

Three Creates All: You Only Sample 3 Steps

Yuren Cai¹, Guangyi Wang¹, Zongqing Li¹, Li Li², Zhihui Liu³, and Songzhi Su¹

¹ School of Informatics, Xiamen University

² School of Artificial Intelligence, Shenzhen Polytechnic University

³ Algorithm Research Department, Truesight

Abstract. Diffusion models deliver high-fidelity generation but remain slow at inference time due to many sequential network evaluations. We find that standard timestep conditioning becomes a key bottleneck for few-step sampling. Motivated by layer-dependent denoising dynamics, we propose **Multi-layer Time Embedding Optimization (MTEO)**, which freeze the pretrained diffusion backbone and distill a small set of step-wise, layer-wise time embeddings from reference trajectories. MTEO is plug-and-play with existing ODE solvers, adds no inference-time overhead, and trains only a tiny fraction of parameters. Extensive experiments across diverse datasets and backbones show state-of-the-art performance in the few-step sampling and substantially narrow the gap between distillation-based and lightweight methods. Code will be available.

Keywords: Diffusion Model · Acceleration · Image Generation

1 Introduction

Diffusion models [9, 23, 28, 32, 33] have become a leading paradigm for generative modeling, delivering strong results across a wide range of modalities, including image synthesis, audio/speech generation [30], and text-conditioned generation such as text-to-image [23] and text-to-video [1]. A key reason behind their success is the simplicity and generality of learning a denoising function that reverses a gradual noising process. However, this advantage comes with a well-known bottleneck at inference time: generating a single sample requires an iterative denoising procedure, which translates into many sequential function evaluations of a large neural network, limiting practical deployment and real-time applications.

A substantial body of work has aimed to accelerate diffusion sampling process. **Training-free** approaches [16–18, 25, 29, 36, 39, 42–44] improve the numerical solver and/or the sampling schedule without retraining the diffusion model, *e.g.*, by using more efficient non-Markovian trajectories [29], dedicated higher-order solvers for diffusion ODEs [16–18, 42–44], or optimized timestep schedules [25, 39]. While these methods introduce little to no training cost, their sample quality typically degrades rapidly in the extremely few-step regime (*e.g.*, 3–5 NFE), where large step sizes amplify discretization errors.

Training-based acceleration methods mitigate this issue by introducing an additional training stage tailored to few-step inference. Existing works can be

roughly grouped into: (i) *lightweight training-based techniques* [2, 5, 14, 34, 37, 38, 45], which make minimal modifications (*e.g.*, plugin predictors, schedule/search refinements, or auxiliary parameterizations) and often achieve strong performance around ~ 5 NFE; and (ii) *distillation-based approaches* [6, 11, 19, 27, 31, 40, 46], which can reach high-fidelity generation at 3–4 NFE (or even one-step in some settings) but usually require substantially heavier training and updating a large fraction of model parameters. Despite this progress, an important gap remains: achieving *distillation-level* few-step quality in the 3–5 NFE regime without paying distillation-level training cost.

In this paper, we revisit a component that is ubiquitous yet often treated as fixed during sampling: *time conditioning*. Modern diffusion backbones condition each denoising step through a timestep embedding, which is injected into network blocks to control feature-wise modulation (*e.g.*, FiLM/AdaLN). By carefully analyzing the full pathway from the scalar time variable to layer-wise feature modulation (see Sec. 2), we identify three coupled limitations that become pronounced under few-step sampling: (1) the conventional single time-variable design is misaligned with the effective optimal update time in large-step solvers (see Sec. 3.1); (2) different network layers exhibit distinct feature trajectories and thus prefer different effective time conditioning (see Sec. 3.2); and (3) the standard timestep embedding has limited expressive degrees of freedom, which can severely constrain the modulation capacity of FiLM (see Sec. 3.3).

To address these issues, we propose **Multi-layer Time Embedding Optimization (MTEO)**, a training-based yet *lightweight* framework for fast diffusion sampling. Instead of retraining (or distilling) the entire diffusion model, MTEO freezes the pretrained backbone and optimizes only a small set of *step-wise, layer-wise* time embeddings via trajectory distillation (see Sec. 3.5). This design (i) decouples time conditioning across layers, (ii) unlocks richer layer-wise feature modulation, and (iii) introduces *no* additional inference-time overhead, while requiring only a tiny fraction of trainable parameters (typically $< 0.2\%$) and modest training data (a small set of trajectories). Across diverse benchmarks and backbones, MTEO achieves state-of-the-art performance in the 3–6 NFE regime and significantly narrows the quality gap to distillation-based methods under extreme acceleration (see Sec. 4).

2 Background

We review only the components directly used by MTEO; additional background (*Overview of Diffusion Models, Trajectory Distillation*) is deferred to Appendix.

2.1 Procedure of Time Information Injected into Diffusion Network

A wide range of neural network architectures have been adopted for diffusion models. Despite differences in specifics, most models are built on backbones such as U-Net [24] or Transformer [21], which typically follow a layer-wise design with

multi-scale feature hierarchies and skip connections. Regardless of the architectural details, the core computational workflow is similar. At each time step t , the noisy input x_t is fed into the network (see Fig. 1) to predict the output: the denoised x_0 or the noise residual ϵ . Crucially, the network is conditioned on the time step through a *time embedding*.

In practice, time information is injected into each layer of the network through the following process: First, the current time step t is encoded using a sinusoidal positional encoding and then passed through an MLP to produce a time embedding vector. As feature information flows through the network layer by layer, the time embedding vector is injected into specific layers, allowing the model to modulate its computation based on the current noise level. This enables each layer to process both the intermediate feature map and the corresponding time embedding, effectively integrating both the feature and temporal information throughout the denoising process. At each individual layer, time information specifically modulates the processing of feature information through Feature-wise Linear Modulation (FiLM).

2.2 Feature-wise Linear Modulation (FiLM)

Feature-wise Linear Modulation (FiLM) is a conditional mechanism that adjusts intermediate features by applying a learnable, feature-wise affine transformation (i.e., scaling and shifting) as a function of the conditioning signal. Concretely, let $s_l \in \mathbb{R}^{c \times w \times h}$ denote the feature tensor at layer l , where c is the number of channels and each channel corresponds to a $w \times h$ feature map. FiLM modulates s_l as

$$s_{\text{modulated}} = \alpha_l \odot s_l + \beta_l, \quad (1)$$

where $\alpha_l, \beta_l \in \mathbb{R}^c$ are channel-wise scaling and bias vectors (broadcast over spatial dimensions), and \odot denotes element-wise multiplication. This operation can be viewed as applying a per-channel linear transformation to the feature representation at each layer.

In the U-Net framework, the time embedding vector injected into layer l is first passed through a layer-specific **affine** projection to produce the corresponding FiLM parameters (α_l, β_l) , which are then used to modulate s_l as above. In the DiT framework, the overall procedure is conceptually the same: the time embedding is provided to each Transformer block and mapped by the block’s **adaLN_modulation** module (which plays a role analogous to the affine projection in U-Net) to generate (α_l, β_l) for feature modulation. Fig. 1 (right, bottom) provides a visual illustration of this process.

3 Method

3.1 Suboptimality of Conventional Single Time Variable Design

We begin by examining why the conventional practice of conditioning the denoiser on a *single global time variable* can be suboptimal in the few-step regime.

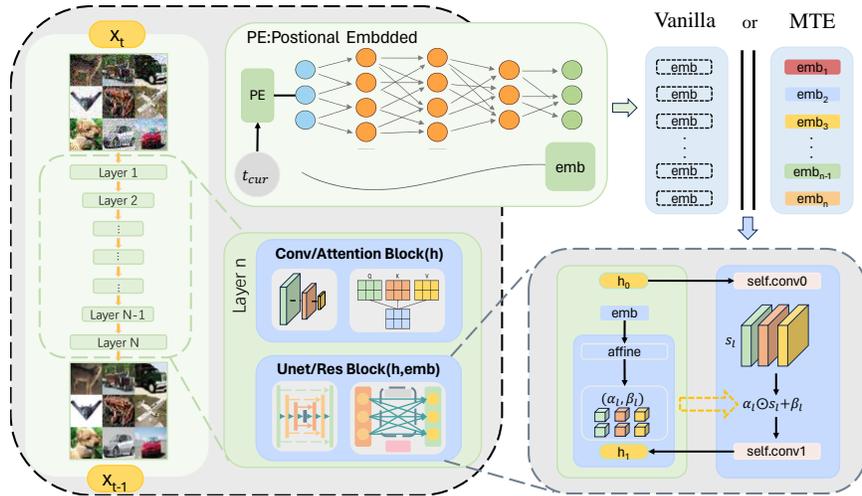


Fig. 1: Overview. (left) The visualization of single sampling step. (mid top) Procedure from t_{cur} to time embedding. (mid bottom) Two type of diffusion inner blocks. (right top) The visualization of multi-layer and vanilla embedding. Different colors represent independence across the layers, dashed box denote the embedding are produced by broadcasting. (right bottom) The inner display of FiLM exert on feature maps.

Most diffusion backbones encode the scalar timestep using a sinusoidal embedding (inspired by the positional encoding in Transformers [35]) and use it to condition the network on the current noise level.

The issue becomes pronounced when the sampling budget is extremely small. With few steps, each solver update spans a large time interval from t_{cur} to t_{next} , yet the standard sampler queries the network using the endpoint time t_{cur} . As a result, a single network evaluation must serve as a coarse proxy for the model behavior over the entire interval, which can introduce systematic discretization errors.

To quantify this mismatch, we generate a high-fidelity reference trajectory using a large number of sampling steps, and construct additional trajectories with fewer steps (“medium-step” and “low-step”). For each solver step from t_{cur} to t_{next} along a low-step trajectory, we treat the conditioning time as a free variable: we sweep τ from 80.00 to 0.002 while keeping the solver interval fixed, and compare the resulting output against the reference. As Fig. 2, we find that the time value yielding the best match (denoted t_{min}) consistently lies *inside* the interval (t_{next}, t_{cur}) , rather than exactly at t_{cur} as assumed by the standard sampler. Moreover, t_{min} deviates further from t_{cur} as the step size increases, while it naturally approaches t_{cur} when the step size becomes small.

These observations indicate that, in the few-step setting, using a single fixed endpoint time to condition the entire network is generally misaligned with the effective time that best matches the reference trajectory, and is therefore sub-optimal.

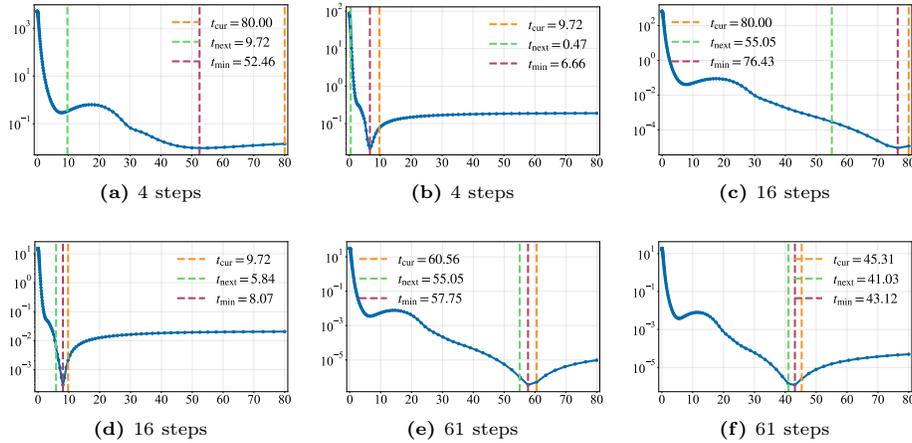


Fig. 2: L2 distance between different sampling step of DDIM and ground-truth trajectories on CIFAR-10. We generate the ground-truth trajectory with iPNDM, 61 steps, `schedule_type=polynomial`, `schedule_rho=7`. The τ is generated by 121 steps.

3.2 Distinct Feature Trajectories Across Layers

It has been observed in prior work [45] that the sampling trajectory of a diffusion model often lies nearly in a two-dimensional subspace of the high-dimensional state space. Motivated by this observation, we hypothesize that, analogous to the model’s overall sampling trajectory, each layer of the network traces its own *feature trajectory* over the course of denoising. To analyze these differences, we captured a reference denoising trajectory and recorded the feature map at every layer along this trajectory. Following the approach of previous work [45], we applied principal component analysis (PCA) to the sequence of feature maps from each layer and visualized each layer’s trajectory in its principal component subspace. This figure (Appendix) shows examples of these trajectories.

As shown in figure (Appendix), these layer-specific trajectories differ from the global one, and from each other, in two major ways: (1) the dimensionality of the low-dimensional subspace that a given layer’s trajectory occupies varies across layers; and (2) although all layers are driven by the same time variable t , the curvature (i.e., the degree of turning) of feature trajectories differs substantially across layers.

Tab. 1 quantifies the dimensionality of these trajectories: in early layers, the first 2 or 3 principal components explain over 90% of the variance, indicating that the feature trajectory lies essentially in a planar (2D or 3D) subspace. In deeper layers, the variance is distributed across more dimensions: 4 or 5 components are required to reach 90% explained variance.

Taken the perspective on the global sampling path, these inter-layer discrepancies suggest that the *effective* time that best aligns with each layer’s feature trajectory may differ across layers. Similar to the experiment in Fig. 2, we apply the same time sweeping procedure to layer-wise feature trajectories. As shown

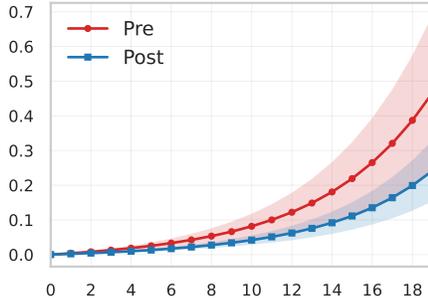


Fig. 3: The L1 distance of FiLM. We sample 64 ground-truth trajectories and extract the feature maps inside the first block. Then we simulate the first evaluation of a 4-step sample from $t = 80.00$ (idx=0) to $t = 9.72$ (idx=19). We calculate the L1 distance between $t = 80.00$ and $t = 9.72$ before (Pre) and after (Post) adjusting the FiLM and present the mean and variance. See more in the appendix.

Index	PC ₁	PC _{1:2}	PC _{1:3}	PC _{1:4}	PC _{1:5}
Shallow					
0	90.13	99.33	99.82	99.94	99.98
1	73.99	88.59	94.94	98.36	99.29
2	67.04	82.43	90.70	96.40	98.30
32	53.16	75.76	85.36	91.48	94.68
33	74.25	89.34	93.85	96.68	97.95
Deep					
12	45.63	70.34	82.45	90.45	94.41
13	44.99	69.76	81.85	90.07	94.26
14	42.10	68.67	83.16	91.31	94.92
16	44.46	69.35	81.80	89.77	93.77
20	47.87	70.88	81.75	89.79	93.39

Table 1: Cumulative explained variance (%) of the first five principal components for trajectory 0. $PC_{1:n}$ denote first n principal components. Complete results are list in the appendix.

in Fig. 4, the optimal conditioning time t_{\min} for a given layer often deviates from the current point t_{cur} , and moreover varies across layers. This empirically indicates that layer-wise feature dynamics are not strongly coupled to a single shared input time, and motivates assigning each layer its own effective time t_l (equivalently, layer-specific time conditioning) during sampling.

3.3 From Time Variable to Time Embedding

Previously, we argued that during sampling, different layers of a diffusion network can exhibit a certain degree of independence with respect to the time variable, i.e., different layers may prefer different effective times. We now further examine the design that maps the scalar time variable to the time embedding injected into the network.

As discussed earlier, the scalar time t is first encoded via a sinusoidal embedding and then passed through an MLP to produce a time embedding vector. As the data flows through the network, this time embedding is injected into each layer and projected by a layer-specific affine transform to generate the FiLM modulation parameters, the channel-wise scaling and shift vectors $(\alpha_l, \beta_l) \in \mathbb{R}^c$, which directly modulate intermediate features.

This raises an important question: does the resulting time embedding provide sufficient expressive power to generate appropriate layer-wise (α_l, β_l) , or does it instead impose a representational bottleneck? To probe this, we collect the time embeddings from a 121-step schedule and perform PCA. As shown in Fig. 5(left), the first two principal components explain **77.72%** and **19.41%** of the variance,

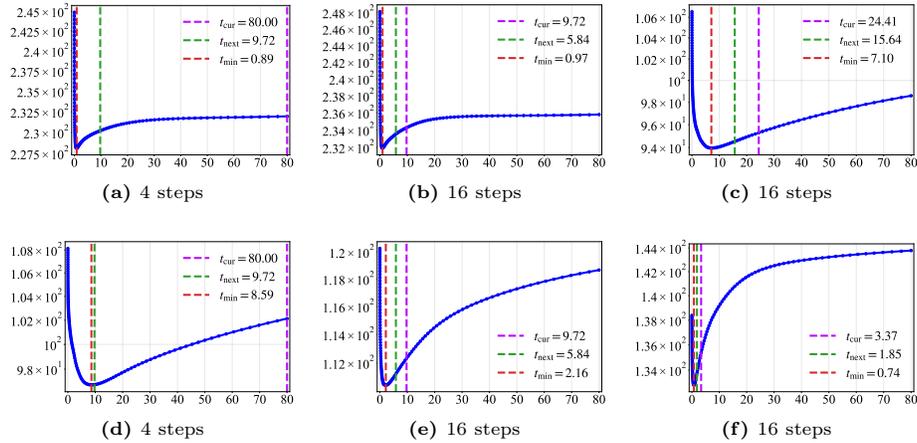


Fig. 4: L2 distance between different feature trajectories. The setting is same as Fig. 2.

respectively, accounting for **97.14%** in total. This indicates that the time embeddings across timesteps lie close to a very low-dimensional subspace.

In contrast, when we perform PCA on the FiLM parameters (α_l, β_l) across time steps and layers, we find substantially higher intrinsic dimensionality: the first two principal components explain only **19.41%** of the variance, and even the first 30 components explain only **90.49%**. This mismatch suggests that the conventional time-embedding design can severely restrict the degrees of freedom available to FiLM modulation, thereby limiting its representational capacity. We also conduct the same PCA procedure on MTE Fig. 5(right), which shows the explosive degree of freedom, further strengthen our claim. We continue the discussion in Appendix.

Finally, we empirically verify whether FiLM itself has sufficient modulation capacity to correct layer-wise feature processing when provided with appropriate parameters. We compare a high-step sampling trajectory (iPNM-61) with an extremely few-step trajectory (DDIM-4). For one layer at time t along the few-step trajectory, we record the feature maps before and after FiLM as s^t and $s_{\text{modulated}}^t$, respectively. For the corresponding teacher trajectory at time τ , we denote the analogous quantities as \hat{s}^τ and $\hat{s}_{\text{modulated}}^\tau$. We define the ℓ_1 discrepancy as

$$\mathcal{L} = \|\hat{s}_{\text{modulated}}^\tau - s_{\text{modulated}}^t\|_1 = \|\hat{s}_{\text{modulated}}^\tau - (\alpha_l \odot s^t + \beta_l)\|_1. \quad (2)$$

In principle, by choosing suitable scaling and bias vectors (α_l, β_l) , FiLM can reduce this discrepancy and align the modulated features. Following [45], for a solver step spanning $(t_{\text{next}}, t_{\text{cur}})$, there exists an intermediate time $t_{\text{min}} \in (t_{\text{next}}, t_{\text{cur}})$ whose direction provides the best update. Therefore, we scan the teacher trajectory within $(t_{\text{next}}, t_{\text{cur}})$ (to approximate t_{min}), compute the original ℓ_1 loss, and then optimize (α_l, β_l) to minimize the loss for each τ , reporting the mean and variance. As shown in Fig. 3, FiLM modulation can substantially

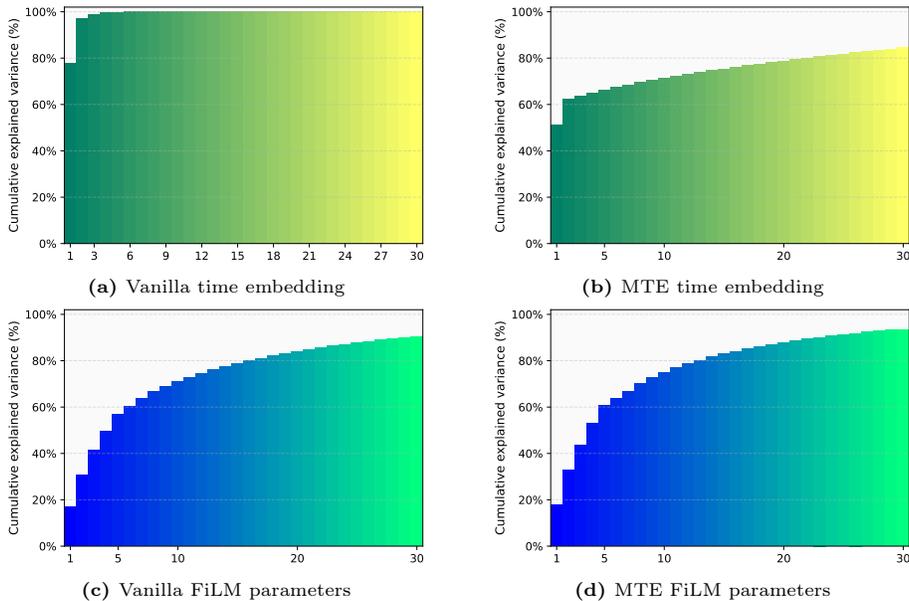


Fig. 5: PCA analysis on conventional embedding design and MTE. The x-axis denote first n principal components. **Green:** time embedding. **Blue:** FiLM parameters.

reduce the feature discrepancy, demonstrating its strong corrective capability. This also suggests that such capability is not fully exploited under the conventional time-embedding pipeline, leaving layers insufficiently modulated and leading to pronounced quality degradation in few-step sampling.

3.4 Multi-layer Time Embedding

Our analyses in Secs. 3.1 to 3.3 point to a consistent message: *time conditioning in diffusion networks is inherently layer-dependent in the few-step regime*. Specifically, (i) different layers exhibit distinct feature trajectories with different geometric complexity, suggesting that their dynamics may align best with different effective times; and (ii) the conventional shared timestep embedding provides limited degrees of freedom for generating diverse layer-wise FiLM modulation parameters. Together, these findings indicate that forcing all layers to share the same time embedding at each solver step can be unnecessarily restrictive, and may under-utilize FiLM’s modulation capacity.

Motivated by this, we propose *Multi-layer Time Embedding (MTE)*. In standard diffusion samplers, a single time embedding computed from the solver time (e.g., t_{cur}) is broadcast to all layers. By contrast, MTE maintains a *layer-specific* embedding for each sampling step: for step i (with solver time t_i), we use a set of learnable vectors $\Phi_i = \{\phi_{i,\ell}\}_{\ell=1}^L$, where $\phi_{i,\ell} \in \mathbb{R}^d$ is used exclusively to condition layer ℓ . During sampling, each layer consumes its own $\phi_{i,\ell}$ to produce its FiLM

modulation parameters, while the backbone denoiser weights remain unchanged. This decoupling removes the implicit synchronization of time conditioning across layers, allowing each block to adapt its modulation to its own feature trajectory rather than being constrained by a shared embedding. Fig. 1 (right top) provides a conceptual illustration of the MTE design. It also naturally raises the next question: how can we obtain MTE for a fixed few-step sampling schedule?

Algorithm 1 Training MTE via Trajectory Distillation

Input: Teacher trajectory $\{\hat{x}_{t_i}\}_{i=0}^{N-1}$, time schedule $\{t_i\}_{i=0}^{N-1}$, ODE solver S , frozen denoiser ϵ_θ , distance metric \mathcal{D} , layer-wise embeddings $\{\Phi_i\}_{i=0}^{N-1}$ with $\Phi_i = \{\phi_{i,\ell}\}_{\ell=1}^L$.
Output: Optimized embeddings $\{\Phi_i\}_{i=0}^{N-1}$.

- 1: $x_{t_0} \leftarrow \hat{x}_{t_0}$
- 2: **for** $i = 0$ **to** $N - 2$ **do**
- 3: **repeat**
- 4: $x_{t_{i+1}} \leftarrow S(x_{t_i}, \epsilon_\theta, t_i, t_{i+1}, \Phi_i)$
- 5: $\mathcal{L} \leftarrow \mathcal{D}(x_{t_{i+1}}, \hat{x}_{t_{i+1}})$
- 6: Update Φ_i using $\nabla_{\Phi_i} \mathcal{L}$
- 7: **until** converged
- 8: $x_{t_{i+1}} \leftarrow S(x_{t_i}, \epsilon_\theta, t_i, t_{i+1}, \Phi_i)$
- 9: **end for**
- 10: **return** $\{\Phi_i\}_{i=0}^{N-1}$

3.5 Learning MTE via Trajectory Distillation

The analysis in Sec. 3.3 suggests that FiLM has sufficient capacity to correct intermediate representations when provided with appropriate modulation parameters. We therefore learn MTE by a trajectory-matching objective: we freeze the pretrained diffusion model and optimize only the layer-wise time embeddings so that a few-step sampler can track a high-fidelity teacher trajectory.

Concretely, we first generate a teacher trajectory $\{\hat{x}_{t_i}\}_{i=0}^{N-1}$ using a high-quality sampler with a large number of function evaluations. We then run a student trajectory $\{x_{t_i}\}_{i=0}^{N-1}$ using a fixed N -step schedule $\{t_i\}_{i=0}^{N-1}$, but with MTE enabled. Let $\Phi_i = \{\phi_{i,\ell}\}_{\ell=1}^L$ denote the collection of learnable embedding vectors at step i across all layers. We initialize $x_{t_0} = \hat{x}_{t_0}$ and, for each step i , perform one solver update

$$x_{t_{i+1}} = S(x_{t_i}, \epsilon_\theta, t_i, t_{i+1}, \Phi_i), \quad (3)$$

where ϵ_θ is the frozen denoiser and $S(\cdot)$ is the chosen ODE solver (sampler step). We optimize Φ_i to match the teacher state at the same time using an ℓ_2 objective:

$$\mathcal{L}_i = \|x_{t_{i+1}} - \hat{x}_{t_{i+1}}\|_2^2. \quad (4)$$

In practice, we adopt a stage-wise training scheme and optimize Φ_i step by step, which avoids excessive overhead and stabilizes optimization.

To efficiently train all embeddings, we use an early-stopping criterion based on the *relative* improvement in loss,

$$\mathcal{L}_{\text{rel}} = \frac{\mathcal{L}_{\text{prev}} - \mathcal{L}_{\text{cur}}}{\mathcal{L}_{\text{prev}}}. \quad (5)$$

Given a threshold ε , patience p , and a maximum epoch budget E_{max} , we stop updating Φ_i if $\mathcal{L}_{\text{rel}} < \varepsilon$ for p consecutive checks, or when E_{max} epochs are reached. For later denoising steps (smaller t), we progressively tighten the threshold ε . For the last step, we disable early stopping and always train up to E_{max} epochs. The full algorithm, including the optimization schedule, is provided in the Appendix.

4 Experiments

4.1 Experimental Settings

Datasets and Checkpoints. We evaluate MTEO under two representative diffusion backbones: (i) EDM-style U-Net models and (ii) Diffusion Transformers (DiT). Under the EDM setting, we consider CIFAR-10 (32×32) [12], FFHQ (64×64) [10], ImageNet-64 (64×64) [3], and LSUN Bedroom (256×256) [41]. We further evaluate text-to-image generation on MS-COCO (512×512) [15] using Stable Diffusion v1.5 [23]. Under the DiT setting, we report results on ImageNet-256 (256×256) [3]. Regarding conditioning, CIFAR-10, FFHQ, and LSUN-Bedroom use unconditional generation checkpoints, while ImageNet-64 and ImageNet-256 are class-conditional. For MS-COCO, we use text prompts and the corresponding text encoder in Stable Diffusion v1.5. All pretrained diffusion checkpoints (and any associated configuration details) are summarized in the Appendix.

Training Configuration. We train MTEO following the trajectory-distillation procedure described in Sec. 3.5 (see also Algorithm 1). As default, We use 256 teacher trajectories (seeds 50000-50255), batch size=64, $\epsilon = 0.01$, $E_{\text{max}} = 300$, $p = 10$, and teacher’s NFE=21,20,20,24 corresponding to student’s NFE=3,4,5,6. For all EDM experiments (across datasets and solvers), we enable the early-stopping and convergence-control strategy in Sec. 3.5 to reduce training overhead while maintaining stable optimization. For DiT, we disable early stopping and instead train for a fixed number of epochs for simplicity and stability. The full set of training hyperparameters for every experiment, and additional training diagnostics (*e.g.*, loss curves), intermediate visualizations are provided in Appendix. The training procedure is highly reproducible.

Evaluation Protocol. We adopt the Fréchet Inception Distance (FID) [8] as the primary metric. Unless otherwise stated, we generate 50,000 samples to compute FID, using a unique random seed per image (seeds 0–49,999). For transparency and reproducibility, the reference statistic files used for FID computation on each dataset are listed in the Appendix. We additionally report Inception

Table 2: FID evaluation on CIFAR-10 and FFHQ Datasets. We compare MTEO against lightweight and training-free methods. Details are presented in the appendix.

Method	CIFAR-10				FFHQ			
	NFE							
	3	4	5	6	3	4	5	6
DDIM	93.36	67.40	49.66	36.08	78.16	57.37	43.85	35.15
iPNDM	47.98	24.81	13.59	7.05	45.90	28.21	17.14	10.00
UniPC	109.60	45.54	23.98	11.51	86.34	44.73	21.36	12.82
DEIS	56.00	25.64	14.37	9.39	54.45	28.30	17.39	12.29
DPM								
+Solver-2	155.70	145.91	57.30	59.97	266.00	238.61	87.10	83.15
+Solver++(3M)	110.00	46.52	24.97	11.99	86.45	45.94	22.51	13.74
AMED								
+Solver	18.49	17.18	7.59	7.04	47.33	31.19	14.76	11.43
+Plugin	10.81	10.43	6.61	6.67	26.90	24.07	12.47	9.95
EPD								
+Solver	10.40	–	4.33	–	21.74	–	7.84	–
+Plugin	10.54	–	4.47	–	19.02	–	7.97	–
MTEO (Ours)								
+DDIM	3.85	2.83	2.62	2.50	5.46	3.66	3.17	2.99
+iPNDM	4.83	3.52	3.01	2.74	7.46	5.63	4.31	3.72
+DPM-Solver++(3M)	3.91	3.11	2.98	2.66	5.29	3.65	3.37	3.13

Score (IS) [26], sFID [20], Precision/Recall [13], and CLIP Score [7, 22]. All results are highly reproducible.

4.2 Main Results

Comparison with Lightweight Samplers. We compare MTEO against a broad range of lightweight fast methods, including DDIM [29], UniPC [43], DEIS [33], iPNDM [42], DPM-Solver variants [17, 18, 44], AMED [45], and the recent EPD [47]. The 3,5 NFE results of AMED and EPD are directly borrow from original papers while others runs on our machine, following the same configuration recommended by authors. Across 3–6 NFE, MTEO achieves sota performance, with pronounced gains at the lowest budget (3 NFE). For instance, at 3 NFE, the previous sota EPD reports FID of 10.40, 19.02, and 18.28 on CIFAR-10, FFHQ, and ImageNet-64, respectively, whereas MTEO achieves 3.85, 5.46, and 7.42, indicating substantially improved few-step sampling quality.

Detailed FID results on CIFAR-10, FFHQ, ImageNet-64, and LSUN Bedroom are reported in Tab. 2 and Appendix(ImageNet-64, LSUN Bedroom), We report the best results of MTEO. Additional metrics are provided in the Appendix. Results on MS-COCO (FID and CLIP Score) are summarized in Tab. 3. For DiT on ImageNet-256, we evaluate MTEO on top of DDIM, which serves as a strong and widely used deterministic baseline. The metrics are compute by *evaluation* file provided by [4]. The corresponding results are reported in Tab. 4.

Table 3: FID and CLIP evaluation on MS-COCO Dataset. We omitted some of the results due to the lack of corresponding pretrained checkpoints and settings.

Method	FID ↓				CLIP ↑			
	3	4	5	6	3	4	5	6
DDIM	36.04	24.02	19.73	17.69	25.87	28.35	29.32	29.79
DPM-Solver++(2M)	35.03	21.57	17.45	15.85	25.95	28.50	29.42	29.83
AMED-Plugin	–	18.92	–	14.84	–	–	–	–
EPD-Solver	–	16.46	–	13.14	–	–	–	–
+DDIM	14.93	12.92	13.65	13.53	28.86	29.65	29.85	30.01
+DPM++(2M)	16.12	13.39	13.18	13.57	28.61	29.56	29.89	30.05

Table 4: IS, FID, sFID, Precision and Recall evaluation on ImageNet-256 Dataset. We implement MTEO on DiT-XL-2.

Method	Metric	NFE=3	4	5	6
DDIM	IS	12.82	32.22	59.53	90.81
	FID	108.08	77.82	52.52	34.36
	sFID	94.32	63.94	41.64	27.72
	Precision	11.13%	23.46%	36.05%	47.37%
	Recall	36.04%	42.05%	44.85%	46.30%
MTEO	IS	133.65	181.52	197.59	211.00
	FID	15.75	7.22	5.17	4.13
	sFID	12.51	6.92	5.42	5.00
	Precision	60.57%	70.38%	73.02%	75.47%
	Recall	54.83%	58.00%	59.16%	59.51%

Table 5: MTEO+DDIM employ on few-step and sampling with medium-step.

Steps	Training Steps	FID
4	–	93.36
5	–	67.40
6	–	49.66
7	–	36.08
11	–	15.68
13	–	11.89
13	4	3.52
13	5	2.40
11	6	2.33
13	7	2.26

Comparison with Distillation-Based Acceleration. We further compare MTEO (applied to DDIM) with state-of-the-art distillation-based acceleration methods on CIFAR-10, ImageNet-64, and LSUN Bedroom, including SFD/SFD-v [46], CTM [11], DMD-cond [40], and ECM [6]. As summarized in Tab. 6 and Appendix(ImageNet-64, LSUN Bedroom), we report not only FID but also the model size, the number of trainable parameters, the percentage of trainable parameters relative to the original model, and the wall-clock training time (measured on solo RTX 3090). Generating 256 teacher trajectories with NFE=21 costs 0.17 minutes on RTX3090, which is <0.003% of embedding optimization time. The comparison highlights a clear operating regime where MTEO is particularly attractive: while distillation methods typically require updating a large fraction (or the entirety) of the model parameters and incur substantial training cost, MTEO fine-tunes only a tiny set of time-embedding parameters (often well below 0.2% of the backbone size) and can reach competitive quality with markedly lower training overhead. In our experiments, MTEO reduces training time by a large margin (ranging from $\sim 2\times$ to $\sim 1000\times$, depending on the baseline and target NFE), while remaining lightweight by design. We compute model sizes and trainable-parameter sizes directly from the official pretrained checkpoints

Table 6: Comparison MTEO against Distillation-Based acceleration on CIFAR-10 Dataset. We report +DDIM as main results. Hours denote solo RTX3090 GPU hour. The Model Size denote the size of original teacher model and The Parameter indicates the parameter count used for distillation in the corresponding method.

Method	Model Size	Parameter	Percent	NFE	FID	Hours
DMD	212.83MB	1464.32MB	688%	1	2.66	-
CTM	212.83MB	460.80MB	216%	1	1.98	153.70
ECM	212.83MB	212.83MB	100%	1	3.60	709.52
ECM	212.83MB	212.83MB	100%	2	2.11	709.52
SFD	212.83MB	212.83MB	100%	2	4.53	1.19
SFD	212.83MB	212.83MB	100%	3	3.58	1.69
SFD	212.83MB	212.83MB	100%	4	3.24	2.17
SFD	212.83MB	212.83MB	100%	5	3.06	2.63
SFD-v	212.83MB	212.83MB	100%	2	4.28	7.84
SFD-v	212.83MB	212.83MB	100%	3	3.50	7.84
SFD-v	212.83MB	212.83MB	100%	4	3.18	7.84
SFD-v	212.83MB	212.83MB	100%	5	2.95	7.84
MTEO	212.83MB	0.27MB	0.125%	3	3.85	0.13
MTEO	212.83MB	0.33MB	0.156%	4	2.83	0.18
MTEO	212.83MB	0.40MB	0.188%	5	2.62	0.19
MTEO	212.83MB	0.47MB	0.219%	6	2.50	0.22

and the embedding files produced by MTEO. The Appendix details (i) how we normalize training time across different GPU types when necessary and (ii) the sources used for distillation baselines’ runtime statistics.

4.3 Ablation Study

This section include follows: *Batch Size*, *Training Set Size*, *Training Heuristics*, *Sensibility to Steps of Teacher*, *Comparison with a Shared (Single) Time Embedding*, and *Which Time Steps Matter*. Additional ablations are reported in Appendix; here we focus on step-importance and Shared (Single) Time Embedding, which directly supports our design.

Which Time Steps Matter. Given a sampling trajectory with N time steps indexed by

$$\{0, 1, \dots, N - 1\}$$

, we select a subset of time-step indices $\mathcal{T} \subseteq \mathbb{U} = \{0, 1, \dots, N - 2\}$ for optimization. Each element in \mathcal{T} corresponds to a distinct time step at which additional optimization is performed. We analyze the importance (and necessity) of step-specific embeddings by evaluating MTEO from two complementary perspectives.

Gain view. We enable MTE only on a subset of steps and use the original time embedding for all other steps, measuring how much this partial MTE improves over the baseline sampler.

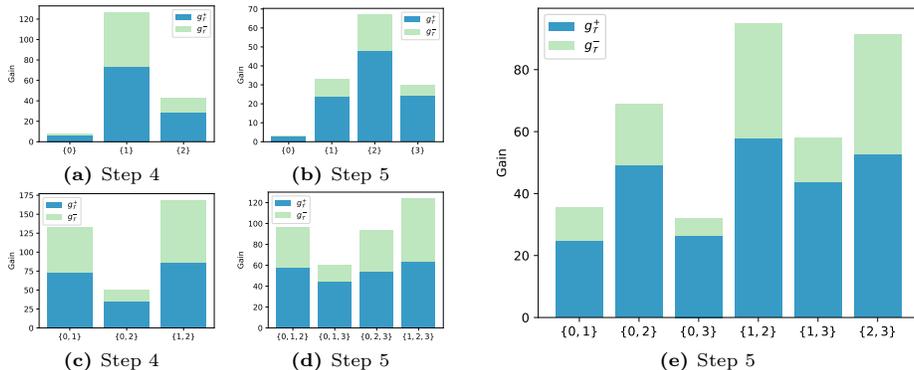


Fig. 6: visualization of both Gain $g_{\mathcal{T}}^+$ and Drop $g_{\mathcal{T}}^-$ view. The x axis denote subset \mathcal{T} .

Drop view. Starting from the full MTE setting, we *remove* MTE from a subset of steps \mathcal{T} while keeping MTE on the remaining steps $\tilde{\mathcal{T}} = \mathbb{U} \setminus \mathcal{T}$, and measure the resulting degradation relative to full MTE. Formally, we define

$$g_{\mathcal{T}}^+ = FID_{\emptyset} - FID_{\mathcal{T}}, \quad g_{\mathcal{T}}^- = FID_{\tilde{\mathcal{T}}} - FID_{\mathbb{U}}, \quad (6)$$

where FID_{\emptyset} denotes the baseline sampler without MTE, $FID_{\mathbb{U}}$ denotes full MTEO, and $FID_{\mathcal{T}}$ denote using MTE on the corresponding step subsets. Under both views, larger $g_{\mathcal{T}}^+$ or $g_{\mathcal{T}}^-$ indicates that the subset \mathcal{T} contributes more positively to sampling quality. The results in Fig. 6 show consistent positive contributions across steps under both views, supporting that MTEO draws benefits from *multiple* steps rather than relying on a single critical step. Finally, we test whether MTE trained on a few-step schedule can transfer to a denser multi-step schedule that includes the same time points. As shown in Tab. 5, few-step embeddings still provide improvements, suggesting that MTE captures transferable correction signals rather than overfitting to a single step count. Details are list in Appendix.

Comparison with a Shared Time Embedding. To isolate the effect of the *multi-layer* design, we train a strong single-embedding baseline under the same trajectory-distillation configuration. Concretely, at each sampling step, we learn *one* time embedding vector and broadcast it to all layers (i.e., the embedding is step-specific but not layer-specific). The comparison is reported in Tab. 7.

Table 7: Ablation of single-layer

Method	NFE	FID ↓		
		vanilla	+single	+MTEO
DDIM	3	93.36	15.60	5.29
	4	67.40	10.43	3.91
	5	49.66	6.81	3.02
	6	36.08	5.18	2.87
iPNM	3	47.98	18.75	6.63
	4	24.81	10.06	4.81
	5	13.59	8.14	4.20
	6	7.05	5.30	3.26
DPM++3M	3	110.00	24.77	5.54
	4	46.52	12.95	3.87
	5	24.97	8.18	3.38
	6	11.99	5.34	3.60

References

1. Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., Ng, C., Wang, R., Ramesh, A.: Video generation models as world simulators (2024), <https://openai.com/research/video-generation-models-as-world-simulators> **1**
2. Chen, D., Zhou, Z., Wang, C., Shen, C., Lyu, S.: On the trajectory regularity of ode-based diffusion sampling. arXiv preprint arXiv:2405.11326 (2024) **2**
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009) **10, 25**
4. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. Advances in neural information processing systems **34**, 8780–8794 (2021) **11**
5. Dockhorn, T., Vahdat, A., Kreis, K.: Genie: Higher-order denoising diffusion solvers. Advances in Neural Information Processing Systems **35**, 30150–30166 (2022) **2**
6. Geng, Z., Pokle, A., Luo, W., Lin, J., Kolter, J.Z.: Consistency models made easy. arXiv preprint arXiv:2406.14548 (2024) **2, 12, 24**
7. Hessel, J., Holtzman, A., Forbes, M., Bras, R.L., Choi, Y.: Clipscore: A reference-free evaluation metric for image captioning. arXiv preprint arXiv:2104.08718 (2021) **11**
8. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017) **10**
9. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. arXiv preprint arxiv:2006.11239 (2020) **1**
10. Karras, T.: A style-based generator architecture for generative adversarial networks. arXiv preprint arXiv:1812.04948 (2019) **10, 25**
11. Kim, D., Lai, C.H., Liao, W.H., Murata, N., Takida, Y., Uesaka, T., He, Y., Mitsufuji, Y., Ermon, S.: Consistency trajectory models: Learning probability flow ode trajectory of diffusion. arXiv preprint arXiv:2310.02279 (2023) **2, 12, 19**
12. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009) **10, 25**
13. Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., Aila, T.: Improved precision and recall metric for assessing generative models. Advances in neural information processing systems **32** (2019) **11**
14. Li, L., Li, H., Zheng, X., Wu, J., Xiao, X., Wang, R., Zheng, M., Pan, X., Chao, F., Ji, R.: Autodiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7105–7114 (2023) **2**
15. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13. pp. 740–755. Springer (2014) **10, 25**
16. Liu, L., Ren, Y., Lin, Z., Zhao, Z.: Pseudo numerical methods for diffusion models on manifolds. arXiv preprint arXiv:2202.09778 (2022) **1**
17. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. Advances in Neural Information Processing Systems **35**, 5775–5787 (2022) **1, 11, 18**

18. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. arXiv preprint arXiv:2211.01095 (2022) [1](#), [11](#)
19. Meng, C., Rombach, R., Gao, R., Kingma, D., Ermon, S., Ho, J., Salimans, T.: On distillation of guided diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14297–14306 (2023) [2](#)
20. Nash, C., Menick, J., Dieleman, S., Battaglia, P.W.: Generating images with sparse representations. arXiv preprint arXiv:2103.03841 (2021) [11](#)
21. Peebles, W., Xie, S.: Scalable diffusion models with transformers. arXiv preprint arXiv:2212.09748 (2022) [2](#)
22. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PmLR (2021) [11](#)
23. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022) [1](#), [10](#)
24. Ronneberger, O., Fischer, P., Brox, T.: 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. pp. 234–241. Springer (2015) [2](#)
25. Sabour, A., Fidler, S., Kreis, K.: Align your steps: Optimizing sampling schedules in diffusion models. arXiv preprint arXiv:2404.14507 (2024) [1](#)
26. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. Advances in neural information processing systems **29** (2016) [11](#)
27. Salimans, T., Ho, J.: Progressive distillation for fast sampling of diffusion models. arXiv preprint arXiv:2202.00512 (2022) [2](#), [19](#)
28. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International conference on machine learning. pp. 2256–2265. PMLR (2015) [1](#)
29. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. arXiv preprint arXiv:2010.02502 (2020) [1](#), [11](#), [18](#)
30. Song, K., Leng, Y., Tan, X., Zou, Y., Qin, T., Li, D.: Transcormer: Transformer for sentence scoring with sliding language modeling. Advances in Neural Information Processing Systems **35**, 11160–11174 (2022) [1](#)
31. Song, Y., Dhariwal, P., Chen, M., Sutskever, I.: Consistency models (2023) [2](#), [19](#)
32. Song, Y., Ermon, S.: Generative modeling by estimating gradients of the data distribution. Advances in neural information processing systems **32** (2019) [1](#)
33. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. arXiv preprint arXiv:2011.13456 (2020) [1](#), [11](#), [18](#)
34. Tong, V., Hoang, T.D., Liu, A., Broeck, G.V.d., Niepert, M.: Learning to discretize denoising diffusion odes. arXiv preprint arXiv:2405.15506 (2024) [2](#)
35. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017) [4](#)
36. Wang, G., Cai, Y., Li, L., Peng, W., Su, S.: Pfdiff: Training-free acceleration of diffusion models through the gradient guidance of past and future. arXiv preprint arXiv:2408.08822 (2024) [1](#)

37. Wang, G., Peng, W., Li, L., Chen, W., Cai, Y., Su, S.: Diffusion sampling correction via approximately 10 parameters. arXiv preprint arXiv:2411.06503 (2024) [2](#)
38. Xia, M., Shen, Y., Lei, C., Zhou, Y., Yi, R., Zhao, D., Wang, W., Liu, Y.j.: Towards more accurate diffusion model acceleration with a timestep aligner. arXiv preprint arXiv:2310.09469 (2023) [2](#)
39. Xue, S., Liu, Z., Chen, F., Zhang, S., Hu, T., Xie, E., Li, Z.: Accelerating diffusion sampling with optimized time steps. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8292–8301 (2024) [1](#)
40. Yin, T., Gharbi, M., Zhang, R., Shechtman, E., Durand, F., Freeman, W.T., Park, T.: One-step diffusion with distribution matching distillation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 6613–6623 (2024) [2](#), [12](#), [24](#)
41. Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., Xiao, J.: Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv preprint arXiv:1506.03365 (2015) [10](#), [25](#)
42. Zhang, Q., Chen, Y.: Fast sampling of diffusion models with exponential integrator. arXiv preprint arXiv:2204.13902 (2022) [1](#), [11](#)
43. Zhao, W., Bai, L., Rao, Y., Zhou, J., Lu, J.: Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. Advances in Neural Information Processing Systems **36** (2024) [1](#), [11](#)
44. Zheng, K., Lu, C., Chen, J., Zhu, J.: Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. Advances in Neural Information Processing Systems **36**, 55502–55542 (2023) [1](#), [11](#)
45. Zhou, Z., Chen, D., Wang, C., Chen, C.: Fast ode-based sampling for diffusion models in around 5 steps. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7777–7786 (2024) [2](#), [5](#), [7](#), [11](#), [24](#)
46. Zhou, Z., Chen, D., Wang, C., Chen, C., Lyu, S.: Simple and fast distillation of diffusion models. arXiv preprint arXiv:2409.19681 (2024) [2](#), [12](#)
47. Zhu, B., Wang, R., Zhao, T., Zhang, H., Zhang, C.: Distilling parallel gradients for fast ode solvers of diffusion models. In: International Conference on Computer Vision (ICCV) (2025) [11](#)

A Complementary Background

A.1 Overview of Diffusion Models

We consider a continuous-time diffusion process $\{x_t\}_{t \in [0,1]}$ that gradually perturbs an initial data sample $x_0 \sim p_{\text{data}}(x)$ into pure noise as time t increases. The forward diffusion (noise-adding process) can be formalized by the *Itô* SDE [33]:

$$dx_t = f(x_t, t)dt + g(t)dw_t, \quad (7)$$

where $f(x_t, t)$ is a drift term, $g(t) \geq 0$ is the diffusion coefficient, and w_t is a standard Wiener process. Let $p_t(x)$ denote the marginal density of x_t . Under mild regularity conditions, one can derive a corresponding reverse-time SDE that transforms noise back into data:

$$dx_t = \left[f(x_t, t) - g(t)^2 \nabla_x \log p_t(x_t) \right] dt + g(t) d\bar{w}_t, \quad (8)$$

where \bar{w}_t is a standard Wiener process running backward in time, and $\nabla_x \log p_t(x_t)$ is the *score function* of the distribution. In practice, since $p_t(x)$ is intractable, training a neural network $s_\theta(x_t, t)$ to approximate the score $\nabla_x \log p_t(x_t)$ via denoising score matching on noisy data is a solution. At inference time, one can generate new samples by starting from $x_T \sim \mathcal{N}(0, I)$ and numerically integrating either the reverse SDE Eq. (8) (a stochastic generative process) or an equivalent ODE that shares the same marginals. This deterministic counterpart, known as the *probability-flow ODE* [33], is given by

$$\frac{dx_t}{dt} = f(x_t, t) - \frac{1}{2}g(t)^2 s_\theta(x_t, t), \quad (9)$$

and yields trajectories with the same distribution as the SDE. Solving this ODE corresponds to a deterministic sampling procedure. For example, the Denoising Diffusion Implicit Models (DDIM) [29] sampler can be viewed as a first-order discretization of Eq. (9). In either case, the integration from x_T to x_0 is carried out in a finite number of steps, each requiring an evaluation of the network s_θ . We refer to the total number of model evaluations as the **number of function evaluations (NFE)**. Achieving high sample fidelity typically requires a large NFE (often hundreds of evaluations), which leads to slow sampling. To mitigate this issue, a variety of *training-free* acceleration techniques have been explored. These methods focus on improving the sampler itself without modifying the learned model. For example, one can apply higher-order numerical solvers or use optimized time-step schedules to integrate Eq. (8) or Eq. (9) more efficiently. The DDIM sampler is a popular first-order method that can reduce sampling time compared to the original diffusion process, but it still usually needs on the order of hundreds of steps for high-quality results. More recently, dedicated high-order solvers such as *DPM-Solver* [17] have been developed to further cut down the required steps, often achieving acceptable image quality with only about 10–20 function evaluations. Nevertheless, even these advanced samplers suffer noticeable degradation in fidelity when the NFE budget is extremely low (e.g.

~ 10), due to accumulating integration error. This limitation motivates *training-based* approaches that learn to preserve diffusion model performance under very low NFE regimes.

A.2 Trajectory Distillation

Maintaining high generation quality with very few sampling steps has prompted the development of several *training-based* acceleration methods that augment the diffusion model through an extra training phase. Notable examples include *progressive distillation* [27], which gradually distills a model to use fewer steps by repeatedly halving the number of steps, and *consistency distillation* [11, 31], which trains the model to produce identical outputs whether one uses many small steps or a single large step. Despite their differing formulations, these methods are all built upon the core concept of **trajectory distillation**. Trajectory distillation aims to use a high-NFE “teacher” trajectory to guide a low-NFE “student” model. In essence, one first generates a reference trajectory of intermediate states using a large number of steps, and then trains a student model so that when it uses a much smaller number of steps, its intermediate results mimic the teacher’s trajectory. Concretely, let $\{\hat{x}_{t_i}\}_{i=0}^{N-1}$ denote the states along a high-NFE trajectory (with N steps) obtained from the teacher process. The student model is optimized such that for a reduced number of steps N (much smaller integration intervals), it produces states x_{t_i} that stay close to the teacher’s states \hat{x}_{t_i} at the corresponding time points. By aligning the student’s denoising trajectory with the teacher’s trajectory, the final sample quality can be preserved even under aggressive step reduction. Algorithm 2 provides a pseudocode of the trajectory distillation training procedure.

Algorithm 2 Trajectory Distillation

Input: Teacher trajectory $\{\hat{x}_{t_i}\}_{i=0}^{N-1}$, ODE solver S , student model s_θ , number of steps N , loss function \mathcal{L} , distance metric \mathcal{D} .

- 1: Initialize $x_{t_0} = \hat{x}_{t_0}$
- 2: **for** $i = 0$ to $N - 2$ **do**
- 3: **repeat**
- 4: $x_{t_{i+1}} = S(x_{t_i}, s_\theta, t_i, t_{i+1})$
- 5: $\mathcal{L} = \mathcal{D}(x_{t_{i+1}}, \hat{x}_{t_{i+1}})$
- 6: Update θ by taking $\nabla_\theta \mathcal{L}$ (optimize student)
- 7: **until** converged
- 8: $x_{t_{i+1}} = S(x_{t_i}, s_\theta, t_i, t_{i+1})$ // advance to next step
- 9: **end for**
- 10: **return** Trained student model s_θ

B Experiments

We present all remaining experimental results. In Sec. B.1 we report comparisons on the remaining datasets (ImageNet-64 and LSUN Bedroom). In Sec. B.2 we provide more detailed results, including additional evaluation metrics. In Sec. B.3 we give the full experimental configurations and hyperparameter details, include training loss curve and intermediate visualization.

B.1 Additional Results

We provide the comparison of lightweight methods for ImageNet-64 and LSUN Bedroom in Tab. 8, and for comparison of distillation-based acceleration, are in Tab. 9 and Tab. 10, respectively.

Table 8: FID evaluation on ImageNet-64 and LSUN Bedroom Datasets

Method	ImageNet-64				LSUN Bedroom			
	NFE							
	3	4	5	6	3	4	5	6
DDIM	75.91	52.53	43.86	24.33	86.13	54.53	34.34	25.26
iPNDM	52.17	28.95	15.61	10.60	80.99	43.89	26.65	20.71
UniPC	85.31	49.69	20.87	12.02	112.3	49.00	23.20	11.64
DEIS	40.31	20.26	12.45	8.94	52.46	23.65	14.42	10.74
DPM								
+Solver-2	140.20	129.74	42.41	43.48	210.60	210.41	80.60	80.19
+Solver++(3M)	91.52	56.31	25.49	15.05	111.90	49.46	23.15	12.28
AMED								
+Solver	38.10	32.69	10.74	10.63	58.21	–	13.20	–
+Plugin	28.06	23.55	13.83	12.05	101.50	–	25.68	–
EPD								
+Solver	18.28	–	6.35	–	13.21	–	7.52	–
+Plugin	19.89	–	8.17	–	14.12	–	8.26	–
MTEO (Ours)								
+DDIM	7.42	4.87	4.11	3.81	12.99	7.01	6.03	4.44
+iPNDM	9.47	5.46	4.72	4.12	23.71	13.43	8.46	6.25
+DPM-Solver++(3M)	7.07	5.10	4.66	4.19	20.77	14.94	6.56	6.45

B.2 Detailed Results

We provide the detailed comparison of CIFAR-10, FFHQ, ImageNet-64 and LSUN Bedroom in Tab. 12, Tab. 13, Tab. 14 and Tab. 11, respectively.

Table 9: Comparison MTEO against Distillation-Based acceleration on ImageNet-64 Dataset. We report +DDIM as main results. Hours denote solo RTX3090 GPU hour.

Method	Model Size	Parameter	Percent	NFE	FID	Hours
DMD-cond	–	–	–	1	2.62	–
CTM	1126.4MB	1228.8MB	109%	1	2.06	1660.86
ECM-S	1126.4MB	1126.4MB	100%	1	5.51	76.50
ECM-S	1126.4MB	1126.4MB	100%	2	3.18	76.50
SFD	1126.4MB	1126.4MB	100%	2	10.25	6.15
SFD	1126.4MB	1126.4MB	100%	3	6.35	8.53
SFD	1126.4MB	1126.4MB	100%	4	4.99	11.07
SFD	1126.4MB	1126.4MB	100%	5	4.33	13.17
SFD-v	1126.4MB	1126.4MB	100%	2	9.47	43.49
SFD-v	1126.4MB	1126.4MB	100%	3	5.78	43.49
SFD-v	1126.4MB	1126.4MB	100%	4	4.72	43.49
SFD-v	1126.4MB	1126.4MB	100%	5	4.21	43.49
MTEO	1126.4MB	0.42MB	0.037%	3	7.42	3.32
MTEO	1126.4MB	0.53MB	0.047%	4	4.87	3.65
MTEO	1126.4MB	0.63MB	0.056%	5	4.11	4.06
MTEO	1126.4MB	0.74MB	0.066%	6	3.81	4.32

Table 10: Comparison MTEO against Distillation-Based acceleration on LSUN Bedroom Dataset. We report +DDIM as main results.

Method	Model Size	Parameter	Percent	NFE	FID
SFD	2007.04MB	2007.04MB	100%	2	10.39
SFD	2007.04MB	2007.04MB	100%	3	6.42
SFD	2007.04MB	2007.04MB	100%	4	5.26
SFD	2007.04MB	2007.04MB	100%	5	4.73
SFD-v	2007.04MB	2007.04MB	100%	2	9.25
SFD-v	2007.04MB	2007.04MB	100%	3	5.36
SFD-v	2007.04MB	2007.04MB	100%	4	4.63
SFD-v	2007.04MB	2007.04MB	100%	5	4.33
MTEO	2007.04MB	0.58MB	0.029%	3	7.02
MTEO	2007.04MB	0.72MB	0.036%	4	5.32
MTEO	2007.04MB	0.87MB	0.043%	5	5.93
MTEO	2007.04MB	1.01MB	0.050%	6	5.98

Table 11: Detailed FID results on LSUN Bedroom

Method	Schedule	ρ	NFE			
			3	4	5	6
DDIM	polynomial	7	86.13	54.53	34.34	25.26
iPNDM	polynomial	7	80.99	43.89	26.65	20.71
DPM-Solver++(3M)	logsnr	–	111.90	49.46	23.15	12.28
MTEO (Ours)						
+DDIM	polynomial	7	12.99	7.01	6.03	4.44
+iPNDM	polynomial	7	23.71	13.43	8.46	6.25
+DPM-Solver++(3M)	logsnr	–	20.77	14.94	6.56	6.45
MTEO (Ours)						
+DDIM	time_uniform	1	7.02	5.32	5.93	5.98
+iPNDM	time_uniform	1	13.94	9.57	7.41	6.02
+DPM-Solver++(3M)	time_uniform	1	6.23	5.37	5.93	7.03

Table 12: Detailed FID results on CIFAR-10. We report different types of schedule on +DDIM, +iPNDM and +DPM++(3M). The time_uniform2 schedule perform best which also report as main result.

Method	Schedule	ρ	NFE			
			3	4	5	6
DDIM	polynomial	7	93.36	67.40	49.66	36.08
iPNDM	polynomial	7	47.98	24.81	13.59	7.05
DPM-Solver++(3M)	logsnr	-	110.00	46.52	24.97	11.99
MTEO (Ours)						
+DDIM	polynomial	7	5.29	3.91	3.02	2.87
+iPNDM	polynomial	7	6.63	4.81	4.20	3.26
+DPM-Solver++(3M)	logsnr	-	5.54	3.87	3.38	3.60
DDIM	logsnr	-	121.70	73.20	53.53	38.19
iPNDM	logsnr	-	88.36	35.58	19.86	10.68
DPM-Solver++(3M)	polynomial	7	70.03	50.39	31.66	17.89
MTEO (Ours)						
+DDIM	logsnr	-	6.40	4.06	3.52	3.17
+iPNDM	logsnr	-	6.15	3.75	3.09	2.80
+DPM-Solver++(3M)	polynomial	7	5.93	4.85	3.96	3.51
DDIM	time_uniform	2	74.51	42.62	28.79	21.21
iPNDM	time_uniform	2	39.91	15.35	9.02	5.26
DPM-Solver++(3M)	time_uniform	2	64.98	29.92	15.47	9.41
MTEO (Ours)						
+DDIM	time_uniform	2	3.85	2.83	2.62	2.50
+iPNDM	time_uniform	2	4.83	3.52	3.01	2.74
+DPM-Solver++(3M)	time_uniform	2	3.91	3.11	2.98	2.66

Table 13: Detailed FID results on FFHQ. We report different types of schedule on +DDIM, +iPNDM and +DPM++(3M). The time_uniform schedule perform best which also report as main result.

Method	Schedule	ρ	NFE			
			3	4	5	6
DDIM	polynomial	7	78.16	57.37	43.85	35.15
iPNDM	polynomial	7	45.90	28.21	17.14	10.00
DPM-Solver++(3M)	logsnr	-	86.45	45.94	22.51	13.74
MTEO (Ours)						
+DDIM	polynomial	7	9.84	6.05	4.45	3.76
+iPNDM	polynomial	7	12.97	9.82	7.95	5.83
+DPM-Solver++(3M)	logsnr	-	9.46	9.17	5.01	4.43
MTEO (Ours)						
+DDIM	time_uniform	2	5.46	3.66	3.17	2.99
+iPNDM	time_uniform	2	7.46	5.63	4.31	3.72
+DPM-Solver++(3M)	time_uniform	2	5.29	3.65	3.37	3.13

Table 14: IS, FID, sFID, Precision and Recall evaluation on ImageNet-64 Dataset.

Method	schedule	ρ	Metric	NFE=3	4	5	6
DDIM	polynomial	7	IS	11.68	16.25	20.49	24.40
			FID	75.91	52.53	43.86	24.33
			sFID	70.94	47.43	33.80	26.53
			Precision	29.25%	38.20%	45.81%	51.15%
			Recall	30.24%	38.81%	44.21%	48.50%
iPNDM	polynomial	7	IS	15.37	23.54	31.34	36.06
			FID	52.17	28.95	15.61	10.60
			sFID	43.28	22.94	12.27	10.76
			Precision	37.92%	50.17%	60.43%	64.33%
			Recall	41.77%	51.65%	58.27%	59.67%
DPM++(3M)	logsnr	-	IS	10.45	16.25	25.56	31.64
			FID	91.52	56.31	25.49	15.05
			sFID	65.35	33.70	15.23	9.58
			Precision	36.59%	41.53%	55.91%	62.25%
			Recall	25.63%	41.79%	54.37%	59.14%
MTEO (Ours)							
+DDIM	polynomial	7	IS	40.69	46.10	46.99	46.73
			FID	11.61	6.94	5.37	4.82
			sFID	7.82	5.55	4.73	4.39
			Precision	63.67%	67.92%	69.68%	69.82%
			Recall	58.07%	61.00%	62.57%	63.19%
+iPNDM	polynomial	7	IS	37.54	42.78	45.28	47.11
			FID	14.45	10.07	7.86	5.87
			sFID	10.11	8.08	6.68	5.71
			Precision	61.78%	66.09%	67.49%	69.44%
			Recall	56.81%	58.65%	61.08%	62.19%
+DPM++(3M)	logsnr	-	IS	39.53	41.41	41.34	40.52
			FID	16.45	13.36	10.72	7.11
			sFID	8.12	5.42	4.74	4.87
			Precision	63.32%	66.60%	67.69%	67.09%
			Recall	58.12%	61.87%	62.86%	63.51%
+DDIM	time_uniform 2	2	IS	43.90	46.31	46.34	46.89
			FID	7.42	4.87	4.11	3.81
			sFID	5.98	4.76	4.55	4.30
			Precision	67.05%	69.74%	70.73%	71.03%
			Recall	59.41%	61.53%	62.26%	62.93%
+iPNDM	time_uniform 2	2	IS	40.51	44.09	44.73	45.86
			FID	9.47	5.46	4.72	4.12
			sFID	7.53	5.57	5.64	4.98
			Precision	64.96%	68.72%	69.75%	70.15%
			Recall	58.67%	61.97%	62.04%	62.88%
+DPM++(3M)	time_uniform 2	2	IS	43.28	43.98	43.03	43.80
			FID	7.07	5.10	4.66	4.19
			sFID	5.80	4.83	4.65	4.34
			Precision	67.69%	70.16%	70.59%	70.32%
			Recall	58.98%	61.40%	63.06%	63.58%

B.3 Configuration

We provide the full hyperparameters, checkpoint and stats file in Tab. 15, Tab. 16 and Tab. 17.

Table 15: Experimental settings for each dataset.

Dataset	Sampler	Steps Schedule	ρ	lr	lr_{min}	ε	ε_{min}	Patience	E_{max}	Train Seeds	Batch Size	Tea Sampler	Tea Steps
CIFAR10/FFHQ	DDIM	4 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	22					
		5 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	21					
		6 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	21					
	iPNDM	7 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	25					
		4 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	22					
		5 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	21					
		6 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	21					
	DPM++(3M)	7 poly/uni 7/2 2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	iPNDM	25					
		4 logsnr -2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	DPM++(3M)	22					
		5 logsnr -2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	DPM++(3M)	21					
	ImageNet64	DDIM	6 logsnr -2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	DPM++(3M)	21				
			7 logsnr -2e-2 1e-3 1e-2 1e-3	10	300	50000-50255	64	DPM++(3M)	25				
			4 poly/uni 7/2 1e-3 1e-3 1e-2 1e-3	10	300	50000-51023	64	iPNDM	22				
		iPNDM	5 poly/uni 7/2 1e-3 1e-3 1e-2 1e-3	10	300	50000-51023	64	iPNDM	21				
6 poly/uni 7/2 1e-3 1e-3 1e-2 1e-3			10	300	50000-51023	64	iPNDM	21					
7 poly/uni 7/2 1e-3 1e-3 1e-2 1e-3			10	300	50000-51023	64	iPNDM	25					
4 logsnr -1e-3 1e-3 1e-2 1e-3			10	300	50000-51023	64	DPM++(3M)	22					
DPM++(3M)	5 logsnr -1e-3 1e-3 1e-2 1e-3	10	300	50000-51023	64	DPM++(3M)	21						
	6 logsnr -1e-3 1e-3 1e-2 1e-3	10	300	50000-51023	64	DPM++(3M)	21						
	7 logsnr -1e-3 1e-3 1e-2 1e-3	10	300	50000-51023	64	DPM++(3M)	25						
LSUN Bedroom	DDIM	4 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	22					
		5 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	21					
		6 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	21					
	iPNDM	7 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	25					
		4 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	22					
		5 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	21					
		6 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	21					
	DPM++(3M)	7 poly/uni 7/1 5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	iPNDM	25					
		4 logsnr -5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	DPM++(3M)	22					
		5 logsnr -5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	DPM++(3M)	21					
	MS COCO	DDIM	6 logsnr -5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	DPM++(3M)	21				
			7 logsnr -5e-2 5e-2 5e-3 5e-5	10	150	50000-50255	64	DPM++(3M)	25				
			4 discrete 1 3e-2 2e-2 1e-2 1e-3	10	200	0-255	64	DPM++(2M)	22				
		DPM++(2M)	5 discrete 1 3e-2 2e-2 1e-3 1e-4	10	200	0-255	64	DPM++(2M)	21				
6 discrete 1 3e-2 2e-2 1e-3 1e-4			10	200	0-255	64	DPM++(2M)	21					
7 discrete 1 3e-2 2e-2 1e-3 1e-4			10	200	0-255	64	DPM++(2M)	25					
4 discrete 1 3e-2 2e-2 1e-3 1e-4			10	200	0-255	64	DPM++(2M)	22					

C Time Estimates Compared to Distillation

C.1 Data Sources

Except for DMD [40] and ECM [6], all training-time data are taken from [45], which reports times on an A100 GPU. (DMD does not specify its training-time overhead.) According to ECM’s Table 8, training took 24 hours on one A6000 for CIFAR-10, and 8.5 hours on four H100s for ImageNet-64.

Table 16: Datasets, Checkpoints, and Licenses

Datasets	
Name	License
CIFAR-10 [12]	\
FFHQ 64X64 [10]	CC BY-NC-SA 4.0
ImageNet 64 [3]	Custom
ImageNet 256 [3]	Custom
LSUN Bedroom [41]	\
MS-COCO [15]	Custom
Checkpoints	
Name	License
edm-cifar10-32x32-uncond-vp.pkl	
edm-ffhq-64x64-uncond-vp.pkl	CC BY-NC-SA 4.0
edm-imagenet-64x64-cond-adm.pkl	
DiT-XL-2-256X256.pt	CC BY-NC 4.0
edm_bedroom256_ema.pt	\
v1-5-pruned-emaonly.ckpt	\

Table 17: Status Files

Status Files
Name
cifar10-32x32.npz
ffhq-64x64.npz
imagenet-64x64.npz
lsun_bedroom-256x256.npz
ms_coco-512x512.npz
VIRTUAL_imagenet64
_labeled.npz
VIRTUAL_imagenet256
_labeled.npz

C.2 GPU Time Conversions

A100 to RTX 3090. We profiled MTEO on CIFAR-10: on an RTX 3090 it required 0.59, 0.52, 0.54, 0.53 hours for 3, 4, 5, and 6 NFE, respectively; on an A100 these were 0.069, 0.097, 0.105, 0.118 hours. The ratios (A100/RTX) are roughly 0.59, 0.52, 0.54, 0.53, averaging 0.54. Thus we approximate $T_{\text{RTX3090}} \approx T_{\text{A100}}/0.54$.

H100/A6000 to RTX 3090. We assume H100 and A6000 are about $2.25\times$ and $1.3\times$ faster than the RTX 3090, respectively, and scale the reported times accordingly.

D Ablation Study

D.1 Batch Size

We evaluated the impact of training batch size on MTEO. The default is 64; we also tried 16, 32, and 128. Experiments on three ODE samplers (see Tab. 19) show that batch size has only a minor effect on final performance.

D.2 Training Set Size

The default MTEO uses 256 trajectories (seeds 50000–50255). We also trained with 64 trajectories (seeds 50000–50063) and 128 trajectories (50000–50127). As shown in Tab. 18, larger training sets generally yield slightly better results, indicating that MTEO benefits modestly from more diverse reference trajectories.

Table 18: Ablation on CIFAR10 based on FID. train seeds denote training datasets generated by corresponding random seeds.

Method	NFE	Train Seeds			
		50000-50063	50127	50255	
DDIM	3	5.99	4.92	5.29	
	4	4.53	3.77	3.91	
	5	3.84	3.17	3.02	
	6	3.36	2.93	2.87	
iPNDM	3	7.54	6.66	6.63	
	4	5.38	4.92	4.81	
	5	4.86	4.30	4.20	
	6	3.67	3.27	3.26	
DPM++(3M)	3	7.22	5.35	5.54	
	4	4.33	4.04	3.87	
	5	4.13	3.73	3.38	
	6	3.46	3.34	3.60	

Table 19: Ablation on CIFAR10 based on FID. Batch size denote the batch size per update during the training process.

Method	NFE	Batch Size			
		16	32	64	128
DDIM	3	4.85	5.02	5.29	5.60
	4	3.96	3.76	3.91	4.10
	5	3.13	3.06	3.02	3.18
	6	3.00	2.75	2.87	2.97
iPNDM	3	6.43	6.62	6.63	6.92
	4	5.02	4.70	4.81	4.97
	5	4.43	4.18	4.20	4.30
	6	3.48	3.25	3.26	3.26
DPM++(3M)	3	5.12	5.10	5.54	6.29
	4	4.80	4.62	3.87	4.73
	5	4.23	3.79	3.38	4.39
	6	3.84	3.82	3.60	3.59

D.3 Training Heuristics

We further evaluate the proposed training heuristics by comparing them against a simple *fixed-epoch* baseline. Specifically, using DDIM as the underlying sampler, we train each step’s embeddings for a fixed number of epochs ($E_{max} \in \{100, 300\}$) and contrast the resulting quality–cost trade-off with our adaptive scheme (Algorithm 3), which allocates training effort per step based on convergence.

As shown in Tab. 20, the fixed-epoch strategy exhibits a clear inefficiency: using a small budget (100 epochs) tends to under-train difficult steps and leaves performance on the table, while a large budget (300 epochs) improves quality but incurs unnecessary computation on steps that converge quickly. In contrast, our heuristics achieve a more favorable Pareto trade-off by (i) terminating updates once the relative loss improvement becomes negligible, while (ii) ensuring that later denoising steps are not prematurely stopped. Overall, the proposed scheme preserves the sample quality of the higher-budget fixed-epoch setting while substantially reducing the total training time, validating its role in balancing training overhead and final performance.

D.4 Sensitivity to Teacher Steps

We varied the length of the teacher trajectory. Tab. 21 shows that increasing or decreasing the teacher sampling steps by a small amount changes the FID only slightly. Even halving the teacher steps in the 3-NFE setting worsened FID by only 0.31. This suggests our main training configuration is well balanced.

Algorithm 3 Training MTE via Trajectory Distillation (specific)

Input: Teacher trajectory $\{\hat{x}_{t_i}\}_{i=0}^{N-1}$, time schedule $\{t_i\}_{i=0}^{N-1}$, ODE solver S , frozen denoiser ϵ_θ , distance metric \mathcal{D} , layer-wise embeddings $\{\Phi_i\}_{i=0}^{N-1}$ with $\Phi_i = \{\phi_{i,\ell}\}_{\ell=1}^L$, Threshold ϵ , Patience P , maximum epoch E_{max} .

Output: Optimized embeddings $\{\Phi_i\}_{i=0}^{N-1}$.

```

1:  $x_{t_0} \leftarrow \hat{x}_{t_0}$ 
2: for  $i = 0$  to  $N - 2$  do
3:    $c = 0, p = 0$ 
4:   while  $p < P$  and  $c < E_{max}$  do
5:      $x_{t_{i+1}} \leftarrow S(x_{t_i}, \epsilon_\theta, t_i, t_{i+1}, \Phi_i)$ 
6:      $\mathcal{L} \leftarrow \mathcal{D}(x_{t_{i+1}}, \hat{x}_{t_{i+1}})$ 
7:     Update  $\Phi_i$  using  $\nabla_{\Phi_i} \mathcal{L}$ 
8:     if  $c == 0$  then
9:        $\mathcal{L}_{pre} \leftarrow \mathcal{L}$ 
10:    else if  $(\mathcal{L}_{pre} - \mathcal{L}) / \mathcal{L}_{pre} < \epsilon$  then
11:       $p = p + 1$ 
12:    else
13:       $p = 0$ 
14:    end if
15:     $c = c + 1$ 
16:  end while
17:   $x_{t_{i+1}} \leftarrow S(x_{t_i}, \epsilon_\theta, t_i, t_{i+1}, \Phi_i)$ 
18: end for
19: return  $\{\Phi_i\}_{i=0}^{N-1}$ 

```

Table 20: Ablation of training heuristics on CIFAR-10. We use +DDIM as baseline and compare with fixed training epochs 100 and 300. To demonstrate the efficiency of our mechanism, we also report training time based on RTX3090 GPU hours.

Epochs	NFE	FID	Hours
100		5.37	0.10
300	3	4.96	0.30
Ours		5.29	0.13
100		4.14	0.13
300	4	3.84	0.40
Ours		3.91	0.18
100		3.26	0.17
300	5	3.18	0.50
Ours		3.02	0.19
100		3.03	0.20
300	6	2.99	0.61
Ours		2.87	0.22

Table 21: Comparison across different Teacher Steps on CIFAR-10. We use +DDIM as baseline.

Teacher Steps	NFE	FID
28		5.19
22 (Ours)	3	5.29
16		5.27
13		5.60
29		3.85
21 (Ours)	4	3.91
17		3.96
13		4.26
31		3.02
21 (Ours)	5	3.02
16		3.26
11		4.30
31		2.86
25 (Ours)	6	2.87
19		2.98
13		3.53

E Application to Stable Diffusion and DiT

E.1 Stable Diffusion

Stable Diffusion exhibits two main issues: (1) In the medium-step regime, increasing NFE often yields diminishing or negative returns, indicating a sampling threshold. (2) In the very-low-step regime, the model can converge to suboptimal modes. For example, with 4 steps it converges early, and even adding many mid-steps between 3 and 4 NFEs does not escape this mode; only after ~ 15 NFEs does it reach the optimal mode. We attribute this to the complexity of the conditional generation task, which may limit MTEO’s corrective range. To mitigate this, we propose two extensions: (i) jointly optimizing the “unconditional_condition” embedding for each step alongside MTE, providing stronger guidance; (ii) combining MTEO with a timestep search strategy to steer the trajectory toward the optimal mode. In this work we implement (i) and defer (ii) to future work.

E.2 DiT (Diffusion Transformers)

The DiT codebase supports only low-order samplers (DDPM/DDIM), making medium-step results poor. We trained MTEO using a teacher trajectory of $100 \rightarrow 3$, $101 \rightarrow 4$, $101 \rightarrow 5$, $103 \rightarrow 6$ and corresponding student NFEs of 4, 5, 6, 7. The overall time-conditioning pipeline remains the same (with AdaLayer-Norm replacing FiLM). We trained each step’s embeddings for 300 epochs (LR 2×10^{-5} , batch 64) with no early stopping or extra heuristics, also including the “unconditional_condition” embedding. Under DiT, MTEO achieves exceptionally strong results, which we attribute to the Transformer’s inherent layer-wise design synergizing with our multi-layer embeddings.

F Discussion

F.1 On the Effective Dimensionality of Timestep Embeddings

As discussed in Sec. 3.3 and visualized in Fig. 5, the original timestep embeddings occupy a highly low-dimensional subset of the embedding space. Concretely, PCA on the embeddings of 121 timesteps embeddings shows that the first two principal components explain **77.72%** and **19.41%** of the variance, accounting for **97.14%** in total. In contrast, the FiLM modulation parameters that directly affect layer-wise feature processing, i.e., $(\alpha_\ell, \beta_\ell)$, exhibit substantially higher effective dimensionality: even the first 30 principal components explain only **90.49%** of the variance.

A natural interpretation is that the conventional timestep embedding is generated from a *scalar* variable t . Although sinusoidal encoding and an MLP map t to a high-dimensional vector, the resulting embeddings across time lie on a low-dimensional manifold induced by a one-dimensional input; PCA then recovers a very low-dimensional linear approximation of this trajectory. This creates

a potential representational bottleneck when the downstream FiLM parameters vary in a richer, higher-dimensional manner across layers and time.

We further apply the same PCA analysis to the learned MTE embeddings. As shown in Fig. 5, MTE substantially increases the effective dimensionality of the embedding space: the first 30 principal components explain only **84.53%** of the variance. This indicates that the optimized embeddings are no longer constrained to a narrow low-dimensional subset, which may partially explain the strong empirical performance of MTEO in few-step sampling.

Interestingly, we also observe a small but consistent change in the geometry of the optimized FiLM parameters: after optimization, the cumulative explained variance of the first 30 principal components increases from **90.49%** to **93.49%**. This suggests that MTEO may make the effective modulation patterns slightly more structured (i.e., variance concentrated in fewer components), potentially reflecting a more coherent use of FiLM degrees of freedom.

F.2 Implications for Interpreting Diffusion Networks

Neural networks demonstrate remarkable capabilities, yet their internal mechanisms often remain opaque. An intriguing empirical finding from our experiments is that MTEO can reach sampling quality comparable to distillation-based acceleration methods while training fewer than **0.2%** additional parameters, whereas many distillation approaches require updating a large fraction (often all) of the model parameters.

From a mechanistic perspective, MTEO primarily acts by adjusting feature-wise affine modulation (FiLM) within the network: it changes how intermediate features are scaled and shifted at different layers and time steps. Since different layers are known to encode different types of information and operate at different spatial/semantic granularities, even simple gain-control-like adjustments can have surprisingly large downstream effects on the denoising trajectory. This observation suggests a promising direction for future work: analyzing the learned modulation patterns across layers and time (*e.g.*, which blocks and which denoising stages are most sensitive) may provide actionable clues toward understanding diffusion models beyond a purely black-box view.

F.3 A Direct FiLM-Parameter Variant: MTEO-deep

As discussed throughout the paper, MTEO ultimately influences sampling by changing the effective FiLM parameters $(\alpha_\ell, \beta_\ell)$. This motivates a natural extension: directly optimizing $(\alpha_\ell, \beta_\ell)$ instead of learning the embeddings that generate them. We call this variant **MTEO-deep** and include its experimental results in Tab. 22.

We do not adopt MTEO-deep as the main method for two practical reasons. First, the parameterization and injection locations of FiLM-style modulation vary across diffusion backbones and implementations (*e.g.*, some blocks use bias-only conditioning, while others use scale-and-shift), which makes direct parameter optimization less portable and less unified across frameworks. Second,

directly parameterizing $(\alpha_\ell, \beta_\ell)$ can introduce a larger, architecture-dependent parameter footprint, whereas MTE leverages a compact embedding interface that is largely consistent across diffusion models (sinusoidal encoding followed by a small MLP). For simplicity and broad applicability, we therefore use MTEO as our primary approach and treat MTEO-deep as an optional extension.

Table 22: Comparison on CIFAR10 between MTEO and MTEO-deep. We train both methods with same configuration

Method	NFE	FID ↓	
		+MTEO	+MTEO-deep
DDIM	3	5.29	5.18
	4	3.91	3.93
	5	3.02	3.48
	6	2.87	3.05
iPNDM	3	6.63	6.57
	4	4.81	5.44
	5	4.20	4.81
	6	3.26	3.90
DPM++(3M)	3	5.54	11.52
	4	3.87	7.87
	5	3.38	3.60
	6	3.60	2.81

G Additional Trajectory Analyses

In this section, we provide further results on feature trajectories. This includes direct visualizations of trajectories(Figs. 7 and 9), detailed PCA variance tables for each layer’s features(Tab. 24), and more experiments validating FiLM’s corrective power across additional layers and inputs(Fig. 8). We also provide the training loss curve(Fig. 10) and visualization of intermediate training process(Fig. 11).

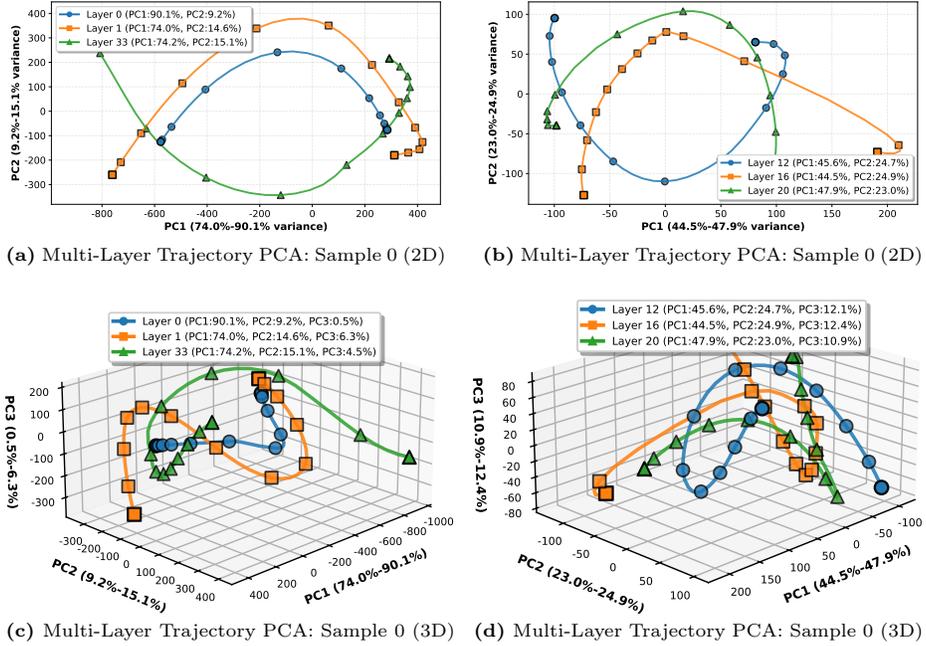


Fig. 7: Visualization of feature trajectory. See more example in appendix.

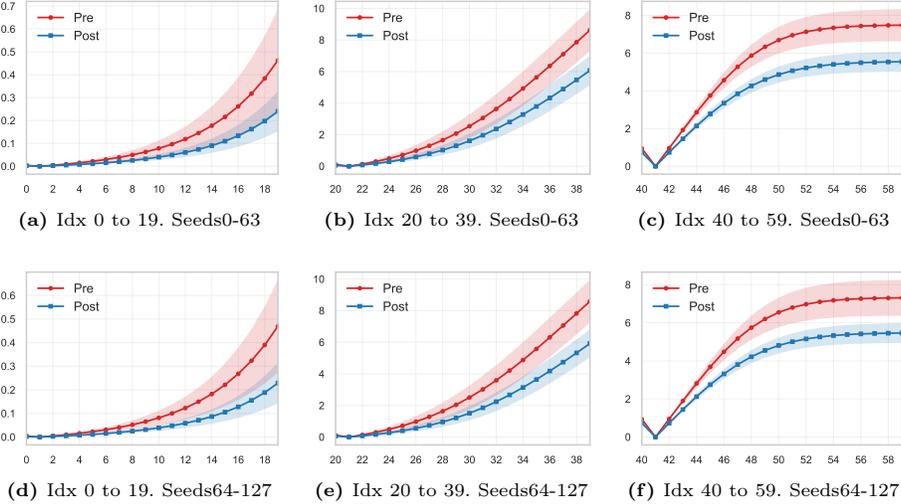


Fig. 8: Additional Experiments on FiLM’s Channel Correction Capability. We conducted experiments on two batches of data, 0-63 and 64-127, and reported the mean values before and after correction, along with the methods used. First, we generated the true trajectory using iPNDM with 60 NFE. Based on this, we simulated a 4-steps sampling process, corresponding to the steps from Idx0 → Idx19 → Idx39 → Idx59.

Table 23: Cumulative explained variance (%) of trajectory 0.

Index	PC ₁	PC _{1:2}	PC _{1:3}	PC _{1:4}	PC _{1:5}
0	90.13	99.33	99.82	99.94	99.98
1	73.99	88.59	94.93	98.36	99.29
2	67.04	82.43	90.70	96.40	98.30
3	59.60	78.60	88.29	94.59	97.48
4	53.04	75.56	86.07	92.05	96.24
5	51.04	72.83	84.53	92.07	96.18
6	48.45	70.89	83.98	91.69	95.92
7	48.06	70.72	83.65	91.71	95.61
8	46.51	69.54	83.20	91.22	95.13
9	44.48	67.86	81.72	90.07	94.47
10	45.38	70.74	83.13	91.21	94.79
11	46.06	70.93	83.23	91.11	94.78
12	45.63	70.34	82.45	90.45	94.41
13	44.99	69.76	81.85	90.07	94.26
14	42.10	68.67	83.16	91.31	94.92
15	40.08	65.92	80.99	89.44	94.00
16	44.46	69.35	81.80	89.77	93.77
17	41.51	69.27	83.16	89.77	93.50
18	47.53	70.22	82.53	90.01	93.67
19	47.97	71.44	82.29	89.97	93.67
20	47.87	70.88	81.75	89.79	93.39
21	47.08	69.57	81.64	89.96	93.39
22	47.40	68.79	81.87	90.12	93.54
23	45.90	66.46	80.18	88.34	92.70
24	45.06	65.38	79.18	87.63	92.32
25	44.96	64.49	79.45	87.79	92.58
26	42.52	63.03	78.57	87.08	92.35
27	44.36	64.57	79.87	87.59	92.78
28	42.33	62.65	79.79	87.44	92.67
29	44.28	68.00	80.93	88.30	93.40
30	44.64	70.60	81.29	89.13	93.55
31	50.39	75.44	84.50	90.69	94.21
32	53.16	75.76	85.36	91.48	94.68
33	74.25	89.34	93.85	96.68	97.95

Table 24: Cumulative explained variance (%) of trajectory 1.

Index	PC ₁	PC _{1:2}	PC _{1:3}	PC _{1:4}	PC _{1:5}
0	90.39	98.88	99.73	99.93	99.97
1	73.61	88.47	94.53	98.19	99.28
2	66.91	83.40	91.06	96.28	98.29
3	59.49	79.36	88.66	93.94	97.21
4	52.92	76.13	86.66	92.01	95.95
5	51.36	74.57	85.70	91.92	95.96
6	46.86	71.34	84.31	91.36	95.64
7	46.43	70.28	83.80	91.39	95.39
8	45.51	69.84	83.34	91.26	95.36
9	43.97	68.71	82.43	90.40	94.72
10	45.04	70.72	83.14	91.64	95.30
11	45.20	70.59	83.04	91.44	95.20
12	44.30	69.89	82.42	90.83	94.92
13	43.17	68.31	81.14	90.17	94.40
14	40.13	66.88	83.68	91.41	94.97
15	39.28	68.70	83.35	90.73	94.45
16	44.58	70.50	82.83	89.84	94.14
17	41.49	69.74	83.09	89.95	93.98
18	46.34	69.15	82.46	90.02	93.50
19	47.65	69.31	80.99	89.49	93.56
20	46.76	68.38	80.85	88.94	93.28
21	47.33	68.20	81.41	89.41	93.49
22	47.84	66.40	81.53	89.44	93.34
23	47.95	66.31	80.31	88.30	92.80
24	46.68	65.11	80.14	87.94	92.78
25	45.86	64.80	80.36	88.21	92.99
26	44.01	66.16	80.80	88.43	93.15
27	43.21	66.09	81.13	88.47	93.21
28	42.71	65.96	81.34	88.95	93.34
29	44.80	71.35	82.82	90.14	93.87
30	45.62	72.61	82.64	90.30	93.95
31	50.74	75.26	85.46	91.10	94.94
32	53.38	76.02	85.12	91.42	94.71
33	73.36	88.76	93.23	96.38	97.68

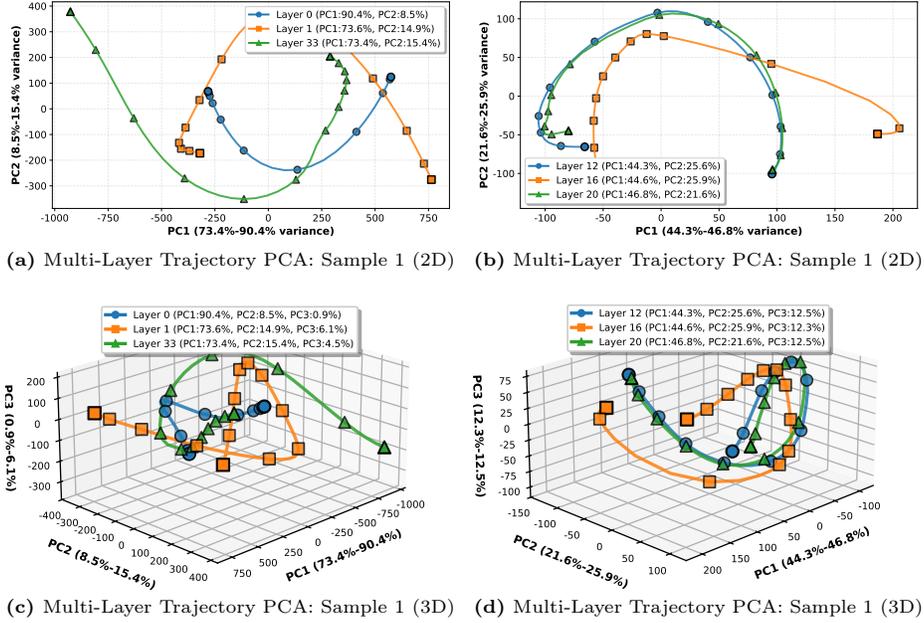


Fig. 9: Additional visualization of feature trajectory with trajectory 1

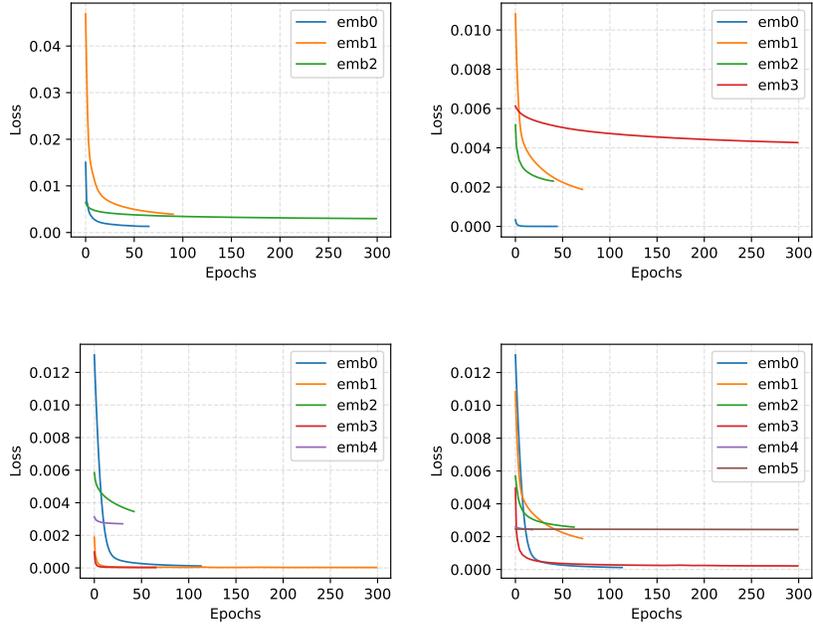


Fig. 10: Training loss curve. We display the training procedure of MTEO+DDIM on CIFAR10 with step 4, 5, 6, 7.

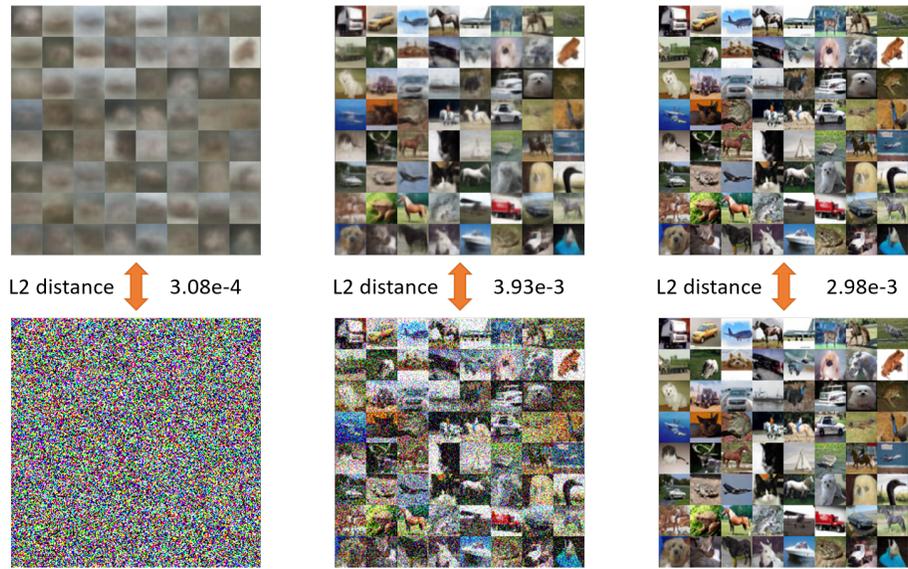


Fig. 11: The intermediate visualization results of training procedures with 4-step MTEO+DDIM.



Fig. 12: The phenomenon of mode non-convergence in Stable Diffusion sampling. As shown in the figure, the sampling result at 4 steps exhibits a pattern that is completely different from that at 11 steps, while the result at 7 steps demonstrates an intermediate transition.

H Visualization



(a) DDIM



(b) DDIM+METO



(c) iPNDM



(d) iPNDM+METO

Fig. 13: Samples on CIFAR10 32×32 with 3 NFE.



(a) DDIM



(b) DDIM+METO



(c) iPNDM



(d) iPNDM+METO

Fig. 14: Samples on FFHQ 64×64 with 3 NFE.

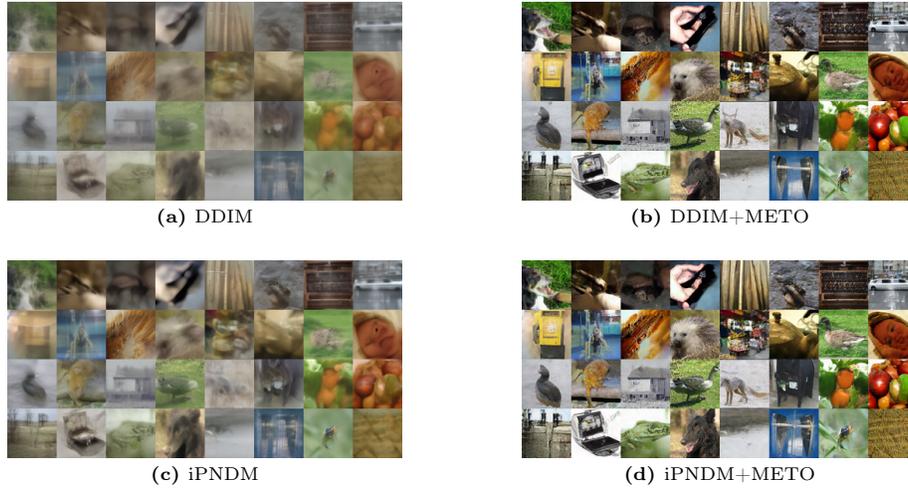


Fig. 15: Samples on ImageNet 64×64 with 3 NFE.

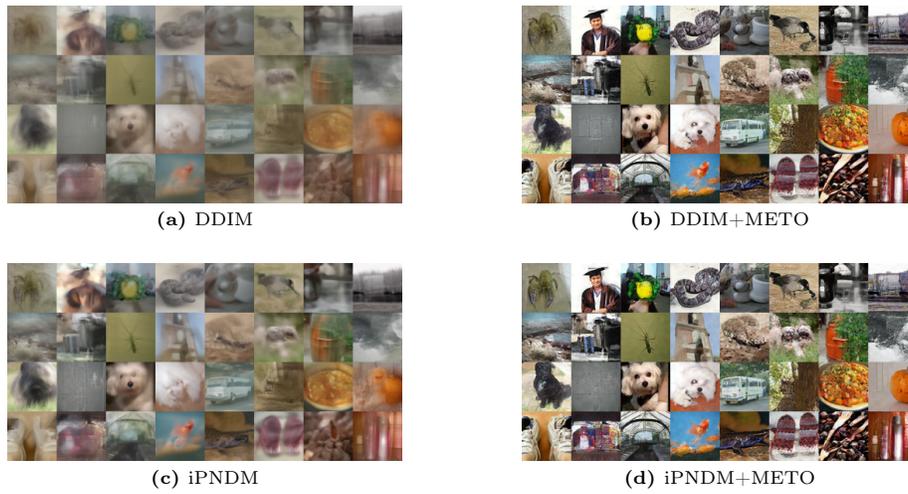


Fig. 16: Samples on ImageNet 64×64 with 3 NFE.



Fig. 17: Samples on ImageNet-256 256×256 with 3 NFE.



Fig. 18: Samples on ImageNet-256 256×256 with 6 NFE.

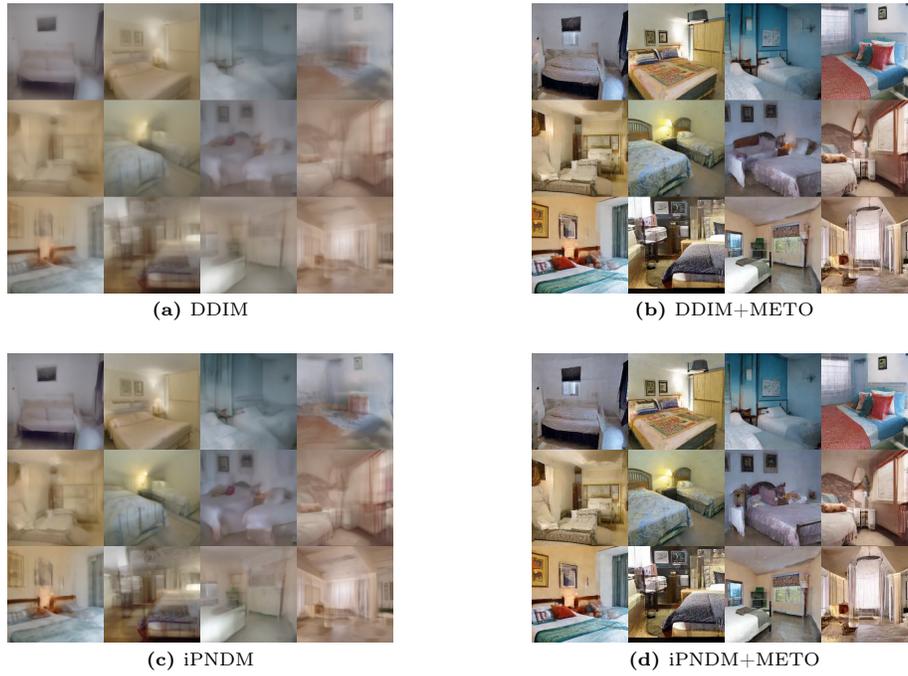


Fig. 19: Samples on LSUN Bedroom 256×256 with 3 NFE.

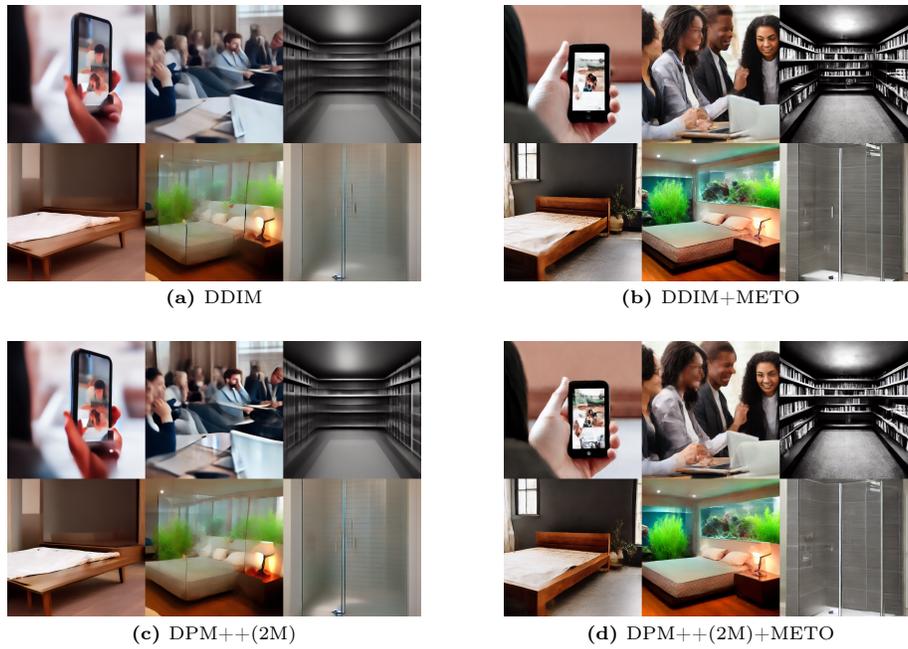


Fig. 20: Samples on MS COCO 512×512 with 3 NFE.