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

Supplementary Material

A. Proof of Proposition 1

This section presents the proof of Prop. 1 in Sec. 4.

Given dataset $\{\boldsymbol{y}_i\}_{i=1}^N$ with N images, we approximate the original dataset distribution by using multi-Dirac distribution, $p_{\text{data}}(\boldsymbol{x}) := \frac{1}{N}\sum_{i=1}^N \delta(\boldsymbol{x}-\boldsymbol{y}_i)$. Then, the distribution of the perturbed image $\boldsymbol{x}$ at random timestep t can be calculated as:

$$p_t(\boldsymbol{x}) = \int_{\mathbb{R}^d} p_t(\boldsymbol{x}|\boldsymbol{x}_0)p_{\text{data}}(\boldsymbol{x}_0)\mathrm{d}\boldsymbol{x}_0 \tag{8}$$

$$= \int_{\mathbb{R}^d} p_{\text{data}}(\boldsymbol{x}_0)\mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{x}_0, s_t^2\sigma_t^2\mathbf{I})\mathrm{d}\boldsymbol{x}_0 \tag{9}$$

$$= \int_{\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^N \delta(\boldsymbol{x}_0-\boldsymbol{y}_i)\mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{x}_0, s_t^2\sigma_t^2\mathbf{I})\mathrm{d}\boldsymbol{x}_0 \tag{10}$$

$$= \frac{1}{N}\sum_{i=1}^N \int_{\mathbb{R}^d} \delta(\boldsymbol{x}_0-\boldsymbol{y}_i)\mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{x}_0, s_t^2\sigma_t^2\mathbf{I})\mathrm{d}\boldsymbol{x}_0 \tag{11}$$

$$= \frac{1}{N}\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})\ . \tag{12}$$

Let the noise prediction loss, which is generally used across various diffusion models, be

$$\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; t) = \mathbb{E}_{\boldsymbol{x}\sim p_t(\boldsymbol{x})}[||\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x},t)||^2] \tag{13}$$

$$= \int_{\mathbb{R}_d} \frac{1}{N}\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})||\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x},t)||^2\mathrm{d}\boldsymbol{x}\ , \tag{14}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. Since the perturbation kernel $p_t(\boldsymbol{x}|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{x}_0, s_t^2\sigma_t^2\mathbf{I})$, then

$$\boldsymbol{x} = s_t\boldsymbol{y}_i + s_t\sigma_t\boldsymbol{\epsilon} \Rightarrow \boldsymbol{\epsilon} = \frac{\boldsymbol{x}-s_t\boldsymbol{y}_i}{s_t\sigma_t}\ . \tag{15}$$

Here, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is a 'denoiser' network for learning the noise $\boldsymbol{\epsilon}$. Plugging Eq. (15) into Eq. (14), the loss can be reparameterized as:

$$\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; t) = \int_{\mathbb{R}_d} \mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)\mathrm{d}\boldsymbol{x}\ , \tag{16}$$

where

$$\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t) = \frac{1}{N}\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})||\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x},t) - \frac{\boldsymbol{x}-s_t\boldsymbol{y}_i}{s_t\sigma_t}||^2\ .$$

Eq. (16) means that we can minimize $\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; t)$ by minimizing $\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)$ for each $\boldsymbol{x}$. As such, we find the 'optimal denoiser' $\boldsymbol{\epsilon}^*_{\boldsymbol{\theta}}$ that minimize the $\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)$, for every given $\boldsymbol{x}$ and t, as:

$$\boldsymbol{\epsilon}^*_{\boldsymbol{\theta}}(\boldsymbol{x}; t) = \text{arg min}_{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x};t)}\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)\ . \tag{17}$$

The above equation is a convex optimization problem with respect to $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ by which the solution can be obtained by setting the gradient of $\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)$ w.r.t $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}; t)$ to zero. Under the assumption that $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ has an infinite model capacity, and can approximate any continuous function to an arbitrary level of accuracy based on the Universal Approximation Theorem [40], then

$$\nabla_{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x};t)}[\mathcal{L}(\boldsymbol{\epsilon}_{\boldsymbol{\theta}}; \boldsymbol{x}, t)] = 0 \tag{18}$$

$$\Rightarrow \frac{1}{N}\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})[\boldsymbol{\epsilon}^*_{\boldsymbol{\theta}}(\boldsymbol{x}; t) - \frac{\boldsymbol{x}-s_t\boldsymbol{y}_i}{s_t\sigma_t}] = 0 \tag{19}$$

$$\Rightarrow \boldsymbol{\epsilon}^*_{\boldsymbol{\theta}}(\boldsymbol{x}; t) = \frac{1}{s_t\sigma_t}[\boldsymbol{x} - s_t\frac{\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})\boldsymbol{y}_i}{\sum_{i=1}^N \mathcal{N}(\boldsymbol{x}; s_t\boldsymbol{y}_i, s_t^2\sigma_t^2\mathbf{I})}] \tag{20}$$

B. Additional Experiments

B.1. Implementation details of the proposed timestep clustering method

We implement a Variance Preserving (VP) [1, 18] perturbation kernel for clustering the time interval. In particular, we use $s_t = \sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min}-1}}, \sigma_t = 1/\sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min}}}, t \in [\epsilon_t, 1], \beta_d = 19.9, \beta_{\min} = 0.1$, and , $\epsilon_t = 10^{-3}$. To obtain t_1 and t_2, we utilize the CIFAR-10 [23] with dataset size $N = 5\times 10^4$. Specifically, Algorithm 1 is configured with $\eta = \frac{2}{256}, \alpha = 0.9$, and $K = 5\times 10^4$. This choice of η ensures the similarity of measurements in the RGB space. We divide $t \in [0,1]$ into 10^4 discrete time steps. Consequently, we employ a grid search to determine optimal values for t_1 and t_2 using the procedure outlined in Algorithm 1. For the CelebA dataset [25], we utilize the same t_1 and t_2. And for Variance Exploding (VE) perturbation kernel, we calculate an equivalent σ_1 and σ_2[3]. Results of Section 5.4 verify that these choices of t_1 and t_2 return the best performance when compared to other clustering methods.

[3]VE utilizes σ instead of t to represent the timestep. We choose σ_1 and σ_2 such that $\text{SNR}_{\text{VE}}(\sigma_1) = \text{SNR}_{\text{VP}}(t_1)$ and $\text{SNR}_{\text{VE}}(\sigma_2) = \text{SNR}_{\text{VP}}(t_2)$.

B.2. Samples of Additional Generated Results

The generation samples displayed in Fig. 3, from the CelebA and CIFAR-10 datasets, demonstrate that our Multistage strategy is capable of producing high-quality results. Notably, it achieves this with reduced training and computational requirements in comparison to baseline methods.

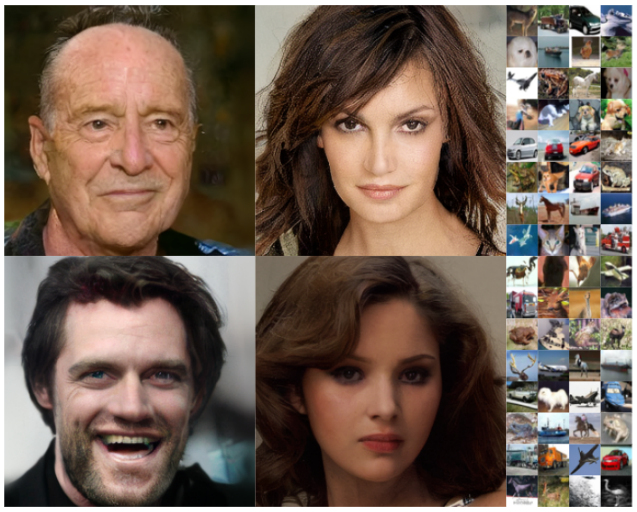

Figure 3. **Sample generations from Multistage LDM (CelebA 256×256) and Multistage DPM-Solver (CIFAR-10 32×32)**

B.3. Comparison with multi-architecture multi-expert (MEME).

In this subsection, we compare our multistage framework with MEME diffusion model [21]. We reimplement the MEME architecture following their released architecture in Table 4 of [21]. We train both our multistage framework and their MEME on the CIFAR-10 dataset and use DPM-Solver for sampling. The results are shown in Tab. 5. From our experiments, Multistage outperforms MEME in FID with much fewer training iterations and total training PFLOPs. However, because [21] only implements the architecture for CelebA-HQ and FFHQ datasets, the method could potentially achieve better results than our reported results with more hyperparameter tunning. Also, the training is unstable and the loss explodes when we try to increase the model capacity of MEME to be comparable to our Multistage.

B.4. Results in terms of the Precision and Recall Metrics

We evaluate the precision and recall [41] of our proposed architecture on the CIFAR-10 dataset, as results shown in Tab. 6. The recall of the multistage diffusion models is slightly better than their vanilla counterparts. This is potentially because the multistage architecture introduces more degree of freedom when $t = 0$ (a separate decoder for this interval), resulting in more diverse data generation.

B.5. Ablation Study on the Network Parameters

In this subsection, we perform a series of experiments to determine the best number of parameters within the attempted settings for each stage of our model. For evaluation, we use the best FID scores from all checkpoints. These evaluations are conducted using the DPM-Solver with 20 NFE on $50,000$ samples. Here, the experimental settings are similar to those used in Section 5.

For the Multistage DPM-solver on CIFAR-10 dataset. The models are trained until they achieve their best FID scores. For calculating FID, we follow the methodology described by [29], using the tensorflow_gan toolbox [4].

The design of the corresponding parameters for encoders and decoders across different architectures, along with their best FID scores, is detailed in Table 7. We explore various architectures, aiming to maintain a consistent total number of parameters while varying the ratio of shared parameters (encoder parameters) to the total number of parameters. Our ablation study illustrates that Architecture 1 (our chosen model) indeed exhibits the best FID performance.

For Latent Diffusion multistage models on CelebA-HQ, models are trained for 500k iterations, and the best checkpoints are used for evaluation. We compute the FID following [8] using the `torch-fidelity` package [42]. The parameter design and best FID are shown in Tab. 8. Our ablation study shows the superiority of utilizing Architecture 4 (our chosen model) in terms of the reported FID values.

B.6. Ablation Study on the Number of Intervals

In this subsection, we conduct ablation study on the number of intervals of our multistage architecture. We utilize Multistage DPM-Solver training on the CIFAR-10 dataset. To extend Algorithm 1, we propose the procedure outlined in Algorithm 2 (Appendix C). The splitting intervals and Results are given in Tab. 9. We compare the stages from one to five. As observed, the three-stage architecture, adopted in the main body of the paper, outperforms all other settings.

B.7. Experimental Setting of the Comparison Study on Different Interval Splitting Methods

This section presents the experimental settings of previous timestep clustering algorithms mentioned in Tab. 4. We refer readers to the following references [20, 21] for implementation details. Specifically, we compare these methods for the 3 intervals case on CIFAR-10 dataset by training our proposed Multistage architecture with calculated intervals until convergence. The timestep and SNR-based clustering algorithms are relatively easy to implement. For the gradient-based clustering, we collect the gradients of the trained network by the DPM-Solver for clustering. These gradients are generated every 50k training iterations with a

[4]https://github.com/tensorflow/gan

Table 5. Training and Sampling Efficiency with related works.

Dataset/Method	Training Iterations(↓)	GFLOPs(↓)	Total Training PFLOPs(↓)	FID(↓)
CIFAR-10 32×32				
MEME DPM-Solver [21]	6.0×10^5	13.08	7.85	3.23
Multistage DPM-Solver (Ours)	$\mathbf{2.5 \times 10^5}$	18.65	**4.66**	**2.71**

Table 6. Precision and Recall on CIFAR-10.

METHOD	**Precision(↑)**	**Recall(↑)**
DPM-Solver	0.660	0.613
EDM	**0.677**	0.620
Multistage DPM-Solver	0.657	0.615
Multistage EDM	0.672	**0.629**

Table 7. **Ablation study of Multistage DPM-solver on CIFAR-10 dataset.** Current/Total: the number of the stage over the total number of stages. Shared: the encoder. Total: combined the encoder and decoders. Total GFLOPS: averaged GFLOPS of all stages weighted by NFE assigned to each stage during sampling of DPM-Solver.

	Current/Total	Parameters	GFLOPS	FID(↓)
1	1/3	169M	29.95	**2.35**
	2/3	108M	17.65	
	3/3	47M	6.31	
	Shared	43M	5.75	
	Total	237M	18.65	
2	1/3	168M	27.57	2.9
	2/3	138M	21.73	
	3/3	72M	9.65	
	Shared	68M	8.98	
	Total	242M	19.7	
3	1/3	189M	29.43	3.1
	2/3	160M	23.91	
	3/3	103M	13.7	
	Shared	98M	12.93	
	Total	256M	22.45	
4	1/3	174M	26.54	3.34
	2/3	126M	17.62	
	3/3	103M	13.7	
	Shared	98M	12.93	
	Total	207M	22.45	

total of 450k iterations. We evaluate the performances of these models based on their best FID scores across training iterations.

Table 8. **Ablation study of Multistage LDM on CelebA dataset.**

	Current/Total	Parameters	GFLOPS	FID(↓)
1	1/3	238M	80.03	9.37
	2/3	189M	64.95	
	3/3	161M	51.87	
	Shared	79M	16.69	
	Total	428M	65.75	
2	1/3	206M	72.61	9.73
	2/3	158M	57.85	
	3/3	108M	34.38	
	Shared	55M	11.6	
	Total	361M	54.37	
3	1/3	274M	88.39	8.43
	2/3	224M	72.97	
	3/3	170M	48.17	
	Shared	108M	22.71	
	Total	452M	69.22	
4	1/3	316M	105.82	**8.38**
	2/3	224M	72.97	
	3/3	170M	48.17	
	Shared	108M	22.71	
	Total	494M	76.19	

C. Generalized Algorithm 1

In this section, we design a procedure to extend Algorithm 1 for the cases of $n \in \{2, 4, 5\}$. Specifically, we partition the timesteps into n intervals $[0, t_1), \ldots, [t_{n-2}, t_{n-1}), [t_{n-1}, 1]$. Intuitively, the intervals need to minimize the summation of the total functional distance within each partition. This corresponds to the following optimization problem.

$$\begin{aligned} \min_{t_1, \cdots, t_{n+1}} & \sum_{k=0}^{n-1} \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \mathcal{S}(\boldsymbol{\epsilon}^*_{t_a}, \boldsymbol{\epsilon}^*_{t_b}) \mathrm{d}t_a \mathrm{d}t_b \\ \text{s.t.} \quad & t_0 = 0, t_n = 1, \\ & 0 < t_1 < t_2 < \cdots < t_{n-1} < 1 \,. \end{aligned} \tag{21}$$

In order to solve Eq. (21) numerically, we propose Algorithm 2. A solution of the program in step 9

Table 9. **Ablation study on the number of stages**

Number of Stages	Current/Total	Interval	Parameters	GFLOPS	FID(↓)
1	1/1	-	108M	17.65	2.75
2	1/2	[0,0.476)	136M	26.59	2.56
	2/2	[0.476, 1]	95M	16.85	
	Total	-	188M	21.49	
3	1/3	[0, 0.442)	169M	29.95	**2.35**
	2/3	[0.442, 0.631)	108M	17.65	
	3/3	[0.631, 1]	47M	6.31	
	Total	-	230M	18.65	
4	1/4	[0, 0.376)	159M	29.78	2.67
	2/4	[0.376, 0.526)	95M	16.85	
	3/4	[0.526, 0.726)	71M	11.97	
	4/4	[0.726, 1]	32M	4.41	
	Total	-	284M	17.33	
5	1/5	[0, 0.376)	154M	30.38	2.88
	2/5	[0.376, 0.476)	88M	16.82	
	3/5	[0.476, 0.626)	88M	16.82	
	4/5	[0.626, 0.776)	27M	4.42	
	5/5	[0.776, 1]	21M	3.28	
	Total	-	336M	17.03	

Algorithm 2 Optimal Denoiser based Timestep Clustering for General Intervals n

```
1: Input: Total samples $K$, optimal denoiser function $\boldsymbol{\epsilon}^*_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$, interval number $n$, dataset $p_{\text{data}}$
2: Output: Timesteps $t_1, t_2, \ldots, t_{n-1}$
3: $\mathcal{S} \leftarrow \emptyset$
4: for $k \in \{1, \ldots, K\}$ do
5:     $\boldsymbol{y}_k \sim p_{\text{data}}, \boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \mathbf{I}), t_{a_k} \sim [0,1], t_{b_k} \sim [0,1]$
6:     $\mathcal{S}^k \leftarrow \mathcal{D}(\boldsymbol{\epsilon}^*_{t_{a_k}}, \boldsymbol{\epsilon}^*_{t_{b_k}}, \boldsymbol{y}_k, \boldsymbol{\epsilon}_k)$
7:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{(t_{a_k}, t_{b_k}, \mathcal{S}^k)\}$
8: end for
9: $t_1, \cdots, t_{i-1} \leftarrow \arg\min_{t_1,\cdots,t_{i-1}} \sum_{k=1}^{K} \sum_{i=0}^{n-1} \sum_{t_i}^{t_{i+1}} \mathcal{S}^k \mathbb{1}(t_{a_k}, t_{b_k} \in [t_i, t_{i+1})),$
10:                    s.t.   $0 < t_1 < t_2 < \cdots < t_{n-1} < 1$
```

is obtained by discretizeing the time index into $t \in \{0.001, 0.026, \ldots, 0.951, 0.976, 1\}$. We use the CIFAR10 dataset with $K = 50000$ to run the experiment. The following table shows the results of different stage numbers n and different split intervals:

D. Details on Training Procedures

Algorithm 3 Memory efficient training (per GPU)

```
1: Input: Total Training iterations T, Parameters $\boldsymbol{\theta}_i$, Interval $[t_{i-1}, t_i]$ for three stages ($i = 1, 2, 3$), GPU index $k$
2: Output: Parameters $\boldsymbol{\theta}^*_i$
3: idx $= k\%3$
4: for iter $\in \{1, \ldots, T\}$ do
5:     Calculate loss and update $\theta_{\text{idx}}$
6: end for
```

For the training, we employ the same training iterations

Algorithm 4 Time efficient training (per GPU)

```
1: Input: Total Training iterations T, Parameters $\boldsymbol{\theta}_i$, Interval $[t_{i-1}, t_i]$ for three stages ($i = 1, 2, 3$)
2: Output: Parameters $\boldsymbol{\theta}_i^*$
3: for iter $\in \{1, \ldots, T\}$ do
4:     for idx $\in \{1, \ldots, 3\}$ do
5:         Calculate loss and update $\theta_{\text{idx}}$
6:     end for
7: end for
```

and loss weights across all stages. Specifically, we propose two algorithms for multi-GPUs parallel training: (*i*) memory-efficient strategy (Algorithm 3) where each GPU updates parameters for a specific stage, requiring less memory, and (*ii*) time-efficient strategy (Algorithm 4) where all GPUs update parameters from all stages. These two algorithms are equivalent from an optimization perspective.