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

Huijie Zhang\textsuperscript{1,} Yifu Lu\textsuperscript{1,} Ismail Alkhouri\textsuperscript{1,2} Saiprasad Ravishankar\textsuperscript{2,3} Dogyoon Song\textsuperscript{1} Qing Qu\textsuperscript{1}

\textsuperscript{1}Department of Electrical Engineering & Computer Science, University of Michigan
\textsuperscript{2}Department of Computational Mathematics, Science & Engineering, Michigan State University
\textsuperscript{3}Department of Biomedical Engineering, Michigan State University

\{huijiezh, yifulu, ismailal, dogyoons, qingqu\}@umich.edu\quad 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 \cite{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 \cite{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 \cite{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 \cite{1,2,18} (Figure 1 (a)). Moreover, compared to previous approaches that completely separate architectures for each sub-interval \cite{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 \cite{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 experiments (Section 5), we show that our 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 \cite{23}, our method improves the FID for DPM-Solver \cite{24} from 2.84 to 2.37, and it improves the FID for EDM \cite{18} from 2.05 (our training result) to 1.96. Moreover, on the CelebA dataset \cite{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) \cite{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 multi-stage approach.

### 2. Preliminaries & Related Work

In this section, we start by reviewing the basic fundamentals of diffusion models \cite{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 $x_0$ to a noisy version $x_{t \in [0,1]}$ via corrupting with the Gaussian kernel. This process can be formulated as a stochastic differential equation (SDE) [2] of the form $dx = x_t f(t)dt + g(t)d\xi$, where $f(t)$ and $g(t)$ are the drift and diffusion coefficients, respectively, that correspond to a pre-defined noise schedule. $\xi \in \mathbb{R}^n$ is the standard Wiener process. Under the forward SDE, the perturbation kernel is given by the conditional distribution defined as $p_t(x_t|x_0) = \mathcal{N}(x_t; s_t x_0, s_t^2 \sigma_t^2 I)$, where

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

The parameters $s_t$ and $\sigma_t$ 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 $dx = [f(t)x_t - g^2(t)\nabla x_t \log p_t(x_t)] dt + g(t)d\xi$, where $\xi \in \mathbb{R}^n$ is the standard Wiener process running backward in time, and $\nabla x_t \log p_t(x_t)$ is the (Stein) score function. In practice, the score function is approximated using a neural network $\theta : \mathbb{R}^n \times [0,1] \to \mathbb{R}^n$ parameterized by $\theta$, which can be trained by the denoising score matching technique [26] as

$$\min_{\theta} \mathbb{E} \left[ \omega(t) \| \phi_{\theta}(x_t, t) + s_t \sigma_t \nabla x_t \log p_t(x_t|x_0) \|_2^2 \right], \quad (2)$$

which can also be written as $\min_{\theta} \mathbb{E} [\omega(t) \| \phi_{\theta}(x_t, t) - \epsilon \|_2^2] + C$, where the expectation is taken over $t \sim [0,1]$, $x_t \sim p_t(x_t|x_0)$, $x_0 \sim p_{data}(x)$, and $\epsilon \sim \mathcal{N}(0, I)$. Here, $C$ is a constant independent of $\theta$, 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 $\phi_{\theta}$ 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., $[0, t_1], [t_1, t_2], \ldots, [t_m, 1]$). 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 denoiser 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 $(\epsilon_{\theta})_{i,a,b}, 0 \leq a < b \leq 1$ denote a U-Net architecture with parameter $\theta$ trained with $i$ iterations and fed with data pairs $(x_t, t)$, 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., $(\epsilon_{\theta})_{i,0,1}$, and separate architectures with varying network parameters (e.g., 47M, 108M, 169M) for each interval; e.g., $(\epsilon_{\theta})_{i,0,t_1}$ 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 this section, we introduce our new multistage framework.

**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.
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 interval 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 provides 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 specific 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 points in practice. For simplicity, we focus on the case where we partition the time \([0, 1]\) into three intervals \([0, t_1), [t_1, t_2), [t_2, 1]\). 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 \(\theta\) using dataset...
Algorithm 1 Optimal Denoiser based Timestep Clustering

1: **Input:** Total samples $K$, optimal denoiser function $\epsilon_\theta(x,t)$, thresholds $\alpha$, $\eta$, dataset $p_{\text{data}}$, $S_0 = S_1 = \emptyset$
2: **Output:** Timesteps $t_1$, $t_2$
3: for $k \in \{1, \ldots, K\}$ do
4: $y_k \sim p_{\text{data}}, \epsilon_k \sim \mathcal{N}(0,1), t_k \sim [0,1]$
5: $S_k^t \leftarrow D(\epsilon_\theta(x_0, \epsilon_0)), S_k^t \leftarrow D(\epsilon_\theta(x_1, \epsilon_1, y_k, \epsilon_k))$
6: $S_0 \leftarrow S_0 \cup \{(t_k, S_k^t)\}$, $S_1 \leftarrow S_1 \cup \{(t_k, S_k^t)\}$
7: end for
8: $t_1 = \arg \max_\tau \left\{ \tau \left| \sum_{(tk, sk^t) \in S_0} |sk^t_0 - \tau| \right| \geq \alpha \right\}$
9: $t_2 = \arg \min_\tau \left\{ \tau \left| \sum_{(tk, sk^t) \in S_1} |sk^t_1 - \tau| \right| \geq \alpha \right\}$

\[
\{y_i \in \mathbb{R}^n\}_{i=1}^N \text{ by minimizing } \min_{\theta} \mathcal{L}(\epsilon_\theta; t) = \mathbb{E}_{x_0, x_1, \epsilon} \left[ ||\epsilon - \epsilon_\theta(x_1, t)|^2 \right] \tag{3}
\]

where $x_0 \sim p_{\text{data}}(x) = \frac{1}{N} \sum_{i=1}^N \delta(x - y_i), \epsilon \sim \mathcal{N}(0,1)$, and $x_1 \sim p_l(x_1|x_0) = \mathcal{N}(x_1; s_1 x_0, s_1^2 \sigma_l^2 I)$ with perturbation parameters $s_1$ and $\sigma_l$ defined in Eq. (1). Then, the optimal denoiser at $t$, defined as $\epsilon_\theta(x; t) = \arg \min_{\epsilon_\theta} \mathcal{L}(\epsilon_\theta; t)$, is given by

\[
\epsilon_\theta(x; t) = \frac{1}{s_1 \sigma_l} \left[ x - s_1 \sum_{i=1}^N \mathcal{N}(x; s_i y_i, s_i^2 \sigma_l^2 I) + s_1 \sum_{i=1}^N \mathcal{N}(x; s_i y_i, s_i^2 \sigma_l^2 I) \right] \tag{4}
\]

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 $p_l(x_1|x_0) = \mathcal{N}(x_1; s_1 x_0, s_1^2 \sigma_l^2 I)$ to encompassing a broader scope of noise perturbation kernels, given by $p_l(x_1|x_0) = \mathcal{N}(x_1; s_1 x_0, s_1^2 \sigma_l^2 I)$. For brevity, we simplify the notation of the optimal denoiser $\epsilon_\theta(x; t)$ in Proposition 1 as $\epsilon_\theta(t)$. 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 $x_0, \epsilon$, we define the function distance of the optimal denoiser at any given timestep $t_{\alpha}$, $t_{\beta}$ as:

\[
D(\epsilon^*_{t_{\alpha}}, \epsilon^*_{t_{\beta}}, x_0, \epsilon) = \frac{1}{n} \sum_{i=1}^n I(\epsilon^*_{t_{\alpha}}(x_1_{\alpha}) - \epsilon^*_{t_{\beta}}(x_1_{\beta})) \geq \eta, \tag{5}
\]

where $I(\cdot)$ is the indicator function, $\eta$ is a pre-specified threshold, $x_{t_{\alpha}} = s_{t_{\alpha}} x_0 + s_{t_{\alpha}} \sigma_{t_{\alpha}} \epsilon$, and $x_{t_{\beta}} = s_{t_{\beta}} x_0 + s_{t_{\beta}} \sigma_{t_{\beta}} \epsilon$. Consequently, we define the functional similarity of the optimal denoiser at timesteps $t_{\alpha}$ and $t_{\beta}$ as:

\[
S(\epsilon^*_{t_{\alpha}}, \epsilon^*_{t_{\beta}}) = \mathbb{E}_{x_0 \sim p_{\text{data}}, \epsilon \sim \mathcal{N}(0,1)} [D(\epsilon^*_{t_{\alpha}}, \epsilon^*_{t_{\beta}}, x_0, \epsilon)] \tag{6}
\]

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

\[
t_1 \leftarrow \arg \max_\tau \left\{ \tau \left| \mathbb{E}_{t_{\tau} \sim [0,1]} [S(\epsilon^*_{t_{\tau}}, \epsilon^*_{t_{\tau}})] \geq \alpha \right\} \tag{6}
\]

\[
t_2 \leftarrow \arg \min_\tau \left\{ \tau \left| \mathbb{E}_{t_{\tau} \sim [t_1,1]} [S(\epsilon^*_{t_{\tau}}, \epsilon^*_{t_{\tau}})] \geq \alpha \right\} \tag{7}
\]

such that the average functional similarity of $\epsilon^*_{t_{\alpha}}$ (resp. $\epsilon^*_{t_{\beta}}$) to $\epsilon^*_{t_{\alpha}}$ (resp. $\epsilon^*_{t_{\beta}}$) in $[0, t_1]$ (resp. $[t_2, 1]$) is larger than or equal to a pre-defined threshold $\alpha$. 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 $(y_k, \epsilon_k, t_k), k \in \{1, \ldots, K\}$ to calculate the distances $D(\epsilon^*_{t_{\alpha}}, \epsilon^*_{t_{\alpha}}, y_k, \epsilon_k)$ and $D(\epsilon^*_{t_{\beta}}, \epsilon^*_{t_{\beta}}, y_k, \epsilon_k)$ (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 $\epsilon^*_0(x_1, t)$ given in Eq. (4). Based on this, we observe: (i) when $t \rightarrow 0$, we have $x_{t \rightarrow 1} = x_0, \epsilon^*_0(x_1, 0) = \epsilon$, so that $\epsilon^*_0$ is a complicated mapping from training data distribution $p_{\text{data}}(x)$ to the gaussian distribution; (ii) when $t \rightarrow 1$, we have $x_{t \rightarrow 1} = \epsilon, \epsilon^*_0(x_1, t) = \epsilon$, so $\epsilon^*_0$ 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 for 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
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 Frechet 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, 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 $4.5 \times 10^5$) 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>
<thead>
<tr>
<th>METHOD</th>
<th>NFE ↓</th>
<th>FID ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDPM</td>
<td>1000</td>
<td>3.17</td>
</tr>
<tr>
<td>Score SDE</td>
<td>2000</td>
<td>2.20</td>
</tr>
<tr>
<td>PFGM</td>
<td>147</td>
<td>2.35</td>
</tr>
<tr>
<td>DDIM</td>
<td>100</td>
<td>4.16</td>
</tr>
<tr>
<td>gDDIM</td>
<td>20</td>
<td>2.97</td>
</tr>
<tr>
<td>DEIS</td>
<td>20</td>
<td>2.86</td>
</tr>
<tr>
<td>DPM-solver</td>
<td>20</td>
<td>2.73</td>
</tr>
<tr>
<td>Multistage DPM-solver (Ours)</td>
<td>20</td>
<td>2.35</td>
</tr>
<tr>
<td>EDM</td>
<td>35</td>
<td>2.05</td>
</tr>
<tr>
<td>Multistage EDM (Ours)</td>
<td>35</td>
<td>1.96</td>
</tr>
</tbody>
</table>

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 $1.4 \times 10^5$ ($3.2 \times 10^5$) 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
Table 2. Training and Sampling Efficiency on More Datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Training Iterations$($\downarrow$)$</th>
<th>GFLOPs$($\downarrow$)$</th>
<th>Total Training GFLOPs($\downarrow$)</th>
<th>FID($\downarrow$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>DPM-Solver [18]</td>
<td>$4.5 \times 10^4$</td>
<td>17.65</td>
<td>7.94</td>
<td>2.73</td>
</tr>
<tr>
<td>$32 \times 32$</td>
<td>Multistage EDM (Ours)</td>
<td>$4.3 \times 10^4$ ($75%$)</td>
<td>19.25 ($109%$)</td>
<td>8.28 ($82%$)</td>
<td>1.44</td>
</tr>
<tr>
<td>CelebA</td>
<td>EDM [18]</td>
<td>$5.7 \times 10^4$</td>
<td>17.65</td>
<td>10.06</td>
<td>1.55</td>
</tr>
<tr>
<td>$32 \times 32$</td>
<td>Multistage EDM (Ours)</td>
<td>$4.3 \times 10^4$ ($75%$)</td>
<td>19.25 ($109%$)</td>
<td>8.28 ($82%$)</td>
<td>1.44</td>
</tr>
<tr>
<td>CelebA</td>
<td>LDM [8]</td>
<td>$4.3 \times 10^4$</td>
<td>88.39</td>
<td>43.31</td>
<td>8.29</td>
</tr>
<tr>
<td>256 $\times$ 256</td>
<td>Multistage LDM (Ours)</td>
<td>$1.7 \times 10^5$ ($35%$)</td>
<td>76.19 ($86%$)</td>
<td>12.95 ($30.0%$)</td>
<td>8.38</td>
</tr>
</tbody>
</table>

Table 4. Ablation study on different clustering methods.

<table>
<thead>
<tr>
<th>Clustering Method</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>FID($\downarrow$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestep [20, 21]</td>
<td>0.330</td>
<td>0.670</td>
<td>3.12</td>
</tr>
<tr>
<td>SNR [20]</td>
<td>0.348</td>
<td>0.709</td>
<td>2.72</td>
</tr>
<tr>
<td>Gradient [20]</td>
<td>0.360</td>
<td>0.660</td>
<td>2.75</td>
</tr>
<tr>
<td>Optimal Denoiser</td>
<td>0.442</td>
<td>0.631</td>
<td>2.35</td>
</tr>
</tbody>
</table>

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.

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 U-Net 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 3. Ablation study on different diffusion model architectures.

<table>
<thead>
<tr>
<th>Method</th>
<th>GFLOPs</th>
<th>FID($\downarrow$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified</td>
<td>17.65</td>
<td>2.73</td>
</tr>
<tr>
<td>Separate</td>
<td>17.65</td>
<td>2.87</td>
</tr>
<tr>
<td>Separate (+ ES)</td>
<td>17.65</td>
<td>2.80</td>
</tr>
<tr>
<td>Separate (+ TP)</td>
<td>18.65</td>
<td>2.68</td>
</tr>
<tr>
<td>Separate (+ ES, TP)</td>
<td>18.65</td>
<td>2.52</td>
</tr>
<tr>
<td><strong>Multistage (Ours)</strong></td>
<td><strong>18.65</strong></td>
<td><strong>2.35</strong></td>
</tr>
</tbody>
</table>

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.
References


[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


