Accelerating Controllable Generation via Hybrid-grained Cache

Lin Liu[1], Huixia Ben[2*], Shuo Wang[1], Jinda Lu[1], Junxiang Qiu[1], Shengeng Tang[3], Yanbin Hao[3]

[1]University of Science and Technology of China
[2]Anhui University of Science and Technology
[3]Hefei University of Technology
{liulin0725,lujd,qiujx}@mail.ustc.edu.cn, huixiaben@mail.hfut.edu.cn
shuowang.edu@gmail.com, tsg1995@mail.hfut.edu.cn, haoyanbin@hotmail.com

Abstract

Controllable generative models have been widely used to improve the realism of synthetic visual content. However, such models must handle control conditions and content generation computational requirements, resulting in generally low generation efficiency. To address this issue, we propose a **H**ybrid-**G**rained **C**ache (**HGC**) approach that reduces computational overhead by adopting cache strategies with different granularities at different computational stages. Specifically, (1) we use a coarse-grained cache (block-level) based on feature reuse to dynamically bypass redundant computations in encoder-decoder blocks between each step of model reasoning. (2) We design a fine-grained cache (prompt-level) that acts within a module, where the fine-grained cache reuses cross-attention maps within consecutive reasoning steps and extends them to the corresponding module computations of adjacent steps. These caches of different granularities can be seamlessly integrated into each computational link of the controllable generation process. We verify the effectiveness of HGC on four benchmark datasets, especially its advantages in balancing generation efficiency and visual quality. For example, on the COCO-Stuff segmentation benchmark, our **HGC** significantly reduces the computational cost (MACs) by 63% (from 18.22T $\rightarrow$ **6.70**T$\downarrow$), while keeping the loss of semantic fidelity (quantized performance degradation) within 1.5%.

1 Introduction

Controllable generative models (Zhang, Rao, and Agrawala 2023; Li et al. 2024) incorporate conditional information into the content generation process, enabling more precise and tailored outputs compared to traditional generative methods. Thus, these models have gained widespread adoption (Tang et al. 2025b,a; Zhang et al. 2025). However, during the denoising generation process, these models must simultaneously compute two critical components: the control module and the generative module. This dual computation significantly increases computational complexity and inference latency, leading to slower generation speeds and reduced usability.

To investigate the impact of various components in the controllable generative models on generation efficiency and quality, we visualize the intermediate results of incorporating conditions at different stages of the model, as shown in Figure 1. Where the model (a) applies conditions throughout all steps, (b) applies conditions only in the first ten steps, and (c) applies conditions only after the first ten steps.

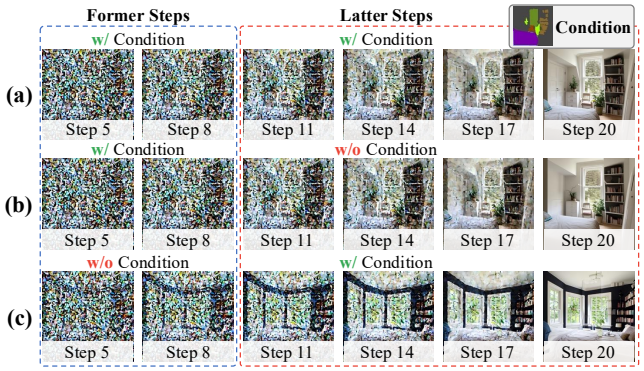


Figure 1: We visualize the intermediate results of adding conditions at different steps in the controllable generative model: (a) adding throughout steps, (b) adding only in the first ten steps, and (c) adding only after ten steps.

Comparing (a) and (b), the image structure increasingly aligns with the conditional image during the latter stages of the model's denoising process, indicating lower computational demands at this point. However, comparing (a) and (c) reveals that omitting the conditional image constraint early on results in significant differences in the final output. Additionally, the discrepancies between (b) and (c) underscore the critical influence of when the conditional image is introduced. This suggests that establishing the semantic structure early is essential, though its importance diminishes significantly during the later stages of detail generation. These results show that there are a lot of redundant calculations in controllable generative models, and not every step requires the introduction of conditional calculations.

To reduce the computational complexity of the model, recent methods have introduced caching mechanisms. For example, Faster Diffusion (Li et al. 2023a) employs temporal-aware optimization, leveraging the gradual evolution of encoder features compared to rapidly changing decoder fea-

*Corresponding author.

tures across timesteps. It selectively skips encoder computations and reuses cached decoder outputs. DeepCache (Ma, Fang, and Wang 2024) targets structural redundancy in denoising paths, exploiting deep-layer feature similarity to bypass recomputation through intelligent caching while preserving shallow-layer updates. T-GATE (Zhang et al. 2024b) accelerates attention mechanisms by caching stabilized cross-attention outputs after convergence, minimizing redundant calculations. However, directly implementing caching in this model may compromise control fidelity, as the control module interacts closely with the generative module, requiring intensive computations. Research (Karras et al. 2022) indicates that the generative module's computational demands surge in later stages, focusing on refining fine-grained details during the final denoising steps. To address this, we propose **H**ybrid-**g**rained **C**aches (**HGC**), which combines block-level caching for stage-wide efficiency with prompt-level caching for granular optimization.

First, we introduce a **block-level caching** strategy to speed up the generation process of the controllable generative model. Specifically, we compute the similarity between block outputs to pinpoint the moment when the structure changes the most, and cache the state at this critical moment to preserve the intermediate representation of the former stage (where the conditional image has the greatest impact). Reusing these cached blocks in subsequent steps can eliminate redundant computations in stages that focus on detail refinement rather than structure redefinition. It reduces computational overhead by avoiding repeated processing of the conditional image in all stages, while ensuring that the final output is consistent with the expected guidance by maintaining early semantic integrity. These operations are used in both the control module and the generative module, but in the generative module, we introduce two control ratios to adjust the cache strength between different modules and denoising stages. Although block-level caching reduces redundant computation by reusing early structural representations, it still requires processing the entire block computation (including the cross-attention layer and multi-layer perceptron (MLP)). Therefore, to further reduce the computational cost, we propose a **prompt-level caching** strategy that achieves intra-block performance savings by focusing on the cross-attention mechanism. By analyzing the similarities between cross-attention outputs of adjacent blocks, we identify moments of minimal change and cache these states at key points, enabling the network to skip repeated computations without sacrificing image quality. This cue-level cache complements the block-level cache by storing and reusing cross-attention states, reducing the need to recompute these outputs and their associated multi-layer perceptrons (MLPs), and further reducing computational overhead while maintaining early semantic fidelity.

We define block-level caching as a coarse-grained cache, and correspondingly, prompt-level caching is defined as a fine-grained cache. These caches, with varying granularities, seamlessly integrate into each stage of the controllable generation process to enhance computational efficiency. Our contributions are threefold:

- We propose a coarse-grained cache that adjusts intervals based on feature similarity across steps, reducing overhead in structural synthesis while keeping quality.
- We design a fine-grained cache that reuses stable cross-attention maps across denoising steps using temporal similarity to skip redundant steps and keep precision.
- Our **HGC** reduces MACs by approximately 63% across diverse datasets and tasks, achieving comparable quality while significantly boosting model speed.

2 Related Work

In this section, we briefly introduce the controllable generative models and the related acceleration methods.

2.1 Controllable Generative Models

Ho et al. (Ho, Jain, and Abbeel 2020) introduced the Denoising Diffusion Probabilistic Model (DDPM), laying a robust foundation for iterative denoising in diffusion-based generative models. Dhariwal and Nichol (Dhariwal and Nichol 2021) added classifier guidance to direct sampling toward specific attributes, while Ho et al. (Ho and Salimans 2022) advanced this with classifier-free guidance, embedding conditions directly into the model for enhanced flexibility and sample quality. For spatially conditioned synthesis, Zhang et al. (Zhang, Rao, and Agrawala 2023) extended text-to-image diffusion models with ControlNet, enabling precise geometric control, later improved by ControlNet++ (Li et al. 2024) through consistency feedback to minimize artifacts. In text-to-image generation, latent diffusion models like Stable Diffusion (Rombach et al. 2022) and Imagen (Saharia et al. 2022) achieved high-fidelity synthesis by embedding text prompts in compact latent spaces, balancing semantic control and efficiency. Video Diffusion Models (VDM) (Ho et al. 2022) adapted these methods for temporally consistent video synthesis, supporting text or frame-based conditioning for dynamic storytelling. In 3D generation, Poole et al. (Poole et al. 2022) proposed Score Distillation Sampling (SDS), using 2D diffusion priors to optimize 3D assets like neural radiance fields. Domain-specific applications, such as MedSegDiff (Wu et al. 2024), customized diffusion for medical image segmentation with anatomical constraints.

2.2 Model Acceleration

Common methods for accelerating in diffusion models are grouped into four categories: **pruning**, **quantization**, **efficient sampling**, and **caching**. **Pruning** simplifies models by removing less essential components while maintaining performance. It is divided into unstructured pruning (Dong, Chen, and Pan 2017; Zhu et al. 2025c), which masks individual parameters, and structured pruning (Liu et al. 2021; Wang et al. 2020), which removes larger structures such as layers or filters. For example, DiP-GO (Zhu et al. 2025a) uses a plugin pruner to optimize constraints for better synthesis quality, and DaTo (Zhang et al. 2024a) dynamically prunes low-variance tokens to enhance temporal feature dynamics in self-attention. **Quantization** compresses models by representing weights and activations in lower-bit formats. Key approaches include Quantization-Aware Training (Bhalgat et al. 2020; Zhu et al. 2024b), which integrates

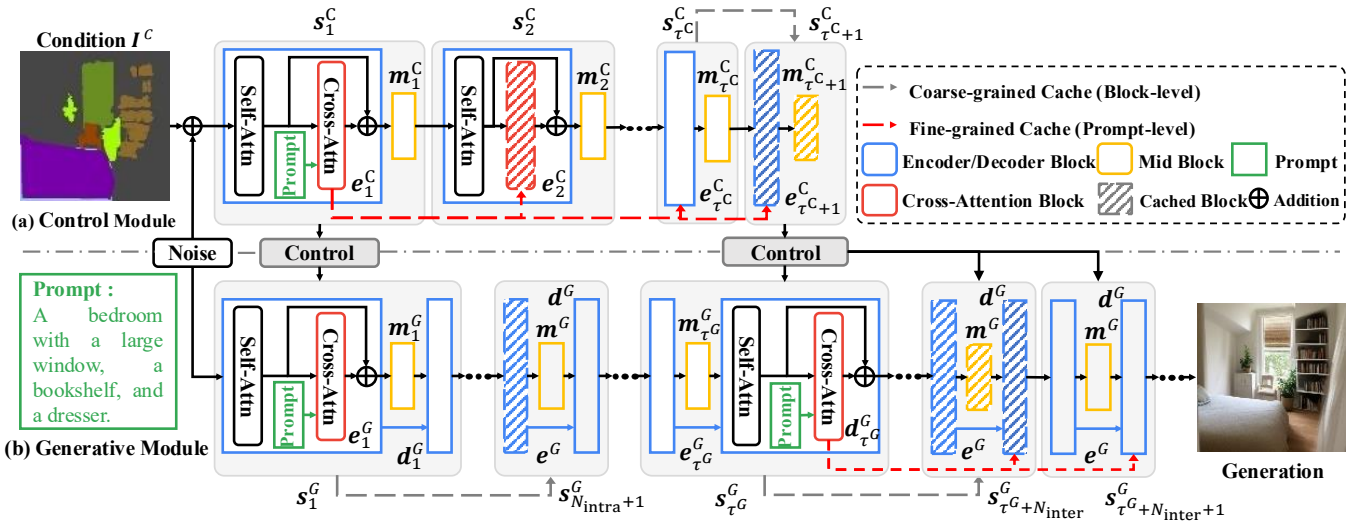


Figure 2: Controllable Generation with our **H**ybrid-**g**rained **C**aches (**HGC**), where the coarse-grained cache performs either full or partial caching of blocks across different steps. The fine-grained cache is governed by the gate step, which activates the cache of cross-attention maps at their respective steps.

quantization during training, and Post-Training Quantization (PTQ) (Li et al. 2021; Zhu et al. 2025b), which quantizes pre-trained models without retraining. For example, Q-Diffusion (Li et al. 2023b) improves calibration with time step-aware sampling and a noise-prediction quantizer. **Efficient sampling** encompasses two paradigms: retraining-based optimization and sampling-algorithm enhancement. Retraining methods, such as knowledge distillation (Salimans and Ho 2022; Zhu et al. 2024c; Wang et al. 2018), modify architectures for fewer-step generation but require additional training resources. Training-free methods refine sampling dynamics. For instance, DDIM (Song, Meng, and Ermon 2020) accelerates inference with non-Markovian deterministic trajectories, while DPM-Solver (Lu et al. 2022) uses high-order solvers to reduce steps theoretically. Consistency models (Song et al. 2023; Zhu et al. 2024a; Guo et al. 2019) enable single-step sampling via learned transition mappings. **Caching** accelerates generation by reusing computations. It falls into two categories: rule-based methods (Selvaraju et al. 2024; Ma, Fang, and Wang 2024; Qiu, Lu, and Wang 2025), which reuse or skip specific steps/blocks based on sampling-induced feature variations, and training-based methods (Ma et al. 2024), where models learn to bypass non-critical modules. These are widely adopted in DiT architectures(Liu et al. 2025; Zheng et al. 2025; Zou et al. 2024) due to their consistent data dimensionality during sampling. U-Net-based methods like DeepCache (Ma, Fang, and Wang 2024) and Faster Diffusion (Li et al. 2023a) achieve low-loss computation skipping via feature reuse. Adapting cache to DiTs is challenging, but Fora (Selvaraju et al. 2024) reuses attention or MLP layer outputs across denoising steps, and Δ-DiT (Chen et al. 2024) targets specific blocks. EOC (Qiu et al. 2025b) and GOC (Qiu et al. 2025a) proposes method that determines when caching steps require optimization and leverages gradients in the caching process to reduce errors in future caching.

3 Method

In this section, we first introduce the preliminaries of controllable generative models, and then describe our acceleration strategy **H**ybrid-**G**rained **C**ache (**HGC**) in detail.

3.1 Preliminaries

Controllable generative models comprise two components: the control module and the generative module, where the control module processes conditional information and feeds it to the generative module for targeted content generation.

Control Module acts as the conditional interface, directing the generation process by processing external guidance signals, such as edge maps or segmentation masks. Given a conditional image ($\boldsymbol{I}^{\mathrm{C}}$), noise ($\boldsymbol{z}$), and prompt text ($\boldsymbol{P}$), it uses T^{C} calculation steps ($\boldsymbol{S}^{\mathrm{C}} = \{\boldsymbol{s}_i^{\mathrm{C}}\}_{i=1}^{T^{\mathrm{C}}}$) to encode these content, where each step contains an encoder and mid-block calculation layer ($\boldsymbol{s}_i^{\mathrm{C}} = [\boldsymbol{e}_i^{\mathrm{C}}, \boldsymbol{m}_i^{\mathrm{C}}]$). Thus, the control output ($\boldsymbol{O}_i^{\mathrm{C}}$) of each step can be formulated as:

$$\begin{aligned}\boldsymbol{O}_i^{\mathrm{C}} &= [\boldsymbol{o}_{\boldsymbol{e},i}^{\mathrm{C}}, \boldsymbol{o}_{\boldsymbol{m},i}^{\mathrm{C}}] \\ &= [\boldsymbol{e}_i^{\mathrm{C}}(\boldsymbol{I}^{\mathrm{C}} + \boldsymbol{z}; \boldsymbol{P}), \boldsymbol{m}_i^{\mathrm{C}}(\boldsymbol{e}_i^{\mathrm{C}}(\boldsymbol{I}^{\mathrm{C}} + \boldsymbol{z}; \boldsymbol{P}))].\end{aligned} \tag{1}$$

Finally, the outputs are used to guide the generation process.

Generative Module, typically a pre-trained model, handles the core synthesis process. It consists of T^{G} generation steps ($\boldsymbol{S}^{\mathrm{G}} = \{\boldsymbol{s}_i^{\mathrm{G}}\}_{i=1}^{T^{\mathrm{G}}}$), and each generation step contains encoder block $\boldsymbol{e}^{\mathrm{G}}$, mid-block $\boldsymbol{m}^{\mathrm{G}}$, and decoder block $\boldsymbol{d}^{\mathrm{G}}$. Therefore, the generation process can be defined as:

$$\boldsymbol{I}_{T^{\mathrm{G}}}^{\mathrm{G}} = p(\boldsymbol{I}_{T^{\mathrm{G}}}^{\mathrm{G}} | \boldsymbol{I}_{T^{\mathrm{G}}-1}^{\mathrm{G}}) = \boldsymbol{s}_{T^{\mathrm{G}}}^{\mathrm{G}}(\boldsymbol{s}_{T^{\mathrm{G}}-1}^{\mathrm{G}}(\boldsymbol{z}; \boldsymbol{P})), \tag{2}$$

where $\boldsymbol{s}^{\mathrm{G}}(\boldsymbol{z}; \boldsymbol{P}) = \boldsymbol{d}^{\mathrm{G}}([\boldsymbol{e}^{\mathrm{G}}; \boldsymbol{m}^{\mathrm{G}}(\boldsymbol{e}^{\mathrm{G}}(\boldsymbol{z}; \boldsymbol{P}))])$, p is a prediction function and $[;]$ denotes the concatenation of features

along the channel dimension. As for controllable generation, the control information generated by the control module is used in each step of the calculation process. Specifically, $\boldsymbol{e}_i^{\mathrm{C}}$ and $\boldsymbol{m}_i^{\mathrm{C}}$ in $\boldsymbol{o}_i^{\mathrm{C}}$ are loaded into the existing $\boldsymbol{s}_i^{\mathrm{G}}$ calculation:

$$\boldsymbol{s}_i^{\mathrm{G}}(\boldsymbol{z};\boldsymbol{P}) = \boldsymbol{d}_i^{\mathrm{G}}([\boldsymbol{e}^{\mathrm{G}}+\boldsymbol{e}_i^{\mathrm{C}};(\boldsymbol{m}_i^{\mathrm{G}}+\boldsymbol{m}_i^{\mathrm{C}})(\boldsymbol{e}_i^{\mathrm{G}}(\boldsymbol{z};\boldsymbol{P}))]). \quad (3)$$

Our **H**ybrid-**g**rained **C**aches (**HGC**) is built on the controllable generative model. The pipeline is shown in Figure 2. The Block-Level Cache focuses on the former steps of the control module, where the features stabilize and become less sensitive to changes. This cache strategy eliminates redundant computations, improving the overall efficiency of the model. In the generative module, the Block-Level Cache improves computational efficiency by controlling the frequency of updates to the model's blocks. The process is divided into two stages, with two parameters λ_{intra} and λ_{inter} governing the cache density at different levels. On the other hand, the Prompt-Level Cache is introduced to address redundancy in prompt signal processing. By reusing cross-attention features, the model reduces computational costs while preserving the fidelity of prompt-specific information throughout the generation process.

3.2 Block-Level (Coarse-grained) Cache

Control Module primarily focuses on the former half of the steps, and the calculation of the latter half has little effect on the overall generated results. Therefore, we directly discard computations in the latter steps and focus solely on the characteristics of features in the former steps.

Denoting the output of all steps of the control module as $\mathcal{O}^{\mathrm{C}} = \{\boldsymbol{o}_i^{\mathrm{C}}\}_{i=1}^{T^{\mathrm{C}}}$, we calculate the similarity between the outputs before $\lfloor T^{\mathrm{C}}/2 \rfloor$ steps by cosine similarity:

$$\begin{aligned} a_{i,j}^{\mathrm{C}} &= \mathrm{Cosine}(\boldsymbol{o}_i^{\mathrm{C}}, \boldsymbol{o}_j^{\mathrm{C}}), \\ &= \frac{1}{2}\left(\frac{\langle \boldsymbol{o}_{\boldsymbol{e},i}^{\mathrm{C}}, \boldsymbol{o}_{\boldsymbol{e},j}^{\mathrm{C}} \rangle}{\|\boldsymbol{o}_{\boldsymbol{e},i}^{\mathrm{C}}\| \|\boldsymbol{o}_{\boldsymbol{e},i}^{\mathrm{C}}\|} + \frac{\langle \boldsymbol{o}_{\boldsymbol{m},i}^{\mathrm{C}}, \boldsymbol{o}_{\boldsymbol{m},j}^{\mathrm{C}} \rangle}{\|\boldsymbol{o}_{\boldsymbol{m},i}^{\mathrm{C}}\| \|\boldsymbol{o}_{\boldsymbol{m},i}^{\mathrm{C}}\|}\right), \end{aligned} \quad (4)$$

where $i = 1, \cdots, \lfloor T^{\mathrm{C}}/2 \rfloor$ and $j = i+1, \cdots, \lfloor T^{\mathrm{C}}/2 \rfloor$. $\langle \cdot, \cdot \rangle$ is the inner product between two vectors.

Then we find the most influential step τ^{C} based on the similarity score between each step and all the subsequent steps. Specifically, denoting the similarity score set of the i-th step and subsequent steps as:

$$\boldsymbol{A}_i^C = a_{i,i+1}^{\mathrm{C}}, a_{i,i+2}^{\mathrm{C}}, \cdots, a_{i,\lfloor T^{\mathrm{C}}/2 \rfloor}^{\mathrm{C}}. \quad (5)$$

Then, we determine whether all similarity scores in this step are greater than the threshold θ: $a_i^C > \theta$, where $\forall a_i^C \in \boldsymbol{A}_i^C$. If the step satisfies all the conditions, we define this i-th step as the cached τ^{C} step and use it in the control module. In other words, we cache the control output at step τ^{C} and replace the control outputs for all subsequent steps with the cached output $\boldsymbol{o}_{\tau^{\mathrm{C}}}^{\mathrm{C}}$:

$$\boldsymbol{o}_j^{\mathrm{C}} \leftarrow \boldsymbol{o}_{\tau^{\mathrm{C}}}^{\mathrm{C}}, j = \tau^{\mathrm{C}}+1, \cdots, \lfloor T^{\mathrm{C}}/2 \rfloor. \quad (6)$$

If the i-th step does not satisfy the above conditions, we continue to judge the next $(i+1)$-th step until the $\lfloor T^{\mathrm{C}}/2 \rfloor$ step. This strategy helps us optimize the generation process by preventing redundant calculations in the control module.

Generative Module typically uses an interval-based cache strategy (Ma, Fang, and Wang 2024). It updates the cache every N time steps:

$$\{\boldsymbol{s}_{i+1}^{\mathrm{G}}, \cdots, \boldsymbol{s}_{i+N-1}^{\mathrm{G}}\} \leftarrow \boldsymbol{s}_i^{\mathrm{G}}, i = 1, 1+N, 1+2N, \cdots. \quad (7)$$

However, this full-scale brute-force caching will be unstable during the detailed calculation process. Therefore, we divide the generative module into the former and the latter parts, just like similar operations in the control module, and design two balance parameters (λ_{intra} and λ_{inter}) to control the caching strength of different steps. Specifically, λ_{intra} is used to control the relative cache density between different blocks throughout the former steps. Thus, the cache interval for the mid-blocks $\boldsymbol{m}^{\mathrm{G}}$ and decoder blocks $\boldsymbol{d}^{\mathrm{G}}$ in the generative module is defined as:

$$N_{\mathrm{intra}} = N \cdot \lambda_{\mathrm{intra}}, \quad (8)$$

where $\lambda_{\mathrm{intra}} \in (0, 1]$ controls the compression ratio. As λ_{intra} increases, the frequency of updates for the control module output into the generative module decreases. Conversely, a smaller λ_{intra} results in more frequent updates. Then, the cache used in the former steps can be formulated as two parts, one of the full cache of the encoder blocks:

$$\{\boldsymbol{e}_{i+1}^{\mathrm{G}}, \cdots, \boldsymbol{e}_{i+N-1}^{\mathrm{G}}\} \leftarrow \boldsymbol{e}_i^{\mathrm{G}}, i = 1, 1+N, 1+2N, \cdots. \quad (9)$$

And the partial cache of the mid-blocks and decoder blocks:

$$\begin{aligned} \{\boldsymbol{m}_{i+1}^{\mathrm{G}}, \cdots, \boldsymbol{m}_{i+N_{\mathrm{intra}}-1}^{\mathrm{G}}\} \leftarrow \boldsymbol{m}_i^{\mathrm{G}}, \\ \{\boldsymbol{d}_{i+1}^{\mathrm{G}}, \cdots, \boldsymbol{d}_{i+N_{\mathrm{intra}}-1}^{\mathrm{G}}\} \leftarrow \boldsymbol{d}_i^{\mathrm{G}}, \end{aligned} \quad (10)$$

where $i = 1, 1+N_{\mathrm{intra}}, 1+2N_{\mathrm{intra}}, \cdots$.

As for the latter steps, we introduce a scalable cache interval via a hyperparameter λ_{inter} as the generative module's high computational demands for refining fine-grained details limit its caching capacity. Thus, the cache operation used in the latter steps can be rewritten as:

$$\{\boldsymbol{s}_{i+1}^{\mathrm{G}}, \cdots, \boldsymbol{s}_{i+N_{\mathrm{inter}}-1}^{\mathrm{G}}\} \leftarrow \boldsymbol{s}_i^{\mathrm{G}}, \quad (11)$$

where $N_{\mathrm{inter}} = N \cdot \lambda_{\mathrm{inter}}$, and $i = \lfloor T^{\mathrm{C}}/2 \rfloor, \lfloor T^{\mathrm{C}}/2 \rfloor + N_{\mathrm{inter}}, \lfloor T^{\mathrm{C}}/2 \rfloor + 2N_{\mathrm{inter}}, \cdots$. Similar to the λ_{intra}, $\lambda_{\mathrm{inter}} \in (0, 1]$ adjusts the cache interval ratio based on the module's computational requirements.

3.3 Prompt-Level (Fine-grained) Cache

Control Module consists of the encoder block and mid block. These blocks have been proven to change minimally during the calculation (Li et al. 2023a). Thus, we opt to cache the cross-attention features at the first computation step of the control module. Denoting the cross-attention features of the first computation step in $\boldsymbol{s}_1^{\mathrm{C}}$ as $\boldsymbol{f}_1^{\mathrm{C}}$, we reuse the $\boldsymbol{f}_1^{\mathrm{C}}$ in the subsequent steps:

$$\boldsymbol{f}_i^{\mathrm{C}} \leftarrow \boldsymbol{f}_1^{\mathrm{C}}, i = 1, \cdots, T^{\mathrm{G}}. \quad (12)$$

Generative Module uses the cross-attention guiding the model to generate images aligned with the textual input. However, as described in T-GATE (Zhang et al. 2024b): the outputs of cross-attention tend to converge after a few inference steps, leading to redundancy in its computational use

	Dataset	Method	ControlNet					ControlNet++				
			FID↓	CS↑	MACs↓	$\mathcal{L}$.↓	$\mathcal{S}$.	FID↓	CS↑	MACs↓	$\mathcal{L}$.↓	$\mathcal{S}$.
Segmentation	ADE20k	NoCache	38.13	**31.10**	18.22T	5.24s	1.00×	30.51	**31.96**	18.22T	5.24s	1.00×
		DeepCache	38.74	30.98	9.30T	3.03s	1.73×	29.91	31.84	9.30T	3.03s	1.73×
		T-GATE	**37.92**	30.79	13.51T	4.27s	1.23×	**28.12**	31.77	13.51T	4.27s	1.23×
		HGC	39.44	30.26	**6.70T**	**2.60s**	**2.02×**	28.93	31.57	**6.70T**	**2.60s**	**2.02×**
	COCOStuff	NoCache	22.50	**31.85**	18.22T	5.24s	1.00×	19.99	**32.37**	18.22T	5.24s	1.00×
		DeepCache	22.98	31.80	9.30T	3.03s	1.73×	20.46	32.36	9.30T	3.03s	1.73×
		T-GATE	**22.15**	31.40	13.51T	4.27s	1.23×	**19.03**	31.99	13.51T	4.27s	1.23×
		HGC	22.93	30.82	**6.70T**	**2.60s**	**2.02×**	20.29	31.52	**6.70T**	**2.60s**	**2.02×**
Edge	Multi-20M	NoCache	21.22	**32.13**	18.22T	5.24s	1.00×	20.14	31.75	18.22T	5.24s	1.00×
		DeepCache	20.69	32.04	9.30T	3.03s	1.73×	19.38	**32.04**	9.30T	3.03s	1.73×
		T-GATE	**20.43**	31.44	13.51T	4.27s	1.23×	**19.11**	31.06	13.51T	4.27s	1.23×
		HGC	20.81	30.97	**6.70T**	**2.60s**	**2.02×**	19.15	30.63	**6.70T**	**2.60s**	**2.02×**
Depth	Multi-20M	NoCache	19.84	**32.32**	18.22T	5.24s	1.00×	15.94	**32.33**	18.22T	5.24s	1.00×
		DeepCache	20.58	32.08	9.30T	3.03s	1.73×	16.23	32.29	9.30T	3.03s	1.73×
		T-GATE	**19.54**	31.22	13.51T	4.27s	1.23×	**14.39**	31.51	13.51T	4.27s	1.23×
		HGC	21.87	30.55	**6.70T**	**2.60s**	**2.02×**	17.26	30.90	**6.70T**	**2.60s**	**2.02×**

Table 1: Main results of three conditions on ControlNet, ControlNet++, where bold denotes the best performance, and underline indicates the second-best.

during the latter steps of image generation. Thus, we introduce a gate step $\tau^{\mathrm{G}} = \lfloor T^{\mathrm{G}}/2 \rfloor$. Here, we split the data fed into the model into two parts, one for prompt calculation and the other for non-prompt calculation. We fuse the results of these two parts in the τ^{G} step. Denoting the results of the cross-attention combined with the prompt and without the prompt as $\boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{p},\tau^{\mathrm{G}}}$ and $\boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{np},\tau^{\mathrm{G}}}$, respectively, the fusion operation can be formulated as:

$$\boldsymbol{f}^{\mathrm{G}}_{\mathrm{fuse}} = (\boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{p},\tau^{\mathrm{G}}} + \boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{np},\tau^{\mathrm{G}}})/2. \tag{13}$$

For subsequent steps $i = \tau^{\mathrm{G}}, \tau^{\mathrm{G}}+1, \cdots, T^{\mathrm{G}}$, we reuse $\boldsymbol{f}^{\mathrm{G}}_{\mathrm{fuse}}$ to reduce the calculation cost:

$$\boldsymbol{f}^{\mathrm{G}}_{i} \leftarrow \boldsymbol{f}^{\mathrm{G}}_{\mathrm{fuse}}. \tag{14}$$

To further enhance efficiency, we halve the batch size in Control and Generative modules during the fusion phase, significantly reducing memory overhead without compromising generation quality. Specifically, During inference with batch size n, the actual processing requires $2n$ computations (n for $\boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{p},\tau^{\mathrm{G}}}$ and n for $\boldsymbol{f}^{\tau^{\mathrm{G}}}_{\mathrm{np},\tau^{\mathrm{G}}}$). At step τ^{G}, we cache both attention maps and subsequently replace the dual computations with a fused version $\boldsymbol{f}^{\mathrm{G}}_{\mathrm{fuse}}$, this fusion reduces the computational batch size from $2n$ back to n, halving the batch size. Notably, for subsequent steps $i = \tau^{\mathrm{G}}, \tau^{\mathrm{G}}+1, \cdots, T^{\mathrm{G}}$, although the cache interval decreases to accommodate fine-grained detail refinement, the actual computational overhead remains minimal due to the batch size halving operation.

4 Experiment

In this section, we conduct experiments to evaluate the performance of our proposed hybrid-grained cache (**HGC**). We intended to address the following research questions (**RQ**):

RQ1: Is Hybrid-grained Caches (HGC) effective?
RQ2: What hyperparameter should be selected?
RQ3: Can HGC be applied to other models?

4.1 Experiment Settings

Models and Datasets. We utilize two baseline models in our experiments: ControlNet (Zhang, Rao, and Agrawala 2023) and ControlNet++ (Li et al. 2024). These models have similar architectures, including control modules and generation modules. We tested our method on several datasets: ADE20K (Zhou et al. 2017) and COCOStuff (Caesar, Uijlings, and Ferrari 2018) for generation with segmentation mask conditions, and the MultiGen-20M dataset from UniControl (Qin et al. 2023), a subset of LAION-Aesthetics (Schuhmann et al. 2022), for generation of canny edge map and depth map conditions. For datasets lacking text captions, such as ADE20K, we use MiniGPT4 (Zhu et al. 2023) to generate one sentence image caption with the prompt: "Please briefly describe this image in one sentence." Then, we use this caption as the prompt to evaluate our method.

Cache Settings. In our method, we set the number of steps in the control module T^{C} and the generative module T^{G} to 20. For Block-level Cache, we set the similarity threshold θ to 0.9, which results in a cached step $\tau^{\mathrm{C}} = 6$. In the generative module, we use a base cache interval $N = 5$, with $\lambda_{\mathrm{intra}} = 0.4$ and $\lambda_{\mathrm{inter}} = 0.6$ to control the caching behavior. For Prompt-level Cache, we get the gate step $\tau^{\mathrm{G}} = 10$, which determines when the model switches to caching the fused attention maps. Additionally, for DeepCache (Ma, Fang, and Wang 2024), we set its cache interval to 5, and for T-GATE (Zhang et al. 2024b), we set the gate step to 5. The complete set of hyperparameters and implementation details is available in the source code.

Method	MACs↓	FID↓	CS↑
NoCache	18.22T	**19.99**	**32.37**
DeepCache	8.03T	24.84	31.68
T-GATE	10.23T	17.32	30.88
HGC	**6.70T**	20.29	31.52

Table 2: Comparison of different acceleration methods.

θ	τ^{C}	FID↓	CS↑	MACs↓
0	1	30.92	31.26	**5.53T**
0.9	6	28.93	**31.57**	6.70T
1.0	10	**28.72**	31.56	7.63T

Table 3: Comparison of different threshold θ.

λ_{intra}	FID↓	CS↑	MACs↓
0	**28.86**	**31.68**	8.418T
0.4	28.93	31.57	6.70T
1.0	29.04	31.49	**5.826T**

Table 4: Comparison of different parameter λ_{intra}.

λ_{inter}	FID↓	CS↑	MACs↓
0	**28.44**	**31.64**	8.61T
0.6	28.93	31.57	6.70T
1.0	30.49	31.40	**6.06T**

Table 5: Comparison of different parameter λ_{inter}.

Evaluation. The images are generated using DDIM (Song, Meng, and Ermon 2020) with a predefined 20 inference steps with guiding scale (7.5) and resized to 512 × 512 resolution. We employed metrics such as Fr'echet Inception Distance (**FID**) and CLIP Score (**CS**) to measure the quality of generated images. To evaluate the efficiency, we use Calflops to count Multiple-Accumulate Operations (**MACs**). Furthermore, we measure the end-to-end latency ($\mathcal{L}$.) and Speedup ($\mathcal{S}$.) for processing a batch of 4 samples on a system powered by one NVIDIA RTX 3090 GPU.

4.2 Main Results (RQ1)

We employed two generative models with three distinct conditions (segmentation, edge, and depth maps) for our experiments. Results are presented in Table 1. We can find that our HGC helps the model achieve a speed improvement while maintaining comparable generation quality. Specifically, compared to the 1.73x acceleration of DeepCache, HGC achieves a 2.02x acceleration (16.8% ↑). As for quality analysis, for the metric FID, the gap between HGC and DeepCache is controlled within 2% for most tasks, with only the depth task showing a gap of around 6%.

Meanwhile, we aligned the acceleration ratios of T-GATE, DeepCache, and HGC to compare the quality of images generated using the COCO-Stuff-Seg dataset with ControlNet. We ensured that the computational requirements (MACs) for T-GATE, DeepCache, and HGC were nearly identical. As shown in Table 2, HGC outperforms other methods in generation quality under similar computational budgets. Although T-GATE achieves the best FID score, its CLIP Score is lower, indicating reduced alignment between generated images and their prompts. Notably, we set T-GATE's gate step to 3, exceeding the recommended range in the original T-GATE paper, suggesting HGC offers greater acceleration potential than both T-GATE and DeepCache.

4.3 Ablation Studies (RQ2)

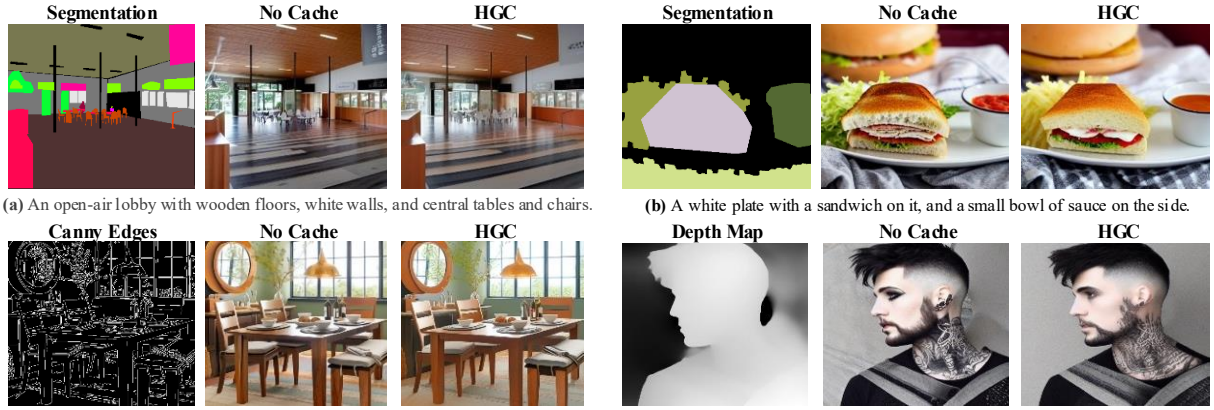
In this section, we conduct experiments on ADE20k with segmentation mask and analyze the impact of various hyperparameters in our method. Specifically, we vary the target parameters while keeping all other parameters constant.

Selection of θ. As shown in Table 3, we evaluate threshold values $\theta \in \{0, 0.9, 1.0\}$, which yield corresponding caching steps $\tau^{\mathrm{C}} = \{0, 0.4, 1.0\}$. When θ is set too low, the quality of the generated images significantly decreases, as evidenced by the FID increasing from 28.93 to 30.92, a increase of 6.88%. This suggests that when threshold is too small, the model reaches the caching step too early, leading to less precise feature processing and a noticeable decline in image quality. On the other hand, setting θ too high results in computational redundancy. For example, increasing θ from 0.9 to 1.0 increases the computational cost (MACs) by 1T, but the image quality remains almost unchanged, with FID only improving slightly from 28.93 to 28.72. This indicates that excessively large thresholds lead to unnecessary computations, with diminishing returns in image quality.

Selection of λ_{intra}. As shown in Table 4, we evaluate three distinct values of $\lambda_{\mathrm{intra}} \in \{0, 0.4, 1.0\}$. Quantitative analysis reveals a consistent performance degradation trend: the FID score increases from 28.86 to 29.04 while the CLIP Score decreases from 31.68 to 31.49 as λ_{intra} varies from 0 to 1.0. This inverse correlation between λ_{intra} and generation quality suggests that excessive dependency on cached features (higher λ_{intra}) compromises the model's ability to maintain optimal image fidelity.

Selection of λ_{inter}. As shown in Table 5, we evaluate three values 0, 0.6 and 1.0 for the ratio λ_{inter}. When λ_{inter} increases, the image quality decreases. Specifically, when λ_{inter} increases from 0 to 0.6, the change in quality is minimal, with the FID increasing slightly from 28.44 to 28.93 and the CLIP Score decreasing from 31.64 to 31.57. However, when λ_{inter} increases from 0.6 to 1.0, there is a significant drop in image quality, with FID increasing to 30.49 and CLIP Score decreasing further to 31.40. This indicates that higher values of λ_{inter} lead to a more severe decline in generation quality, while speed only improves slightly.

Different Caching Components. As shown in Table 6, all four independent components successfully reduced computational costs while maintaining generation quality comparable to the Baseline. From an internal comparison perspective, the acceleration achieved for the generative module at the same granularity level was more substantial than that for the control module, which aligns with the fact that the gen-



(a) An open-air lobby with wooden floors, white walls, and central tables and chairs.

(b) A white plate with a sandwich on it, and a small bowl of sauce on the side.

(c) "Basque Honey 65"" Dining Table"

(d) The Best Short Goth Hairstyles For Guys Hair Pinterest

Figure 3: The visualization of the generation with or without HGC: (a) and (b) generation with segmentation condition. (c) generation with edge map. (d) generation with the depth map.

Method	MACs↓	FID↓	CS↑
NoCache	18.22T	19.99	32.37
GM Block	12.09T	19.75	**32.40**
CM Block	14.95T	20.86	32.22
GM Prompt	13.51T	**19.14**	31.99
CM Prompt	15.77T	20.27	32.19
HGC	**6.70T**	19.15	31.52

Table 6: Comparison of cache strategy performance across different cache components, where GM denotes the generative module and CM refers to the control module.

Method	MACs↓	FID↓
NoCache	48.29T	11.03
HGC	28.88T	11.70

Table 7: Comparison of video generation performance between baseline and HGC approaches.

erative module accounts for the majority of computations in Controllable Generation. From a granularity perspective, coarse-grained approaches demonstrated greater acceleration potential compared to fine-grained methods.

4.4 Exploration on Video Generation (RQ3)

We extend the evaluation of our HGC method to video generation tasks by integrating it with the CTRL-Adapter framework (Lin et al. 2024), using DAVIS 2017 dataset (Pont-Tuset et al. 2017). As shown in Table 7, our approach achieves a significant 40% reduction in computational cost (from 48.29T to 28.88T MACs for 14-frame generation) while maintaining reasonable output quality, with the FID score increasing from 11.03 to 11.70. This acceleration demonstrates the effectiveness of our caching strategy for video generation tasks, where the trade-off between computational efficiency and visual fidelity remains within acceptable limits. Results suggest that our method can be successfully adapted to sequential generation tasks while preserving its core advantages in computational reduction.

4.5 Visualization

Visualization results are shown in Figure 3, which includes: (a) ADE20K dataset with segmentation masks, (b) COCO-Stuff dataset with segmentation masks, (c) MultiGen-20M dataset with Canny edges, and (d) MultiGen-20M dataset with depth maps. Our comparative analysis under identical control images and prompts reveals that NoCache and HGC methods generate images that faithfully align with the structural and thematic requirements of the input constraints. However, close inspection highlights HGC's subtle trade-off between efficiency and micro-detail fidelity: while it retains macro-structural integrity, it exhibits reduced precision in high-frequency details such as wood grain textures, shadow gradations around televisions, and fine textural patterns in clothing. This discrepancy stems from HGC's block-level cache mechanism—reusing intermediate features across denoising steps inherently smooths out transient details accumulated through iterative refinement. Despite this, HGC achieves comparable visual coherence to NoCache in all evaluated scenarios, successfully balancing computational efficiency with perceptually acceptable quality degradation.

5 Conclusion

We propose HGC, a dual-level cache framework accelerating controllable generation through joint optimization of prompt-level and block-level cache mechanisms. Experiments show HGC achieves nearly 2× speedup across diverse control tasks while maintaining output quality. However, limitations emerge in geometrically complex scenarios, where 3–5% CLIP Score degradation occurs due to insufficient adaptability of cached cross-attention maps during rapid scene transitions. These challenges highlight directions for future work, particularly developing variance-aware adaptive scheduling and attention-guided dynamic cache invalidation strategies. Future work will explore more general acceleration techniques to reduce performance loss.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (No.U24B20180, No. 62576330, No.62472393) , National Natural Science Foundation of Anhui (No.2508085MF143) and the advanced computing resources provided by the Supercomputing Center of the USTC.

References

Bhalgat, Y.; Lee, J.; Nagel, M.; Blankevoort, T.; and Kwak, N. 2020. 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*, 696–697.

Caesar, H.; Uijlings, J.; and Ferrari, V. 2018. Coco-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1209–1218.

Chen, P.; Shen, M.; Ye, P.; Cao, J.; Tu, C.; Bouganis, C.-S.; Zhao, Y.; and Chen, T. 2024. $\Delta - DiT$: A Training-Free Acceleration Method Tailored for Diffusion Transformers. *arXiv preprint arXiv:2406.01125*.

Dhariwal, P.; and Nichol, A. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34: 8780–8794.

Dong, X.; Chen, S.; and Pan, S. J. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in neural information processing systems*, volume 30.

Guo, D.; Wang, S.; Tian, Q.; and Wang, M. 2019. Dense Temporal Convolution Network for Sign Language Translation. In *IJCAI*, volume 2, 8.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Ho, J.; and Salimans, T. 2022. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*.

Ho, J.; Salimans, T.; Gritsenko, A.; Chan, W.; Norouzi, M.; and Fleet, D. J. 2022. Video diffusion models. *Advances in Neural Information Processing Systems*, 35: 8633–8646.

Karras, T.; Aittala, M.; Aila, T.; and Laine, S. 2022. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35: 26565–26577.

Li, M.; Yang, T.; Kuang, H.; Wu, J.; Wang, Z.; Xiao, X.; and Chen, C. 2024. ControlNet++: Improving Conditional Controls with Efficient Consistency Feedback: Project Page: liming-ai. github. io/ControlNet_Plus_Plus. In *European Conference on Computer Vision*, 129–147. Springer.

Li, S.; Hu, T.; Khan, F. S.; Li, L.; Yang, S.; Wang, Y.; Cheng, M.-M.; and Yang, J. 2023a. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *CoRR*.

Li, X.; Liu, Y.; Lian, L.; Yang, H.; Dong, Z.; Kang, D.; et al. 2023b. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 17535–17545.

Li, Y.; Gong, R.; Tan, X.; Yang, Y.; Hu, P.; Zhang, Q.; et al. 2021. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*.

Lin, H.; Cho, J.; Zala, A.; and Bansal, M. 2024. Ctrl-adapter: An efficient and versatile framework for adapting diverse controls to any diffusion model. *arXiv preprint arXiv:2404.09967*.

Liu, J.; Zou, C.; Lyu, Y.; Ren, F.; Wang, S.; Li, K.; and Zhang, L. 2025. Speca: Accelerating diffusion transformers with speculative feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 10024–10033.

Liu, L.; Zhang, S.; Kuang, Z.; Zhou, A.; Xue, J.; Wang, X.; et al. 2021. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, 7021–7032. PMLR.

Lu, C.; Zhou, Y.; Bao, F.; Chen, J.; Li, C.; and Zhu, J. 2022. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, volume 35, 5775–5787.

Ma, X.; Fang, G.; Mi, M. B.; and Wang, X. 2024. Learning-to-Cache: Accelerating Diffusion Transformer via Layer Caching. *arXiv preprint arXiv:2406.01733*.

Ma, X.; Fang, G.; and Wang, X. 2024. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 15762–15772.

Pont-Tuset, J.; Perazzi, F.; Caelles, S.; Arbeláez, P.; Sorkine-Hornung, A.; and Van Gool, L. 2017. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*.

Poole, B.; Jain, A.; Barron, J. T.; and Mildenhall, B. 2022. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*.

Qin, C.; Zhang, S.; Yu, N.; Feng, Y.; Yang, X.; Zhou, Y.; Wang, H.; Niebles, J. C.; Xiong, C.; Savarese, S.; et al. 2023. Unicontrol: A unified diffusion model for controllable visual generation in the wild. *arXiv preprint arXiv:2305.11147*.

Qiu, J.; Liu, L.; Wang, S.; Lu, J.; Chen, K.; and Hao, Y. 2025a. Accelerating diffusion transformer via gradient-optimized cache. *arXiv preprint arXiv:2503.05156*.

Qiu, J.; Lu, J.; and Wang, S. 2025. Multimodal Generation with Consistency Transferring. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 504–513.

Qiu, J.; Wang, S.; Lu, J.; Liu, L.; Jiang, H.; Zhu, X.; and Hao, Y. 2025b. Accelerating diffusion transformer via error-optimized cache. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 9588–9597.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10684–10695.

Saharia, C.; Chan, W.; Saxena, S.; Li, L.; Whang, J.; Denton, E. L.; Ghasemipour, K.; Gontijo Lopes, R.; Karagol Ayan,

B.; Salimans, T.; et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35: 36479–36494.

Salimans, T.; and Ho, J. 2022. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*.

Schuhmann, C.; Beaumont, R.; Vencu, R.; Gordon, C.; Wightman, R.; Cherti, M.; Coombes, T.; Katta, A.; Mullis, C.; Wortsman, M.; et al. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35: 25278–25294.

Selvaraju, P.; Ding, T.; Chen, T.; Zharkov, I.; and Liang, L. 2024. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*.

Song, J.; Meng, C.; and Ermon, S. 2020. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*.

Song, Y.; Dhariwal, P.; Chen, M.; and Sutskever, I. 2023. Consistency models. *arXiv preprint arXiv:2303.01469*.

Tang, S.; He, J.; Guo, D.; Wei, Y.; Li, F.; and Hong, R. 2025a. Sign-idd: Iconicity disentangled diffusion for sign language production. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 7266–7274.

Tang, S.; Xue, F.; Wu, J.; Wang, S.; and Hong, R. 2025b. Gloss-driven conditional diffusion models for sign language production. *ACM Transactions on Multimedia Computing, Communications and Applications*, 21(4): 1–17.

Wang, S.; Guo, D.; Zhou, W.-g.; Zha, Z.-J.; and Wang, M. 2018. Connectionist temporal fusion for sign language translation. In *Proceedings of the 26th ACM international conference on Multimedia*, 1483–1491.

Wang, S.; Yue, J.; Liu, J.; Tian, Q.; and Wang, M. 2020. Large-scale few-shot learning via multi-modal knowledge discovery. In *European Conference on Computer Vision*, 718–734. Springer.

Wu, J.; Fu, R.; Fang, H.; Zhang, Y.; Yang, Y.; Xiong, H.; Liu, H.; and Xu, Y. 2024. Medsegdiff: Medical image segmentation with diffusion probabilistic model. In *Medical Imaging with Deep Learning*, 1623–1639. PMLR.

Zhang, E.; Xiao, B.; Tang, J.; et al. 2024a. Token Pruning for Caching Better: 9 Times Acceleration on Stable Diffusion for Free. *arXiv preprint arXiv:2501.00375*.

Zhang, L.; Rao, A.; and Agrawala, M. 2023. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, 3836–3847.

Zhang, L.; Song, P.; Dong, J.; Li, K.; and Yang, X. 2025. Enhancing Partially Relevant Video Retrieval with Robust Alignment Learning. *arXiv preprint arXiv:2509.01383*.

Zhang, W.; Liu, H.; Xie, J.; Faccio, F.; Shou, M. Z.; and Schmidhuber, J. 2024b. Cross-attention makes inference cumbersome in text-to-image diffusion models. *arXiv e-prints*, arXiv–2404.

Zheng, Z.; Wang, X.; Zou, C.; Wang, S.; and Zhang, L. 2025. Compute only 16 tokens in one timestep: Accelerating diffusion transformers with cluster-driven feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 10181–10189.

Zhou, B.; Zhao, H.; Puig, X.; Fidler, S.; Barriuso, A.; and Torralba, A. 2017. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 633–641.

Zhu, D.; Chen, J.; Shen, X.; Li, X.; and Elhoseiny, M. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

Zhu, H.; Tang, D.; Liu, J.; et al. 2025a. DiP-GO: A Diffusion Pruner via Few-step Gradient Optimization. *Advances in Neural Information Processing Systems*, 37: 92581–92604.

Zhu, X.; Wang, S.; Lu, J.; Hao, Y.; Liu, H.; and He, X. 2024a. Boosting Few-Shot Learning via Attentive Feature Regularization. In *AAAI*, 7793–7801. AAAI Press.

Zhu, X.; Wang, S.; Zhu, B.; Li, M.; Li, Y.; Fang, J.; Wang, Z.; Wang, D.; and Zhang, H. 2025b. Dynamic Multimodal Prototype Learning in Vision-Language Models. *CoRR*, abs/2507.03657.

Zhu, X.; Zhu, B.; Tan, Y.; Wang, S.; Hao, Y.; and Zhang, H. 2024b. Enhancing Zero-Shot Vision Models by Label-Free Prompt Distribution Learning and Bias Correcting. In *NeurIPS*.

Zhu, X.; Zhu, B.; Tan, Y.; Wang, S.; Hao, Y.; and Zhang, H. 2024c. Selective Vision-Language Subspace Projection for Few-shot CLIP. In *ACM Multimedia*, 3848–3857. ACM.

Zhu, X.; Zhu, B.; Wang, S.; Zhao, K.; and Zhang, H. 2025c. Enhancing CLIP Robustness via Cross-Modality Alignment. *arXiv preprint arXiv:2510.24038*.

Zou, C.; Liu, X.; Liu, T.; Huang, S.; and Zhang, L. 2024. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*.