PFDiff: Training-free Acceleration of Diffusion Models through the Gradient Guidance of Past and Future

Anonymous Author(s)
Affiliation
Address
email

Abstract

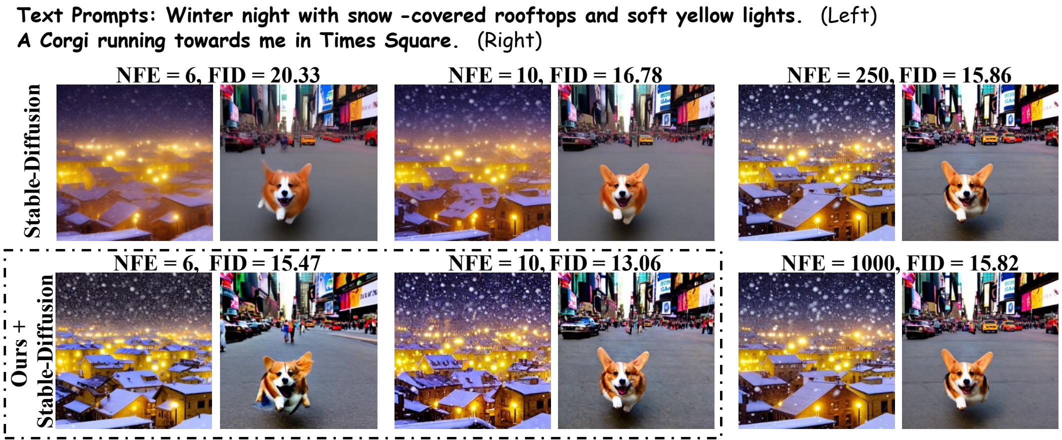
Diffusion Probabilistic Models (DPMs) have shown remarkable potential in image generation, but their sampling efficiency is hindered by the need for numerous denoising steps. Most existing solutions accelerate the sampling process by proposing fast ODE solvers. However, the inevitable discretization errors of the ODE solvers are significantly magnified when the number of function evaluations (NFE) is fewer. In this work, we propose *PFDiff*, a novel *training-free* and *orthogonal* timestep-skipping strategy, which enables existing fast ODE solvers to operate with fewer NFE. Based on two key observations: a significant similarity in the model's outputs at time step size that is not excessively large during the denoising process of existing ODE solvers, and a high resemblance between the denoising process and SGD. PFDiff, by employing gradient replacement from past time steps and foresight updates inspired by Nesterov momentum, rapidly updates intermediate states, thereby reducing unnecessary NFE while correcting for discretization errors inherent in first-order ODE solvers. Experimental results demonstrate that PFDiff exhibits flexible applicability across various pre-trained DPMs, particularly excelling in conditional DPMs and surpassing previous state-of-the-art training-free methods. For instance, using DDIM as a baseline, we achieved 16.46 FID (4 NFE) compared to 138.81 FID with DDIM on ImageNet 64x64 with classifier guidance, and 13.06 FID (10 NFE) on Stable Diffusion with 7.5 guidance scale.

1 Introduction

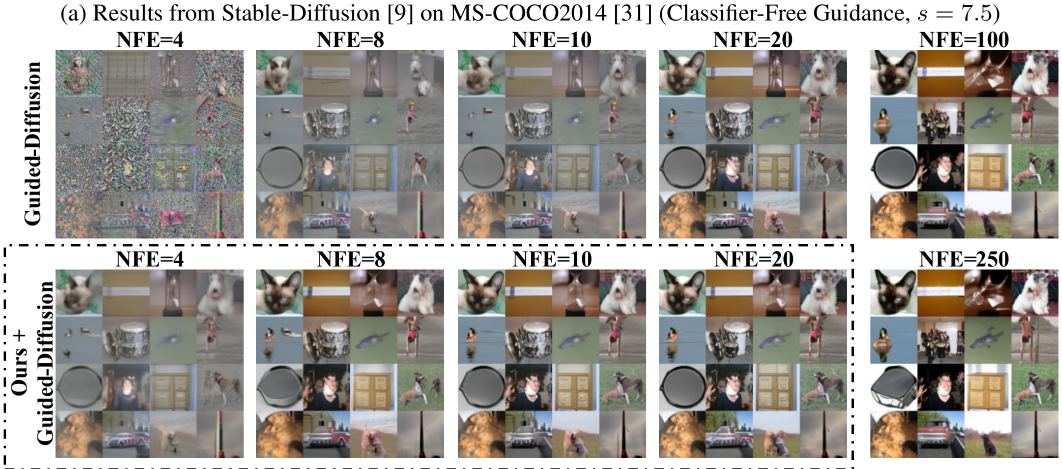
In recent years, Diffusion Probabilistic Models (DPMs) [1–4] have demonstrated exceptional modeling capabilities across various domains including image generation [5–7], video generation [8], text-to-image generation [9, 10], speech synthesis [11], and text-to-3D generation [12, 13]. They have become a key driving force advancing deep generative models. DPMs initiate with a forward process that introduces noise onto images, followed by utilizing a neural network to learn a backward process that incrementally removes noise, thereby generating images [2, 4]. Compared to other generative methods such as Generative Adversarial Networks (GANs) [14] and Variational Autoencoders (VAEs) [15], DPMs not only possess a simpler optimization target but also are capable of producing higher quality samples [5]. However, the generation of high-quality samples via DPMs requires hundreds or thousands of denoising steps, significantly lowering their sampling efficiency and becoming a major barrier to their widespread application.

Existing techniques for rapid sampling in DPMs primarily fall into two categories. First, training-based methods [16–19], which can significantly compress sampling steps, even achieving single-step sampling [19]. However, this compression often comes with a considerable additional training cost, and these methods are challenging to apply to large pre-trained models. Second, training-free samplers [20–30], which typically employ implicit or analytical solutions to Stochastic Differential Equations

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.



(a) Results from Stable-Diffusion [9] on MS-COCO2014 [31] (Classifier-Free Guidance, $s = 7.5$)



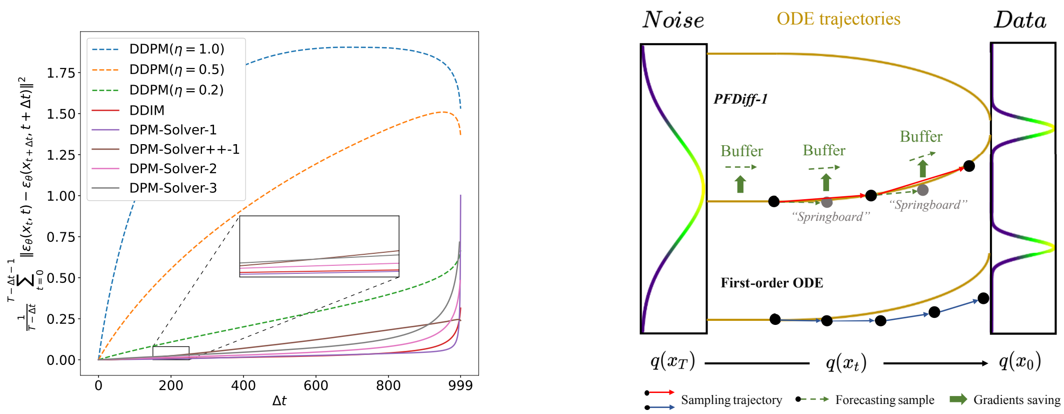
(b) Results from Guided-Diffusion [5] on ImageNet 64x64 [32] (Classifier Guidance, $s = 1.0$)

Figure 1: Sampling by conditional pre-trained DPMs [5, 9] using DDIM [20] and our method PFDiff (dashed box) with DDIM as a baseline, varying the number of function evaluations (NFE).

(SDE)/Ordinary Differential Equations (ODE) for lower-error sampling processes. For instance, Lu et al. [21, 22], by analyzing the semi-linear structure of the ODE solvers for DPMs, have sought to analytically derive optimally the solutions for DPMs' ODE solvers. These training-free sampling strategies can often be used in a plug-and-play fashion, compatible with existing pre-trained DPMs. However, when the NFE is below 10, the discretization error of these training-free methods will be significantly amplified, leading to convergence issues [21, 22], which can still be time-consuming.

To further enhance the sampling speed of DPMs, we have analyzed the potential for improvement in existing training-free accelerated methods. Initially, we observed a notably high similarity in the model's outputs for the existing ODE solvers of DPMs when time step size Δt is not extremely large, as illustrated in Fig. 2a. This observation led us to utilize the gradients that have been computed from past time steps to approximate current gradients, thereby reducing unnecessary estimation of noise network. Furthermore, due to the similarities between the sampling process of DPMs and Stochastic Gradient Descent (SGD) [33] as noted in Remark 1, we incorporated a *foresight* update mechanism using Nesterov momentum [34], known for accelerating SGD training. Specifically, we ingeniously employ prior observation to predict future gradients, then utilize the future gradients as a "*springboard*" to facilitate larger update step size Δt, as shown in Fig. 2b.

Motivated by these insights, we propose *PFDiff*, a timestep-skipping sampling algorithm that rapidly updates the current intermediate state through the gradient guidance of past and future. Notably, PFDiff is *training-free* and *orthogonal* to existing DPMs sampling algorithms, providing a new orthogonal axis for DPMs sampling. Unlike previous orthogonal sampling algorithms that compromise



(a) Gradient Changes in SDE/ODE Solvers

(b) Comparison of Sampling Trajectories

Figure 2: (a) The trend of the MSE of the noise network output $\epsilon_\theta(x_t, t)$ over time step size Δt, where η in DDPM [2] comes from $\bar{\sigma}_t$ in Eq. (6). Solid lines: ODE solvers, dashed lines: SDE solvers. (b) Comparison of partial sampling trajectories between PFDiff-1 and a first-order ODE solver, where the update directions are guided by the tangent direction of the sampling trajectories.

sampling quality for speed [28], we prove that PFDiff corrects for errors in the sampling trajectories of first-order ODE solvers. This improves sampling quality while reducing unnecessary NFE in existing ODE solvers, as illustrated in Fig. 2b. To validate the orthogonality and effectiveness of PFDiff, extensive experiments were conducted on both unconditional [2, 4, 20] and conditional [5, 9] pre-trained DPMs, with the visualization experiment of conditional DPMs depicted in Fig. 1. The results indicate that PFDiff significantly enhances the sampling performance of existing ODE solvers. Particularly in conditional DPMs, PFDiff, using only DDIM as the baseline, surpasses the previous state-of-the-art training-free sampling algorithms.

2 Background

2.1 Diffusion SDEs

Diffusion Probabilistic Models (DPMs) [1–4] aim to generate D-dimensional random variables $x_0 \in \mathbb{R}^D$ that follow a data distribution $q(x_0)$. Taking Denoising Diffusion Probabilistic Models (DDPM) [2] as an example, these models introduce noise to the data distribution through a forward process defined over discrete time steps, gradually transforming it into a standard Gaussian distribution $x_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The forward process's latent variables $\{x_t\}_{t\in[0,T]}$ are defined as follows:

$$q(x_t \mid x_0) = \mathcal{N}(x_t \mid \alpha_t x_0, \sigma_t^2 \boldsymbol{I}), \tag{1}$$

where α_t is a scalar function related to the time step t, with $\alpha_t^2 + \sigma_t^2 = 1$. In the model's reverse process, DDPM utilizes a neural network model $p_\theta(x_{t-1} \mid x_t)$ to approximate the transition probability $q(x_{t-1} \mid x_t, x_0)$,

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t, t), \sigma_\theta^2(t)\boldsymbol{I}), \tag{2}$$

where $\sigma_\theta^2(t)$ is defined as a scalar function related to the time step t. By sampling from a standard Gaussian distribution and utilizing the trained neural network, samples following the data distribution $p_\theta(x_0) = \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$ can be generated.

Furthermore, Song et al. [4] introduced SDE to model DPMs over continuous time steps, where the forward process is defined as:

$$\mathrm{d}x_t = f(t)x_t\mathrm{d}t + g(t)\mathrm{d}w_t, \quad x_0 \sim q(x_0), \tag{3}$$

where w_t represents a standard Wiener process, and f and g are scalar functions of the time step t. It's noteworthy that the forward process in Eq. (1) is a discrete form of Eq. (3), where $f(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}$ and $g^2(t) = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\sigma_t^2$. Song et al. [4] further demonstrated that there exists an equivalent reverse process from time step T to 0 for the forward process in Eq. (3):

$$\mathrm{d}x_t = \left[f(t)x_t - g^2(t)\nabla_x \log q_t(x_t)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{w}_t, \quad x_T \sim q(x_T), \tag{4}$$

where $\bar{w}$ denotes a standard Wiener process. In this reverse process, the only unknown is the *score function* $\nabla_x \log q_t(x_t)$, which can be approximated through neural networks.

2.2 Diffusion ODEs

In DPMs based on SDE, the discretization of the sampling process often requires a significant number of time steps to converge, such as the $T = 1000$ time steps used in DDPM [2]. This requirement primarily stems from the randomness introduced at each time step by the SDE. To achieve a more efficient sampling process, Song et al. [4] utilized the Fokker-Planck equation [35] to derive a *probability flow ODE* related to the SDE, which possesses the same marginal distribution at any given time t as the SDE. Specifically, the reverse process ODE derived from Eq. (3) can be expressed as:

$$\mathrm{d}x_t = \left[f(t)x_t - \frac{1}{2}g^2(t)\nabla_x \log q_t(x_t)\right]\mathrm{d}t, \quad x_T \sim q(x_T). \tag{5}$$

Unlike SDE, ODE avoids the introduction of randomness, thereby allowing convergence to the data distribution in fewer time steps. Song et al. [4] employed a high-order RK45 ODE solver [36], achieving sample quality comparable to SDE at 1000 NFE with only 60 NFE. Furthermore, research such as DDIM [20] and DPM-Solver [21] explored discrete ODE forms capable of converging in fewer NFE. For DDIM, it breaks the Markov chain constraint on the basis of DDPM, deriving a new sampling formula expressed as follows:

$$x_{t-1} = \sqrt{\alpha_{t-1}}\left(\frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t,t)}{\sqrt{\alpha_t}}\right) + \sqrt{1-\alpha_{t-1}-\bar{\sigma}_t^2}\epsilon_\theta(x_t,t) + \bar{\sigma}_t\epsilon_t, \tag{6}$$

where $\bar{\sigma}_t = \eta\sqrt{(1-\alpha_{t-1})\,/\,(1-\alpha_t)}\sqrt{1-\alpha_t/\alpha_{t-1}}$, and α_t corresponds to α_t^2 in Eq. (1). When $\eta = 1$, Eq. (6) becomes a form of DDPM [2]; when $\eta = 0$, it degenerates into an ODE, the form adopted by DDIM [20], which can obtain high-quality samples in fewer time steps.

Remark 1. *In this paper, we regard the gradient $\mathrm{d}\bar{x}_t$, the noise network output $\epsilon_\theta(x_t,t)$, and the score function $\nabla_x \log q_t(x_t)$ as expressing equivalent concepts. This is because Song et al. [4] demonstrated that $\epsilon_\theta(x_t,t) = -\sigma_t\nabla_x \log q_t(x_t)$. Moreover, we have discovered that any first-order solver of DPMs can be parameterized as $x_{t-1} = \bar{x}_t - \gamma_t\mathrm{d}\bar{x}_t + \xi\epsilon_t$. Taking DDIM [20] as an example, where $\bar{x}_t = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}}x_t$, $\gamma_t = \sqrt{\frac{\alpha_{t-1}}{\alpha_t} - \alpha_{t-1}} - \sqrt{1-\alpha_{t-1}}$, $\mathrm{d}\bar{x}_t = \epsilon_\theta(x_t,t)$, and $\xi = 0$. This indicates the similarity between SGD and the sampling process of DPMs, a discovery also implicitly suggested in the research of Xue et al. [30] and Wang et al. [37].*

3 Method

3.1 Solving for reverse process diffusion ODEs

By substituting $\epsilon_\theta(x_t,t) = -\sigma_t\nabla_x \log q_t(x_t)$ [4], Eq. (5) can be rewritten as:

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = s(\epsilon_\theta(x_t,t),x_t,t) := f(t)x_t + \frac{g^2(t)}{2\sigma_t}\epsilon_\theta(x_t,t), \quad x_T \sim q(x_T). \tag{7}$$

Given an initial value x_T, we define the time steps $\{t_i\}_{i=0}^T$ to progressively decrease from $t_0 = T$ to $t_T = 0$. Let $\tilde{x}_{t_0} = x_T$ be the initial value. Using T steps of iteration, we compute the sequence $\{\tilde{x}_{t_i}\}_{i=0}^T$ to obtain the solution of this ODE. By integrating both sides of Eq. (7), we can obtain the exact solution of this sampling ODE.

$$\tilde{x}_{t_i} = \tilde{x}_{t_{i-1}} + \int_{t_{i-1}}^{t_i} s(\epsilon_\theta(x_t,t),x_t,t)\mathrm{d}t. \tag{8}$$

For any p-order ODE solver, Eq. (8) can be discretely represented as:

$$\tilde{x}_{t_{i-1}\to t_i} \approx \tilde{x}_{t_{i-1}} + \sum_{n=0}^{p-1} h(\epsilon_\theta(\tilde{x}_{\hat{t}_n},\hat{t}_n),\tilde{x}_{\hat{t}_n},\hat{t}_n)\cdot\Delta\hat{t}, \quad i \in [1,\dots,T], \tag{9}$$

where $\hat{t}_0 = t_{i-1}$, $\hat{t}_p = t_i$, and $\Delta\hat{t} = \hat{t}_{n+1} - \hat{t}_n$ denote the time step size. The function h represents the different solution methodologies applied by various p-order ODE solvers to the function s. For the Euler-Maruyama solver [38], h is the identity mapping of s. Further, we define

$\phi(Q, \tilde{x}_{t_{i-1}}, t_{i-1}, t_i) := \tilde{x}_{t_{i-1}} + \sum_{n=0}^{p-1} h(\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n), \tilde{x}_{\hat{t}_n}, \hat{t}_n) \cdot \Delta\hat{t}$. Here, ϕ is any p-order ODE solver, and buffer $Q = \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-1}, t_i\right)$, where $\hat{t}_0 = t_{i-1}$ and $\hat{t}_p = t_i$.

When using the ODE solver defined in Eq. (9) for sampling, the choice of $T = 1000$ leads to significant inefficiencies in DPMs. The study on DDIM [20] first revealed that by constructing a new forward sub-state sequence of length $M+1$ ($M \leq T$), $\{\tilde{x}_{t_i}\}_{i=0}^{M}$, from a subsequence of time steps $[0, \ldots, T]$ and reversing this sub-state sequence, it is possible to converge to the data distribution in fewer time steps. However, as illustrated in Fig. 2a, for ODE solvers, as the time step $\Delta t = t_i - t_{i-1}$ increases, the gradient direction changes slowly initially, but undergoes abrupt changes as $\Delta t \to T$. This phenomenon indicates that under minimal NFE (i.e., maximal time step size Δt) conditions, the discretization error in Eq. (9) is significantly amplified. Consequently, existing ODE solvers, when sampling under minimal NFE, must sacrifice sampling quality to gain speed, making it an extremely challenging task to reduce NFE to below 10 [21, 22]. Given this, we aim to develop an efficient timestep-skipping sampling algorithm, which reduces NFE while correcting discretization errors, thereby ensuring that sampling quality is not compromised, and may even be improved.

3.2 Sampling guided by past gradients

For any p-order timestep-skipping sampling algorithm for DPMs, the sampling process can be reformulated according to Eq. (9) as follows:

$$\tilde{x}_{t_i} \approx \phi(Q, \tilde{x}_{t_{i-1}}, t_{i-1}, t_i), \quad i \in [1, \ldots, M], \tag{10}$$

where buffer $Q = \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-1}, t_i\right)$ and $[1, \ldots, M]$ is an increasing subsequence of $[1, \ldots, T]$. As illustrated in Fig. 2a, when the time step size Δt (i.e., $t_i - t_{i-1}$) is not excessively large, the MSE of the noise network, defined as $\frac{1}{T-\Delta t}\sum_{t=0}^{T-\Delta t-1} \|\epsilon_\theta(x_t, t) - \epsilon_\theta(x_{t+\Delta t}, t+\Delta t)\|^2$, is remarkably similar. This phenomenon is especially pronounced in ODE-based sampling algorithms, such as DDIM [20] and DPM-Solver [21]. This observation suggests that there are many unnecessary time steps in ODE-based sampling methods during the complete sampling process (e.g., when $T = 1000$), which is one of the reasons these methods can generate samples in fewer steps. Based on this, we propose replacing the noise network of the current timestep with the output from a previous timestep to reduce unnecessary NFE without compromising the quality of the final generated samples. Initially, we store the output of the previous timestep's noise network in a *buffer* as follows:

$$Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-1}, t_i\right), \quad \text{where } \hat{t}_0 = t_{i-1}, \hat{t}_p = t_i. \tag{11}$$

Then, in the current timestep, we directly use the noise network output saved in the buffer from the previous timestep to replace the current timestep's noise network output, thereby updating the intermediate states to the next timestep, as detailed below:

$$\tilde{x}_{t_{i+1}} \approx \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+1}), \quad \text{where } Q = \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-1}, t_i\right). \tag{12}$$

By using this approach, we can effectively accelerate the sampling process, reduce unnecessary NFE, and ensure the quality of the samples is not affected. The convergence proof is in Appendix B.1.

3.3 Sampling guided by future gradients

As stated in Remark 1, considering the similarities between the sampling process of DPMs and SGD [33], we introduce a *foresight* update mechanism of Nesterov momentum, utilizing future gradient information as a "*springboard*" to assist the current intermediate state in achieving more efficient leapfrog updates. Specifically, for the intermediate state $\tilde{x}_{t_{i+1}}$ predicted using past gradients as discussed in Sec. 3.2, we first estimate the future gradient and *update* the current *buffer* as follows:

$$Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+1}, t_{i+2}\right), \quad \text{where } \hat{t}_0 = t_{i+1}, \hat{t}_p = t_{i+2}. \tag{13}$$

Subsequently, leveraging the concept of foresight updates, we predict a further future intermediate state $\tilde{x}_{t_{i+2}}$ using the current intermediate state $\tilde{x}_{t_i}$ along with the future gradient information corresponding to $\tilde{x}_{t_{i+1}}$, as shown below:

$$\tilde{x}_{t_{i+2}} \approx \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+2}), \quad \text{where } Q = \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+1}, t_{i+2}\right). \tag{14}$$

Furthermore, Zhou et al. [39] performed a Principal Component Analysis (PCA) on the sampling trajectories generated by ODE solvers for DPMs and discovered they almost lie in a two-dimensional plane embedded within a high-dimensional space. This implies that the *Mean Value Theorem* approximately holds during the sampling process using ODE solvers. Specifically, updating the current intermediate state $\tilde{x}_{t_i}$ at an optimal time point s with the corresponding gradient information, *ground truth* $\epsilon_\theta(\tilde{x}_{t_s}, t_s)$, results in the smallest update error, where s is between time points i and $i+2$. Further, we can reason that for any *first-order* ODE solver, under the same time step, the use of future gradient information $\epsilon_\theta(\tilde{x}_{t_{i+1}}, t_{i+1})$ from Eq. (13) to update the current intermediate state $\tilde{x}_{t_i}$ results in a smaller sampling error compared to using the gradient information at the current time point $\epsilon_\theta(\tilde{x}_{t_i}, t_i)$. A detailed proof is provided in Appendix B.2. However, for higher-order ODE solvers, the solving process implicitly utilizes future gradients as mentioned in Sec. 3.5, and the additional explicit introduction of future gradients increases sampling error. Therefore, when using higher-order ODE solvers as a baseline, the sampling process is accelerated by only using past gradients. It is only necessary to modify Eq. (14) to $\tilde{x}_{t_{i+2}} \approx \phi(Q, \tilde{x}_{t_{i+1}}, t_{i+1}, t_{i+2})$ while keeping Q constant. Ablation experiments can be found in Sec. 4.3.

3.4 PFDiff: sampling guided by past and future gradients

Combining Sec. 3.2 and Sec. 3.3, the intermediate state $\tilde{x}_{t_{i+1}}$ obtained through Eq. (12) is used to update the buffer Q in Eq. (13). In this way, we achieve our proposed efficient timestep-skipping algorithm, which we name PFDiff, as shown in Algorithm 1. For higher-order ODE solvers ($p > 1$), PFDiff only utilizes past gradient information, while for first-order ODE solvers ($p = 1$), it uses both past and future gradient information to predict further future intermediate states. Notably, during the iteration from intermediate state $\tilde{x}_{t_i}$ to $\tilde{x}_{t_{i+2}}$, we only perform a single batch computation (NFE = p) of the noise network in Eq. (13). Furthermore, we propose that in a single iteration process, $\tilde{x}_{t_{i+2}}$ in Eq. (14) can be modified to $\tilde{x}_{t_{i+(k+1)}}$, achieving a k-step skip to sample more distant future intermediate states. Additionally, when $k \neq 1$, the buffer Q, which acts as an intermediate "springboard" from Eq. (13), has various computational origins. This can be accomplished by modifying $\tilde{x}_{t_{i+1}}$ in Eq. (12) to $\tilde{x}_{t_{i+l}}$. We collectively refer to this multi-step skipping and different "springboard" selection strategy as PFDiff-k_l ($l \leq k$). Further algorithmic details can be found in Appendix C. Finally, through the comparison of sampling trajectories between PFDiff-1 and a first-order ODE sampler, as shown in Fig. 2b, PFDiff-1 showcases its capability to correct the sampling trajectory of the first-order ODE sampler while reducing the NFE.

Proposition 3.1. *For any given DPM first-order ODE solver ϕ, the PFDiff-k_l algorithm can describe the sampling process within an iteration cycle through the following formula:*

$$\tilde{x}_{t_{i+(k+1)}} \approx \phi(\epsilon_\theta(\phi(\epsilon_\theta(\tilde{x}_{t_{i-(k-l+1)}}, t_{i-(k-l+1)}), \tilde{x}_{t_i}, t_i, t_{i+l}), t_{i+l}), \tilde{x}_{t_i}, t_i, t_{i+(k+1)}), \tag{15}$$

Algorithm 1 PFDiff-1

Require: initial value x_T, NFE N, model ϵ_θ, any p-order solver ϕ
Define time steps $\{t_i\}_{i=0}^{M}$ with $M = 2N - 1p$
$\tilde{x}_{t_0} \leftarrow x_T$
$Q \xleftarrow{\text{buffer}} \left(\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\}_{n=0}^{p-1}, t_0, t_1\right)$, where $\hat{t}_0 = t_0, \hat{t}_p = t_1$ ▷ Initialize buffer
$\tilde{x}_{t_1} = \phi(Q, \tilde{x}_{t_0}, t_0, t_1)$
for $i \leftarrow 1$ to $\frac{M}{p} - 2$ **do**
  if $(i-1) \mod 2 = 0$ **then**
    $\tilde{x}_{t_{i+1}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+1})$ ▷ Updating guided by past gradients
    $Q \xleftarrow{\text{buffer}} \left(\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\}_{n=0}^{p-1}, t_{i+1}, t_{i+2}\right)$ ▷ Update buffer (overwrite)
    if $p = 1$ **then**
      $\tilde{x}_{t_{i+2}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+2})$ ▷ Anticipatory updating guided by future gradients
    else if $p > 1$ **then**
      $\tilde{x}_{t_{i+2}} = \phi(Q, \tilde{x}_{t_{i+1}}, t_{i+1}, t_{i+2})$ ▷ The higher-order solver uses only past gradients
    end if
  end if
end for
return $\tilde{x}_{t_M}$

---

where the value of $\epsilon_\theta(\tilde{x}_{t_{i-(k-l+1)}}, t_{i-(k-l+1)})$ can be directly obtained from the buffer Q, without the need for additional computations. The iterative process defined by Eq. (15) ensures that the sampling outcomes converge to the data distribution consistent with the solver ϕ, while effectively correcting errors in the sampling process (Proof in Appendix B).

It is noteworthy that, although the PFDiff is conceptually orthogonal to the SDE/ODE solvers of DPMs, even when the time size Δt is relatively small, the MSE of the noise network in the SDE solver exhibits significant differences, as shown in Fig. 2a. Consequently, PFDiff shows marked improvements on the ODE solver, and our experiments are almost exclusively based on ODE solvers, with exploratory experiments on SDE solvers referred to Sec. 4.1.

3.5 Connection with other samplers

Relationship with p-order solver [21, 22, 27]. According to Eq. (10), a single iteration of the p-order solver can be represented as:

$$\tilde{x}_{t_{i+1}} \approx \mathrm{Solver-p}\left(\left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_i, t_{i+1}\right), \tilde{x}_{t_i}, t_i, t_{i+1}\right), \quad i \in [0, \dots, M-1]. \tag{16}$$

A single iteration of the p-order solver uses p NFE to predict the next intermediate state. The intermediate step gradients obtained during this process can be considered as an approximation of future gradients. This approximation is implicitly contained within the sampling guided by future gradients that we propose. Furthermore, as shown in Eq. (15), a single iteration update of PFDiff based on a first-order solver can be seen as using a 2-order solver with only one NFE.

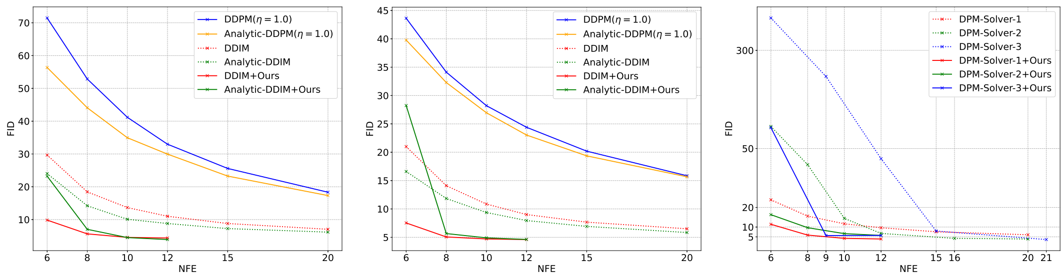
4 Experiments

In this section, we validate the effectiveness of PFDiff as an *orthogonal* and *training-free* sampler through a series of extensive experiments. This sampler can be integrated with any order of ODE solvers, thereby significantly enhancing the sampling efficiency of various types of pre-trained DPMs. To systematically showcase the performance of PFDiff, we categorize the pre-trained DPMs into two main types: conditional and unconditional. Unconditional DPMs are further subdivided into discrete and continuous, while conditional DPMs are subdivided into classifier guidance and classifier-free guidance. In choosing ODE solvers, we utilized the widely recognized first-order DDIM [20], Analytic-DDIM [23], and the higher-order DPM-Solver [21] as baselines. For each experiment, we use the Fréchet Inception Distance (FID$\downarrow$) [40] as the primary evaluation metric, and provide the experimental results of the Inception Score (IS$\uparrow$) [41] in the Appendix D.7 for reference. Lastly, apart from the ablation studies on parameters k and l discussed in Sec. 4.3, we showcase the optimal results of PFDiff-k_l (where $k = 1, 2, 3$ and $l \leq k$) across six configurations as a performance demonstration of PFDiff. As described in Appendix C, this does not increase the computational burden in practical applications. All experiments were conducted on an NVIDIA RTX 3090 GPU.

4.1 Unconditional sampling

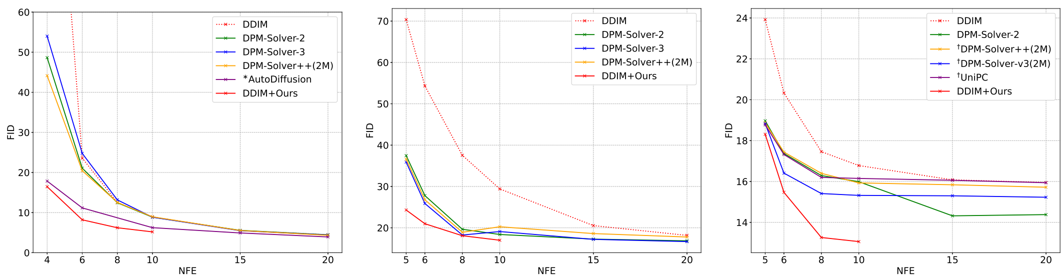
For unconditional DPMs, we selected discrete DDPM [2] and DDIM [20], as well as pre-trained models from continuous ScoreSDE [4], to assess the effectiveness of PFDiff. For these pre-trained models, all experiments sampled 50k instances to compute evaluation metrics.

For unconditional discrete DPMs, we first select first-order ODE solvers DDIM [20] and Analytic-DDIM [23] as baselines, while implementing SDE-based DDPM [2] and Analytic-DDPM [23] methods for comparison, where $\eta = 1.0$ is from $\bar{\sigma}_t$ in Eq. (6). We conduct experiments on the CIFAR10 [42] and CelebA 64x64 [43] datasets using the quadratic time steps employed by DDIM. By varying the NFE from 6 to 20, the evaluation metric FID$\downarrow$ is shown in Figs. 3a and 3b. Additionally, experiments with uniform time steps are conducted on the CelebA 64x64, LSUN-bedroom 256x256 [44], and LSUN-church 256x256 [44] datasets, with more results available in Appendix D.2. Our experimental results demonstrate that PFDiff, based on pre-trained models of discrete unconditional DPMs, significantly improves the sampling efficiency of DDIM and Analytic-DDIM samplers across multiple datasets. For instance, on the CIFAR10 dataset, PFDiff combined with DDIM achieves a FID of 4.10 with only 15 NFE, comparable to DDIM's performance of 4.04 FID with 1000 NFE. This is something other time-step skipping algorithms [23, 28] that sacrifice sampling quality for speed



(a) CIFAR10 (Discrete) (b) CelebA 64x64 (Discrete) (c) CIFAR10 (Continuous)

Figure 3: Unconditional sampling results. We report the FID↓ for different methods by varying the number of function evaluations (NFE), evaluated on 50k samples.



(a) ImageNet 64x64 (Guided-Diffusion) (Classifier Guidance, $s = 1.0$)

(b) MS-COCO2014 (Stable-Diffusion) (Classifier-Free Guidance, $s = 1.5$)

(c) MS-COCO2014 (Stable-Diffusion) (Classifier-Free Guidance, $s = 7.5$)

Figure 4: Conditional sampling results. We report the FID↓ for different methods by varying the NFE. Evaluated: ImageNet 64x64 with 50k, others with 10k samples. *AutoDiffusion [26] method requires additional search costs. †We borrow the results reported in DPM-Solver-v3 [27] directly.

cannot achieve. Furthermore, in Appendix D.2, by varying η from 1.0 to 0.0 in Eq. (6) to control the scale of noise introduced by SDE, we observe that as η decreases (reducing noise introduction), the performance of PFDiff gradually improves. This once again validates our assumption proposed in Sec. 3.2, based on Fig. 2a, that there is a significant similarity in the model's outputs at the time step size that is not excessively large for the existing ODE solvers.

For unconditional continuous DPMs, we choose the DPM-Solver-1, -2 and -3 [21] as the baseline to verify the effectiveness of PFDiff as an orthogonal timestep-skipping algorithm on the first and higher-order ODE solvers. We conducted experiments on the CIFAR10 [42] using quadratic time steps, varying the NFE. The experimental results using FID↓ as the evaluation metric are shown in Fig. 3c. More experimental details can be found in Appendix D.3. We observe that PFDiff consistently improves the sampling performance over the baseline with fewer NFE settings, particularly in cases where higher-order ODE solvers fail to converge with a small NFE (below 10) [21].

4.2 Conditional sampling

For conditional DPMs, we selected the pre-trained models of the widely recognized classifier guidance paradigm, ADM-G [5], and the classifier-free guidance paradigm, Stable-Diffusion [9], to validate the effectiveness of PFDiff. We employed uniform time steps setting and the DDIM [20] ODE solver as a baseline across all datasets. Evaluation metrics were computed by sampling 50k samples on the ImageNet 64x64 [32] dataset for ADM-G and 10k samples on other datasets, including ImageNet 256x256 [32] in ADM-G and MS-COCO2014 [31] in Stable-Diffusion.

For conditional DPMs employing the classifier guidance paradigm, we conducted experiments on the ImageNet 64x64 dataset [32] with a guidance scale (s) set to 1.0. For comparison, we implemented DPM-Solver-2 and -3 [21], and DPM-Solver++(2M) [22], which exhibit the best performance on conditional DPMs. Additionally, we introduced the AutoDiffusion method [26] using DDIM as a baseline for comparison, noting that this method incurs additional search costs. We compared FID↓ scores by varying the NFE as depicted in Fig. 4a, with corresponding visual comparisons shown in Fig. 1b. We observed that PFDiff reduced the FID from 138.81 with 4 NFE in DDIM to 16.46, achieving an 88.14% improvement in quality. The visual results in Fig. 1b further demonstrate that, at the same NFE setting, PFDiff achieves higher-quality sampling. Furthermore, we evaluated PFDiff's sampling performance based on DDIM on the large-scale ImageNet 256x256 dataset [32]. Detailed results are provided in Appendix D.4.

For conditional, classifier-free guidance paradigms of DPMs, we employed the `sd-v1-4` checkpoint and computed the FID↓ scores on the validation set of MS-COCO2014 [31]. We conducted experiments with a guidance scale (s) set to 7.5 and 1.5. For comparison, we implemented DPM-Solver-2 and -3 [21], and DPM-Solver++(2M) [22] methods. At $s = 7.5$, we introduced the state-of-the-art method reported in DPM-Solver-v3 [27] for comparison, along with DPM-Solver++(2M) [22], UniPC [29], and DPM-Solver-v3(2M) [27]. The FID↓ metrics by varying the NFE are presented in Figs. 4b and 4c, with additional visual results illustrated in Fig. 1a. We observed that PFDiff, solely based on DDIM, achieved state-of-the-art results during the sampling process of Stable-Diffusion, thus demonstrating the efficacy of PFDiff. Further experimental details can be found in Appendix D.5.

4.3 Ablation study

We conducted ablation experiments on the six different algorithm configurations of PFDiff mentioned in Appendix C, with $k = 1, 2, 3$ ($l \leq k$). Specifically, we evaluated the FID↓ scores on the unconditional and conditional pre-trained DPMs [2, 4, 5, 9] by varying the NFE. Detailed experimental setups and results can be found in Appendix D.6.1. The experimental results indicate that for various pre-trained DPMs, the choice of parameters k and l is not critical, as most combinations of k and l within PFDiff can enhance the sampling efficiency over the baseline. Moreover, with $k = 1$ fixed, PFDiff-1 can significantly improve the baseline's sampling quality within the range of 8∼20 NFE. For even better sampling quality, one can sample a small subset of examples (e.g., 5k) to compute evaluation metrics or directly conduct visual analysis, easily identifying the most effective k and l combinations.

To validate the PFDiff algorithm as mentioned in Sec. 3.3, which necessitates the joint guidance of past and future gradients for first-order ODE solvers, and only past gradients for higher-order ODE solvers, offering a more effective means of accelerating baseline sampling. This study employs the first-order ODE solver DDIM [20] as the baseline, isolating the effects of both past and future gradients, and uses the higher-order ODE solver DPM-Solver [21] as the baseline, removing the influence of future gradients for ablation experiments. Specific experimental configurations and results are shown in Appendix D.6.2. The results indicate that, as described by the PFDiff algorithm in Sec. 3.3, it is possible to further enhance the sampling efficiency of ODE solvers of any order.

5 Conclusion

In this paper, based on the recognition that the ODE solvers of DPMs exhibit significant similarity in model outputs when the time step size is not excessively large, and with the aid of a foresight update mechanism, we propose PFDiff, a novel method that leverages the gradient guidance from both past and future to rapidly update the current intermediate state. This approach effectively reduces the unnecessary number of function evaluations (NFE) in the ODE solvers and significantly corrects the errors of first-order ODE solvers during the sampling process. Extensive experiments demonstrate the orthogonality and efficacy of PFDiff on both unconditional and conditional pre-trained DPMs, especially in conditional pre-trained DPMs where PFDiff outperforms previous state-of-the-art training-free sampling methods.

Limitations and broader impact Although PFDiff can effectively accelerate the sampling speed of existing ODE solvers, it still lags behind the sampling speed of training-based acceleration methods and one-step generation paradigms such as GANs. Moreover, there is no universal setting for the optimal combination of parameters k and l in PFDiff; adjustments are required according to different pre-trained DPMs and NFE. It is noteworthy that PFDiff may be utilized to accelerate the generation of malicious content, thereby having a detrimental impact on society.

References

[1] 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.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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

[4] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

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

[6] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.

[7] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[8] Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M Alabdulmohsin, et al. Patch n'pack: Navit, a vision transformer for any aspect ratio and resolution. *Advances in Neural Information Processing Systems*, 36, 2024.

[9] 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.

[10] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023.

[11] Kaitao Song, Yichong Leng, Xu Tan, Yicheng Zou, Tao Qin, and Dongsheng Li. Transcormer: Transformer for sentence scoring with sliding language modeling. *Advances in Neural Information Processing Systems*, 35:11160–11174, 2022.

[12] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[13] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

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

[16] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[17] 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.

[18] Zhendong Wang, Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Diffusion-gan: Training gans with diffusion. *arXiv preprint arXiv:2206.02262*, 2022.

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

[20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[21] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[22] 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.

[23] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-dpm: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. *arXiv preprint arXiv:2201.06503*, 2022.

[24] Fan Bao, Chongxuan Li, Jiacheng Sun, Jun Zhu, and Bo Zhang. Estimating the optimal covariance with imperfect mean in diffusion probabilistic models. *arXiv preprint arXiv:2206.07309*, 2022.

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

[26] Lijiang Li, Huixia Li, Xiawu Zheng, Jie Wu, Xuefeng Xiao, Rui Wang, Min Zheng, Xin Pan, Fei Chao, and Rongrong Ji. Autodiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7105–7114, 2023.

[27] Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *arXiv preprint arXiv:2310.13268*, 2023.

[28] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. *arXiv preprint arXiv:2312.00858*, 2023.

[29] Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[31] 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.

[32] 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.

[33] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[34] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k** 2). *Doklady Akademii Nauk SSSR*, 269(3):543, 1983.

[35] Bernt Øksendal and Bernt Øksendal. *Stochastic differential equations*. Springer, 2003.

[36] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[37] Kai Wang, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

[38] Peter E Kloeden, Eckhard Platen, Peter E Kloeden, and Eckhard Platen. *Stochastic differential equations*. Springer, 1992.

[39] Zhenyu Zhou, Defang Chen, Can Wang, and Chun Chen. Fast ode-based sampling for diffusion models in around 5 steps. *arXiv preprint arXiv:2312.00094*, 2023.

[40] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[41] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

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

[43] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[44] 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.

[45] Elliott Ward Cheney, EW Cheney, and W Cheney. *Analysis for applied mathematics*, volume 1. Springer, 2001.

A Related work

While the solvers for Diffusion Probabilistic Models (DPMs) are categorized into two types, SDE and ODE, most current accelerated sampling techniques are based on ODE solvers due to the observation that the stochastic noise introduced by SDE solvers hampers rapid convergence. ODE-based solvers are further divided into training-based methods [16–19] and training-free samplers [20–30]. Training-based methods can notably reduce the number of sampling steps required for DPMs. An example of such a method is the knowledge distillation algorithm proposed by Song et al. [19], which achieves one-step sampling for DPMs. This sampling speed is comparable to that of GANs [14] and VAEs [15]. However, these methods often entail significant additional costs for distillation training. This requirement poses a challenge when applying them to large pre-trained DPM models. Therefore, our work primarily focuses on training-free, ODE-based accelerated sampling strategies.

Training-free accelerated sampling techniques based on ODE can generally be applied in a plug-and-play manner, adapting to existing pre-trained DPMs. These methods can be categorized based on the order of the ODE solver—that is, the NFE required per sampling iteration—into first-order [20, 23–25] and higher-order [21, 22, 27, 29, 36]. Typically, higher-order ODE solvers tend to sample at a faster rate, but may fail to converge when the NFE is low (below 10), sometimes performing even worse than first-order ODE solvers. In addition, there are orthogonal techniques for accelerated sampling. For instance, Li et al. [26] build upon existing ODE solvers and use search algorithms to find optimal sampling sub-sequences and model structures to further speed up the sampling process; Ma et al. [28] observe that the low-level features of noise networks at adjacent time steps exhibit similarities, and they use caching techniques to substitute some of the network's low-level features, thereby further reducing the number of required time steps.

The algorithm we propose belongs to the class of training-free and orthogonal accelerated sampling techniques, capable of further accelerating the sampling process on the basis of existing first-order and higher-order ODE solvers. Compared to the aforementioned orthogonal sampling techniques, even though the skipping strategy proposed by Ma et al. [28] effectively accelerates the sampling process, it may do so at the cost of reduced sampling quality, making it challenging to reduce the NFE below 50. Although Li et al. [26] can identify more optimal subsampling sequences and model structures, this implies higher search costs. In contrast, our proposed orthogonal sampling algorithm is more efficient in skipping time steps. First, our skipping strategy does not require extensive search costs. Second, we can correct the sampling errors of first-order ODE solvers while reducing the number of sampling steps required by existing ODE solvers, achieving more efficient accelerated sampling.

B Proof of convergence and error correction for PFDiff

In this section, we prove the convergence of PFDiff and elaborate on how it theoretically corrects first-order ODE solver errors. To delve deeper into PFDiff, we propose the following assumptions:

Assumption B.1. *When the time step size $\Delta t = t_i - t_{i-(k-l+1)}$ is not excessively large, the output estimates of the noise network based on the p-order ODE solver at different time steps are approximately the same, that is,* $\left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_i, t_{i+l}\right) \approx \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-(k-l+1)}, t_i\right)$.

Assumption B.2. *For the integral from time step t_i to $t_{i+(k+1)}$, $\int_{t_i}^{t_{i+(k+1)}} s(\epsilon_\theta(x_t, t), x_t, t)\mathrm{d}t$, there exist intermediate time steps $t_{\tilde{s}}, t_s \in (t_i, t_{i+(k+1)})$ such that $\int_{t_i}^{t_{i+(k+1)}} s(\epsilon_\theta(x_t, t), x_t, t)\mathrm{d}t = s(\epsilon_\theta(x_{t_{\tilde{s}}}, t_{\tilde{s}}), x_{t_{\tilde{s}}}, t_{\tilde{s}}) \cdot (t_{i+(k+1)} - t_i) = h(\epsilon_\theta(x_{t_s}, t_s), x_{t_s}, t_s) \cdot (t_{i+(k+1)} - t_i)$ holds, where the definition of the function h remains consistent with Sec. 3.1.*

The first assumption is based on the observation in Fig. 2a that when Δt is not excessively large, the MSE of the noise network remains almost unchanged across different time steps. The second assumption is based on the *Mean Value Theorem* for Integrals, which states that if $f(x)$ is a continuous real-valued function on a closed interval $[a, b]$, then there exists at least one point $c \in [a, b]$ such that $\int_a^b f(x)\mathrm{d}x = f(c)(b-a)$ holds. It is important to note that the Mean Value Theorem for Integrals originally applies to real-valued functions and does not directly apply to vector-valued functions [45]. However, the study by Zhou et al. [39] using Principal Component Analysis (PCA) on the trajectories of the ODE solvers for DPMs demonstrates that these trajectories are primarily distributed on a two-dimensional plane, which allows the Mean Value Theorem for Integrals to approximately hold under Assumption B.2.

B.1 Proof of convergence for sampling guided by past gradients

Starting from Eq. (8), we consider an iteration process of a p-order ODE solver from $\tilde{x}_{t_i}$ to $\tilde{x}_{t_{i+l}}$, where l is the "*springboard*" choice determined by PFDiff-k_l. This iterative process can be expressed as:

$$\tilde{x}_{t_{i+l}} = \tilde{x}_{t_i} + \int_{t_i}^{t_{i+l}} s(\epsilon_\theta(x_t, t), x_t, t)\mathrm{d}t. \tag{B.1}$$

Discretizing Eq. (B.1) yields:

$$\tilde{x}_{t_i \to t_{i+l}} \approx \tilde{x}_{t_i} + \sum_{n=0}^{p-1} h(\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n), \tilde{x}_{\hat{t}_n}, \hat{t}_n) \cdot (\hat{t}_{n+1} - \hat{t}_n), \tag{B.2}$$

where $\hat{t}_0 = t_i$ and $\hat{t}_p = t_{i+l}$. Consistent with Sec. 3.1, the function h represents the different solution methodologies applied by various p-order ODE solvers to the function s. To accelerate sampler convergence and reduce unnecessary evaluations of the NFE, we adopt Assumption B.1, namely guiding the sampling of the current intermediate state by utilizing past gradient information. Specifically, we approximate that $\left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_i, t_{i+l}\right) \approx \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-(k-l+1)}, t_i\right)$, where k represents the number of steps skipped in one iteration by PFDiff-k_l. Eq. (B.2) can be rewritten as:

$$\begin{aligned}
\tilde{x}_{t_i \to t_{i+l}} &\approx \tilde{x}_{t_i} + \sum_{n=i}^{i+l-1} h(\epsilon_\theta(\tilde{x}_{t_n}, t_n), \tilde{x}_{t_n}, t_n) \cdot (t_{n+1} - t_n) \\
&\approx \tilde{x}_{t_i} + \sum_{n=i-(k-l+1)}^{i-1} h(\epsilon_\theta(\tilde{x}_{t_n}, t_n), \tilde{x}_{t_n}, t_n) \cdot (t_{n+1} - t_n) \\
&= \phi(\left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-(k-l+1)}, t_i\right), \tilde{x}_{t_i}, t_i, t_{i+l}),
\end{aligned} \tag{B.3}$$

where ϕ is any p-order ODE solver. Eq. (B.3) demonstrates that under Assumption B.1, for any p-order ODE solver ϕ, PFDiff-k_l utilizes past gradient information as a substitute for current gradient information to update the current intermediate state. This method not only reduces the NFE but also approximates the solution of $\tilde{x}_{t_{i+l}}$, ensuring convergence to the data distribution corresponding to the solver ϕ. It is important to note that the sampling process described in Eq. (B.3) relies solely on past gradient information and does not estimate the output of the noise network based on the current intermediate state.

In particular, within Proposition 3.1 for any first-order ($p = 1$) ODE solver ϕ, according to Eq. (B.3), we can approximate $\tilde{x}_{t_{i+l}} \approx \phi(\epsilon_\theta(\tilde{x}_{t_{i-(k-l+1)}}, t_{i-(k-l+1)}), \tilde{x}_{t_i}, t_i, t_{i+l})$. Thus, Eq. (15) can be rewritten as:

$$\tilde{x}_{t_{i+(k+1)}} \approx \phi(\epsilon_\theta(\tilde{x}_{t_{i+l}}, t_{i+l}), \tilde{x}_{t_i}, t_i, t_{i+(k+1)}). \tag{B.4}$$

For any first-order ODE solver ϕ, Eq. (B.3) and (B.4) utilize the gradient information from both past and future to constitute a complete sampling iteration process for PFDiff-k_l. Eq. (B.4) indicates that under the Assumption B.1 and upon the convergence of Eq. (B.4), PFDiff-k_l is guaranteed to converge to the data distribution corresponding to the sampler ϕ for any first-order ODE solver.

B.2 Error correction and proof of convergence of Proposition 3.1

Based on Eq. (8), we consider an iteration process of a first-order ($p = 1$) ODE solver from $\tilde{x}_{t_i}$ to $\tilde{x}_{t_{i+(k+1)}}$, which can be expressed as:

$$\begin{aligned}
\tilde{x}_{t_{i+(k+1)}} &= \tilde{x}_{t_i} + \int_{t_i}^{t_{i+(k+1)}} s(\epsilon_\theta(x_t, t), x_t, t)\mathrm{d}t \\
&\approx \tilde{x}_{t_i} + h(\epsilon_\theta(\tilde{x}_{t_i}, t_i), \tilde{x}_{t_i}, t_i) \cdot (t_{i+(k+1)} - t_i) \\
&= \phi(\epsilon_\theta(\tilde{x}_{t_i}, t_i), \tilde{x}_{t_i}, t_i, t_{i+(k+1)}),
\end{aligned} \tag{B.5}$$

where the second line of Eq. (B.5) is obtained by discretizing the first line with an existing first-order ODE solver ($p = 1$), and the definition of ϕ and h are consistent with Appendix B.1. It is well-known that the discretization method used in Eq. (B.5) restricts the sampling step size $\Delta t = t_{i+(k+1)} - t_i$ of the first-order ODE solver. A too-large step size will cause the first-order ODE solver to not converge. This indicates that although Assumption B.2 points out that the sampling trajectory of the first-order ODE solver lies on a two-dimensional plane, this trajectory is not a straight line (if it were a straight line, a larger sampling step size could be used). Therefore, using $\epsilon_\theta(\tilde{x}_{t_i}, t_i)$ for discretized sampling in Eq. (B.5) introduces a significant sampling error, as shown by the first-order ODE sampling trajectory in Fig. 2b. To reduce the first-order ODE solver sampling error, we have revised Eq. (B.5) based on Assumption B.2, as follows:

$$\begin{aligned}
\tilde{x}_{t_{i+(k+1)}} &= \tilde{x}_{t_i} + \int_{t_i}^{t_{i+(k+1)}} s(\epsilon_\theta(x_t, t), x_t, t)\mathrm{d}t \\
&= \tilde{x}_{t_i} + s(\epsilon_\theta(\tilde{x}_{t_{\tilde{s}}}, t_{\tilde{s}}), \tilde{x}_{t_{\tilde{s}}}, t_{\tilde{s}}) \cdot (t_{i+(k+1)} - t_i) \\
&= \tilde{x}_{t_i} + h(\epsilon_\theta(\tilde{x}_{t_s}, t_s), \tilde{x}_{t_s}, t_s) \cdot (t_{i+(k+1)} - t_i) \\
&= \phi(\epsilon_\theta(\tilde{x}_{t_s}, t_s), \tilde{x}_{t_i}, t_i, t_{i+(k+1)}) \\
&\approx \phi(\epsilon_\theta(\tilde{x}_{t_{i+l}}, t_{i+l}), \tilde{x}_{t_i}, t_i, t_{i+(k+1)}),
\end{aligned} \tag{B.6}$$

where k and l are determined by the selected PFDiff-k_l and the second and third lines are obtained based on Assumption B.2. Combining Eq. (B.6) and Eq. (B.3) leads to the complete Eq. (15), thereby completing the convergence proof of Proposition 3.1. Moreover, t_s falls within the interval $[t_i, t_{i+(k+1)}]$, and since the sampling trajectory of the first-order ODE solver is not a straight line, generally $t_s \neq t_i$ and $t_s \neq t_{i+(k+1)}$. The interval $[t_i, t_{i+(k+1)}]$ is amended to $(t_i, t_{i+(k+1)})$. By adopting the *foresight* update mechanism of the Nesterov momentum [34], and guiding the current intermediate state sampling with future gradient information, we replace $\epsilon_\theta(\tilde{x}_{t_s}, t_s)$ with $\epsilon_\theta(\tilde{x}_{t_{i+l}}, t_{i+l})$. According to the definition of PFDiff-k_l, t_{i+l} also lies within the interval $(t_i, t_{i+(k+1)})$, and for different combinations of k and l, this means searching and approximating the *ground truth* t_s within the interval $(t_i, t_{i+(k+1)})$. Among the six different versions of PFDiff-k_l defined in Appendix C, we believe that the optimal t_s has been approximated. Compared to the direct discretization of $\epsilon_\theta(x_t, t)$ in Eq. (B.5), we corrected the sampling error of the first-order ODE solver and proved its convergence by guiding sampling based on the future gradient information $\epsilon_\theta(\tilde{x}_{t_{i+l}}, t_{i+l})$ under Assumption B.2, as shown in the sampling trajectory of PFDiff-1 in Fig. 2b. Together with this section and Appendix B.1, this completes the error correction and convergence proof of Proposition 3.1.

C Algorithms of PFDiffs

As described in Sec. 3.4, during a single iteration, we can leverage the *foresight* update mechanism to skip to a more distant future. Specifically, we modify Eq. (14) to $\tilde{x}_{t_{i+(k+1)}} \approx \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+(k+1)})$ to achieve a k-step skip. We refer to this method as PFDiff-k. Additionally, when $k \neq 1$, the computation of the buffer Q, originating from Eq. (13), presents different selection choices. We modify Eq. (12) to $\tilde{x}_{t_{i+l}} \approx \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+l}), l \leq k$ to denote different "*springboard*" choices with the parameter l. This strategy of multi-step skips and varying "springboard" choices is collectively termed as PFDiff-k_l ($l \leq k$). Consequently, based on modifications to parameters k and l in Eq. (12) and Eq. (14), Eq. (13) is updated to $Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+l}, t_{i+(k+1)}\right)$, and Eq. (11) is updated to $Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i-(k-l+1)}, t_i\right)$. When $k = 1$, since $l \leq k$, then $l = 1$, and PFDiff-k_l is the same as PFDiff-1, as shown in Algorithm 1 in Sec. 3.4. When $k = 2$, l can be either 1 or 2, forming Algorithms PFDiff-2_1 and PFDiff-2_2, as shown in Algorithm 2. Furthermore, when $k = 3$, this forms three different versions of PFDiff-3, as shown in Algorithm 3. In this study, we utilize the optimal results from the six configurations of PFDiff-k_l ($k = 1, 2, 3$ ($l \leq k$)) to demonstrate the performance of PFDiff. As described in Appendix B.2, this is essentially an approximation of the *ground truth* t_s. Through these six different algorithm configurations, we approximately search for the optimal t_s. It is important to note that despite using six different algorithm configurations, this does not increase the computational burden in practical applications. This is because, by visual analysis of a small number of generated images or computing specific evaluation metrics, one can effectively select the algorithm configuration with the best performance. Moreover, even without any selection, directly using the PFDiff-1 configuration can achieve significant performance improvements on top of existing ODE solvers, as shown in the ablation study results in Sec. 4.3.

Algorithm 2 PFDiff-2

Require: initial value x_T, NFE N, model ϵ_θ, any p-order solver ϕ, skip type l

Define time steps $\{t_i\}_{i=0}^{M}$ with $M = 3N - 2p$
$\tilde{x}_{t_0} \leftarrow x_T$
$Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_0, t_1\right)$, where $\hat{t}_0 = t_0, \hat{t}_p = t_1$ ▷ Initialize buffer
$\tilde{x}_{t_1} = \phi(Q, \tilde{x}_{t_0}, t_0, t_1)$
for $i \leftarrow 1$ to $\frac{M}{p} - 3$ **do**
  if $(i-1) \mod 3 = 0$ **then**
    if $l = 1$ **then** ▷ PFDiff-2_1
      $\tilde{x}_{t_{i+1}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+1})$ ▷ Updating guided by past gradients
      $Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+1}, t_{i+3}\right)$ ▷ Update buffer (overwrite)
    else if $l = 2$ **then** ▷ PFDiff-2_2
      $\tilde{x}_{t_{i+2}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+2})$ ▷ Updating guided by past gradients
      $Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+2}, t_{i+3}\right)$ ▷ Update buffer (overwrite)
    end if
    if $p = 1$ **then**
      $\tilde{x}_{t_{i+3}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+3})$ ▷ Anticipatory updating guided by future gradients
    else if $p > 1$ **then**
      $\tilde{x}_{t_{i+3}} = \phi(Q, \tilde{x}_{t_{i+l}}, t_{i+l}, t_{i+3})$ ▷ The higher-order solver uses only past gradients
    end if
  end if
end for
return $\tilde{x}_{t_M}$

Algorithm 3 PFDiff-3

Require: initial value x_T, NFE N, model ϵ_θ, any p-order solver ϕ, skip type l

Define time steps $\{t_i\}_{i=0}^{M}$ with $M = 4N - 3p$
$\tilde{x}_{t_0} \leftarrow x_T$
$Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_0, t_1\right)$, where $\hat{t}_0 = t_0, \hat{t}_p = t_1$ ▷ Initialize buffer
$\tilde{x}_{t_1} = \phi(Q, \tilde{x}_{t_0}, t_0, t_1)$
for $i \leftarrow 1$ to $\frac{M}{p} - 4$ **do**
  if $(i-1) \mod 4 = 0$ **then**
    $\tilde{x}_{t_{i+4}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+l})$ ▷ Updating guided by past gradients
    $Q \xleftarrow{\text{buffer}} \left(\left\{\epsilon_\theta(\tilde{x}_{\hat{t}_n}, \hat{t}_n)\right\}_{n=0}^{p-1}, t_{i+l}, t_{i+4}\right)$ ▷ Update buffer (overwrite)
    if $p = 1$ **then**
      $\tilde{x}_{t_{i+4}} = \phi(Q, \tilde{x}_{t_i}, t_i, t_{i+4})$ ▷ Anticipatory updating guided by future gradients
    else if $p > 1$ **then**
      $\tilde{x}_{t_{i+4}} = \phi(Q, \tilde{x}_{t_{i+l}}, t_{i+l}, t_{i+4})$ ▷ The higher-order solver uses only past gradients
    end if
  end if
end for
return $\tilde{x}_{t_M}$

D Additional experiment results

In this section, we provide further supplements to the experiments on both unconditional and conditional pre-trained Diffusion Probabilistic Models (DPMs) as mentioned in Sec. 4. Through these additional supplementary experiments, we more fully validate the effectiveness of PFDiff as an orthogonal and training-free sampler. Unless otherwise stated, the selection of pre-trained DPMs, choice of baselines, algorithm configurations, GPU utilization, and other related aspects in this section are consistent with those described in Sec. 4.

D.1 License

In this section, we list the used datasets, codes, and their licenses in Table 1.

Table 1: The used datasets, codes, and their licenses.

Name	URL	License
CIFAR10 [42]	https://www.cs.toronto.edu/~kriz/cifar.html	\
CelebA 64x64 [43]	https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html	\
LSUN-Bedroom [44]	https://www.yf.io/p/lsun	\
LSUN-Church [44]	https://www.yf.io/p/lsun	\
ImageNet [32]	https://image-net.org/	\
MS-COCO2014 [31]	https://cocodataset.org/	CC BY 4.0
ScoreSDE [4]	https://github.com/yang-song/score_sde_pytorch	Apache-2.0
DDIM [20]	https://github.com/ermongroup/ddim/tree/main	MIT
Analytic-DPM [23]	https://github.com/baofff/Analytic-DPM	\
DPM-Solver [21]	https://github.com/LuChengTHU/dpm-solver	MIT
DPM-Solver++ [22]	https://github.com/LuChengTHU/dpm-solver	MIT
Guided-Diffusion [5]	https://github.com/openai/guided-diffusion	MIT
Stable-Diffusion [9]	https://github.com/CompVis/stable-diffusion	CreativeML Open RAIL-M

D.2 Additional results for unconditional discrete-time sampling

In this section, we report on experiments with unconditional, discrete DPMs on the CIFAR10 [42] and CelebA 64x64 [43] datasets using quadratic time steps. The FID↓ scores for the PFDiff algorithm are reported for changes in the number of function evaluations (NFE) from 4 to 20. Additionally, we present FID scores on the CelebA 64x64 [43], LSUN-bedroom 256x256 [44], and LSUN-church 256x256 [44] datasets, utilizing uniform time steps. The experimental results are summarized in Table 2. Results indicate that using DDIM [20] as the baseline, our method (PFDiff) nearly achieved significant performance improvements across all datasets and NFE settings. Notably, PFDiff facilitates rapid convergence of pre-trained DPMs to the data distribution with NFE settings below 10, validating its effectiveness on discrete pre-trained DPMs and the first-order ODE solver DDIM. It is important to note that on the CIFAR10 and CelebA 64x64 datasets, we have included the FID scores of Analytic-DDIM [23], which serves as another baseline. Analytic-DDIM modifies the variance in DDIM and introduces some random noise. With NFE lower than 10, the presence of minimal random noise amplifies the error introduced by the gradient information approximation in PFDiff, reducing its error correction capability compared to the Analytic-DDIM sampler. Thus, in fewer-step sampling (NFE$<$10), using DDIM as the baseline is more effective than using Analytic-DDIM, which requires recalculating the optimal variance for different pre-trained DPMs, thereby introducing additional computational overhead. In other experiments with pre-trained DPMs, we validate the efficacy of the PFDiff algorithm by combining it with the overall superior performance of the DDIM solver.

Furthermore, to validate the motivation proposed in Sec. 3.2 based on Fig. 2a—that at not excessively large time step size Δt, an ODE-based solver shows considerable similarity in the noise network outputs—we compare it with the SDE-based solver DDPM [2]. Even at smaller Δt, the mean squared error (MSE) of the noise outputs from DDPM remains high, suggesting that the effectiveness of PFDiff may be limited when based on SDE solvers. Further, we adjusted the η parameter in Eq. (6) (which controls the amount of noise introduced in DDPM) from 1.0 to 0.0 (at $\eta = 0.0$, the SDE-based DDPM degenerates into the ODE-based DDIM [20]). As shown in Fig. 2a, as η decreases, the MSE of the noise network outputs gradually decreases at the same time step size Δt, indicating that reducing noise introduction can enhance the effectiveness of PFDiff. To verify this motivation, we utilized quadratic time steps on CIFAR10 and CelebA 64x64 datasets and controlled

Table 2: Sample quality measured by FID↓ on the CIFAR10 [42], CelebA 64x64 [43], LSUN-bedroom 256x256 [44], and LSUN-church 256x256 [44] using unconditional discrete-time DPMs, varying the number of function evaluations (NFE). Evaluated on 50k samples. PFDiff uses DDIM [20] and Analytic-DDIM [23] as baselines and introduces DDPM [2] and Analytic-DDPM [23] with $\eta = 1.0$ from Eq. (6) for comparison.

+PFDiff	Method	NFE						
		4	6	8	10	12	15	20
CIFAR10 (discrete-time model [2], quadratic time steps)								
$\times$	DDPM($\eta = 1.0$) [2]	108.05	71.47	52.87	41.18	32.98	25.59	18.34
$\times$	Analytic-DDPM [23]	65.81	56.37	44.09	34.95	29.96	23.26	17.32
$\times$	Analytic-DDIM [23]	106.86	24.02	14.21	10.09	8.80	7.25	6.17
$\times$	DDIM [20]	65.70	29.68	18.45	13.66	11.01	8.80	7.04
✓	Analytic-DDIM	289.84	23.24	7.03	**4.51**	**3.91**	**3.75**	**3.65**
✓	DDIM	**22.38**	**9.48**	**5.64**	4.57	4.39	4.10	3.68
CelebA 64x64 (discrete-time model [20], quadratic time steps)								
$\times$	DDPM($\eta = 1.0$) [2]	59.38	43.63	34.12	28.21	24.40	20.19	15.85
$\times$	Analytic-DDPM [23]	32.10	39.78	32.29	26.96	23.03	19.36	15.67
$\times$	Analytic-DDIM [23]	69.75	16.60	11.84	9.37	7.95	6.92	5.84
$\times$	DDIM [20]	37.76	20.99	14.10	10.86	9.01	7.67	6.50
✓	Analytic-DDIM	360.21	28.24	5.66	4.90	4.62	**4.55**	**4.55**
✓	DDIM	**13.29**	**7.53**	**5.06**	**4.71**	**4.60**	4.70	4.68
CelebA 64x64 (discrete-time model [20], uniform time steps)								
$\times$	DDPM($\eta = 1.0$) [2]	65.39	49.52	41.65	36.68	33.45	30.27	26.76
$\times$	Analytic-DDPM [23]	102.45	42.43	34.36	33.85	30.38	28.90	25.89
$\times$	Analytic-DDIM [23]	90.44	24.85	16.45	16.67	15.11	15.00	13.40
$\times$	DDIM [20]	**44.36**	29.12	23.19	20.50	18.43	16.71	14.76
✓	Analytic-DDIM	308.58	56.04	14.07	10.98	8.97	**6.39**	**5.19**
✓	DDIM	51.87	**12.79**	**8.82**	**8.93**	**7.70**	6.44	5.66
LSUN-bedroom 256x256 (discrete-time model [2], uniform time steps)								
$\times$	DDIM [20]	**115.63**	47.40	26.73	19.26	15.23	11.68	9.26
✓	DDIM	140.40	**18.72**	**11.50**	**9.28**	**8.36**	**7.76**	**7.14**
LSUN-church 256x256 (discrete-time model [2], uniform time steps)								
$\times$	DDIM [20]	121.95	50.02	30.04	22.04	17.66	14.58	12.49
✓	DDIM	**72.86**	**18.30**	**14.34**	**13.27**	**12.05**	**11.77**	**11.12**

the amount of noise introduced by adjusting η, to demonstrate that PFDiff can leverage the temporal redundancy present in ODE solvers to boost its performance. The experimental results, as shown in Table 3, illustrate that with the reduction of η from 1.0 (SDE) to 0.0 (ODE), PFDiff's sampling performance significantly improves at fewer time steps (NFE$\leq$20). The experiment results regarding FID variations with NFE as presented in Table 3, align with the trends of MSE of noise network outputs with changes in time step size Δt as depicted in Fig. 2a. This reaffirms the motivation we proposed in Sec. 3.2.

D.3 Additional results for unconditional continuous-time sampling

In this section, we supplement the specific FID↓ scores for the unconditional, continuous pre-trained DPMs models with first-order and higher-order ODE solvers, DPM-Solver-1, -2 and -3, [21] as baselines, as shown in Table 4. For all experiments in this section, we conducted tests on the CIFAR10 dataset [42], using the checkpoint `checkpoint_8.pth` under the

Table 3: Sample quality measured by FID↓ on the CIFAR10 [42] and CelebA 64x64 [43] using unconditional discrete-time DPMs with and without our method (PFDiff), varying the number of function evaluations (NFE) and η from Eq. (6). Evaluated on 50k samples.

Method	NFE						
	4	6	8	10	12	15	20
CIFAR10 (discrete-time model [2], quadratic time steps)							
DDPM($\eta = 1.0$) [2]	108.05	71.47	52.87	41.18	32.98	25.59	18.34
+PFDiff (Ours)	475.47	432.24	344.96	332.41	285.88	158.90	28.05
DDPM($\eta = 0.5$) [20]	71.08	34.32	22.37	16.63	13.37	10.75	8.38
+PFDiff (Ours)	432.50	349.09	311.62	167.65	59.93	23.17	10.61
DDPM($\eta = 0.2$) [20]	66.33	30.26	18.94	14.01	11.25	9.00	7.18
+PFDiff (Ours)	316.15	189.02	18.55	7.73	5.70	4.53	4.00
DDIM($\eta = 0.0$) [20]	65.70	29.68	18.45	13.66	11.01	8.80	7.04
+PFDiff (Ours)	**22.38**	**9.48**	**5.64**	**4.57**	**4.39**	**4.10**	**3.68**
CelebA 64x64 (discrete-time model [20], quadratic time steps)							
DDPM($\eta = 1.0$) [2]	59.38	43.63	34.12	28.21	24.40	20.19	15.85
+PFDiff (Ours)	433.25	439.19	415.41	317.43	324.58	326.50	171.41
DDPM($\eta = 0.5$) [20]	40.58	23.72	16.74	13.15	11.27	9.36	7.73
+PFDiff (Ours)	435.27	417.58	314.63	310.10	252.19	69.31	19.23
DDPM($\eta = 0.2$) [20]	38.20	21.35	14.55	11.22	9.47	7.99	6.71
+PFDiff (Ours)	394.03	319.02	45.15	12.71	7.85	5.10	4.96
DDIM($\eta = 0.0$) [20]	37.76	20.99	14.10	10.86	9.01	7.67	6.50
+PFDiff (Ours)	**13.29**	**7.53**	**5.06**	**4.71**	**4.60**	**4.70**	**4.68**

Table 4: Sample quality measured by FID↓ of different orders of DPM-Solver [21] on the CIFAR10 [42] using unconditional continuous-time DPMs with and without our method (PFDiff), varying the number of function evaluations (NFE). Evaluated on 50k samples.

Method	order	NFE						
		4	6	8	10	12	16	20
CIFAR10 (continuous-time model [4], quadratic time steps)								
DPM-Solver-1 [21]	1	**40.55**	23.86	15.57	11.64	9.64	7.23	6.06
+PFDiff (Ours)	1	113.74	**11.41**	**5.90**	**4.23**	**3.92**	**3.73**	**3.75**
DPM-Solver-2 [21]	2	298.79	106.05	41.79	14.43	6.75	**4.24**	**3.91**
+PFDiff (Ours)	2	**85.22**	**16.30**	**9.67**	**6.64**	**5.74**	5.12	4.78
			6	9		12	15	21
DPM-Solver-3 [21]	3		382.51	233.56		44.82	7.98	**3.63**
+PFDiff (Ours)	3		**103.22**	**5.67**		**5.72**	**5.62**	5.24

vp/cifar10_ddpmpp_deep_continuous configuration provided by ScoreSDE [4]. For the hyper-parameter method of DPM-Solver [21], we adopted singlestep_fixed; to maintain consistency with the discrete-time model in Appendix D.2, the parameter skip was set to time_quadratic (i.e., quadratic time steps). Unless otherwise specified, we used the parameter settings recommended by DPM-Solver. The results in Table 4 show that by using the PFDiff method described in Sec. 3.4 and taking DPM-Solver as the baseline, we were able to further enhance sampling performance on the basis of first-order and higher-order ODE solvers. Particularly, in the 6$\sim$12 NFE range, PFDiff significantly improved the convergence issues of higher-order ODE solvers under fewer NFEs. For instance, at 9 NFE, PFDiff reduced the FID of DPM-Solver-3 from 233.56 to 5.67, improving the sampling quality by 97.57%. These results validate the effectiveness of using PFDiff with first-order or higher-order ODE solvers as the baseline.

D.4 Additional results for classifier guidance

Table 5: Sample quality measured by FID$\downarrow$ on the ImageNet 64x64 [32] and ImageNet 256x256 [32], using ADM-G [5] model with guidance scales of 1.0 and 2.0, varying the number of function evaluations (NFE). Evaluated: ImageNet 64x64 with 50k, ImageNet 256x256 with 10k samples. *We directly borrowed the results reported by AutoDiffusion [26], and AutoDiffusion requires additional search costs. *We directly borrowed the results reported by AutoDiffusion [26], and AutoDiffusion requires additional search costs. "\" represents missing data in the original paper.

Method	Step	NFE					
		4	6	8	10	15	20
ImageNet 64x64 (pixel DPMs model [5], uniform time steps, guidance scale 1.0)							
DDIM [20]	Single	138.81	23.58	12.54	8.93	5.52	4.45
DPM-Solver-2 [21]	Single	327.09	292.66	264.97	236.80	166.52	120.29
DPM-Solver-2 [21]	Multi	48.64	21.08	12.45	8.86	5.57	4.46
DPM-Solver-3 [21]	Single	383.71	376.86	380.51	378.32	339.34	280.12
DPM-Solver-3 [21]	Multi	54.01	24.76	13.17	8.85	5.48	4.41
DPM-Solver++(2M) [22]	Multi	44.15	20.44	12.53	8.95	5.53	4.33
*AutoDiffusion [26]	Single	17.86	11.17	\	6.24	4.92	3.93
DDIM+PFDiff (Ours)	Single	**16.46**	**8.20**	**6.22**	**5.19**	**4.20**	**3.83**
ImageNet 256x256 (pixel DPMs model [5], uniform time steps, guidance scale 2.0)							
DDIM [20]	Single	51.79	23.48	16.33	12.93	9.89	9.05
DDIM+PFDiff (Ours)	Single	**37.81**	**18.15**	**12.22**	**10.33**	**8.59**	**8.08**

In this section, we provide the specific FID scores for pre-trained DPMs in the conditional, classifier guidance paradigm on the ImageNet 64x64 [32] and ImageNet 256x256 datasets [32], as shown in Table 5. We now describe the experimental setup in detail. For the pre-trained models, we used the ADM-G [5] provided `64x64_diffusion.pt` and `64x64_classifier.pt` for the ImageNet 64x64 dataset, and `256x256_diffusion.pt` and `256x256_classifier.pt` for the ImageNet 256x256 dataset. All experiments were conducted with uniform time steps and used DDIM as the baseline [20]. We implemented the second-order and third-order methods from DPM-Solver [21] for comparison and explored the `method` hyperparameter provided by DPM-Solver for both `singlestep` (corresponding to "Single" in Table 5) and `multistep` (corresponding to "Multi" in Table 5). Additionally, we implemented the best-performing method from DPM-Solver++ [22], multi-step DPM-Solver++(2M), as a comparative measure. Furthermore, we also introduced the superior-performing AutoDiffusion [26] method as a comparison. *We directly borrowed the results reported in the original paper, emphasizing that although AutoDiffusion does not require additional training, it incurs additional search costs. "\" represents missing data in the original paper. The specific experimental results of the configurations mentioned are shown in Table 5. The results demonstrate that PFDiff, using DDIM as the baseline on the ImageNet 64x64 dataset, significantly enhances the sampling efficiency of DDIM and surpasses previous optimal training-free sampling methods. Particularly, in cases where NFE$\leq$10, PFDiff improved the sampling quality of DDIM by 41.88%$\sim$88.14%. Moreover, on the large ImageNet 256x256 dataset, PFDiff demonstrates a consistent performance improvement over the DDIM baseline, similar to the improvements observed on the ImageNet 64x64 dataset.

D.5 Additional results for classifier-free guidance

In this section, we supplemented the specific FID$\downarrow$ scores for the Stable-Diffusion [9] (conditional, classifier-free guidance paradigm) setting with a guidance scale (s) of 7.5 and 1.5. Specifically, for the pre-trained model, we conducted experiments using the `sd-v1-4.ckpt` checkpoint provided by Stable-Diffusion. All experiments used the MS-COCO2014 [31] validation set to calculate FID↓ scores, with uniform time steps. PFDiff employs the DDIM [20] method as the baseline. Initially, under the recommended $s = 7.5$ configuration by Stable-Diffusion, we implemented DPM-Solver-2 and -3 as comparative methods, and conducted searches for the `method` hyperparameters provided by DPM-Solver as `singlestep` (corresponding to "Single" in Table 6) and `multistep` (corresponding to "Multi" in Table 6). Additionally, we introduced previous state-of-the-art training-free methods, including DPM-Solver++(2M) [22], UniPC [29], and DPM-Solver-v3(2M) [27] for comparison. The experimental results are shown in Table 6. †We borrow the results reported in DPM-Solver-v3 [27] directly. The results indicate that on Stable-Diffusion, PFDiff, using only DDIM as a baseline, surpasses the previous state-of-the-art training-free sampling methods in terms of sampling quality in fewer steps (NFE<20). Particularly, at NFE=10, PFDiff achieved a 13.06 FID, nearly converging to the data distribution, which is a 14.25% improvement over the previous state-of-the-art method DPM-Solver-v3 at 20 NFE, which had a 15.23 FID. Furthermore, to further validate the effectiveness of PFDiff on Stable-Diffusion, we conducted experiments using the $s = 1.5$ setting with the same experimental configuration as $s = 7.5$. For the comparative methods, we only experimented with the multi-step versions of DPM-Solver-2 and -3 and DPM-Solver++(2M), which had faster convergence at fewer NFE under the $s = 7.5$ setting. As for UniPC and DPM-Solver-v3(2M), since DPM-Solver-v3 did not provide corresponding experimental results at $s = 1.5$, we did not list their comparative results. The experimental results show that PFDiff, using DDIM as the baseline under the $s = 1.5$ setting, demonstrated consistent performance improvements as seen in the $s = 7.5$ setting, as shown in Table 6.

Table 6: Sample quality measured by FID↓ on the validation set of MS-COCO2014 [31] using Stable-Diffusion model [9] with guidance scales of 7.5 and 1.5, varying the number of function evaluations (NFE). Evaluated on 10k samples. †We borrow the results reported in DPM-Solver-v3 [27] directly.

Method	Step	NFE					
		5	6	8	10	15	20
MS-COCO2014 (latent DPMs model [9], uniform time steps, guidance scale 7.5)							
DDIM [20]	Single	23.92	20.33	17.46	16.78	16.08	15.95
DPM-Solver-2 [21]	Single	84.15	74.02	31.87	17.63	15.15	13.77
DPM-Solver-2 [21]	Multi	18.97	17.37	16.29	15.99	14.32	14.38
DPM-Solver-3 [21]	Single	156.27	102.59	54.52	26.29	16.95	14.85
DPM-Solver-3 [21]	Multi	18.89	17.34	16.25	16.11	14.10	**13.44**
†DPM-Solver++(2M) [22]	Multi	18.87	17.44	16.40	15.93	15.84	15.72
†UniPC [29]	Multi	18.77	17.32	16.20	16.15	16.06	15.94
†DPM-Solver-v3(2M) [27]	Multi	18.83	16.41	15.41	15.32	15.30	15.23
DDIM+PFDiff (Ours)	Single	**18.31**	**15.47**	**13.26**	**13.06**	**13.57**	13.97
MS-COCO2014 (latent DPMs model [9], uniform time steps, guidance scale 1.5)							
DDIM [20]	Single	70.36	54.32	37.54	29.41	20.54	18.17
DPM-Solver-2 [21]	Multi	37.47	27.79	19.65	18.39	17.27	16.85
DPM-Solver-3 [21]	Multi	35.90	25.88	18.26	19.10	17.21	16.67
DPM-Solver++(2M) [22]	Multi	36.58	26.78	18.92	20.26	18.61	17.78
DDIM+PFDiff (Ours)	Single	**24.31**	**20.99**	**18.09**	**17.00**	**16.03**	**15.57**

D.6 Additional ablation study results

D.6.1 Additional results for PFDiff hyperparameters study

In this section, we extensively investigate the impact of the hyperparameters k and l on the performance of the PFDiff algorithm, supplemented by a series of ablation experiments regarding their configurations and outcomes. Specifically, we first conducted experiments on the CIFAR10 dataset

Table 7: Ablation of the impact of k and l on PFDiff in CIFAR10 [42], ImageNet 64x64 and MS-COCO2014 using DDPM [2], ScoreSDE [4], ADM-G [5] and Stable-Diffusion [9] models. We report the FID↓, varying the number of function evaluations (NFE). Evaluated: MS-COCO2014 with 10k, others with 50k samples.

Method	NFE					
	4	6	8	10	15	20
CIFAR10 (discrete-time model [2], quadratic time steps)						
DDIM [20]	65.70	29.68	18.45	13.66	8.80	7.04
+PFDiff-1	124.73	19.45	5.78	4.95	4.25	4.14
+PFDiff-2_1	59.61	**9.84**	7.01	6.31	5.18	4.78
+PFDiff-2_2	167.12	53.22	8.43	4.95	**4.10**	3.78
+PFDiff-3_1	**22.38**	13.40	9.40	7.70	6.03	5.05
+PFDiff-3_2	129.18	19.35	**5.64**	**4.57**	4.19	4.08
+PFDiff-3_3	205.87	76.62	20.84	5.71	4.41	**3.68**
CIFAR10 (continuous-time model [4], quadratic time steps)						
DPM-Solver-1 [21]	**40.55**	23.86	15.57	11.64	7.59	6.06
+PFDiff-1	250.56	76.78	6.53	4.28	**3.78**	**3.75**
+PFDiff-2_1	178.70	**11.41**	**5.90**	5.01	4.27	4.07
+PFDiff-2_2	289.06	250.48	71.08	9.17	4.09	3.83
+PFDiff-3_1	113.74	11.82	7.91	6.34	4.97	4.37
+PFDiff-3_2	264.88	130.24	8.92	**4.23**	**3.78**	3.78
+PFDiff-3_3	275.10	287.77	183.11	30.72	4.69	4.01
ImageNet 64x64 (pixel DPMs model [5], uniform time steps, $s = 1.0$)						
DDIM [20]	138.81	23.58	12.54	8.93	5.52	4.45
+PFDiff-1	26.86	11.39	7.47	5.83	4.76	4.39
+PFDiff-2_1	17.14	8.94	6.38	5.46	4.30	**3.83**
+PFDiff-2_2	23.66	9.93	6.86	5.72	4.49	3.94
+PFDiff-3_1	16.74	9.43	7.19	5.86	4.69	4.44
+PFDiff-3_2	**16.46**	**8.20**	**6.22**	**5.19**	**4.20**	4.28
+PFDiff-3_3	23.06	9.73	6.92	5.55	4.47	4.49
MS-COCO2014 (latent DPMs model [9], uniform time steps, $s = 7.5$)						
DDIM [20]	35.48	20.33	17.46	16.78	16.08	15.95
+PFDiff-1	98.78	23.06	**13.26**	**13.06**	13.72	14.09
+PFDiff-2_1	33.39	**15.47**	15.05	15.01	15.24	15.35
+PFDiff-2_2	178.10	53.77	16.92	13.55	**13.57**	14.08
+PFDiff-3_1	**29.02**	16.38	15.69	15.66	15.52	15.51
+PFDiff-3_2	75.73	17.60	14.46	14.52	14.84	14.99
+PFDiff-3_3	217.86	80.03	21.99	14.38	13.61	**13.97**

[42] using quadratic time steps, based on pre-trained unconditional discrete DDPM [2] and continuous ScoreSDE [4] DPMs. For the conditional DPMs, we used uniform time steps in classifier guidance ADM-G [5] pre-trained DPMs, setting the guidance scale (s) to 1.0 for experiments on the ImageNet 64x64 dataset [32]; for the classifier-free guidance Stable-Diffusion [9] pre-trained DPMs, we set the guidance scale (s) to 7.5. All experiments were conducted using the DDIM [20] algorithm as a baseline, and PFDiff-k_l configurations ($k = 1, 2, 3$ ($l \leq k$)) were tested in six different algorithm configurations. The change in NFE and the corresponding FID↓ scores are shown in Table 7. The experimental results show that under various combinations of k and l, PFDiff is able to enhance the sampling performance of the DDIM baseline in most cases across different types of pre-trained DPMs. Particularly when $k = 1$ is fixed, PFDiff-1 significantly improves the sampling performance of the DDIM baseline within the range of 8∼20 NFE. For practical applications requiring higher sampling quality at fewer NFE, optimal combinations of k and l can be identified by fixing NFE and sampling a small number of samples for visual analysis or computing specific metrics, without significantly

increasing the computational burden. However, as discussed in Sec. 5, although the experimental results presented in Table 7 demonstrate the excellent performance of the combinations of k and l under various pre-trained DPMs and NFE settings, no universally optimal configuration exists. This finding somewhat limits the generality of the proposed PFDiff algorithm and sets objectives for our future research.

D.6.2 Ablation study of gradient guidance

To further investigate the impact of gradient guidance from the past or future on the rapid updating of current intermediate states, this section supplements experimental results and analysis using first-order and higher-order ODE solvers as baselines. Specifically, as described in Sec. 3.3, PFDiff uses a first-order ODE solver as a baseline, where future gradient guidance corrects sampling errors, with detailed proofs provided in Appendix B.2. Hence, using the first-order ODE solver DDIM [20] as a baseline, we removed past and future gradients separately and employed quadratic time steps. This was based on the pre-trained model from DDPM [2] on the CIFAR10 [42] dataset, evaluating the FID↓ metric by changing the number of function evaluations (NFE). For higher-order ODE solvers, the solving process implicitly utilizes future gradients as mentioned in Sec. 3.5, and the additional explicit introduction of future gradients increases sampling error. Therefore, when using higher-order ODE solvers as a baseline, PFDiff accelerates the sampling process using only past gradients. Specifically, for higher-order ODE solvers, we selected DPM-Solver-2 and -3 [21] as the baseline, also employing quadratic time steps, and based on the ScoreSDE [4] pre-trained model on CIFAR10 [42]. Only the future gradients were removed to validate the effectiveness of the PFDiff algorithm by changing the NFE and evaluating the FID↓ metric. As shown in Table 8, the experimental results indicate that using the first-order ODE solver DDIM as a baseline, employing only past gradients (similar to DeepCache [28]), or only future gradients, only slightly improves the baseline's sampling performance; however, combining both significantly enhances the baseline's sampling performance. Meanwhile, using higher-order ODE solvers DPM-Solver-2 and -3 as the baseline, because the algorithm inherently contains future gradients, continuing to explicitly introduce future gradients increases the overall error. Therefore, using only past gradients (PFDiff) significantly improves the baseline's sampling efficiency, especially under fewer steps (NFE<10), where PFDiff markedly ameliorates the non-convergence issues of the higher-order ODE solvers.

Table 8: Ablation of the impact of the past and future gradients on PFDiff, using different orders of ODE Solver as the baseline, in CIFAR10 [42] using DDPM [2] and ScoreSDE [4] models. We report the FID↓, varying the number of function evaluations (NFE). Evaluated on 50k samples.

+PFDiff	Method	NFE						
		4	6	8	10	12	16	20
CIFAR10 (discrete-time model [2], quadratic time steps, baseline: 1-order ODE solver)								
×	DDIM [20]	65.70	29.68	18.45	13.66	11.01	8.80	7.04
×	+Past (similar to [28])	52.81	27.47	17.87	13.64	10.79	8.20	7.02
×	+Future	66.06	25.39	11.93	8.06	6.04	4.17	4.07
✓	+Past & Future	**22.38**	**9.84**	**5.64**	**4.57**	**4.39**	**4.10**	**3.68**
CIFAR10 (continuous-time model [4], quadratic time steps, baseline: 2-order ODE solver)								
×	DPM-Solver-2 [21]	298.79	106.05	41.79	14.43	6.75	**4.24**	**3.91**
✓	+Past	**85.22**	**16.30**	**9.67**	**6.64**	**5.74**	5.12	4.78
×	+Past & Future	351.78	159.13	57.15	28.24	15.57	6.47	4.73
CIFAR10 (continuous-time model [4], quadratic time steps, baseline: 3-order ODE solver)								
			6	9		12	15	21
×	DPM-Solver-3 [21]		382.51	233.56		44.82	7.98	**3.63**
✓	+Past		**103.22**	**5.67**		**5.72**	**5.62**	5.24
×	+Past & Future		336.26	88.99		27.54	9.59	5.12

D.7 Inception score experimental results

To evaluate the effectiveness of the PFDiff algorithm and the widely used Fréchet Inception Distance (FID↓) metric [40] in the sampling process of Diffusion Probabilistic Models (DPMs), we have also incorporated the Inception Score (IS↑) metric [41] for both unconditional and conditional pre-trained DPMs. Specifically, for the unconditional discrete-time pre-trained DPMs DDPM [2], we maintained the experimental configurations described in Table 2 of Appendix D.2, and added IS scores for the CIFAR10 dataset [42]. For the unconditional continuous-time pre-trained DPMs ScoreSDE[4], the experimental configurations are consistent with Table 4 in Appendix D.3, and IS scores for the CIFAR10 dataset were also added. For the conditional classifier guidance paradigm of pre-trained DPMs ADM-G [5], the experimental setup aligned with Table 5 in Appendix D.4, including IS scores for the ImageNet 64x64 and ImageNet 256x256 datasets [32]. Considering that the computation of IS scores relies on features extracted using `InceptionV3` pre-trained on the ImageNet dataset, calculating IS scores for non-ImageNet datasets was not feasible, hence no IS scores were provided for the classifier-free guidance paradigm of Stable-Diffusion [9]. The experimental results are presented in Table 9. A comparison between the FID↓ metrics in Tables 2, 4, and 5 and the IS↑ metrics in Table 9 shows that both IS and FID metrics exhibit similar trends under the same experimental settings, i.e., as the number of function evaluations (NFE) changes, lower FID scores correspond to higher IS scores. Further, Figs. 1a and 1b, along with the visualization experiments in Appendix D.8, demonstrate that lower FID scores and higher IS scores correlate with higher image quality and richer

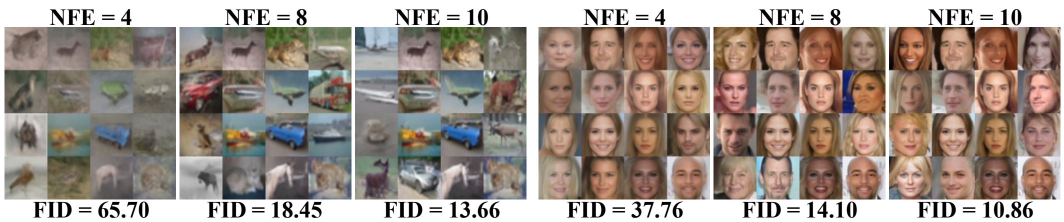
Table 9: Sample quality measured by IS↑ on the CIFAR10 [42], ImageNet 64x64 [32] and ImageNet 256x256 [32] using DDPM [2], ScoreSDE [4] and ADM-G [5] models, varying the number of function evaluations (NFE). Evaluated: ImageNet 256x256 with 10k, others with 50k samples. *We directly borrowed the results reported by AutoDiffusion [26], and AutoDiffusion requires additional search costs. "\" represents missing data in the original paper and DPM-Solver-2 [21] implementation.

+PFDiff	Method	NFE					
		4	6	8	10	15	20
CIFAR10 (discrete-time model [2], quadratic time steps)							
×	DDPM($\eta = 1.0$) [2]	4.32	5.66	6.55	7.08	7.91	8.25
×	Analytic-DDPM [23]	5.76	6.29	6.93	7.42	8.07	8.33
×	Analytic-DDIM [23]	4.46	7.47	8.11	8.43	8.72	8.89
×	DDIM [20]	5.68	7.21	7.92	8.26	8.62	8.81
✓	Analytic-DDIM	1.62	8.78	9.43	**9.61**	**9.35**	**9.29**
✓	DDIM	**7.79**	**9.29**	**9.62**	9.43	9.29	**9.29**
CIFAR10 (continuous-time model [4], quadratic time steps)							
×	DPM-Solver-1 [21]	**7.20**	8.30	8.85	8.98	9.43	9.51
×	DPM-Solver-2 [21]	1.70	5.29	7.94	9.09	\	9.74
✓	DPM-Solver-1	4.29	**9.25**	**9.76**	**9.86**	**9.85**	**9.97**
✓	DPM-Solver-2	6.96	8.58	8.75	9.26	\	9.69
ImageNet 64x64 (pixel DPMs model [5], uniform time steps, guidance scale 1.0)							
×	DDIM [20]	7.02	31.13	40.51	46.06	54.37	59.09
×	DPM-Solver-2(Multi) [21]	19.03	33.75	44.65	51.79	62.18	67.69
×	DPM-Solver-3(Multi) [21]	17.46	29.80	41.86	50.90	62.68	68.44
×	DPM-Solver++(2M) [22]	20.72	34.22	43.62	50.02	60.00	65.66
×	*AutoDiffusion [26]	34.88	43.37	\	57.85	64.03	68.05
✓	DDIM	**35.67**	**50.14**	**58.42**	**59.78**	**64.54**	**69.09**
ImageNet 256x256 (pixel DPMs model [5], uniform time steps, guidance scale 2.0)							
×	DDIM [20]	37.72	95.90	122.13	144.13	165.91	179.27
✓	DDIM	**55.90**	**122.56**	**158.57**	**169.72**	**183.07**	**192.70**

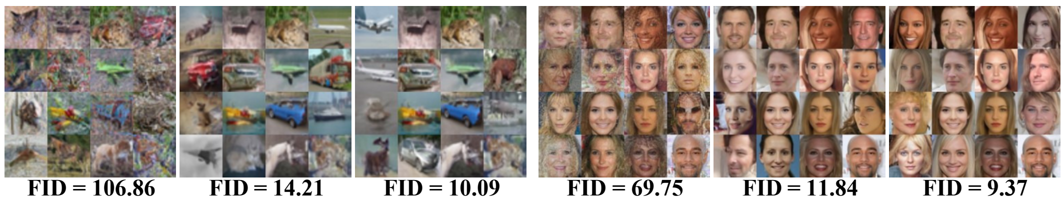
details generated by the PFDiff sampling algorithm. These results further confirm the effectiveness of the PFDiff algorithm and the FID metric in evaluating the performance of sampling algorithms.

D.8 Additional visualize study results

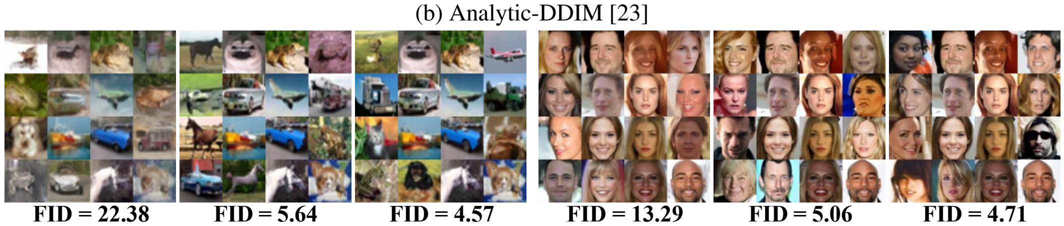
To demonstrate the effectiveness of PFDiff, we present the visual sampling results on the CIFAR10 [42], CelebA 64x64 [43], LSUN-bedroom 256x256 [44], LSUN-church 256x256 [44], ImageNet 64x64 [32], ImageNet 256x256 [32], and MS-COCO2014 [31] datasets in Figs. 5-10. These results illustrate that PFDiff, using different orders of ODE solvers as a baseline, is capable of generating samples of higher quality and richer detail on both unconditional and conditional pre-trained Diffusion Probabilistic Models (DPMs).



(a) DDIM [20]

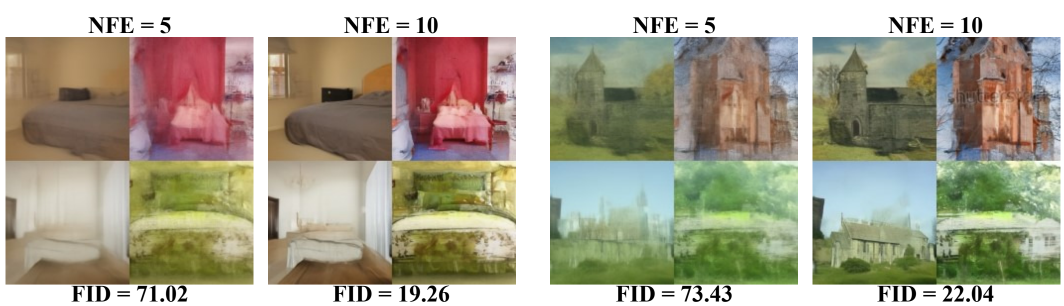


(b) Analytic-DDIM [23]

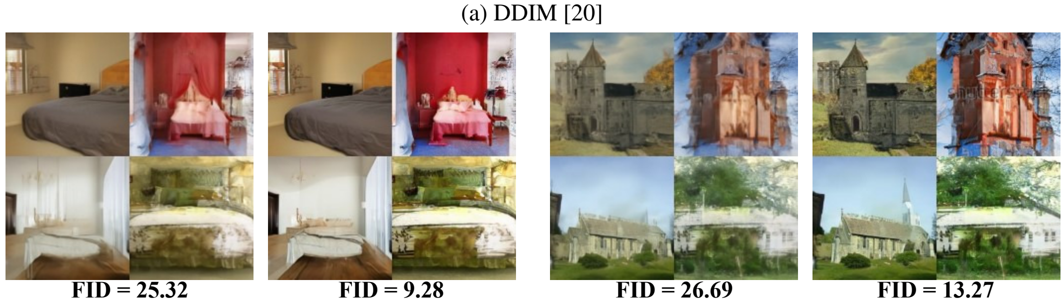


(c) DDIM+PFDiff (Ours)

Figure 5: Random samples by DDIM [20], Analytic-DDIM [23], and PFDiff (baseline: DDIM) with 4, 8, and 10 number of function evaluations (NFE), using the same random seed, quadratic time steps, and pre-trained discrete-time DPMs [2, 20] on CIFAR10 [42] (left) and CelebA 64x64 [43] (right).

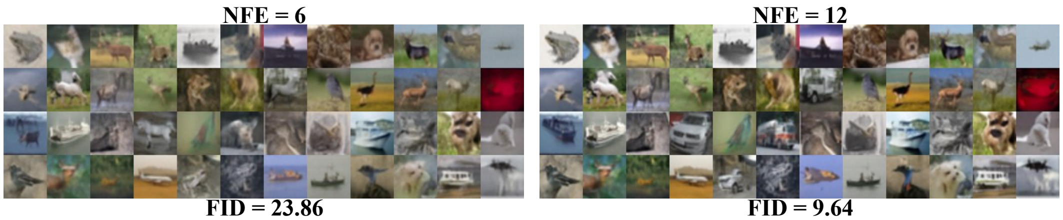


(a) DDIM [20]

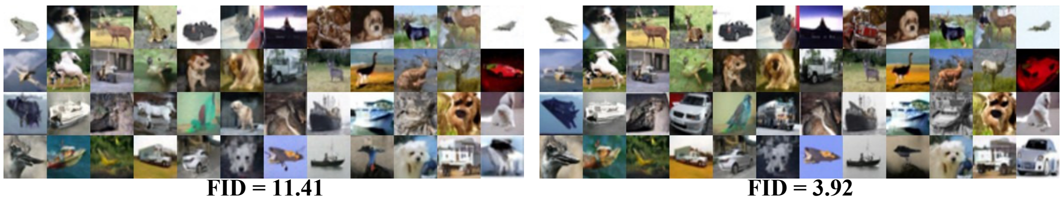


(b) DDIM+PFDiff (Ours)

Figure 6: Random samples by DDIM [20] and PFDiff (baseline: DDIM) with 5 and 10 number of function evaluations (NFE), using the same random seed, uniform time steps, and pre-trained discrete-time DPMs [2] on LSUN-bedroom 256x256 [44] (left) and LSUN-church 256x256 [44] (right).



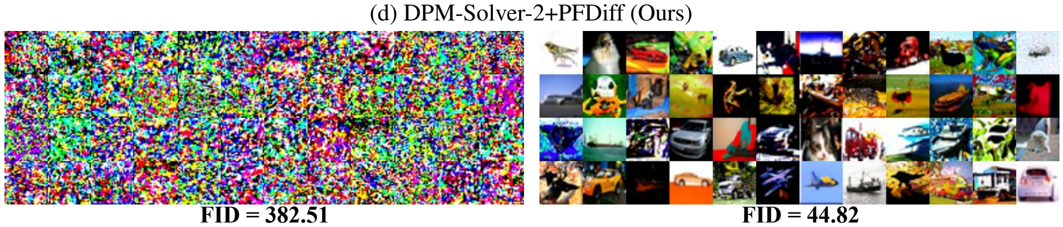
(a) DPM-Solver-1 [21]



(b) DPM-Solver-1+PFDiff (Ours)

(c) DPM-Solver-2 [21]

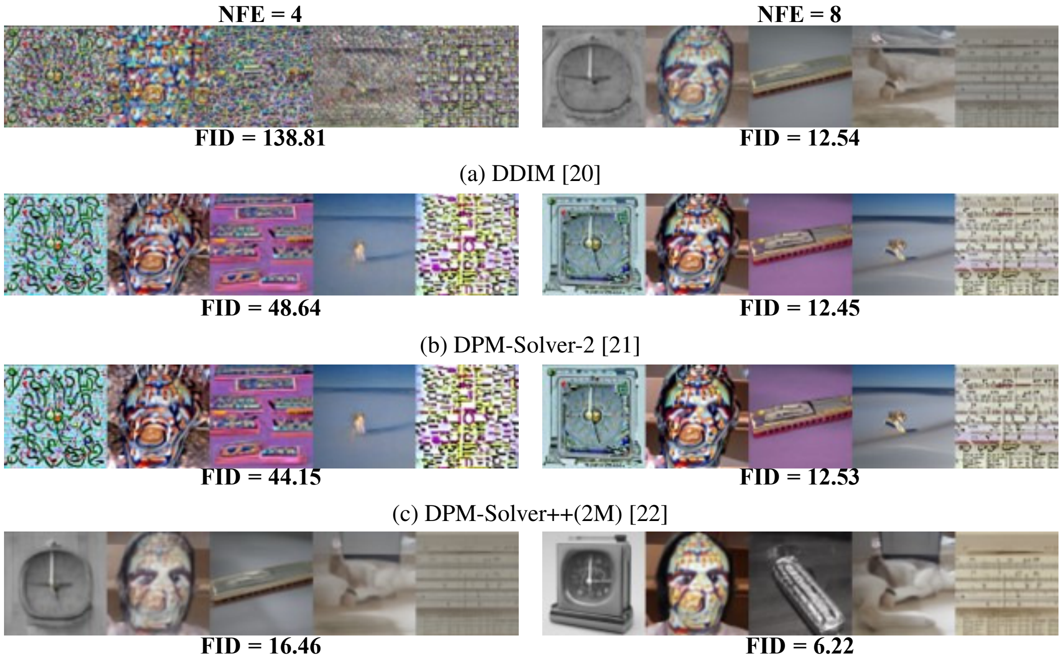
(d) DPM-Solver-2+PFDiff (Ours)



(e) DPM-Solver-3 [21]

(f) DPM-Solver-3+PFDiff (Ours)

Figure 7: Random samples by DPM-Solver-1, -2, and -3 [21] with and without our method (PFDiff) with 6 and 12 number of function evaluations (NFE), using the same random seed, quadratic time steps, and pre-trained continuous-time DPMs [4] on CIFAR10 [42].



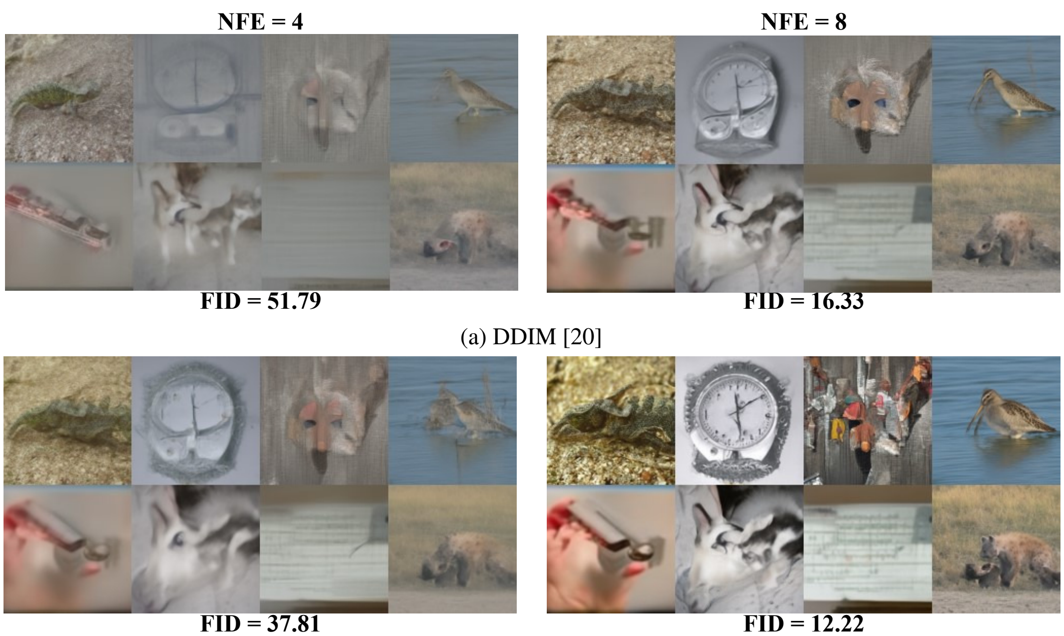
(a) DDIM [20]

(b) DPM-Solver-2 [21]

(c) DPM-Solver++(2M) [22]

(d) DDIM+PFDiff (Ours)

Figure 8: Random samples by DDIM [20], DPM-Solver-2 [21], DPM-Solver++(2M) [22], and PFDiff (baseline: DDIM) with 4 and 8 number of function evaluations (NFE), using the same random seed, uniform time steps, and pre-trained Guided-Diffusion [5] on ImageNet 64x64 [32] with a guidance scale of 1.0.



(a) DDIM [20]

(b) DDIM+PFDiff (Ours)

Figure 9: Random samples by DDIM [20] and PFDiff (baseline: DDIM) with 4 and 8 number of function evaluations (NFE), using the same random seed, uniform time steps, and pre-trained Guided-Diffusion [5] on ImageNet 256x256 [32] with a guidance scale of 2.0.

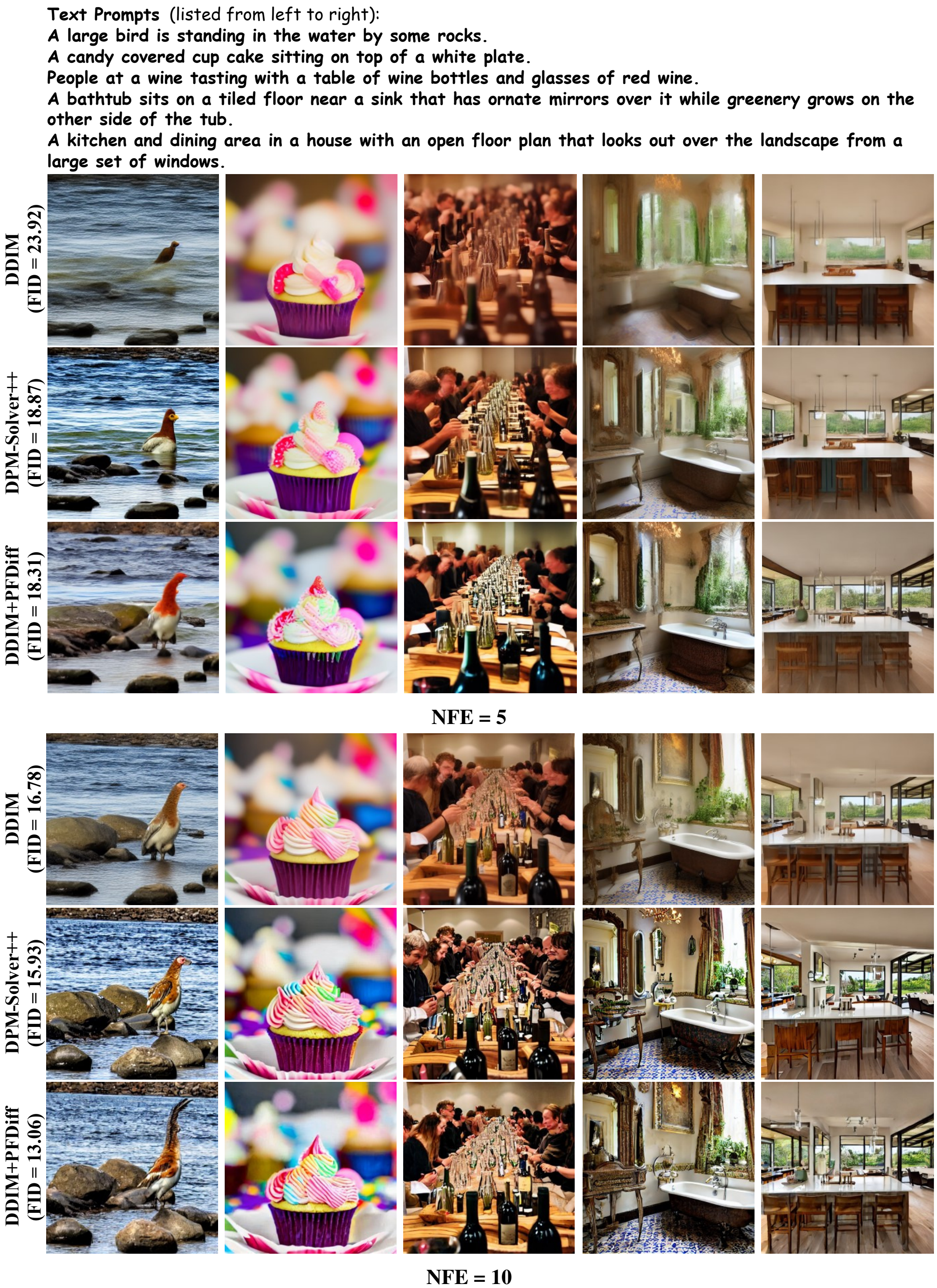


Figure 10: Random samples by DDIM [20], DPM-Solver++(2M) [22], and PFDiff (baseline: DDIM) with 5 and 10 number of function evaluations (NFE), using the same random seed, uniform time steps, and pre-trained Stable-Diffusion [9] with a guidance scale of 7.5. Text prompts are a random sample from the MS-COCO2014 [31] validation set.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: See abstract and section 1.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See section 5.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: See Appendix B.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See section 4 and Appendix D.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code in the supplemental materials, which will be made available on the GitHub platform.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Our method is training-free, but we also report the hyperparameters used when evaluating our proposed method. Details can be found in section 4 and Appendix D.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We generate 50k images for 32x32, 64x64 datasets and 10k for 256x256 to evaluate the FID metric. According to previous works [20, 21, 26, 27], when evaluating with the generated samples mentioned above, the standard deviation of the FID evaluations is rather small (mainly less than 0.01). These small standard deviations do not change the conclusions.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: In section 4, we mentioned that all experiments were conducted on an NVIDIA RTX 3090 GPU.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work is conducted with the NeurIPS code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See section 5.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our method is a training-free accelerated sampling approach. It relies on existing pre-trained DPMs and does not involve the release of data or models, thus there is no need for safeguards.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: See Appendix D.1.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We do not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: We do not involve human subjects nor crowdsourcing.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.