A Related Work

Diffusion probabilistic models (DPMs) [50, 15, 54], also known as score-based generative models (SGMs), have achieved remarkable generation ability on image domain [10, 22], yielding extensive applications such as speech, singing and video synthesis [6, 29, 17, 14], controllable image generation, translation and editing [40, 44, 45, 37, 60, 8], likelihood estimation [53, 24, 32, 62], data compression [24] and inverse problem solving [7, 23].

A.1 Fast Sampling Methods for DPMs

Fast sampling methods based on extra training or optimization include learning variances in the reverse process [48, 39, 2], learning sampling schedule [26, 56], learning high-order model derivatives [11], model refinement [31] and model distillation [47, 35, 52]. Though distillation-based methods can generate high-quality samples in less than 5 steps, they additionally bring onerous training costs. Moreover, the distillation process will inevitably lose part of the information of the original model, and is hard to be adapted to pre-trained large DPMs [45, 46, 44] and conditional sampling [36]. Some of distillation-based methods also lack the ability to make flexible trade-offs between sample speed and sample quality.

In contrast, training-free samplers are more lightweight and flexible. Among them, samplers based on diffusion ODEs generally requires less steps than those based on diffusion SDEs [54, 42, 3, 51, 34], since SDEs introduce more randomness and make the denoising harder in the sampling process. Previous samplers handle the diffusion ODEs with different methods, such as Heun's methods [22], splitting numerical methods [57], pseudo numerical methods [30], Adams methods [27] or exponential integrators [59, 33, 34, 61].

A.2 Comparison with Existing Solvers Based on Exponential Integrators

Table 1: Comparison between DPM-Solver-v3 and other high-order diffusion ODE solvers based on exponential integrators.

	DEIS [59]	DPM-Solver [33]	DPM-Solver++ [34]	UniPC [61]	DPM-Solver-v3 (**Ours**)
First-Order	DDIM	DDIM	DDIM	DDIM	Improved DDIM
Taylor Expanded Predictor	$\boldsymbol{\epsilon}_\theta$ for t	$\boldsymbol{\epsilon}_\theta$ for λ	$\boldsymbol{x}_\theta$ for λ	$\boldsymbol{x}_\theta$ for λ	$\boldsymbol{g}_\theta$ for λ
Solver Type (High-Order)	Multistep	Singlestep	Multistep	Multistep	Multistep
Applicable for Guided Sampling	✓	✗	✓	✓	✓
Corrector Supported	✗	✗	✗	✓	✓
Model-Specific	✗	✗	✗	✗	✓

In this section, we make some theoretical comparisons between DPM-Solver-v3 and existing diffusion ODE solvers that are based on exponential integrators [59, 33, 34, 61]. We summarize the results in Table 1, and provide some analysis below.

Previous ODE formulation as special cases of ours. First we compare the formulation of the ODE solution. Our reformulated ODE solution in Eq. (9) involves extra coefficients $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$, and it corresponds to a new predictor $\boldsymbol{g}_\theta$ to be approximated by Taylor expansion. By comparing ours with previous ODE formulations in Eq. (3) and their corresponding noise/data prediction, we can easily figure out that they are special cases of ours by setting $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$ to specific values:

- Noise prediction: $\boldsymbol{l}_\lambda = \boldsymbol{0}, \boldsymbol{s}_\lambda = -\boldsymbol{1}, \boldsymbol{b}_\lambda = \boldsymbol{0}$
- Data prediction: $\boldsymbol{l}_\lambda = \boldsymbol{1}, \boldsymbol{s}_\lambda = \boldsymbol{0}, \boldsymbol{b}_\lambda = \boldsymbol{0}$

It's worth noting that, though our ODE formulation can degenerate to previous ones, it may still result in different solvers, since previous works conduct some equivalent substitution of the same order in the local approximation (for example, the choice of $B_1(h) = h$ or $B_2(h) = e^h - 1$ in UniPC [61]). We never conduct such substitution, thus saving the efforts to tune it.

Moreover, under our framework, we find that **DPM-Solver++ is a model-agnostic approximation of DPM-Solver-v3, under the Gaussian assumption**. Specifically, according to Eq. (5), we have

$$\boldsymbol{l}_\lambda^* = \mathop{\mathrm{argmin}}_{\boldsymbol{l}_\lambda} \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)} \|\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{l}_\lambda\|_F^2, \tag{17}$$

If we assume $q_\lambda(\boldsymbol{x}_\lambda) \approx \mathcal{N}(\boldsymbol{x}_\lambda | \alpha_\lambda \boldsymbol{x}_0, \sigma_\lambda^2 \boldsymbol{I})$ for some fixed $\boldsymbol{x}_0$, then the optimal noise predictor is

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) \approx -\sigma_\lambda \nabla_{\boldsymbol{x}} \log q_\lambda(\boldsymbol{x}_\lambda) = \frac{\boldsymbol{x}_\lambda - \alpha_t \boldsymbol{x}_0}{\sigma_\lambda}. \tag{18}$$

It follows that $\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) \approx \boldsymbol{I}$, thus $\boldsymbol{l}_\lambda^* \approx \mathbf{1}$ by Eq. (5), which corresponds the data prediction model used in DPM-Solver++. Moreover, for small enough λ (i.e., t near to T), the Gaussian assumption is almost true (see Section 4.2), thus the data-prediction DPM-Solver++ approximately computes all the linear terms at the initial stage. To the best of our knowledge, this is the first explanation for the reason why the data-prediction DPM-Solver++ outperforms the noise-prediction DPM-Solver.

First-order discretization as improved DDIM Previous methods merely use noise/data parameterization, whether or not they change the time domain from t to λ. While they differ in high-order cases, they are proven to coincide in the first-order case, which is DDIM [51] (deterministic case, $\eta = 0$):

$$\hat{\boldsymbol{x}}_t = \frac{\alpha_t}{\alpha_s} \hat{\boldsymbol{x}}_s - \alpha_t \left(\frac{\sigma_s}{\alpha_s} - \frac{\sigma_t}{\alpha_t} \right) \boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_s, \lambda_s) \tag{19}$$

However, the first-order case of our method is

$$\begin{aligned} \hat{\boldsymbol{x}}_t = \frac{\alpha_t}{\alpha_s} \boldsymbol{A}(\lambda_s, \lambda_t) \left(\left(1 + \boldsymbol{l}_{\lambda_s} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \mathrm{d}\lambda \right) \hat{\boldsymbol{x}}_s - \left(\sigma_s \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \mathrm{d}\lambda \right) \boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_s, \lambda_s) \right) \\ - \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \boldsymbol{B}_{\lambda_s}(\lambda) \mathrm{d}\lambda \end{aligned} \tag{20}$$

which is not DDIM since we choose a better parameterization by the estimated EMS. Empirically, our first-order solver performs better than DDIM, as detailed in Appendix G.

B Proofs

B.1 Assumptions

In this section, we will give some mild conditions under which the local order of accuracy of Algorithm 1 and the global order of convergence of Algorithm 2 (predictor) are guaranteed.

B.1.1 Local

First we will give the assumptions to bound the local truncation error.

Assumption B.1. The total derivatives of the noise prediction model $\frac{\mathrm{d}^k \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)}{\mathrm{d}\lambda^k}, k = 1, \dots, n$ exist and are continuous.

Assumption B.2. The coefficients $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$ are continuous and bounded. $\frac{\mathrm{d}^k \boldsymbol{l}_\lambda}{\mathrm{d}\lambda^k}, \frac{\mathrm{d}^k \boldsymbol{s}_\lambda}{\mathrm{d}\lambda^k}, \frac{\mathrm{d}^k \boldsymbol{b}_\lambda}{\mathrm{d}\lambda^k}, k = 1, \dots, n$ exist and are continuous.

Assumption B.3. $\delta_k = \Theta(\lambda_t - \lambda_s), k = 1, \dots, n$

Assumption B.1 is required for the Taylor expansion which is regular in high-order numerical methods. Assumption B.2 requires the boundness of the coefficients as well as regularizes the coefficients' smoothness to enable the Taylor expansion for $\boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda)$, which holds in practice given the smoothness of $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)$ and $p_\lambda^\theta(\boldsymbol{x}_\lambda)$. Assumption B.3 makes sure δ_k and $\lambda_t - \lambda_s$ is of the same order, i.e., there exists some constant $r_k = \mathcal{O}(1)$ so that $\delta_k = r_k(\lambda_t - \lambda_s)$, which is satisfied in regular multistep methods.

B.1.2 Global

Then we will give the assumptions to bound the global error.

Assumption B.4. The noise prediction model $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, t)$ is Lipschitz w.r.t. to $\boldsymbol{x}$.

Assumption B.5. $h = \max_{1\le i\le M}(\lambda_i - \lambda_{i-1}) = \mathcal{O}(1/M)$.

Assumption B.6. The starting values $\hat{\boldsymbol{x}}_i, 1 \le i \le n$ satisfies $\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i = \mathcal{O}(h^{n+1})$.

Assumption B.4 is common in the analysis of ODEs, which assures $\boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_t, t) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) = \mathcal{O}(\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t)$. Assumption B.5 implies that the step sizes are rather uniform. Assumption B.6 is common in the convergence analysis of multistep methods [5].

B.2 Order of Accuracy and Convergence

In this section, we prove the local and global order guarantees detailed in Theorem 3.1 and Theorem 3.3.

B.2.1 Local

Proof. (Proof of Theorem 3.1) Denote $h \coloneqq \lambda_t - \lambda_s$. Subtracting the Taylor-expanded exact solution in Eq. (12) from the local approximation in Eq. (14), we have

$$\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t = -\alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \sum_{k=1}^{n} \frac{\hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) - \boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s)}{k!} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda + \mathcal{O}(h^{n+2}) \tag{21}$$

First we examine the order of $\boldsymbol{A}(\lambda_s, \lambda_t)$ and $\int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda$. Under Assumption B.2, there exists some constant C_1, C_2 such that $-\boldsymbol{l}_\lambda < C_1, \boldsymbol{l}_\lambda + \boldsymbol{s}_\lambda < C_2$. So

$$\begin{aligned} \boldsymbol{A}(\lambda_s, \lambda_t) &= e^{-\int_{\lambda_s}^{\lambda_t} \boldsymbol{l}_\tau \mathrm{d}\tau} \\ &< e^{C_1 h} \\ &= \mathcal{O}(1) \end{aligned} \tag{22}$$

$$\begin{aligned} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda &= \int_{\lambda_s}^{\lambda_t} e^{\int_{\lambda_s}^{\lambda} (\boldsymbol{l}_\tau + \boldsymbol{s}_\tau)\mathrm{d}\tau} (\lambda - \lambda_s)^k \mathrm{d}\lambda \\ &< \int_{\lambda_s}^{\lambda_t} e^{C_2(\lambda - \lambda_s)} (\lambda - \lambda_s)^k \mathrm{d}\lambda \\ &= \mathcal{O}(h^{k+1}) \end{aligned} \tag{23}$$

Next we examine the order of $\frac{\hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) - \boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s)}{k!}$. Under Assumption B.1 and Assumption B.2, since $\boldsymbol{g}_\theta$ is elementary function of $\boldsymbol{\epsilon}_\theta$ and $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$, we know $\boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s), k = 1, \ldots, n$ exist and are continuous. Adopting the notations in Eq. (13), by Taylor expansion, we have

$$\begin{aligned} \boldsymbol{g}_{i_1} &= \boldsymbol{g}_s + \delta_1 \boldsymbol{g}_s^{(1)} + \delta_1^2 \boldsymbol{g}_s^{(2)} + \cdots + \delta_1^n \boldsymbol{g}_s^{(n)} + \mathcal{O}(\delta_1^{n+1}) \\ \boldsymbol{g}_{i_2} &= \boldsymbol{g}_s + \delta_2 \boldsymbol{g}_s^{(1)} + \delta_2^2 \boldsymbol{g}_s^{(2)} + \cdots + \delta_2^n \boldsymbol{g}_s^{(n)} + \mathcal{O}(\delta_2^{n+1}) \\ &\cdots \\ \boldsymbol{g}_{i_n} &= \boldsymbol{g}_s + \delta_n \boldsymbol{g}_s^{(1)} + \delta_n^2 \boldsymbol{g}_s^{(2)} + \cdots + \delta_n^n \boldsymbol{g}_s^{(n)} + \mathcal{O}(\delta_n^{n+1}) \end{aligned} \tag{24}$$

Comparing it with Eq. (13), we have

$$\begin{pmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^n \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n & \delta_n^2 & \cdots & \delta_n^n \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} - \boldsymbol{g}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)} - \boldsymbol{g}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)} - \boldsymbol{g}_s^{(n)}}{n!} \end{pmatrix} = \begin{pmatrix} \mathcal{O}(\delta_1^{n+1}) \\ \mathcal{O}(\delta_2^{n+1}) \\ \vdots \\ \mathcal{O}(\delta_n^{n+1}) \end{pmatrix} \tag{25}$$

From Assumption B.3, we know there exists some constants r_k so that $\delta_k = r_k h, k = 1, \dots, n$. Thus

$$\begin{pmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^n \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n & \delta_n^2 & \cdots & \delta_n^n \end{pmatrix} = \begin{pmatrix} r_1 & r_1^2 & \cdots & r_1^n \\ r_2 & r_2^2 & \cdots & r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_n & r_n^2 & \cdots & r_n^n \end{pmatrix} \begin{pmatrix} h & & & \\ & h^2 & & \\ & & \ddots & \\ & & & h^n \end{pmatrix}, \begin{pmatrix} \mathcal{O}(\delta_1^{n+1}) \\ \mathcal{O}(\delta_2^{n+1}) \\ \vdots \\ \mathcal{O}(\delta_n^{n+1}) \end{pmatrix} = \begin{pmatrix} \mathcal{O}(h^{n+1}) \\ \mathcal{O}(h^{n+1}) \\ \vdots \\ \mathcal{O}(h^{n+1}) \end{pmatrix} \tag{26}$$

And finally we have

$$\begin{aligned} \begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} - \boldsymbol{g}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)} - \boldsymbol{g}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)} - \boldsymbol{g}_s^{(n)}}{n!} \end{pmatrix} &= \begin{pmatrix} h^{-1} & & & \\ & h^{-2} & & \\ & & \ddots & \\ & & & h^{-n} \end{pmatrix} \begin{pmatrix} r_1 & r_1^2 & \cdots & r_1^n \\ r_2 & r_2^2 & \cdots & r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_n & r_n^2 & \cdots & r_n^n \end{pmatrix}^{-1} \begin{pmatrix} \mathcal{O}(h^{n+1}) \\ \mathcal{O}(h^{n+1}) \\ \vdots \\ \mathcal{O}(h^{n+1}) \end{pmatrix} \\ &= \begin{pmatrix} \mathcal{O}(h^n) \\ \mathcal{O}(h^{n-1}) \\ \vdots \\ \mathcal{O}(h^1) \end{pmatrix} \end{aligned} \tag{27}$$

Substitute Eq. (22), Eq. (23) and Eq. (27) into Eq. (21), we can conclude that $\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t = \mathcal{O}(h^{n+2})$. □

B.2.2 Global

First we provide a lemma which gives the local truncation error given inexact previous values when estimating the high-order derivatives.

Lemma B.7. *(Local truncation error with inexact previous values) Suppose inexact values $\hat{\boldsymbol{x}}_{\lambda_{i_k}}, k = 1, \dots, n$ and $\hat{\boldsymbol{x}}_s$ are used in Eq.* (13) *to estimate the high-order derivatives, then the local truncation error of the local approximation Eq.* (14) *satisfies*

$$\Delta_t = \frac{\alpha_t \boldsymbol{A}(\lambda_s, \lambda_t)}{\alpha_s} \Delta_s + \mathcal{O}(h) \left(\mathcal{O}(\Delta_s) + \sum_{k=1}^{n} \mathcal{O}(\Delta_{\lambda_{i_k}}) + \mathcal{O}(h^{n+1}) \right) \tag{28}$$

where $\Delta_\cdot \coloneqq \hat{\boldsymbol{x}}_\cdot - \boldsymbol{x}_\cdot, h \coloneqq \lambda_t - \lambda_s$.

Proof. By replacing $\boldsymbol{x}_\cdot$ with $\hat{\boldsymbol{x}}_\cdot$ in Eq. (13) and subtracting Eq. (12) from Eq. (14), the expression for the local truncation error becomes

$$\begin{aligned} \Delta_t = \frac{\alpha_t \boldsymbol{A}(\lambda_s, \lambda_t)}{\alpha_s} \Delta_s &- \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \left(\boldsymbol{g}_\theta(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) \right) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \mathrm{d}\lambda \\ &- \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \sum_{k=1}^{n} \frac{\hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) - \boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s)}{k!} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda + \mathcal{O}(h^{n+2}) \end{aligned} \tag{29}$$

And the linear system for solving $\boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s), k = 1, \dots, n$ becomes

$$\begin{pmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^n \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n & \delta_n^2 & \cdots & \delta_n^n \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)}}{n!} \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{g}}_{i_1} - \hat{\boldsymbol{g}}_s \\ \hat{\boldsymbol{g}}_{i_2} - \hat{\boldsymbol{g}}_s \\ \vdots \\ \hat{\boldsymbol{g}}_{i_n} - \hat{\boldsymbol{g}}_s \end{pmatrix} \tag{30}$$

where $\hat{\boldsymbol{g}}_\cdot = \boldsymbol{g}_\theta(\hat{\boldsymbol{x}}_{\lambda_\cdot}, \lambda_\cdot)$. Under Assumption B.4, we know $\hat{\boldsymbol{g}}_\cdot - \boldsymbol{g}_\cdot = \mathcal{O}(\Delta_{\lambda_\cdot})$. Thus, under Assumption B.1, Assumption B.2 and Assumption B.3, similar to the deduction in the last section,

we have

$$\begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} - \boldsymbol{g}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)} - \boldsymbol{g}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)} - \boldsymbol{g}_s^{(n)}}{n!} \end{pmatrix} = \begin{pmatrix} h^{-1} & & & \\ & h^{-2} & & \\ & & \ddots & \\ & & & h^{-n} \end{pmatrix} \begin{pmatrix} r_1 & r_1^2 & \cdots & r_1^n \\ r_2 & r_2^2 & \cdots & r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_n & r_n^2 & \cdots & r_n^n \end{pmatrix}^{-1} \begin{pmatrix} \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_1}}) \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_2}}) \\ \vdots \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_n}}) \end{pmatrix} \tag{31}$$

Besides, under Assumption B.2, the orders of the other coefficients are the same as we obtain in the last section:

$$\boldsymbol{A}(\lambda_s, \lambda_t) = \mathcal{O}(1), \quad \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda = \mathcal{O}(h^{k+1}) \tag{32}$$

Thus

$$\begin{aligned}
&\sum_{k=1}^{n} \frac{\hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) - \boldsymbol{g}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s)}{k!} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^k \mathrm{d}\lambda \\
=& \begin{pmatrix} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^1 \mathrm{d}\lambda \\ \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^2 \mathrm{d}\lambda \\ \vdots \\ \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)(\lambda - \lambda_s)^n \mathrm{d}\lambda \end{pmatrix}^\top \begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} - \boldsymbol{g}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)} - \boldsymbol{g}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)} - \boldsymbol{g}_s^{(n)}}{n!} \end{pmatrix} \\
=& \begin{pmatrix} \mathcal{O}(h^2) \\ \mathcal{O}(h^3) \\ \vdots \\ \mathcal{O}(h^{n+1}) \end{pmatrix}^\top \begin{pmatrix} h^{-1} & & & \\ & h^{-2} & & \\ & & \ddots & \\ & & & h^{-n} \end{pmatrix} \begin{pmatrix} r_1 & r_1^2 & \cdots & r_1^n \\ r_2 & r_2^2 & \cdots & r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_n & r_n^2 & \cdots & r_n^n \end{pmatrix}^{-1} \begin{pmatrix} \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_1}}) \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_2}}) \\ \vdots \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_n}}) \end{pmatrix} \\
=& \begin{pmatrix} \mathcal{O}(h) \\ \mathcal{O}(h) \\ \vdots \\ \mathcal{O}(h) \end{pmatrix}^\top \begin{pmatrix} r_1 & r_1^2 & \cdots & r_1^n \\ r_2 & r_2^2 & \cdots & r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_n & r_n^2 & \cdots & r_n^n \end{pmatrix}^{-1} \begin{pmatrix} \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_1}}) \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_2}}) \\ \vdots \\ \mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_n}}) \end{pmatrix} \\
=& \sum_{k=1}^{n} \mathcal{O}(h)\mathcal{O}(h^{n+1} + \Delta_s + \Delta_{\lambda_{i_k}})
\end{aligned} \tag{33}$$

Combining Eq. (29), Eq. (32) and Eq. (33), we can obtain the conclusion in Eq. (28). □

Then we prove Theorem 3.3 below.

Proof. (Proof of Theorem 3.3)

As we have discussed, the predictor step from t_{m-1} to t_m is a special case of the local approximation Eq. (14) with $(t_{i_n}, \ldots, t_{i_1}, s, t) = (t_{m-n-1}, \ldots, t_{m-2}, t_{m-1}, t_m)$. By Lemma B.7 we have

$$\Delta_m = \frac{\alpha_{t_m} \boldsymbol{A}(\lambda_{t_{m-1}}, \lambda_{t_m})}{\alpha_{t_{m-1}}} \Delta_{m-1} + \mathcal{O}(h) \left(\sum_{k=0}^{n} \mathcal{O}(\Delta_{m-k-1}) + \mathcal{O}(h^{n+1}) \right) \tag{34}$$

It follows that there exists constants C, C_0 irrelevant to h, so that

$$|\Delta_m| \leq \left(\frac{\alpha_{t_m} \boldsymbol{A}(\lambda_{t_{m-1}}, \lambda_{t_m})}{\alpha_{t_{m-1}}} + Ch \right) |\Delta_{m-1}| + Ch \sum_{k=0}^{n} |\Delta_{m-k-1}| + C_0 h^{n+2} \tag{35}$$

Denote $f_m \coloneqq \max_{0 \leq i \leq m} |\Delta_i|$, we then have

$$|\Delta_m| \leq \left(\frac{\alpha_{t_m} \boldsymbol{A}(\lambda_{t_{m-1}}, \lambda_{t_m})}{\alpha_{t_{m-1}}} + C_1 h \right) f_{m-1} + C_0 h^{n+2} \tag{36}$$

Since $\frac{\alpha_{t_m}\boldsymbol{A}(\lambda_{t_{m-1}},\lambda_{t_m})}{\alpha_{t_{m-1}}} \to 1$ when $h \to 0$ and it has bounded first-order derivative due to Assumption B.2, there exists a constant C_2, so that for any $C \geq C_2$, $\frac{\alpha_{t_m}\boldsymbol{A}(\lambda_{t_{m-1}},\lambda_{t_m})}{\alpha_{t_{m-1}}} + Ch > 1$ for sufficiently small h. Thus, by taking $C_3 = \max\{C_1, C_2\}$, we have

$$f_m \leq \left(\frac{\alpha_{t_m}\boldsymbol{A}(\lambda_{t_{m-1}},\lambda_{t_m})}{\alpha_{t_{m-1}}} + C_3 h\right) f_{m-1} + C_0 h^{n+2} \tag{37}$$

Denote $A_{m-1} \coloneqq \frac{\alpha_{t_m}\boldsymbol{A}(\lambda_{t_{m-1}},\lambda_{t_m})}{\alpha_{t_{m-1}}} + C_3 h$, by repeating Eq. (37), we have

$$f_M \leq \left(\prod_{i=n}^{M-1} A_i\right) f_n + \left(\sum_{i=n+1}^{M}\prod_{j=i}^{M-1} A_j\right) C_0 h^{n+2} \tag{38}$$

By Assumption B.5, $h = \mathcal{O}(1/M)$, so we have

$$\begin{aligned}
\prod_{i=n}^{M-1} A_i &= \frac{\alpha_{t_M}\boldsymbol{A}(\lambda_{t_n},\lambda_{t_M})}{\alpha_{t_n}} \prod_{i=n}^{M-1}\left(1 + \frac{\alpha_{t_{i-1}} C_3 h}{\alpha_{t_i}\boldsymbol{A}(\lambda_{t_{i-1}},\lambda_{t_i})}\right) \\
&\leq \frac{\alpha_{t_M}\boldsymbol{A}(\lambda_{t_n},\lambda_{t_M})}{\alpha_{t_n}} \prod_{i=n}^{M-1}\left(1 + \frac{\alpha_{t_{i-1}} C_4}{\alpha_{t_i}\boldsymbol{A}(\lambda_{t_{i-1}},\lambda_{t_i}) M}\right) \\
&\leq \frac{\alpha_{t_M}\boldsymbol{A}(\lambda_{t_n},\lambda_{t_M})}{\alpha_{t_n}} \left(1 + \frac{\sigma}{M}\right)^{M-n} \\
&\leq C_5 e^{\sigma}
\end{aligned} \tag{39}$$

where $\sigma = \max_{n \leq i \leq M-1} \frac{\alpha_{t_{i-1}} C_4}{\alpha_{t_i}\boldsymbol{A}(\lambda_{t_{i-1}},\lambda_{t_i})}$. Then denote $\beta \coloneqq \max_{n+1 \leq i \leq M} \frac{\alpha_{t_M}\boldsymbol{A}(\lambda_{t_i},\lambda_{t_M})}{\alpha_{t_i}}$, we have

$$\begin{aligned}
\sum_{i=n+1}^{M}\prod_{j=i}^{M-1} A_j &\leq \sum_{i=n+1}^{M} \frac{\alpha_{t_M}\boldsymbol{A}(\lambda_{t_i},\lambda_{t_M})}{\alpha_{t_i}} \left(1 + \frac{\sigma}{M}\right)^{M-i} \\
&\leq \beta \sum_{i=0}^{M-n-1} \left(1 + \frac{\sigma}{M}\right)^{i} \\
&= \frac{\beta M}{\sigma}\left[\left(1 + \frac{\sigma}{M}\right)^{M-n} - 1\right] \\
&\leq C_6 \left(e^{\sigma} - 1\right) M
\end{aligned} \tag{40}$$

Then we substitute Eq. (39) and Eq. (40) into Eq. (38). Note that $M = \mathcal{O}(1/h)$ by Assumption B.5, and $f_n = \mathcal{O}(h^{n+1})$ by Assumption B.6, finally we conclude that $|\Delta_M| \leq f_M = \mathcal{O}(h^{n+1})$. □

B.3 Pseudo-Order Solver

First we provide a lemma which gives the explicit solution to Eq. (15).

Lemma B.8. *The solution to Eq.* (15) *is*

$$\frac{\hat{\boldsymbol{g}}_s^{(k)}}{k!} = \sum_{p=1}^{k} \frac{\boldsymbol{g}_{i_p} - \boldsymbol{g}_{i_0}}{\prod_{q=0, q\neq p}^{k}(\delta_p - \delta_q)} \tag{41}$$

Proof. Denote

$$\boldsymbol{R}_k = \begin{pmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^k \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^k \\ \vdots & \vdots & \ddots & \vdots \\ \delta_k & \delta_k^2 & \cdots & \delta_k^k \end{pmatrix} \tag{42}$$

Then the solution to Eq. (15) can be expressed as

$$\frac{\hat{\boldsymbol{g}}_s^{(k)}}{k!} = \sum_{p=1}^{k} (\boldsymbol{R}_k^{-1})_{kp} (\boldsymbol{g}_{i_p} - \boldsymbol{g}_{i_0}) \tag{43}$$

where $(\boldsymbol{R}_k^{-1})_{kp}$ is the element of $\boldsymbol{R}_k^{-1}$ at the k-th row and the p-th column. From previous studies of the inversion of the Vandermonde matrix [12], we know

$$(\boldsymbol{R}_k^{-1})_{kp} = \frac{1}{\delta_p \prod_{q=1,q\neq p}^{k} (\delta_p - \delta_q)} = \frac{1}{(\delta_p - \delta_0) \prod_{q=1,q\neq p}^{k} (\delta_p - \delta_q)} \tag{44}$$

Substituting Eq. (44) into Eq. (43), we finish the proof. □

Then we prove Theorem 3.4 below:

Proof. (Proof of Theorem 3.4) First we use mathematical induction to prove that

$$D_l^{(k)} = \tilde{D}_l^{(k)} \coloneqq \sum_{p=1}^{k} \frac{\boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l}}{\prod_{q=0,q\neq p}^{k} (\delta_{l+p} - \delta_{l+q})}, \quad 1 \leq k \leq n, 0 \leq l \leq n-k \tag{45}$$

For $k = 1$, Eq. (45) holds by the definition of $D_l^{(k)}$. Suppose the equation holds for k, we then prove it holds for $k+1$.

Define the Lagrange polynomial which passes $(\delta_{l+p}, \boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l})$ for $0 \leq p \leq k$:

$$P_l^{(k)}(x) \coloneqq \sum_{p=1}^{k} \left(\boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l}\right) \prod_{q=0,q\neq p}^{k} \frac{x - \delta_{l+q}}{\delta_{l+p} - \delta_{l+q}}, \quad 1 \leq k \leq n, 0 \leq l \leq n-k \tag{46}$$

Then $\tilde{D}_l^{(k)} = P_l^{(k)}(x)[x^k]$ is the coefficients before the highest-order term x^k in $P_l^{(k)}(x)$. We then prove that $P_l^{(k)}(x)$ satisfies the following recurrence relation:

$$P_l^{(k)}(x) = \tilde{P}_l^{(k)}(x) \coloneqq \frac{(x - \delta_l) P_{l+1}^{(k-1)}(x) - (x - \delta_{l+k}) P_l^{(k-1)}(x)}{\delta_{l+k} - \delta_l} \tag{47}$$

By definition, $P_{l+1}^{(k-1)}(x)$ is the $(k-1)$-th order polynomial which passes $(\delta_{l+p}, \boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l})$ for $1 \leq p \leq k$, and $P_l^{(k-1)}(x)$ is the $(k-1)$-th order polynomial which passes $(\delta_{l+p}, \boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l})$ for $0 \leq p \leq k-1$.

Thus, for $1 \leq p \leq k-1$, we have

$$\tilde{P}_l^{(k)}(\delta_{l+p}) = \frac{(\delta_{l+p} - \delta_l) P_{l+1}^{(k-1)}(\delta_{l+p}) - (\delta_{l+p} - \delta_{l+k}) P_l^{(k-1)}(\delta_{l+p})}{\delta_{l+k} - \delta_l} = \boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l} \tag{48}$$

For $p = 0$, we have

$$\tilde{P}_l^{(k)}(\delta_l) = \frac{(\delta_l - \delta_l) P_{l+1}^{(k-1)}(\delta_l) - (\delta_l - \delta_{l+k}) P_l^{(k-1)}(\delta_l)}{\delta_{l+k} - \delta_l} = \boldsymbol{g}_{i_l} - \boldsymbol{g}_{i_l} \tag{49}$$

for $p = k$, we have

$$\tilde{P}_l^{(k)}(\delta_{l+k}) = \frac{(\delta_{l+k} - \delta_l) P_{l+1}^{(k-1)}(\delta_{l+k}) - (\delta_{l+k} - \delta_{l+k}) P_l^{(k-1)}(\delta_{l+k})}{\delta_{l+k} - \delta_l} = \boldsymbol{g}_{i_{l+k}} - \boldsymbol{g}_{i_l} \tag{50}$$

Therefore, $\tilde{P}_l^{(k)}(x)$ is the k-th order polynomial which passes $k+1$ distince points $(\delta_{l+p}, \boldsymbol{g}_{i_{l+p}} - \boldsymbol{g}_{i_l})$ for $0 \leq p \leq k$. Due to the uniqueness of the Lagrange polynomial, we can conclude that $P_l^{(k)}(x) = \tilde{P}_l^{(k)}(x)$. By taking the coefficients of the highest-order term, we obtain

$$\tilde{D}_l^{(k)} = \frac{\tilde{D}_{l+1}^{(k-1)} - \tilde{D}_l^{(k-1)}}{\delta_{l+k} - \delta_l} \tag{51}$$

where by the induction hypothesis we have $D_{l+1}^{(k-1)} = \tilde{D}_{l+1}^{(k-1)}, D_l^{(k-1)} = \tilde{D}_l^{(k-1)}$. Comparing Eq. (51) with the recurrence relation of $D_l^{(k)}$ in Eq. (16), it follows that $D_l^{(k)} = \tilde{D}_l^{(k)}$, which completes the mathematical induction.

Finally, by comparing the expression for $\tilde{D}_l^{(k)}$ in Eq. (45) and the expression for $\hat{\boldsymbol{g}}_s^{(k)}$ in Lemma B.8, we can conclude that $\hat{\boldsymbol{g}}_s^{(k)} = k! D_0^{(k)}$. □

B.4 Local Unbiasedness

Proof. (Proof of Theorem 3.2) Subtracting the local exact solution in Eq. (9) from the $(n+1)$-th order local approximation in Eq. (14), we have the local truncation error

$$\begin{aligned}
\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t &= \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \left(\int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda) \mathrm{d}\lambda - \sum_{k=0}^{n} \hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \frac{(\lambda - \lambda_s)^k}{k!} \mathrm{d}\lambda \right) \\
&= \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \left(\boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) \right) \mathrm{d}\lambda \\
&- \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \sum_{k=1}^{n} \hat{\boldsymbol{g}}_\theta^{(k)}(\boldsymbol{x}_{\lambda_s}, \lambda_s) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \frac{(\lambda - \lambda_s)^k}{k!} \mathrm{d}\lambda \\
&= \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \left(\boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) \right) \mathrm{d}\lambda \\
&- \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t) \sum_{k=1}^{n} \left(\sum_{l=1}^{n} (\boldsymbol{R}_n^{-1})_{kl} (\boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_{i_l}}, \lambda_{i_l}) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s)) \right) \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \frac{(\lambda - \lambda_s)^k}{k!} \mathrm{d}\lambda
\end{aligned} \tag{52}$$

where $\boldsymbol{x}_\lambda$ is on the ground-truth ODE trajectory passing $\boldsymbol{x}_{\lambda_s}$, and $(\boldsymbol{R}_n^{-1})_{kl}$ is the element of the inverse matrix $\boldsymbol{R}_n^{-1}$ at the k-th row and the l-th column, as discussed in the proof of Lemma B.8. By Newton-Leibniz theorem, we have

$$\boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) = \int_{\lambda_s}^{\lambda} \boldsymbol{g}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) \mathrm{d}\tau \tag{53}$$

Also, since $\boldsymbol{x}_{\lambda_{i_l}}, l = 1, \ldots, n$ are on the ground-truth ODE trajectory passing $\boldsymbol{x}_{\lambda_s}$, we have

$$\boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_{i_l}}, \lambda_{i_l}) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) = \int_{\lambda_s}^{\lambda_{i_l}} \boldsymbol{g}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) \mathrm{d}\tau \tag{54}$$

where

$$\boldsymbol{g}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) = e^{-\int_{\lambda_s}^{\tau} \boldsymbol{s}_r \mathrm{d}r} \left(\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) - \boldsymbol{s}_\tau \boldsymbol{f}_\theta(\boldsymbol{x}_\tau, \tau) - \boldsymbol{b}_\tau \right) \tag{55}$$

Note that $\boldsymbol{s}_\lambda, \boldsymbol{l}_\lambda$ are the solution to the least square problem in Eq. (11), which makes sure $\mathbb{E}_{p_\tau^\theta(\boldsymbol{x}_\tau)} \left[\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) - \boldsymbol{s}_\tau \boldsymbol{f}_\theta(\boldsymbol{x}_\tau, \tau) - \boldsymbol{b}_\tau \right] = 0$. It follows that $\mathbb{E}_{p_{\lambda_s}^\theta(\boldsymbol{x}_{\lambda_s})} \left[\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\tau, \tau) - \boldsymbol{s}_\tau \boldsymbol{f}_\theta(\boldsymbol{x}_\tau, \tau) - \boldsymbol{b}_\tau \right] = 0$, since $\boldsymbol{x}_\tau$ is on the ground-truth ODE trajectory passing $\boldsymbol{x}_{\lambda_s}$. Therefore, we have $\mathbb{E}_{p_{\lambda_s}^\theta(\boldsymbol{x}_{\lambda_s})} \left[\boldsymbol{g}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) \right] = 0$ and $\mathbb{E}_{p_{\lambda_s}^\theta(\boldsymbol{x}_{\lambda_s})} \left[\boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_{i_l}}, \lambda_{i_l}) - \boldsymbol{g}_\theta(\boldsymbol{x}_{\lambda_s}, \lambda_s) \right] = 0$. Substitute them into Eq. (52), we conclude that $\mathbb{E}_{p_{\lambda_s}^\theta(\boldsymbol{x}_{\lambda_s})} \left[\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t \right] = 0$.

□

C Implementation Details

C.1 Computing the EMS and Related Integrals in the ODE Formulation

The ODE formulation and local approximation require computing some complex integrals involving $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$. In this section, we'll give details about how to estimate $\boldsymbol{l}_\lambda^*, \boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$ on a few datapoints, and how to use them to compute the integrals efficiently.

C.1.1 Computing the EMS

First for the computing of $\boldsymbol{l}_\lambda^*$ in Eq. (5), note that

$$\nabla_{\boldsymbol{x}} \boldsymbol{N}_\theta(\boldsymbol{x}_\lambda, \lambda) = \sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}(\boldsymbol{x}_\lambda, \lambda) - \text{diag}(\boldsymbol{l}_\lambda) \tag{56}$$

Since $\text{diag}(\boldsymbol{l}_\lambda)$ is a diagonal matrix, minimizing $\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\|\nabla_{\boldsymbol{x}} \boldsymbol{N}_\theta(\boldsymbol{x}_\lambda, \lambda)\|_F^2\right]$ is equivalent to minimizing $\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\|\text{diag}^{-1}(\nabla_{\boldsymbol{x}} \boldsymbol{N}_\theta(\boldsymbol{x}_\lambda, \lambda))\|_2^2\right] = \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\|\text{diag}^{-1}(\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}(\boldsymbol{x}_\lambda, \lambda)) - \boldsymbol{l}_\lambda\|_2^2\right]$, where diag^{-1} denotes the operator that takes the diagonal of a matrix as a vector. Thus we have $\boldsymbol{l}_\lambda^* = \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\text{diag}^{-1}(\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}(\boldsymbol{x}_\lambda, \lambda))\right]$.

However, this formula for $\boldsymbol{l}_\lambda^*$ requiring computing the diagonal of the full Jacobian of the noise prediction model, which typically has $\mathcal{O}(d^2)$ time complexity for d-dimensional data and is unacceptable when d is large. Fortunately, the cost can be reduced to $\mathcal{O}(d)$ by utilizing stochastic diagonal estimators and employing the efficient Jacobian-vector-product operator provided by forward-mode automatic differentiation in deep learning frameworks.

For a d-by-d matrix $\boldsymbol{D}$, its diagonal can be unbiasedly estimated by [4]

$$\text{diag}^{-1}(\boldsymbol{D}) = \left[\sum_{k=1}^{s} (\boldsymbol{D}\boldsymbol{v}_k) \odot \boldsymbol{v}_k\right] \oslash \left[\sum_{k=1}^{s} \boldsymbol{v}_k \odot \boldsymbol{v}_k\right] \tag{57}$$

where $\boldsymbol{v}_k \sim p(\boldsymbol{v})$ are d-dimensional i.i.d. samples with zero mean, $\odot$ is the element-wise multiplication i.e., Hadamard product, and $\oslash$ is the element-wise division. The stochastic diagonal estimator is analogous to the famous Hutchinson's trace estimator [21]. By taking $p(\boldsymbol{v})$ as the Rademacher distribution, we have $\boldsymbol{v}_k \odot \boldsymbol{v}_k = \boldsymbol{1}$, and the denominator can be omitted. For simplicity, we use regular multiplication and division symbol, assuming they are element-wise between vectors. Then $\boldsymbol{l}_\lambda^*$ can be expressed as:

$$\boldsymbol{l}_\lambda^* = \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)p(\boldsymbol{v})}\left[(\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)\boldsymbol{v})\boldsymbol{v}\right] \tag{58}$$

which is unbiased estimation when we replace the expectation with mean on finite samples $\boldsymbol{x}_\lambda \sim p_\lambda^\theta(\boldsymbol{x}_\lambda), \boldsymbol{v} \sim p(\boldsymbol{v})$. The process for estimating $\boldsymbol{l}_\lambda^*$ can easily be paralleled on multiple devices by computing $\sum(\sigma_\lambda \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)\boldsymbol{v})\boldsymbol{v}$ on separate datapoints and gather them in the end.

Next, for the computing of $\boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$ in Eq. (11), note that it's a simple least square problem. By taking partial derivatives w.r.t. $\boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$ and set them to 0, we have

$$\begin{cases} \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\left(\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{s}_\lambda^* \boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{b}_\lambda^*\right) \boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)\right] = 0 \\ \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{s}_\lambda^* \boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{b}_\lambda^*\right] = 0 \end{cases} \tag{59}$$

And we obtain the explicit formula for $\boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$

$$\boldsymbol{s}_\lambda^* = \frac{\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda)\right] - \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)\right]\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}\left[\boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda)\right]}{\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)] - \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)]\mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)]} \tag{60}$$

$$\boldsymbol{b}_\lambda^* = \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}[\boldsymbol{f}^{(1)}(\boldsymbol{x}_\lambda, \lambda)] - \boldsymbol{s}_\lambda^* \mathbb{E}_{p_\lambda^\theta(\boldsymbol{x}_\lambda)}[\boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)] \tag{61}$$

which are unbiased least square estimators when we replace the expectation with mean on finite samples $\boldsymbol{x}_\lambda \sim p_\lambda^\theta(\boldsymbol{x}_\lambda)$. Also, the process for estimating $\boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$ can be paralleled on multiple devices by computing $\sum \boldsymbol{f}_\theta, \sum \boldsymbol{f}_\theta^{(1)}, \sum \boldsymbol{f}_\theta \boldsymbol{f}_\theta, \sum \boldsymbol{f}_\theta \boldsymbol{f}_\theta^{(1)}$ on separate datapoints and gather them in the end. Thus, the estimation of $\boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$ involving evaluating $\boldsymbol{f}_\theta$ and $\boldsymbol{f}_\theta^{(1)}$ on $\boldsymbol{x}_\lambda$. $\boldsymbol{f}_\theta$ is a direct transformation of $\boldsymbol{\epsilon}_\theta$ and requires a single forward pass. For $\boldsymbol{f}_\theta^{(1)}$, we have

$$\begin{aligned} \boldsymbol{f}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda) &= \frac{\partial \boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)}{\partial \lambda} + \nabla_{\boldsymbol{x}} \boldsymbol{f}_\theta(\boldsymbol{x}_\lambda, \lambda)\frac{\mathrm{d}\boldsymbol{x}_\lambda}{\mathrm{d}\lambda} \\ &= e^{-\lambda}\left(\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)\right) - \frac{\dot{\boldsymbol{l}}_\lambda \alpha_\lambda - \dot{\alpha}_\lambda \boldsymbol{l}_\lambda}{\alpha_\lambda^2}\boldsymbol{x}_\lambda - \frac{\boldsymbol{l}_\lambda}{\alpha_\lambda}\left(\frac{\dot{\alpha}_\lambda}{\alpha_\lambda}\boldsymbol{x}_\lambda - \sigma_\lambda \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda)\right) \\ &= e^{-\lambda}\left((\boldsymbol{l}_\lambda - 1)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) + \boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda)\right) - \frac{\dot{\boldsymbol{l}}_\lambda \boldsymbol{x}_\lambda}{\alpha_\lambda} \end{aligned} \tag{62}$$

After we obtain $\boldsymbol{l}_\lambda$, $\dot{\boldsymbol{l}}_\lambda$ can be estimated by finite difference. To compute $\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda)$, we have

$$\begin{aligned}\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_\lambda, \lambda) &= \partial_\lambda \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) + \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) \frac{\mathrm{d}\boldsymbol{x}_\lambda}{\mathrm{d}\lambda} \\ &= \partial_\lambda \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) + \nabla_{\boldsymbol{x}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) \left(\frac{\dot{\alpha}_\lambda}{\alpha_\lambda} \boldsymbol{x}_\lambda - \sigma_\lambda \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\lambda, \lambda) \right)\end{aligned} \tag{63}$$

which can also be computed with the Jacobian-vector-product operator.

In conclusion, for any λ, $\boldsymbol{l}_\lambda^*, \boldsymbol{s}_\lambda^*, \boldsymbol{b}_\lambda^*$ can be efficiently and unbiasedly estimated by sampling a few datapoints $\boldsymbol{x}_\lambda \sim p_\lambda^\theta(\boldsymbol{x}_\lambda)$ and using the Jacobian-vector-product.

C.1.2 Integral Precomputing

In the local approximation in Eq. (14), there are three integrals involving the EMS, which are $\boldsymbol{A}(\lambda_s, \lambda_t)$, $\int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \boldsymbol{B}_{\lambda_s}(\lambda) \mathrm{d}\lambda$, $\int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \frac{(\lambda - \lambda_s)^k}{k!} \mathrm{d}\lambda$. Define the following terms, which are also evaluated at $\lambda_{j_0}, \lambda_{j_1}, \ldots, \lambda_{j_N}$ and can be precomputed in $\mathcal{O}(N)$ time:

$$\begin{aligned}\boldsymbol{L}_\lambda &= \int_{\lambda_T}^{\lambda} \boldsymbol{l}_\tau \mathrm{d}\tau \\ \boldsymbol{S}_\lambda &= \int_{\lambda_T}^{\lambda} \boldsymbol{s}_\tau \mathrm{d}\tau \\ \boldsymbol{B}_\lambda &= \int_{\lambda_T}^{\lambda} e^{-\int_{\lambda_T}^{r} \boldsymbol{s}_\tau \mathrm{d}\tau} \boldsymbol{b}_r \mathrm{d}r = \int_{\lambda_T}^{\lambda} e^{-\boldsymbol{S}_r} \boldsymbol{b}_r \mathrm{d}r \\ \boldsymbol{C}_\lambda &= \int_{\lambda_T}^{\lambda} \left(e^{\int_{\lambda_T}^{u} (\boldsymbol{l}_\tau + \boldsymbol{s}_\tau) \mathrm{d}\tau} \int_{\lambda_T}^{u} e^{-\int_{\lambda_T}^{r} \boldsymbol{s}_\tau \mathrm{d}\tau} \boldsymbol{b}_r \mathrm{d}r \right) \mathrm{d}u = \int_{\lambda_T}^{\lambda} e^{\boldsymbol{L}_u + \boldsymbol{S}_u} \boldsymbol{B}_u \mathrm{d}u \\ \boldsymbol{I}_\lambda &= \int_{\lambda_T}^{\lambda} e^{\int_{\lambda_T}^{r} (\boldsymbol{l}_\tau + \boldsymbol{s}_\tau) \mathrm{d}\tau} \mathrm{d}r = \int_{\lambda_T}^{\lambda} e^{\boldsymbol{L}_r + \boldsymbol{S}_r} \mathrm{d}r\end{aligned} \tag{64}$$

Then for any λ_s, λ_t, we can verify that the first two integrals can be expressed as

$$\begin{aligned}\boldsymbol{A}(\lambda_s, \lambda_t) &= e^{\boldsymbol{L}_{\lambda_s} - \boldsymbol{L}_{\lambda_t}} \\ \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \boldsymbol{B}_{\lambda_s}(\lambda) \mathrm{d}\lambda &= e^{-\boldsymbol{L}_{\lambda_s}} (\boldsymbol{C}_{\lambda_t} - \boldsymbol{C}_{\lambda_s} - \boldsymbol{B}_{\lambda_s} (\boldsymbol{I}_{\lambda_t} - \boldsymbol{I}_{\lambda_s}))\end{aligned} \tag{65}$$

which can be computed in $\mathcal{O}(1)$ time. For the third and last integral, denote it as $\boldsymbol{E}_{\lambda_s, \lambda_t}^{(k)}$, i.e.,

$$\boldsymbol{E}_{\lambda_s, \lambda_t}^{(k)} = \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda) \frac{(\lambda - \lambda_s)^k}{k!} \mathrm{d}\lambda \tag{66}$$

We need to compute it for $0 \leq k \leq n$ and for every local transition time pair (λ_s, λ_t) in the sampling process. For $k = 0$, we have

$$\boldsymbol{E}_{\lambda_s, \lambda_t}^{(0)} = e^{-\boldsymbol{L}_{\lambda_s} - \boldsymbol{S}_{\lambda_s}} (\boldsymbol{I}_{\lambda_t} - \boldsymbol{I}_{\lambda_s}) \tag{67}$$

which can also be computed in $\mathcal{O}(1)$ time. But for $k > 0$, we no longer have such simplification technique. Still, for any fixed timestep schedule $\{\lambda_i\}_{i=0}^M$ during the sampling process, we can use a lazy precomputing strategy: compute $\boldsymbol{E}_{\lambda_{i-1}, \lambda_i}^{(k)}, 1 \leq i \leq M$ when generating the first sample, store it with a unique key (k, i) and retrieve it in $\mathcal{O}(1)$ in the following sampling process.

C.2 Algorithm

We provide the pseudocode of the local approximation and global solver in Algorithm 1 and Algorithm 2, which concisely describes how we implement DPM-Solver-v3.

Algorithm 1 $(n+1)$-th order local approximation: LUpdate_{n+1}

Require: noise schedule α_t, σ_t, coefficients $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$
Input: transition time pair (s, t), $\boldsymbol{x}_s$, n extra timesteps $\{t_{i_k}\}_{k=1}^n$, $\boldsymbol{g}_\theta$ values $(\boldsymbol{g}_{i_n}, \dots, \boldsymbol{g}_{i_1}, \boldsymbol{g}_s)$ at $\{(\boldsymbol{x}_{\lambda_{i_k}}, t_{i_k})\}_{k=1}^n$ and $(\boldsymbol{x}_s, s)$
Input Format: $\{t_{i_n}, \boldsymbol{g}_{i_n}\}, \dots, \{t_{i_1}, \boldsymbol{g}_{i_1}\}, \{s, \boldsymbol{x}_s, \boldsymbol{g}_s\}, t$

Compute $\boldsymbol{A}(\lambda_s, \lambda_t), \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)\boldsymbol{B}_{\lambda_s}(\lambda)\mathrm{d}\lambda, \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)\frac{(\lambda-\lambda_s)^k}{k!}\mathrm{d}\lambda$ (Appendix C.1.2)
$\delta_k = \lambda_{i_k} - \lambda_s, \quad k = 1, \dots, n$
$$\begin{pmatrix} \hat{\boldsymbol{g}}_s^{(1)} \\ \frac{\hat{\boldsymbol{g}}_s^{(2)}}{2!} \\ \vdots \\ \frac{\hat{\boldsymbol{g}}_s^{(n)}}{n!} \end{pmatrix} \leftarrow \begin{pmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^n \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n & \delta_n^2 & \cdots & \delta_n^n \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{g}_{i_1} - \boldsymbol{g}_s \\ \boldsymbol{g}_{i_2} - \boldsymbol{g}_s \\ \vdots \\ \boldsymbol{g}_{i_n} - \boldsymbol{g}_s \end{pmatrix} \text{ (Eq. (13))}$$
$$\hat{\boldsymbol{x}}_t \leftarrow \alpha_t \boldsymbol{A}(\lambda_s, \lambda_t)\left(\frac{\boldsymbol{x}_s}{\alpha_s} - \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)\boldsymbol{B}_{\lambda_s}(\lambda)\mathrm{d}\lambda - \sum_{k=0}^{n} \hat{\boldsymbol{g}}_s^{(k)} \int_{\lambda_s}^{\lambda_t} \boldsymbol{E}_{\lambda_s}(\lambda)\frac{(\lambda-\lambda_s)^k}{k!}\mathrm{d}\lambda\right)$$ (Eq. (14))

Output: $\hat{\boldsymbol{x}}_t$

Algorithm 2 $(n+1)$-th order multistep predictor-corrector algorithm

Require: noise prediction model $\boldsymbol{\epsilon}_\theta$, noise schedule α_t, σ_t, coefficients $\boldsymbol{l}_\lambda, \boldsymbol{s}_\lambda, \boldsymbol{b}_\lambda$, cache Q_1, Q_2
Input: timesteps $\{t_i\}_{i=0}^M$, initial value $\boldsymbol{x}_0$

$Q_1 \overset{cache}{\leftarrow} \boldsymbol{x}_0$
$Q_2 \overset{cache}{\leftarrow} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_0, t_0)$
for $m = 1$ **to** M **do**
  $n_m \leftarrow \min\{n+1, m\}$
  $\hat{\boldsymbol{x}}_{m-n_m}, \dots, \hat{\boldsymbol{x}}_{m-1} \overset{fetch}{\leftarrow} Q_1$
  $\hat{\boldsymbol{\epsilon}}_{m-n_m}, \dots, \hat{\boldsymbol{\epsilon}}_{m-1} \overset{fetch}{\leftarrow} Q_2$
  $\hat{\boldsymbol{g}}_l \leftarrow e^{-\int_{\lambda_{m-1}}^{\lambda_l} \boldsymbol{s}_\tau \mathrm{d}\tau}\frac{\sigma_{\lambda_l}\hat{\boldsymbol{\epsilon}}_l - \boldsymbol{l}_{\lambda_l}\hat{\boldsymbol{x}}_l}{\alpha_{\lambda_l}} - \int_{\lambda_{m-1}}^{\lambda_l} e^{-\int_{\lambda_s}^{r} \boldsymbol{s}_\tau \mathrm{d}\tau}\boldsymbol{b}_r \mathrm{d}r, \quad l = m - n_m, \dots, m-1$ (Eq. (8))
  $\hat{\boldsymbol{x}}_m \leftarrow \text{LUpdate}_{n_m}(\{t_{m-n_m}, \hat{\boldsymbol{g}}_{m-n_m}\}, \dots, \{t_{m-2}, \hat{\boldsymbol{g}}_{m-2}\}, \{t_{m-1}, \hat{\boldsymbol{x}}_{m-1}, \hat{\boldsymbol{g}}_{m-1}\}, t_m)$
  if $m \neq M$ **then**
    $\hat{\boldsymbol{\epsilon}}_m \leftarrow \boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_m, t_m)$
    $\hat{\boldsymbol{g}}_m \leftarrow e^{-\int_{\lambda_{m-1}}^{\lambda_m} \boldsymbol{s}_\tau \mathrm{d}\tau}\frac{\sigma_{\lambda_m}\hat{\boldsymbol{\epsilon}}_m - \boldsymbol{l}_{\lambda_m}\hat{\boldsymbol{x}}_m}{\alpha_{\lambda_m}} - \int_{\lambda_{m-1}}^{\lambda_m} e^{-\int_{\lambda_s}^{r} \boldsymbol{s}_\tau \mathrm{d}\tau}\boldsymbol{b}_r \mathrm{d}r$ (Eq. (8))
    $\hat{\boldsymbol{x}}_m^c \leftarrow \text{LUpdate}_{n_m}(\{t_{m-n_m+1}, \hat{\boldsymbol{g}}_{m-n_m+1}\}, \dots, \{t_{m-2}, \hat{\boldsymbol{g}}_{m-2}\}, \{t_m, \hat{\boldsymbol{g}}_m\}, \{t_{m-1}, \hat{\boldsymbol{x}}_{m-1}, \hat{\boldsymbol{g}}_{m-1}\}, t_m)$
    $\hat{\boldsymbol{\epsilon}}_m^c \leftarrow \hat{\boldsymbol{\epsilon}}_m + \boldsymbol{l}_{\lambda_m}(\hat{\boldsymbol{x}}_m^c - \hat{\boldsymbol{x}}_m)/\sigma_{\lambda_m}$ (to ensure $\hat{\boldsymbol{g}}_m^c = \hat{\boldsymbol{g}}_m$)
    $Q_1 \overset{cache}{\leftarrow} \hat{\boldsymbol{x}}_m^c$
    $Q_2 \overset{cache}{\leftarrow} \hat{\boldsymbol{\epsilon}}_m^c$
  end if
end for

Output: $\hat{\boldsymbol{x}}_M$

D Experiment Details

In this section, we provide more experiment details for each setting, including the codebases and the configurations for evaluation, EMS computing and sampling. Unless otherwise stated, we utilize the forward-mode automatic differentiation (`torch.autograd.forward_ad`) provided by PyTorch [41] to compute the Jacobian-vector-products (JVPs). Also, as stated in Section 3.4, we draw datapoints

$\boldsymbol{x}_\lambda$ from the marginal distribution q_λ defined by the forward diffusion process starting from some data distribution q_0, instead of the model distribution p_λ^θ.

D.1 ScoreSDE on CIFAR10

Codebase and evaluation For unconditional sampling on CIFAR10 [25], one experiment setting is based on the pretrained pixel-space diffusion model provided by ScoreSDE [54]. We use their official codebase of PyTorch implementation, and their checkpoint `checkpoint_8.pth` under `vp/cifar10_ddpmpp_deep_continuous` config. We adopt their own statistic file and code for computing FID.

EMS computing We estimate the EMS at $N = 1200$ uniform timesteps $\lambda_{j_0}, \lambda_{j_1}, \dots, \lambda_{j_N}$ by drawing $K = 4096$ datapoints $\boldsymbol{x}_{\lambda_0} \sim q_0$, where q_0 is the distribution of the training set. We compute two sets of EMS, corresponding to start time $\epsilon = 10^{-3}$ (NFE$\leq$ 10) and $\epsilon = 10^{-4}$ (NFE>10) in the sampling process respectively. The total time for EMS computing is $\sim$7h on 8 GPU cards of NVIDIA A40.

Sampling Following previous works [33, 34, 61], we use start time $\epsilon = 10^{-3}$ (NFE$\leq$ 10) and $\epsilon = 10^{-4}$ (NFE>10), end time $T = 1$ and adopt the uniform logSNR timestep schedule. For DPM-Solver-v3, we use 3rd-order predictor with 3rd-order corrector by default. Specially, we change to pseudo 3rd-order predictor at 5 NFE to further boost the performance.

D.2 EDM on CIFAR10

Codebase and evaluation For unconditional sampling on CIFAR10 [25], another experiment setting is based on the pretrained pixel-space diffusion model provided by EDM [22]. We use their official codebase of PyTorch implementation, and their checkpoint `edm-cifar10-32x32-uncond-vp.pkl`. For consistency, we borrow the statistic file and code from ScoreSDE [54] for computing FID.

EMS computing Since the pretrained models of EDM are stored within the pickles, we fail to use `torch.autograd.forward_ad` for computing JVPs. Instead, we use `torch.autograd.functional.jvp`, which is much slower since it employs the double backwards trick. We estimate two sets of EMS. One corresponds to $N = 1200$ uniform timesteps $\lambda_{j_0}, \lambda_{j_1}, \dots, \lambda_{j_N}$ and $K = 1024$ datapoints $\boldsymbol{x}_{\lambda_0} \sim q_0$, where q_0 is the distribution of the training set. The other corresponds to $N = 120, K = 4096$. They are used when NFE<10 and NFE$\geq$10 respectively. The total time for EMS computing is $\sim$3.5h on 8 GPU cards of NVIDIA A40.

Sampling Following EDM, we use start time $t_{\min} = 0.002$ and end time $t_{\max} = 80.0$, but adopt the uniform logSNR timestep schedule which performs better in practice. For DPM-Solver-v3, we use 3rd-order predictor and additionally employ 3rd-order corrector when NFE$\leq$ 6. Specially, we change to pseudo 3rd-order predictor at 5 NFE to further boost the performance.

D.3 Latent-Diffusion on LSUN-Bedroom

Codebase and evaluation The unconditional sampling on LSUN-Bedroom [58] is based on the pretrained latent-space diffusion model provided by Latent-Diffusion [45]. We use their official codebase of PyTorch implementation and their default checkpoint. We borrow the statistic file and code from Guided-Diffusion [10] for computing FID.

EMS computing We estimate the EMS at $N = 120$ uniform timesteps $\lambda_{j_0}, \lambda_{j_1}, \dots, \lambda_{j_N}$ by drawing $K = 1024$ datapoints $\boldsymbol{x}_{\lambda_0} \sim q_0$, where q_0 is the distribution of the latents of the training set. The total time for EMS computing is $\sim$12min on 8 GPU cards of NVIDIA A40.

Sampling Following previous works [61], we use start time $\epsilon = 10^{-3}$, end time $T = 1$ and adopt the uniform t timestep schedule. For DPM-Solver-v3, we use 3rd-order predictor with pseudo 4th-order corrector.

D.4 Guided-Diffusion on ImageNet-256

Codebase and evaluation The conditional sampling on ImageNet-256 [9] is based on the pretrained pixel-space diffusion model provided by Guided-Diffusion [10]. We use their official codebase of PyTorch implementation and their two checkpoints: the conditional diffusion model

256x256_diffusion.pt and the classifier 256x256_classifier.pt. We adopt their own statistic file and code for computing FID.

EMS computing We estimate the EMS at $N = 500$ uniform timesteps $\lambda_{j_0}, \lambda_{j_1}, \dots, \lambda_{j_N}$ by drawing $K = 1024$ datapoints $\boldsymbol{x}_{\lambda_0} \sim q_0$, where q_0 is the distribution of the training set. Also, we find that the FID metric on ImageNet-256 dataset behaves specially, and degenerated $\boldsymbol{l}_\lambda$ ($\boldsymbol{l}_\lambda = \mathbf{1}$) performs better. The total time for EMS computing is $\sim$9.5h on 8 GPU cards of NVIDIA A40.

Sampling Following previous works [33, 34, 61], we use start time $\epsilon = 10^{-3}$, end time $T = 1$ and adopt the uniform t timestep schedule. For DPM-Solver-v3, we use 2nd-order predictor with pseudo 3rd-order corrector.

D.5 Stable-Diffusion on MS-COCO2014 prompts

Codebase and evaluation The text-to-image sampling on MS-COCO2014 [28] prompts is based on the pretrained latent-space diffusion model provided by Stable-Diffusion [45]. We use their official codebase of PyTorch implementation and their checkpoint sd-v1-4.ckpt. We compute MSE on randomly selected captions from the MS-COCO2014 validation dataset, as detailed in Section 4.1.

EMS computing We estimate the EMS at $N = 250$ uniform timesteps $\lambda_{j_0}, \lambda_{j_1}, \dots, \lambda_{j_N}$ by drawing $K = 1024$ datapoints $\boldsymbol{x}_{\lambda_0} \sim q_0$. Since Stable-Diffusion is trained on LAION-5B dataset [49], there is a gap between the images in MS-COCO2014 validation dataset and the images generated by Stable-Diffusion with certain guidance scale. Thus, we choose q_0 to be the distribution of the latents generated by Stable-Diffusion with corresponding guidance scale, using 200-step DPM-Solver++ [34]. We generate these latents with random captions and Gaussian noise different from those we use to compute MSE. The total time for EMS computing is $\sim$11h on 8 GPU cards of NVIDIA A40 for each guidance scale.

Sampling Following previous works [34, 61], we use start time $\epsilon = 10^{-3}$, end time $T = 1$ and adopt the uniform t timestep schedule. For DPM-Solver-v3, we use 2nd-order predictor with pseudo 3rd-order corrector.

D.6 License

Table 2: The used datasets, codes and their licenses.

Name	URL	Citation	License
CIFAR10	https://www.cs.toronto.edu/~kriz/cifar.html	[25]	\
LSUN-Bedroom	https://www.yf.io/p/lsun	[58]	\
ImageNet-256	https://www.image-net.org	[9]	\
MS-COCO2014	https://cocodataset.org	[28]	CC BY 4.0
ScoreSDE	https://github.com/yang-song/score_sde_pytorch	[54]	Apache-2.0
EDM	https://github.com/NVlabs/edm	[22]	CC BY-NC-SA 4.0
Guided-Diffusion	https://github.com/openai/guided-diffusion	[10]	MIT
Latent-Diffusion	https://github.com/CompVis/latent-diffusion	[45]	MIT
Stable-Diffusion	https://github.com/CompVis/stable-diffusion	[45]	CreativeML Open RAIL-M
DPM-Solver++	https://github.com/LuChengTHU/dpm-solver	[34]	MIT
UniPC	https://github.com/wl-zhao/UniPC	[61]	\

We list the used datasets, codes and their licenses in Table 2.

E Runtime Comparison

As we have mentioned in Section 4, the runtime of DPM-Solver-v3 is almost the same as other solvers (DDIM [51], DPM-Solver [33], DPM-Solver++ [34], UniPC [61], etc.) as long as they use the same NFE. This is because the main computation costs are the serial evaluations of the large neural network $\boldsymbol{\epsilon}_\theta$, and the other coefficients are either analytically computed [51, 33, 34, 61], or precomputed (DPM-Solver-v3), thus having neglectable costs.

Table 3 shows the runtime of DPM-Solver-v3 and some other solvers on a single NVIDIA A40 under different settings. We use torch.cuda.Event and torch.cuda.synchronize to accurately compute the runtime. We evaluate the runtime on 8 batches (dropping the first batch since it contains

Table 3: Runtime of different methods to generate a single batch (second / batch, ±std) on a single NVIDIA A40, varying the number of function evaluations (NFE). We don't include the runtime of the decoding stage for latent-space DPMs.

Method	NFE			
	5	10	15	20
CIFAR10 [25], ScoreSDE [54] (batch size = 128)				
DPM-Solver++ [34]	1.253(±0.0014)	2.503(±0.0017)	3.754(±0.0042)	5.010(±0.0048)
UniPC [61]	1.268(±0.0012)	2.532(±0.0018)	3.803(±0.0037)	5.080(±0.0049)
DPM-Solver-v3	1.273(±0.0005)	2.540(±0.0023)	3.826(±0.0039)	5.108(±0.0055)
CIFAR10 [25], EDM [22] (batch size = 128)				
DPM-Solver++ [34]	1.137(±0.0011)	2.278(±0.0015)	3.426(±0.0024)	4.569(±0.0031)
UniPC [61]	1.142(±0.0016)	2.289(±0.0019)	3.441(±0.0035)	4.590(±0.0021)
DPM-Solver-v3	1.146(±0.0010)	2.293(±0.0015)	3.448(±0.0018)	4.600(±0.0027)
LSUN-Bedroom [58], Latent-Diffusion [45] (batch size = 32)				
DPM-Solver++ [34]	1.302(±0.0009)	2.608(±0.0010)	3.921(±0.0023)	5.236(±0.0045)
UniPC [61]	1.305(±0.0005)	2.616(±0.0019)	3.934(±0.0033)	5.244(±0.0043)
DPM-Solver-v3	1.302(±0.0010)	2.620(±0.0027)	3.932(±0.0028)	5.290(±0.0030)
ImageNet256 [9], Guided-Diffusion [10] (batch size = 4)				
DPM-Solver++ [34]	1.594(±0.0011)	3.194(±0.0018)	4.792(±0.0031)	6.391(±0.0045)
UniPC [61]	1.606(±0.0026)	3.205(±0.0025)	4.814(±0.0049)	6.427(±0.0060)
DPM-Solver-v3	1.601(±0.0059)	3.229(±0.0031)	4.807(±0.0068)	6.458(±0.0257)
MS-COCO2014 [28], Stable-Diffusion [45] (batch size = 4)				
DPM-Solver++ [34]	1.732(±0.0012)	3.464(±0.0020)	5.229(±0.0027)	6.974(±0.0013)
UniPC [61]	1.735(±0.0012)	3.484(±0.0364)	5.212(±0.0015)	6.988(±0.0035)
DPM-Solver-v3	1.731(±0.0008)	3.471(±0.0011)	5.211(±0.0030)	6.945(±0.0022)

extra initializations) and report the mean and std. We can see that the runtime is proportional to NFE and has a difference of about ±1% for different solvers, which confirms our statement. Therefore, the speedup for the NFE is almost the actual speedup of the runtime.

F Quantitative Results

Table 4: Quantitative results on CIFAR10 [25]. We report the FID↓ of the methods with different numbers of function evaluations (NFE), evaluated on 50k samples. †We borrow the results reported in their original paper directly.

Method	Model	NFE							
		5	6	8	10	12	15	20	25
†DEIS [59]	ScoreSDE [54]	15.37	\	\	4.17	\	3.37	2.86	\
DPM-Solver++ [34]		28.53	13.48	5.34	4.01	4.04	3.32	2.90	2.76
UniPC [61]		23.71	10.41	5.16	3.93	3.88	3.05	2.73	2.65
DPM-Solver-v3		**12.76**	**7.40**	**3.94**	**3.40**	**3.24**	**2.91**	**2.71**	**2.64**
Heun's 2nd [22]	EDM [22]	320.80	103.86	39.66	16.57	7.59	4.76	2.51	2.12
DPM-Solver++ [34]		24.54	11.85	4.36	2.91	2.45	2.17	2.05	2.02
UniPC [61]		23.52	11.10	3.86	2.85	2.38	**2.08**	**2.01**	**2.00**
DPM-Solver-v3		**12.21**	**8.56**	**3.50**	**2.51**	**2.24**	2.10	2.02	**2.00**

We present the detailed quantitative results of Section 4.1 for different datasets in Table 4, Table 5, Table 6 and Table 7 respectively. They clearly verify that DPM-Solver-v3 achieves consistently better or comparable performance under various settings, especially in 5∼10 NFEs.

Table 5: Quantitative results on LSUN-Bedroom [58]. We report the FID↓ of the methods with different numbers of function evaluations (NFE), evaluated on 50k samples. †DPM-Solver++ abnormally collapses at 15 and 20 NFE, so we instead use its singlestep version with order 3 and 2 respectively.

Method	Model	NFE						
		5	6	8	10	12	15	20
†DPM-Solver++ [34]	Latent-Diffusion [45]	18.59	8.50	4.19	3.63	3.43	3.25	4.33
UniPC [61]		12.24	6.19	4.00	3.56	3.34	3.18	3.07
DPM-Solver-v3		**7.54**	**4.79**	**3.53**	**3.16**	**3.06**	**3.05**	**3.05**

Table 6: Quantitative results on ImageNet-256 [9]. We report the FID↓ of the methods with different numbers of function evaluations (NFE), evaluated on 10k samples.

Method	Model	NFE						
		5	6	8	10	12	15	20
DPM-Solver++ [34]	Guided-Diffusion [10] ($s = 2.0$)	16.87	13.09	9.95	9.12	8.72	8.37	8.11
UniPC [61]		15.62	11.91	9.29	8.35	7.95	7.64	7.44
DPM-Solver-v3		**15.10**	**11.39**	**8.96**	**8.27**	**7.94**	**7.62**	**7.39**

G Ablations

In this section, we conduct some ablations to further evaluate and analyze the effectiveness of DPM-Solver-v3.

G.1 Varying the number of timesteps and datapoints for the EMS

First we'd like to investigate how the number of timesteps N and the number of datapoints K for computing the EMS affects the performance. We conduct experiments with the DPM ScoreSDE [54] on CIFAR10 [25], by decreasing N and K from our default choice $N = 1200, K = 4096$.

We list the FID results using the EMS of different N and K in Table 8. We can observe that the number of datapoints K is crucial to the performance, while the number of timesteps N is less significant and affects mainly the performance in 5$\sim$10 NFEs. When NFE>10, we can decrease N to as little as 50, which gives even better FID. Note that the time cost for computing the EMS is proportional to NK, so how to choose appropriate N and K for both efficiency and accuracy is worth studying.

G.2 First-order comparison

As stated in Appendix A, the first-order case of DPM-Solver-v3 (*DPM-Solver-v3-1*) is different from DDIM [51], which is the previous best first-order solver for DPMs. Note that DPM-Solver-v3-1 applies no corrector, since any corrector has an order of at least 2.

In Table 9 and Figure 7, we compare DPM-Solver-v3-1 with DDIM both quantitatively and qualitatively, using the DPM ScoreSDE [54] on CIFAR10 [25]. The results verify our statement that DPM-Solver-v3-1 performs better than DDIM.

G.3 Effects of pseudo-order solvers

We now demonstrate the effectiveness of pseudo-order solvers, including pseudo-order predictor and pseudo-order corrector.

Pseudo-order predictor The pseudo-order predictor is only applied in few cases (at 5 NFE on CIFAR10 [25]) to achieve maximum performance improvement. In such cases, without pseudo-order predictor, the FID results will degenerate from 12.76 to 15.91 for ScoreSDE [54], and from 12.21 to 12.72 for EDM [22]. While they are still better than previous methods, the pseudo-order predictor is proven to further boost the performance at NFEs as small as 5.

Table 7: Quantitative results on MS-COCO2014 [28] prompts. We report the MSE↓ of the methods with different numbers of function evaluations (NFE), evaluated on 10k samples.

Method	Model	NFE						
		5	6	8	10	12	15	20
DPM-Solver++ [34]	Stable-Diffusion [45] ($s = 1.5$)	0.076	0.056	0.028	0.016	0.012	0.016	0.0085
UniPC [61]		0.055	0.039	**0.024**	0.012	0.0065	0.0046	**0.0018**
DPM-Solver-v3		**0.037**	**0.027**	**0.024**	**0.0065**	**0.0048**	**0.0014**	0.0022
DPM-Solver++ [34]	Stable-Diffusion [45] ($s = 7.5$)	0.60	0.65	0.50	0.46	**0.42**	0.39	0.30
UniPC [61]		0.65	0.71	0.56	0.46	0.43	0.35	0.31
DPM-Solver-v3		**0.55**	**0.64**	**0.49**	**0.40**	0.45	**0.34**	**0.29**

Table 8: Ablation of the number of timesteps N and datapoints K for the EMS, experimented with ScoreSDE [54] on CIFAR10 [25]. We report the FID↓ with different numbers of function evaluations (NFE), evaluated on 50k samples.

N	K	NFE						
		5	6	8	10	12	15	20
1200	256	18.84	7.90	4.49	3.74	3.88	3.52	3.12
1200	1024	15.52	7.55	4.17	3.56	3.37	3.03	2.78
120	4096	13.67	7.60	4.09	3.49	3.24	**2.90**	**2.70**
250	4096	13.28	7.56	4.00	3.45	**3.22**	2.92	**2.70**
1200	4096	**12.76**	**7.40**	**3.94**	**3.40**	3.24	2.91	2.71

Pseudo-order corrector We show the comparison between true and pseudo-order corrector in Table 10. We can observe a consistent improvement when altering to pseudo-order corrector. Thus, it suggests that if we use n-th order predictor, we'd better combine it with pseudo $(n+1)$-th order corrector rather than $(n+1)$-th order corrector.

G.4 Effects of half-corrector

We demonstrate the effects of half-corrector in Table 11, using the popular Stable-Diffusion model [45]. We can observe that under the relatively large guidance scale 7.5 which is necessary for producing samples of high-quality, the corrector adopted by UniPC [61] has a negative effect on the convergence to the ground-truth samples, making UniPC even worse than DPM-Solver++ [34]. When we employ the half-corrector technique, the problem is partially alleviated. Still, it lags behind our DPM-Solver-v3, since we further incorporate the EMS.

H Additional Samples

We provide more visual samples in Figure 8, Figure 9, Figure 10 and Table 12 to demonstrate the qualitative effectiveness of DPM-Solver-v3. It can be seen that the visual quality of DPM-Solver-v3 outperforms previous state-of-the-art solvers. Our method can generate images which have reduced bias (less "shallow"), higher saturation level and more visual details, as mentioned in Section 4.3.

Table 9: Quantitative comparison of first-order solvers (DPM-Solver-v3-1 and DDIM [51]), experimented with ScoreSDE [54] on CIFAR10 [25]. We report the FID↓ with different numbers of function evaluations (NFE), evaluated on 50k samples.

Method	NFE						
	5	6	8	10	12	15	20
DDIM [51]	54.56	41.92	27.51	20.11	15.64	12.05	9.00
DPM-Solver-v3-1	**39.18**	**29.82**	**20.03**	**14.98**	**11.86**	**9.34**	**7.19**

Figure 7: Random samples by first-order solvers (DPM-Solver-v3-1 and DDIM [51]) of ScoreSDE [54] on CIFAR10 dataset [25], using 5, 10 and 20 NFE.

Table 10: Effects of pseudo-order corrector under different settings. We report the FID↓ with different numbers of function evaluations (NFE).

Method	NFE						
	5	6	8	10	12	15	20
LSUN-Bedroom [58], Latent-Diffusion [45]							
4th-order corrector	8.83	5.28	3.65	3.27	3.17	3.14	3.13
→pseudo (default)	**7.54**	**4.79**	**3.53**	**3.16**	**3.06**	**3.05**	**3.05**
ImageNet-256 [9], Guided-Diffusion [10] ($s = 2.0$)							
3rd-order corrector	15.87	11.91	9.27	8.37	7.97	**7.62**	7.47
→pseudo (default)	**15.10**	**11.39**	**8.96**	**8.27**	**7.94**	**7.62**	**7.39**
MS-COCO2014 [28], Stable-Diffusion [45] ($s = 1.5$)							
3rd-order corrector	**0.037**	0.028	0.028	0.014	0.0078	0.0024	**0.0011**
→pseudo (default)	**0.037**	**0.027**	**0.024**	**0.0065**	**0.0048**	**0.0014**	0.0022

Table 11: Ablation of half-corrector/full-corrector on MS-COCO2014 [28] prompts with Stable-Diffusion model [45] and guidance scale 7.5. We report the MSE↓ of the methods with different numbers of function evaluations (NFE), evaluated on 10k samples.

Method	Corrector Usage	NFE						
		5	6	8	10	12	15	20
DPM-Solver++ [34]	no corrector	0.60	0.65	0.50	0.46	0.42	0.39	0.30
UniPC [61]	full-corrector	0.65	0.71	0.56	0.46	0.43	0.35	0.31
	→half-corrector	0.59	0.66	0.50	0.46	**0.41**	0.38	0.30
DPM-Solver-v3	full-corrector	0.65	0.67	**0.49**	**0.40**	0.47	**0.34**	0.30
	→half-corrector	**0.55**	**0.64**	0.51	0.44	0.45	0.36	**0.29**

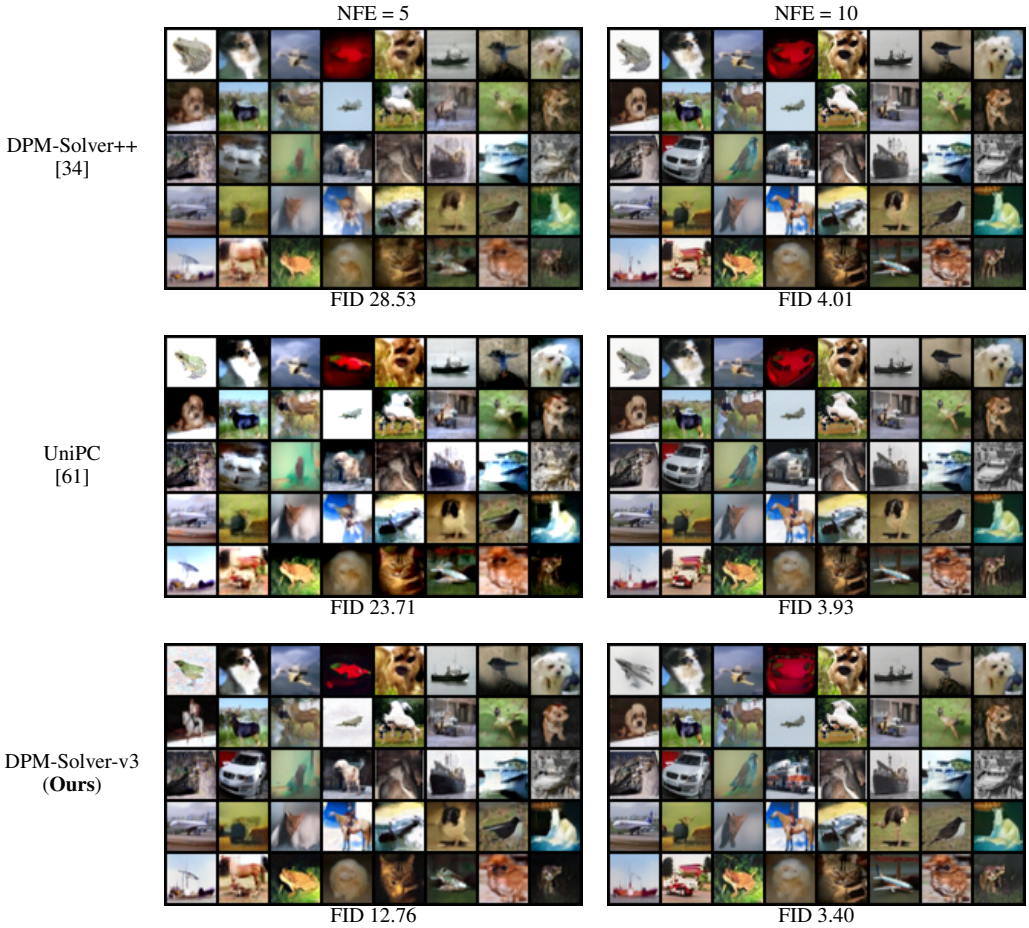


Figure 8: Random samples of ScoreSDE [54] on CIFAR10 dataset [25] with only 5 and 10 NFE.

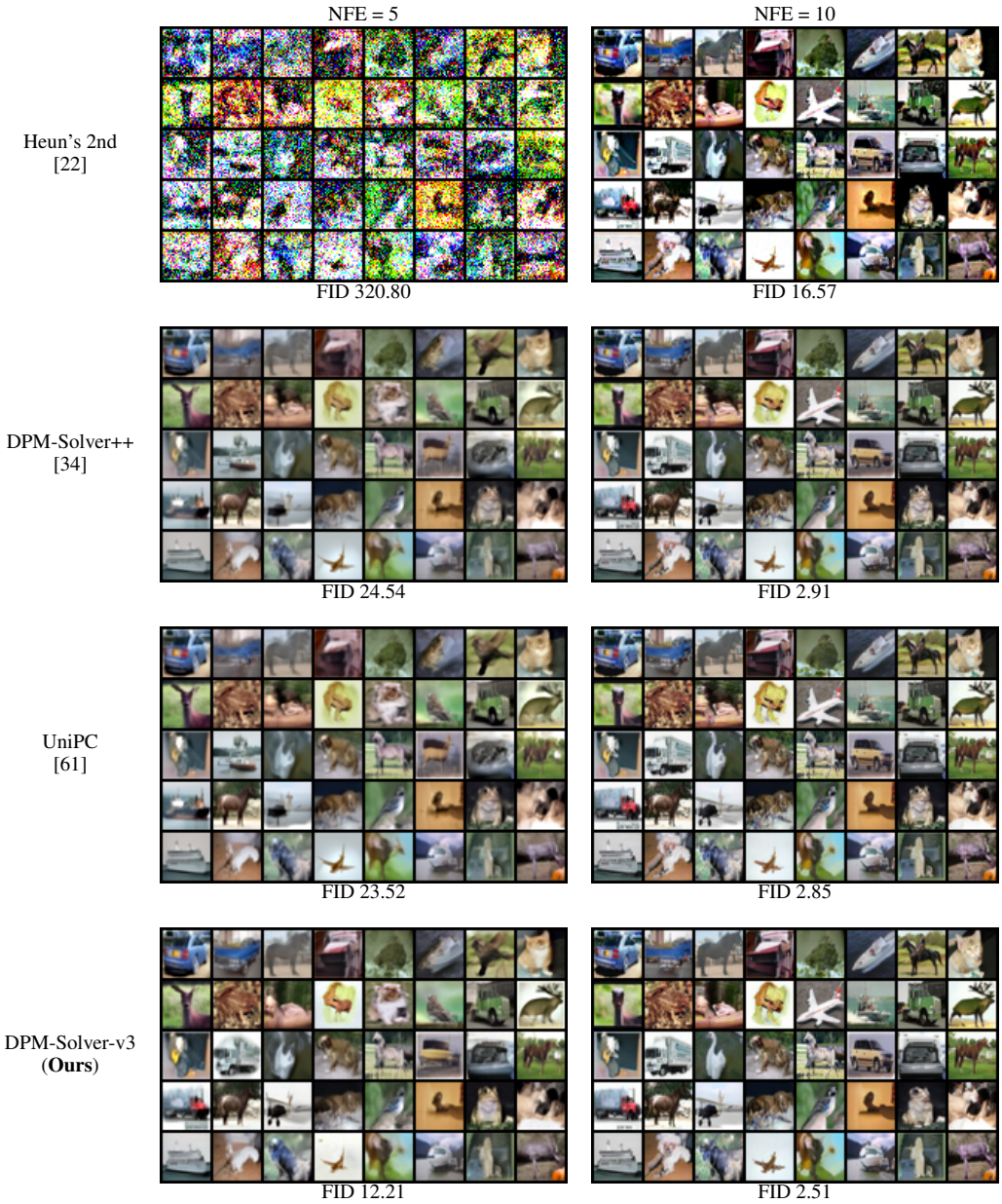


Figure 9: Random samples of EDM [22] on CIFAR10 dataset [25] with only 5 and 10 NFE.

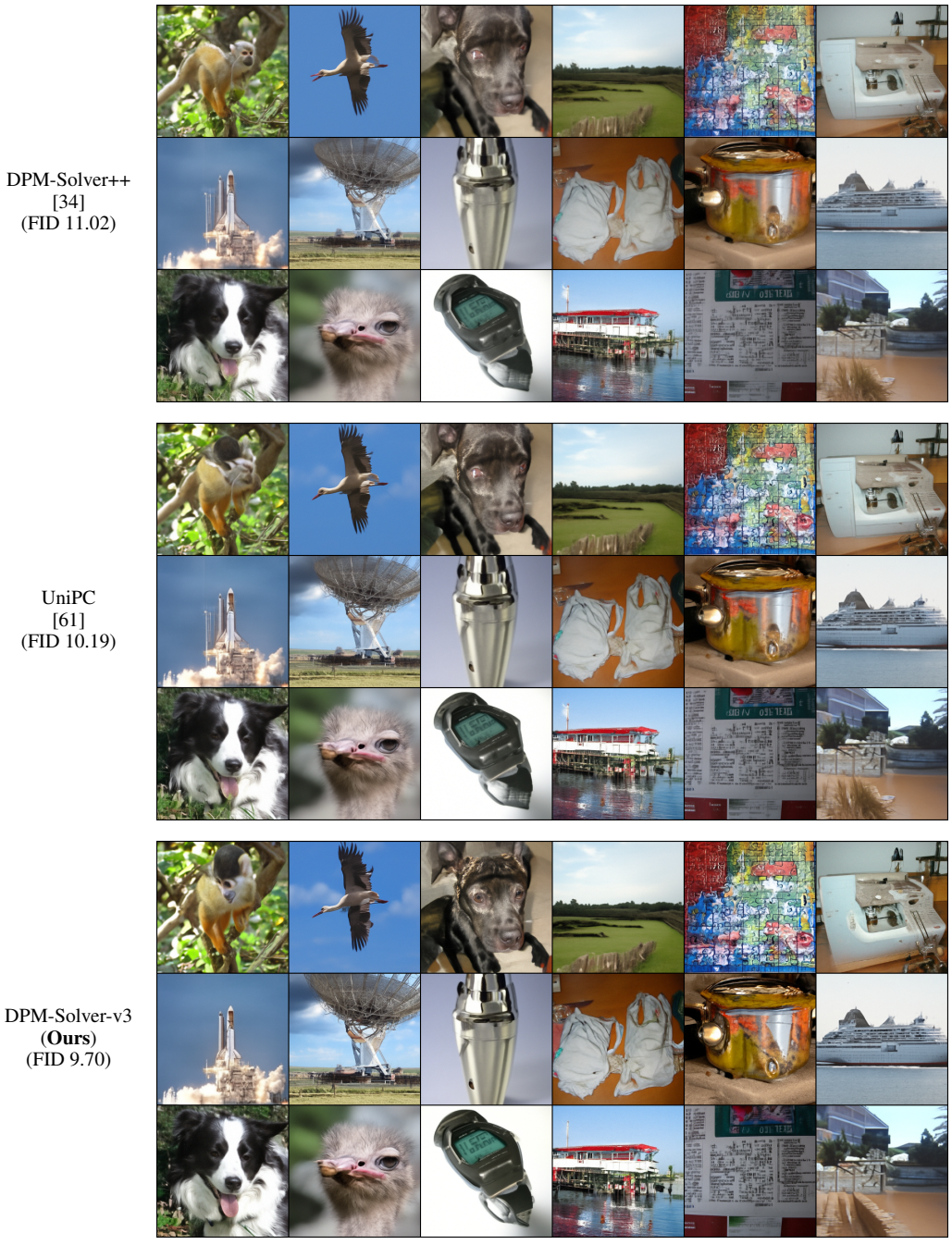
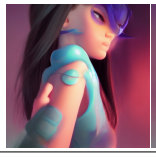
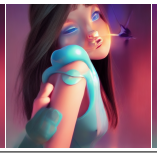
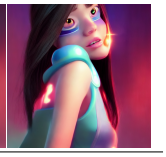
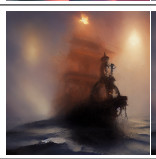
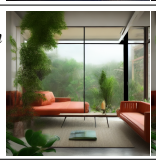
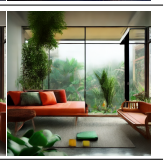
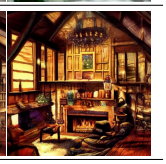
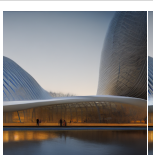
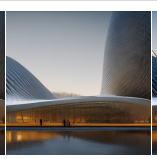
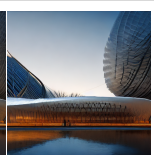
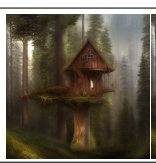
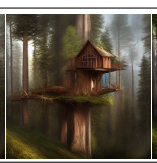
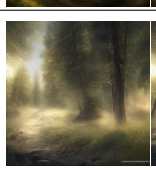
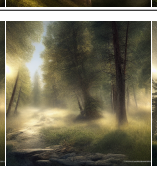
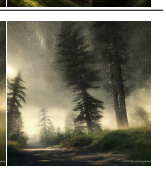
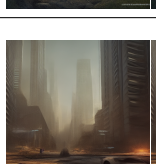
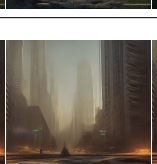
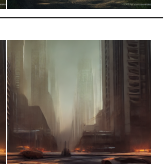
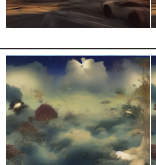
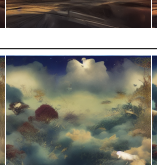
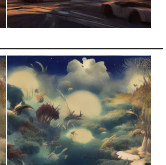


Figure 10: Random samples of Guided-Diffusion [10] on ImageNet-256 dataset [9] with a classifier guidance scale 2.0, using only 7 NFE. We manually remove the potentially disturbing images such as those containing snakes or insects.

Table 12: Additional samples of Stable-Diffusion [45] with a classifier-free guidance scale 7.5, using only 5 NFE and selected text prompts. Some displayed prompts are truncated.

Text Prompts	DPM-Solver++ [34] (MSE 0.60)	UniPC [61] (MSE 0.65)	DPM-Solver-v3 (**Ours**) (MSE 0.55)
"pixar movie still portrait photo of madison beer, jessica alba, woman, as hero catgirl cyborg woman by pixar, by greg rutkowski, wlop, rossdraws, artgerm, weta, marvel, rave girl, leeloo, unreal engine, glossy skin, pearlescent, wet, bright morning, anime, sci-fi, maxim magazine cover"			
"oil painting with heavy impasto of a pirate ship and its captain, cosmic horror painting, elegant intricate artstation concept art by craig mullins detailed"			
"environment living room interior, mid century modern, indoor garden with fountain, retro, m vintage, designer furniture made of wood and plastic, concrete table, wood walls, indoor potted tree, large window, outdoor forest landscape, beautiful sunset, cinematic, concept art, sunstainable architecture, octane render, utopia, ethereal, cinematic light"			
"the living room of a cozy wooden house with a fireplace, at night, interior design, concept art, wallpaper, warm, digital art. art by james gurney and larry elmore."			
"Full page concept design how to craft life Poison, intricate details, infographic of alchemical, diagram of how to make potions, captions, directions, ingredients, drawing, magic, wuxia"			
"Fantasy art, octane render, 16k, 8k, cinema 4d, back-lit, caustics, clean environment, Wood pavilion architecture, warm led lighting, dusk, Landscape, snow, arctic, with aqua water, silver Guggenheim museum spire, with rays of sunshine, white fabric landscape, tall building, zaha hadid and Santiago calatrava, smooth landscape, cracked ice, igloo, warm lighting, aurora borialis, 3d cgi, high definition, natural lighting, realistic, hyper realism"			
"tree house in the forest, atmospheric, hyper realistic, epic composition, cinematic, landscape vista photography by Carr Clifton & Galen Rowell, 16K resolution, Landscape veduta photo by Dustin Lefevre & tdraw, detailed landscape painting by Ivan Shishkin, DeviantArt, Flickr, rendered in Enscape, Miyazaki, Nausicaa Ghibli, Breath of The Wild, 4k detailed post processing, artstation, unreal engine"			
"A trail through the unknown, atmospheric, hyper realistic, 8k, epic composition, cinematic, octane render, artstation landscape vista photography by Carr Clifton & Galen Rowell, 16K resolution, Landscape veduta photo by Dustin Lefevre & tdraw, 8k resolution, detailed landscape painting by Ivan Shishkin, DeviantArt, Flickr, rendered in Enscape, Miyazaki, Nausicaa Ghibli, Breath of The Wild, 4k detailed post processing, artstation, rendering by octane, unreal engine"			
"postapocalyptic city turned to fractal glass, ctane render, 8 k, exploration, cinematic, trending on artstation, by beeple, realistic, 3 5 mm camera, unreal engine, hyper detailed, photo–realistic maximum detai, volumetric light, moody cinematic epic concept art, realistic matte painting, hyper photorealistic, concept art, cinematic epic, octane render, 8k, corona render, movie concept art, octane render, 8 k, corona render, trending on artstation, cinematic composition, ultra–detailed, hyper–realistic, volumetric lighting"			
""WORLDS": zoological fantasy ecosystem infographics, magazine layout with typography, annotations, in the style of Elena Masci, Studio Ghibli, Caspar David Friedrich, Daniel Merriam, Doug Chiang, Ivan Aivazovsky, Herbert Bauer, Edward Tufte, David McCandless"			

References

[1] Kendall Atkinson, Weimin Han, and David E Stewart. *Numerical solution of ordinary differential equations*, volume 108. John Wiley & Sons, 2011.

[2] Fan Bao, Chongxuan Li, Jiacheng Sun, Jun Zhu, and Bo Zhang. Estimating the optimal covariance with imperfect mean in diffusion probabilistic models. In *International Conference on Machine Learning*, pages 1555–1584. PMLR, 2022.

[3] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-DPM: An analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *International Conference on Learning Representations*, 2022.

[4] Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 57(11-12):1214–1229, 2007.

[5] Mari Paz Calvo and César Palencia. A class of explicit multistep exponential integrators for semilinear problems. *Numerische Mathematik*, 102:367–381, 2006.

[6] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2021.

[7] Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.

[8] Guillaume Couairon, Jakob Verbeek, Holger Schwenk, and Matthieu Cord. Diffedit: Diffusion-based semantic image editing with mask guidance. *arXiv preprint arXiv:2210.11427*, 2022.

[9] 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*, pages 248–255. IEEE, 2009.

[10] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794, 2021.

[11] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Genie: Higher-order denoising diffusion solvers. *arXiv preprint arXiv:2210.05475*, 2022.

[12] Alfredo Eisinberg and Giuseppe Fedele. On the inversion of the vandermonde matrix. *Applied mathematics and computation*, 174(2):1384–1397, 2006.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680, 2014.

[14] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[16] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

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

[18] Marlis Hochbruck and Alexander Ostermann. Explicit exponential Runge-Kutta methods for semilinear parabolic problems. *SIAM Journal on Numerical Analysis*, 43(3):1069–1090, 2005.

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

[20] Marlis Hochbruck, Alexander Ostermann, and Julia Schweitzer. Exponential rosenbrock-type methods. *SIAM Journal on Numerical Analysis*, 47(1):786–803, 2009.

[21] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

[22] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, 2022.

[23] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. In *Advances in Neural Information Processing Systems*, 2022.

[24] Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *Advances in Neural Information Processing Systems*, 2021.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Max WY Lam, Jun Wang, Rongjie Huang, Dan Su, and Dong Yu. Bilateral denoising diffusion models. *arXiv preprint arXiv:2108.11514*, 2021.

[27] Shengmeng Li, Luping Liu, Zenghao Chai, Runnan Li, and Xu Tan. Era-solver: Error-robust adams solver for fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2301.12935*, 2023.

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

[29] Jinglin Liu, Chengxi Li, Yi Ren, Feiyang Chen, and Zhou Zhao. Diffsinger: Singing voice synthesis via shallow diffusion mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11020–11028, 2022.

[30] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022.

[31] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[32] Cheng Lu, Kaiwen Zheng, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Maximum likelihood training for score-based diffusion odes by high order denoising score matching. In *International Conference on Machine Learning*, pages 14429–14460. PMLR, 2022.

[33] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, 2022.

[34] 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*, 2022.

[35] Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *arXiv preprint arXiv:2101.02388*, 2021.

[36] Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. *arXiv preprint arXiv:2210.03142*, 2022.

[37] Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022.

[38] Eric Harold Neville. *Iterative interpolation*. St. Joseph's IS Press, 1934.

[39] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[40] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *International Conference on Machine Learning*, pages 16784–16804. PMLR, 2022.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[42] Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Sadekova, Mikhail Kudinov, and Jiansheng Wei. Diffusion-based voice conversion with fast maximum likelihood sampling scheme. In *International Conference on Learning Representations*, 2022.

[43] 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*, pages 8748–8763. PMLR, 2021.

[44] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.

[45] 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*, pages 10684–10695, 2022.

[46] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo-Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems*.

[47] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022.

[48] Robin San-Roman, Eliya Nachmani, and Lior Wolf. Noise estimation for generative diffusion models. *arXiv preprint arXiv:2104.02600*, 2021.

[49] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

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

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

[52] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

[53] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. In *Advances in Neural Information Processing Systems*, volume 34, pages 1415–1428, 2021.

[54] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[55] Hideyuki Tachibana, Mocho Go, Muneyoshi Inahara, Yotaro Katayama, and Yotaro Watanabe. Itô-taylor sampling scheme for denoising diffusion probabilistic models using ideal derivatives. *arXiv e-prints*, pages arXiv–2112, 2021.

[56] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality. In *International Conference on Learning Representations*, 2022.

[57] Suttisak Wizadwongsa and Supasorn Suwajanakorn. Accelerating guided diffusion sampling with splitting numerical methods. In *The Eleventh International Conference on Learning Representations*.

[58] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[59] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.

[60] Min Zhao, Fan Bao, Chongxuan Li, and Jun Zhu. Egsde: Unpaired image-to-image translation via energy-guided stochastic differential equations. In *Advances in Neural Information Processing Systems*, 2022.

[61] Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. UniPC: A unified predictor-corrector framework for fast sampling of diffusion models. *arXiv preprint arXiv:2302.04867*, 2023.

[62] Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. Improved techniques for maximum likelihood estimation for diffusion odes. *arXiv preprint arXiv:2305.03935*, 2023.