Table 2: Formulations that are invariant to the choice of the noise schedules. The maximum likelihood training loss w.r.t. λ is equivalent to the objectives in [10, 41], and the exact solution of the diffusion ODEs are proposed in Proposition 3.1.

Method	Invariance Formulation
Maximum likelihood training	$\int_{\lambda_T}^{\lambda_0} \mathbb{E}_{q_0(\boldsymbol{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\|\boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)-\boldsymbol{\epsilon}\|_2^2\right]\mathrm{d}\lambda$
Sampling by diffusion ODEs	$\hat{\boldsymbol{x}}_{\lambda_t} = \frac{\hat{\alpha}_{\lambda_t}}{\hat{\alpha}_{\lambda_s}}\hat{\boldsymbol{x}}_{\lambda_s} - \hat{\alpha}_{\lambda_t}\int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)\mathrm{d}\lambda$

A Sampling with Invariance to the Noise Schedule

In this section, we discuss more about the exact solution in Proposition 3.1 and give some insights about the formulation. Below we firstly restate the proposition w.r.t. λ (i.e. the half-logSNR).

Proposition 3.1 (Exact solution of diffusion ODEs)**.** *Given an initial value $\hat{\boldsymbol{x}}_{\lambda_s}$ at time s with the corresponding half-logSNR λ_s, the solution $\hat{\boldsymbol{x}}_{\lambda_t}$ at time t of diffusion ODEs in Eq.* (2.7) *with the corresponding half-logSNR λ_t is:*

$$\hat{\boldsymbol{x}}_{\lambda_t} = \frac{\alpha_t}{\alpha_s}\hat{\boldsymbol{x}}_{\lambda_s} - \alpha_t\int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)\mathrm{d}\lambda. \tag{A.1}$$

In the following subsections, we will show that such formulation decouples the model $\boldsymbol{\epsilon}_\theta$ from the specific noise schedule, and thus is invariant to the noise schedule. Moreover, such *change-of-variable* for λ in Proposition 3.1 is highly related to the maximum likelihood training of diffusion models [10, 41]. We show that both the maximum likelihood training and the sampling of diffusion models have invariance formulations that are independent of the noise schedule.

A.1 Decoupling the Sampling Solution from the Noise Schedule

In this section, we show that Proposition 3.1 can decouples the exact solutions of the diffusion ODEs from the specific noise schedules (i.e. choice of the functions $\alpha_t = \alpha(t)$ and $\sigma_t = \sigma(t)$). Namely, given a starting point λ_s, a ending point λ_t, an initial value $\hat{\boldsymbol{x}}_{\lambda_s}$ at λ_s and a noise prediction model $\hat{\boldsymbol{\epsilon}}_\theta$, **the solution of $\hat{\boldsymbol{x}}_{\lambda_t}$ is invariant of the noise schedule between λ_s and λ_t.**

We firstly consider the VP type diffusion models, which is equivalent to the original DDPM [2, 3]. For VP type diffusion models, we always have $\alpha_t^2 + \sigma_t^2 = 1$, so defining the noise schedule is equivalent to defining the function $\alpha_t = \alpha(t)$ (For example, DDPM [2] uses a noise schedule such that $\beta(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}$ is a linear function of t, and i-DDPM [16] uses a noise schedule such that $\beta(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}$ is a cosine function of t). As $\lambda_t = \log\alpha_t - \log\sigma_t$, we have $\alpha_t = \sqrt{\frac{1}{1+e^{-2\lambda_t}}}$ and $\sigma_t = \sqrt{\frac{1}{1+e^{2\lambda_t}}}$. Thus, we can directly compute the α_t and σ_t for a given λ_t. Denote $\hat{\alpha}_\lambda := \sqrt{\frac{1}{1+e^{-2\lambda}}}$, we have

$$\hat{\boldsymbol{x}}_{\lambda_t} = \frac{\hat{\alpha}_{\lambda_t}}{\hat{\alpha}_{\lambda_s}}\hat{\boldsymbol{x}}_{\lambda_s} - \hat{\alpha}_{\lambda_t}\int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)\mathrm{d}\lambda. \tag{A.2}$$

We should notice that the integrand $e^{-\lambda}\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)$ is a function of λ, so its integral from λ_s to λ_t is only dependent on the starting point λ_s, the ending point λ_t and the function $\hat{\boldsymbol{\epsilon}}_\theta$, which is independent of the intermediate values. As other coefficients ($\hat{\alpha}_{\lambda_s}$ and $\hat{\alpha}_{\lambda_t}$) are also only dependent on the starting point λ_s and the ending point λ_t, we can conclude that $\hat{\boldsymbol{x}}_{\lambda_t}$ is invariant of the specific choice of the noise schedules. Intuitively, this is because we converts the original integral of time t in Eq. (3.1) to the integral of λ, and the functions $f(t)$ and $g(t)$ are converted to an **analytical** formulation $e^{-\lambda}$, which is invariant to the specific choices of $f(t)$ and $g(t)$. Finally, for other types of diffusion models (such as the VE type and the subVP type), they are all equivalent to the VP type by equivalently rescaling the noise prediction models, as proved in [10]. Therefore, the solutions of these types also have such property.

In summary, Proposition 3.1 decouples the solution of diffusion ODEs from the noise schedules, which gives us an opportunity to design tailor-made samplers for DPMs. In fact, as shown in Sec. 3.2, the only approximation of the proposed DPM-Solver is about the Taylor expansion of the neural network $\hat{\epsilon}_\theta$ w.r.t. λ, and DPM-Solver **analytically** computes other coefficients (which are corresponding to the specific noise schedules). Intuitively, DPM-Solver keeps the known information as much as possible, and only approximates the intractable integral of the neural network, so it can generate comparable samples within much fewer steps.

A.2 Choosing Time Steps for λ is Invariant to the Noise Schedule

As mentioned in Appendix A.1, the formulation of Proposition 3.1 decouples the sampling solution from the noise schedule. The solution depends on the starting point λ_s and the ending point λ_t, and is invariant to the intermediate noise schedule. Similarly, the updating equations of the algorithm of DPM-Solver are also invariant to the intermediate noise schedule. Therefore, if we have chosen the time steps $\{\lambda_i\}_{i=0}^M$, then the solution of DPM-Solver is also determined and is invariant to the intermediate noise schedule.

A simple way for choosing time steps for λ is uniformly splitting $[\lambda_T, \lambda_\epsilon]$, which is the setting in our experiments. However, we believe that there exists more precise ways for choosing the time steps, and we leave it for future work.

A.3 Relationship with the Maximum Likelihood Training of Diffusion Models

Interestingly, the maximum likelihood training of diffusion SDEs in continuous time also has such invariance property [10]. Below we briefly review the maximum likelihood training loss of diffusion SDEs, and then propose a new insight for understanding diffusion models.

Denote the data distribution as $q_0(\boldsymbol{x}_0)$, the distribution of the forward process at each time t as $q_t(\boldsymbol{x}_t)$, the distribution of the reverse process at each time t as $p_t(\boldsymbol{x}_t)$ with $p_T = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. In [3], it is proved that the KL-divergence between q_0 and p_0 can be bounded by a weighted score matching loss:

$$D_{\mathrm{KL}}(q_0 \parallel p_0) \le D_{\mathrm{KL}}(q_T \parallel p_T) + \frac{1}{2}\int_0^T \frac{g^2(t)}{\sigma_t^2}\mathbb{E}_{q_0(\boldsymbol{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\Big[\|\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) - \boldsymbol{\epsilon}\|_2^2\Big]\mathrm{d}t + C, \quad \text{(A.3)}$$

where $\boldsymbol{x}_t = \alpha_t \boldsymbol{x}_0 + \sigma_t \boldsymbol{\epsilon}$ and C is a constant independent of θ. As shown in Sec. 3.1, we have

$$g^2(t) = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\sigma_t^2 = 2\sigma_t^2\left(\frac{\mathrm{d}\log\sigma_t}{\mathrm{d}t} - \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\right) = -2\sigma_t^2\frac{\mathrm{d}\lambda_t}{\mathrm{d}t}, \quad \text{(A.4)}$$

so by applying *change-of-variable* w.r.t. λ, we have

$$D_{\mathrm{KL}}(q_0 \parallel p_0) \le D_{\mathrm{KL}}(q_T \parallel p_T) + \int_{\lambda_T}^{\lambda_0} \mathbb{E}_{q_0(\boldsymbol{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\Big[\|\boldsymbol{\epsilon}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) - \boldsymbol{\epsilon}\|_2^2\Big]\mathrm{d}\lambda + C, \quad \text{(A.5)}$$

which is equivalent to the importance sampling trick in [41, Sec. 5.1] and the continuous-time diffusion loss in [10, Eq. (22)]. Compared to Proposition 3.1, we can find that the sampling and the maximum likelihood training of diffusion models can both be converted to an integral w.r.t. λ, such that the formulation is invariant to the specific noise schedules, and we summarize it in Table 2. Such invariance property for both training and sampling brings a new insight for understanding diffusion models. For instance, we can directly define the noise prediction model $\boldsymbol{\epsilon}_\theta$ w.r.t. the (half-)logSNR λ instead of the time t, then the training and sampling for diffusion models can be done **without further choosing any ad-hoc noise schedules**. Such finding may unify the different ways of the training and the inference of diffusion models, and we leave it for future study.

B Proof of Theorem 3.2

B.1 Assumptions

Throughout this section, we denote $\boldsymbol{x}_s$ as the solution of the diffusion ODE Eq. (2.7) starting from $\boldsymbol{x}_T$. For DPM-Solver-k we make the following assumptions:

Assumption B.1. The total derivatives $\frac{\mathrm{d}^j \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)}{\mathrm{d}\lambda^j}$ (as a function of λ) exist and are continuous for $0 \leq j \leq k+1$.

Assumption B.2. The function $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, s)$ is Lipschitz w.r.t. to its first parameter $\boldsymbol{x}$.

Assumption B.3. $h_{max} = \mathcal{O}(1/M)$.

We note that the first assumption is required by Taylor's theorem Eq. (3.6), and the second assumption is used to replace $\epsilon_\theta(\tilde{\boldsymbol{x}}_s, s)$ with $\epsilon_\theta(\boldsymbol{x}_s, s) + \mathcal{O}(\boldsymbol{x}_s - \tilde{\boldsymbol{x}}_s)$ so that the Taylor expansion w.r.t. λ_s is applicable. The last one is a technical assumption to exclude a significantly large step-size.

B.2 General Expansion of the Exponentially Weighted Integral

Firstly, we derive the Taylor expansion of the exponentially weighted integral. Let $t < s$ and then $\lambda_t > \lambda_s$. Denote $h \coloneqq \lambda_t - \lambda_s$, and the k-th order total derivative $\hat{\boldsymbol{\epsilon}}_\theta^{(k)}(\hat{\boldsymbol{x}}_\lambda, \lambda) \coloneqq \frac{\mathrm{d}^k \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda,\lambda)}{\mathrm{d}\lambda^k}$. For $n \geq 0$, the n-th order Taylor expansion of $\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda)$ w.r.t. λ is

$$\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) = \sum_{k=0}^{n} \frac{(\lambda - \lambda_s)^k}{k!} \hat{\boldsymbol{\epsilon}}_\theta^{(k)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^{n+1}). \tag{B.1}$$

To expand the exponential integrator, we further define [31]:

$$\varphi_k(z) \coloneqq \int_0^1 e^{(1-\delta)z} \frac{\delta^{k-1}}{(k-1)!} \mathrm{d}\delta, \qquad \varphi_0(z) = e^z \tag{B.2}$$

and it satisfies $\varphi_k(0) = \frac{1}{k!}$ and a recurrence relation $\varphi_{k+1}(z) = \frac{\varphi_k(z) - \varphi_k(0)}{z}$. By taking the Taylor expansion of $\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda)$, the exponential integrator can be rewritten as

$$\int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) \mathrm{d}\lambda = \frac{\sigma_t}{\alpha_t} \sum_{k=0}^{n} h^{k+1} \varphi_{k+1}(h) \hat{\boldsymbol{\epsilon}}_\theta^{(k)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^{n+2}). \tag{B.3}$$

So the solution of $\boldsymbol{x}_t$ in Eq. (3.4) can be expanded as

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s} \boldsymbol{x}_s - \sigma_t \sum_{k=0}^{n} h^{k+1} \varphi_{k+1}(h) \hat{\boldsymbol{\epsilon}}_\theta^{(k)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^{n+2}). \tag{B.4}$$

Finally, we list the closed-forms of φ_k for $k = 1, 2, 3$:

$$\varphi_1(h) = \frac{e^h - 1}{h}, \tag{B.5}$$

$$\varphi_2(h) = \frac{e^h - h - 1}{h^2}, \tag{B.6}$$

$$\varphi_3(h) = \frac{e^h - h^2/2 - h - 1}{h^3}. \tag{B.7}$$

B.3 Proof of Theorem 3.2 when $k = 1$

Proof. Taking $n = 0, t = t_i, s = t_{i-1}$ in Eq. (B.4), we obtain

$$\boldsymbol{x}_{t_i} = \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \boldsymbol{x}_{t_{i-1}} - \sigma_t (e^{h_i} - 1) \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_{i-1}}, t_{i-1}) + \mathcal{O}(h_i^2). \tag{B.8}$$

By Assumption B.2 and Eq. (3.7), it holds that

$$\begin{aligned}
\tilde{\boldsymbol{x}}_{t_i} &= \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{t_i} (e^{h_i} - 1) \boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}) \\
&= \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{t_i} (e^{h_i} - 1) \left(\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_{i-1}}, t_{i-1}) + \mathcal{O}(\tilde{\boldsymbol{x}}_{t_{i-1}} - \boldsymbol{x}_{t_{i-1}}) \right) \\
&= \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \boldsymbol{x}_{t_{i-1}} - \sigma_{t_i} (e^{h_i} - 1) \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t_{i-1}}, t_{i-1}) + \mathcal{O}(\tilde{\boldsymbol{x}}_{t_{i-1}} - \boldsymbol{x}_{t_{i-1}}) \\
&= \boldsymbol{x}_{t_i} + \mathcal{O}(h_{max}^2) + \mathcal{O}(\tilde{\boldsymbol{x}}_{t_{i-1}} - \boldsymbol{x}_{t_{i-1}}).
\end{aligned}$$

Repeat this argument, we find that

$$\tilde{\boldsymbol{x}}_{t_M} = \boldsymbol{x}_{t_0} + \mathcal{O}(M h_{max}^2) = \boldsymbol{x}_{t_0} + \mathcal{O}(h_{max}),$$

and thus completes the proof. □

B.4 Proof of Theorem 3.2 when $k = 2$

We prove the discretization error of the general form of DPM-Solver-2 in Algorithm 4.

Proof. First, we consider the following update for $0 < t < s < T, h := \lambda_t - \lambda_s$.

$$s_1 = t_\lambda\left(\lambda_s + r_1 h\right), \tag{B.9a}$$

$$\bar{\boldsymbol{u}} = \frac{\alpha_{s_1}}{\alpha_s}\boldsymbol{x}_s - \sigma_{s_1}\left(e^{r_1 h} - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s), \tag{B.9b}$$

$$\bar{\boldsymbol{x}}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \frac{\sigma_t}{2r_1}(e^h - 1)(\boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s)). \tag{B.9c}$$

Note that the above update is the same as a single step of DPM-Solver-2 with $s = t_{i-1}$ and $t = t_i$, except that $\tilde{x}_{t_{i-1}}$ is replaced with the exact solution $\boldsymbol{x}_{t_{i-1}}$. Once we have proven that $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t + \mathcal{O}(h^3)$, we can show that $\tilde{\boldsymbol{x}}_{t_i} = \boldsymbol{x}_{t_i} + \mathcal{O}(h_{max}^3) + \mathcal{O}(\tilde{\boldsymbol{x}}_{t_{i-1}} - \boldsymbol{x}_{t_{i-1}})$ by a similar argument as in Appendix B.3, and therefore completes the proof.

In this remaining part we prove that $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t + \mathcal{O}(h^3)$.

Taking $n = 1$ in Eq. (B.4), we obtain

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t h\varphi_1(h)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \sigma_t h^2\varphi_2(h)\hat{\boldsymbol{\epsilon}}_\theta^{(1)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^3). \tag{B.10}$$

From Eq. (B.1), we have

$$\begin{aligned}
\bar{\boldsymbol{x}}_t &= \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \frac{\sigma_t}{2r_1}(e^h - 1)(\boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s)) \\
&= \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \frac{\sigma_t}{2r_1}\left(e^h - 1\right)\left[\boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{s_1}, s_1)\right] \\
&\quad - \frac{\sigma_t}{2r_1}\left(e^h - 1\right)\left[(\lambda_{s_1} - \lambda_s)\hat{\boldsymbol{\epsilon}}_\theta^{(1)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^2)\right].
\end{aligned}$$

Note that by the Lipschitzness of $\boldsymbol{\epsilon}_\theta$ w.r.t. $\boldsymbol{x}$ (Assumption B.2),

$$\|\boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{s_1}, s_1)\| = \mathcal{O}(\|\bar{\boldsymbol{u}} - \boldsymbol{x}_{s_1}\|) = \mathcal{O}(h^2),$$

where the last equation follows from a similar argument in the proof of $k = 1$. Since $e^h - 1 = \mathcal{O}(h)$, the second term of the above display is $\mathcal{O}(h^3)$.

As $\lambda_{s_1} - \lambda_s = r_1 h$, $\varphi_i(h) = (e^h - 1)/h$ and $\varphi_2(h) = (e^h - h - 1)/h^2$, we find

$$\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t = \sigma_t\left[h^2\varphi_2(h) - (e^h - 1)\frac{\lambda_{s_1} - \lambda_s}{2r_1}\right]\hat{\boldsymbol{\epsilon}}_\theta^{(1)}(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^3).$$

Then, the proof is completed by noticing that

$$h^2\varphi_2(h) - (e^h - 1)\frac{\lambda_{s_1} - \lambda_s}{2r_1} = (2e^h - h - 2 - he^h)/2 = \mathcal{O}(h^3).$$

□

B.5 Proof of Theorem 3.2 when $k = 3$

Proof. As in Appendix B.4, it suffices to show that the following update has error $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t + \mathcal{O}(h^4)$ for $0 < t < s < T$ and $h = \lambda_s - \lambda_t$.

$$s_1 = t_\lambda\left(\lambda_s + r_1 h\right), \quad s_2 = t_\lambda\left(\lambda_s + r_2 h\right), \tag{B.11a}$$

$$\bar{\boldsymbol{u}}_1 = \frac{\alpha_{s_1}}{\alpha_s}\boldsymbol{x}_s - \sigma_{s_1}\left(e^{r_1 h} - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s), \tag{B.11b}$$

$$\boldsymbol{D}_1 = \boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}_1, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s), \tag{B.11c}$$

$$\bar{\boldsymbol{u}}_2 = \frac{\alpha_{s_2}}{\alpha_s}\boldsymbol{x}_s - \sigma_{s_2}\left(e^{r_2 h} - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \frac{\sigma_{s_2} r_2}{r_1}\left(\frac{e^{r_2 h} - 1}{r_2 h} - 1\right)\boldsymbol{D}_1, \tag{B.11d}$$

$$\boldsymbol{D}_2 = \boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}_2, s_2) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s), \tag{B.11e}$$

$$\bar{\boldsymbol{x}}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \frac{\sigma_t}{r_2}\left(\frac{e^h - 1}{h} - 1\right)\boldsymbol{D}_2. \tag{B.11f}$$

First, we prove that

$$\bar{\boldsymbol{u}}_2 = \boldsymbol{x}_{s_2} + \mathcal{O}(h^3). \tag{B.12}$$

Similar to the proof in Appendix B.4, since $\frac{e^{r_2 h - 1}}{r_2 h} - 1 = \mathcal{O}(h)$ and $\bar{\boldsymbol{u}}_1 = \boldsymbol{x}_{s_1} + \mathcal{O}(h^2)$, then

$$\begin{aligned}
\bar{\boldsymbol{u}}_2 = & \frac{\alpha_{s_2}}{\alpha_s}\boldsymbol{x}_s - \sigma_{s_2}\left(e^{r_2 h} - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) \\
& - \sigma_{s_2}\frac{r_2}{r_1}\left(\frac{e^{r_2 h} - 1}{r_2 h} - 1\right)\left(\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{s_1}, s_1) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s)\right) + \mathcal{O}(h^3) \\
= & \frac{\alpha_{s_2}}{\alpha_s}\boldsymbol{x}_s - \sigma_{s_2}\left(e^{r_2 h} - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) \\
& - \sigma_{s_2}\frac{r_2}{r_1}\left(\frac{e^{r_2 h} - 1}{r_2 h} - 1\right)\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_s, s)(\lambda_{s_1} - \lambda_s) + \mathcal{O}(h^3).
\end{aligned}$$

Let $h_2 = r_2 h$, then following the same line of arguments in the proof of Appendix B.4, it suffices to check that

$$\begin{aligned}
&\varphi_1(h_2)h_2 = e^{h_2} - 1, \\
&\varphi_2(h_2)h_2^2 = \frac{r_2}{r_1}\left(\frac{e^{h_2} - 1}{h_2} - 1\right)(\lambda_{s_1} - \lambda_s) + \mathcal{O}(h^3),
\end{aligned}$$

which holds by applying Taylor expansion.

Using $\bar{\boldsymbol{u}}_2 = \boldsymbol{x}_{s_2} + \mathcal{O}(h^3)$ and $\lambda_{s_2} - \lambda_s = r_2 h = \frac{2}{3}h$, we find that

$$\begin{aligned}
\bar{\boldsymbol{x}}_t &= \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \sigma_t\frac{1}{r_2}\left(\frac{e^h - 1}{h} - 1\right)\left(\boldsymbol{\epsilon}_\theta(\bar{\boldsymbol{u}}_2, s_2) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s)\right) \\
&= \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \sigma_t\frac{1}{r_2}\left(\frac{e^h - 1}{h} - 1\right)\left(\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{s_2}, s_2) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s)\right) + \mathcal{O}(h^4) \\
&= \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^h - 1\right)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) \\
&\quad - \sigma_t\frac{1}{r_2}\left(\frac{e^h - 1}{h} - 1\right)\left(\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_s, s)r_2 h + \frac{1}{2}\boldsymbol{\epsilon}_\theta^{(2)}(\boldsymbol{x}_s, s)r_2^2 h^2\right) + \mathcal{O}(h^4).
\end{aligned}$$

Comparing with the Taylor expansion in Eq. (B.4) with $n = 2$:

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t h\varphi_1(h)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_s, s) - \sigma_t h^2\varphi_2(h)\boldsymbol{\epsilon}_\theta^{(1)}(\boldsymbol{x}_s, s) - \sigma_t h^3\varphi_3(h)\boldsymbol{\epsilon}_\theta^{(2)}(\boldsymbol{x}_s, s) + \mathcal{O}(h^4),$$

we need to check the following conditions:

$$\begin{aligned}
h\varphi_1(h) &= e^h - 1, \\
h^2\varphi_2(h) &= \left(\frac{e^h - 1}{h} - 1\right)h, \\
h^3\varphi_3(h) &= \left(\frac{e^h - 1}{h} - 1\right)\frac{r_2 h^2}{2} + \mathcal{O}(h^4).
\end{aligned}$$

The first two conditions are clear. The last condition follows from

$$h^3\varphi_3(h) = e^h - 1 - h - \frac{h^2}{2} = \frac{h^3}{6} + \mathcal{O}(h^4) = \left(\frac{e^h - 1}{h} - 1\right)\frac{r_2 h^2}{2}.$$

Therefore, $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t + \mathcal{O}(h^4)$. □

B.6 Connections to Explicit Exponential Runge-Kutta (expRK) Methods

Assume we have an ODE with the following form:

$$\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} = \alpha\boldsymbol{x}_t + \boldsymbol{N}(\boldsymbol{x}_t, t),$$

where $\alpha \in \mathbb{R}$ and $\boldsymbol{N}(\boldsymbol{x}_t, t) \in \mathbb{R}^D$ is a non-linear function of $\boldsymbol{x}_t$. Given an initial value $\boldsymbol{x}_t$ at time t, for $h > 0$, the true solution at time $t + h$ is

$$\boldsymbol{x}_{t+h} = e^{\alpha h}\boldsymbol{x}_t + e^{\alpha h}\int_0^h e^{-\alpha\tau}\boldsymbol{N}(\boldsymbol{x}_{t+\tau}, t+\tau)\mathrm{d}\tau.$$

The exponential Runge-Kutta methods [25, 31] use some intermediate points to approximate the integral $\int e^{-\alpha\tau}\boldsymbol{N}(\boldsymbol{x}_{t+\tau}, t+\tau)\mathrm{d}\tau$. Our proposed DPM-Solver is inspired by the same technique for approximating the same integral with $\alpha = 1$ and $\boldsymbol{N} = \tilde{\boldsymbol{\epsilon}}_\theta$. However, DPM-Solver is different from the expRK methods, because their linear term $e^{\alpha h}\boldsymbol{x}_t$ is different from our linear term $\frac{\alpha_{t+h}}{\alpha_t}\boldsymbol{x}_t$. In summary, DPM-Solver is inspired by the same technique of expRK for deriving high-order approximations of the exponentially weighted integral, but the formulation of DPM-Solver is different from expRK, and DPM-Solver is customized for the specific formulation of diffusion ODEs.

C Algorithms of DPM-Solvers

We firstly list the detailed DPM-Solver-1, 2, 3 in Algorithms 3, 4, 5. Note that DPM-Solver-2 is the general case with $r_1 \in (0, 1)$, and we usually set $r_1 = 0.5$ for DPM-Solver-2, as in Sec. 3.

Algorithm 3 DPM-Solver-1.

```
Require: initial value $\boldsymbol{x}_T$, time steps $\{t_i\}_{i=0}^M$, model $\boldsymbol{\epsilon}_\theta$
 1: def dpm-solver-1($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i$):
 2:     $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
 3:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{t_i}\left(e^{h_i} - 1\right)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})$
 4:     return $\tilde{\boldsymbol{x}}_{t_i}$
 5: $\tilde{\boldsymbol{x}}_{t_0} \leftarrow \boldsymbol{x}_T$
 6: for $i \leftarrow 1$ to $M$ do
 7:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow$ dpm-solver-1($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i$)
 8: end for
 9: return $\tilde{\boldsymbol{x}}_{t_M}$
```

Algorithm 4 DPM-Solver-2 (general version).

```
Require: initial value $\boldsymbol{x}_T$, time steps $\{t_i\}_{i=0}^M$, model $\boldsymbol{\epsilon}_\theta$, $r_1 = 0.5$
 1: def dpm-solver-2($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i, r_1$):
 2:     $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
 3:     $s_i \leftarrow t_\lambda\left(\lambda_{t_{i-1}} + r_1 h_i\right)$
 4:     $\boldsymbol{u}_i \leftarrow \frac{\alpha_{s_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{s_i}\left(e^{r_1 h_i} - 1\right)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})$
 5:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i}-1)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}) - \frac{\sigma_{t_i}}{2r_1}(e^{h_i}-1)(\boldsymbol{\epsilon}_\theta(\boldsymbol{u}_i, s_i) - \boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}))$
 6:     return $\tilde{\boldsymbol{x}}_{t_i}$
 7: $\tilde{\boldsymbol{x}}_{t_0} \leftarrow \boldsymbol{x}_T$
 8: for $i \leftarrow 1$ to $M$ do
 9:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow$ dpm-solver-2($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i, r_1$)
10: end for
11: return $\tilde{\boldsymbol{x}}_{t_M}$
```

Then we list the adaptive step size algorithms, named as *DPM-Solver-12* (combining 1 and 2; Algorithm 6) and *DPM-Solver-23* (combining 2 and 3; Algorithm 7). We follow [20] to let the absolute tolerance $\epsilon_{\text{atol}} = \frac{\boldsymbol{x}_{\text{max}} - \boldsymbol{x}_{\text{min}}}{256}$ for image data, which is 0.0078 for VP type DPMs. We can tune the relative tolerance ϵ_{rtol} to balance the accuracy and NFE, and we find that $\epsilon_{\text{rtol}} = 0.05$ is good enough and can converge quickly.

In practice, the inputs of the adaptive step size solvers are batch data. We simply choose E_2 and E_3 as the maximum value of all the batch data. Besides, we implement the comparison $s > \epsilon$ by $|s - \epsilon| > 10^{-5}$ to avoid numerical issues.

Algorithm 5 DPM-Solver-3.

```
Require: initial value $\boldsymbol{x}_T$, time steps $\{t_i\}_{i=0}^M$, model $\boldsymbol{\epsilon}_\theta$, $r_1 = \frac{1}{3}$, $r_2 = \frac{2}{3}$
 1: def dpm-solver-3($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i, r_1, r_2$):
 2:     $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
 3:     $s_{2i-1} \leftarrow t_\lambda\left(\lambda_{t_{i-1}} + r_1 h_i\right), \quad s_{2i} \leftarrow t_\lambda\left(\lambda_{t_{i-1}} + r_2 h_i\right)$
 4:     $\boldsymbol{u}_{2i-1} \leftarrow \frac{\alpha_{s_{2i-1}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{s_{2i-1}}\left(e^{r_1 h_i} - 1\right)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})$
 5:     $\boldsymbol{D}_{2i-1} \leftarrow \boldsymbol{\epsilon}_\theta(\boldsymbol{u}_{2i-1}, s_{2i-1}) - \boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})$
 6:     $\boldsymbol{u}_{2i} \leftarrow \frac{\alpha_{s_{2i}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{s_{2i}}\left(e^{r_2 h_i} - 1\right)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}) - \frac{\sigma_{s_{2i}} r_2}{r_1}\left(\frac{e^{r_2 h_i} - 1}{r_2 h_i} - 1\right)\boldsymbol{D}_{2i-1}$
 7:     $\boldsymbol{D}_{2i} \leftarrow \boldsymbol{\epsilon}_\theta(\boldsymbol{u}_{2i}, s_{2i}) - \boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})$
 8:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{x}}_{t_{i-1}} - \sigma_{t_i}\left(e^{h_i} - 1\right)\boldsymbol{\epsilon}_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}) - \frac{\sigma_{t_i}}{r_2}\left(\frac{e^{h_i} - 1}{h} - 1\right)\boldsymbol{D}_{2i}$
 9:     return $\tilde{\boldsymbol{x}}_{t_i}$
10: $\tilde{\boldsymbol{x}}_{t_0} \leftarrow \boldsymbol{x}_T$
11: for $i \leftarrow 1$ to $M$ do
12:     $\tilde{\boldsymbol{x}}_{t_i} \leftarrow$ dpm-solver-3($\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}, t_i, r_1, r_2$)
13: end for
14: return $\tilde{\boldsymbol{x}}_{t_M}$
```

Algorithm 6 (**DPM-Solver-12**) Adaptive step size algorithm by combining DPM-Solver-1 and 2.

```
Require: start time $T$, end time $\epsilon$, initial value $\boldsymbol{x}_T$, model $\boldsymbol{\epsilon}_\theta$, data dimension $D$, hyperparameters
    $\epsilon_{\text{rtol}} = 0.05$, $\epsilon_{\text{atol}} = 0.0078$, $h_{\text{init}} = 0.05$, $\theta = 0.9$
Ensure: the approximated solution $\boldsymbol{x}_\epsilon$ at time $\epsilon$
 1: $s \leftarrow T$, $h \leftarrow h_{\text{init}}$, $\boldsymbol{x} \leftarrow \boldsymbol{x}_T$, $\boldsymbol{x}_{\text{prev}} \leftarrow \boldsymbol{x}_T$, $r_1 \leftarrow \frac{1}{2}$, NFE $\leftarrow 0$
 2: while $s > \epsilon$ do
 3:     $t \leftarrow t_\lambda(\lambda_s + h)$
 4:     $\boldsymbol{x}_1 \leftarrow$ dpm-solver-1($\boldsymbol{x}, s, t$)
 5:     $\boldsymbol{x}_2 \leftarrow$ dpm-solver-2($\boldsymbol{x}, s, t, r_1$) (Share the same function value $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, s)$ with dpm-solver-1.)
 6:     $\boldsymbol{\delta} \leftarrow \max(\epsilon_{\text{atol}}, \epsilon_{\text{rtol}} \max(|\boldsymbol{x}_1|, |\boldsymbol{x}_{\text{prev}}|))$
 7:     $E_2 \leftarrow \frac{1}{\sqrt{D}}\|\frac{\boldsymbol{x}_1 - \boldsymbol{x}_2}{\boldsymbol{\delta}}\|_2$
 8:     if $E_2 \leq 1$ then
 9:         $\boldsymbol{x}_{\text{prev}} \leftarrow \boldsymbol{x}_1$, $\boldsymbol{x} \leftarrow \boldsymbol{x}_2$, $s \leftarrow t$
10:     end if
11:     $h \leftarrow \min(\theta h E_2^{-\frac{1}{2}}, \lambda_\epsilon - \lambda_s)$
12:     NFE $\leftarrow$ NFE $+ 2$
13: end while
14: return $\boldsymbol{x}$, NFE
```

D Implementation Details of DPM-Solver

D.1 End Time of Sampling

Theoretically, we need to solve diffusion ODEs from time T to time 0 to generate samples. Practically, the training and evaluation for the noise prediction model $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ usually start from time T to time ϵ to avoid numerical issues for t near to 0, where $\epsilon > 0$ is a hyperparameter [3].

In contrast to the sampling methods based on diffusion SDEs [2, 3], We don't add the "denoising" trick at the final step at time ϵ (which is to set the noise variance to zero), and we just solve diffusion ODEs from T to ϵ by DPM-Solver, since we find it performs well enough.

For discrete-time DPMs, we firstly convert the model to continuous time (see Appendix D.2), and then solver it from time T to time t.

Algorithm 7 (**DPM-Solver-23**) Adaptive step size algorithm by combining DPM-Solver-2 and 3.

```
Require: start time $T$, end time $\epsilon$, initial value $\boldsymbol{x}_T$, model $\boldsymbol{\epsilon}_\theta$, data dimension $D$, hyperparameters
    $\epsilon_{\text{rtol}} = 0.05$, $\epsilon_{\text{atol}} = 0.0078$, $h_{\text{init}} = 0.05$, $\theta = 0.9$
Ensure: the approximated solution $\boldsymbol{x}_\epsilon$ at time $\epsilon$
 1: $s \leftarrow T$, $h \leftarrow h_{\text{init}}$, $\boldsymbol{x} \leftarrow \boldsymbol{x}_T$, $\boldsymbol{x}_{\text{prev}} \leftarrow \boldsymbol{x}_T$, $r_1 \leftarrow \frac{1}{3}$, $r_2 \leftarrow \frac{2}{3}$, NFE $\leftarrow 0$
 2: while $s > \epsilon$ do
 3:     $t \leftarrow t_\lambda(\lambda_s + h)$
 4:     $\boldsymbol{x}_2 \leftarrow$ dpm-solver-2$(\boldsymbol{x}, s, t, r_1)$
 5:     $\boldsymbol{x}_3 \leftarrow$ dpm-solver-3$(\boldsymbol{x}, s, t, r_1, r_2)$ (Share the same function values with dpm-solver-2.)
 6:     $\boldsymbol{\delta} \leftarrow \max(\epsilon_{\text{atol}}, \epsilon_{\text{rtol}} \max(|\boldsymbol{x}_2|, |\boldsymbol{x}_{\text{prev}}|))$
 7:     $E_3 \leftarrow \frac{1}{\sqrt{D}} \| \frac{\boldsymbol{x}_2 - \boldsymbol{x}_3}{\boldsymbol{\delta}} \|_2$
 8:     if $E_3 \leq 1$ then
 9:         $\boldsymbol{x}_{\text{prev}} \leftarrow \boldsymbol{x}_2$, $\boldsymbol{x} \leftarrow \boldsymbol{x}_3$, $s \leftarrow t$
10:     end if
11:     $h \leftarrow \min(\theta h E_3^{-\frac{1}{3}}, \lambda_\epsilon - \lambda_s)$
12:     NFE $\leftarrow$ NFE $+ 3$
13: end while
14: return $\boldsymbol{x}$, NFE
```

D.2 Sampling from Discrete-Time DPMs

In this section, we discuss the more general case for discrete-time DPMs, in which we consider the 1000-step DPMs [2] and the 4000-step DPMs [16], and we also consider the end time ϵ for sampling.

Discrete-time DPMs [2] train the noise prediction model at N fixed time steps $\{t_n\}_{n=1}^N$. In practice, $N = 1000$ or $N = 4000$, and the implementation of the 4000-step DPMs [16] converts the time steps of 4000-step DPMs to the range of 1000-step DPMs. Specifically, the noise prediction model is parameterized by $\tilde{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_n, \frac{1000n}{N})$ for $n = 0, \dots, N-1$, where each $\boldsymbol{x}_n$ is corresponding to the value at time t_{n+1}. In practice, these discrete-time DPMs usually choose uniform time steps between $[0, T]$, thus $t_n = \frac{nT}{N}$, for $n = 1, \dots, N$.

However, the discrete-time noise prediction model cannot predict the noise at time less than the smallest time t_1. As the smallest time step $t_1 = \frac{T}{N}$ and the corresponding discrete-time noise prediction model at time t_1 is $\tilde{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_0, 0)$, we need to "scale" the discrete time steps $[t_1, t_N] = [\frac{T}{N}, T]$ to the continuous time range $[\epsilon, T]$. We propose two types of scaling as following.

Type-1. Scale the discrete time steps $[t_1, t_N] = [\frac{T}{N}, T]$ to the continuous time range $[\frac{T}{N}, T]$, and let $\boldsymbol{\epsilon}_\theta(\cdot, t) = \boldsymbol{\epsilon}_\theta(\cdot, \frac{T}{N})$ for $t \in [\epsilon, \frac{T}{N}]$. In this case, we can define the continuous-time noise prediction model by

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, t) = \tilde{\boldsymbol{\epsilon}}_\theta\left(\boldsymbol{x}, 1000 \cdot \max\left(t - \frac{T}{N}, 0\right)\right), \tag{D.1}$$

where the continuous time $t \in [\epsilon, \frac{T}{N}]$ maps to the discrete input 0, and the continuous time T maps to the discrete input $\frac{1000(N-1)}{N}$.

Type-2. Scale the discrete time steps $[t_1, t_N] = [\frac{T}{N}, T]$ to the continuous time range $[0, T]$. In this case, we can define the continuous-time noise prediction model by

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, t) = \tilde{\boldsymbol{\epsilon}}_\theta\left(\boldsymbol{x}, 1000 \cdot \frac{(N-1)t}{NT}\right), \tag{D.2}$$

where the continuous time 0 maps to the discrete input 0, and the continuous time T maps to the discrete input $\frac{1000(N-1)}{N}$.

Note that the input time of $\tilde{\boldsymbol{\epsilon}}_\theta$ may not be integers, but we find that the noise prediction model can still work well, and we hypothesize that it is because of the smooth time embeddings (e.g., position embeddings [2]). By such reparameterization, the noise prediction model can adopt the continuous-time steps as input, and thus we can also use DPM-Solver for fast sampling.

In practice, we have $T = 1$, and the smallest discrete time $t_1 = 10^{-3}$. For fixed K number of function evaluations, we empirically find that for small K, the Type-1 with $\epsilon = 10^{-3}$ may have better sample quality, and for large K, the Type-2 with $\epsilon = 10^{-4}$ may have better sample quality. We refer to Appendix E for detailed results.

D.3 DPM-Solver in 20 Function Evaluations

Given a fixed budget $K \leq 20$ of the number of function evaluations, we uniformly divide the interval $[\lambda_T, \lambda_\epsilon]$ into $M = (\lfloor K/3 \rfloor + 1)$ segments, and take M steps to generate samples. The M steps are dependent on the remainder R of K mod 3 to make sure the total number of function evaluations is exactly K.

- If $R = 0$, we firstly take $M-2$ steps of DPM-Solver-3, and then take 1 step of DPM-Solver-2 and 1 step of DPM-Solver-1. The total number of function evaluations is $3 \cdot (\frac{K}{3} - 1) + 2 + 1 = K$.
- If $R = 1$, we firstly take $M-1$ steps of DPM-Solver-3 and then take 1 step of DPM-Solver-1. The total number of function evaluations is $3 \cdot (\frac{K-1}{3}) + 1 = K$.
- If $R = 2$, we firstly take $M-1$ steps of DPM-Solver-3 and then take 1 step of DPM-Solver-2. The total number of function evaluations is $3 \cdot (\frac{K-2}{3}) + 2 = K$.

We empirically find that this design of time steps can greatly improve the generation quality, and DPM-Solver can generate comparable samples in 10 steps and high-quality samples in 20 steps.

D.4 Analytical Formulation of the function $t_\lambda(\cdot)$ (the inverse function of $\lambda(t)$)

The costs of computing $t_\lambda(\cdot)$ is negligible, because for the noise schedules of α_t and σ_t used in previous DPMs ("linear" and "cosine") [2, 16], both $\lambda(t)$ and its inverse function $t_\lambda(\cdot)$ have analytic formulations. We mainly consider the variance preserving type here, since it is the most widely-used type. The functions of other types (variance exploding and sub-variance preserving type) can be similarly derived.

Linear Noise Schedule [2]. We have

$$\log \alpha_t = -\frac{(\beta_1 - \beta_0)}{4} t^2 - \frac{\beta_0}{2} t,$$

where $\beta_0 = 0.1$ and $\beta_1 = 20$, following [3]. As $\sigma_t = \sqrt{1 - \alpha_t^2}$, we can compute λ_t analytically. Moreover, the inverse function is

$$t_\lambda(\lambda) = \frac{1}{\beta_1 - \beta_0} \left(\sqrt{\beta_0^2 + 2(\beta_1 - \beta_0) \log\left(e^{-2\lambda} + 1\right)} - \beta_0 \right).$$

To reduce the influence of numerical issues, we can compute t_λ by the following equivalent formulation:

$$t_\lambda(\lambda) = \frac{2 \log\left(e^{-2\lambda} + 1\right)}{\sqrt{\beta_0^2 + 2(\beta_1 - \beta_0) \log\left(e^{-2\lambda} + 1\right)} + \beta_0}.$$

And we solve diffusion ODEs between $[\epsilon, T]$, where $T = 1$.

Cosine Noise Schedule [16]. Denote

$$\log \alpha_t = \log\left(\cos\left(\frac{\pi}{2} \cdot \frac{t + s}{1 + s} \right) \right) - \log\left(\cos\left(\frac{\pi}{2} \cdot \frac{s}{1 + s} \right) \right),$$

where $s = 0.008$, following [16]. As [16] clipped the derivatives to ensure the numerical stability, we also clip the maximum time $T = 0.9946$. As $\sigma_t = \sqrt{1 - \alpha_t^2}$, we can compute λ_t analytically. Moreover, given a fixed λ, let

$$f(\lambda) = -\frac{1}{2} \log\left(e^{-2\lambda} + 1\right),$$

which computes the corresponding $\log \alpha$ for λ. Then the inverse function is

$$t_\lambda(\lambda) = \frac{2(1 + s)}{\pi} \arccos\left(e^{f(\lambda) + \log \cos\left(\frac{\pi s}{2(1+s)}\right)} \right) - s.$$

And we solve diffusion ODEs between $[\epsilon, T]$, where $T = 0.9946$.

D.5 Conditional Sampling by DPM-Solver

DPM-Solver can also be used for conditional sampling, with a simple modification. The conditional generation needs to sample from the conditional diffusion ODE [3, 4] which includes the conditional noise prediction model. We follow the classifier guidance method [4] to define the conditional noise prediction model as $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t, y) := \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) - s \cdot \sigma_t \nabla_{\boldsymbol{x}} \log p_t(y|\boldsymbol{x}_t; \theta)$, where $p_t(y|\boldsymbol{x}_t; \theta)$ is a pre-trained classifier and s is the classifier guidance scale (default is 1.0). Thus, we can use DPM-Solver to solve this diffusion ODE for fast conditional sampling, as shown in Fig. 1.

D.6 Numerical Stability

As we need to compute $e^{h_i} - 1$ in the algorithm of DPM-Solver, we follow [10] to use `expm1(`h_i`)` instead of `exp(`h_i`)-1` to improve numerical stability.

E Experiment Details

We test our method for sampling the most widely-used *variance-preserving* (VP) type DPMs [1, 2]. In this case, we have $\alpha_t^2 + \sigma_t^2 = 1$ for all $t \in [0, T]$ and $\tilde{\sigma} = 1$. In spite of this, our method and theoretical results are general and independent of the choice of the noise schedule α_t and σ_t.

For all experiments, we evaluate DPM-Solver on NVIDIA A40 GPUs. However, the computation resource can be other types of GPU, such as NVIDIA GeForce RTX 2080Ti, because we can tune the batch size for sampling.

E.1 Diffusion ODEs w.r.t. λ

Alternatively, the diffusion ODE can be reparameterized to the λ domain. In this section, we propose the formulation of diffusion ODEs w.r.t. λ for VP type, and other types can be similarly derived.

For a given λ, denote $\hat{\alpha}_\lambda := \alpha_{t(\lambda)}$, $\hat{\sigma}_\lambda := \sigma_{t(\lambda)}$. As $\hat{\alpha}_\lambda^2 + \hat{\sigma}_\lambda^2 = 1$, we can prove that $\frac{\mathrm{d}\lambda}{\mathrm{d}\hat{\alpha}_\lambda} = \frac{1}{\hat{\alpha}_\lambda \hat{\sigma}_\lambda^2}$, so $\frac{\mathrm{d}\log\hat{\alpha}_\lambda}{\mathrm{d}\lambda} = \hat{\sigma}_\lambda^2$. Applying change-of-variable to Eq. (2.7), we have

$$\frac{\mathrm{d}\hat{\boldsymbol{x}}_\lambda}{\mathrm{d}\lambda} = \hat{\boldsymbol{h}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) := \hat{\sigma}_\lambda^2 \hat{\boldsymbol{x}}_\lambda - \hat{\sigma}_\lambda \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda). \tag{E.1}$$

The ODE Eq. (E.1) can be also solved directly by RK methods, and we use such formulation for the experiments of RK2 (λ) and RK3 (λ) in Table 1.

E.2 Code Implementation

We implement our code with both JAX (for continuous-time DPMs) and PyTorch (for discrete-time DPMs), and our code is released at `https://github.com/LuChengTHU/dpm-solver`.

E.3 Sample Quality Comparison with Continuous-Time Sampling Methods

Table 3 shows the detailed FID results, which is corresponding to Fig. 2a. We use the official code and checkpoint in [3], the code license is Apache License 2.0. We use their released "checkpoint_8" of the "VP deep" type. We compare methods for $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$. We find that the sampling methods based on diffusion SDEs can achieve better sample quality with $\epsilon = 10^{-3}$; and that the sampling methods based on diffusion ODEs can achieve better sample quality with $\epsilon = 10^{-4}$. For DPM-Solver, we find that DPM-Solver with less than 15 NFE can achieve better FID with $\epsilon = 10^{-3}$ than $\epsilon = 10^{-4}$, while DPM-Solver with more than 15 NFE can achieve better FID with $\epsilon = 10^{-4}$ than $\epsilon = 10^{-3}$.

For the diffusion SDEs with Euler discretization, we use the PC sampler in [3] with "euler_maruyama" predictor and no corrector, which uses uniform time steps between T and ϵ. We add the "denoise" trick at the final step, which can greatly improve the FID score for $\epsilon = 10^{-3}$.

For the diffusion SDEs with Improved Euler discretization [20], we follow the results in their original paper, which only includes the results with $\epsilon = 10^{-3}$. The corresponding relative tolerance ϵ_{rel} are 0.50, 0.10 and 0.05, respectively.

Table 3: Sample quality measured by FID ↓ on CIFAR-10 dataset with continuous-time methods, varying the number of function evaluations (NFE).

Sampling method \ NFE			10	12	15	20	50	200	1000
CIFAR-10 (continuous-time model (VP deep) [3], linear noise schedule)									
SDE	Euler (denoise) [3]	$\epsilon = 10^{-3}$	304.73	278.87	248.13	193.94	66.32	12.27	**2.44**
		$\epsilon = 10^{-4}$	444.63	427.54	395.95	300.41	101.66	22.98	5.01
	Improved Euler [20]	$\epsilon = 10^{-3}$	82.42(NFE=48), 2.73(NFE=151), 2.44(NFE=180)						
ODE	RK45 Solver [28, 3]	$\epsilon = 10^{-3}$	19.55(NFE=26), 17.81(NFE=38), 3.55(NFE=62)						
		$\epsilon = 10^{-4}$	51.66(NFE=26), 21.54(NFE=38), 12.72(NFE=50), 2.61(NFE=62)						
	DPM-Solver (**ours**)	$\epsilon = 10^{-3}$	**4.70**	**3.75**	**3.24**	3.99	3.84 (NFE = 42)		
		$\epsilon = 10^{-4}$	6.96	4.93	3.35	**2.87**	**2.59 (NFE = 51)**		

For the diffusion ODEs with RK45 Solver, we use the code in [3], and tune the `atol` and `rtol` of the solver. For the NFE from small to large, we use the same `atol` = `rtol` = 0.1, 0.01, 0.001 for the results of $\epsilon = 10^{-3}$, and the same `atol` = `rtol` = 0.1, 0.05, 0.02, 0.01, 0.001 for the results of $\epsilon = 10^{-4}$, respectively.

For the diffusion ODEs with DPM-Solver, we use the method in Appendix D.3 for NFE ≤ 20, and the adaptive step size solver in Appendix C. For $\epsilon = 10^{-3}$, we use DPM-Solver-12 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$. For $\epsilon = 10^{-4}$, we use DPM-Solver-23 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$.

E.4 Sample Quality Comparison with RK Methods

Table 1 shows the different performance of RK methods and DPM-Solver-2 and 3. We list the detailed settings in this section.

Assume we have an ODE with

$$\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} = \boldsymbol{F}(\boldsymbol{x}_t, t),$$

Starting with $\tilde{\boldsymbol{x}}_{t_{i-1}}$ at time t_{i-1}, we use RK2 to approximate the solution $\tilde{\boldsymbol{x}}_{t_i}$ at time t_i in the following formulation (which is known as the explicit midpoint method):

$$\begin{aligned}
h_i &= t_i - t_{i-1}, \\
s_i &= t_{i-1} + \frac{1}{2}h_i, \\
\boldsymbol{u}_i &= \tilde{\boldsymbol{x}}_{t_{i-1}} + \frac{h_i}{2}\boldsymbol{F}(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}), \\
\tilde{\boldsymbol{x}}_{t_i} &= \tilde{\boldsymbol{x}}_{t_{i-1}} + h_i\boldsymbol{F}(\boldsymbol{u}_i, s_i).
\end{aligned}$$

And we use the following RK3 to approximate the solution $\tilde{\boldsymbol{x}}_{t_i}$ at time t_i (which is known as "Heun's third-order method"), because it is very similar to our proposed DPM-Solver-3:

$$\begin{aligned}
h_i &= t_i - t_{i-1}, \quad r_1 = \frac{1}{3}, \quad r_2 = \frac{2}{3}, \\
s_{2i-1} &= t_{i-1} + r_1 h_i, \quad s_{2i} = t_{i-1} + r_2 h_i, \\
\boldsymbol{u}_{2i-1} &= \tilde{\boldsymbol{x}}_{t_{i-1}} + r_1 h_i \boldsymbol{F}(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}), \\
\boldsymbol{u}_{2i} &= \tilde{\boldsymbol{x}}_{t_{i-1}} + r_2 h_i \boldsymbol{F}(\boldsymbol{u}_{2i-1}, s_{2i-1}), \\
\tilde{\boldsymbol{x}}_{t_i} &= \tilde{\boldsymbol{x}}_{t_{i-1}} + \frac{h_i}{4}\boldsymbol{F}(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1}) + \frac{3h_i}{4}\boldsymbol{F}(\boldsymbol{u}_{2i}, s_{2i}).
\end{aligned}$$

We use $\boldsymbol{F}(\boldsymbol{x}_t, t) = \boldsymbol{h}_\theta(\boldsymbol{x}_t, t)$ in Eq. (2.7) for the results with RK2 (t) and RK3 (t), and $\boldsymbol{F}(\hat{\boldsymbol{x}}_\lambda, \lambda) = \hat{\boldsymbol{h}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda)$ in Eq. (E.1) for the results with RK2 (λ) and RK3 (λ). For all experiments, we use the uniform step size w.r.t. t or λ.

E.5 Sample Quality Comparison with Discrete-Time Sampling Methods

We compare DPM-Solver with other discrete-time sampling methods for DPMs, as shown in Table 4 and Table 5. We use the code in [19] for sampling with DDPM and DDIM, and the code license

Table 4: Sample quality measured by FID ↓ on CIFAR-10, CelebA 64×64 and ImageNet 64×64 with discrete-time DPMs, varying the number of function evaluations (NFE). The method †GGDM needs extra training, and some results are missing in their original papers, which are replaced by "\".

Sampling method \ NFE		10	12	15	20	50	200	1000
CIFAR-10 (discrete-time model [2], linear noise schedule)								
DDPM [2]	Discrete	278.67	246.29	197.63	137.34	32.63	4.03	**3.16**
Analytic-DDPM [21]	Discrete	35.03	27.69	20.82	15.35	7.34	4.11	3.84
Analytic-DDIM [21]	Discrete	14.74	11.68	9.16	7.20	4.28	3.60	3.86
†GGDM [18]	Discrete	8.23	\	6.12	4.72	\	\	\
DDIM [19]	Discrete	13.58	11.02	8.92	6.94	4.73	4.07	3.95
DPM-Solver (Type-1 discrete)	$\epsilon = 10^{-3}$	**6.37**	**4.65**	**3.78**	4.28	3.90 (NFE = 44)		
	$\epsilon = 10^{-4}$	11.32	7.31	4.75	3.80	3.57 (NFE = 46)		
DPM-Solver (Type-2 discrete)	$\epsilon = 10^{-3}$	6.42	4.86	4.39	5.52	5.22 (NFE = 42)		
	$\epsilon = 10^{-4}$	10.16	6.26	4.17	**3.72**	**3.48 (NFE = 44)**		
CelebA 64×64 (discrete-time model [19], linear noise schedule)								
DDPM [2]	Discrete	310.22	277.16	207.97	120.44	29.25	3.90	3.50
Analytic-DDPM [21]	Discrete	28.99	25.27	21.80	18.14	11.23	6.51	5.21
Analytic-DDIM [21]	Discrete	15.62	13.90	12.29	10.45	6.13	3.46	3.13
DDIM [19]	Discrete	10.85	9.99	7.78	6.64	5.23	4.78	4.88
DPM-Solver (Type-1 discrete)	$\epsilon = 10^{-3}$	7.15	5.51	4.28	4.40	4.23 (NFE = 36)		
	$\epsilon = 10^{-4}$	6.92	4.20	**3.05**	**2.82**	**2.71 (NFE = 36)**		
DPM-Solver (Type-2 discrete)	$\epsilon = 10^{-3}$	7.33	6.23	5.85	6.87	6.68 (NFE = 36)		
	$\epsilon = 10^{-4}$	**5.83**	**3.71**	3.11	3.13	3.10 (NFE = 36)		
ImageNet 64×64 (discrete-time model [16], cosine noise schedule)								
DDPM [2]	Discrete	305.43	287.66	256.69	209.73	83.86	28.39	17.58
Analytic-DDPM [21]	Discrete	60.65	53.66	45.98	37.67	22.45	**17.16**	**16.14**
Analytic-DDIM [21]	Discrete	70.62	54.88	41.56	30.88	19.23	17.49	17.57
†GGDM [18]	Discrete	37.32	\	24.69	20.69	\	\	\
DDIM [19]	Discrete	67.07	52.69	40.49	30.67	20.10	17.84	17.73
DPM-Solver (Type-1 discrete)	$\epsilon = 10^{-3}$	24.44	20.03	19.31	18.59	17.50 (NFE = 48)		
	$\epsilon = 10^{-4}$	27.74	23.66	20.09	19.06	17.56 (NFE = 51)		
DPM-Solver (Type-2 discrete)	$\epsilon = 10^{-3}$	**24.40**	**19.97**	**19.23**	**18.53**	**17.47 (NFE = 57)**		
	$\epsilon = 10^{-4}$	27.72	23.75	20.02	19.08	17.62 (NFE = 48)		

Table 5: Sample quality measured by FID ↓ on ImageNet 128×128 with classifier guidance and on LSUN bedroom 256×256, varying the number of function evaluations (NFE). For DDIM and DDPM, we use uniform time steps for all the experiments, except that the experiment† uses the fine-tuned time steps by [4]. For DPM-Solver, we use the uniform logSNR steps as described in Appendix D.3.

Sampling method \ NFE		10	12	15	20	50	100	250
ImageNet 128×128 (discrete-time model [4], linear noise schedule, classifier guidance scale: 1.25)								
DDPM [2]	Discrete	199.56	172.09	146.42	119.13	49.38	23.27	**2.97**
DDIM [19]	Discrete	11.12	9.38	8.22	7.15	5.05	4.18	3.54
DPM-Solver (Type-1 discrete)	$\epsilon = 10^{-3}$	**7.32**	**4.08**	**3.60**	3.89	3.63	3.62	3.63
	$\epsilon = 10^{-4}$	13.91	5.84	4.00	**3.52**	**3.13**	**3.10**	3.09
LSUN bedroom 256×256 (discrete-time model [4], linear noise schedule)								
DDPM [2]	Discrete	274.67	251.26	224.88	190.14	82.70	34.89	†2.02
DDIM [19]	Discrete	10.05	7.51	5.90	4.98	2.92	2.30	2.02
DPM-Solver (Type-1 discrete)	$\epsilon = 10^{-3}$	**6.10**	4.29	3.30	3.09	2.53	2.46	2.46
	$\epsilon = 10^{-4}$	8.04	**4.21**	**2.94**	**2.60**	**2.01**	**1.95**	**1.94**

is MIT License. We use the code in [21] for sampling with Analytic-DDPM and Analytic-DDIM, whose license is unknown. We directly follow the best results in the original paper of GGDM [18].

For the CIFAR-10 experiments, we use the pretrained checkpoint by [2], which is also provided in the released code in [19]. We use quadratic time steps for DDPM and DDIM, which empirically has better FID performance than the uniform time steps [19]. We use the uniform time steps for Analytic-DDPM and Analytic-DDIM. For DPM-Solver, we use both Type-1 discrete and Type-2 discrete methods to convert the discrete-time model to the continuous-time model. We use the method in Appendix D.3 for NFE ≤ 20, and the adaptive step size solver in Appendix C for NFE > 20. For all the experiments, we use DPM-Solver-12 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$.

For the CelebA 64x64 experiments, we use the pretrained checkpoint by [19]. We use quadratic time steps for DDPM and DDIM, which empirically has better FID performance than the uniform time steps [19]. We use the uniform time steps for Analytic-DDPM and Analytic-DDIM. For DPM-Solver, we use both Type-1 discrete and Type-2 discrete methods to convert the discrete-time model to the continuous-time model. We use the method in Appendix D.3 for NFE ≤ 20, and the adaptive step size solver in Appendix C for NFE > 20. For all the experiments, we use DPM-Solver-12 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$. Note that our best FID results on CelebA 64x64 is even better than the 1000-step DDPM (and all the other methods).

For the ImageNet 64x64 experiments, we use the pretrained checkpoint by [16], and the code license is MIT License. We use the uniform time steps for DDPM and DDIM, following [19]. We use the uniform time steps for Analytic-DDPM and Analytic-DDIM. For DPM-Solver, we use both Type-1 discrete and Type-2 discrete methods to convert the discrete-time model to the continuous-time model. We use the method in Appendix D.3 for NFE ≤ 20, and the adaptive step size solver in Appendix C for NFE > 20. For all the experiments, we use DPM-Solver-23 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$. Note that the ImageNet dataset includes real human photos and it may have privacy issues, as discussed in [42].

For the ImageNet 128x128 experiments, we use classifier guidance for sampling with the pretrained checkpoints (for both the diffusion model and the classifier model) by [4], and the code license is MIT License. We use the uniform time steps for DDPM and DDIM, following [19]. For DPM-Solver, we only use Type-1 discrete method to convert the discrete-time model to the continuous-time model. We use the method in Appendix D.3 for NFE ≤ 20, and the adaptive step size solver DPM-Solver-12 with relative tolerance $\epsilon_{\text{rtol}} = 0.05$ (detailed in Appendix C) for NFE > 20. For all the experiments, we set the classifier guidance scale $s = 1.25$, which is the best setting for DDIM in [4] (we refer to their Table 14 for details).

For the LSUN bedroom 256x256 experiments, we use the unconditional pretrained checkpoint by [4], and the code license is MIT License. We use the uniform time steps for DDPM and DDIM, following [19]. For DPM-Solver, we only use Type-1 discrete method to convert the discrete-time model to the continuous-time model. We use the method in Appendix D.3 for DPM-Solver.

E.6 Comparing Different Orders of DPM-Solver

We also compare the sample quality of the different orders of DPM-Solver, as shown in Table 6. We use DPM-Solver-1,2,3 with uniform time steps w.r.t. λ, and the fast version in Appendix D.3 for NFE less than 20, and we name it as *DPM-Solver-fast*. For the discrete-time models, we only compare the Type-2 discrete method, and the results of Type-1 are similar.

As the actual NFE of DPM-Solver-2 is $2 \times \lfloor \text{NFE}/2 \rfloor$ and the actual NFE of DPM-Solver-3 is $3 \times \lfloor \text{NFE}/3 \rfloor$, which may be smaller than NFE, we use the notation † to note that the actual NFE is less than the given NFE. We find that for NFE less than 20, the proposed fast version (DPM-Solver-fast) is usually better than the single order method, and for larger NFE, DPM-Solver-3 is better than DPM-Solver-2, and DPM-Solver-2 is better than DPM-Solver-1, which matches our proposed convergence rate analysis.

E.7 Runtime Comparison between DPM-Solver and DDIM

Theoretically, for the same NFE, the runtime of DPM-Solver and DDIM are almost the same (linear to NFE) because the main computation costs are the serial evaluations of the large neural network $\boldsymbol{\epsilon}_\theta$ and the other coefficients are **analytically** computed with ignorable costs.

Table 6: Sample quality measured by FID ↓ of different orders of DPM-Solver, varying the number of function evaluations (NFE). The results with † means the actual NFE is smaller than the given NFE because the given NFE cannot be divided by 2 or 3. For DPM-Solver-fast, we only evaluate it for NFE less than 20, because it is almost the same as DPM-Solver-3 for large NFE.

Sampling method \ NFE		10	12	15	20	50	200	1000
CIFAR-10 (VP deep continuous-time model [3])								
$\epsilon = 10^{-3}$	DPM-Solver-1	11.83	9.69	7.78	6.17	4.28	3.85	3.83
	DPM-Solver-2	5.94	4.88	†4.30	3.94	3.78	3.74	3.74
	DPM-Solver-3	†18.37	5.53	4.08	†4.04	†3.81	†3.78	†3.78
	DPM-Solver-fast	**4.70**	**3.75**	**3.24**	3.99	\	\	\
$\epsilon = 10^{-4}$	DPM-Solver-1	11.29	9.07	7.15	5.50	3.32	2.72	2.64
	DPM-Solver-2	7.30	5.28	†4.23	3.26	2.69	**2.60**	**2.59**
	DPM-Solver-3	†54.56	6.03	3.55	†2.90	†**2.65**	†2.62	†2.62
	DPM-Solver-fast	6.96	4.93	3.35	**2.87**	\	\	\
CIFAR-10 (DDPM discrete-time model [2]), DPM-Solver with Type-2 discrete								
$\epsilon = 10^{-3}$	DPM-Solver-1	16.69	13.63	11.08	8.90	6.24	5.44	5.29
	DPM-Solver-2	7.90	6.15	†5.53	5.24	5.23	5.25	5.25
	DPM-Solver-3	†24.37	8.20	5.73	†5.43	†5.29	†5.25	†5.25
	DPM-Solver-fast	**6.42**	**4.86**	4.39	5.52	\	\	\
$\epsilon = 10^{-4}$	DPM-Solver-1	13.61	10.98	8.71	6.79	4.36	3.63	3.49
	DPM-Solver-2	11.80	6.31	†5.23	3.95	3.50	3.46	3.46
	DPM-Solver-3	†67.02	9.45	5.21	†3.81	†**3.49**	†**3.45**	†**3.45**
	DPM-Solver-fast	10.16	6.26	**4.17**	**3.72**	\	\	\
CelebA 64×64 (discrete-time model [19], linear noise schedule), DPM-Solver with Type-2 discrete								
$\epsilon = 10^{-3}$	DPM-Solver-1	18.66	16.30	13.92	11.84	8.85	7.24	6.93
	DPM-Solver-2	5.89	5.83	†6.08	6.38	6.78	6.84	6.85
	DPM-Solver-3	†11.45	5.46	6.18	†6.51	†6.87	†6.84	†6.85
	DPM-Solver-fast	7.33	6.23	5.85	6.87	\	\	\
$\epsilon = 10^{-4}$	DPM-Solver-1	13.24	11.13	9.08	7.24	4.50	3.48	3.25
	DPM-Solver-2	**4.28**	**3.40**	†3.30	3.17	**3.19**	**3.20**	**3.20**
	DPM-Solver-3	†49.48	3.84	**3.09**	†3.15	†3.20	†**3.20**	†**3.20**
	DPM-Solver-fast	5.83	3.71	3.11	**3.13**	\	\	\
ImageNet 64×64 (discrete-time model [16], cosine noise schedule), DPM-Solver with Type-2 discrete								
$\epsilon = 10^{-3}$	DPM-Solver-1	32.84	28.54	24.79	21.71	18.30	17.45	17.18
	DPM-Solver-2	29.20	24.97	†22.26	19.94	17.79	17.29	**17.27**
	DPM-Solver-3	†57.48	24.62	19.76	†18.95	†**17.52**	**17.26**	**17.27**
	DPM-Solver-fast	**24.40**	**19.97**	**19.23**	**18.53**	\	\	\
$\epsilon = 10^{-4}$	DPM-Solver-1	32.31	28.44	25.15	22.38	19.14	17.95	17.44
	DPM-Solver-2	33.16	27.28	†24.26	20.58	18.04	17.46	17.41
	DPM-Solver-3	†162.27	27.28	22.38	†19.39	†17.71	†17.43	†17.41
	DPM-Solver-fast	27.72	23.75	20.02	19.08	\	\	\

Table 7 shows the runtime of DPM-Solver and DDIM on a single NVIDIA A40, varying different datasets and NFE. We use `torch.cuda.Event` and `torch.cuda.synchronize` for accurately computing the runtime. We use the discrete-time pretrained diffusion models for each dataset. We evaluate the runtime for 8 batches and computes the mean and std of the runtime. We use 64 batch size for LSUN bedroom 256x256 due to the GPU memory limitation, and 128 batch size for other datasets.

For DDIM, we use the official implementation[3]. We find that our implementation of DPM-Solver reduces some repetitive computation of the coefficients, so under the same NFE, DPM-Solver is slightly faster than DDIM of their implementation. Nevertheless, the runtime evaluation results show

[3]`https://github.com/ermongroup/ddim`

Table 7: Runtime of a single batch (second / batch, ±std) on a single NVIDIA A40 of DDIM and DPM-Solver for sampling by discrete-time diffusion models, varying the number of function evaluations (NFE).

Sampling method \ NFE	10	20	50	100
CIFAR-10 32×32 (batch size = 128)				
DDIM	0.956(±0.011)	1.924(±0.016)	4.838(±0.024)	9.668(±0.013)
DPM-Solver	0.923(±0.006)	1.833(±0.004)	4.580(±0.005)	9.204(±0.011)
CelebA 64×64 (batch size = 128)				
DDIM	3.253(±0.015)	6.438(±0.029)	16.132(±0.050)	32.255(±0.044)
DPM-Solver	3.126(±0.003)	6.272(±0.006)	15.676(±0.008)	31.269(±0.012)
ImageNet 64×64 (batch size = 128)				
DDIM	5.084(±0.018)	10.194(±0.022)	25.440(±0.044)	50.926(±0.042)
DPM-Solver	4.992(±0.004)	9.991(±0.003)	24.948(±0.007)	49.835(±0.028)
ImageNet 128×128 (batch size = 128, with classifier guidance)				
DDIM	29.082(±0.015)	58.159(±0.012)	145.427(±0.011)	290.874(±0.134)
DPM-Solver	28.865(±0.011)	57.645(±0.008)	144.124(±0.035)	288.157(±0.022)
LSUN bedroom 256×256 (batch size = 64)				
DDIM	37.700(±0.005)	75.316(±0.013)	188.275(±0.172)	378.790(±0.105)
DPM-Solver	36.996(±0.039)	73.873(±0.023)	184.590(±0.010)	369.090(±0.076)

that the runtime of DPM-Solver and DDIM are almost the same for the same NFE, and the runtime is approximately linear to the NFE. Therefore, the speedup for the NFE is almost the actual speedup of the runtime, so the proposed DPM-Solver can greatly speedup the sampling of DPMs.

E.8 Conditional Sampling on ImageNet 256x256

For the conditional sampling in Fig. 1, we use the pretrained checkpoint in [4] with classifier guidance (ADM-G), and the classifier scale is 1.0. The code license is MIT License. We use uniform time step for DDIM, and the fast version for DPM-Solver in Appendix D.3 (DPM-Solver-fast) with 10, 15, 20 and 100 steps.

Fig. 3 shows the conditional sample results by DDIM and DPM-Solver. We find that DPM-Solver with 15 NFE can generate comparable samples with DDIM with 100 NFE.

E.9 Additional Samples

Additional sampling results on CIFAR-10, CelebA 64x64, ImageNet 64x64, LSUN bedroom 256x256 [40], ImageNet 256x256 are reported in Figs. 4-8.

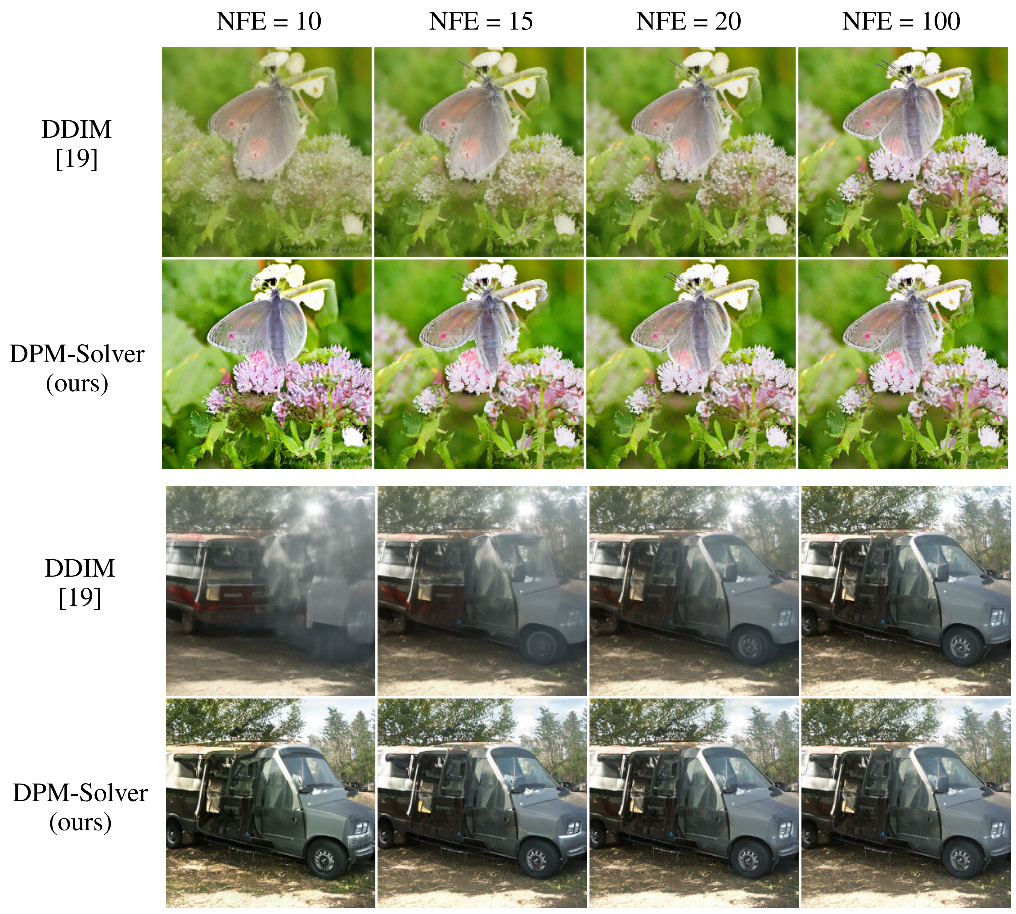

Figure 3: Samples by DDIM [19] and DPM-Solver (ours) with 10, 15, 20, 100 number of function evaluations (NFE) with the same random seed, using the pre-trained DPMs on ImageNet 256×256 with classifier guidance [4].

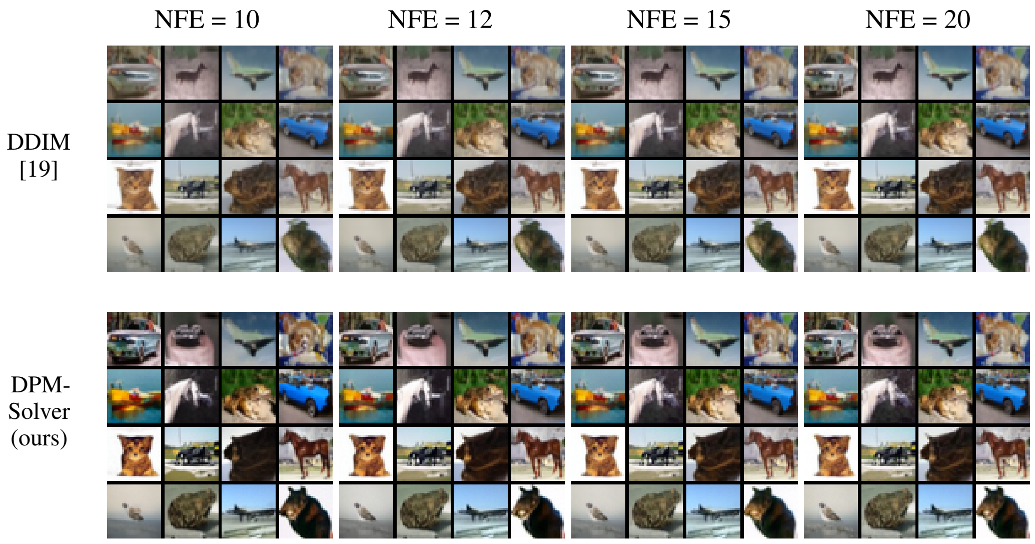

Figure 4: Random samples by DDIM [19] (quadratic time steps) and DPM-Solver (ours) with 10, 12, 15, 20 number of function evaluations (NFE) with the same random seed, using the pre-trained discrete-time DPMs [2] on CIFAR-10.

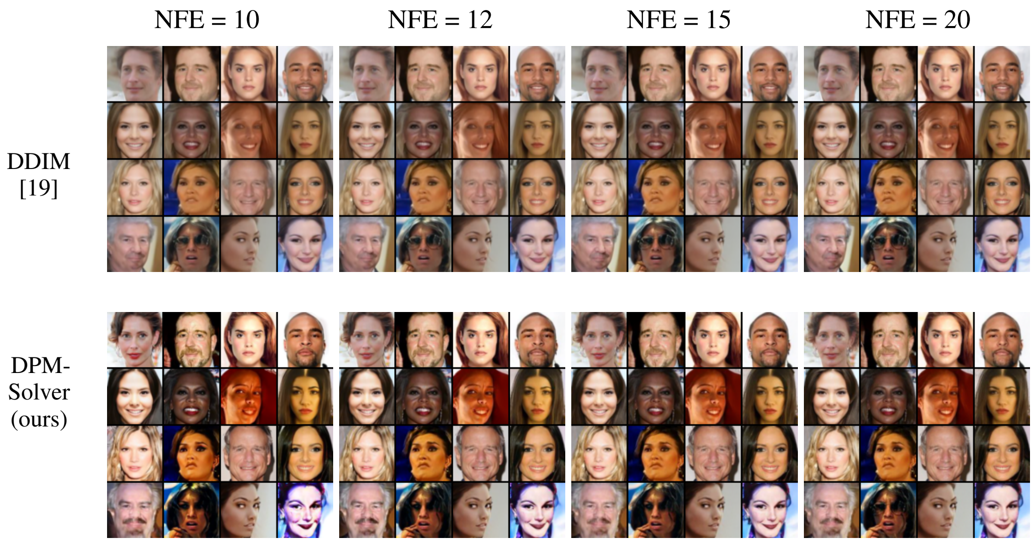

Figure 5: Random samples by DDIM [19] (quadratic time steps) and DPM-Solver (ours) with 10, 12, 15, 20 number of function evaluations (NFE) with the same random seed, using the pre-trained discrete-time DPMs [19] on CelebA 64x64.

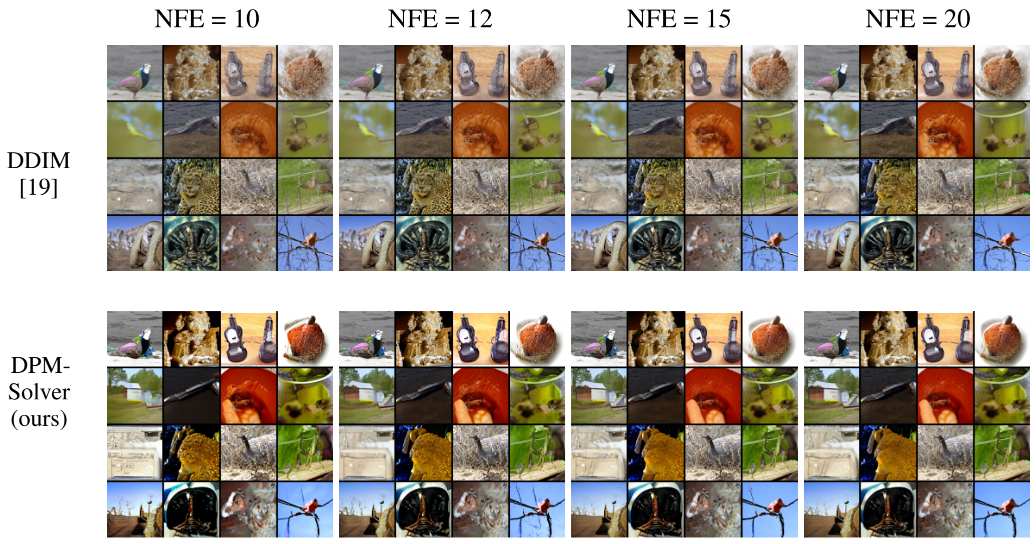

Figure 6: Random samples by DDIM [19] (uniform time steps) and DPM-Solver (ours) with 10, 12, 15, 20 number of function evaluations (NFE) with the same random seed, using the pre-trained discrete-time DPMs [16] on ImageNet 64x64.

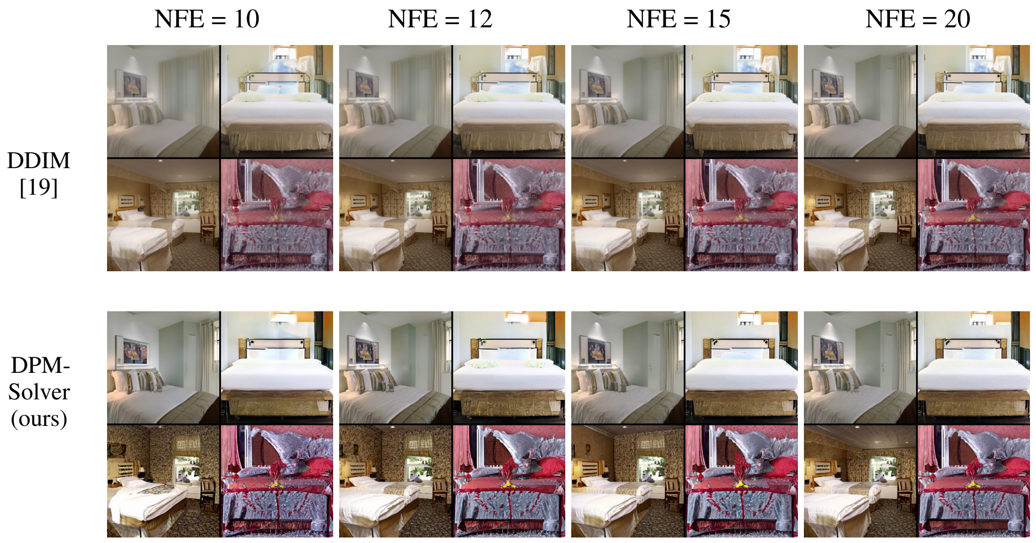

Figure 7: Random samples by DDIM [19] (uniform time steps) and DPM-Solver (ours) with 10, 12, 15, 20 number of function evaluations (NFE) with the same random seed, using the pre-trained discrete-time DPMs [4] on LSUN bedroom 256x256.

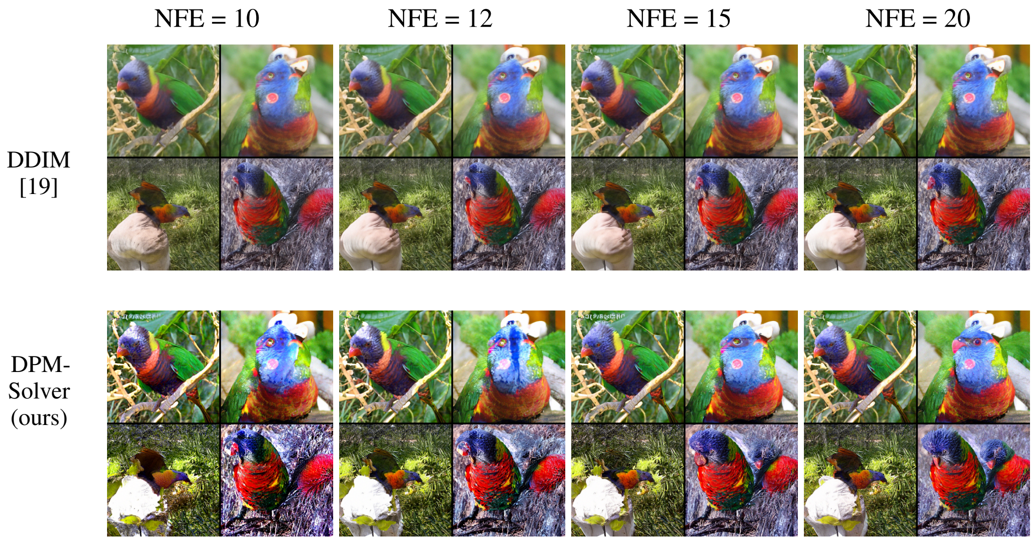

Figure 8: Random class-conditional samples (class: 90, lorikeet) by DDIM [19] (uniform time steps) and DPM-Solver (ours) with 10, 12, 15, 20 number of function evaluations (NFE) with the same random seed, using the pre-trained discrete-time DPMs [4] on ImageNet 256x256 with classifier guidance (classifier scale: 1.0).