Accelerating Diffusion Transformer via Gradient-Optimized Cache

Junxiang Qiu[1], Lin Liu[1], Shuo Wang[1]*, Jinda Lu[1], Kezhou Chen[1], Yanbin Hao[2]
[1]University of Science and Technology of China; [2]Hefei University of Technology.
{qiujx, liulin0725, lujd, chenkezhou}@mail.ustc.edu.cn, shuowang.edu@gmail.com, haoyanbin@hotmail.com

Abstract

Feature caching has emerged as an effective strategy to accelerate diffusion transformer (DiT) sampling through temporal feature reuse. It is a challenging problem since (1) Progressive error accumulation from cached blocks significantly degrades generation quality, particularly when over 50% of blocks are cached; (2) Current error compensation approaches neglect dynamic perturbation patterns during the caching process, leading to suboptimal error correction. To solve these problems, we propose the Gradient-Optimized Cache (GOC) with two key innovations: (1) Cached Gradient Propagation: A gradient queue dynamically computes the gradient differences between cached and recomputed features. These gradients are weighted and propagated to subsequent steps, directly compensating for the approximation errors introduced by caching. (2) Inflection-Aware Optimization: Through statistical analysis of feature variation patterns, we identify critical inflection points where the denoising trajectory changes direction. By aligning gradient updates with these detected phases, we prevent conflicting gradient directions during error correction. Extensive evaluations on ImageNet demonstrate GOC's superior trade-off between efficiency and quality. With 50% cached blocks, GOC achieves IS 216.28 (26.3%↑) and FID 3.907 (43%↓) compared to baseline DiT, while maintaining identical computational costs. These improvements persist across various cache ratios, demonstrating robust adaptability to different acceleration requirements. Code is available at https://github.com/qiujx0520/GOC_ICCV2025.git.

1. Introduction

Diffusion Transformers (DiT) [7, 19, 26, 28, 38] have shown a powerful ability in content generation in synthesizing multimodal content spanning textual[20, 44], visual[35, 42, 43], and temporal[8, 27, 34] domains. It is widely applied in numerous fields, including intelligent creation assistance, information processing, and digital entertainment content generation. However, it is time-consuming to sample during their forward and reverse processes [10, 30]. This significantly hinders the technology's rapid deployment and flexible application in the real world. Therefore, accelerating the content generation process and improving its inference efficiency have become an important issue [22].

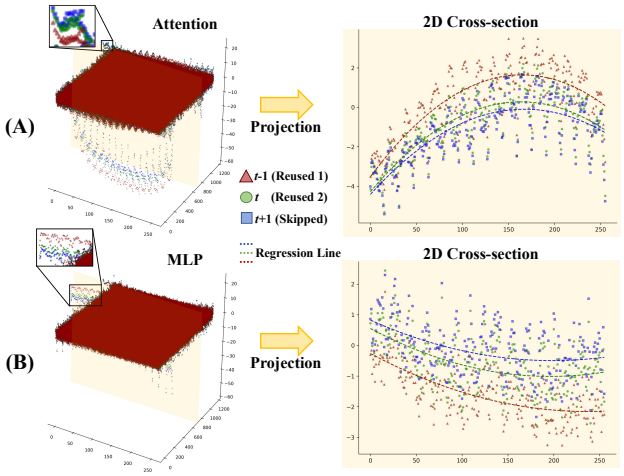

Figure 1. Outputs of the two most recently reused steps $t-1, t$ (red triangles and green circles) and the step $t+1$ (blue squares) that will be skipped in the DiT sample process for the (A) attention layer and the (B) MLP layer. Meanwhile, to better observe the direction of these points, we also provide the fitting curves of these points (in red, green, and blue dotted lines).

Recent studies usually use pruning [17] or caching [30] strategies to accelerate the generation process. In contrast to pruning techniques that achieve acceleration through structural simplification by removing redundant parameters or deactivating non-critical neurons, caching-based approaches leverage the temporal coherence in sequential generation processes to reuse intermediate computational states across multiple sampling steps. This paradigm shift preserves the model's original expressive capacity and style characteristics while significantly enhancing sampling ef-

*Shuo Wang is the corresponding author

ficiency through computational reuse. However, current caching implementations exhibit a critical limitation: The prevalent practice of direct state reuse in consecutive steps lacks rigorous error analysis of cached computations [22, 30, 36]. Specifically, the recursive application of cached features introduces error propagation in multi-step generation processes, where approximation errors accumulate through successive caching operations.

Smoothcache[16] uses numerical gradients at different time steps in the sampling process to locate modules suitable for caching, without discussing how to reduce the errors that have already occurred. Knowledge editing[23] affects generated content by introducing perturbations. These works provide us with inspiration for reducing caching errors. We believe that these caching errors can be offset by incorporating these gradients into the caching process.

To intuitively illustrate this, we visualized the outputs of the Attention and MLP layers during the DiT computation process in Figure 1(A) and (B), respectively, where the red triangles, green circles, and blue squares respectively represent: the outputs from the two most recently reused computational steps, and the output of the step to be skipped in subsequent calculations. Previous caching methods typically employ the green circles (recently reused outputs) to directly substitute the blue squares (computations to be skipped). However, we identify a persistent yet systematic positional deviation between these geometrically similar patterns. Our visualization analysis reveals that incorporating the temporal dimension of the red triangles (historical reused states) exposes a critical directional relationship: the compensation error manifests as a vector space displacement where adjusting the green circle's position along the inverse gradient direction of the red triangle can effectively minimize its deviation from the target blue square. Thus, we leverage the gradient compensation mechanism to strategically integrate gradients from the two most recent reused computations to adaptively adjust cached content, thereby addressing the positional drift in next-step cache generation.

Specifically, we first propose a gradient caching method to reduce caching errors. Concurrently, we leverage statistical information from the model to guide gradient caching implementation, avoiding inverse gradient optimization pitfalls while ensuring high-quality generation. To achieve this, we integrate a queue mechanism into the caching process. By calculating gradients between the two most recently cached contents and incorporating them into subsequent cached content, we align reused features more closely with skipped features, thereby minimizing caching errors. Additionally, we observe that a subset of skipped features exhibit gradient direction misalignment with cached gradients. Applying gradient optimization to these blocks introduces noise. Notably, perturbations introduced in early sampling steps allow partial error correction through subsequent mappings, resulting in limited sampling side effects. In contrast, late-step perturbations propagate errors directly to the final generated image.

To further enhance Gradient Cache's error elimination capability, we calculate inverse-gradient block proportions per step using model statistics. Combining these proportions with step positions in the sampling process, we devise a gradient cache optimization decision strategy. This approach prevents major error introduction while confining residual minor errors to later sampling stages, thereby maximizing gradient optimization benefits. We call the entire cache calculation process Gradient-Optimized Cache (GOC). In summary, the contributions are threefold:

- We design a new basic model caching strategy that leverages the unique gradients in the caching process to reduce errors in future caching.
- We propose a method to determine when gradient cache optimization needs to be applied, aiming to reduce the errors associated with gradient cache optimization.
- Our method reduces caching errors and optimizes the generation quality. It can increase the upper limit of the image generation speed without sacrificing quality or introducing additional computational costs.

2. Related Work

In this section, we first review existing diffusion acceleration methods and then outline the differences between our proposed GOC and these related methods.

2.1. Traditional Sampling Acceleration Method

Common approaches for accelerating sampling in diffusion models fall into three categories: pruning, quantization, and efficient sampling methods. **Pruning** reduces model complexity by removing less critical components while maintaining performance. Methods are broadly categorized into unstructured pruning [6, 11], which masks individual parameters, and structured pruning [17], which eliminates larger structures like layers or filters. DiP-GO [41] introduces a plugin pruner that optimizes pruning constraints to maximize synthesis capability. DaTo [39] dynamically prunes tokens with low activation variance, retaining only high-variance tokens for self-attention computation, thereby enhancing temporal feature dynamics. **Quantization** compresses models by representing weights and activations in lower-bit formats. Key strategies include Quantization-Aware Training (QAT) [1], which embeds quantization into training, and Post-Training Quantization (PTQ) [14, 25], which directly quantizes pre-trained models without retraining. For diffusion models, Q-Diffusion [13] improves calibration through time step-aware data sampling and introduces a specialized noise-prediction network quantizer. PTQ4DiT [37] achieves 8-bit (W8A8) quantization for Diffusion Transformers (DiTs) with minimal quality

loss and pioneers 4-bit weight quantization (W4A8). **Efficient Sampling** seeks to minimize computational overhead while preserving generation quality in diffusion models. This is primarily through two paradigms: retraining-based optimization and sampling-algorithm enhancement. Retraining approaches like knowledge distillation [2, 29] modify model architectures to enable fewer-step generation, albeit at the cost of additional training resources. Conversely, training-free methods focus on refining sampling dynamics. Notably, DDIM [32] accelerates inference via non-Markovian deterministic trajectories, while DPM-Solver [18] employs high-order differential equation solvers to reduce step counts theoretically. Meanwhile, consistency models [33] enable single-step sampling through learned transition mappings. Parallel frameworks like DSNO [40] and ParaDiGMS [31] exploit temporal dependencies between denoising steps for hardware-accelerated throughput.

2.2. Model Caching

In addition to the aforementioned methods, model caching provides a low-cost, efficient, and versatile approach for accelerating diffusion generation. Common caching strategies are primarily divided into two categories: rule-based [22, 28, 30, 36] methods, which reuse or skip specific steps/blocks by analyzing sampling-induced feature variations, and training-based [21] methods, where models learn to skip non-critical modules through training. These strategies are widely integrated into DiT architectures due to their strong learning capabilities, and their effectiveness is further facilitated by the unchanged data dimensionality during sampling. Initially, U-Net-based methods like DeepCache [22] and Faster Diffusion [12] achieved low-loss computation skipping through feature reuse, while Cache-Me-if-You-Can [36] reduced caching errors via teacher-student mimicry. However, such techniques are challenging to adapt directly to DiT. To extend caching to DiT-based models, Fora [30] stores and reuses attention/MLP layer outputs across denoising steps, Δ-DiT [4] accelerates specific blocks via dedicated caching. Additionally, Learning-to-Cache [21] employs a trainable router to dynamically skip layers, achieving higher acceleration ratios but incurring significant computational costs.

Differences: Based on the analysis of related work, our approach falls under the branch of model caching that focuses on optimizing cache errors. The method most closely related to ours is Fora [30]. The key differences between our method and the aforementioned methods are threefold: Firstly, the existing caching methods primarily aim to accelerate sampling by locating and skipping weakly correlated layers. In contrast, our method builds upon their caching mechanisms and leverages gradients to reduce cache errors. Secondly, in GOC, we conduct statistical analysis on the output of each block in the model. Combined with step positions, this approach reduces gradient errors and maximizes the benefits brought by gradient caching. Thirdly, our method can be applied to various existing rule-based and training-based methods, and it can enhance the performance of these methods.

3. Method

In this section, we first briefly revisit the preliminaries of the diffusion method. Second, we illustrate our Gradient-Optimized Cache (GOC) in three parts: (1) Gradient Cache (GC), (2) Model Information Extraction and Statistics, and (3) Gradient Optimization Determination (GOD).

3.1. Preliminaries

Diffusion Models. Diffusion models [10, 32] learn denoising methods to restore random noise $\boldsymbol{x_t}$ back to the original image $\boldsymbol{x_0}$. To achieve this step-by-step, the model needs to learn the reverse process of adding noise. Using Markov chains $\boldsymbol{\mathcal{N}}$, the reverse process $\boldsymbol{p_\theta(x_{t-1} \mid x_t)}$ can be modeled as follows:

$$\boldsymbol{\mathcal{N}}\left(\boldsymbol{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon_\theta}(\boldsymbol{x}_t, t)\right)\right), \quad (1)$$

where t represents the denoising step, β_t denotes the noise variance schedule, $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$, and $\boldsymbol{T}$ represents the total number of denoising steps. $\boldsymbol{\epsilon_\theta}$ is a deep network parameterized by $\boldsymbol{\theta}$. It takes $\boldsymbol{x}_t$ as input and outputs the prediction of the noise required for the denoising process. The aforementioned repeated inference is the primary source of computational cost in diffusion models.

Diffusion Transformer. Diffusion Transformer (DiT) [26] typically consists of stacked self-attention layers $\boldsymbol{S}$, cross-attention layers $\boldsymbol{C}$, and multilayer perceptrons (MLP) $\boldsymbol{M}$, all with identical dimensionality. It can be described by the following formulation:

$$\begin{aligned} \boldsymbol{\mathcal{H}} &= \boldsymbol{\mathcal{G}}_1 \circ \boldsymbol{\mathcal{G}}_2 \circ \cdots \circ \boldsymbol{\mathcal{G}}_l \circ \cdots \circ \boldsymbol{\mathcal{G}}_T, \quad \text{where} \\ \boldsymbol{\mathcal{G}}_t &= \boldsymbol{g}_t^1 \circ \boldsymbol{g}_t^2 \circ \cdots \circ \boldsymbol{g}_t^l \circ \cdots \circ \boldsymbol{g}_t^L, \quad \text{where} \\ \boldsymbol{g}_t^l &= \boldsymbol{S}_t^l \circ \boldsymbol{C}_t^l \circ \boldsymbol{M}_t^l, \end{aligned} \quad (2)$$

where $\boldsymbol{\mathcal{H}}$ represents the entire DiT process, $\boldsymbol{\mathcal{G}}_t$ denotes the process at the t-th step, and L represents the depth of the DiT model at each step. $\circ$ is the elements-wise product. $\boldsymbol{g}_t^l$ refers to the l-th module in the DiT architecture at the t-th step, while $\boldsymbol{S}_t^l$, $\boldsymbol{C}_t^l$ and $\boldsymbol{M}_t^l$ represent the self-attention layer, cross-attention layer, and MLP in a single DiT block, respectively. Subsequently, the general computation between $\boldsymbol{S}_t^l$, $\boldsymbol{C}_t^l$, $\boldsymbol{M}_t^l$ and $\boldsymbol{x}_t^l$ can be written as:

$$\begin{aligned} \boldsymbol{S}_t^l &= \boldsymbol{x}_t^l + \text{AdaLN} \circ \boldsymbol{s}_t^l(\boldsymbol{x}_t^l), \\ \boldsymbol{C}_t^l &= \boldsymbol{S}_t^l + \text{AdaLN} \circ \boldsymbol{c}_t^l(\boldsymbol{S}_t^l), \\ \boldsymbol{x}_{t+1}^l = \boldsymbol{M}_t^l &= \boldsymbol{C}_t^l + \text{AdaLN} \circ \boldsymbol{m}_t^l(\boldsymbol{C}_t^l), \end{aligned} \quad (3)$$

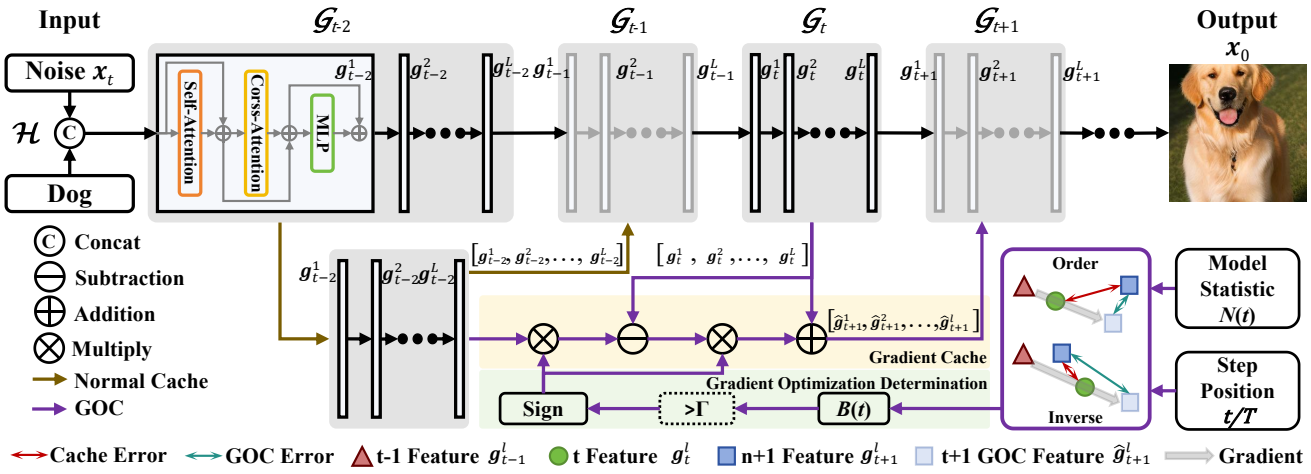

Figure 2. Pipeline of GOC. The input of GOC includes four parts: features $\boldsymbol{g}^l_{t-1}, \boldsymbol{g}^l_t$ from the most recent two Step Caches, model statistical information $\boldsymbol{N}(t)$, and step position t/T. Gradient Cache: the most recent two features are used to calculate the gradient, which is then multiplied by a coefficient and added back to the most recent feature, replacing the feature of the next step that will be skipped. Gradient Optimization Determination: By calculating the weighted sum of the model's statistical information and the step position, and comparing it with a pre-set threshold, it determines whether to use the gradient values computed by GC in the next step.

where $\boldsymbol{x}^l_t$ represents the residual connection, and $\boldsymbol{s}^l_t$, $\boldsymbol{c}^l_t$, $\boldsymbol{m}^l_t$ denote the computation of the self-attention layer, cross-attention layer, and MLP layer, respectively. AdaLN [9] is applied after the computations of $\boldsymbol{s}^l_t(\boldsymbol{x}^l_t)$, $\boldsymbol{c}^l_t(\boldsymbol{x}^l_t)$ and $\boldsymbol{m}^l_t(\boldsymbol{x}^l_t)$, which helps stabilize the training process and enhances the overall performance of the DiT model in handling complex data patterns.

Feature Caching and Reuse. Feature caching [22] aims to reuse time-consuming computation results from past steps to skip and replace future step computations. In our method, if the feature is fully computed at step t, then the computation required at step $t+1$ will be skipped and directly replaced by the computation from step t. Taking the l-th block as an example, the caching process can be expressed as $\boldsymbol{U}[l] := [\boldsymbol{s}^l_t(\boldsymbol{x}^l_t), \boldsymbol{c}^l_t(\boldsymbol{x}^l_t), \boldsymbol{m}^l_t(\boldsymbol{x}^l_t)]$, where ":=" denotes the assignment operation. $\boldsymbol{U}[l]$ caches the computation outputs of the attention layer and the MLP layer from the l-th block at step t. Subsequently, the computation at step $t+1$ is skipped by reusing the cached features, which can be represented as $[\boldsymbol{s}^l_{t+1}(\boldsymbol{x}^l_{t+1}), \boldsymbol{c}^l_{t+1}(\boldsymbol{x}^l_{t+1}), \boldsymbol{m}^l_{t+1}(\boldsymbol{x}^l_{t+1})] := \boldsymbol{U}[l]$.

3.2. Gradient-Optimized Cache

The overview of our framework is depicted in Figure 2. To minimize the error introduced by caching, we compute the gradients of the features from the two most recent steps, $\boldsymbol{g}^l_{t-1}$ and $\boldsymbol{g}^l_t$, multiply them by a coefficient θ, and add them to the features of $\boldsymbol{g}^l_t$. These modified features are then used to replace the features of the next skipped step $t+1$. Additionally, we determine whether to apply normal caching or GOC to a skipped step by comprehensively analyzing the model's statistical information and the position of the step.

Below, we introduce the three steps involved in GOC.

Gradient Cache (GC). Storing features in two steps is necessary to obtain gradients and enable gradient caching. Therefore, we introduce a queue to simultaneously store the features from the two most recent steps, $\boldsymbol{g}^l_{t-1}$ and $\boldsymbol{g}^l_t$. In the early stages of the sampling process when gradients have not yet formed, as indicated by the brown arrow in Figure 2, we employ normal caching as:

$$\boldsymbol{g}^l_1 := \boldsymbol{g}^l_0. \tag{4}$$

This involves reusing the results of $\boldsymbol{g}^l_0$ and skipping the computation of $\boldsymbol{g}^l_1$. When $t > 3$, we use the cached features from the two steps preceding the skipped step to compute the final reused feature. The computation method of Gradient Cache (GC) is as follows:

$$\boldsymbol{g}^l_{t+1} := \hat{\boldsymbol{g}}^l_{t+1} := \boldsymbol{g}^l_t + \eta * (\boldsymbol{g}^l_t - \boldsymbol{g}^l_{t-1}), \tag{5}$$

where $\hat{\boldsymbol{g}}^l_{t+1}$ is the feature computed from $\boldsymbol{g}^l_{t+1}$ that will be reused, and η is a positive parameter used to adjust the magnitude of the gradient.

Model Information Extraction and Statistics. However, excessive application of GC introduces additional gradient noise during error mitigation, ultimately degrading image generation quality. The fundamental source of this gradient noise stems from directional conflicts in gradient vectors - specifically, the expected input vector $\boldsymbol{g}^l_{t+1}$ (represented by the blue square in Figure 2) deviates in the opposite direction from the gradient path connecting $\boldsymbol{g}^l_{t-1}$ (red triangle) to $\boldsymbol{g}^l_t$ (green circle). As illustrated in Figure 2, the computed GC feature $\hat{\boldsymbol{g}}^l_{t+1}$ (light blue square) exhibits a displacement from the ideal $\boldsymbol{g}^l_{t+1}$ position, quantified by two

distinct error metrics: the mint green arrow represents the GC approximation error, while the red arrow indicates the inherent caching error in normal operations. This geometric relationship reveals that GC achieves optimal caching fidelity when gradient directions remain consistent, but introduces progressively larger approximation errors as directional conflicts between successive gradients intensify.

To avoid introducing errors, we need to analyze the data $\boldsymbol{g}_t^l$ for each block of the DiT model to identify steps where inverse gradients are likely to occur. We achieve this by repeatedly performing the sampling process with different prompts and recording the outputs of the self-attention layers, cross-attention layers (if present), and MLP layers, denoted as $\boldsymbol{f}_{\text{sAttn}}(\boldsymbol{x})$, $\boldsymbol{f}_{\text{cAttn}}(\boldsymbol{x})$, and $\boldsymbol{f}_{\text{MLP}}(\boldsymbol{x})$. We collectively refer to these matrices as $\boldsymbol{F}_t^l(k) \in \mathbb{R}^{m\times n}$, where k represents different prompts, and $m \times n$ denotes the dimensionality of the feature. Next, we calculate the average $\boldsymbol{A}_t^l$ for each block's $\boldsymbol{F}_t^l(k)$ as follows:

$$\boldsymbol{A}_t^l = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{F}_t^l(k), \tag{6}$$

where K represents the total number of prompts. If there exists η, it can satisfy:

$$\boldsymbol{J}(\boldsymbol{A}_{t+1}^l - \boldsymbol{A}_t^l) > \boldsymbol{J}(\boldsymbol{A}_{t+1}^l - (\boldsymbol{A}_t^l + \eta * (\boldsymbol{A}_t^l - \boldsymbol{A}_{t-1}^l))), \tag{7}$$

and we believe that $\boldsymbol{g}_{t-1}^l$, $\boldsymbol{g}_t^l$, and $\boldsymbol{g}_{t+1}^l$ form a positive gradient if they align in the expected direction; otherwise, they form an inverse gradient. $\boldsymbol{J}$ is defined as the sum of the absolute values of each element s in the $m \times n$ matrix, where a larger result indicates a greater error. Finally, we record the number of blocks $\mathbf{N}(t)$ exhibiting inverse gradients at each step t. Steps with a higher count of such blocks are more likely to introduce gradient errors.

Gradient Optimization Determination (GOD). Determining whether a step will introduce gradient errors is not sufficient by merely calculating inverse gradients. Previous studies [23, 24] have shown that perturbations introduced closer to the end of the sampling process have a more significant impact on the generated results. From another perspective, even if errors are introduced in the early stages of the sampling process, they can be mapped to correct patterns through subsequent steps. However, errors introduced in the later stages of the sampling process are difficult to eliminate and may lead to artifacts. Therefore, we comprehensively consider the number of inverse gradients and the position of the step to measure the negative impact $\boldsymbol{B}(t)$ of introducing gradient errors at a specific step t:

$$\boldsymbol{B}(t) = \gamma * (1 - \frac{t}{T}) + (1 - \gamma) * \boldsymbol{N}(t), \tag{8}$$

where γ is a parameter used to adjust the balance between the step position and $\boldsymbol{N}(t)$. We set a threshold $\boldsymbol{\Gamma}$, and only when $\boldsymbol{B}(t) < \boldsymbol{\Gamma}$ do we perform GC at step t.

4. Experiment

In this section, we evaluate the proposed GOC method and compare it with rule-based and training-based caching methods. We also use ablation experiments to verify the effectiveness of the method proposed in this paper. We aim to address the following research questions (**RQ**):

RQ1: Is Gradient Caching (GC) effective?

RQ2: Is Gradient Optimization Determination (GOD) effective?

RQ3: What hyperparameter should be selected?

RQ4: What are the advantages and general applicability of GOC?

4.1. Experiment Settings

Datasets. To comprehensively evaluate the performance of diffusion models, we conduct experiments on two benchmark datasets: ImageNet [5] (1,000 classes) for class-conditional generation and MS-COCO [15] (30,000 text prompts) for text-to-image synthesis. Unless explicitly stated otherwise, the ablation experiments default to using the ImageNet dataset.

Model Configuration. We used two models to verify the effectiveness of our method: DiT [22] and Pixart [3]. For the DiT architecture, we adopt DDIM sampler [32] with 20 denoising steps throughout all experiments, while Pixart models use the DPM-Solver [18] under the same 20-steps. To ensure fair comparison across different caching strategies, we generate 50,000 samples for DiT and 30,000 samples for Pixart respectively, matching each model's native output characteristics. The experiments are conducted on an NVIDIA A40 GPU.

Evaluation Metrics. We employ a comprehensive set of quantitative metrics: such as Inception Score(IS↑), Fréchet Inception Distance(FID↓), Sliced Fréchet Inception Distance(sFID↓), Precision(↑), and Recall(↑). All metrics are computed at 256×256 resolution using randomly generated samples to ensure statistical significance.

In our experiment, each cache is reused at most once. For convenience, we provide the abbreviations of Gradient Cache (GC), Gradient Optimization Determination (GOD), and Gradient-Optimized Cache (GOC). In addition, the data of L2C is completely reproduced using the open-source code provided by the authors of Learning-to-Cache[21]. And the data of FORA is implemented by summarizing the experience from previous work [30], where 25% and 50% of the blocks are cached.

4.2. Gradient Cache

Effect of GC (RQ1). To evaluate the performance characteristics of Gradient Caching (GC), Figure 3 presents three comparative elements: single-step GC metrics (blue trajectory), multi-step cumulative GC metrics (orange trajectory),

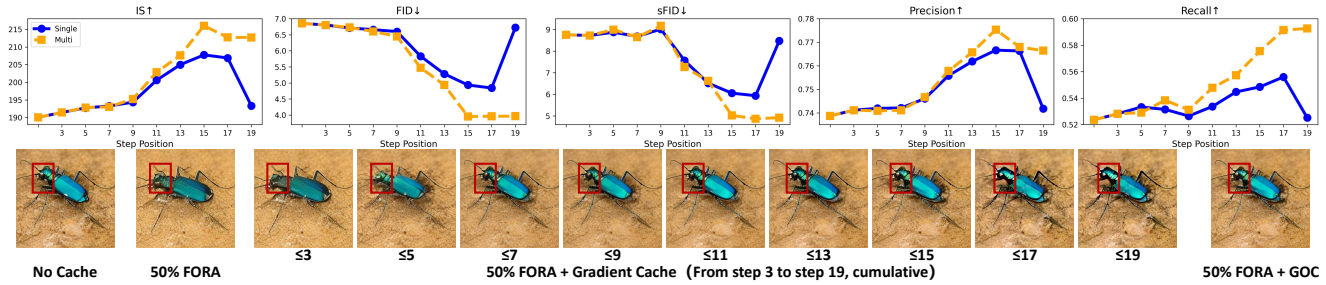

Figure 3. The line chart above shows the metrics generated by single-step GC (solid blue line) and cumulative GC (dashed orange line). And the beetle generated images below include those from No Cache, 50% rule-based caching (FORA[30]), 50% rule-based caching (FORA) with varying strengths of GC, and 50% FORA + GOC.

Cache Strategy	Caching Level	IS↑	FID↓	sFID↓	Precision↑	Recall↑
No Cache[26]	0%	223.490	3.484	4.892	0.788	0.571
FORA[30]	50%	190.056	6.857	8.757	0.739	0.524
FORA+GC		212.691	3.964	**4.923**	0.766	**0.593**
FORA+GOC		**216.280**	**3.907**	4.972	**0.775**	0.574

Table 1. Metrics of the generated images under No Cache, as well as FORA, FORA+GC, and FORA+GOC with 50% caching level.

and corresponding beetle image generations. The blue trajectory demonstrates consistent metric improvement across all optimization steps compared to the baseline initialization (leftmost data point). The orange trajectory reveals progressive metric enhancement through successive GC layer integration until step 17, where we observe severe degradation in both Inception Score (IS) and precision metrics, indicative of accumulated gradient approximation errors. By comparing the beetle images generated with 50% cache + Gradient Cache, 50% Cache, and No Cache, we can find that 50% cache + GC exhibit progressive quality improvements through step 15, progressively approximating the quality of the No Cache configuration. However, post-step 17 generations display pronounced artifacts in the cephalic region and thoracic segments (highlighted in red boxes). Therefore, an appropriate level of GC can overcome most errors introduced by Normal Caching. However, excessive GC introduces gradient errors and reduces image quality, making it necessary to select for GC application.

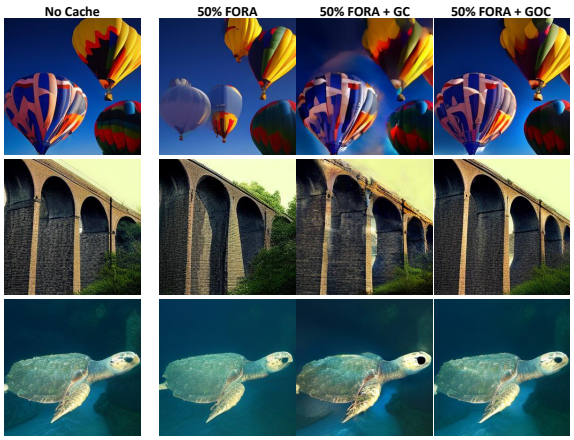

Figure 4. Comparison of generated images are compared under No Cache, 50% FORA, 50% FORA + GC, and 50% FORA + GOC. Here, GC is used at each caching step.

4.3. Gradient Optimization Determination

To verify the effectiveness and rationality of GOD, we design a two-part experiment. The first part demonstrates the improvements in metrics and images achieved by incorporating GOD, while the second part focuses on the selection of GOD parameters.

Effect of GOD (RQ2). To systematically evaluate caching optimization strategies, Table 1 and Figure 4 provide comparative analyses of four configurations: No Cache, FORA (Reduced Caching), FORA+GC (FORA with Gradient Caching), and FORA+GOD (GC-optimized FORA). Quantitative metrics in Table 1 reveal that the 50% FORA configuration significantly degrades all image quality parameters. Implementing GC across all caching stages substantially enhances performance, achieving optimal sFID (4.923) and Recall (0.593). The FORA+GOD configuration further refines this approach by selectively removing GC steps with adverse effects, attaining peak scores in IS (216.280), FID (3.907), and Precision (0.775). By observing Figure 4, we can see that FORA, which skips half of the computational steps, causes the balloon to deform, the

Cache Strategy	Caching level	η	IS↑	FID↓	sFID↓	Precision↑	Recall↑
FORA[30]+GOC	50%	1.4	204.760	4.432	5.623	0.752	**0.599**
		1.3	213.479	3.940	**4.862**	0.767	0.588
		1.2	216.280	**3.907**	4.972	0.775	0.574
		1.1	**216.941**	4.015	5.262	0.776	0.571
		1	216.013	4.163	5.552	**0.777**	0.566
	25%	2.1	214.884	3.681	4.914	0.7693	**0.593**
		2	218.581	3.526	4.635	0.776	0.587
		1.9	221.082	**3.480**	**4.613**	0.780	0.585
		1.8	222.210	3.498	4.694	0.783	0.580
		1.7	**222.805**	3.524	4.805	**0.784**	0.581
L2C[21]+GOC	22%	1.2	235.369	3.205	**4.629**	**0.802**	0.558
		1.1	236.256	3.196	4.634	0.803	**0.560**
		1	**236.748**	**3.192**	4.644	0.803	0.559
		0.9	236.653	3.202	4.652	0.805	0.558
		0.8	236.469	3.211	4.656	0.804	0.555

Table 2. Metrics of the generated images under FORA+GOC with 50% and 25% caching levels, as well as L2C+GOC with 22% caching level, under different constraint of η.

Cache Strategy	Caching level	IS↑	FID↓	sFID↓	Precision↑	Recall↑	FLOPs(T)↓	Speed↑
No Cache[26]	0%	223.490	3.484	4.892	0.788	0.571	11.868	1×
FORA[30]	50%	190.046	6.857	8.757	0.739	0.524	5.934	2.0000×
	25%	220.011	3.870	5.185	0.783	0.569	8.900	1.3335×
FORA+GOC	50%	**216.280**	**3.907**	**4.972**	**0.775**	**0.574**	5.934	1.9999×
	25%	**222.805**	**3.524**	**4.805**	**0.784**	**0.581**	8.900	1.3334×
L2C[21]	22%	225.004	3.539	4.710	0.788	**0.563**	9.257	1.2821×
L2C+GOC	22%	**236.748**	**3.192**	**4.644**	**0.803**	0.559	9.257	1.2820×

Table 3. Metrics of the generated images before and after applying GOC to FORA or L2C under different caching levels.

bridge arch to disappear, and insufficient details in the turtle. FORA+GC can restore the general shape of the image but introduces artifacts, making the balloon blurry, the bridge distorted, and the turtle's eyes unclear. FORA+GOD can avoid the artifacts of FORA+GC and generate images that closely resemble those of No Cache. While FORA+GC restores basic geometric structures, it introduces visible artifacts: blurred balloon surfaces, warped bridge geometries, and undefined ocular features in the turtle. In contrast, FORA+GOD achieves near-equivalent visual fidelity to the No Cache baseline, successfully resolving both the structural degradation of FORA and the artifact interference from FORA+GC. It demonstrates that GC implementation effectively compensates for FORA's quality degradation but introduces secondary artifacts. The GOD-enhanced configuration achieves optimal balance by retaining GC's corrective benefits while eliminating its error-prone steps, thereby maintaining structural coherence and metric superiority.

Selection of hyperparameter (RQ3). The parameter η determines whether the gradient can maximally compensate for the error introduced by normal caching. As shown in Table 2, we select FORA+GOC with 50% and 25% of blocks cached, as well as L2C+GOC with 22% of blocks cached. We record five values around the optimal η for each case, with a step size of 0.1. We found that when $\eta = 1.2$, the 50% FORA+GOC achieves the most balanced metrics. When $\eta = 1.9$, the 25% FORA+GOC achieves the most balanced metrics. When $\eta = 1$, the 22% L2C+GOC achieves the most balanced metrics. These different values of η reveal a phenomenon: for the case of caching the first half of the odd steps (25% FORA+GOC), a larger η can more effectively reduce the caching error but also introduces a significant amount of gradient error. Since GOC is applied at the earlier steps, the model has sufficient steps to map the gradient error back to normal. For L2C+GOC, the sampling process includes knowledge obtained from train-

Figure 5. Comparison of images generated by the weight trained on the MS-COCO dataset under No Cache, 25% FORA, and 25% FORA + GOC conditions. We have attached the corresponding prompts below the images.

Cache Strategy	Caching level	FID↓
No Cache[3]	0%	21.964
FORA[30]	25%	21.984
FORA+GOC		**21.971**

Table 4. Metrics of the generated images generated by the weight trained on the MS-COCO dataset under No Cache, as well as FORA and FORA+GOC with 50% caching level.

ing, which weakens the effect of GOC. Therefore, the value of η is lower than in the other two cases.

4.4. Comparison with Other Methods

Advantages and general applicability of GOC (RQ4). As shown in Table 3, GOC can be applied to FORA and L2C with different caching levels and can improve the quality of generation. Specifically, in the case of 50% FORA, GOC can significantly increase IS from 190.046 to 216.280, decrease FID from 6.857 to 3.907, and reduce sFID from 8.757 to 4.972. In the case of 25% FORA, GOC can increase IS from 220.011 to 222.805, decrease FID from 3.870 to 3.524, and reduce sFID from 5.185 to 4.805. The improved metrics are very close to those of No Cache, and the sFID even exceeds it. In the case of 22% L2C, GOC can significantly increase IS to 236.748 and decrease FID to 3.192. The improvement is greater than that of the similar proportion FORA, indicating that GOC and training-based caching methods can complement each other. In addition, we use FLOPs(T) to represent the computational load in the sampling process, and it can be seen that the computational cost brought by GOC is negligible.

To demonstrate that GOC is adaptable to both text-to-image and class-to-image tasks, we employ a Pixart model trained on the MS-COCO dataset and utilized the same strategy of caching 25% of the blocks. We then examine the impact on images before and after incorporating GOC. As illustrated in Figure 5, GOC significantly compensates for the detail loss caused by 25% FORA. For instance, GOC enhances the integrity of trains and refrigerators. In addition, Table 4 shows that the introduction of GOC improves the FID score of FORA.

Cache Strategy	FID↓	latency(s)↓
FORA[30] (50%)	6.857	1.789 ± 0.046
FORA+GOC (50%)	**3.907**	1.824 ± 0.029
FORA (25%)	3.870	2.271 ± 0.032
FORA+GOC (25%)	**3.524**	2.305 ± 0.016
L2C[21] (22%)	3.539	1.992 ± 0.011
L2C+GOC (22%)	**3.192**	2.016 ± 0.004

Table 5. Comparison of the average time taken to generate eight images under FORA and FORA+GOC with 50% and 25% caching levels, as well as L2C and L2C+GOC with 22% caching level.

Finally, the original intention of caching is to reduce sampling time, so we need to pay attention to the increased computational load introduced by GOC, which may lead to longer sampling times. As shown in Table 5, the latency increases by approximately 2.0%, 1.5%, and 1.2% for FORA+GOC (50%), FORA+GOC (25%), and L2C+GOC (22%), respectively. Given the significant improvement in FID, these time costs are acceptable.

5. Conclusion

In this paper, we propose Gradient-Optimized Cache (GOC), which leverages a queue to compute cached gradients during the sampling process and integrates these gradients, weighted appropriately, into the original cached features to reduce caching errors. Meanwhile, we utilize the model's statistical information to identify inflection points in the sampling process and combine step positions to avoid counter-gradient optimization, thereby further reducing gradient errors. The proposed GOC can generate higher-quality samples with almost no additional computational cost. Through model comparisons and ablation studies, we achieve better performance under different datasets and various caching strengths for both FORA and L2C, thereby validating the effectiveness of our method.

Note that the implementation of GOC is still limited by fixed parameters and bound steps. Thus, we plan to investigate the development of a general embedding procedure for GOC in the future.

Acknowledgements

This research is supported by the National Natural Science Foundation of China (No.U24B20180, No.62472393) and the advanced computing resources provided by the Supercomputing Center of the USTC.

References

[1] Yash Bhalgat, Jaehoon Lee, Michael Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 696–697, 2020. 2

[2] Corentin Chadebec, Omer Tasar, Elie Benaroche, et al. Flash diffusion: Accelerating any conditional diffusion model for few steps image generation. *arXiv preprint arXiv:2406.02347*, 2024. 3

[3] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv preprint arXiv:2310.00426*, 2023. 5, 8

[4] Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. $\Delta - dit$: A training-free acceleration method tailored for diffusion transformers. *arXiv preprint arXiv:2406.01125*, 2024. 3

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[6] Xuanyi Dong, Shiyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in neural information processing systems*, 2017. 2

[7] Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 23164–23173, 2023. 1

[8] Dan Guo, Shuo Wang, Qi Tian, and Meng Wang. Dense temporal convolution network for sign language translation. In *IJCAI*, pages 744–750, 2019. 1

[9] Yunhui Guo, Chaofeng Wang, Stella X Yu, Frank McKenna, and Kincho H Law. Adaln: a vision transformer for multidomain learning and predisaster building information extraction from images. *Journal of Computing in Civil Engineering*, 36(5):04022024, 2022. 4

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

[11] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H Torr. A signal propagation perspective for pruning neural networks at initialization. *arXiv preprint arXiv:1906.06307*, 2019. 2

[12] Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *CoRR*, 2023. 3

[13] Xiangyu Li, Yujun Liu, Li Lian, Hui Yang, Zhen Dong, Dongdong Kang, et al. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023. 2

[14] Yixuan Li, Rui Gong, Xiaoyu Tan, Yibo Yang, Peng Hu, Qian Zhang, et al. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021. 2

[15] 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 *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pages 740–755. Springer, 2014. 5

[16] Joseph Liu, Joshua Geddes, Ziyu Guo, Haomiao Jiang, and Mahesh Kumar Nandwana. Smoothcache: A universal inference acceleration technique for diffusion transformers. *arXiv preprint arXiv:2411.10510*, 2024. 2

[17] Le Liu, Sheng Zhang, Zenghui Kuang, Aocheng Zhou, Jinhao Xue, Xiaogang Wang, et al. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 1, 2

[18] Chenlin Lu, Yihao Zhou, Fan Bao, Jianfei Chen, Chongxiao 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*, pages 5775–5787, 2022. 3, 5

[19] Jinda Lu, Shuo Wang, Xinyu Zhang, Yanbin Hao, and Xiangnan He. Semantic-based selection, synthesis, and supervision for few-shot learning. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 3569–3578, 2023. 1

[20] Jinda Lu, Junkang Wu, Jinghan Li, Xiaojun Jia, Shuo Wang, YiFan Zhang, Junfeng Fang, Xiang Wang, and Xiangnan He. Damo: Data-and model-aware alignment of multi-modal llms. *arXiv preprint arXiv:2502.01943*, 2025. 1

[21] Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. *arXiv preprint arXiv:2406.01733*, 2024. 3, 5, 7, 8

[22] 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*, pages 15762–15772, 2024. 1, 2, 3, 4, 5

[23] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35: 17359–17372, 2022. 2, 5

[24] Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass editing memory in a transformer. *arXiv preprint arXiv:2210.07229*, 2022. 5

[25] Michael Nagel, Rehan A Amjad, Max Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International*

Conference on Machine Learning, pages 7197–7206. PMLR, 2020. 2

[26] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. 1, 3, 6, 7

[27] Junxiang Qiu, Jinda Lu, and Shuo Wang. Multimodal generation with consistency transferring. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 504–513, 2025. 1

[28] Junxiang Qiu, Shuo Wang, Jinda Lu, Lin Liu, Houcheng Jiang, Xingyu Zhu, and Yanbin Hao. Accelerating diffusion transformer via error-optimized cache. *arXiv preprint arXiv:2501.19243*, 2025. 1, 3

[29] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 3

[30] Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*, 2024. 1, 2, 3, 5, 6, 7, 8

[31] Andrew Shih, Shashank Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. In *Advances in Neural Information Processing Systems*, 2024. 3

[32] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 3, 5

[33] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023. 3

[34] Shuo Wang, Dan Guo, Wen gang Zhou, Zheng jun Zha, and Meng Wang. Connectionist temporal fusion for sign language translation. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1483–1491, 2018. 1

[35] Yuan Wang, Ouxiang Li, Tingting Mu, Yanbin Hao, Kuien Liu, Xiang Wang, and Xiangnan He. Precise, fast, and low-cost concept erasure in value space: Orthogonal complement matters. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 28759–28768, 2025. 1

[36] Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6211–6220, 2024. 2, 3

[37] Jiajun Wu, Haoyu Wang, Yilun Shang, Mubarak Shah, and Yan Yan. Ptq4dit: Post-training quantization for diffusion transformers. *arXiv preprint arXiv:2405.16005*, 2024. 2

[38] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024. 1

[39] En Zhang, Bing Xiao, Jie Tang, et al. Token pruning for caching better: 9 times acceleration on stable diffusion for free. *arXiv preprint arXiv:2501.00375*, 2024. 2

[40] Hengrui Zheng, Weijia Nie, Arash Vahdat, Kamyar Azizzadenesheli, and Anima Anandkumar. Fast sampling of diffusion models via operator learning. In *International conference on machine learning*, pages 42390–42402. PMLR, 2023. 3

[41] Heng Zhu, Di Tang, Jie Liu, et al. Dip-go: A diffusion pruner via few-step gradient optimization. *Advances in Neural Information Processing Systems*, 37:92581–92604, 2025. 2

[42] Xingyu Zhu, Shuo Wang, Jinda Lu, Yanbin Hao, Haifeng Liu, and Xiangnan He. Boosting few-shot learning via attentive feature regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7793–7801, 2024. 1

[43] Xingyu Zhu, Beier Zhu, Yi Tan, Shuo Wang, Yanbin Hao, and Hanwang Zhang. Enhancing zero-shot vision models by label-free prompt distribution learning and bias correcting. *Advances in Neural Information Processing Systems*, 37:2001–2025, 2024. 1

[44] Xingyu Zhu, Beier Zhu, Yi Tan, Shuo Wang, Yanbin Hao, and Hanwang Zhang. Selective vision-language subspace projection for few-shot clip. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 3848–3857, 2024. 1