Diffusion on Demand: Selective Caching and Modulation for Efficient Generation

Hee Min Choi [1]*
chm.choi@samsung.com

Hyoa Kang [1 2]*
hyoa.kang@samsung.com

Dokwan Oh [1]
dokwan.oh@samsung.com

Nam Ik Cho [2]†
nicho@snu.ac.kr

[1] Samsung Electronics Co., Ltd.

[2] Seoul National University

Abstract

Diffusion transformers demonstrate significant potential for various generation tasks but are challenged by high computational cost. Recently, feature caching methods have been introduced to improve inference efficiency by storing features at certain timesteps and reusing them at subsequent timesteps. However, their effectiveness is limited as they rely only on choosing between cached features and performing model inference. Motivated by high cosine similarity between features across consecutive timesteps, we propose a cache-based framework that reuses features and selectively adapts them through linear modulation. In our framework, the selection is performed via a modulation gate, and both the gate and modulation parameters are learned. Extensive experiments show that our method achieves similar generation performance to the original sampler while requiring significantly less computation. For example, FLOPs and inference latency are reduced by $2.93\times$ and $2.15\times$ for DiT-XL/2 and by $2.83\times$ and $1.50\times$ for PixArt-α, respectively. We find that modulation is effective when applied to as little as 2% of layers, resulting in negligible computation overhead.

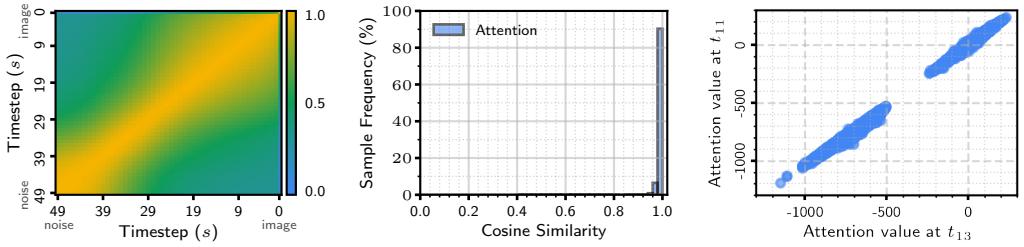
1 Introduction

Diffusion models [14, 45, 46] have emerged as promising generative methods across different modalities such as images [8, 37] in recent years. Across various backbone architectures for diffusion models, transformers [49] are increasingly popular for generating high-quality images [35, 1, 54, 3], videos [27, 36, 6] and 3D content [32, 2]. Although diffusion transformers leverage the scalability of the architecture, they are challenged by high computational cost and slow inference speed.

Numerous frameworks have been developed to improve inference efficiency in diffusion models. As the sampling cost scales proportionally with the number of sampling steps and the model size, efficiency improvements primarily aim to reduce the number of steps and network inference cost. Several studies reduce the number of sampling steps by optimizing the denoising trajectory [44, 25, 26] or employing distillation methods [39, 47, 28, 16, 50]. Other approaches mainly focus on model compression by pruning [15, 20, 11] or quantization [19, 48, 12, 4]. Different works use dynamic inference mechanisms by allocating models of varying sizes for different timesteps [53, 34].

*Equal contributions
†Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) Similarity heatmap (b) Similarity at t_s and t_{s+2} (c) 28th attention at t_{13} and t_{11}

Figure 1: Visualization of attention features in DiT-XL/2 with 50 DDIM timesteps $\{t_s\}_{s=0}^{49}$. Heatmap (a) reveals high cosine similarity in features across consecutive timesteps. Histogram (b) shows the cosine similarity distribution between feature pairs at a two-timestep interval $\{(t_s, t_{s+2})\}_{s=0}^{47}$, with most values near 1. Scatter plot (c) visualizes feature values of the 28th attention layer at timesteps 13 and 11, showing a strong linear relationship.

A new approach to dynamic inference in diffusion models involves a special caching mechanism. The main idea is to cache intermediate features at certain timesteps for reusing them at subsequent timesteps by leveraging high similarity between features at consecutive steps. Some caching-based methods for diffusion transformers [41, 5, 57, 56] rely on handcrafted deterministic strategies to select features for caching. In contrast, other studies [30, 42] propose to learn optimal caching schemes, yet their improvements are limited because they only exploit adjacent step redundancy. Efficiency gains in both training-free and learning-based methods for diffusion transformers are constrained, as they only involve choosing between cached features and performing network inference.

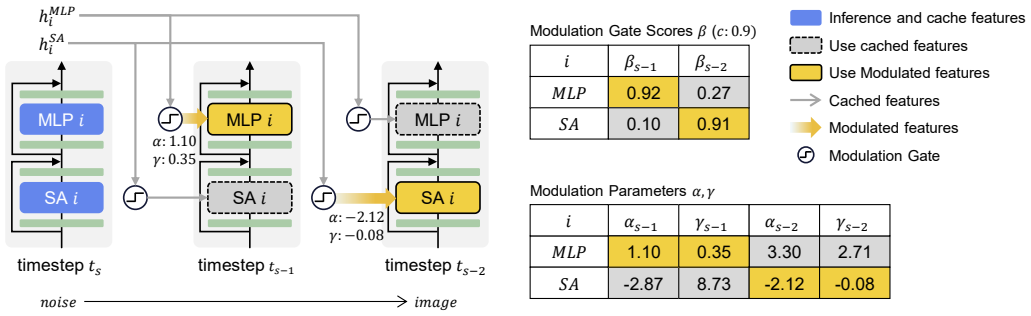
In this paper, we propose a cache-based framework for diffusion transformers that improves inference efficiency by reusing features and selectively adapting them through transformations. Our key idea builds on the observation, which is also noted in the previous work [41], that attention and multi-layer perceptron (MLP) features in diffusion transformer blocks exhibit high cosine similarity across consecutive timesteps (Figure 1). Instead of choosing between reusing features or performing inference, we propose leveraging *lightweight* linear modulation to approximate the original inference given the high cosine similarity.

In our framework, we learn modulation gate scores in a differentiable manner to decide whether to apply the modulation, while simultaneously training scale and translation parameters of the modulator. At inference time, we use a threshold to enable hard selection. Both the modulation gate and modulator are time-dependent but input invariant, enabling the construction of a static computation graph for inference similar to the prior work [30, 42]. Also, the optimization does not require model parameter updates, ensuring cost-efficiency and ease of implementation.

We perform experiments using popular diffusion transformers. Our framework obtains comparable generation performance to the original sampler with significantly less computational cost. For instance, FLOPs are reduced by $2.93\times$ and $2.83\times$ for DiT-XL/2 [35] and PixArt-α [3], respectively. The inference speed-up is $2.15\times$ and $1.50\times$ for DiT-XL/2 and PixArt-α. Also, the proposed method achieves superior generation quality compared to existing cache-based approaches and fewer-step samplers, while maintaining similar FLOPs. We find that the modulation is effective when applied to a small percentage of layers as low as 2%, resulting in negligible computational overhead.

In summary, we made the following contributions:

- We propose a cache-based framework for diffusion transformers that improves inference efficiency by reusing features and selectively adapting them through linear modulation. To our knowledge, this is the first method to approximate diffusion transformer inference using *lightweight* cache transformations.
- We show that applying the modulation to as little as 2% of layers is effective, leading to negligible computational overhead.
- Experiments demonstrate our method generally achieves similar generation performance with significantly less computation compared to the original sampler, while outperforming cache-based approaches and fewer-step samplers in generation quality under similar FLOPs.



i	β_{s-1}	β_{s-2}
MLP	0.92	0.27
SA	0.10	0.91

i	α_{s-1}	γ_{s-1}	α_{s-2}	γ_{s-2}
MLP	1.10	0.35	3.30	2.71
SA	-2.87	8.73	-2.12	-0.08

Figure 2: Example of the sampling process in the proposed framework with activation cycle $\mathcal{N}$=3 at i-th basic block. Given $\mathcal{N}$ consecutive timesteps $\{t_s, t_{s-1}, t_{s-2}\}$, network inference is performed at timestep t_s and cache features of attention h_i^{SA} and MLP h_i^{MLP} in the i-th basic block. Then, for each step $m \in \{s-1, s-2\}$ and layer, if the modulation gate score β_m is greater than a predefined threshold c, the linear modulation with scale α_m and translation γ_m is applied, and the modulated feature replaces the layer computation at timestep t_m. Otherwise, the cached feature itself does this.

2 Related Work

Transformers in diffusion. Transformers [49] have been adopted as an alternative to UNet [38] in diffusion models. Recently, DiT [35] demonstrates the scalability of diffusion transformers, enabling various applications such as text-to-image and video generation [3, 36, 27] and 3D generation [32, 2].

Efficiency improvement for diffusion. Diffusion sampling involves multiple network inferences, making it computationally inefficient. DDIM [44] improves the original DDPM [14] to non-Markovian cases, whereas DPM [25, 26] proposes advanced approximations of diffusion ODE solutions. Some methods [39, 47, 28, 16, 50] use distillation techniques to efficiently reduce the number of sampling steps. Model compression strategies have also been explored, such as network pruning [15, 11, 20] and quantization [19, 48, 12, 4]. Other studies use dynamic inference, allocating models of varying sizes to different steps [53, 34].

Cache in diffusion for image generation. Feature caching methods have been introduced to improve inference efficiency by storing features from previous timesteps and reusing them in subsequent timesteps, leveraging feature similarity. Early works focus on UNet architecture [31, 51]. One such work [51] applies a parameter-heavy channel-wise linear transformation as a post-processing to all cached features selected by a deterministic criterion, whereas our method jointly learns both whether to apply modulation and the corresponding lightweight, layer-wise modulation parameters. More recently, caching strategies have been extended to diffusion transformers [18, 43]. Training-free strategies utilize different features for caching such as attention features [23], attention and MLP features [41], feature residuals [5], and token-wise selected features [57, 56], yet they rely on handcrafted deterministic criteria for caching decisions. In contrast, other works [30, 42] learn optimal caching strategies but achieve limited improvement by only exploiting redundancy between adjacent steps. Although no prior work explored transformation of cached features in the image domain, a concurrent study [29] applies modulation in the video generation, where two cached features are combined using a timestep-dependent deterministic weighting function.

3 Method

We propose a cache-based framework for diffusion transformers that improves inference efficiency by reusing features and selectively modulating them through learned linear transformations. We begin with a preliminary overview of diffusion, then present caching and modulation mechanism, followed by the training objective that unifies these components.

3.1 Preliminary

Diffusion. The forward process of diffusion transforms a sample x_0 from the data distribution $q(x_0)$ by gradually adding Gaussian noise: $x_t = \kappa_t x_0 + \sigma_t \epsilon$ where $\epsilon \sim N(0, I)$ and κ_t and σ_t are noise

coefficients. In the reverse (denoising) process, given x_s at timestep $s > 0$ and $t < s$, x_t is computed as [25]:

$$x_t = \frac{\kappa_t}{\kappa_s} x_s - \kappa_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \epsilon_\theta \left(x_{t_\lambda(\lambda)}, t_\lambda(\lambda) \right) d\lambda \tag{1}$$

where $\lambda_t = \log(\kappa_t / \sigma_t)$ and $t_\lambda(\cdot)$ is the inverse function of λ_t that satisfies $t_\lambda(\lambda_t) = t$. Here, $\epsilon_\theta(\cdot)$ represents the learned model, which in our case, is the diffusion transformer. The integral term in (1) can be approximated by adopting Taylor expansion at λ_s in the first-order [44] or higher-orders [25].

Diffusion transformer. Diffusion transformers consist of a sequence of basic blocks with self-attention f^{SA}, multi-layer perceptron f^{MLP}, and cross-attention f^{CA} (for conditional generation) blocks: $f_L \circ \cdots \circ f_2 \circ f_1$ where $f_i = \{f_i^{\mathrm{SA}}, f_i^{\mathrm{CA}}, f_i^{\mathrm{MLP}}\}$ and L denotes the depth of the model. Each block can be represented as

$$f_i^l(z_t^{l,i}, t) = z_t^{l,i} + g_i(t) * h_i^l(z_t^{l,i}, t),\ l \in \{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}, \tag{2}$$

consisting of a residual connection. Here, $g_i(t)$ is a timestep t-conditioned scalar, h_i^l is the self-attention, cross-attention, or MLP layer in the i-th basic block, and $z_t^{l,i}$ denotes the input to the i-th l block at timestep t. For simplicity, we suppress the condition y in conditional generation.

3.2 Caching for Diffusion Transformers

The main idea of feature caching methods is to approximate $\epsilon_\theta(\cdot)$ by reusing intermediate features. We first describe the naive feature caching scheme [41]. For the first set of $\mathcal{N}$ consecutive timesteps $\{t_{T-1}, t_{T-2}, \ldots, t_{T-\mathcal{N}}\}$ in the T-step diffusion model, the naive caching performs network inference at the first timestep t_{T-1} and stores all intermediate features of self-attention, cross-attention and MLP layers, $h_i^l(z_{t_{T-1}}^{l,i}, t_{T-1})$ in (2). Then in the next $\mathcal{N} - 1$ timesteps, the corresponding operations

Algorithm 1 Training

Input: Data distribution $q(\cdot)$, diffusion model $\epsilon_\theta(\cdot)$, optimizer, ODE solver $\Psi(\cdot)$, total steps T, the step schedule $\{t_s\}_{s=0}^{T-1}$ in $\Psi(\cdot)$ and activation cycle $\mathcal{N}$
Randomly initialize β, α, and γ
Compute $r = (T-1)\%\mathcal{N}$
repeat
  $x_0 \sim q(x_0)$ and $n \sim \mathcal{U}[0, (T-1)//\mathcal{N} - 1]$
  $s \leftarrow \mathcal{N} * n + r$
    # Step t_s for calculating states for caching
  $x_s \sim N(x_s;\ \kappa_s x_0, \sigma_s^2 I)$
  $\epsilon_s \leftarrow \epsilon_\theta(x_s, t_s)$ and cache $h_i^l(\cdot, t_s)$'s in Eq (2)
  $d \sim \mathcal{U}[1, \mathcal{N} - 1]$
  $m \leftarrow s - d$
    # Step t_m for using cached states
  $\tilde{x}_m \leftarrow \Psi(\epsilon_s, t_s, t_m)$
  $x_{m+1} \sim N(x_{m+1};\ \kappa_{m+1} x_0, \sigma_{m+1}^2 I)$
  $\epsilon_{m+1} \leftarrow \epsilon_\theta(x_{m+1}, t_{m+1})$
  $x_m \leftarrow \Psi(\epsilon_{m+1}, t_{m+1}, t_m)$
  $\beta \leftarrow \mathrm{Sigmoid}(\beta)$
    # Optimize
  Calculate $\tilde{\epsilon}_\theta(\tilde{x}_m, t_m; \beta, \alpha, \gamma)$ by Eq (4) with Eq (5)
  $\mathcal{L} \leftarrow \|\tilde{\epsilon}_\theta(\tilde{x}_m, t_m) - \epsilon_\theta(x_m, t_m)\|_2^2 + \lambda \sum \beta$
  Compute gradients for β, α, γ with respect to loss $\mathcal{L}$ and update them with optimizer
until converged

Algorithm 2 Sampling

Input: Diffusion model $\epsilon_\theta(\cdot)$, modulation gate score β, modulation parameters α, γ, ODE solver $\Psi(\cdot)$, total steps T, the step schedule $\{t_s\}_{s=0}^{T-1}$ in $\Psi(\cdot)$, activation cycle $\mathcal{N}$ and threshold c
Compute $r = (T-1)\%\mathcal{N}$
$x_{T-1} \sim N(0, I)$
for $s = T-1$ **to** 0 **do**
  for l in $\{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}$ and $i = 1$ **to** L **do**
    if $s\%\mathcal{N} = r$ **then**
      Calculate $f_i^l(\cdot, t_s)$ and cache $h_i^l(\cdot, t_s)$'s in Eq (2)
        # Inference and cache features
    else
      $\beta_s^{l,i} \leftarrow \mathrm{Sigmoid}(\beta_s^{l,i})$
      $\beta_s^{l,i} \leftarrow 1_{\{\beta_s^{l,i} > c\}}$
      Calculate $\tilde{f}_i^l(\cdot, t_s)$ by Eq (5)
    end if
  end for
  if $s\%\mathcal{N} = r$ **then**
    $\epsilon_s \leftarrow \epsilon_\theta(x_s, t_s)$
      # Original inference
  else
    $\epsilon_s \leftarrow \tilde{\epsilon}_\theta(\tilde{x}_s, t_s; \beta, \alpha, \gamma)$
      # Use cached features
  end if
  $x_{s-1} \leftarrow \Psi(\epsilon_s, t_s, t_{s-1})$
end for
return x_0

are substituted with the cached features. That is, for $l \in \{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}$, $i = 1, \ldots, L$, and $t_m \in \{t_{T-2}, \ldots, t_{T-\mathcal{N}}\}$,

$$h_i^l(z_{t_m}^{l,i}, t_m) := h_i^l(z_{t_{T-1}}^{l,i}, t_{T-1}) \tag{3}$$

where $:=$ indicates the assignment operation. After completing the $\mathcal{N}$ timesteps, the feature cache resets and begins a new cycle by re-initializing the cache as described above. Here, $\mathcal{N}$ is called *activation cycle*.

The effectiveness of feature caching is attributed to high similarity between features at the same depths across adjacent timesteps. However, the features are not identical, resulting in degradation of the generation quality, for which fine-grained caching methods are proposed [30, 57]. Although the advanced approaches improve the quality issue, they offer limited gains in computational efficiency as they rely only on choosing between caching and network inference.

3.3 Cache Modulation

A key question now is how similarity between features can be effectively leveraged to emulate full T-step network evaluation $\{\epsilon_\theta(x_s, t_s)\}_{s=0}^{T-1}$. Our core idea is that we approximate some parts of network inference using transformations of the cached features, while the approximation would not increase computational cost too much. Specifically, for consecutive timesteps $\{t_s, t_{s-1}, \ldots, t_{s-\mathcal{N}+1}\}$ with $\mathcal{N}$ activation cycle and cached feature $h_i^l(z_{t_s}^{l,i}, t_s)$, we find a lightweight transformation $\mathcal{L}_{t_m}^{l,i}$ that approximates the original inference $f_i^l(z_{t_m}^{l,i}, t_m)$ in the basic block (2) by:

$$\tilde{f}_i^l(z_{t_m}^{l,i}, t_m) := z_{t_m}^{l,i} + g_i(t_m) * \mathcal{L}_{t_m}^{l,i}(h_i^l(z_{t_s}^{l,i}, t_s)), \tag{4}$$

where $l \in \{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}$, $i = 1, \ldots, L$ and $t_m \in \{t_{s-1}, \ldots, t_{s-\mathcal{N}+1}\}$. Based on observations from previous methods [30, 41] that feature differences vary across timesteps and layers, we use timestep-wise (t_m) and layer-wise (l, i) transformations.

We now specifically define the lightweight transformation. Motivated by high cosine similarity and a strong linear relationship between features among consecutive timesteps illustrated in Figure 1, we propose using a linear modulation scheme. Meanwhile, experimental results in the previous study [41] indicate that the naive caching mechanism (3) alone ensures decent generative performance, so we add a modulation gate to allow selective incorporation of the modulation. In summary, we define the lightweight transformation $\mathcal{L}_{t_m}^{l,i}$ in (4) by:

$$\mathcal{L}_{t_m}^{l,i}\left(h_i^l(z_{t_s}^{l,i}, t_s);\, \beta_m^{l,i}, \alpha_m^{l,i}, \gamma_m^{l,i}\right) = \left(1 - \beta_m^{l,i}\right) h_i^l(z_{t_s}^{l,i}, t_s) + \beta_m^{l,i}\left(\alpha_m^{l,i}\, h_i^l(z_{t_s}^{l,i}, t_s) + \gamma_m^{l,i}\right) \tag{5}$$

where $\alpha_m^{l,i} \in \mathbb{R}$ and $\gamma_m^{l,i} \in \mathbb{R}$ are scale and translation parameters of the modulation, and $\beta_m^{l,i} \in \{0, 1\}$ is the modulation gate score. We denote the approximation of the original network evaluation $\epsilon_\theta(x_m, t_m)$ using (4) with (5) by $\tilde{\epsilon}_\theta(x_m, t_m; \beta, \alpha, \gamma)$ where $\alpha = \{\alpha_m^{l,i}\}, \beta = \{\beta_m^{l,i}\}$ and $\gamma = \{\gamma_m^{l,i}\}$.

One simple way to find $\alpha_m^{l,i}$ and $\gamma_m^{l,i}$ in (5) is taking the least square solution for each timestep and layer in a data-driven way. Specifically, set $\beta_m^{l,i} = 1$, and for many input z's to i-th l layer at timestep t_m, find $\alpha_m^{l,i}$ and $\gamma_m^{l,i}$ such that

$$\text{minimize} \sum_z \left\| f_i^l(z, t_m) - \tilde{f}_i^l(z, t_m) \right\|^2. \tag{6}$$

Then, given $\alpha_m^{l,i}$ and $\gamma_m^{l,i}$, the modulation gate $\beta_m^{l,i}$ can be determined by comparing mean square errors between the full network inference output $\epsilon_\theta(x_m, t_m)$ and all possible approximations $\tilde{\epsilon}_\theta(x_m, t_m; \beta, \alpha, \gamma)$'s obtained from selection combinations $\beta = \{\beta_m^{l,i}\}$ where each $\beta_m^{l,i} \in \{0, 1\}$. However, evaluating all possible configurations is computationally too expensive, as the number of combinations grows exponentially with the model's depth L. Furthermore, this method cannot effectively handle the sequential nature of diffusion, where an error in a particular layer or timestep can propagate and affect subsequent layers as well as future timestep inferences.

3.4 Training Objective

To address this, we propose to learn the modulation gate score $\beta = \{\beta_m^{l,i}\}$ and modulator parameters $\alpha = \{\alpha_m^{l,i}\}$ and $\gamma = \{\gamma_m^{l,i}\}$. Recall that our goal is to find $\tilde{\epsilon}_\theta(x_m, t_m; \beta, \alpha, \gamma)$ that approximates the original inference $\epsilon_\theta(x_m, t_m)$ with minimal computational cost. We can write this as a regularized optimization problem. To be specific, given T timesteps $\{t_i\}_{i=0}^{T-1}$ and $\mathcal{N}$ activation cycle $\{t_s, t_{s-1}, \ldots, t_{s-\mathcal{N}+1}\}$ with caching step t_s, for $t_m \in \{t_{s-1}, \ldots, t_{s-\mathcal{N}+1}\}$,

$$\arg\min_{\beta,\alpha,\gamma} \|\tilde{\epsilon}_\theta(x_m, t_m; \beta, \alpha, \gamma) - \epsilon_\theta(x_m, t_m)\|^2 + \lambda \sum_{i,l} \beta_m^{l,i}\,, \tag{7}$$

where λ is the regularization hyperparameter, $l \in \{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}$ and $i = 1, \ldots, L$. Although β's need to be discrete at inference time, we design β to be continuous and normalized between $[0, 1]$ using the sigmoid operation to make the computation differentiable when training. After training, we use a threshold c to discretize β be either 0 or 1, where β turned to become a modulation gate. We note that the model parameters in the diffusion transformer are frozen and only modulation gate score β and modulator parameters α and γ are updated in our algorithm. We provide a pseudo code for training and sampling in Algorithms 1 and 2 and illustrate an example of the sampling process with activation cycle $\mathcal{N} = 3$ in Figure 2.

4 Experiments

4.1 Implementation Details

Model settings. We perform experiments using popular diffusion transformers across different generation tasks, including DiT-XL/2 [35] for class-conditional image generation and PixArt-α [3], Sana-0.6B [52] and SD3-medium [10] for text-to-image generation. We utilize diverse sampling methods with different number of sampling steps. DDIM [44] and DPM++ [26] are used for DiT-XL/2 and PixArt-α, respectively, and Flow-Euler [22] is adopted for Sana-0.6B and SD3-medium.

Training. We test multiple activation cycles $\mathcal{N} \in \{2, 3, \ldots, 10\}$. The optimization is performed only on the modulation gate score and modulator, leading to a very small number of trainable parameters. For example, the number of parameters is 3,276 for PixArt-α with 20 sampling steps and $\mathcal{N} = 3$. We use the training set of ImageNet [7] for DiT-XL/2 and COCO2014 [21] for text-to-image generation models. We train the modulation gate scores and modulator parameters on 4 H100 GPUs with AdamW optimizer [24] and learning rate 10^{-3} for 200K iterations.

Evaluation. For class-conditional image generation, we generate 50K images with a resolution of 256×256 by randomly sampling 1K classes on ImageNet. For text-to-image generation, we randomly select 5K/30K captions from the COCO2014 validation set and generate one image per caption, testing at different resolutions: 256×256, 512×512 and 1024×1024. We use the default classifier-free guidance for each model (e.g., 1.5 for DiT-XL/2). Unless specified, the threshold c for the modulation gate is set to 0.9. To evaluate generation quality, we use Frechet Inception Distance (FID), sFID [33], Inception Score (IS) [40], precision and recall [17], and we also report CLIPScore [13] for text-to-image generation. FLOPs are calculated using pytorch-OpCounter [55], and inference latency is measured on a single H100 GPU. More implementation details are in the appendix.

Table 1: Qualitative results for class-conditional 50K image generation with DiT-XL/2 and DDIM 50 steps on ImageNet 1K classes. Our method requires significantly less computation than the baseline to achieve comparable image generation quality and outperforms other approaches with similar FLOPs in terms of generation performance.

Method	FLOPs↓	Speed↑	FID↓	sFID↓	IS↑	Precision↑	Recall↑
DiT-XL/2 (50 steps, baseline)	11.44 T	1.00×	2.29	4.32	239.54	0.81	0.60
50% steps	5.72 T	1.99×	2.92	**4.51**	228.15	0.80	0.58
36% steps	4.12 T	2.76×	3.94	5.16	216.29	0.78	0.58
FORA ($\mathcal{N}$=3) [41]	3.91 T	2.16×	3.28	5.72	228.47	0.80	0.56
ToCa ($\mathcal{N}$=3, R=93%) [57]	5.12 T	1.40×	3.04	4.74	-	0.80	0.57
ToCa ($\mathcal{N}$=4, R=93%) [57]	4.37 T	1.49×	3.60	5.11	-	0.79	0.56
Ours ($\mathcal{N}$=3)	**3.91** T	2.15×	**2.90**	4.60	**231.54**	**0.80**	**0.58**

Table 2: Qualitative results for text-to-image generation with PixArt-α and DPM++ (20 steps, 256×256) on COCO2014 30K validation prompts. Our method significantly reduces FLOPs compared to the baseline while slightly improving image quality and outperforms other approaches with similar FLOPs in terms of FID.

Method	FLOPs↓	FID↓	CLIP↑
PixArt-α (20 steps, baseline)	11.43 T	27.76	15.90
40% steps	4.57 T	28.50	16.18
FORA ($\mathcal{N}$=2) [41]	5.75 T	26.17	16.03
FORA ($\mathcal{N}$=3) [41]	4.04 T	27.03	16.07
ToCa ($\mathcal{N}$=3, R=60%) [57]	6.33 T	24.17	16.18
ToCa ($\mathcal{N}$=3, R=90%) [57]	4.26 T	26.12	16.13
Ours ($\mathcal{N}$=3)	4.04 T	24.20	16.13

Table 3: Text-to-image generation performance of SD3-medium and Sana-0.6B with Flow-Euler sampler (20 steps, 256×256) on COCO2014 30K validation prompts. Our method outperforms the naive caching method in image quality, while incurring only a marginal increase in FLOPs. The second column (Mod) indicates the percentage of layers with modulation applied.

Method	Mod	FLOPs↓	FID↓
SD3 (baseline) [10]	-	11.08 T	23.58
FORA ($\mathcal{N}$=3) [41]	0.00%	3.92 T	47.27
Ours ($\mathcal{N}$=3)	1.50%	3.92 T + 11.01 M	33.60
Sana (baseline) [52]	-	4.45 T	20.40
FORA ($\mathcal{N}$=3) [41]	0.00%	1.62 T	23.19
Ours ($\mathcal{N}$=3)	1.83%	1.62 T + 11.80 M	22.25

Table 4: Modulation across varying steps. Modulation improves generation quality, especially at fewer sampling steps, with a minimal increase in FLOPs. The second column (Mod) shows the percentage of layers with modulation applied.

Steps	Mod	FLOPs↓	FID↓
20	0.00%	3.22 T	14.16
	0.55%	3.22 T + 9.44 M	11.33 (-2.83)
50	0.00%	7.81 T	3.28
	0.38%	7.81 T + 16.52 M	2.90 (-0.38)
100	0.00%	15.63 T	2.44
	0.35%	15.63 T + 30.67 M	2.38 (-0.06)

Table 5: Large activation cycles. The proposed method achieves significantly better image generation performance than DDIM with 12% steps under comparable FLOPs. Furthermore, our framework requires substantially less computation than DDIM with 18% steps in order to attain similar image quality.

Method	FLOPs↓	FID↓
DiT-XL/2 (50 steps)	22.89 T	2.29
18% steps	4.12 T	16.48
12% steps	2.74 T	52.17
Ours ($\mathcal{N}$=10)	2.33 T	16.62

4.2 Main Results

Class-conditional image generation. Quantitative results of our framework for class-conditional image generation using DiT-XL/2 [35] are in Table 1. In comparison to the baseline DDIM [44] 50 steps, our method requires $2.93\times$ fewer FLOPs and $2.15\times$ less inference latency to obtain similar generation quality. Against DDIM 36% steps, which operates with similar FLOPs, our method demonstrates substantially higher generation quality. For example, our approach achieves a superior IS of +15.25. Our framework also outperforms existing caching-based methods, FORA ($\mathcal{N}$=3) [41] and ToCa ($\mathcal{N}$=3, R=93%[3]) [57], which have similar FLOPs, across all image quality metrics.

Text-to-image generation. We show the performance of our method using PixArt-α [3] with DPM++ [26] 20 steps in Table 2. To achieve comparable generation performance to our method, DPM++ 20 steps, ToCa ($\mathcal{N}$=3, R=60%) and FORA ($\mathcal{N}$=2) require $2.83\times$, $1.57\times$ and $1.42\times$ more computation, respectively. Our inference speed-up over the baseline is $1.50\times$. Also, the proposed method outperforms DPM++ 40% steps and existing caching-based approaches, FORA ($\mathcal{N}$=3) and ToCa ($\mathcal{N}$=3, R=90%) in terms of FID, while maintaining similar FLOPs.

We also evaluate our method using SD3-medium [10] and Sana-0.6B [52] with Flow-Euler [22] 20-step sampler. Table 3 shows that the proposed method with $\mathcal{N}$=3 consistently yields better generation quality than the naive caching method (FORA), with negligible additional FLOPs.

4.3 Ablation Studies

We conduct ablation studies on class-conditional 50K image generation using DiT-XL/2 [35] with DDIM [44] sampler on ImageNet-1K classes [7].

[3]For ToCa, "$\mathcal{N}$=3, R=93%" means that the activation cycle is 3, performing full inference once every 3 timesteps, and for the other timesteps, on average, R=93% of tokens reuse cached values, while (1-R)=7% of tokens require network inference.

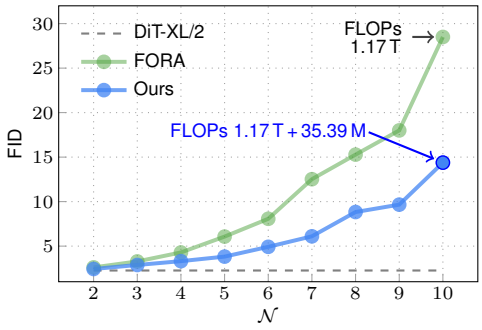


Figure 3: Modulation across activation cycles ($\mathcal{N}$). Modulation consistently enhances image generation quality, with larger $\mathcal{N}$ leading to greater improvements.

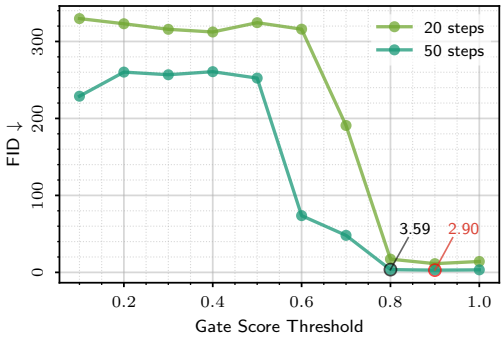


Figure 4: Relationship between thresholds and FID. A 0.9 threshold, where the majority of cached features are used without modification and a small portion is modulated, is optimal.

Effectiveness of modulation. A key component of our framework lies in the use of modulation. To test its effectiveness, we compare configurations with (Ours) and without the modulation (FORA [41]). We first evaluated different number of activation cycles $\mathcal{N} \in \{2, 3, \ldots, 10\}$ with 50 DDIM steps. Figure 3 demonstrates that modulation enhances generation quality, with larger activation cycles $\mathcal{N}$ leading to more noticeable gains. Furthermore, we investigate the effect of varying the number of sampling steps with activation cycle $\mathcal{N}$=3. Table 4 shows that the use of modulation consistently improves the generation performance, with a greater improvement observed when fewer steps are used. These results highlight the effectiveness of modulation, as models with larger activation cycles and fewer steps tend to have lower feature similarity, making modulation-based adjustments more significant. Moreover, since the modulator is applied under 1% of the layers, it results in negligible computational overhead.

Choice of threshold. Recall that if the modulation gate score (β) exceeds a predefined threshold (c), the modulation is applied to the cached feature; otherwise, the cached feature itself is used. In Figure 4, we examine the relationship between thresholds and generation quality using DiT-XL/2 with 20 and 50 DDIM steps and the activation cycle $\mathcal{N}$=3. A 0.9 threshold, where most cached features are utilized without modification and only a few features are modulated (0.55% for the DDIM 20 steps and 0.38% for the DDIM 50 steps in Table 4), marks the sweet spot for optimal performance. We see that when the threshold is larger (utilizing more cache without modulation), FID tends to be better compared to smaller thresholds (applying modulation more). We conjecture that the modulation is only needed for a few features, since the similarity between features at the same depth across steps is very high (Figure 1). Also, the modulation involves simple scale and translation operations, so its limited representation power may hinder generative performance if applied too extensively.

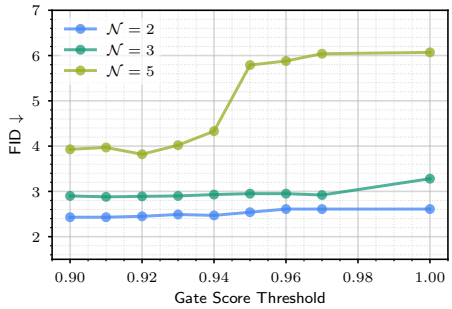


Figure 5: Various activation cycles ($\mathcal{N}$) and modulation thresholds (c). The optimal threshold appears around 0.9 for all activation cycles. Modulation becomes more effective with larger $\mathcal{N}$ compared to the caching-only case (c=1.00).

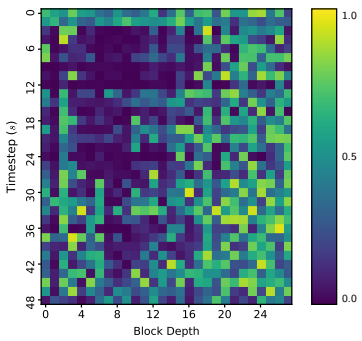


Figure 6: Learned pattern of modulation gate scores for attention. The modulation gate scores are higher in later denoising steps and deeper layers, which tend to need more modulation.

Table 6: High-resolution generation performance of Sana-0.6B with Flow-Euler sampler (20 steps, 512×512 and 1024×1024) on COCO2014 5K validation. Our method shows better quality than the naive caching method with negligible additional FLOPs. The second column (Mod) is the percentage of layers with modulation applied.

Method	Mod	FLOPs↓	FID↓
Sana-512 [52]	-	12.19 T	31.21
FORA ($\mathcal{N}$=3) [41]	0.00%	4.33 T	29.66
Ours ($\mathcal{N}$=3)	1.10%	4.33 T + 28.31 M	29.16
Sana-1024 [52]	-	43.11 T	29.91
FORA ($\mathcal{N}$=3) [41]	0.00%	15.16 T	29.45
Ours ($\mathcal{N}$=3)	0.55%	15.16 T + 66.06 M	29.17

Table 7: Low-step generation performance of Sana-0.6B with Flow-Euler sampler (8 steps, 512×512) on COCO2014 5K validation prompts. Under similar FLOPs settings, our framework outperforms both fewer-step and caching-based approaches in terms of image quality.

Method	FLOPs↓	FID↓
Sana-512 [52]	4.88 T	31.08
62.5% steps	3.07 T	34.45
50% steps	2.44 T	48.87
L2C ($\mathcal{N}$=2) [30]	2.49 T	32.37
FORA ($\mathcal{N}$=2) [41]	2.46 T	32.46
Ours ($\mathcal{N}$=2)	2.46 T	31.89

Different number of activation cycles. We investigate the relationship between different activation cycles ($\mathcal{N}$) and gate thresholds (c) using DiT-XL/2 with DDIM 50-step sampler. When examining performance across thresholds in coarse increments of 0.1, the best performance is observed at 0.9 (Figure 4). Therefore, we now explore the range between 0.9 and 1.0 in a more fine-grained way. Figure 5 shows that the optimal threshold appears around 0.9 for all $\mathcal{N} \in \{2, 3, 5\}$. Similarly, as shown in Table 4, the modulation becomes more effective as the feature gap between timesteps increases (i.e., $\mathcal{N} \uparrow$). For example, with $\mathcal{N}$=5, FID decreases by more than 2 compared to the caching-only case (c=1.00), whereas with $\mathcal{N}$=2, the improvement is less than 1.

4.4 More Analysis

Large activation cycles. We verify the effectiveness of our framework under a large activation cycle $\mathcal{N}$ setting. We use DiT-XL/2 [35] with DDIM [44] 50-step sampler and set the modulation gate threshold (c) to 0.93. Table 5 shows that DDIM with 18% steps achieves comparable image quality to our method with $\mathcal{N}$=10, but requires 1.76× more FLOPs. DDIM at 12% steps matches the FLOPs of our method with $\mathcal{N}$=10, but yields significantly worse FID scores (+35.55). **High-resolution performance.** To evaluate whether the proposed modulation remains effective for high-resolution text-to-image generation, we apply it to Sana-0.6B [52] models generating 512×512 and 1024×1024 images on the COCO2014 5K validation set [21]. Table 6 shows that applying the modulation less than 2% of the layers effectively improves generation quality with negligible computational overhead. This agrees with the results in Table 3.

Low-step performance. We now investigate the effectiveness of our method in a low-step setting using Sana-0.6B (512×512) with 8-step Flow-Euler sampler [22]. Table 7 shows that our framework

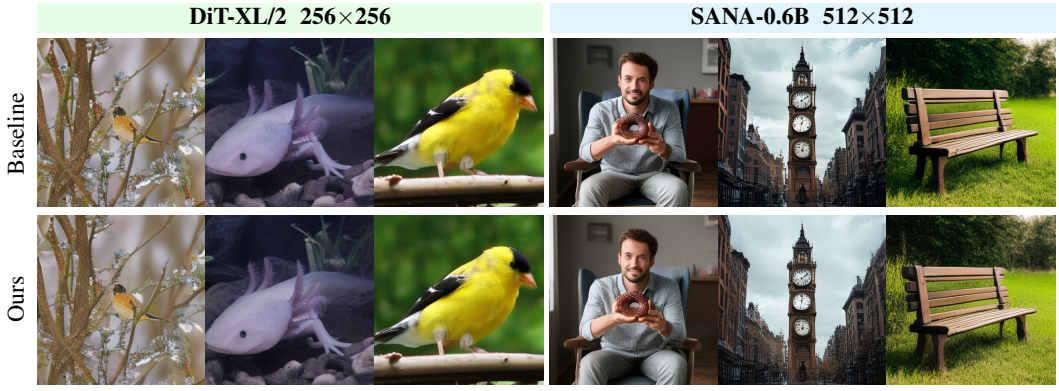


Figure 7: Qualitative results. The first three images in the top row are generated by DiT-XL/2 (ImageNet, 256×256, 50-step DDIM) and the others by Sana-0.6B (COCO2014, 512×512, 20-step Flow-Euler), while the bottom shows results from our method ($\mathcal{N} = 3$). Generation quality is almost identical, despite our method reduces FLOPs by 2.83× for DiT-XL/2 and 2.82× for Sana-0.6B.

outperforms fewer-step and other caching-based methods under comparable FLOPs. **Learned pattern of the modulation gate.** We plot learned modulation gate scores β for attention features using DiT-XL/2 (DDIM 50-step, 256×256) under $\mathcal{N}$=3. Figure 6 indicates that later denoising stage features and deeper layer features tend to need more modulation (i.e., larger scores).

Qualitative analysis. We qualitatively compare the generation performance of our method and baseline methods in Figure 7. The first three images are generated with DiT-XL/2 for ImageNet [7] classes at 256×256 resolution, and the other three with Sana-0.6B for COCO2014 validation prompts at 512×512 resolution. The top row shows baseline outputs: 50-step DDIM for DiT-XL/2 and 20-step Flow-Euler for Sana-0.6B. The bottom row shows our results with activation cycle $\mathcal{N} = 3$. The qualitative generation performance is nearly identical, while our method reduces FLOPs by $2.83\times$ for DiT-XL/2 (Table 1) and $2.82\times$ for Sana (Table 6). Additional results are in the appendix.

5 Limitations and Conclusion

Limitations. The core idea of the proposed method is to selectively adapt cached features through learned linear modulation for efficient diffusion transformer inference. We find applying modulation to all features leads to image quality degradation. We conjecture that the modulation is intentionally lightweight —using real-valued scalars—which limits its representational capacity.

Conclusion. We have introduced a cache-based framework for diffusion transformers that enhances inference efficiency by reusing features and selectively adapting them through lightweight linear modulation. Experimental results demonstrate our method generally achieves similar generation quality to that of the original sampler with significantly less computation and outperforms existing approaches under similar FLOPs settings. To our knowledge, this is the first method to improve inference efficiency by approximating diffusion transformer inference through lightweight cache transformations. We hope that our work inspires further exploration of advanced caching methods.

References

[1] Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A ViT backbone for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[2] Ziang Cao, Fangzhou Hong, Tong Wu, Liang Pan, and Ziwei Liu. Large-vocabulary 3D diffusion model with transformer. In *International Conference on Learning Representations (ICLR)*, 2024.

[3] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024.

[4] Lei Chen, Yuan Meng, Chen Tang, Xinzhu Ma, Jingyan Jiang, Xin Wang, Zhi Wang, and Wenwu Zhu. Q-DiT: Accurate post-training quantization for diffusion transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[5] Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Δ-Dit: A training-free acceleration method tailored for diffusion transformers. *arXiv*, abs/2406.01125, 2024. URL `https://arxiv.org/abs/2406.01125`.

[6] Shoufa Chen, Mengmeng Xu, Jiawei Ren, Yuren Cong, Sen He, Yanping Xie, Animesh Sinha, Ping Luo, Tao Xiang, and Juan-Manuel Perez-Rua. GenTron: Diffusion transformers for image and video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[8] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[9] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[10] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning (ICML)*, 2024.

[11] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[12] Yefei He, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. EfficientDM: Efficient quantization-aware fine-tuning of low-bit diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

[13] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7514–7528, 2021.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[15] Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. BK-SDM: Architecturally compressed stable diffusion for efficient text-to-image generation. In *ICML Workshop on Efficient Systems for Foundation Models*, 2023.

[16] Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ODE trajectory of diffusion. In *International Conference on Learning Representations (ICLR)*, 2024.

[17] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[18] Senmao Li, Taihang Hu, Joost van de Weijer, Fahad Khan, Tao Liu, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of the encoder for diffusion model inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[19] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[20] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. SnapFusion: Text-to-image diffusion model on mobile devices within two seconds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[22] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[23] Haozhe Liu, Wentian Zhang, Jinheng Xie, Francesco Faccio, Mengmeng Xu, Tao Xiang, Mike Zheng Shou, Juan-Manuel Perez-Rua, and Jürgen Schmidhuber. Faster diffusion through temporal attention decomposition. *Transactions on Machine Learning Research (TMLR)*, 2025. URL `https://openreview.net/forum?id=xXs2GKXPnH`.

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[25] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[26] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv*, abs/2211.01095, 2023. URL https://arxiv.org/abs/2211.01095.

[27] Haoyu Lu, Guoxing Yang, Nanyi Fei, Yuqi Huo, Zhiwu Lu, Ping Luo, and Mingyu Ding. VDT: General-purpose video diffusion transformers via mask modeling. In *International Conference on Learning Representations (ICLR)*, 2024.

[28] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv*, abs/2310.04378, 2023. URL https://arxiv.org/abs/2310.04378.

[29] Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. FasterCache: Training-free video diffusion model acceleration with high quality. In *International Conference on Learning Representations (ICLR)*, 2025.

[30] Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-Cache: Accelerating diffusion transformer via layer caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[31] Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[32] Shentong Mo, Enze Xie, Ruihang Chu, Lanqing Hong, Matthias Nießner, and Zhenguo Li. DiT-3D: Exploring plain diffusion transformers for 3d shape generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[33] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W. Battaglia. Generating images with sparse representations. In *International Conference on Machine Learning (ICML)*, 2021.

[34] Zizheng Pan, Bohan Zhuang, De-An Huang, Weili Nie, Zhiding Yu, Chaowei Xiao, Jianfei Cai, and Anima Anandkumar. T-Stitch: Accelerating sampling in pre-trained diffusion models with trajectory stitching. In *International Conference on Learning Representations (ICLR)*, 2025.

[35] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[36] Yiran Qin, Zhelun Shi, Jiwen Yu, Xijun Wang, Enshen Zhou, Lijun Li, Zhenfei Yin, Xihui Liu, Lu Sheng, Jing Shao, Lei Bai, Wanli Ouyang, and Ruimao Zhang. WorldSimBench: Towards video generation models as world simulators. *arXiv*, abs/2410.18072, 2024. URL https://arxiv.org/abs/2410.18072.

[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

[39] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations (ICLR)*, 2022.

[40] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[41] Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. FORA: Fast-forward caching in diffusion transformer acceleration. *arXiv*, 2407.01425, 2024. URL https://arxiv.org/abs/2407.01425.

[42] Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Yanyu Li, Yifan Gong, Kai Zhang, Hao Tan, Jason Kuen, Henghui Ding, Zhihao Shu, Wei Niu, Pu Zhao, Yanzhi Wang, and Jiuxiang Gu. LazyDiT: Lazy learning for the acceleration of diffusion transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[43] Junhyuk So, Jungwon Lee, and Eunhyeok Park. FRDiff: Feature reuse for universal training-free acceleration of diffusion models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, page 328–344, 2024.

[44] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*, 2021.

[45] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[46] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[47] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning (ICML)*, 2023.

[48] Yang Sui, Yanyu Li, Anil Kag, Yerlan Idelbayev, Junli Cao, Ju Hu, Dhritiman Sagar, Bo Yuan, Sergey Tulyakov, and Jian Ren. BitsFusion: 1.99 bits weight quantization of diffusion model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[50] Fu-Yun Wang, Zhaoyang Huang, Alexander William Bergman, Dazhong Shen, Peng Gao, Michael Lingelbach, Keqiang Sun, Weikang Bian, Guanglu Song, Yu Liu, Xiaogang Wang, and Hongsheng Li. Phased consistency models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[51] Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[52] Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, and Song Han. SANA: Efficient high-resolution text-to-image synthesis with linear diffusion transformers. In *International Conference on Learning Representations (ICLR)*, 2025.

[53] Shuai Yang, Yukang Chen, Luozhou Wang, Shu Liu, and Ying-Cong Chen. Denoising diffusion step-aware models. In *International Conference on Learning Representations (ICLR)*, 2024.

[54] Xiulong Yang, Sheng-Min Shih, Yinlin Fu, Xiaoting Zhao, and Shihao Ji. Your ViT is secretly a hybrid discriminative-generative diffusion model. *arXiv*, 2022. URL `https://arxiv.org/abs/2208.07791`.

[55] Ligeng Zhu. thop: Pytorch-opcounter, 2018. URL `https://github.com/Lyken17/pytorch-OpCounter.git`.

[56] Chang Zou, Evelyn Zhang, Runlin Guo, Haohang Xu, Conghui He, Xuming Hu, and Linfeng Zhang. Accelerating diffusion transformers with dual feature caching. *arXiv*, abs/2412.18911, 2024. URL `https://arxiv.org/abs/2412.18911`.

[57] Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In *International Conference on Learning Representations (ICLR)*, 2025.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: Yes, the claims are well supported by the experimental results.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Yes, please refer to "Limitations" section.

Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [NA]

   Justification: NA, the paper does not include theoretical results.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Yes, pseudocode for both training and sampling is provided to support reproducibility.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

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

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

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

   Answer: [No]

   Justification: No, the code is not open-sourced, but the paper includes pseudocode, model links, and implementation details in the supplementary material to support accessibility.

   Guidelines:

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

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Yes, detailed experimental settings are included in the supplementary material.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Yes, error bars with random seeds are reported in the supplementary material for selected experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Yes, the supplementary material provides details such as training iterations and the type of GPU used.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Yes, the work adheres to the NeurIPS Code of Ethics in all respects.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [No]

    Justification: No, This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

Answer: [No]

Justification: No, safeguards regarding potential misuse are not discussed in the paper.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: Yes, all external assets used in the paper are properly credited, and their licenses and terms of use are respected.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [NA]

Justification: NA, the paper does not release new assets.

Guidelines:

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

14. **Crowdsourcing and research with human subjects**

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

Answer: [NA]

Justification: NA, the paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

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

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

Answer: [NA]

Justification: NA, the paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: NA, the core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Implementation Details

A.1 Training Details

Models. In Table 8, we provide a summary that lists checkpoint URLs and license information for each model-resolution pair used in our experiments. For simplicity, we denote image resolutions of 256×256, 512×512, and 1024×1024 as 256, 512, and 1024, respectively.

Training settings. The optimization is performed only on the modulation gate and modulator. We use the training set of ImageNet [7] for DiT-XL/2 [35] and COCO2014 [21] for text-to-image generation models. We train the modulation gate scores and modulator parameters on 4 H100 GPUs with AdamW optimizer [24] and learning rate 10^{-3} for 200K iterations. The global batch size for 256×256 experiments is set for 64, and the corresponding value for 512×512 and 1024×1024 experiments is 8.

Table 8: Checkpoint URLs and license information for model-resolution pairs used in our experiments.

Model	Resolution	Checkpoint URL	License
DiT-XL/2 [35]	256	`https://dl.fbaipublicfiles.com/DiT/models/DiT-XL-2-256x256.pt`	CC-BY-NC
PixArt-α [3]	256	`https://huggingface.co/PixArt-alpha/PixArt-alpha/blob/main/PixArt-XL-2-256x256.pth`	GNU AGPLv3
	1024	`https://huggingface.co/PixArt-alpha/PixArt-alpha/blob/main/PixArt-XL-2-1024-MS.pth`	GNU AGPLv3
SD3-medium [10]	256	`https://huggingface.co/stabilityai/stable-diffusion-3-medium/blob/main/sd3_medium_incl_clips.safetensors`	CreativeML Open RAIL++-M
Sana-0.6B [52]	256, 512	`https://huggingface.co/Efficient-Large-Model/Sana_600M_512px/blob/main/checkpoints/Sana_600M_512px_MultiLing.pth`	NSCL v2-custom
	1024	`https://huggingface.co/Efficient-Large-Model/Sana_600M_1024px/blob/main/checkpoints/Sana_600M_1024px_MultiLing.pth`	NSCL v2-custom

A.2 Evaluation Details

We describe the methods used for measuring FLOPs and latency, as well as the selection of the COCO2014 validation set.

FLOPs. FLOPs were calculated by multiplying the MACs (Multiply-Accumulates) measured with `pytorch-OpCounter` [55] by 2.

Latency. We evaluated the latency of generating a batch of 8 images using classifier-free guidance by conducting 10 test runs and computing the average execution time. To ensure accurate measurement, we performed 3 warm-up iterations before recording the execution time for the subsequent 10 runs. The speed metric is defined as the ratio of the baseline model's latency to the latency of each method. The hardware setup consists of a single NVIDIA H100 GPU (80GB HBM3) and an Intel Xeon Gold 6442Y CPU. The evaluation was performed using CUDA 12.1, Python 3.10, and PyTorch 2.2.2.

COCO validation set. COCO2014 [21] dataset contains sufficient captions in both train and validation sets (40K each), so we used COCO2014 instead of COCO2017. For validation, we randomly select 5K/30K captions from COCO2014 validation set, following the methodology outlined in the prior work [3]. Some methods such as ToCa [57] provide test results on a custom 30K COCO validation set, where some prompts are extracted from the COCO2014 trainset. Since we used COCO2014 trainset for training our framework, we evaluated all methods on COCO2014 validation set for a fair comparison. We uploaded the list files containing 5K/30K validation prompts to an anonymous Github page, and the URLs are provided in Table 9.

Table 9: COCO2014 5K/30K validation sets.

Set	# Prompts	URL
COCO-5K	5,000	`https://github.com/dod2025/evallist/blob/main/coco_captions_val2014_5k.txt`
COCO-30K	30,000	`https://github.com/dod2025/evallist/blob/main/coco_captions_val2014_30k.txt`

A.3 Algorithms for Flow Models

We now provide the training and sampling algorithms of the proposed framework for flow-based models [22].

Flow models. We define x_t as a linear interpolation between a data point from the data distribution $x_0 \sim q(x_0)$ and a noise point $x_1 \sim N(0, I)$. The velocity v_t is the direction from the noise to the data point:

$$x_t = (1-t)x_1 + tx_0 \quad \text{and} \quad v_t = x_0 - x_1\,. \tag{8}$$

Flow models learn a neural network to predict the conditional expectation $\bar{v}_t = E[v_t \mid x_t]$. To sample from a flow model, a random noise x_1 is iteratively updated to x_0 using a denoising ODE with the learned flow model $\bar{v}_\theta(x_t, t)$. This iterative step is often approximated using Euler sampling over small discrete time intervals.

Training and sampling. Algorithms 1 and 2 apply the proposed method to diffusion models with DDIM [44] and DPM++ [26] samplers. The corresponding versions adapted for flow-based models are presented in Algorithms 3 and 4. While the samplers differ, the core idea, caching and selective modulation, is the same. In the algorithms, we denote by $\tilde{v}_\theta$ the approximation of the original flow model ($\bar{v}_\theta$) obtained from our method.

Algorithm 3 Training

Input: Data distribution $q(\cdot)$, flow model $\bar{v}_\theta(\cdot)$, optimizer, ODE solver $\Psi(\cdot)$, total steps T, the step schedule $\{t_s\}_{s=0}^{T-1}$ in $\Psi(\cdot)$ and activation cycle $\mathcal{N}$
Randomly initialize β, α, and γ
Compute $r = (T-1)\%\mathcal{N}$
repeat
  $x_0 \sim q(x_0)$ and $x_{T-1} \sim N(0, I)$.
  $s \leftarrow \mathcal{N} * n + r,\ n \sim \mathcal{U}[0, (T-1)//\mathcal{N} - 1]$
    # Step t_s for calculating states for caching
  $x_s = (1-t_s)x_{T-1} + t_s x_0$
  $v_s \leftarrow \bar{v}_\theta(x_s, t_s)$ and cache $h_i^l(\cdot, t_s)$'s in Eq (2)
  $d \sim \mathcal{U}[1, \mathcal{N} - 1]$
  $m \leftarrow s - d$
    # Step t_m for using cached states
  $\tilde{x}_m \leftarrow \Psi(v_s, t_s, t_m)$
  $x_{m+1} = (1-t_{m+1})x_{T-1} + t_{m+1}x_0$
  $v_{m+1} \leftarrow \bar{v}_\theta(x_{m+1}, t_{m+1})$
  $x_m \leftarrow \Psi(v_{m+1}, t_{m+1}, t_m)$
  $\beta \leftarrow \text{Sigmoid}(\beta)$
    # Optimize
  Calculate $\tilde{v}_\theta(\tilde{x}_m, t_m; \beta, \alpha, \gamma)$ by Eq (4) with Eq (5)
  $\mathcal{L} \leftarrow \|\tilde{v}_\theta(\tilde{x}_m, t_m) - \bar{v}_\theta(x_m, t_m)\|_2^2 + \lambda \sum \beta$
  Compute gradients for β, α, γ with respect to loss $\mathcal{L}$ and update them with optimizer
until converged

Algorithm 4 Sampling

Input: Flow model $\bar{v}_\theta(\cdot)$, modulation gate score β, modulation parameters α, γ, ODE solver $\Psi(\cdot)$, total steps T, the step schedule $\{t_s\}_{s=0}^{T-1}$ in $\Psi(\cdot)$, activation cycle $\mathcal{N}$ and threshold c
Compute $r = (T-1)\%\mathcal{N}$
$x_{T-1} \sim N(0, I)$
for $s = T-1$ **to** 0 **do**
  for l in $\{\text{SA}, \text{CA}, \text{MLP}\}$ and $i = 1$ **to** L **do**
    if $s\%\mathcal{N} = r$ **then**
      Calculate $f_i^l(\cdot, t_s)$ and cache $h_i^l(\cdot, t_s)$'s in Eq (2)
        # Inference and cache features
    else
      $\beta_s^{l,i} \leftarrow \text{Sigmoid}(\beta_s^{l,i})$
      $\beta_s^{l,i} \leftarrow 1_{\{\beta_s^{l,i} > c\}}$
      Calculate $\tilde{f}_i^l(\cdot, t_s)$ by Eq (5)
    end if
  end for
  if $s\%\mathcal{N} = r$ **then**
    $v_s \leftarrow \bar{v}_\theta(x_s, t_s)$
      # Original inference
  else
    $v_s \leftarrow \tilde{v}_\theta(\tilde{x}_s, t_s; \beta, \alpha, \gamma)$
      # Use cached features
  end if
  $x_{s-1} \leftarrow \Psi(v_s, t_s, t_{s-1})$
end for
return x_0

B Additional Analysis

B.1 Effect of Random Seed

Table 1 presents quantitative results for class-conditional 50K image generation on ImageNet-1K [7] using DiT-XL/2 [35] with DDIM [44] (50 steps). To assess statistical significance, we report mean $\pm$ standard deviation over multiple runs with different random seeds $\{2, 55, 777\}$ for our method, and the results are compared with those of the naive caching baseline (FORA). Table 10 shows that the proposed framework consistently achieves higher generation performance than the naive caching baseline across most metrics.

Table 10: Qualitative performance for class-conditional 50K image generation with DiT-XL/2 and DDIM 50 steps on ImageNet 1K classes. Our results, reported as mean $\pm$ standard deviation over multiple random seeds, show that the proposed framework consistently achieves higher generation performance than the naive caching baseline across most metrics.

Method	FID ↓	sFID ↓	IS ↑	Precision ↑	Recall ↑
FORA ($\mathcal{N}$=3) [41]	3.28	5.72	228.47	0.80	0.56
Ours ($\mathcal{N}$=3)	2.90±0.0677	4.60±0.093	231.54±1.4466	0.80±0.0039	0.58±0.0045

B.2 Effectiveness of Modulation

We now compare the performance of the proposed modulator (threshold 0.9) against the cache-only baseline, using PixArt-α [3] and 20-step DPM++ [26] sampler with activation cycles $\mathcal{N} \in \{2, 3, 4\}$. This comparison extends the experiment shown in Figure 3 to a different architecture. Table 11 shows that the use of modulation consistently improves the generation performance, with a more pronounced effect observed in the case of larger $\mathcal{N}$. This highlights the effectiveness of modulation, since the model with larger $\mathcal{N}$ has a greater time spacing between features for the same sampling steps. Consequently, feature similarity decreases, making adjustments through the modulation more important. Moreover, since the modulator is applied to fewer layers ($\leq$2.02%), this results in a minimal increase in computation.

Table 11: Effectiveness of modulation. Modulation improves generation quality, especially at larger activation cycles $\mathcal{N}$, with a minimal increase in FLOPs. The second column (Mod) shows the percentage of layers with modulation applied.

$\mathcal{N}$	Mod	FLOPs ↓	FID ↓
2	0.00%	5.75T	26.17
	2.02%	5.75T+40.11M	24.03
3	0.00%	4.04T	27.03
	1.28%	4.04T+33.03M	24.20
4	0.00%	2.90T	31.72
	0.79%	2.90T+23.59M	25.63

B.3 Various Activation Cycles

We explore the relationship between different activation cycles ($\mathcal{N}$) and the modulation gate threshold (c) with DPM++ [26] 20-step PixArt-α [3]. This comparison extends the experiment shown in Figure 5 to a different architecture. Figure 8 shows that the optimal threshold appears around 0.9 across all combinations similar to the results observed in Figure 5. Also, the modulation is more effective as the feature gap between timesteps increases (i.e., with larger $\mathcal{N}$). For instance, with $\mathcal{N}$=4, FID decreases by more than 6 compared to caching-only baseline, whereas with $\mathcal{N}$=2, the improvement is around 2.

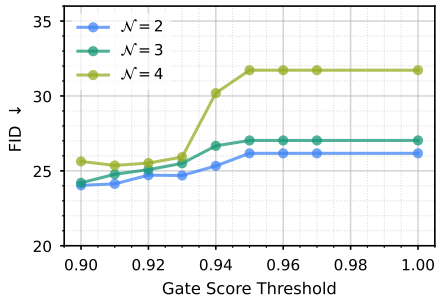


Figure 8: Various activation cycles ($\mathcal{N}$) and gate score thresholds. The optimal threshold appears around 0.9 for all activation cycles. The proposed modulation becomes more effective with larger $\mathcal{N}$ compared to naive caching (threshold=1.00).

B.4 Robustness Across Different Sampling Methods

We assess the robustness of the default threshold under various sampling methods using PixArt-α [3]. Recall that we utilize the default sampling method DPM++ [26] for PixArt-α (256×256) and the threshold for modulation gate is set to 0.9. To demonstrate the robustness, we also apply DDIM [44] sampler to PixArt-α. We use the 20-step model with three activation cycles ($\mathcal{N}$) and report results on COCO2014 [21] 30K validation set. Table 12 shows that with the threshold fixed, modulation consistently improves FID performance, and the improvement becomes more pronounced as $\mathcal{N}$ increases. This trend is observed in both DPM++ and DDIM.

Table 12: Robustness of the default threshold across different sampling methods. With the threshold fixed, modulation consistently improves FID performance across different samplers, with larger gains observed for larger activation cycles ($\mathcal{N}$).

$\mathcal{N}$	Mod	FLOPs↓	FID (DPM++)↓	FID (DDIM)↓
2	0.00%	5.75T	26.17	28.30
	2.02%	5.75T+40.11M	24.03	27.13
3	0.00%	4.04T	27.03	29.37
	1.28%	4.04T+33.03M	24.20	27.37
4	0.00%	2.90T	31.72	34.65
	0.79%	2.90T+23.59M	25.63	25.48

B.5 Nonlinear Modulation Functions

We further explore lightweight nonlinear modulators based on common activation functions such as SiLU [9], sigmoid, hyperbolic tangent (tanh) and softplus. Details of the modulator formulations are found in Table 13, where x denotes the cached feature and a and b are trainable scalar parameters. We trained a and b under similar setups to the proposed linear modulator (Appendix A). For evaluation, we used the configuration from Table 1: DiT-XL/2 [35] with a 50-step DDIM sampler [44] and an activation cycle $\mathcal{N} = 3$, comparing performance of linear and nonlinear modulators. Table 13 reports 50K image generation performance on ImageNet-1K [7] classes. All linear and nonlinear modulators outperformed the naive caching baseline FORA [41], which applies no modulation. However, we observed some variation across different nonlinear function types. Among them, the SiLU-based modulator achieved performance comparable to the proposed linear variant. These results indicate that the effectiveness of nonlinear modulation depends on the choice of function, supporting our use of a simple yet effective linear design with lower computational overhead. Although we experimented with diverse combinations (e.g., initialization, learning rate, threshold, and training iteration) for each nonlinear modulator and reported the best result, further hyperparameter tuning could potentially yield better performance.

Table 13: Results of class-conditional 50K image generation on ImageNet-1K classes using DiT-XL/2 with 50 steps using different nonlinear modulation functions. Table (a) shows the pseudo code of modulation functions. Table (b) demonstrates that all modulators outperformed the naive caching baseline, with SiLU performing on par with the linear variant, supporting the use of our linear design with lower computational overhead.

(a) Formulations of modulation functions.

Method	Pseudo Code
Linear	`a * x + b`
SiLU [9]	`a * silu(x) + b`
SoftGate	`sigmoid(a * x + b) * x`
Tanh	`a * tanh(x) + b`
Softplus & Log	`log(clamp((softplus(a) + 1e-6) * x, min=1e-6, max=1e4)) + b`

(b) Results of class-conditional generation.

Method	FID ↓	sFID ↓	IS ↑	Precision ↑	Recall ↑
DiT-XL/2 (50 steps, baseline)	2.29	4.32	239.54	0.81	0.60
FORA (no modulators) [41]	3.28	5.72	228.47	0.80	0.56
Ours (linear)	2.90	4.60	231.54	0.80	0.58
SiLU [9]	2.99	4.74	230.22	0.79	0.58
SoftGate	3.00	4.78	231.93	0.80	0.57
Tanh	3.13	4.66	226.21	0.79	0.58
Softplus & Log	3.25	5.73	229.53	0.80	0.56

B.6 Validation under Low-Precision Settings

We validate the effectiveness of our framework under a low-precision setting (8-bit weights and 16-bit activations), using DiT-XL/2 [35] with 50-step DDIM sampler [44], activation cycle $\mathcal{N} = 3$ and modulation gate threshold of 0.90. To obtain FP8 weights, we apply basic channel-wise symmetric quantization. Following the setup in Table 1, we evaluate performance on the 50K ImageNet [7] 1K-class generation task. As shown in Table 14, compared to DDIM 50%-steps with similar generation quality, our method requires $1.46\times$ fewer FLOPs. Moreover, compared to FORA [41] and DDIM with 36% of the steps (which incur similar FLOPs), our method achieves better image quality. These results demonstrate that our method remains effective even under low-precision settings, not just in full-precision (32-bit) scenarios.

Table 14: Validation under low-precision settings (FP8 weights and FP16 activations) for class-conditional 50K image generation with DiT-XL/2 and 50-step DDIM sampler on ImageNet-1K classes. Our method reduces FLOPs compared to fewer-step sampler at similar quality, achieves better image quality than naive caching and fewer-step samplers under similar FLOPs.

Method	FLOPs ↓	FID ↓	sFID ↓	IS ↑	Precision ↑	Recall ↑
DiT-XL/2 (50 steps, baseline)	11.44 T	2.46	4.70	237.76	0.80	0.60
50% steps	5.72 T	3.18	5.30	231.12	0.79	0.58
36% steps	4.12 T	4.37	6.43	214.37	0.77	0.56
FORA ($\mathcal{N}$=3) [41]	3.91 T	3.66	7.09	224.34	0.79	0.55
Ours ($\mathcal{N}$=3)	3.91 T	3.22	5.13	224.87	0.79	0.57

B.7 Feature Similarity Analysis under Consistency Models

To explore the potential of combining our method with consistency models, we analyze feature similarity in PixArt-α [3] with LCM sampler [28]. Specifically, we compute cosine similarity between feature pairs sampled at a timestep interval of $\mathcal{N} = 2$ for 4-step generation, and at intervals of $\mathcal{N} = 2$ and $\mathcal{N} = 3$ for 8-step generation. The analysis is conducted on self-attention, cross-attention, and MLP layers, where our method is applied, and we focus on low-step settings (4 and 8 steps), as LCM is a step distillation method that reduces the number of inference steps through full-network finetuning. The resulting distributions, shown as histograms in Figure 9, reveal low similarity across timesteps. This observation suggests that when using LCM sampler, features vary significantly between steps, which may limit the effectiveness of our caching-based method. In line with our observation and conjecture, applying a caching-based method, Δ-DiT [5], to PixArt-α with LCM resulted in minimal effectiveness (Table 3 in [5]).

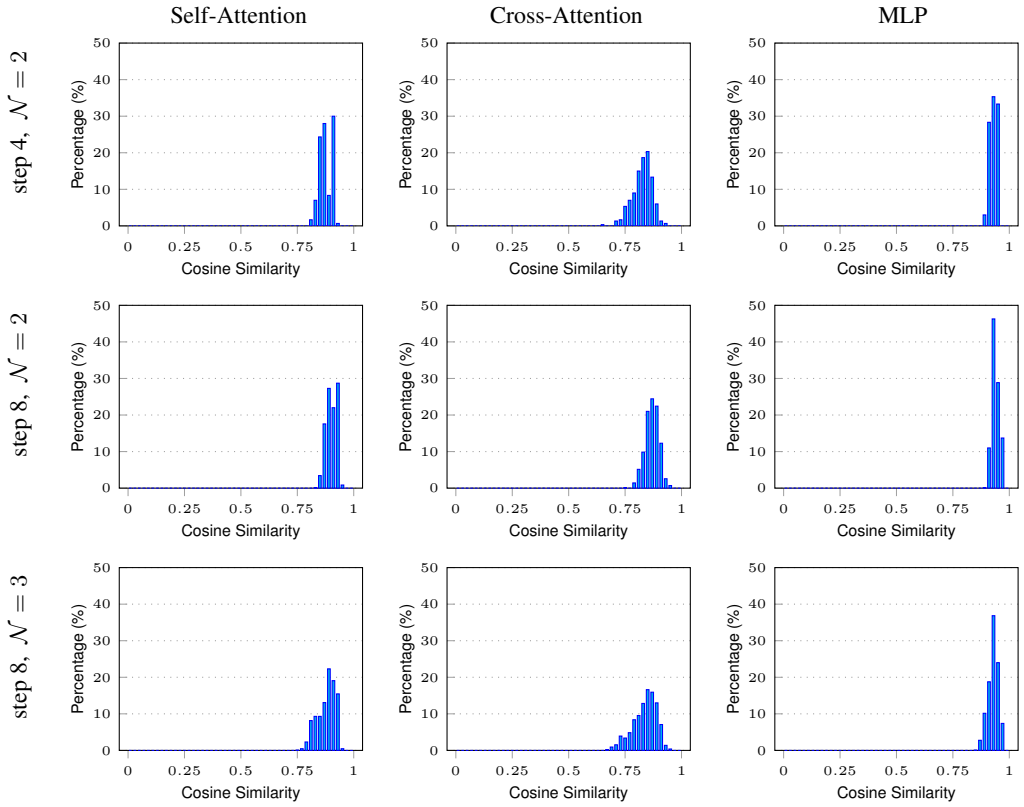


Figure 9: Visualization of self-attention, cross-attention, and MLP features in PixArt-α with LCM sampler. The top three histograms show the cosine similarity distribution between feature pairs sampled at a two-timestep interval in the 4-step model. The middle three histograms depict the distributions for the 8-step model with a two-timestep interval. The bottom three plots present the distributions for the 8-step model with a three-timestep interval. Across all cases, the distributions indicate low cosine similarity between features across timesteps.

C Further Comparison with Prior Methods

We report additional evaluations of prior caching-based methods under their primary experimental settings, which could not be included in Section 4 due to differences in configurations.

C.1 Comparison with Various Methods under $\mathcal{N} = 2$

We compared the proposed framework with existing efficiency improvement methods when the activation cycle is $\mathcal{N}$=2, meaning the network performs full network inference and save intermediate features every two steps. Table 15 shows results for class-conditional 50K image generation using DiT-XL/2 [35] with DDIM [44] 50-step sampler on ImageNet 1K classes [7]. Our method demonstrates superior generation performance compared to other techniques with similar FLOPs, such as 52% steps, 50% steps, L2C (35 steps) [30], LazyDiT [42] and FORA [41]. Additionally, our framework achieves similar image quality with approximately 50% of the FLOPs and $1.68\times$ inference speed required for the original DDIM 50-step inference. Latency for all methods are measured on an H100 GPU to evaluate the speed-up. However, since LazyDiT's pretrained model is not publicly available, its latency is omitted in the table. LazyDiT is expected to have similar inference latency to L2C, as it similarly alternates between full inference and caching.

Table 15: Results for activation cycle $\mathcal{N}$=2. Our method outperforms other efficiency improvement techniques in generation performance at comparable FLOPs. The proposed framework requires almost 50% computation to achieve similar image quality compared to the baseline DiT-XL/2 50-step sampling.

Method	FLOPs↓	Speed↑	FID↓	sFID↓	IS↑	Precision↑	Recall↑
DiT-XL/2 (50 steps, baseline)	11.44 T	1.00×	2.26	4.29	238.60	0.80	0.60
52% steps	5.95 T	1.92×	2.82	4.49	232.22	0.80	0.59
50% steps	5.72 T	1.99×	2.85	4.54	231.20	0.80	0.58
L2C (50 steps) [30]	8.72 T	1.24×	2.27	4.23	244.10	0.81	0.59
L2C (35 steps) [30]	5.92 T	1.81×	2.92	4.60	229.52	0.79	0.59
LazyDiT [42]	5.74 T	-	2.70	4.47	237.03	0.80	0.59
FORA [41]	5.74 T	1.68×	2.61	4.69	236.14	0.81	0.58
ToCa (R=93%) [57]	6.77 T	1.25×	2.92	4.51	225.77	0.79	0.58
Ours	5.74 T	1.68×	2.43	4.57	236.63	0.80	0.59

C.2 Comparison with DuCa

In Table 1, we compared our method with various existing caching strategies under the class-conditional 50K image generation setting using DiT-XL/2 [35] with DDIM [44] (50 steps) on ImageNet-1K [7]. Additionally, we include a comparison with DuCa [56], a recently proposed token-wise caching method in Table 16. DuCa reports results under multiple configurations. DuCa(a) and DuCa(b) differ in the placement of caching steps: DuCa(a) inserts aggressive caching immediately after the fresh step, while DuCa(b) places conservative caching at the earliest position. S-A score denotes token selection based on self-attention scores, and V-norm uses vector norms for caching. Among the DuCa variants, DuCa(b) (S-A) and DuCa(b) (V-Caching) achieve FID scores comparable to ours, but both require $1.23\times$ more FLOPs than our method. While results for several configurations are reported in DuCa, the current implementation[4] lacks sufficient documentation or clarification of specific experimental settings, making it difficult to measure certain metrics such as runtime or Inception Score (IS) at the time of submission. Accordingly, we omit these entries from our comparison table.

Table 16: Qualitative results for class-conditional 50K image generation with DiT-XL/2 and DDIM 50 steps on ImageNet 1K classes. Our method requires significantly less computation than the baseline to achieve comparable image generation quality and outperforms other approaches with similar FLOPs in terms of generation performance.

Method	FLOPs↓	Speed↑	FID↓	sFID↓	IS↑	Precision↑	Recall↑
DiT-XL/2 (50 steps, baseline)	11.44 T	1.00×	2.29	4.32	239.54	0.81	0.60
50% steps	5.72 T	1.99×	2.92	4.51	228.15	0.80	0.58
36% steps	4.12 T	2.76×	3.94	5.16	216.29	0.78	0.58
FORA ($\mathcal{N}$=3) [41]	3.91 T	2.16×	3.28	5.72	228.47	0.80	0.56
ToCa ($\mathcal{N}$=3, R=93%) [57]	5.12 T	1.40×	3.04	4.74	-	0.80	0.57
ToCa ($\mathcal{N}$=4, R=93%) [57]	4.37 T	1.49×	3.60	5.11	-	0.79	0.56
DuCa(a) (S-A score) [56]	4.38 T	-	3.19	4.66	-	0.79	0.57
DuCa(b) (S-A score) [56]	4.79 T	-	3.05	4.66	-	0.80	0.57
DuCa(a) (V-Caching) [56]	4.38 T	-	3.19	4.68	-	0.80	0.57
DuCa(b) (V-Caching) [56]	4.79 T	-	3.06	4.69	-	0.80	0.57
Ours ($\mathcal{N}$=3)	3.91 T	2.15×	2.90	4.60	231.54	0.80	0.58

C.3 Comparison with T-GATE

To ensure a fair comparison with T-GATE [23], we evaluated our model using a similar number of MACs to those reported for their PixArt-α 25-step model [3] in Table 4 of [23]. This comparison is performed on COCO-10K validation prompts[5]. Table 17 shows that our model achieves significantly better generation performance and faster inference speed.

Table 17: Qualitative results for text-to-image generation using PixArt-α with DPM++ (25 steps, 1024×1024) on COCO-10K validation prompts. Our method outperforms T-GATE in image quality and latency under comparable MAC settings.

Method	Latency (s)↓	FPS↑	MACs↓	FID↓
PixArt-α (25 steps, baseline) [3]	1.43	0.70	107.03 T	38.67
T-GATE [23]	0.62	1.63	64.14 T	38.42
Ours ($\mathcal{N}$=4)	0.53	1.91	60.03 T	32.77

[4] `https://github.com/Shenyi-Z/DuCa`

[5] `https://github.com/HaozheLiu-ST/T-GATE/files/15369063/idx_caption.txt`

D Comparison with Additional Metrics

This section presents the complete results of several experiments for which performance on various metrics could not be reported in Section 4 due to space constraints.

D.1 Results on SD3 and Sana

In Table 3, we evaluated our method using SD3-medium [10] and Sana-0.6B [52] with Flow-Euler [22] 20-step sampler. We additionally report CLIP score [13] and inference speed in Table 18. We see that the proposed method with activation cycle $\mathcal{N} = 3$ generally yields better generation quality than the naive caching method (FORA), with negligible additional FLOPs and inference latency.

Table 18: Text-to-image generation performance of SD3-medium and Sana-0.6B with Flow-Euler sampler (20 steps, 256×256) on COCO2014 30K validation prompts. Our method generally outperforms the naive caching method in image quality, while incurring only a marginal increase in FLOPs and inference latency. The second column (Mod) indicates the percentage of layers with modulation applied.

Model	Mod	FLOPs↓	Speed↑	FID↓	CLIP↑
SD3 (baseline) [10]	-	11.08 T	1.00×	23.58	15.81
FORA ($\mathcal{N}$=3) [41]	0.00%	3.92 T	1.56×	47.27	15.60
Ours ($\mathcal{N}$=3)	1.50%	3.92 T + 11.01 M	1.54×	33.60	15.55
Sana (baseline) [52]	-	4.45 T	1.00×	20.40	16.19
FORA ($\mathcal{N}$=3) [41]	0.00%	1.62 T	1.74×	23.19	16.15
Ours ($\mathcal{N}$=3)	1.83%	1.62 T + 11.80 M	1.73×	22.25	16.18

D.2 Results on High-resolution

Table 6 reports results on high-resolution text-to-image generation, where we apply our framework to Sana-0.6B [52] for generating 512×512 and 1024×1024 images on the COCO2014 5K validation set [21]. We also provide sFID, IS [40], Precision, Recall [17] and CLIP score [13] in Table 19. Our results show that applying the proposed modulation to fewer than 2% of the layers generally improves generation quality across most evaluation metrics, while incurring negligible computational overhead.

Table 19: High-resolution performance of Sana-0.6B with Flow-Euler sampler (20 steps, 512×512 and 1024×1024) on COCO2014 5K validation prompts. Our method generally shows better image quality than the naive caching method with negligible additional FLOPs. The second column (Mod) indicates the percentage of layers with modulation applied.

Model	Mod	FLOPs↓	FID↓	sFID↓	IS↑	Precision↑	Recall↑	CLIP↑
Sana-512 (baseline) [52]	-	12.19 T	31.21	72.28	39.62	0.64	0.34	16.11
FORA ($\mathcal{N}$=3) [41]	0.00%	4.33 T	29.66	80.73	39.52	0.59	0.35	16.20
Ours ($\mathcal{N}$=3)	1.10%	4.33 T + 28.31 M	29.16	79.48	40.19	0.60	0.36	16.22
Sana-1024 (baseline) [52]	-	43.11 T	29.91	70.00	42.38	0.61	0.39	16.14
FORA ($\mathcal{N}$=3) [41]	0.00%	15.16 T	29.45	81.89	41.05	0.55	0.39	16.28
Ours ($\mathcal{N}$=3)	0.55%	15.16 T + 66.06 M	29.17	81.54	41.33	0.55	0.40	16.28

E Additional Figures

E.1 Scatter Plots

In addition to Figure 1, we show more scatter plots to visualize feature values of attention layers at two-timestep intervals in Figure 10. In the early denoising stage (larger timestep and smaller block index), features tend to exhibit high linearity between two-timestep intervals, whereas the linearity decreases as the process progresses to the later stage (smaller timestep and larger block index).

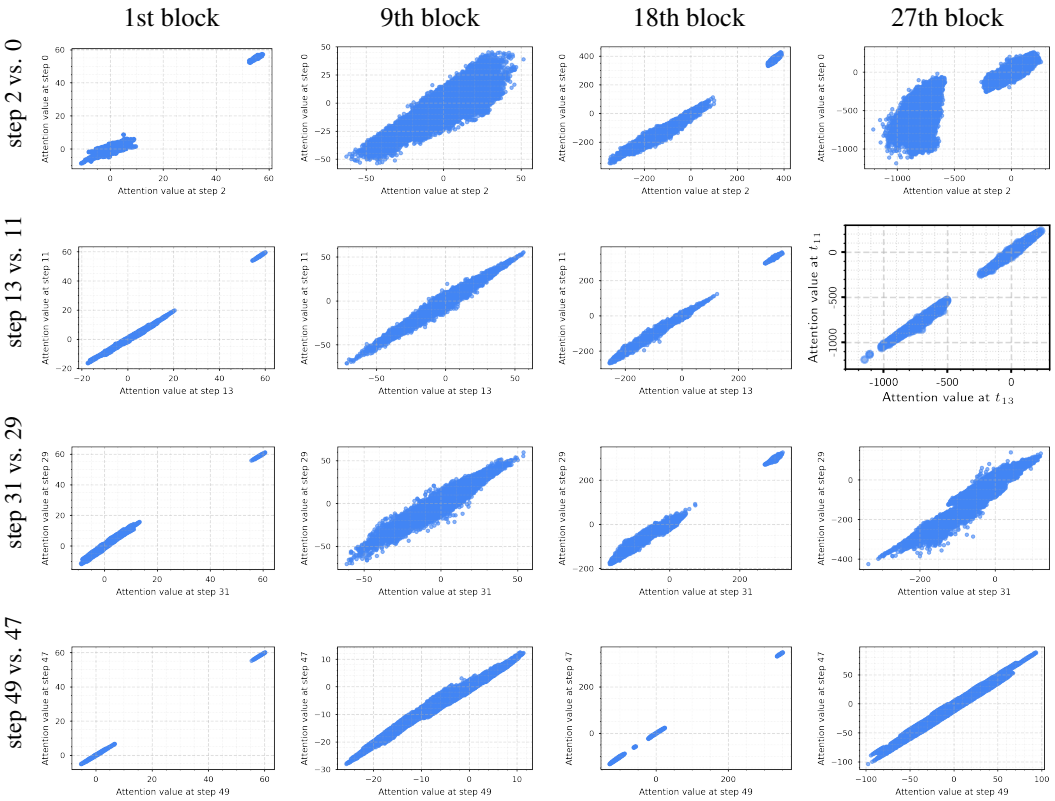


Figure 10: Scatter plots for visualizing feature values of attention layers at two-timestep intervals.

E.2 Learning Pattern of Modulation Gate

Due to space constraints, only the final heatmap of modulation gate scores for attention is included in Figure 6. We plot how the modulation gate scores β change as the training progresses in Figure 11. As gate scores are randomly initialized, they exhibit no distinct structure in the early phase of training. However, as the training progresses, the scores are separated. In both attention and MLP, modulation gate scores are higher in later denoising steps and deeper layers, which tend to need more modulation. Additionally, for MLP features, gate scores are large in the shallow layers.

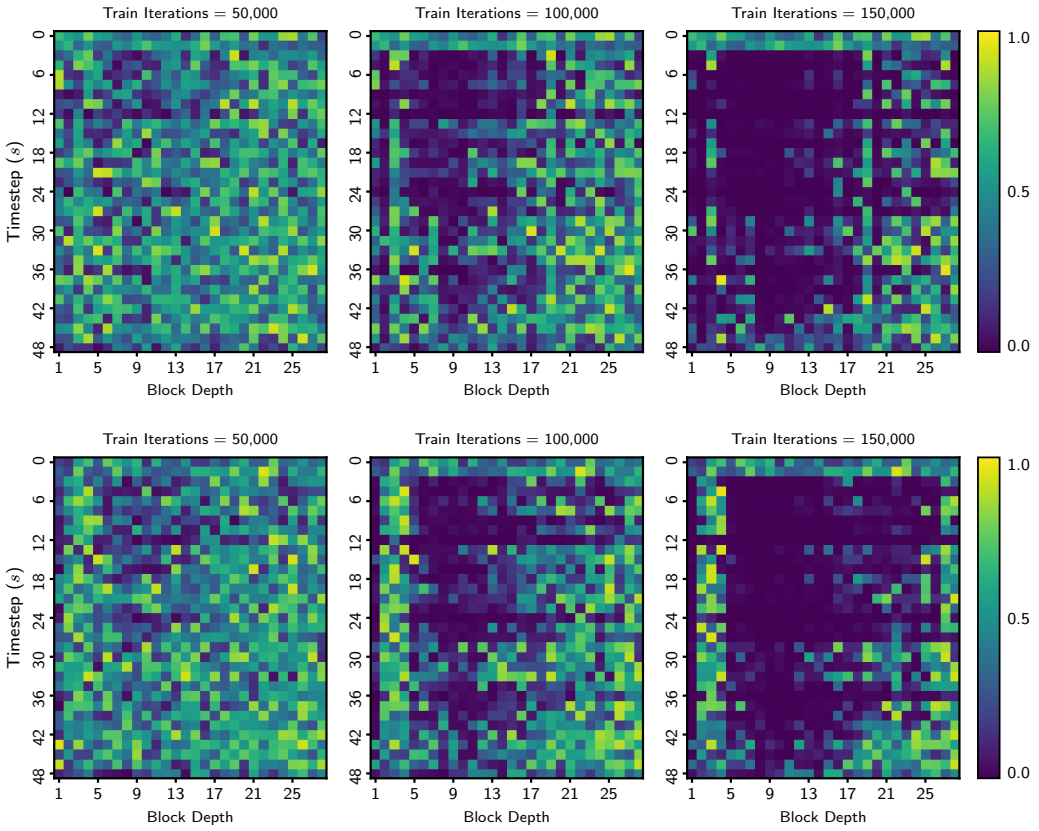


Figure 11: Learning pattern of modulation gate scores for attention (top) and MLP (bottom). There is no clear pattern during the initial phase of training. As training progresses, gate scores for both attention and MLP show an increase in values for later denoising steps and deeper layers, indicating a need for modulation in those parts. Additionally, for MLP features, gate scores tend to be larger in the shallow layers.

E.3 Generated Images

To demonstrate qualitative results of our method, we apply both our approach and a caching-only method (FORA [41]) to DiT-XL/2 [35] with DDIM [44] (50 steps, baseline) and an activation cycle of $\mathcal{N} = 3$. Figure 12 show 256×256 resolution generations for five ImageNet [7] classes. Overall, FORA tends to produce blurrier regions, while our method preserves higher sharpness. In the keeshond example, FORA distorts the appearance of the eyes. In the car wheel image, the orange paint appears faded. For bullfrog, the fine texture details of the rock are lost. In the case of nail, part of the head is missing, and in green mamba, the entire image appears noticeably more blurry.

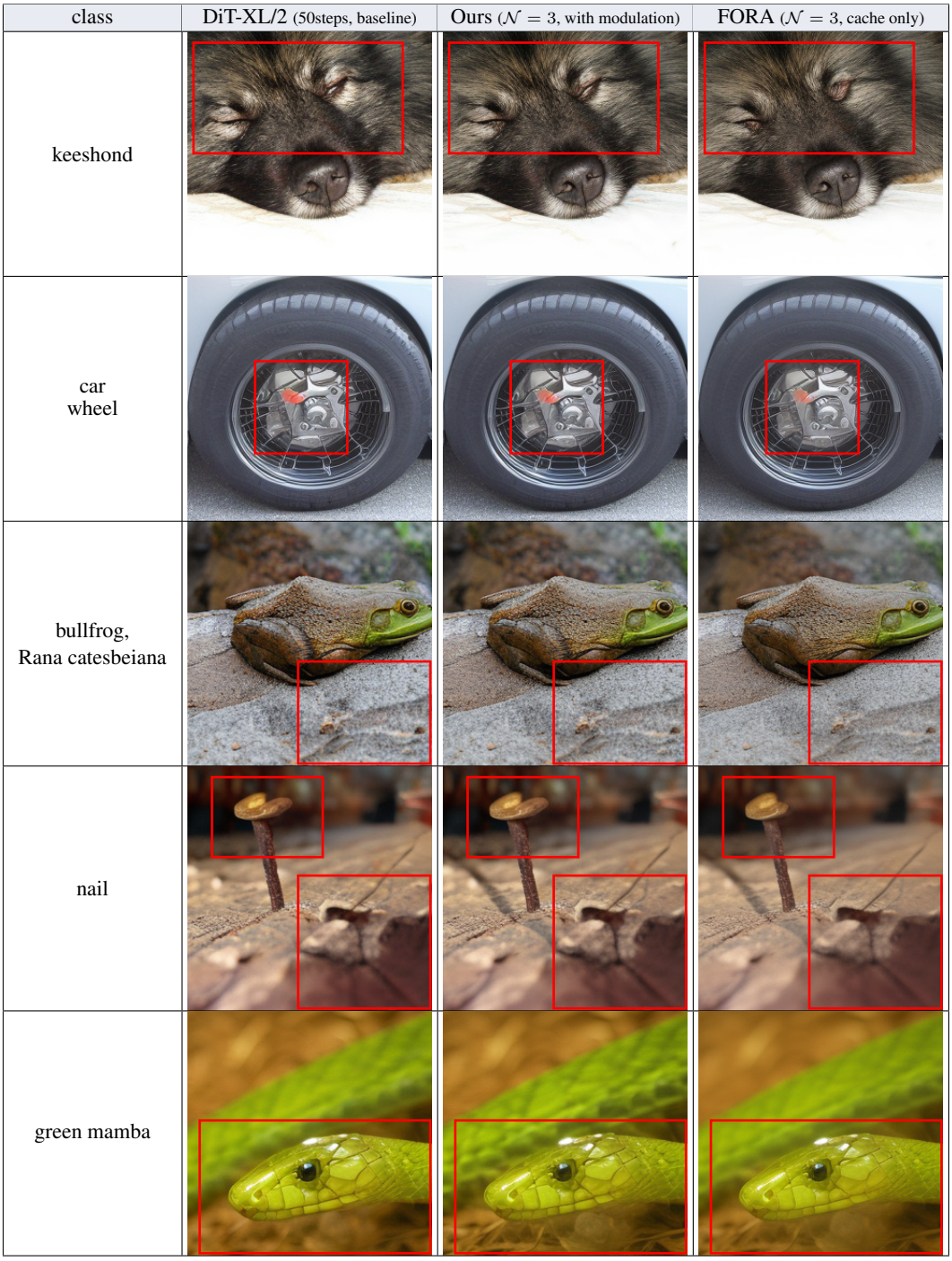


Figure 12: Qualitative comparison of baseline DiT-XL/2, FORA, and our method on ImageNet.

We further qualitatively compare the generation performance of our method and baseline method in Figure 13. The images in the top row are generated by Sana-0.6B [52] (COCO2014 [21], 512×512, 20-step Flow-Euler [22]), while the bottom shows results from our method ($\mathcal{N}$ = 3). Generation quality is almost identical, despite our method reduces FLOPs by 2.82×.

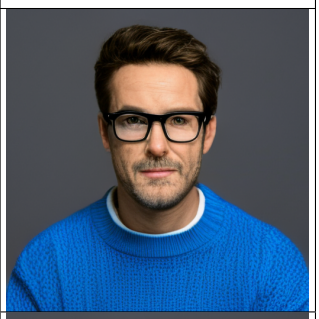
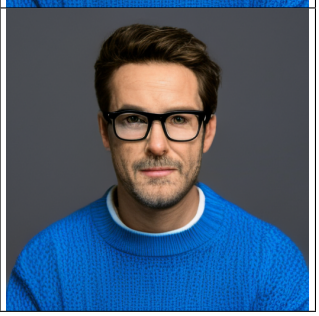
Prompt	A person standing with a broken umbrella.	A man sitting on his surfboard looking out into the ocean.	A man wearing glasses and a blue sweater.
Baseline			
Ours			

Figure 13: Qualitative comparison of baseline Sana-0.6B and our method on COCO2014.