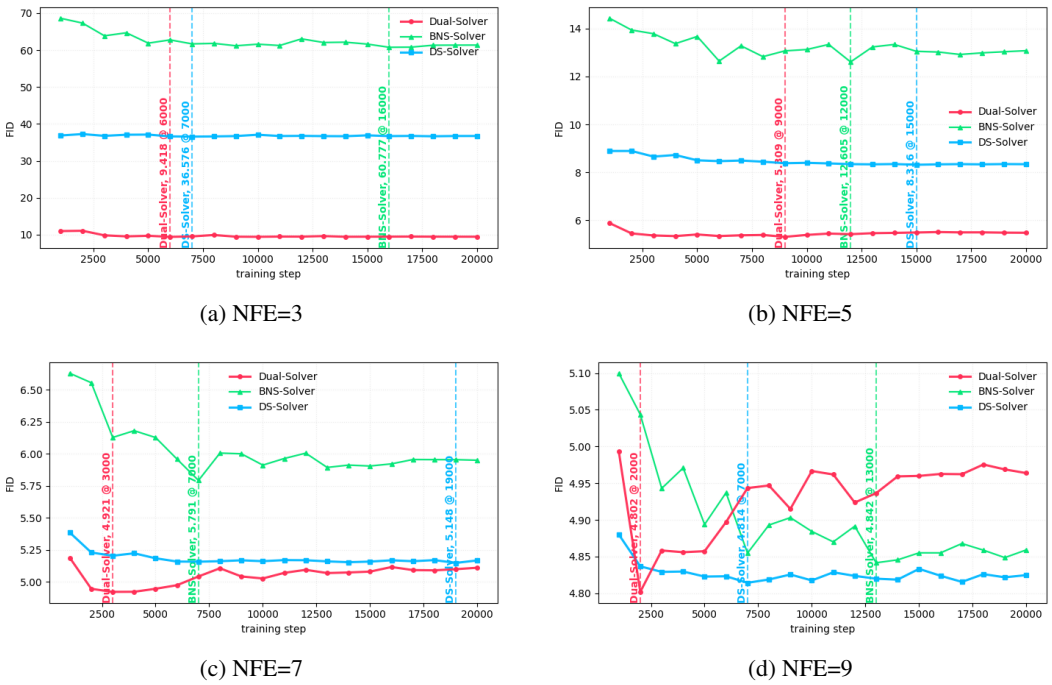


Figure 12: **Sampling results by classifier.** (continued).

E Analysis of Convergence Time



(a) NFE=3 (b) NFE=5

(c) NFE=7 (d) NFE=9

Figure 13: **FID vs. training step** for three solvers on GM-DiT (CFG=1.4): Dual-Solver (Ours), BNS-Solver (Shaul et al., 2024), and DS-Solver (Wang et al., 2025).

In Sec. 6.4, we analyze the per-step computational cost. Table 10 provides the corresponding summary statistics. In this section, we analyze the time to convergence. As reported in Appendix C, we train up to 20k steps, but the best-performing checkpoint is typically found earlier. Because FID evaluates a different criterion than the regression- and classification-based objectives, training longer does not ensure better FID.

Stage	Dual-Solver	BNS-Solver	DS-Solver
forward	135.39	140.57	138.33
decoding	73.67	–	–
classifier	10.19	–	–
backward	253.04	296.82	297.02
sum	**472.29**	**437.39**	**435.35**

Table 10: Training time per step (ms; batch size = 10, GM-DiT). Means over 100 runs.

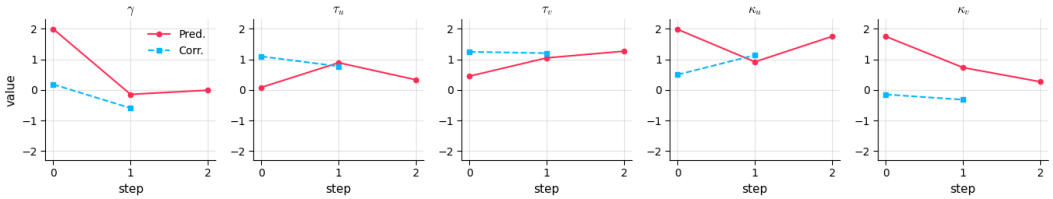
Fig. 13 reports FID evaluated every 1k steps during training up to 20k. We use GM-DiT (Chen et al., 2025) as the backbone and compute FID over 10k samples. All three models—Dual-Solver (ours), BNS-Solver (Shaul et al., 2024), and DS-Solver (Wang et al., 2025)—reach their minimum FID before 20k steps.

For NFE$=3$, Dual-Solver attains its minimum at 6k steps, DS-Solver at 7k steps, and BNS-Solver at 16k steps. For NFE$=5$, Dual-Solver at 9k, BNS-Solver at 12k, and DS-Solver at 15k. For NFE$=7$, Dual-Solver at 3k, BNS-Solver at 7k, and DS-Solver at 19k. For NFE$=9$, Dual-Solver at 2k, DS-Solver at 7k, and BNS-Solver at 13k.

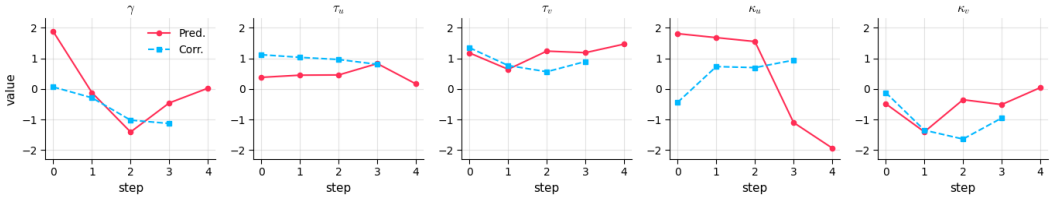
For Dual-Solver, FID often increases after the first minimum, which we attribute to overfitting toward classifier decision regions that reduces recall (Appendix D). Across NFE$=3,5,7,9$, Dual-Solver converges faster; we attribute this to the classification objective, which only requires samples to enter the correct decision region rather than matching per-sample targets.

F LEARNED PARAMETERS

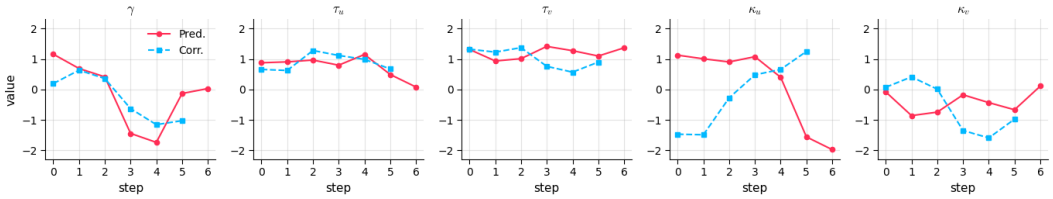
In Figs. 14, 15, 16, 17 we plot the Dual-Solver parameters learned via classification-based learning (Sec. 5.2), as defined in Sec. 4.2. For the DiT, GM-DiT, SANA, and PixArt-α backbones, we set CFG to 1.5, 1.4, 4.5, and 3.5, respectively, and train for 20k steps. We plot results at NFE = 3, 5, 7, 9 for DiT and GM-DiT, and at NFE = 3, 4, 5, 6 for SANA and PixArt-α. Separate parameter sets are learned for the predictor and the corrector, and they are shown in different colors. Within the same backbone, the parameter curves are similar even across different NFEs. We conjecture that this arises from the backbone's intrinsic trajectory.



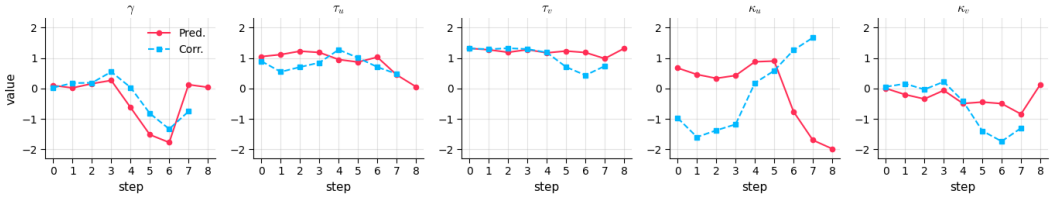
(a) NFE = 3



(b) NFE = 5

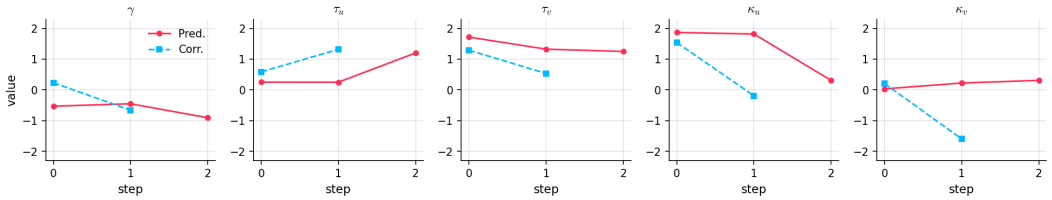


(c) NFE = 7

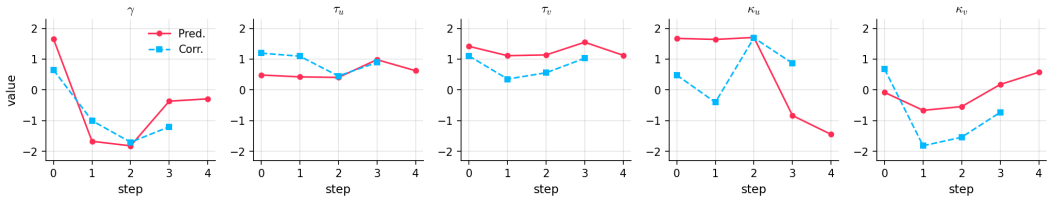


(d) NFE = 9

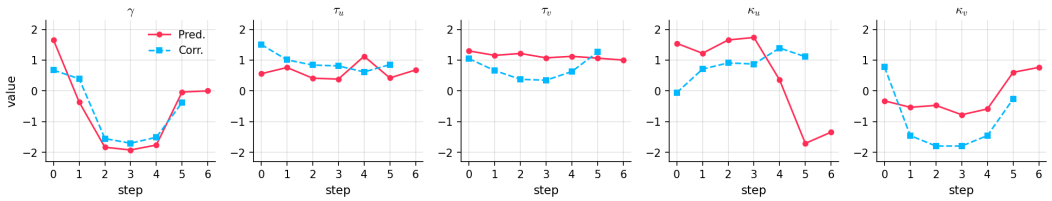
Figure 14: **Learned parameters.** $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ for DiT (Peebles & Xie, 2023).



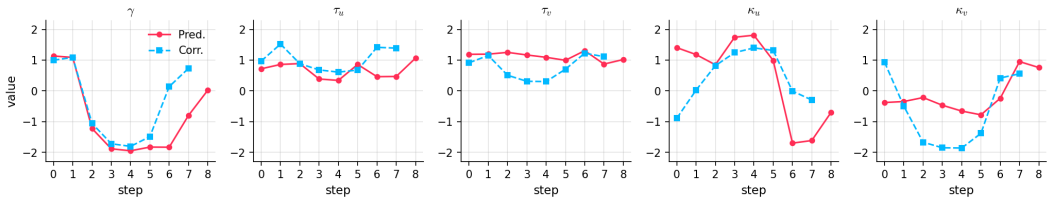
(a) NFE = 3



(b) NFE = 5

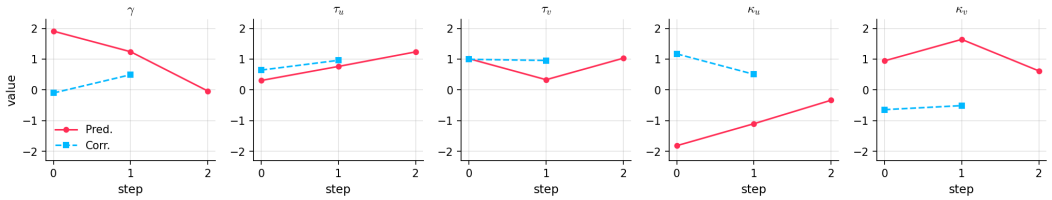


(c) NFE = 7

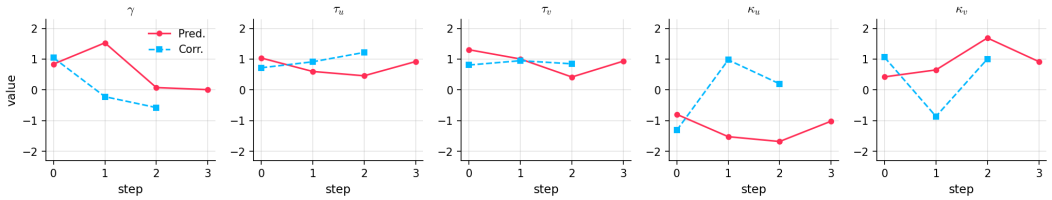


(d) NFE = 9

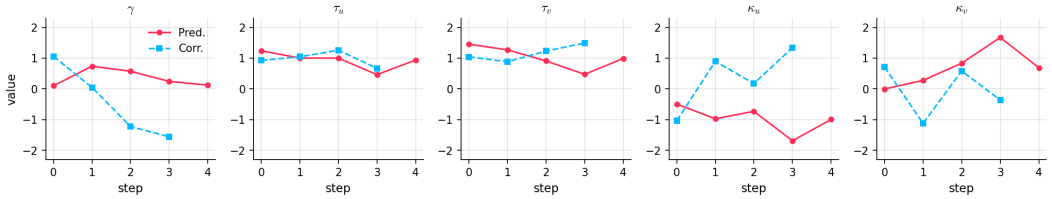
Figure 15: **Learned parameters.** $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ for GM-DiT (Chen et al., 2025).



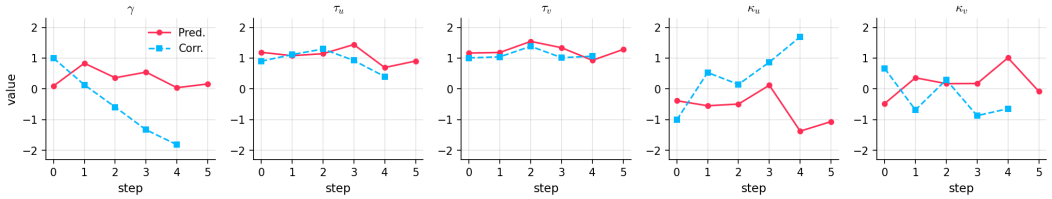
(a) NFE = 3



(b) NFE = 4

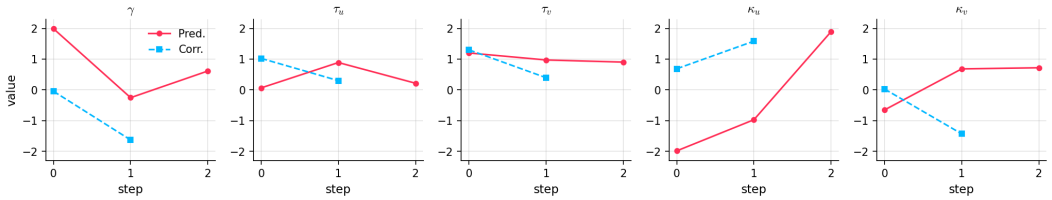


(c) NFE = 5

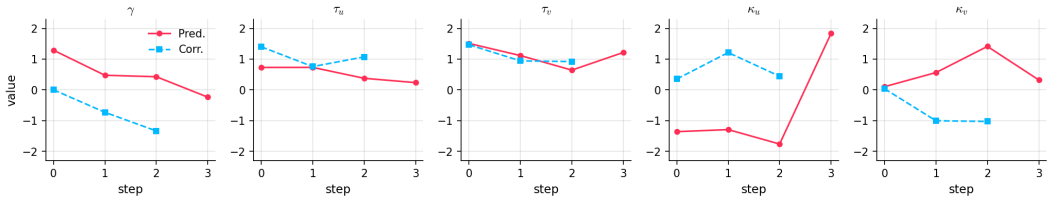


(d) NFE = 6

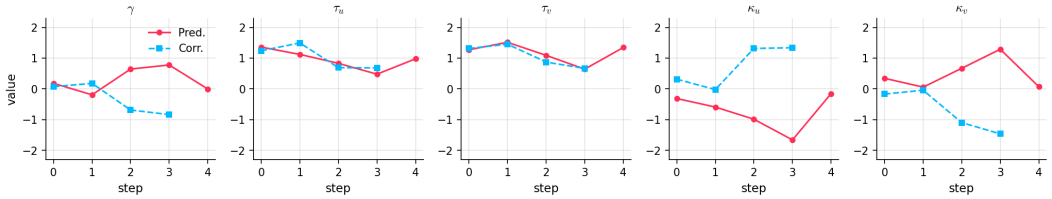
Figure 16: **Learned parameters.** $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ for SANA (Xie et al., 2024).



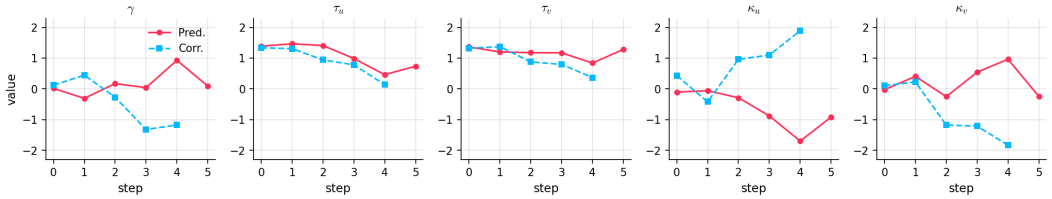
(a) NFE = 3



(b) NFE = 4



(c) NFE = 5



(d) NFE = 6

Figure 17: **Learned parameters.** $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ for PixArt-α (Chen et al., 2023).

G Details of parameter interpolation across NFEs

In this section, we describe how to interpolate the parameters discussed in Sec. 6.3 so that they can be used at other NFEs. The parameters of Dual-Solver,

$$\phi = \{\gamma^{\text{pred}}, \tau_u^{\text{pred}}, \tau_v^{\text{pred}}, \kappa_u^{\text{pred}}, \kappa_v^{\text{pred}}, \gamma^{\text{corr}}, \tau_u^{\text{corr}}, \tau_v^{\text{corr}}, \kappa_u^{\text{corr}}, \kappa_v^{\text{corr}}\},$$

are given as arrays whose length equals the NFE. (The corrector parameters have length NFE-1, and we match the length to NFE by repeating the last element.) For example, $\gamma^{\text{pred}} = (\gamma_0^{\text{pred}}, \ldots, \gamma_{NFE-1}^{\text{pred}})$. To interpolate these parameters, we consider the following linear interpolation scheme for a generic array.

Definition G.1 (Linear interpolation)**.** Let $f^{(M)} = (f_0^{(M)}, \ldots, f_{M-1}^{(M)})$ be an array of length M. The linearly interpolated array $\text{Interp}(f^{(M)}; N)$ of length N is defined as follows. First, set

$$t_i = \frac{i}{N-1}(M-1), \qquad j_i = \lfloor t_i \rfloor, \qquad \alpha_i = t_i - j_i,$$

and for each $i = 0, \ldots, N-1$ define

$$\text{Interp}(f^{(M)}; N)[i] = (1-\alpha_i)\, f_{j_i}^{(M)} + \alpha_i\, f_{j_i+1}^{(M)}.$$

Definition G.2 (Averaged linear interpolation)**.** Let $M < N < L$ be three NFEs, and let $f^{(M)} \in \mathbb{R}^M$ and $f^{(L)} \in \mathbb{R}^L$ be the corresponding arrays. We first obtain their linearly interpolated versions of length N,

$$\tilde{f}^{(M)} = \text{Interp}(f^{(M)}; N), \qquad \tilde{f}^{(L)} = \text{Interp}(f^{(L)}; N).$$

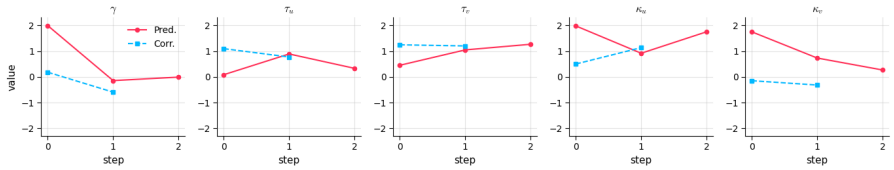
We then use the relative position of N between M and L as weights and define

$$w_M = \frac{L-N}{L-M}, \qquad w_L = \frac{N-M}{L-M},$$

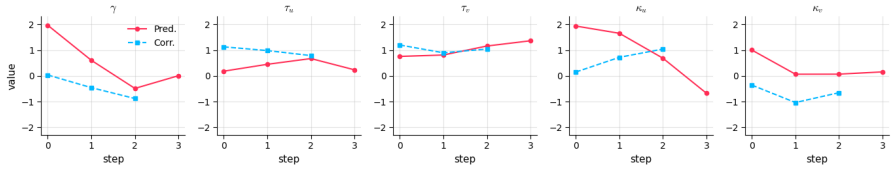
and, for each $i = 0, \ldots, N-1$,

$$\text{Interp}(f^{(M)}, f^{(L)}; N)[i] = w_M\, \tilde{f}^{(M)}[i] + w_L\, \tilde{f}^{(L)}[i].$$

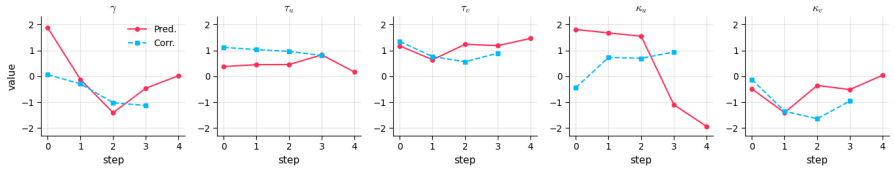
Using this procedure, we obtain the array for an intermediate NFE from the two arrays at neighboring NFEs, and apply the same construction to every parameter in ϕ. Examples of interpolated parameters are shown in Fig. 18. Specifically, we obtain the parameters for NFE=4 by interpolating those learned at NFE=(3, 5), for NFE=6 from NFE=(5, 7), and for NFE=8 from NFE=(7, 9).



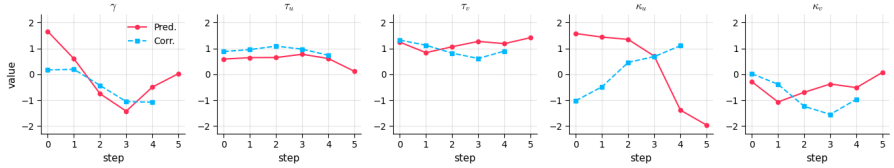
(a) NFE = 3



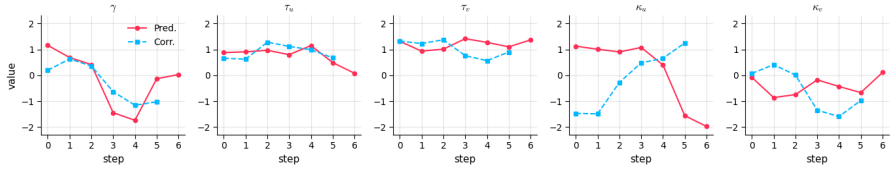
(b) NFE = 4, interpolated from (3, 5)



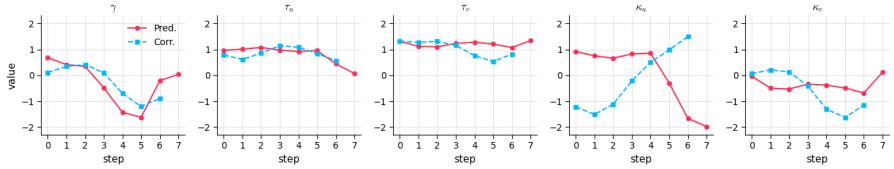
(c) NFE = 5



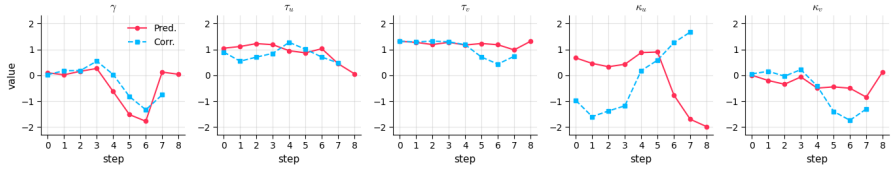
(d) NFE = 6, interpolated from (5, 7)



(e) NFE = 7



(f) NFE = 8, interpolated from (7, 9)



(g) NFE = 9

Figure 18: **Interpolated parameters.** $\{\gamma, \tau_u, \tau_v, \kappa_u, \kappa_v\}$ for DiT (Peebles & Xie, 2023).

H SINGLE PREDICTION REPARAMETERIZATION

In this section, we show that the first-order predictor in Eq. 13 and the second-order corrector in Eq. 21 can each be expressed using a single prediction type (noise, data, or velocity).

H.1 FIRST-ORDER PREDICTOR

For $\gamma \geq 0$, Eq. 13 gives:

$$\boldsymbol{x}_{t_{i+1}}^{\text{1st-pred.}} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\Big[\boldsymbol{x}_\theta(t_i)\, K(u_i) + \boldsymbol{\epsilon}_\theta(t_i)\, K(v_i)\Big], \tag{22}$$

where for brevity we denote

$$\boldsymbol{x}_\theta(t_i) := \boldsymbol{x}_\theta(\boldsymbol{x}_{t_i}, t_i), \quad \boldsymbol{\epsilon}_\theta(t_i) := \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_i}, t_i),$$

and

$$K(u_i) := \Delta L^{-1}(u_i; \tau_u) + B(\Delta u_i; \kappa_u), \quad K(v_i) := \Delta L^{-1}(v_i; \tau_v) + B(\Delta v_i; \kappa_v).$$

The case $\gamma < 0$ admits an analogous derivation by replacing σ^γ with $\alpha^{-\gamma}$ as in Eq. 13.

Reparameterization to noise prediction. Assume the backbone outputs only the noise prediction $\boldsymbol{\epsilon}_\theta(t)$. Using the deterministic transform between predictions from Eq. 4,

$$\boldsymbol{x}_\theta(t_i) = \frac{\boldsymbol{x}_{t_i} - \sigma_{t_i}\boldsymbol{\epsilon}_\theta(t_i)}{\alpha_{t_i}}, \tag{23}$$

Eq. 22 becomes

$$\begin{aligned}
\boldsymbol{x}_{t_{i+1}}^{\text{1st-pred.}} &= \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\left[\frac{\boldsymbol{x}_{t_i} - \sigma_{t_i}\boldsymbol{\epsilon}_\theta(t_i)}{\alpha_{t_i}}\, K(u_i) + \boldsymbol{\epsilon}_\theta(t_i)\, K(v_i)\right] \\
&= \left[\left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} + \sigma_{t_{i+1}}^{\gamma}\frac{K(u_i)}{\alpha_{t_i}}\right]\boldsymbol{x}_{t_i} + \left[\sigma_{t_{i+1}}^{\gamma}\left(K(v_i) - \frac{\sigma_{t_i}}{\alpha_{t_i}}K(u_i)\right)\right]\boldsymbol{\epsilon}_\theta(t_i).
\end{aligned} \tag{24}$$

Thus, for a noise-prediction backbone, the predictor is a linear combination of $\boldsymbol{x}_{t_i}$ and $\boldsymbol{\epsilon}_\theta(t_i)$.

Reparameterization to data prediction. Assume the backbone outputs only the data prediction $\boldsymbol{x}_\theta(t)$. From Eq. 4 we have the inverse relation

$$\boldsymbol{\epsilon}_\theta(t_i) = \frac{\boldsymbol{x}_{t_i} - \alpha_{t_i}\boldsymbol{x}_\theta(t_i)}{\sigma_{t_i}}. \tag{25}$$

Substituting Eq. 25 into Eq. 22 yields

$$\begin{aligned}
\boldsymbol{x}_{t_{i+1}}^{\text{1st-pred.}} &= \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} \boldsymbol{x}_{t_i} + \sigma_{t_{i+1}}^{\gamma}\left[\boldsymbol{x}_\theta(t_i)\, K(u_i) + \frac{\boldsymbol{x}_{t_i} - \alpha_{t_i}\boldsymbol{x}_\theta(t_i)}{\sigma_{t_i}}\, K(v_i)\right] \\
&= \left[\left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^{\gamma} + \sigma_{t_{i+1}}^{\gamma}\frac{K(v_i)}{\sigma_{t_i}}\right]\boldsymbol{x}_{t_i} + \left[\sigma_{t_{i+1}}^{\gamma}\left(K(u_i) - \frac{\alpha_{t_i}}{\sigma_{t_i}}K(v_i)\right)\right]\boldsymbol{x}_\theta(t_i).
\end{aligned} \tag{26}$$

Therefore, for a data-prediction backbone, the predictor is a linear combination of $\boldsymbol{x}_{t_i}$ and $\boldsymbol{x}_\theta(t_i)$.

Reparameterization to velocity prediction. Assume the backbone outputs only the velocity prediction $\boldsymbol{v}_\theta(t)$. Eq. 4 gives the linear relations

$$\begin{bmatrix} \boldsymbol{x}_{t_i} \\ \boldsymbol{v}_\theta(t_i) \end{bmatrix} = M_{t_i}\begin{bmatrix} \boldsymbol{x}_\theta(t_i) \\ \boldsymbol{\epsilon}_\theta(t_i) \end{bmatrix}, \qquad M_{t_i} := \begin{bmatrix} \alpha_{t_i} & \sigma_{t_i} \\ \dot{\alpha}_{t_i} & \dot{\sigma}_{t_i} \end{bmatrix}, \tag{27}$$

where $\dot{\alpha}_{t_i} := \frac{d\alpha_{t_i}}{dt}$ and $\dot{\sigma}_{t_i} := \frac{d\sigma_{t_i}}{dt}$. Let

$$\det M_{t_i} = \alpha_{t_i}\dot{\sigma}_{t_i} - \sigma_{t_i}\dot{\alpha}_{t_i}.$$

Inverting Eq. 27 gives

$$\boldsymbol{x}_\theta(t_i) = (\det M_{t_i})^{-1}\Big(\dot\sigma_{t_i}\,\boldsymbol{x}_{t_i} - \sigma_{t_i}\,\boldsymbol{v}_\theta(t_i)\Big), \quad \boldsymbol{\epsilon}_\theta(t_i) = (\det M_{t_i})^{-1}\Big(-\dot\alpha_{t_i}\,\boldsymbol{x}_{t_i} + \alpha_{t_i}\,\boldsymbol{v}_\theta(t_i)\Big). \tag{28}$$

Substituting Eq. 28 into Eq. 22 yields

$$\begin{aligned}\boldsymbol{x}^{\text{1st-pred.}}_{t_{i+1}} &= \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma \boldsymbol{x}_{t_i} + \sigma^\gamma_{t_{i+1}}(\det M_{t_i})^{-1}\Big[K(u_i)(\dot\sigma_{t_i}\boldsymbol{x}_{t_i} - \sigma_{t_i}\boldsymbol{v}_\theta(t_i)) + K(v_i)(-\dot\alpha_{t_i}\boldsymbol{x}_{t_i} + \alpha_{t_i}\boldsymbol{v}_\theta(t_i))\Big] \\ &= \left[\left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma + \sigma^\gamma_{t_{i+1}}\frac{K(u_i)\dot\sigma_{t_i} - K(v_i)\dot\alpha_{t_i}}{\det M_{t_i}}\right]\boldsymbol{x}_{t_i} + \left[\sigma^\gamma_{t_{i+1}}\frac{\alpha_{t_i}K(v_i) - \sigma_{t_i}K(u_i)}{\det M_{t_i}}\right]\boldsymbol{v}_\theta(t_i).\end{aligned} \tag{29}$$

Thus, for a velocity-prediction backbone, the predictor is a linear combination of $\boldsymbol{x}_{t_i}$ and $\boldsymbol{v}_\theta(t_i)$.

H.2 Second-order corrector

For $\gamma \geq 0$, Eq. 21 gives:

$$\boldsymbol{x}^{\text{2nd-corr.}}_{t_{i+1}} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma \boldsymbol{x}_{t_i} + \sigma^\gamma_{t_{i+1}}\left[\boldsymbol{x}_\theta(t_i)\,\Delta L^{-1}(u_i) + \frac{\Delta\boldsymbol{x}_\theta(t_i)}{2}\,K(u_i) + \boldsymbol{\epsilon}_\theta(t_i)\,\Delta L^{-1}(v_i) + \frac{\Delta\boldsymbol{\epsilon}_\theta(t_i)}{2}\,K(v_i)\right], \tag{30}$$

where

$$\Delta\boldsymbol{x}_\theta(t_i) := \boldsymbol{x}_\theta(t_{i+1}) - \boldsymbol{x}_\theta(t_i), \qquad \Delta\boldsymbol{\epsilon}_\theta(t_i) := \boldsymbol{\epsilon}_\theta(t_{i+1}) - \boldsymbol{\epsilon}_\theta(t_i),$$

and

$$K(u_i) := \Delta L^{-1}(u_i) + B(\Delta u_i;\kappa_u), \quad K(v_i) := \Delta L^{-1}(v_i) + B(\Delta v_i;\kappa_v).$$

As in standard predictor– corrector schemes, the predictions at t_{i+1} are evaluated using the first–order predictor $\boldsymbol{x}^{\text{1st-pred.}}_{t_{i+1}}$.

Reparameterization to noise prediction. Assume the backbone outputs only the noise prediction $\boldsymbol{\epsilon}_\theta(t)$. Using Eq. 4,

$$\boldsymbol{x}_\theta(t_j) = \frac{\boldsymbol{x}_{t_j} - \sigma_{t_j}\boldsymbol{\epsilon}_\theta(t_j)}{\alpha_{t_j}}, \qquad j \in \{i, i+1\}, \tag{31}$$

we obtain

$$\Delta\boldsymbol{x}_\theta(t_i) = \frac{\boldsymbol{x}'_{t_{i+1}} - \sigma_{t_{i+1}}\boldsymbol{\epsilon}_\theta(t_{i+1})}{\alpha_{t_{i+1}}} - \frac{\boldsymbol{x}_{t_i} - \sigma_{t_i}\boldsymbol{\epsilon}_\theta(t_i)}{\alpha_{t_i}}, \qquad \Delta\boldsymbol{\epsilon}_\theta(t_i) = \boldsymbol{\epsilon}_\theta(t_{i+1}) - \boldsymbol{\epsilon}_\theta(t_i),$$

where we denote $\boldsymbol{x}'_{t_{i+1}} = \boldsymbol{x}^{\text{1st-pred.}}_{t_{i+1}}$. Substituting these into Eq. 30 and collecting terms with respect to $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{\epsilon}_\theta(t_i)$, and $\boldsymbol{\epsilon}_\theta(t_{i+1})$ yields

$$\boldsymbol{x}^{\text{2nd-corr.}}_{t_{i+1}} = C^{(x)}_i\,\boldsymbol{x}_{t_i} + C^{(x)}_{i+1}\,\boldsymbol{x}'_{t_{i+1}} + C^{(\epsilon_\theta)}_i\,\boldsymbol{\epsilon}_\theta(t_i) + C^{(\epsilon_\theta)}_{i+1}\,\boldsymbol{\epsilon}_\theta(t_{i+1}), \tag{32}$$

where

$$C^{(x)}_i = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma + \sigma^\gamma_{t_{i+1}}\left(\frac{\Delta L^{-1}(u_i)}{\alpha_{t_i}} - \frac{K(u_i)}{2\,\alpha_{t_i}}\right), \tag{33}$$

$$C^{(x)}_{i+1} = \sigma^\gamma_{t_{i+1}}\frac{K(u_i)}{2\,\alpha_{t_{i+1}}}, \tag{34}$$

$$C^{(\epsilon_\theta)}_i = \sigma^\gamma_{t_{i+1}}\left(-\frac{\sigma_{t_i}}{\alpha_{t_i}}\Delta L^{-1}(u_i) + \Delta L^{-1}(v_i) + \frac{\sigma_{t_i}}{2\,\alpha_{t_i}}K(u_i) - \frac{1}{2}K(v_i)\right), \tag{35}$$

$$C^{(\epsilon_\theta)}_{i+1} = \sigma^\gamma_{t_{i+1}}\left(-\frac{\sigma_{t_{i+1}}}{2\,\alpha_{t_{i+1}}}K(u_i) + \frac{1}{2}K(v_i)\right). \tag{36}$$

Thus, for a noise–prediction backbone, the second–order corrector is a linear combination of $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{\epsilon}_\theta(t_i)$, and $\boldsymbol{\epsilon}_\theta(t_{i+1})$.

Reparameterization to data prediction. Assume the backbone outputs only the data prediction $\boldsymbol{x}_\theta(t)$. Using the inverse relation

$$\boldsymbol{\epsilon}_\theta(t_j) = \frac{\boldsymbol{x}_{t_j} - \alpha_{t_j}\boldsymbol{x}_\theta(t_j)}{\sigma_{t_j}}, \qquad j \in \{i, i+1\}, \tag{37}$$

we obtain

$$\Delta\boldsymbol{\epsilon}_\theta(t_i) = \frac{\boldsymbol{x}'_{t_{i+1}} - \alpha_{t_{i+1}}\boldsymbol{x}_\theta(t_{i+1})}{\sigma_{t_{i+1}}} - \frac{\boldsymbol{x}_{t_i} - \alpha_{t_i}\boldsymbol{x}_\theta(t_i)}{\sigma_{t_i}}, \qquad \Delta\boldsymbol{x}_\theta(t_i) = \boldsymbol{x}_\theta(t_{i+1}) - \boldsymbol{x}_\theta(t_i),$$

where we denote $\boldsymbol{x}'_{t_{i+1}} = \boldsymbol{x}^{\text{1st-pred.}}_{t_{i+1}}$. Substituting into Eq. 30 and collecting terms with respect to $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{x}_\theta(t_i)$, and $\boldsymbol{x}_\theta(t_{i+1})$ yields

$$\boldsymbol{x}^{\text{2nd-corr.}}_{t_{i+1}} = C_i^{(x)}\,\boldsymbol{x}_{t_i} + C_{i+1}^{(x)}\,\boldsymbol{x}'_{t_{i+1}} + C_i^{(x_\theta)}\,\boldsymbol{x}_\theta(t_i) + C_{i+1}^{(x_\theta)}\,\boldsymbol{x}_\theta(t_{i+1}), \tag{38}$$

where

$$C_i^{(x)} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma + \sigma_{t_{i+1}}^\gamma\left(\frac{\Delta L^{-1}(v_i)}{\sigma_{t_i}} - \frac{K(v_i)}{2\,\sigma_{t_i}}\right), \tag{39}$$

$$C_{i+1}^{(x)} = \sigma_{t_{i+1}}^\gamma\,\frac{K(v_i)}{2\,\sigma_{t_{i+1}}}, \tag{40}$$

$$C_i^{(x_\theta)} = \sigma_{t_{i+1}}^\gamma\left(\Delta L^{-1}(u_i) - \frac{\alpha_{t_i}}{\sigma_{t_i}}\Delta L^{-1}(v_i) - \frac{1}{2}K(u_i) + \frac{\alpha_{t_i}}{2\,\sigma_{t_i}}K(v_i)\right), \tag{41}$$

$$C_{i+1}^{(x_\theta)} = \sigma_{t_{i+1}}^\gamma\left(\frac{1}{2}K(u_i) - \frac{\alpha_{t_{i+1}}}{2\,\sigma_{t_{i+1}}}K(v_i)\right). \tag{42}$$

Therefore, for a data–prediction backbone, the corrector is a linear combination of $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{x}_\theta(t_i)$, and $\boldsymbol{x}_\theta(t_{i+1})$.

Reparameterization to velocity prediction. Assume the backbone outputs only the velocity prediction $\boldsymbol{v}_\theta(t)$. Eq. 4 implies the linear system

$$\begin{bmatrix}\boldsymbol{x}_{t_j}\\ \boldsymbol{v}_\theta(t_j)\end{bmatrix} = M_{t_j}\begin{bmatrix}\boldsymbol{x}_\theta(t_j)\\ \boldsymbol{\epsilon}_\theta(t_j)\end{bmatrix}, \qquad M_{t_j} := \begin{bmatrix}\alpha_{t_j} & \sigma_{t_j}\\ \dot\alpha_{t_j} & \dot\sigma_{t_j}\end{bmatrix}, \quad j \in \{i, i+1\}, \tag{43}$$

where $\dot\alpha_{t_j} := \frac{d\alpha_{t_j}}{dt}$ and $\dot\sigma_{t_j} := \frac{d\sigma_{t_j}}{dt}$, and

$$\det M_{t_j} = \alpha_{t_j}\dot\sigma_{t_j} - \sigma_{t_j}\dot\alpha_{t_j}.$$

Inverting Eq. 43 gives

$$\boldsymbol{x}_\theta(t_j) = (\det M_{t_j})^{-1}\Big(\dot\sigma_{t_j}\,\boldsymbol{x}_{t_j} - \sigma_{t_j}\,\boldsymbol{v}_\theta(t_j)\Big), \quad \boldsymbol{\epsilon}_\theta(t_j) = (\det M_{t_j})^{-1}\Big(-\dot\alpha_{t_j}\,\boldsymbol{x}_{t_j} + \alpha_{t_j}\,\boldsymbol{v}_\theta(t_j)\Big). \tag{44}$$

Accordingly,

$$\Delta\boldsymbol{x}_\theta(t_i) = \boldsymbol{x}_\theta(t_{i+1}) - \boldsymbol{x}_\theta(t_i), \qquad \Delta\boldsymbol{\epsilon}_\theta(t_i) = \boldsymbol{\epsilon}_\theta(t_{i+1}) - \boldsymbol{\epsilon}_\theta(t_i),$$

where each $\boldsymbol{x}_\theta(t_j)$ and $\boldsymbol{\epsilon}_\theta(t_j)$ is an linear combination of $\boldsymbol{x}_{t_j}$ and $\boldsymbol{v}_\theta(t_j)$ via Eq. 44. Substituting Eq. 44 into Eq. 30 and collecting terms with respect to $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{v}_\theta(t_i)$, and $\boldsymbol{v}_\theta(t_{i+1})$ yields

$$\boldsymbol{x}^{\text{2nd-corr.}}_{t_{i+1}} = C_i^{(x)}\,\boldsymbol{x}_{t_i} + C_{i+1}^{(x)}\,\boldsymbol{x}'_{t_{i+1}} + C_i^{(v_\theta)}\,\boldsymbol{v}_\theta(t_i) + C_{i+1}^{(v_\theta)}\,\boldsymbol{v}_\theta(t_{i+1}), \tag{45}$$

where

$$C_i^{(x)} = \left(\frac{\sigma_{t_{i+1}}}{\sigma_{t_i}}\right)^\gamma + \sigma_{t_{i+1}}^\gamma\,\frac{\dot\sigma_{t_i}\Delta L^{-1}(u_i) - \dot\alpha_{t_i}\Delta L^{-1}(v_i) - \frac{1}{2}\dot\sigma_{t_i}K(u_i) + \frac{1}{2}\dot\alpha_{t_i}K(v_i)}{\det M_{t_i}}, \tag{46}$$

$$C_{i+1}^{(x)} = \sigma_{t_{i+1}}^{\gamma} \frac{\frac{1}{2}\dot{\sigma}_{t_{i+1}} K(u_i) - \frac{1}{2}\dot{\alpha}_{t_{i+1}} K(v_i)}{\det M_{t_{i+1}}}, \tag{47}$$

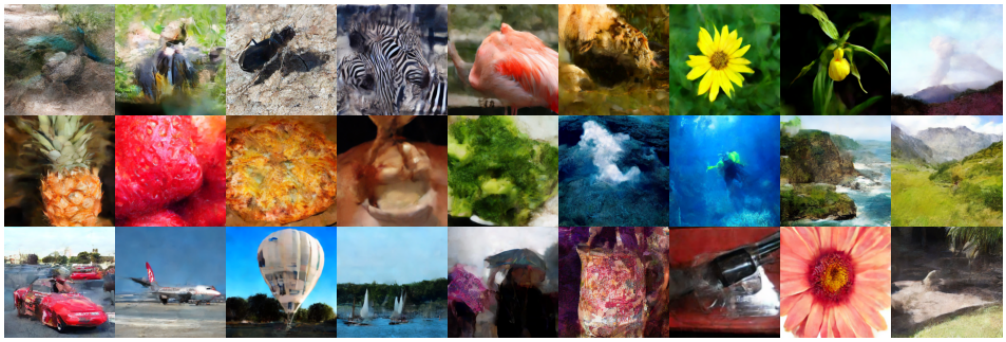
$$C_{i}^{(v_\theta)} = \sigma_{t_{i+1}}^{\gamma} \frac{-\sigma_{t_i} \Delta L^{-1}(u_i) + \alpha_{t_i} \Delta L^{-1}(v_i) + \frac{1}{2}\sigma_{t_i} K(u_i) - \frac{1}{2}\alpha_{t_i} K(v_i)}{\det M_{t_i}}, \tag{48}$$

$$C_{i+1}^{(v_\theta)} = \sigma_{t_{i+1}}^{\gamma} \frac{-\frac{1}{2}\sigma_{t_{i+1}} K(u_i) + \frac{1}{2}\alpha_{t_{i+1}} K(v_i)}{\det M_{t_{i+1}}}. \tag{49}$$

Thus, for a velocity–prediction backbone, the second–order corrector is a linear combination of $\boldsymbol{x}_{t_i}$, $\boldsymbol{x}'_{t_{i+1}}$, $\boldsymbol{v}_\theta(t_i)$, and $\boldsymbol{v}_\theta(t_{i+1})$.

(a) DPM-Solver++ (Lu et al., 2022b)

(b) Dual-Solver (Ours)



(c) BNS-Solver (Shaul et al., 2024)

(d) DS-Solver (Wang et al., 2025)

Figure 19: **Additional sampling results.** DiT-XL/2 256×256 (NFE=4, CFG=1.5)

(a) DPM-Solver++ (Lu et al., 2022b)

(b) Dual-Solver (Ours)

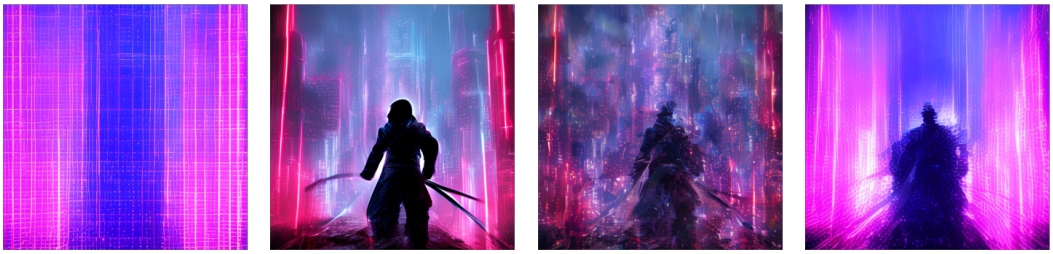
(c) BNS-Solver (Shaul et al., 2024)

(d) DS-Solver (Wang et al., 2025)

Figure 20: **Additional sampling results.** GM-DiT 256×256 (NFE=3, CFG=1.4)

Golden autumn park of falling leaves, graceful girl playing violin, flowing satin dress, impressionist brush strokes

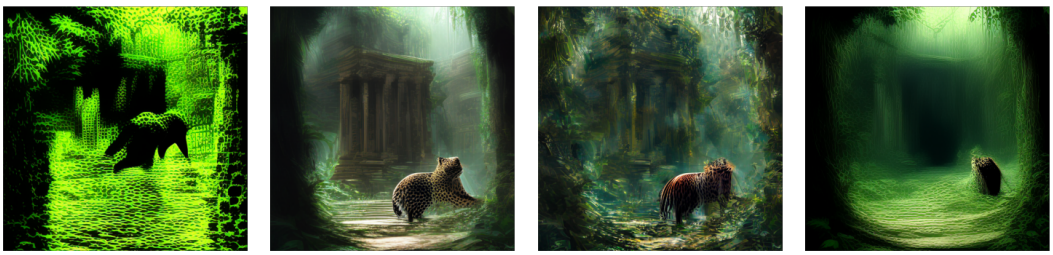
Towering neon-lit cyberpunk skyline, armored samurai with glowing katana, chrome textures, cinematic digital art



Remote desert oasis at sunrise, powerful white stallion galloping, glossy mane and muscles, hyperreal photo realism

Vibrant tropical beach at sunset, pirate ship anchored offshore, weathered wood hull, playful cartoon illustration

Overgrown jungle temple ruins, spotted leopard stalking prey, sleek fur patterns, painterly realism concept art

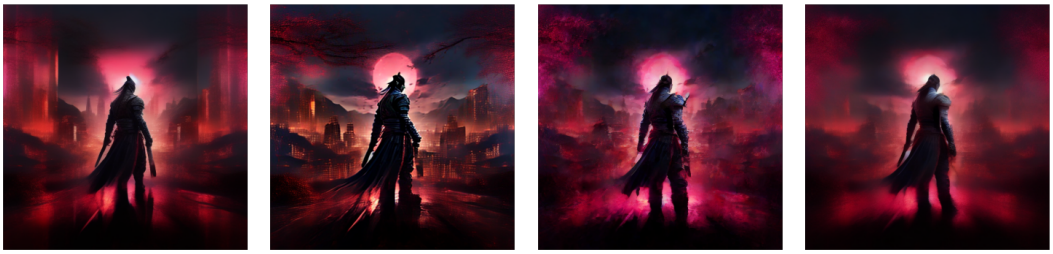


(a) DPM-Solver++ (Lu et al., 2022b) (b) Dual-Solver (Ours) (c) BNS-Solver (Shaul et al., 2024) (d) DS-Solver (Wang et al., 2025)

Figure 21: **Additional sampling results.** SANA (Xie et al., 2024), NFE=3, CFG=4.5.

In a misty emerald forest clearing, a majestic golden wolf with silky glowing fur, painted in luminous oil style

Towering neon-lit cyberpunk skyline, armored samurai with glowing katana, chrome textures, cinematic digital art



Vast desert beneath a starry cosmos, colossal sphinx carved in sandstone, rough surface, surreal dreamlike painting

Sleek futuristic laboratory interior, humanoid AI robot in motion, polished steel surfaces, sci-fi blueprint style

Storm-drenched battlefield ruins, giant mech rising from fire, rusted armor plates, gritty photorealistic concept art

(a) DPM-Solver++ (Lu et al., 2022b) (b) Dual-Solver (Ours) (c) BNS-Solver (Shaul et al., 2024) (d) DS-Solver (Wang et al., 2025)

Figure 22: **Additional sampling results.** PixArt-α (Chen et al., 2023), NFE=5, CFG=3.5.