Improving Training Efficiency of Diffusion Models via Multi-Stage Framework and Tailored Multi-Decoder Architecture

Huijie Zhang[1] *   Yifu Lu[1] *   Ismail Alkhouri[1,2]   Saiprasad Ravishankar[2,3]
Dogyoon Song[1]   Qing Qu[1]

[1]Department of Electrical Engineering & Computer Science, University of Michigan
[2]Department of Computational Mathematics, Science & Engineering, Michigan State University
[3]Department of Biomedical Engineering, Michigan State University

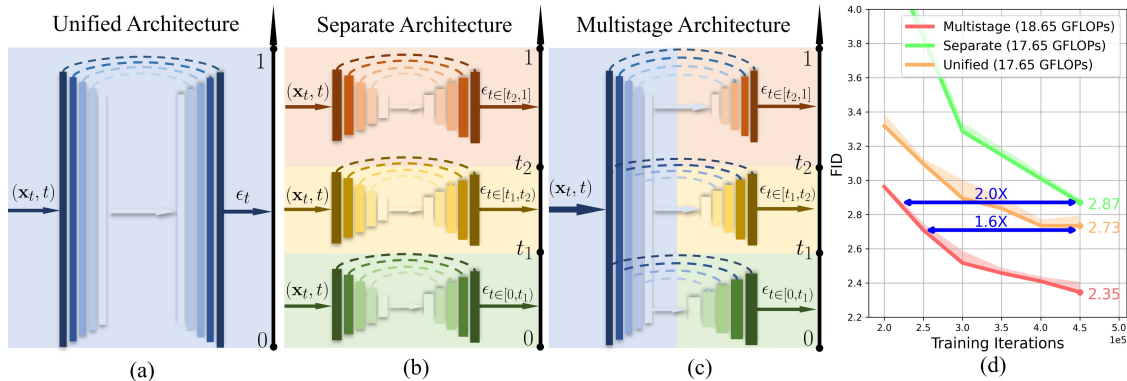{huijiezh, yifulu, ismailal, dogyoons, qingqu}@umich.edu   ravisha3@msu.edu

Figure 1. **Overview of three diffusion model architectures:** (a) unified, (b) separate, and (c) our proposed multistage architecture. Compared with (a) and (b), our approach improves sampling quality, and significantly enhances training efficiency, as indicated by the FID scores and their corresponding training iterations (d).

Abstract

Diffusion models, emerging as powerful deep generative tools, excel in various applications. They operate through a two-steps process: introducing noise into training samples and then employing a model to convert random noise into new samples (e.g., images). However, their remarkable generative performance is hindered by slow training and sampling. This is due to the necessity of tracking extensive forward and reverse diffusion trajectories, and employing a large model with numerous parameters across multiple timesteps (i.e., noise levels). To tackle these challenges, we present a multi-stage framework inspired by our empirical findings. These observations indicate the advantages of employing distinct parameters tailored to each timestep while retaining universal parameters shared across all time steps. Our approach involves segmenting the time interval into multiple stages where we employ custom multi-decoder U-net architecture that blends time-dependent models with a universally shared encoder. Our framework enables the efficient distribution of computational resources and mitigates inter-stage interference, which substantially improves training efficiency. Extensive numerical experiments affirm the effectiveness of our framework, showcasing significant training and sampling efficiency enhancements on three state-of-the-art diffusion models, including large-scale latent diffusion models. Furthermore, our ablation studies illustrate the impact of two important components in our framework: (i) a novel timestep clustering algorithm for stage division, and (ii) an innovative multi-decoder U-net architecture, seamlessly integrating universal and customized hyperparameters.

1. Introduction

Recently, diffusion models have made remarkable progress as powerful deep generative modeling tools, showcasing remarkable performance in various applications, ranging from unconditional image generation [1, 2], conditional image generation [3, 4], image-to-image translation [5–7], text-to-image generation [8–10], inverse problem solving [11–14],

video generation [15, 16], and so on. These models employ a training process involving continuous injection of noise into training samples ("diffusion"), which are then utilized to generate new samples, such as images, by transforming random noise instances through a reverse diffusion process guided by the "score function " of the data distribution learned by the model. Moreover, recent work demonstrates that those diffusion models enjoy optimization stability and model reproducibility compared with other types of generative models [17]. However, diffusion models suffer from slow training and sampling despite their remarkable generative capabilities, which hinders their use in applications where real-time generation is desired [1, 2]. These drawbacks primarily arise from the necessity of tracking extensive forward and reverse diffusion trajectories, as well as managing a large model with numerous parameters across multiple timesteps (i.e., diffusion noise levels).

In this paper, we address these challenges based on two key observations: (i) there exists substantial parameter redundancy in current diffusion models, and (ii) they are trained inefficiently due to dissimilar gradients across different noise levels. Specifically, we find that training diffusion models require fewer parameters to accurately learn the score function at high noise levels, while larger parameters are needed at low noise levels. Furthermore, we also observe that when learning the score function, distinct shapes of distributions at different noise levels result in dissimilar gradients, which appear to slow down the training process driven by gradient descent.

Building on these insights, we propose a multi-stage framework with two key components: (i) a multi-decoder U-net architecture, and (ii) a new partitioning algorithm to cluster timesteps (noise levels) into distinct stages. In terms of our new architecture, we design a multi-decoder U-Net that incorporates one universal encoder shared across all intervals and individual decoders tailored to each time stage; see Figure 1 (c) for an illustration. This approach combines the advantages of both universal and stage-specific architectures, which is much more efficient than the unified architecture for the entire training process [1, 2, 18] (Figure 1 (a)). Moreover, compared to previous approaches that completely separate architectures for each sub-interval [19–22] (Figure 1 (b)), our approach can effectively mitigate overfitting, leading to improved training efficiency. On the other hand, when it comes to partitioning the training stages of our network, we designed an algorithm aimed at grouping the timesteps. This is achieved by minimizing the functional distance within each cluster in the training objective and making use of the optimal denoiser formulation [18]. By integrating these two key components, our framework enables efficient allocation of computational resources (e.g., U-net parameters) and stage-tailored parameterization. Throughout our extensive numerical experi-

ments (Section 5), we show that out framework effectively improves both training and sampling efficiency. These experiments are performed on diverse benchmark datasets, demonstrating significant acceleration by using our framework when compared to three state-of-the-art (SOTA) diffusion model architectures. As a summary, the major contributions of this work can be highlighted as follows:

- **Identifying two key sources of inefficiency.** We identified two key sources that cause inefficiencies in training diffusion models across various time step: (i) a significant variation in the requirement of model capacity, and (ii) the gradient dissimilarity. As such, using a unified network cannot meet with the changing requirement at different time steps.
- **A new multi-stage framework.** We introduced a new multi-stage architecture, illustrated in Fig. 1 (c). We tackle these two sources of inefficiency by segmenting the time interval into multiple stages, where we employ customized multi-decoder U-net architectures that blends time-dependent models with a universally shared encoder.
- **Improved training and sampling efficiency.** With comparable computational resources for unconditional image generation, we demonstrate that our multi-stage approach improves the Fréchet Inception Distance (FID) score for all SOTA methods. For example, on CIFAR-10 dataset [23], our method improves the FID for DPM-Solver [24] from 2.84 to **2.37**, and it improves the FID for EDM [18] from 2.05 (our training result) to **1.96**. Moreover, on the CelebA dataset [25], while maintaining a similar generation quality, our approach significantly reduces the required training FLOPS of EDM by **82%** and the Latent Diffusion Model (LDM) [8] by **30%**.

Organization. In Sec. 2, we provide preliminaries and an overview of related literatures. In Sec. 3, we present our observations and analysis that motivated the proposed multistage framework, justifying its development. In Sec. 4, we describe our proposed multistage framework for diffusion models, outlining the two core components. Finally, in Sec. 5, we provide the results from our numerical experiments that validate the effectiveness of the proposed multistage approach.

2. Preliminaries & Related Work

In this section, we start by reviewing the basic fundamentals of diffusion models [1, 2, 18]. Subsequently, we delve into prior approaches aimed at improving the training and efficiency of diffusion models through the partitioning of the timestep interval. Lastly, we survey prior studies that significantly decrease the number of required sampling iterations.

Background on diffusion models. Let $x_0 \in \mathbb{R}^n$ denote a sample from the data distribution $p_{\text{data}}(x)$. Diffu-

sion models operate within forward and reverse processes. The forward process progressively perturbs data $\boldsymbol{x}_0$ to a noisy version $\boldsymbol{x}_{t\in[0,1]}$ via corrupting with the Gaussian kernel. This process can be formulated as a stochastic differential equation (SDE) [2] of the form $\mathrm{d}\boldsymbol{x} = \boldsymbol{x}_t f(t)\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}_t$, where $f(t)$ and $g(t)$ are the drift and diffusion coefficients, respectively, that correspond to a pre-defined noise schedule. $\boldsymbol{w}_t \in \mathbb{R}^n$ is the standard Wiener process. Under the forward SDE, the perturbation kernel is given by the conditional distribution defined as $p_t(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; s_t\boldsymbol{x}_0, s_t^2\sigma_t^2\mathbf{I})$, where

$$s_t = \exp\left(\int_0^t f(\xi)\mathrm{d}\xi\right), \text{ and } \sigma_t = \sqrt{\int_0^t \frac{g^2(\xi)}{s_\xi^2}\mathrm{d}\xi}. \quad (1)$$

The parameters s_t and σ_t are designed such that: (i) the data distribution is approximately estimated when $t = 0$, and (ii) a nearly standard Gaussian distribution is obtained when $t = 1$. The objective of diffusion models is to learn the corresponding reverse SDE, defined as $\mathrm{d}\boldsymbol{x} = \left[f(t)\boldsymbol{x}_t - g^2(t)\nabla_{\boldsymbol{x}_t}\log p_t(\boldsymbol{x}_t)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\boldsymbol{w}}$, where $\bar{\boldsymbol{w}} \in \mathbb{R}^n$ is the standard Wiener process running backward in time, and $\nabla_{\boldsymbol{x}_t}\log p_t(\boldsymbol{x}_t)$ is the (Stein) score function. In practice, the score function is approximated using a neural network $\boldsymbol{\epsilon_\theta} : \mathbb{R}^n \times [0,1] \to \mathbb{R}^n$ parameterized by $\boldsymbol{\theta}$, which can be trained by the denoising score matching technique [26] as

$$\min_{\boldsymbol{\theta}} \mathbb{E}\left[\omega(t)\|\boldsymbol{\epsilon_\theta}(\boldsymbol{x}_t, t) + s_t\sigma_t\nabla_{\boldsymbol{x}_t}\log p_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\|_2^2\right], \quad (2)$$

which can also be written as $\min_{\boldsymbol{\theta}} \mathbb{E}[\omega(t)\|\boldsymbol{\epsilon_\theta}(\boldsymbol{x}_t, t) - \boldsymbol{\epsilon}\|^2] + C$, where the expectation is taken over $t \sim [0,1]$, $\boldsymbol{x}_t \sim p_t(\boldsymbol{x}_t|\boldsymbol{x}_0)$, $\boldsymbol{x}_0 \sim p_{\text{data}}(\boldsymbol{x})$, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Here, C is a constant independent of $\boldsymbol{\theta}$, and $\omega(t)$ is a scalar representing the weight of the loss as a function of t. In DDPM [1], it is simplified to $\omega(t) = 1$. Once the parameterized score function $\boldsymbol{\epsilon_\theta}$ is trained, it can be utilized to approximate the reverse-time SDE using numerical solvers such as Euler-Maruyama.

Timestep clustering methods. Diffusion models have demonstrated exceptional performance but face efficiency challenges in training and sampling. In response, several studies proposed to cluster the timestep range $t \in [0,1]$ into multiple intervals (e.g., $[0, t_1), [t_1, t_2), \ldots, [t_n, 1]$). Notably, Choi et al. [19] reconfigured the loss weights for different intervals to enhance performance. Deja et al. [27] divide the entire process into a denoiesr and a generator based on their functionalities. Balaji et al. [28] introduced "expert denoisers", which proposed using distinct architectures for different time intervals in text-to-image diffusion models. Go et al. [22] further improved the efficiency of these expert denoisers through parameter-efficient fine-tuning and data-free knowledge transfer. Lee et al. [21] designed separate

architectures for each interval based on frequency characteristics. Moreover, Go et al. [20] treated different intervals as distinct tasks and employed multi-task learning strategies for diffusion model training, along with various timestep clustering methods.

Our approach distinguishes itself from the aforementioned methods in two key aspects. The first key component is our tailored U-net architecture using a unified encoder coupled with different decoders for different intervals. Previous models have either adopted a unified architecture, as seen in [19, 20], or employed separate architectures (referred to as expert denoisers) for each interval [21, 22, 28]. In comparison, our multistage architecture surpasses these methodologies, as demonstrated in Sec. 5.3. Second, we developed a new timestep clustering method leveraging a general optimal denoiser (Prop. 1) that showcases superior performance (see Sec. 5.4). In contrast, prior works rely on (i) a simple timestep-based clustering cost function [20–22, 28], (ii) Signal-to-Noise Ratio (SNR) based clustering [20], or (iii) gradients-based partitioning that uses task affinity scores [20].

Reducing the sampling iterations methods. Efforts to improve sampling efficiency of diffusion models have led to many recent advancements in SDE and Ordinary Differential Equation (ODE) samplers [2]. For instance, the Denoising Diffusion Implicit Model (DDIM) [29] formulates the forward diffusion as a non-Markovian process with a deterministic generative path, significantly reducing the number of function evaluations (NFE) required for sampling (from thousands to hundreds). Generalized DDIM (gDDIM) [30] further optimized DDIM by modifying the parameterization of the scoring network. Furthermore, the works in [24] and [31], termed the Diffusion Probabilistic Model solver (DPM-solver) and the Diffusion Exponential Integrator Sampler (DEIS), respectively, introduced fast higher-order solvers, employing exponential integrators that require 10s NFE for comparable generation quality. Moreover, the consistency model [32] introduced a novel training loss and parameterization, achieving high-quality generation with merely 1-2 NFE.

We remark that while the aforementioned methods are indirectly related to our work, our experiments in Sec. 5.1 and Sec. 5.2 show that our approach can be easily integrated into these techniques, further improving diffusion models' overall training and sampling efficiency.

3. Identification of Key Sources of Inefficiency

Conventional diffusion model architectures, as exemplified by [1, 2, 18], treat the training of the diffusion model as a unified process across all timesteps. Recent researches, such as [19–22], have highlighted the benefits of recognizing distinctions between different timesteps and the poten-

tial efficiency gains from treating them as separate tasks during the training process. However, our experimental results demonstrate that both unified and separate architectures suffer inefficiency for training diffusion models, where the inefficiency comes from (i) overparameterization, (ii) gradient dissimilarity, and (iii) overfitting.

3.1. Empirical Observations on the Key Sources of Inefficiency

To illustrate the inefficiency in each interval, we isolate the interval by using a separate architecture from the rest.

Experiment setup. In our experiments, we consider three-stage training and divide the time steps into three intervals: $[0, t_1), [t_1, t_2), [t_2, 1]$[1]. Let $(\epsilon_{\boldsymbol{\theta}})_i^{[a,b]}, 0 \le a < b \le 1$ denote a U-Net architecture with parameter $\boldsymbol{\theta}$ trained with i iterations and fed with data pairs $(\boldsymbol{x}_t, t)$, where $t \in [a, b]$. We then train models using two different strategies: a unified architecture with 108M network parameters for all intervals, i.e., $(\epsilon_{\boldsymbol{\theta}})_i^{[0,1]}$, and separate architectures with varying network parameters (e.g., 47M, 108M, 169M) for each interval; e.g., $(\epsilon_{\boldsymbol{\theta}})_i^{[0,t_1)}$ for the interval $[0, t_1)$, etc. It is worth noting that, apart from the differences in network parameters, we utilize the same network architecture (e.g., U-Net) for both the unified and the separate approaches. We assessed the training progress of each model by evaluating image generation quality at different training iterations. Notably, because some of the models are only trained on one interval, we need to provide a ground truth score for the other intervals. In Fig. 2, the sampling process is shown top and the experimental results are shown bottom.

Inefficiency in unified architectures. From Fig. 2, we observe the following:

- *Overparameterization and underfitting emerge simultaneously for unified architectures*. In Fig. 2a, we observe that increasing the number of parameters in Interval 0 can improve the image generation quality (as indicated by a lower FID score). In contrast, Fig. 2b reveals that increasing the number of parameters in Interval 2 has minimal impact on the quality of image generation. This implies that using a unified architecture will result in underfitting in Interval 0 and overparameterization in Interval 2. The current unified architecture's parameter redundancy leaves significant room for improving its efficiency. To optimize the computational usage, we should allocate more parameters to Interval 0 while allocating fewer parameters to Interval 2.
- *Gradient dissimilarity hinders training for unified architecture*. Quantitative results from [20] demonstrate dissimilarity in gradients caused by differences within each interval. This can also be observed from our results based

upon both Fig. 2a and Fig. 2b. For the unified and separate architectures using the same number of parameters (108M), separate architecture achieves a significantly lower FID with the same training iterations, implying that dissimilar gradients among intervals may hinder training when using a unified architecture. Here, the only difference between training separate and unified architectures is that the batch gradient for unified architecture is calculated based on all timesteps while the batch gradien2 for separate architecture is calculated only from a specific interval.

Inefficiency in existing separate architectures. Although separated architecture [19, 21, 22] better allocates computational resources for each interval, it suffers from overfitting. This can be illustrated based upon training separate architectures (169M) and (108M) in Interval 0 shown in Fig. 2a, where increasing the number of parameters will lead to overfitting. This also happens in Interval 2, when we compare all separate architectures in Fig. 2b. In comparison, the unified networks with 108M parameters are less prone to overfitting for both Interval 0 and Interval 2. This suggests that we can reduce overfitting by training shared weights across different intervals together.

3.2. Tackling the Inefficiency via Multistage U-Net Architectures

In a unified architecture applied across all timesteps, there is often a dual challenge: requirements for more parameters (169M) in the interval $[0, t_1)$ but fewer parameters (47M) in the interval $[t_2, 1]$. This issue is compounded by the gradient dissimilarity across different timesteps, which can impede effective training. Alternatively, employing separate architectures for different intervals might lead to overfitting and a lack of robust early stopping mechanisms. To address these challenges, our proposed multistage architecture in Sec. 4 combines shared parameters to reduce overfitting with interval-specific parameters to mitigate the impact of gradient dissimilarity. This tailored approach for each interval ensures improved adaptability. Furthermore, we conduct an in-depth ablation study in Sec. 5.3 to showcase the effectiveness of our multi-stage architecture over the existing models.

4. Proposed Multistage Framework

In this section, we introduce our new multistage framework (as illustrated in Fig. 1 (c)). Specifically, we first introduce the multi-stage U-Net architecture design in Sec. 4.1, following a new clustering method for choosing the optimal interval to partition the entire timestep $[0, 1]$ into intervals in Sec. 4.2. , and discuss the rationale of the proposed architecture in Sec. 4.3.

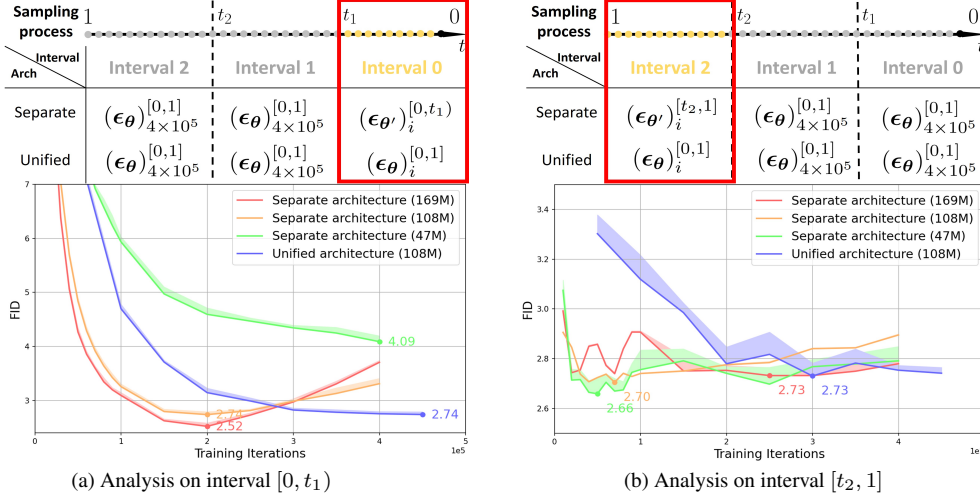[1]Details for interval clustering can be found in Appendix B.1.

Figure 2. **Comparison between separate architecture and unified architecture w.r.t. the image generation quality in different intervals:** (a) analysis on interval $[0, t_1)$; and (b) analysis on interval $[t_2, 1]$. As illustrated on top of each figure, we *only* train separate architectures within specific intervals for the sampling process in both (a) and (b). For the remaining period of sampling, we use a well-trained diffusion model $(\epsilon_\theta)_{4\times10^5}^{[0,1]}$ to approximate the ground truth score function. As shown in the above figure of (a), e.g. for the separate architecture on interval 1, sampling utilizes trained model $(\epsilon_{\theta'})_i^{[0,t_1)}$ for interval 0 and well-trained model $(\epsilon_\theta)_{4\times10^5}^{[0,1]}$ for interval 1 and 2. Notably, for both $(\epsilon_\theta)_i^{[0,1]}$ and $(\epsilon_\theta)_{4\times10^5}^{[0,1]}$, we are using the model with 108M parameters. For separate architecture, the number in the parentheses represents the number of parameters of the model $(\epsilon_{\theta'})_i^{[a,b]}$. For example, for separate architecture (169M) in (a), the model $(\epsilon_{\theta'})_i^{[0,t_1)}$ has 169M parameters for θ'. The bottom figures in (a-b) illustrate the FID of the generation from each architecture under different training iterations.

4.1. Proposed Multi-stage U-Net Architectures

As discussed in Sec. 3, most existing diffusion models either employ a unified architecture across all intervals [1, 2, 18] to share features for all timesteps, or use completely separate architectures for different timestep intervals [21, 22, 28] where the goal is to take advantage of the benign properties within different intervals.

To harness the advantages of both unified and separate architectures employed in prior studies, we introduce a multistage U-Net architecture, as illustrated in Figure 1(c). Specifically, we partition the entire timestep $[0, 1]$ into several intervals, e.g., three intervals $[0, t_1), [t_1, t_2), [t_2, 1]$ in Fig. 1. For the architecture, we introduce:

- **One shared encoder across all time intervals.** For each timestep interval, we implement a shared encoder architecture (plotted in blue in Fig. 1 (c)), which is similar to the architecture employed in the original U-Net framework [33]. Unlike separate architecture, the shared encoder provide shared information across all timesteps, preventing models from overfitting (see Sec. 5.3 for a discussion).
- **Separate decoders for different time intervals.** Motivated by the multi-head structure introduced in the Mask Region-based Convolutional Neural Networks (Mask-RCNN) method [34], we propose to use multiple distinct decoders (plotted in colors for different intervals in Fig. 1(c)), where each decoder is tailored to a spe-

cific timestep interval. The architecture of each decoder closely resembles the one utilized in [2], with deliberate adjustments made to the embedding dimensions to optimize performance.

As we observe, the primary difference in the architecture resides within the decoder structure. Intuitively, we use a decoder with fewer number of parameters for intervals closer to the noise, because the learning task is easier. We use a decoder with a larger number of parameters for intervals closer to the image.

4.2. Optimal Denoiser-based Timestep Clustering

Next, we discuss how we principally choose the interval partition time points in practice. For simplicity, we focus on the case where we partition the time $[0, 1]$ into three intervals $[0, t_1), [t_1, t_2), [t_2, 1]$, and we develop a timestep clustering method to choose the optimal t_1 and t_2. Of course, our method can be generalized to multi-stage networks with arbitrary interval numbers. However, in practice, we find that the choice of three intervals strikes a good balance between effectiveness and complexity; see our ablation study in Appendix B.6.

To partition the time interval, we employ the optimal denoiser established in Proposition 1.

Proposition 1. *Suppose we train a diffusion model denoiser function $\epsilon_\theta(x, t)$ with parameters θ using dataset*

Algorithm 1 Optimal Denoiser based Timestep Clustering

1: **Input**: Total samples K, optimal denoiser function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*(\boldsymbol{x}, t)$, thresholds α, η, dataset p_{data}, $\mathcal{S}_0 = \mathcal{S}_1 = \emptyset$
2: **Output**: Timesteps t_1, t_2
3: **for** $k \in \{1, \ldots, K\}$ **do**
4:    $\boldsymbol{y}_k \sim p_{\text{data}}, \boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \mathbf{I}), t_k \sim [0, 1]$
5:    $\mathcal{S}_0^k \leftarrow \mathcal{D}(\boldsymbol{\epsilon}_{t_k}^*, \boldsymbol{\epsilon}_0^*, \boldsymbol{y}_k, \boldsymbol{\epsilon}_k), \mathcal{S}_1^k \leftarrow \mathcal{D}(\boldsymbol{\epsilon}_{t_k}^*, \boldsymbol{\epsilon}_1^*, \boldsymbol{y}_k, \boldsymbol{\epsilon}_k)$
6:    $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \{(t_k, \mathcal{S}_0^k)\}, \mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{(t_k, \mathcal{S}_1^k)\}$
7: **end for**

8: $t_1 = \arg\max_{\tau} \left\{ \tau \; \middle| \; \frac{\sum_{(t_k, \mathcal{S}_0^k) \in \mathcal{S}_0}[\mathcal{S}_0^k \cdot \mathbb{1}(t_k \leq \tau)]}{\sum_{(t_k, \mathcal{S}_0^k) \in \mathcal{S}_0}[\mathbb{1}(t_k \leq \tau)]} \geq \alpha \right\}$

9: $t_2 = \arg\min_{\tau} \left\{ \tau \; \middle| \; \frac{\sum_{(t_k, \mathcal{S}_1^k) \in \mathcal{S}_0}[\mathcal{S}_1^k \cdot \mathbb{1}(t_k \geq \tau)]}{\sum_{(t_k, \mathcal{S}_1^k) \in \mathcal{S}_0}[\mathbb{1}(t_k \geq \tau)]} \geq \alpha \right\}$

$\{\boldsymbol{y}_i \in \mathbb{R}^n\}_{i=1}^N$ by

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; t) = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{x}_t, \boldsymbol{\epsilon}}[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2], \quad (3)$$

where $\boldsymbol{x}_0 \sim p_{data}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^N \delta(\boldsymbol{x} - \boldsymbol{y}_i)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$, and $\boldsymbol{x}_t \sim p_t(\boldsymbol{x}_t | \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; s_t \boldsymbol{x}_0, s_t^2 \sigma_t^2 \boldsymbol{I})$ with perturbation parameters s_t and σ_t defined in Eq. (1). Then, the optimal denoiser at t, defined as $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*(\boldsymbol{x}; t) = \arg\min_{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}} \mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; t)$, is given by

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*(\boldsymbol{x}; t) = \frac{1}{s_t \sigma_t} \left[\boldsymbol{x} - s_t \frac{\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t \boldsymbol{y}_i, s_t^2 \sigma_t^2 \boldsymbol{I}) \boldsymbol{y}_i}{\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t \boldsymbol{y}_i, s_t^2 \sigma_t^2 \boldsymbol{I})} \right]. \tag{4}$$

The proof is provided in Appendix A, and the result can be generalized from recent work of Karras et al. [18], extending from a specific kernel $p_t(\boldsymbol{x}_t | \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \boldsymbol{x}_0, \sigma_t^2 \boldsymbol{I})$ to encompassing a broader scope of noise perturbation kernels, given by $p_t(\boldsymbol{x}_t | \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; s_t \boldsymbol{x}_0, s_t^2 \sigma_t^2 \boldsymbol{I})$. For brevity, we simplify the notation of the optimal denoiser $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*(\boldsymbol{x}, t)$ in Proposition 1 as $\boldsymbol{\epsilon}_t^*(\boldsymbol{x})$.

To obtain the optimal interval, our rationale is to *homogenize* the regression task as much as possible within each individual time interval. To achieve this goal, given sampled $\boldsymbol{x}_0$, $\boldsymbol{\epsilon}$, we define the function distance of the optimal denoiser at any given timestep t_a, t_b as:

$$\mathcal{D}(\boldsymbol{\epsilon}_{t_a}^*, \boldsymbol{\epsilon}_{t_b}^*, \boldsymbol{x}_0, \boldsymbol{\epsilon}) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}\left(|\boldsymbol{\epsilon}_{t_a}^*(\boldsymbol{x}_{t_a}) - \boldsymbol{\epsilon}_{t_b}^*(\boldsymbol{x}_{t_b})|_i \leq \eta\right),$$

where $\mathbb{1}(\cdot)$ is the indicator function, η is a pre-specified threshold, $\boldsymbol{x}_{t_a} = s_{t_a} \boldsymbol{x}_0 + s_{t_a} \sigma_{t_a} \boldsymbol{\epsilon}$, and $\boldsymbol{x}_{t_b} = s_{t_b} \boldsymbol{x}_0 + s_{t_b} \sigma_{t_b} \boldsymbol{\epsilon}$. Consequently, we define the functional similarity of the optimal denoiser at timesteps t_a and t_b as:

$$\mathcal{S}(\boldsymbol{\epsilon}_{t_a}^*, \boldsymbol{\epsilon}_{t_b}^*) = \mathbb{E}_{\boldsymbol{x}_0 \sim p_{\text{data}}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})}[\mathcal{D}(\boldsymbol{\epsilon}_{t_a}^*, \boldsymbol{\epsilon}_{t_b}^*, \boldsymbol{x}_0, \boldsymbol{\epsilon})]. \tag{5}$$

Based upon the definition, we design the following optimization problem to find the largest t_1 and smallest t_2 as:

$$t_1 \leftarrow \arg\max_{\tau} \left\{ \tau \middle| \mathbb{E}_{t \sim [0, \tau)}[\mathcal{S}(\boldsymbol{\epsilon}_t^*, \boldsymbol{\epsilon}_0^*)] \geq \alpha \right\}, \quad (6)$$

$$t_2 \leftarrow \arg\min_{\tau} \left\{ \tau \middle| \mathbb{E}_{t \sim [\tau, 1]}[\mathcal{S}(\boldsymbol{\epsilon}_t^*, \boldsymbol{\epsilon}_1^*)] \geq \alpha \right\}, \quad (7)$$

such that the average functional similarity of $\boldsymbol{\epsilon}_t^*$ (resp. $\boldsymbol{\epsilon}_t^*$) to $\boldsymbol{\epsilon}_0^*$ (resp. $\boldsymbol{\epsilon}_1^*$) in $[0, t_1)$ (resp. $[t_2, 1]$) is larger than or equal to a pre-defined threshold α. As the above optimization problems are intractable, we propose the procedure outlined in Algorithm 1 to obtain an approximate solution. In particular, the algorithm samples K pairs $(\boldsymbol{y}_k, \boldsymbol{\epsilon}_k, t_k), k \in \{1, \ldots, K\}$ to calculate the distances $\mathcal{D}(\boldsymbol{\epsilon}_{t_k}^*, \boldsymbol{\epsilon}_0^*, \boldsymbol{y}_k, \boldsymbol{\epsilon}_k)$ and $\mathcal{D}(\boldsymbol{\epsilon}_{t_k}^*, \boldsymbol{\epsilon}_1^*, \boldsymbol{y}_k, \boldsymbol{\epsilon}_k)$ (step 6). Based on those distances, we solve the optimization problems defined in the lines 8 and 9 of Algorithm 1 to obtain t_1 and t_2.

4.3. Rationales for the proposed architecture

Finally, we summarize the rationales of our proposed architecture based on the empirical observations in Sec. 3 and previous works.

Rationales for the shared encoder. (i) *Prevent overfitting:* If we treat training separate stages of diffusion models as multi-task learning, [35] suggests that shared parameters across tasks in multi-task learning can mitigate overfitting. (ii) *Maintain consistency in h-space.* The output of the encoder for UNet is named as *h-space* [36], which has properties for semantic manipulation such as homogeneity, linearity, robustness, and consistency across timesteps. Therefore, sharing the encoder can maintain better consistency of the *h-space* among all timesteps compared to separate encoders.

Rationales for network parameter design. First, we provide some intuitions as to why the learning task at $t = 0$ is harder than that at $t = 1$. Suppose diffusion models could converge to the optimal denoiser $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*(\boldsymbol{x}_t, t)$ given in Eq. (4). Based on this, we observe: (i) when $t \rightarrow 0$, we have $\boldsymbol{x}_{t \rightarrow 0} = \boldsymbol{x}_0$, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*|_{t \rightarrow 0} = \boldsymbol{\epsilon}$, so that $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*$ is a *complicated* mapping from training data distribution $p_{\text{data}}(\boldsymbol{x})$ to the gaussian distribution; (ii) when $t \rightarrow 1$, we have $\boldsymbol{x}_{t \rightarrow 1} = \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*|_{t \rightarrow 1} = \boldsymbol{\epsilon}$, so $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}^*$ is an identity mapping. The two extreme cases reveal that the identity mapping for $t \rightarrow 1$ (close to noise) is easier to learn than that for $t \rightarrow 0$. Second, our choice of network parameters across different stages share similar spirits with the recent work [37], which employs a high dimensional subspace as $t \rightarrow 0$ and gradually decreases the dimensionality of each subspace until $t = 1$.

5. Experiments

In this section, we start by providing the experimental setups. Next, we present results in terms of generation quality (Sec. 5.1). Subsequently, results for training and sampling efficiency are presented. Finally, in Sec. 5.3 and

Sec. 5.4, comprehensive ablation studies on timestep clustering methods and multistage architectures are given.

Multistage architectures. Our multistage architecture, inspired by the U-Net model [33] used in DDPM++ [1, 2, 18], is modified for interval-specific channel dimensions. The proposed architecture is adopted to three diffusion models: DPM-Solver [24], EDM [18], and LDM [8]. In particular, for the cases of DPM-Solver and EDM, the encoder's channel dimensions are standardized at 128, while the decoders are configured with 192, 128, and 16 channels for intervals $[0, t_1)$, $[t_1, t_2)$, and $[t_2, 1]$, respectively. In the LDM case, we use 224 channels across the encoder for all intervals whereas the decoders are configured with 256, 192, and 128 channels for the respective intervals. To decide the specific number of parameters for each decoder, we apply ablation studies in Appendix B.5. Training details are in Appendix D.

Datasets, evaluation metrics, & baselines. We use CIFAR-10 (32×32), CelebA (32×32), and CelebA (256×256) datasets for our experiments. To evaluate the performance of our multistage diffusion model in terms of the generation quality, we use the standard Fréchet Inception Distance (FID) metric [38]. We assess the sampling efficiency using the NFE, and giga-floating point operations (GFLOPs) per function evaluation. For both separate architecture and our multistage architecture, equivalent GFLOPs are computed as a weighted summation of GFLOPs for each interval. Training efficiency is evaluated using total training iterations multiplied by the GFLOPs per function evaluation,[2] measured by peta-floating point operations (PFLOPs). For baselines, we consider DDPM [1], Score SDE [2], Poisson Flow Generative Models (PFGM) [39], DDIM [29], gDDIM [30], DEIS [31], DPM-solver [24], and EDM [18].

5.1. Image Generation Quality Results

In this subsection, we demonstrate the effectiveness of our approach by comparing the image generation quality (measured by FID) with comparable training and sampling computations (measured by NFE). Specifically, Tab. 1 presents FID scores to measure the sampling quality, and NFE to measure the number of sampling iterations required using the CIFAR-10 dataset. Our method is compared to 8 baselines. As observed, our multistage DPM-Solver outperforms DPM-Solver in terms of the reported FID values while requiring similar training iterations (both are 4.5×10^5) and model GFLOPs (18.65 for multistage DPM-Solver versus 17.65 for DPM-Solver). A similar observation holds when we compare our multistage EDM and the vanilla EDM, where we reduce FID from 2.05 to **1.96** by using the multi-stage architecture. Remarkably, utilizing only 20 NFE, our Multistage DPM-Solver returns the same FID score as the one reported for the PFGM method, which requires 147 NFEs. These results also highlight the adaptability of our framework to higher-order ODE solvers; see the 8th and last row of Tab. 1.

Table 1. Sampling quality on CIFAR-10 Dataset.

METHOD	NFE($\downarrow$)	FID($\downarrow$)
DDPM	1000	3.17
Score SDE	2000	2.20
PFGM	147	2.35
DDIM	100	4.16
gDDIM	20	2.97
DEIS	20	2.86
DPM-solver	20	2.73
Multistage DPM-solver (Ours)	20	**2.35**
EDM	35	2.05
Multistage EDM (Ours)	35	**1.96**

5.2. Training & Sampling Efficiency Results

In this subsection, we further demonstrate the superiority of our method by comparing the training and sampling efficiency under comparable image generation quality. Specifically, in Tab. 2, we present the number of training iterations, GFLOPs, and total training PFLOPs of our approach, DPM-solver, EDM, and LDM using CIFAR-10 and CelebA datasets. Using the CIFAR-10 dataset, our multistage DPM-solver achieves similar FID scores (2.71 vs 2.73) while requiring nearly half the training iterations when compared to the vanilla DPM-solver. For the case of EDM (resp. LDM), our approach returns an FID score of 1.44 (resp. 8.29), requiring 1.4×10^5 (3.2×10^5) less iterations when compared to vanilla DPM-solver (resp. LDM). For the cases of DPM-solver and EDM, we can achieve a substantial reduction of training iterations, which is demonstrated by a marginal increase in the number of GFLOPs. For the LDM case, we also achieve a significant reduction of both training iterations and GFLOPs. These promising results highlight the significantly improved computational efficiency achieved by using the proposed multistage framework.

5.3. Comparison of Different Architectures

In Sec. 3, we highlighted the limitations of both unified and separate diffusion model architectures in terms of training efficiency (see Fig. 2). In this part, we further illustrate these limitations through extensive experiments as shown

---

[2]Here we simplify it by ignoring the FLOPs for backward propagation, which is approximately proportional to FLOPs of forward evaluation.

Table 2. Training and Sampling Efficiency on More Datasets.

Dataset	Method	Training Iterations($\downarrow$)	GFLOPs($\downarrow$)	Total Training PFLOPs($\downarrow$)	FID($\downarrow$)
CIFAR-10	DPM-Solver [18]	4.5×10^5	**17.65**	7.94	2.73
32×32	Multistage DPM-Solver (Ours)	**2.5×10^5(56%)**	18.65 (106%)	**4.66 (59%)**	2.71
CelebA	EDM [18]	5.7×10^5	**17.65**	10.06	1.55
32×32	Multistage EDM (Ours)	**4.3×10^5(75%)**	19.25 (109%)	**8.28 (82%)**	1.44
CelebA	LDM [8]	4.9×10^5	88.39	43.31	8.29
256×256	Multistage LDM (Ours)	**1.7×10^5(35%)**	**76.19 (86%)**	**12.95 (30.0%)**	8.38

in Tab. 3. Here, we use the U-Net architecture, trained on the CIFAR-10 dataset, and utilize the DPM-Solver for sampling. For the unified case, we use a single U-Net model with 128 channels. For the separate case, three distinct U-Nets with 128 channels are used. For improved performance of the separate architecture, we implement two techniques: early stopping (ES) and tailored parameters (TP) to tackle the overfitting and parameter inefficiency discussed in Sec. 3. Under ES, the criteria is to stop training prior to overfitting. For TP, the three U-Nets are configured with 192, 128, and 16 channels decoders for Intervals 0, 1, and 2, respectively.

Our comparison and analysis in Tab. 3 reveal notable insights of our network design. Comparisons between the 2nd and 3rd rows (and between the 4th and 5th rows) on the separate architectures indicate that early stopping effectively mitigates overfitting and enhances generation quality. When comparing the 2nd and 4th rows (as well as the 3rd and 5th rows) on the separate architectures, we observe that optimizing parameter usage can achieve a significant decrease in FID under comparable GFLOPs. Most importantly, our multistage architecture, as shown in the 6th row, benefits from both unified and separate architectures, achieving the best FID (2.35, compared to 2.73 and 2.52). Comparing the 2nd row and the 4th row, the shared encoder not only prevents overfitting but also improves the convergence of the diffusion model as reported by the FID scores.

Table 3. Ablation study on different diffusion model architectures.

Method	GFLOPs	FID($\downarrow$)
Unified	17.65	2.73
Separate	17.65	2.87
Separate (+ ES)	17.65	2.80
Separate (+ TP)	18.65	2.68
Separate (+ ES, TP)	18.65	2.52
Multistage (Ours)	18.65	**2.35**

5.4. Comparison of Timestep Clustering Methods

As previously stated in Sec. 2, various timestep clustering methods are proposed including timestep-based, SNR-based, and gradient-based clustering approaches [20, 21]. In this subsection, we conduct an experiment to demonstrate the superiority of our clustering method compared to previous arts. Specifically, we apply the clustering methods in [20, 21] to partition the interval along with our proposed multistage UNet architecture. The computed intervals are shown in the Tab. 4. We use the multistage DPM-Solver with these different intervals trained on the CIFAR-10 dataset. As observed, our optimal denoiser-based clustering method achieves the highest FID score, consistently outperforming all other clustering methods.

Table 4. Ablation study on different clustering methods.

Clustering Method	t_1	t_2	FID($\downarrow$)
Timestep [20, 21]	0.330	0.670	3.12
SNR [20]	0.348	0.709	2.72
Gradient [20]	0.360	0.660	2.75
Optimal Denoiser	0.442	0.631	**2.35**

6. Conclusion & Future Work

In this paper, we introduced a novel multi-stage framework for diffusion models (DM) to improve the training and sampling efficiency. We proposed an algorithm that divides the timestep into several stages. Based on these stages, we designed a stage-specific multi-decoder U-net architecture and a shared encoder across all stages. We conducted thorough numerical experiments with several SOTA diffusion model frameworks and confirmed the effectiveness of our strategy using small scale and large scale datasets.

In future research, it would be interesting to expand our multi-stage approach beyond unconditional diffusion models by considering conditional DMs and DM-based inverse problems solvers. Our experiment in Sec. 5.2 demonstrate that training latent diffusion models within our multi-stage framework requires only 30% of the computational effort needed for training standard latent diffusion models on the CelebA dataset. Thus, employing a multi-stage strategy could significantly reduce the computational demands for training large-scale stable diffusion models, such as those described in [8], which typically requires significant computations.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 2, 3, 5, 7

[2] 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. 1, 2, 3, 5, 7

[3] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. 1

[4] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 1

[5] Xuan Su, Jiaming Song, Chenlin Meng, and Stefano Ermon. Dual diffusion implicit bridges for image-to-image translation. *arXiv preprint arXiv:2203.08382*, 2022. 1

[6] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022.

[7] Min Zhao, Fan Bao, Chongxuan Li, and Jun Zhu. Egsde: Unpaired image-to-image translation via energy-guided stochastic differential equations. *Advances in Neural Information Processing Systems*, 35:3609–3623, 2022. 1

[8] 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. 1, 2, 7, 8

[9] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[10] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021. 1

[11] Bowen Song, Soo Min Kwon, Zecheng Zhang, Xinyu Hu, Qing Qu, and Liyue Shen. Solving inverse problems with latent diffusion models via hard data consistency. *arXiv preprint arXiv:2307.08123*, 2023. 1

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

[13] Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. Solving inverse problems in medical imaging with score-based generative models. *arXiv preprint arXiv:2111.08005*, 2021.

[14] Ismail Alkhouri, Shijun Liang, Rongrong Wang, Qing Qu, and Saiprasad Ravishankar. Diffusion-based adversarial purification for robust deep mri reconstruction. *arXiv preprint arXiv:2309.05794*, 2023. 1

[15] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022. 2

[16] William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Weilbach, and Frank Wood. Flexible diffusion modeling of long videos. *Advances in Neural Information Processing Systems*, 35:27953–27965, 2022. 2

[17] Huijie Zhang, Jinfan Zhou, Yifu Lu, Minzhe Guo, Liyue Shen, and Qing Qu. The emergence of reproducibility and consistency in diffusion models. *arXiv preprint arXiv:2310.05264*, 2023. 2

[18] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *arXiv preprint arXiv:2206.00364*, 2022. 2, 3, 5, 6, 7, 8, 1

[19] Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. Perception prioritized training of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11472–11481, 2022. 2, 3, 4

[20] Hyojun Go, JinYoung Kim, Yunsung Lee, Seunghyun Lee, Shinhyeok Oh, Hyeongdon Moon, and Seungtaek Choi. Addressing negative transfer in diffusion models. *arXiv preprint arXiv:2306.00354*, 2023. 3, 4, 8, 2

[21] Yunsung Lee, Jin-Young Kim, Hyojun Go, Myeongho Jeong, Shinhyeok Oh, and Seungtaek Choi. Multi-architecture multi-expert diffusion models. *arXiv preprint arXiv:2306.04990*, 2023. 3, 4, 5, 8, 2

[22] Hyojun Go, Yunsung Lee, Jin-Young Kim, Seunghyun Lee, Myeongho Jeong, Hyun Seung Lee, and Seungtaek Choi. Towards practical plug-and-play diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1962–1971, 2023. 2, 3, 4, 5

[23] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 1

[24] 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. *arXiv preprint arXiv:2206.00927*, 2022. 2, 3, 7

[25] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 2, 1

[26] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011. 3

[27] Kamil Deja, Anna Kuzina, Tomasz Trzcinski, and Jakub Tomczak. On analyzing generative and denoising capabilities of diffusion-based deep generative models. *Advances in Neural Information Processing Systems*, 35:26218–26229, 2022. 3

[28] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022. 3, 5

[29] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 3, 7, 2

[30] Qinsheng Zhang, Molei Tao, and Yongxin Chen. gddim: Generalized denoising diffusion implicit models. *arXiv preprint arXiv:2206.05564*, 2022. 3, 7

[31] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022. 3, 7

[32] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023. 3

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 5, 7

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 5

[35] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 6

[36] Mingi Kwon, Jaeseok Jeong, and Youngjung Uh. Diffusion models already have a semantic latent space. *arXiv preprint arXiv:2210.10960*, 2022. 6

[37] Bowen Jing, Gabriele Corso, Renato Berlinghieri, and Tommi Jaakkola. Subspace diffusion generative models. In *European Conference on Computer Vision*, pages 274–289. Springer, 2022. 6

[38] 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. 7

[39] Yilun Xu, Ziming Liu, Max Tegmark, and Tommi Jaakkola. Poisson flow generative models. *Advances in Neural Information Processing Systems*, 35:16782–16795, 2022. 7

[40] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 1

[41] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019. 2

[42] Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. High-fidelity performance metrics for generative models in pytorch, 2020. URL https://github.com/toshas/torch-fidelity. Version: 0.2.0, DOI: 10.5281/zenodo.3786540. 2