FreqCa: Accelerating Image generation and Editing via Frequency-Aware Caching

Jiacheng Liu[1,2,3] **Peiliang Cai**[1] **Qinming Zhou**[1,4] **Yuqi Lin**[1,5]
Deyang Kong[1,7] Benhao Huang[1,6] Yupei Pan[1,7] Haowen Xu[1]
Chang Zou[1,2,7] Junshu Tang[2] Shikang Zheng[1,8] Linfeng Zhang[1*]

[1]EPIC Lab, SJTU [2]Tencent Hunyuan [3]SDU [4]THU
[5]JLU [6]CMU [7]UESTC [8]SCUT



Figure 1: Images sampled by Qwen-image with *FreqCa* with 7.14× acceleration.

Abstract

The application of diffusion transformers is suffering from their significant inference costs. Recently, feature caching has been proposed to solve this problem by reusing features from previous timesteps, thereby skipping computation in future timesteps. However, previous feature caching assumes that features in adjacent timesteps are similar or continuous, which does not always hold in all settings. To investigate this, this paper begins with an analysis from the frequency domain, which reveals that *different frequency bands in the features of diffusion models exhibit different dynamics across timesteps.* Concretely, low-frequency components, which decide the structure of images, exhibit higher *similarity* but poor continuity. In contrast, the high-frequency bands, which decode the details of images, show significant continuity but poor similarity. These interesting observations motivate us to propose **Freq**uency-aware **Ca**ching (***FreqCa***) which directly reuses features of low-frequency components based on their similarity, while using a second-order Hermite interpolator to predict the volatile high-frequency ones based on their continuity. Besides, we further propose to cache Cumulative Residual Feature (CRF) instead of the features in all the layers, which reduces the memory footprint of feature caching by **99%**. Extensive experiments on FLUX.1-dev, FLUX.1-Kontext-dev, Qwen-Image, and Qwen-Image-Edit demonstrate its effectiveness in both generation and editing. Our codes have been released in Github: https://github.com/Tammytcl/FreqCa

*Corresponding author. Email: zhanglinfeng@sjtu.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

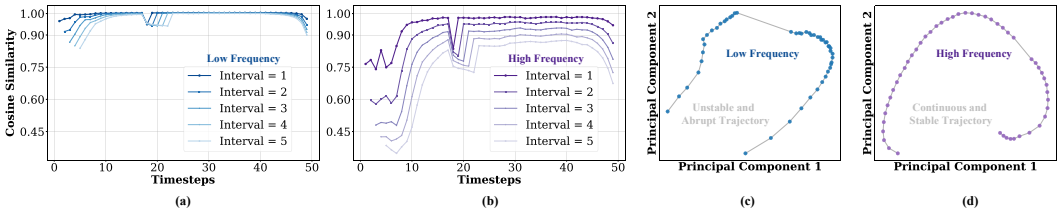


Figure 2: **Analysis from the frequency perspective. (a)-(b)**:Temporal similarity analysis using cosine similarity for low-frequency and high-frequency components across different step intervals. **(c)-(d)**: Feature trajectory visualized via Principal Component Analysis (PCA).

1 Introduction

Diffusion Models (DMs) have achieved remarkable success in generative tasks like image synthesis and video generation [1, 11]. The recent introduction of Diffusion Transformers [27] has further advanced generation quality and diversity, establishing them as the predominant architecture for large-scale visual content creation. However, diffusion transformers typically rely on a stack of heavy transformer blocks and multi-step sampling, making computational efficiency a critical bottleneck for their practical deployment. To address this, the paradigm of feature caching has emerged, which exploits the high temporal redundancy between adjacent timesteps for acceleration [4, 24, 32, 50, 51].

The debate of caching paradigms. Feature caching has gradually emerged into two different paradigms. The paradigm of "Cache-Then-Reuse" assumes that the features of DM in adjacent timesteps are highly **similar** and thus proposes to directly **reuse** the features of previous timesteps in the future timesteps [32]. In contrast, the paradigm of "Cache-Then-Forecast" assumes that features of DM are "continuous" and thus proposes to forecast features in the future timesteps based on features in the previous timesteps with non-parametric predictors such as Taylor expansion. Although the paradigm "Cache-Then-Forecast" tends to show better performance in recent works, their assumption of continuity does not always hold perfectly. For instance, Liu *et al.* demonstrates that features of FLUX are not high-order continuous, making TaylorSeer degenerate into a linear prediction method and thus suffer from quality loss [17]. Based on these findings, this paper begins with an in-depth analysis of the temporal dynamics of diffusion models.

Analysis from the frequency perspective. In classical image processing, the high-frequency and low-frequency components of images are usually considered as carrying different semantic information, which motivates us to study the dynamics of high-frequency and low-frequency components in the feature of diffusion models separately. As shown in Fig. 2(a)-(b), surprisingly, we find different frequency exhibits significantly different dynamics. Concretely, the similarity of low-frequency is higher than *0.90* at most timesteps, while the high-frequency exhibits clearly low similarity. On the other hand, as shown in Fig. 2(c)-(d), the feature trajectory of high-frequency shows perfect stability and continuity, while the feature trajectory of low-frequency is unstable and accompanied by sudden mutation, indicating that the high-frequency information can be accurately predicted while the low-frequency information fails.

Based on the above observations, this paper introduces **Freq**uency-aware Feature **Ca**ching (**FreqCa**), which aims to decouple the frequency of the features in diffusion models and treat them in different paradigms. Concretely, *FreqCa* applies any frequency decomposition (*e.g, Fourier Transformation*) to the cached features. For the low-frequency bands, we directly reuse them in the future timesteps because of their *high similarity*. For the high-frequency bands, we predict their values in the future timesteps with any sequential predictor (*Hermite polynomial predictor*) for their *good continuity*. Then, in the future timesteps, *FreqCa* reconstructs the features based on the reused low-frequency bands and predicted high-frequency bands, enabling it to skip the computation over diffusion transformers, achieving the best cooperation between the previous caching paradigms.

Memory-Efficient Feature Caching. The previous caching methods usually cache all the features from attention and FFN layers, leading to significant memory costs (*e.g,* $\geq$ 10G memory costs on FLUX in ToCa), preventing feature caching methods from real-world applications. As discussed by [39], neural networks with residual connections can be considered as an ensemble of features in all blocks, which motivates us to propose caching the Cumulative Residual Feature (CRF), *i.e.,* the cumulative features of all the residual connections from attention and FFN blocks. This trick helps us

reduce the memory footprint from caching $2 \times L$ (L indicates layer counts) features into a single feature vector, slashing cache memory usage by up to 99%. Besides, it also reduces the number of frequency (reverse) decomposition operations by $2L$ times, making them account for only $\leq 0.01\%$ latency costs during the whole diffusion process.

In summary, this contribution of this paper is as follows.

- **Frequency Heterogeneity in Diffusion Features:** We reveal that different frequency components of diffusion features follow distinct temporal dynamics. Low-frequency components show high cross-timestep similarity, while high-frequency components exhibit stronger temporal continuity, motivating frequency-specific caching strategies.
- **Frequency-Aware Feature Caching:** Based on this observation, we propose *FreqCa*, a frequency-aware caching framework that applies different strategies to different frequency bands: directly reusing stable low-frequency features and forecasting dynamic high-frequency features. This design unifies reuse-based and forecast-based caching paradigms in a single framework.
- **Efficient and Practical Acceleration:** To make frequency-aware caching practical for large-scale diffusion models, we further introduce Cumulative Residual Feature caching to reduce cache memory complexity to $\mathcal{O}(1)$. Extensive experiments show that *FreqCa* achieves 6–7$\times$ acceleration with negligible quality degradation, demonstrating strong robustness and deployment value.

2 Related Works

Diffusion models [37, 38] have become a dominant paradigm for visual generation, evolving from U-Net [29] to Diffusion Transformers [27], yet their inherently iterative sampling still incurs substantial inference cost, motivating acceleration via fewer steps or more efficient denoising.

2.1 Sampling Timestep Reduction

One primary strategy seeks to minimize the number of required sampling iterations while preserving generation quality. Seminal work like DDIM [37] introduced deterministic sampling to reduce step counts without significant fidelity loss. This concept was further refined by the DPM-Solver series, which employed high-order ODE solvers to achieve faster convergence [22, 23]. Other notable approaches include knowledge distillation, which trains a student model to emulate multiple denoising steps of a larger teacher model [26, 31], and Rectified Flow, which learns to straighten the generation path between noise and data distributions [20]. More recently, Consistency Models have enabled high-quality synthesis in a single step by directly mapping noise to clean data, circumventing the need for a sequential path [38]. Recent paradigms directly enable few-step or one-step synthesis without explicit distillation [8, 9]. However, these methods require additional training and often suffer from reduced fidelity or limited generalization under aggressive compression.

2.2 Denoising Network Acceleration

Model Compression and Dynamic-Resolution Acceleration. A common approach is to compress the denoising network via pruning [6] or quantization [33]. Complementary approaches reduce token computation through token merging [2] or dynamic-resolution processing [28, 43, 48, 49]. Although effective, these approaches can trade model capacity or spatial detail for speed, and some require fine-tuning or retraining to recover performance [14].

Feature Caching-based Acceleration. A compelling alternative is feature caching, which exploits temporal redundancy in the denoising process [16]. This paradigm was originally proposed for U-Net architectures [25], with subsequent work such as FreeU [36] incorporating frequency-aware feature modulation. It was later extended to diffusion transformers (DiTs) [27], where early methods such as FORA [32] and PAB [45] focus on structured feature reuse, while FasterCache [24] analyzes frequency discrepancies under classifier-free guidance. More recent approaches improve efficiency through selective updates and adaptive reuse [15, 50, 51], learnable feature prediction [3, 34, 52], and adaptive sampling [21]. A notable paradigm shift is the transition from "cache then reuse" to "cache then forecast", as introduced by TaylorSeer [17] and further extended with refined prediction and correction schemes [5, 7, 18, 35, 46, 47], achieving better acceleration-quality trade-offs. Despite these advances, most existing methods treat features as monolithic representations, overlooking their heterogeneous frequency characteristics.

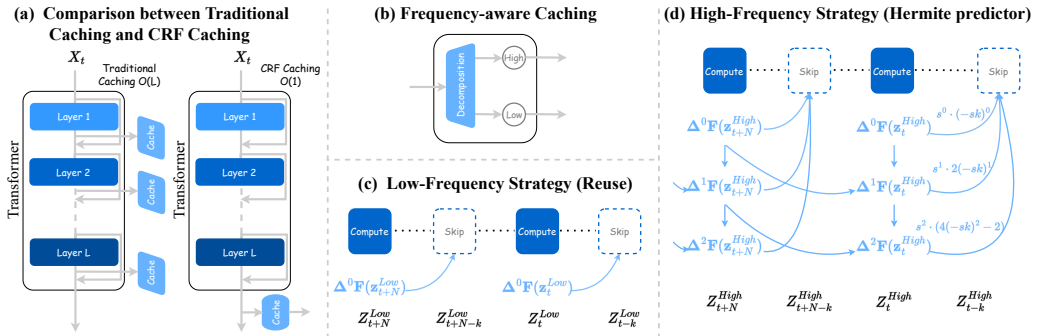


Figure 3: **Overview of the FreqCa framework.** ***(a) Cumulative Residual Feature Caching*** : Instead of caching features at every layer, we cache only the single Cumulative Residual Feature (CRF) at the end. ***(b) Frequency-aware Caching***: The cached features are separated into low- and high-frequency bands using frequency decomposition techniques. ***(c) Low-Frequency Strategy***: Low-frequency component is directly reused from the prior step. ***(d) High-Frequency Strategy***: High-frequency component is forecasted using a Hermite predictor fitted on the last two activated steps.

3 Method

3.1 Preliminary

Diffusion Transformer Architecture. The Diffusion Transformer (DiT) [27] employs a hierarchical structure $\mathcal{G} = g_1 \circ g_2 \circ \cdots \circ g_L$, where each module $g_l = \mathcal{F}^l_{\text{SA}} \circ \mathcal{F}^l_{\text{CA}} \circ \mathcal{F}^l_{\text{MLP}}$ is composed of self-attention (SA), cross-attention (CA), and multilayer perceptron (MLP) components. In DiT, these components are dynamically adapted over time to handle different noise levels during the image generation process. The input $\mathbf{x}_t = \{x_i\}_{i=1}^{H\times W}$ is represented as a sequence of tokens corresponding to image patches. Each module integrates information through residual connections of the form $\mathcal{F}(\mathbf{x}) = \mathbf{x} + \text{AdaLN} \circ f(\mathbf{x})$, where AdaLN denotes adaptive layer normalization, which stabilizes training and improves learning effectiveness.

Feature Caching in Diffusion Models Feature caching aims to accelerate diffusion inference by avoiding redundant feature computations across adjacent timesteps. A straightforward strategy is to directly reuse features computed at timestep t for earlier latents:

$$\mathcal{F}(x^l_{t-k}) \approx \mathcal{F}(x^l_t), \quad k \in \{1, \ldots, N-1\}. \tag{1}$$

While this reuse-based scheme offers an ideal N-fold speedup, it implicitly assumes temporal invariance of features, leading to rapidly accumulating errors as N increases. To mitigate this issue, recent *forecasting-based* methods recast feature caching as a temporal prediction problem. By exploiting the smooth evolution of features across diffusion timesteps, features at $(t-k)$ can be estimated via low-order finite-difference extrapolation:

$$\mathcal{F}_{\text{pred},m}(x^l_{t-k}) = \mathcal{F}(x^l_t) + \sum_{i=1}^{m} \frac{\Delta^i \mathcal{F}(x^l_t)}{i!\, N^i}(-k)^i. \tag{2}$$

where $\Delta^i \mathcal{F}(x^l_t)$ represents the i-th order discrete difference. This formulation substantially improves approximation accuracy while preserving most of the computational gains.

3.2 Frequency-Aware Cache Acceleration Framework

In this section, we introduce the **FreqCa (Frequency-aware Feature Caching)** framework, which is built upon three key components: (i) performing frequency decomposition on the feature to be cached and applying separate strategies for its low- and high-frequency components; (ii) employing a nonlinear Hermite-polynomial-based predictor for the high-frequency part to improve prediction accuracy; and (iii) identifying the Cumulative Residual Feature (CRF) as a novel, highly efficient single-tensor caching target that encapsulates the entire transformation history of the model.

Frequency-Decomposed Caching Strategy. Our differentiated caching strategy is motivated by the distinct temporal dynamics of frequency components, as illustrated in Fig. 2. We observe that

low-frequency components exhibit high similarity but low temporal continuity, making them stable across timesteps yet difficult to extrapolate. In contrast, high-frequency components show lower similarity but stronger temporal continuity, rendering them more volatile in magnitude but more predictable along diffusion trajectories. This fundamental asymmetry indicates that a one-size-fits-all caching strategy is suboptimal, and instead calls for a frequency-differentiated design.

To implement this, we decompose the feature $\mathbf{z}_t$ into frequency components using a generic linear frequency transform $\mathcal{T}$. Our formulation is agnostic to the specific choice of transform, and $\mathcal{T}$ can be instantiated by standard options such as the discrete cosine transform (DCT) or the Fourier transform. A radial frequency mask $\mathbf{M}$ with cutoff frequency τ is then applied to separate the spectrum into low- and high-frequency components:

$$\mathbf{z}_t^{\text{low/high}} = \mathcal{T}^{-1}(\mathcal{T}(\mathbf{z}_t) \odot \mathbf{M}_{\text{low/high}}) \tag{3}$$

Based on their distinct temporal dynamics, we apply differentiated caching strategies to the two components. For the low-frequency component, whose structure is stable but weakly continuous, we adopt a direct reuse strategy from the most recent full-computation step t_{prev}:

$$\widehat{\mathbf{z}}_t^{\text{low}} = \mathbf{z}_{t_{prev}}^{\text{low}}, \tag{4}$$

In contrast, the high-frequency component exhibits stronger temporal continuity but higher volatility, making naive reuse prone to detail degradation. We therefore model its evolution along the diffusion trajectory using a second-order Hermite interpolation, implemented via an efficient finite-difference formulation. Specifically, let $\Delta\mathbf{F}^{(1)}$ and $\Delta\mathbf{F}^{(2)}$ denote the first- and second-order discrete differences computed from high-frequency features at previous full-computation steps. The high-frequency feature at the current prediction step is then estimated as

$$\widehat{\mathbf{z}}_t^{\text{high}}(d) = \mathbf{z}_{t_{\text{prev}}}^{\text{high}} + \alpha_1(d)\,\Delta\mathbf{F}^{(1)} + \alpha_2(d)\,\Delta\mathbf{F}^{(2)}, \tag{5}$$

where d denotes the interval distance from t_{prev}, and $\alpha_k(d)$ are interpolation coefficients derived from Hermite polynomials. This closed-form predictor enables accurate reconstruction of high-frequency details with negligible computational overhead. Finally, the predicted feature is obtained by recombining the two components:

$$\widehat{\mathbf{z}}_t = \widehat{\mathbf{z}}_t^{\text{low}} + \widehat{\mathbf{z}}_t^{\text{high}}. \tag{6}$$

Both the frequency decomposition and the high-frequency prediction are generic design choices. Our framework is agnostic to the specific transform or numerical method used.

Cumulative Residual Feature (CRF) A Diffusion Transformer consists of a deep stack of L residual blocks, where each block performs an incremental update rather than a full feature replacement:

$$\mathbf{h}^{(l+1)} \;=\; \mathbf{h}^{(l)} \;+\; \mathcal{F}^{(l)}\big(\mathbf{h}^{(l)},\, t\big), \tag{7}$$

with $\mathcal{F}^{(l)}(\cdot, t)$ denoting the transformation at layer l, modulated by the diffusion timestep t. Unrolling the residual structure reveals that the final output can be expressed as

$$\phi_L(\mathbf{x}_t) \;=\; \mathbf{h}^{(0)} \;+\; \sum_{l=0}^{L-1} \mathcal{F}^{(l)}\big(\mathbf{h}^{(l)},\, t\big). \tag{8}$$

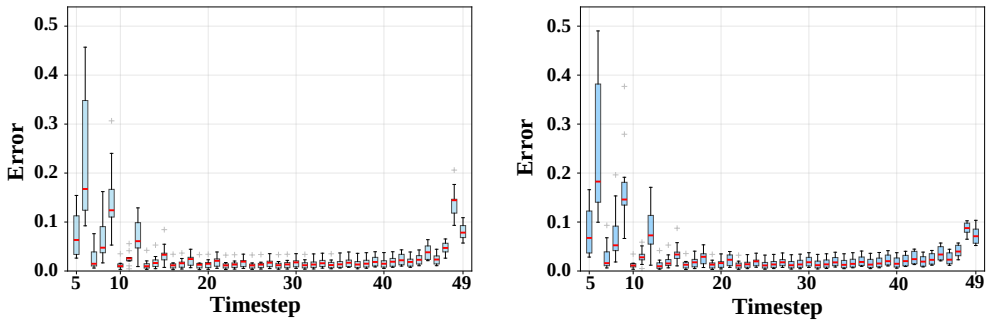
i.e., the accumulation of the input feature and all subsequent residual updates. We denote this output as $\mathbf{z}_t \triangleq \phi_L(\mathbf{x}_t)$ and refer to it as the *Cumulative Residual Feature (CRF)*.

This formulation motivates a memory-efficient caching strategy. Instead of storing all intermediate features $\{\mathbf{h}^{(l)}\}_{l=0}^{L-1}$, we cache only the CRF, which already encodes the entire transformation history. As shown in Fig. 4, CRF-only caching achieves reconstruction fidelity comparable to full layer-wise caching, incurring only a $\sim$4% higher MSE on average. This result indicates that the CRF serves as a near-lossless compression of the full computational path, enabling a reduction in memory complexity from $\mathcal{O}(L)$ to $\mathcal{O}(1)$ with negligible quality degradation.

4 Experiment

4.1 Experiment Settings

Model Configurations and Evaluation and Metrics. Experiments are performed using four models: FLUX.1-dev [12], Qwen-Image [41], FLUX.1-Kontext-dev [13], and Qwen-Image-Edit [41].



(a) Layer-wise Feature Caching **(b) Cumulative Residual Feature Caching**

Figure 4: Box plots of Mean Squared Error between ground-truth and predicted features per timestep.

For the text-to-image generation evaluation, we adopt the DrawBench [30] benchmark. The generated samples are systematically evaluated using ImageReward [42] and CLIP Score [10], which jointly measure image quality and text–image semantic alignment. To assess visual fidelity, we further employ PSNR, SSIM [40] and LPIPS [44], thereby capturing both pixel-level similarity and perceptual consistency. Additionally, we evaluate general-purpose image editing using the GEdit benchmark [19], which systematically assesses instruction-driven editing fidelity and alignment to target modifications under textual and visual guidance. Best and second-best results are highlighted in **bold** and underlined, respectively. Methods marked with † indicate a significant degradation in image quality.

Table 1: Quantitative comparison for FLUX.1-dev and FLUX.1-schnell.

Method	Acceleration				Quality Metrics		Perceptual Metrics		
	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	ImageReward↑	CLIP↑	PSNR↑	SSIM↑	LPIPS↓
[dev]: 50 steps	23.24 (+0.0%)	1.00×	3726.87	1.00×	0.99 (+0.0%)	32.64 (+0.0%)	∞	1.00	0.00
60% **steps**	14.12 (-39.2%)	1.65×	2236.12	1.67×	0.97 (-2.0%)	32.66 (+0.1%)	30.31	0.78	0.25
50% **steps**	11.82 (-49.1%)	1.97×	1863.44	2.00×	0.97 (-2.0%)	32.57 (-0.2%)	29.56	0.73	0.31
PAB	17.84 (-23.2%)	1.30×	3013.13	1.24×	0.95 (-4.0%)	32.55 (-0.3%)	28.84	0.67	0.40
DBCache	16.88 (-27.4%)	1.38×	2384.29	1.56×	1.01 (+2.0%)	32.53 (-0.3%)	33.86	0.87	0.12
FORA ($\mathcal{N}=3$)	**9.06** (-61.0%)	**2.57**×	**1267.89**	**2.94**×	0.93 (-6.1%)	**32.89** (+0.8%)	28.86	0.66	0.40
TeaCache ($l=0.6$)	9.13 (-60.7%)	2.55×	1342.20	2.78×	0.91 (-8.1%)	32.11 (-1.6%)	29.03	0.68	0.40
TaylorSeer ($\mathcal{N}=3, O=2$)	10.16 (-56.3%)	2.29×	1416.92	2.63×	**1.01** (+2.0%)	32.86 (+0.7%)	30.77	0.78	0.23
FreqCa	9.37 (-59.7%)	2.48×	1417.40	2.63×	1.00 (+1.0%)	32.61 (-0.1%)	**33.03**	**0.86**	**0.13**
FORA($\mathcal{N}=5$) †	5.97 (-74.3%)	3.89×	820.80	4.54×	0.82 (-17.2%)	32.48 (-0.5%)	28.44	0.60	0.50
ToCa($\mathcal{N}=8, \mathcal{R}=75\%$)	12.39 (-46.7%)	1.88×	829.86	4.49×	0.95 (-4.0%)	32.60 (-0.1%)	29.07	0.64	0.43
DuCa($\mathcal{N}=8, \mathcal{R}=70\%$)	9.40 (-59.6%)	2.47×	858.27	4.34×	0.94 (-5.1%)	32.58 (-0.2%)	29.06	0.64	0.43
TeaCache($l=1.0$) †	7.07 (-69.6%)	3.29×	820.55	4.54×	0.84 (-15.2%)	31.88 (-2.3%)	28.61	0.64	0.48
TaylorSeer($\mathcal{N}=6, O=2$)	6.73 (-71.0%)	3.45×	746.28	**4.99**×	1.00 (+1.0%)	32.91 (+0.8%)	28.94	0.66	0.40
FreqCa	**5.19** (-77.7%)	**4.48**×	**746.03**	4.99×	**1.01** (+2.0%)	**32.98** (+1.0%)	**30.00**	**0.71**	**0.31**
FORA($\mathcal{N}=7$) †	5.09 (-78.1%)	4.57×	**597.25**	**6.24**×	0.68 (-31.3%)	31.90 (-2.3%)	28.32	0.59	0.54
ToCa($\mathcal{N}=12, \mathcal{R}=85\%$) †	9.82 (-57.7%)	2.37×	618.57	6.02×	0.80 (-19.2%)	32.32 (-1.0%)	28.70	0.60	0.52
DuCa($\mathcal{N}=12, \mathcal{R}=80\%$) †	7.74 (-66.7%)	3.00×	646.97	5.76×	0.77 (-22.2%)	32.20 (-1.3%)	28.71	0.60	0.53
TeaCache($l=1.4$) †	6.14 (-73.6%)	3.79×	671.51	5.55×	0.74 (-25.3%)	31.78 (-2.6%)	28.12	0.48	0.68
TaylorSeer($\mathcal{N}=9, O=2$) †	5.85 (-74.8%)	3.97×	**597.25**	**6.24**×	0.86 (-13.1%)	32.04 (-1.8%)	28.38	0.59	0.51
FreqCa	**4.25** (-81.7%)	**5.47**×	597.47	6.24×	**0.97** (-2.0%)	**32.53** (-0.3%)	**28.77**	**0.62**	**0.43**
[schnell]: 4 steps	2.17(+0.0%)	1.00×	278.41	1.00×	0.93(+0.0%)	34.09(+0.0%)	∞	1.00	0.00
FreqCa	1.29 (-40.6%)	1.68×	139.23	2.00×	0.95 (+2.2%)	34.47 (+1.1%)	30.30	0.81	0.16

4.2 Text-to-Image Generation

FLUX As shown in Tab. 1, FreqCa consistently outperforms state-of-the-art acceleration methods across various speedup levels. Specifically, at 2.63× FLOPs speedup, FreqCa achieves an ImageReward of 1.00, clearly surpassing FORA and TeaCache. At 4.99× speedup, it maintains lossless quality with an ImageReward of 1.01. Even under an aggressive 6.24× speedup, FreqCa incurs only a minimal 2% drop in ImageReward (0.97), whereas TaylorSeer suffers a significant degradation of 13.1%, highlighting the superior robustness of our method. Furthermore, to verify the compatibility on distilled models, we apply FreqCa to FLUX.1-schnell. It achieves a 2.00× speedup while improving the ImageReward from 0.93 to 0.95, demonstrating its broad applicability even on few-step generation tasks.

Qwen-Image As shown in Tab. 2, FreqCa demonstrates superior performance across different acceleration levels. At a 5.00× FLOPs speedup, FreqCa achieves an ImageReward of 1.20, significantly outperforming TaylorSeer (1.01). Notably, under an extreme 7.14× speedup, FreqCa maintains an ImageReward of 1.02 (representing only an 18.4% drop), whereas TaylorSeer suffers a severe

Table 2: Quantitative comparison of Qwen-Image and Qwen-Image-Lightning.

Method	Acceleration				Quality Metrics		Perceptual Metrics		
	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	ImageReward↑	CLIP↑	PSNR↑	SSIM↑	LPIPS↓
50 **steps**	127.40 (+0.0%)	1.00×	12917.56	1.00×	1.25 (+0.0%)	35.59 (+0.0%)	∞	1.00	0.00
50% **steps**	64.10 (-49.6%)	1.99×	6458.78	2.00×	1.20 (-4.0%)	35.31 (-0.6%)	30.54	0.75	0.28
20% **steps** †	25.92 (-79.6%)	4.89×	2583.51	5.00×	0.94 (-24.8%)	34.95 (-1.0%)	28.59	0.61	0.52
FORA($\mathcal{N}=4$)†	38.43 (-70.0%)	3.32×	3359.99	3.84×	0.93 (-25.6%)	34.40 (-3.3%)	28.66	0.59	0.51
ToCa($\mathcal{N}=8, \mathcal{R}=75\%$)†	61.37 (-51.8%)	2.08×	2991.34	4.32×	1.02 (-18.4%)	34.96 (-1.8%)	28.93	0.63	0.44
DuCa($\mathcal{N}=9, \mathcal{R}=80\%$)†	34.73 (-72.7%)	3.67×	2958.13	4.37×	0.77 (-38.4%)	34.62 (-2.7%)	28.45	0.58	0.55
TaylorSeer($\mathcal{N}=6$)	30.75 (-75.9%)	4.14×	**2583.97**	**5.00**×	1.01 (-19.2%)	34.71 (-2.5%)	28.58	0.62	0.46
FreqCa	**29.80** (-76.6%)	**4.28**×	2584.70	5.00×	**1.20** (-4.0%)	**35.39** (-0.6%)	**29.67**	**0.78**	**0.33**
FORA($\mathcal{N}=6$)†	28.69 (-77.5%)	4.44×	2326.74	5.55×	0.48 (-61.6%)	33.34 (-6.3%)	28.48	0.55	0.59
ToCa($\mathcal{N}=12, \mathcal{R}=85\%$)†	50.95 (-60.0%)	2.50×	2406.20	5.37×	0.55 (-56.0%)	34.08 (-4.2%)	28.69	0.57	0.53
DuCa($\mathcal{N}=12, \mathcal{R}=90\%$)†	28.57 (-77.6%)	4.46×	2171.56	5.95×	0.41 (-67.2%)	33.38 (-6.2%)	28.38	0.57	0.60
TaylorSeer($\mathcal{N}=9$)†	24.64 (-80.7%)	5.17×	2067.29	6.25×	0.73 (-41.6%)	32.97 (-7.4%)	28.25	0.56	0.58
FreqCa	**22.45** (-82.4%)	**5.68**×	**1809.38**	**7.14**×	**1.02** (-18.4%)	**35.00** (-1.7%)	**28.86**	**0.64**	**0.44**
Qwen-Image-Lightning: 8 steps	7.27 (+0.0%)	1.00×	560.96	1.00×	1.30 (+0.0%)	35.26 (+0.0%)	∞	1.00	0.00
FreqCa	5.29 (-27.2%)	1.37×	350.61	1.60×	**1.29** (-0.8%)	35.23 (-0.1%)	**33.01**	**0.83**	**0.12**
FreqCa	4.63 (-36.3%)	1.57×	282.26	1.99×	1.28 (-1.5%)	35.10 (-0.5%)	31.36	0.77	0.18
FreqCa	**3.29** (-54.9%)	**2.21**×	**140.25**	**4.00**×	1.29 (-0.8%)	**35.94** (+1.9%)	29.25	0.62	0.36

Table 3: Quantitative comparison of image editing for FLUX.1-Kontext-dev.

Method	Acceleration				GEdit-EN (Full)		
	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	Q_SC ↑	Q_PQ ↑	Q_O ↑
[Kontext]50 steps	50.20 (+0.0%)	1.00×	8299.54	1.00×	6.481	7.331	6.213 (+0.0%)
50% steps	25.42 (-49.4%)	1.97×	4149.77	2.00×	6.544	7.286	6.253 (+0.6%)
20% steps	10.47 (-79.1%)	4.79×	1659.91	5.00×	6.603	7.184	6.286 (+1.2%)
ToCa($\mathcal{N}$=8, $\mathcal{R}$=70%)	29.56 (-41.1%)	1.70×	1841.35	4.51×	6.432	7.256	6.125 (-1.4%)
DuCa($\mathcal{N}$=8, $\mathcal{R}$=60%)	13.12 (-73.9%)	3.83×	1669.08	4.97×	6.469	7.195	6.150 (-1.0%)
TaylorSeer($\mathcal{N}$=6, $\mathcal{O}$=2)	13.95 (-72.2%)	3.60×	**1660.95**	**5.00**×	6.477	**7.296**	6.170 (-0.7%)
FreqCa	**10.77** (-78.5%)	**4.66**×	1661.37	5.00×	**6.480**	7.271	**6.195** (-0.3%)
ToCa($\mathcal{N}$=12, $\mathcal{R}$=75%)	20.72 (-58.7%)	2.42×	1359.61	6.10×	6.393	6.919	6.041 (-2.8%)
DuCa($\mathcal{N}$=12, $\mathcal{R}$=70%)	10.39 (-79.3%)	4.83×	1376.55	6.03×	**6.597**	7.005	6.173 (-0.6%)
TaylorSeer($\mathcal{N}$=9, $\mathcal{O}$=2)	12.05 (-76.0%)	4.17×	**1329.02**	**6.24**×	6.407	6.995	6.074 (-2.2%)
FreqCa	**8.74** (-82.6%)	**5.74**×	1329.33	6.24×	6.550	**7.160**	**6.190** (-0.4%)

41.6% quality loss (0.73) even at a lower speedup of 6.25×. A detailed visual sample is shown in Fig. 7. Furthermore, on the distilled Qwen-Image-Lightning, FreqCa achieves a 4.00× speedup with negligible quality degradation, validating its effectiveness on few-step models.

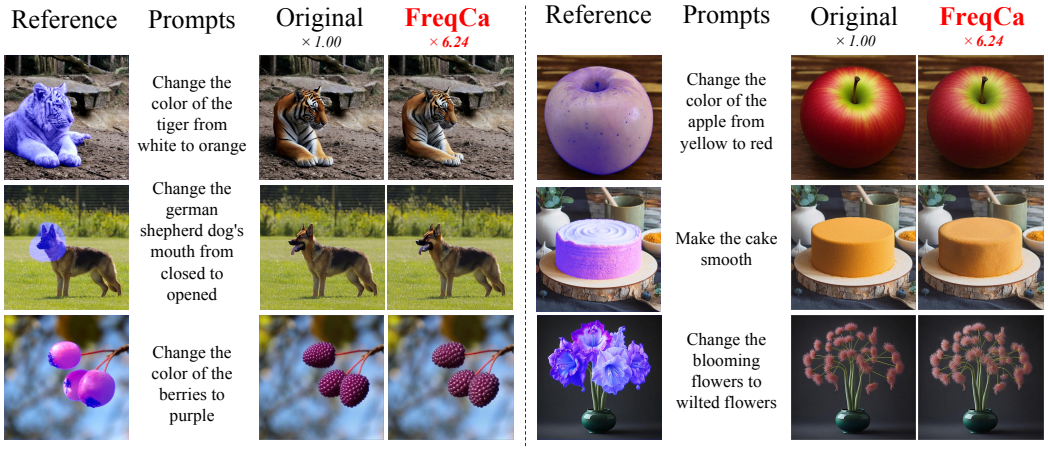


Figure 5: On FLUX.1-Fill, *FreqCa* achieves 6.24× speedup with indistinguishable quality.

4.3 Image Editing

FLUX.1-Kontext-dev FreqCa consistently outperforms other acceleration methods in preserving editing quality which shown in Tab. 3. At a 5.00× speedup, FreqCa achieves a QO score of 6.195, surpassing both TaylorSeer and DuCa. Even under an aggressive 6.24× speedup, FreqCa demonstrates exceptional fidelity with only a minimal 0.4% drop in QO score (maintaining 6.190). In contrast, competitors like TaylorSeer and ToCa suffer more noticeable degradation, dropping to 6.074

and 6.041 respectively. This indicates that FreqCa not only accelerates the inference significantly but also effectively maintains the perceptual consistency required for high-quality image editing.

Table 4: Quantitative comparison of image editing using Qwen-Image-Edit.

Method	Acceleration				GEdit-CN (Full)			GEdit-EN (Full)		
	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	Q_SC ↑	Q_PQ ↑	Q_O ↑	Q_SC ↑	Q_PQ ↑	Q_O ↑
[full]50 steps	284.51 (+0.0%)	1.00×	28190.88	1.00×	7.68	7.51	7.41 (+0.0%)	7.82	7.54	7.54 (+0.0%)
50% steps	143.29 (-49.6%)	1.99×	14095.44	2.00×	7.70	7.53	7.44 (+0.4%)	7.77	7.52	7.47 (-0.9%)
20% steps	58.45 (-79.5%)	4.87×	5638.18	5.00×	7.65	7.42	7.35 (-0.8%)	7.73	7.46	7.44 (-1.3%)
FORA($\mathcal{N}$=5)	63.15 (-77.8%)	4.51×	5643.13	5.00×	7.60	7.31	7.25 (-2.2%)	7.62	7.34	7.28 (-3.4%)
DuCa($\mathcal{N}$=7, $\mathcal{R}$=95%)	69.54 (-75.5%)	4.09×	5699.89	4.95×	7.73	7.44	7.44 (+0.4%)	7.80	7.40	7.45 (-1.2%)
TaylorSeer($\mathcal{N}$=6, $\mathcal{O}$=2)	65.66 (-76.9%)	4.33×	5643.13	5.00×	7.25	7.09	6.92 (-6.6%)	7.26	7.14	6.89 (-8.6%)
FreqCa	**62.89** (-77.9%)	**4.52×**	**5642.24**	**5.00×**	**7.75**	**7.54**	**7.49** (+1.1%)	**7.83**	**7.48**	**7.52** (-0.3%)
FORA($\mathcal{N}$=7)	52.20 (-81.7%)	5.45×	4515.74	6.24×	7.42	7.13	7.06 (-4.7%)	7.43	**7.19**	7.06 (-6.3%)
DuCa($\mathcal{N}$=10, $\mathcal{R}$=95%)	59.81 (-79.0%)	4.76×	5158.45	5.46×	7.50	5.75	6.39 (-13.8%)	7.52	5.77	6.41 (-15.0%)
TaylorSeer($\mathcal{N}$=9, $\mathcal{O}$=2)	53.92 (-81.1%)	5.28×	4515.74	6.24×	6.61	6.65	6.31 (-14.8%)	6.67	6.63	6.31 (-16.3%)
FreqCa	**51.09** (-82.0%)	**5.57×**	**4514.48**	**6.24×**	**7.62**	**7.18**	**7.27** (-1.9%)	**7.66**	7.12	**7.21** (-4.3%)

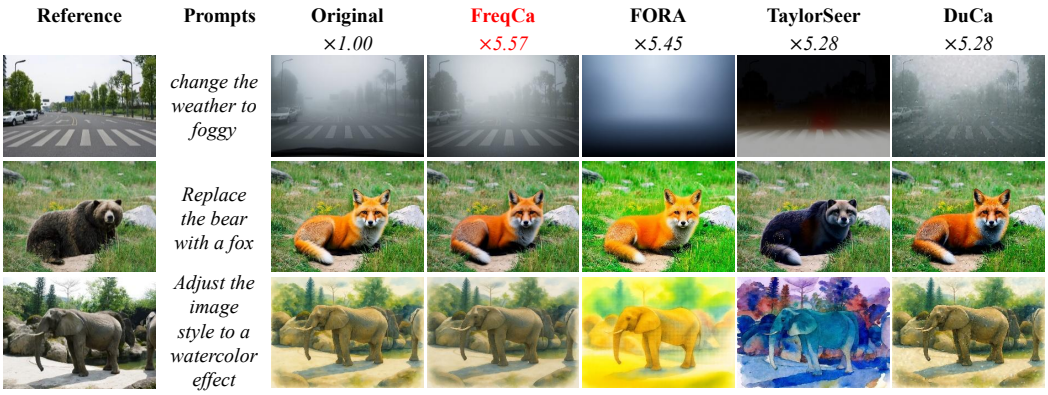


Figure 6: On Qwen-Image-Edit, *FreqCa* delivers higher speedup with near-original editing quality

Qwen-Image-Edit FreqCa demonstrates superior capability in handling bilingual editing tasks. As shown in Tab. 4, at a 5.00× speedup, it achieves high QO scores of 7.49 (GEdit-CN) and 7.52 (GEdit-EN), significantly surpassing TaylorSeer. The robustness of FreqCa is further highlighted under an aggressive 6.24× speedup, where it limits quality degradation to merely 1.9% and 4.3%, whereas TaylorSeer suffers severe drops of 14.8% and 16.3%. This quantitative advantage is shown in Fig.6. While other methods exhibit noticeable artifacts, FreqCa maintains visual consistency comparable to the original model.

4.4 Ablation Studies

1.High-Frequency Predictor Choice We compare Hermite, Taylor, Chebyshev, and Laguerre interpolation for high-frequency prediction. **Second-order Hermite performs best**, as it preserves local curvature by leveraging both values and derivatives. Taylor accumulates errors under nonlinear diffusion, while Chebyshev and Laguerre introduce oscillations. As shown in Tab. 5, at 7.14× speedup, Hermite achieves the highest ImageReward (1.04), outperforming Taylor (0.98), Laguerre (0.97), and Chebyshev (0.95). TaylorSeer performs significantly worse (0.73), highlighting the importance of frequency decoupling. Overall, Hermite is the most reliable high-frequency predictor.

Table 7: Overhead of frequency decomposition and reconstruction across resolutions.

Method	Resolution	Full Step(ms)	Cached Step(ms)	Decomp.(ms)	Decomp.(%)	Recon.(ms)	Recon.(%)
FFT	512 × 512	185.03	0.98	0.54	0.29	0.02	2.04
	1024 × 1024	690.59	0.98	0.80	0.12	0.02	2.04
	1328 × 1328	1287.35	0.99	0.81	0.06	0.02	2.02
	2048 × 2048	4215.03	0.99	0.86	0.02	0.02	2.02
DCT	512 × 512	187.16	0.84	2.07	1.11	0.02	2.38
	1024 × 1024	693.33	0.86	2.31	0.33	0.02	2.33
	1328 × 1328	1290.31	0.85	2.30	0.18	0.02	2.35
	2048 × 2048	4215.04	0.84	2.33	0.06	0.02	2.38

Table 5: Effect of high-frequency predictor choice in FreqCa.

Method	Pred.	Speed	IR
TaylorSeer	Taylor	6.25×	0.73
FreqCa	Hermite	7.14×	**1.04**
FreqCa	Taylor	7.14×	0.98
FreqCa	Chebyshev	7.14×	0.95
FreqCa	Laguerre	7.14×	0.97

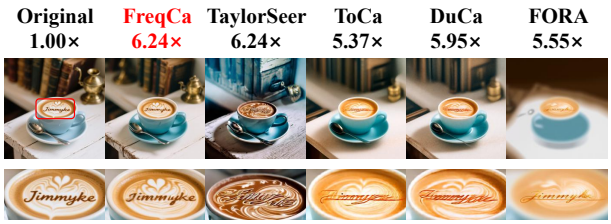


Figure 7: *FreqCa* achieves 6.24× speedup with higher fidelity than prior methods on Qwen-Image.

2.Frequency Decomposition We compare FFT, DCT, and a no-decomposition baseline under a unified setting. As shown in Tab. 6 and Fig. 12, all methods perform similarly at low acceleration ratios, but diverge as the caching interval increases. Frequency decomposition (FFT/DCT) consistently preserves generation quality, whereas the no-decomposition baseline degrades significantly. Notably, FFT and DCT achieve comparable results, indicating that the gains mainly arise from frequency-aware feature separation rather than the specific transform. As shown in Tab. 7, the additional computational cost is negligible.

Table 6: **Ablation of frequency decomposition operators measured by ImageReward (↑)**

Model	Decomp.	$N=3$	$N=6$	$N=9$
FLUX.1-dev	None	0.9934	0.9966	0.9310
	FFT	0.9866	1.0169	0.9776
	DCT	0.9976	1.0153	0.9977
Qwen-Image	None	1.2031	0.7648	0.2899
	FFT	1.2128	1.1978	1.0990
	DCT	1.2009	1.1681	1.0166

3.Cache Overhead Analysis Let $\mathcal{K}$ denote the cache memory overhead (abstracted as cache units) and $\mathcal{C}$ represent the computational cost (FLOPs). Conventional layer-wise caching methods necessitate storing features (e.g., Attention and MLP outputs, $N = 2$) across all layers to support m-th order prediction. This results in a memory complexity of $\mathcal{O}(L)$, specifically $\mathcal{K}_{\text{layer}} = N(m+1)L$. For a deep model like FLUX.1-dev ($L = 57$) using second-order prediction ($m = 2$), this requires a staggering 342 cache units.

In stark contrast, **FreqCa** operates solely on the Cumulative Residual Feature caching (CRF), effectively decoupling memory cost from model depth ($\mathcal{O}(1)$). By adopting a frequency-decoupled strategy: caching the reused low-frequency component (1 unit) and high-frequency history for Hermite interpolation (3 units), the total cost is constant ($\mathcal{K}_{\text{FreqCa}} = 4$). The relative cache-memory footprint is:

$$R = \frac{\mathcal{K}_{\text{FreqCa}}}{\mathcal{K}_{\text{layer}}} = \frac{4}{(m+1)\cdot N \cdot L} \approx 1.17\%. \tag{9}$$

Computationally, since $C_{\text{pred}} \ll C_{\text{full}}$:

$$\bar{\mathcal{C}} = \tfrac{1}{\mathcal{N}} C_{\text{full}} + \left(1 - \tfrac{1}{\mathcal{N}}\right) C_{\text{pred}} \Rightarrow \text{Speedup} \approx \mathcal{N}. \tag{10}$$

Thus, *FreqCa* achieves near-$\mathcal{N}\times$ acceleration with only $\sim 1\%$ memory overhead.

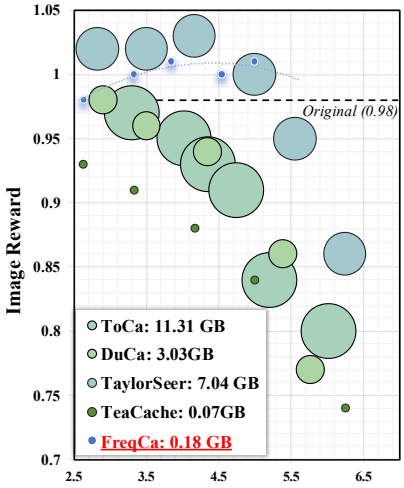


Figure 8: **Quality–efficiency trade-off on FLUX.1-dev.** The legend reports their additional memory usage relative to the full-compute baseline.

5 Conclusion

In this work, we present ***FreqCa***, a frequency-aware feature caching framework for efficient diffusion inference. We reveal that diffusion features follow heterogeneous temporal dynamics across frequency bands: low-frequency components are more suitable for direct reuse, while high-frequency components benefit from high-order prediction. Based on this observation, *FreqCa* decomposes features into frequency-specific components and applies tailored caching strategies, thereby unifying reuse- and forecast-based paradigms. To further improve practicality, we introduce Cumulative Residual Feature (CRF) caching to reduce cache memory complexity to $\mathcal{O}(1)$. Experiments across diverse diffusion models show that *FreqCa* achieves 6–7× acceleration with negligible quality degradation, demonstrating strong robustness and deployment value.

References

[1] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.

[2] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4599–4603, 2023.

[3] Peiliang Cai, Jiacheng Liu, Haowen Xu, Xinyu Wang, Chang Zou, and Linfeng Zhang. Lesa: Learnable stage-aware predictors for diffusion model acceleration, 2026. URL https://arxiv.org/abs/2602.20497.

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

[5] Zegang Cheng, Zhikai Wang, Jiacheng Liu, Xiaobing Tu, Chang Zou, Junjie Chen, Yue Ma, Zhiyuan Ma, Jinkui Ren, Likai Zou, and Linfeng Zhang. Z-cache: Accelerating diffusion transformers via self-reflection, 2026. URL https://arxiv.org/abs/2611.01846.

[6] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models, 2023. URL https://arxiv.org/abs/2305.10924.

[7] Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, and Linfeng Zhang. Hicache: A plug-in scaled-hermite upgrade for taylor-style cache-then-forecast diffusion acceleration, 2026. URL https://arxiv.org/abs/2508.16984.

[8] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models, 2025. URL https://arxiv.org/abs/2410.12557.

[9] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling, 2025. URL https://arxiv.org/abs/2505.13447.

[10] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021.

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

[12] Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024.

[13] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space, 2025. URL https://arxiv.org/abs/2506.15742.

[14] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36, 2024.

[15] Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model, 2024. URL https://arxiv.org/abs/2411.19108.

[16] Jiacheng Liu, Xinyu Wang, Yuqi Lin, Zhikai Wang, Peiru Wang, Peiliang Cai, Qinming Zhou, Zhengan Yan, Zexuan Yan, Zhengyi Shi, Chang Zou, Yue Ma, and Linfeng Zhang. A survey on cache methods in diffusion models: Toward efficient multi-modal generation, 2025. URL https://arxiv.org/abs/2510.19755.

[17] Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers, 2025. URL https://arxiv.org/abs/2503.06923.

[18] Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Fei Ren, Shaobo Wang, Kaixin Li, and Linfeng Zhang. Speca: Accelerating diffusion transformers with speculative feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, page 10024–10033. ACM, October 2025. doi: 10.1145/3746027.3755331. URL http://dx.doi.org/10.1145/3746027.3755331.

[19] Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, Guopeng Li, Yuang Peng, Quan Sun, Jingwei Wu, Yan Cai, Zheng Ge, Ranchen Ming, Lei Xia, Xianfang Zeng, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, Gang Yu, and Daxin Jiang. Step1x-edit: A practical framework for general image editing, 2025. URL https://arxiv.org/abs/2504.17761.

[20] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL https://arxiv.org/abs/2209.03003.

[21] Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers, 2025. URL https://arxiv.org/abs/2502.10389.

[22] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[23] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022.

[24] Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. Fastercache: Training-free video diffusion model acceleration with high quality, 2025. URL https://arxiv.org/abs/2410.19355.

[25] 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.

[26] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models, 2023. URL https://arxiv.org/abs/2210.03142.

[27] 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.

[28] Haoran Qin, Zhengan Yan, Shikang Zheng, Xiaobing Tu, Jiacheng Liu, Yuqi Lin, Chang Zou, JinShan Liu, Peiliang Cai, Xiantao Zhang, Jinkui Ren, and Linfeng Zhang. Avits: Adaptive spatiotemporal token selection for efficient dynamic-resolution generation, 2026. URL https://arxiv.org/abs/2608.17995.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[30] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. URL http://arxiv.org/abs/2205.11487.

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

[32] 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.

[33] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models, 2023. URL https://arxiv.org/abs/2211.15736.

[34] Zhirong Shen, Rui Huang, Jiacheng Liu, Chang Zou, Peiliang Cai, Shikang Zheng, Zhengyi Shi, Liang Feng, and Linfeng Zhang. Beyond fixed formulas: Data-driven linear predictor for efficient diffusion models, 2026. URL https://arxiv.org/abs/2604.26365.

[35] Zhirong Shen, Rui Huang, Chang Zou, Shikang Zheng, Jiacheng Liu, Peiliang Cai, Zhengyi Shi, Yaosong Du, Liang Feng, Xiaobing Tu, Jinkui Ren, Xiantao Zhang, and Linfeng Zhang. Accelerating diffusion transformers with gaussian process rectified feature cache, 2026. URL https://arxiv.org/abs/2609.05981.

[36] Chenyang Si, Ziqi Huang, Yuming Jiang, and Ziwei Liu. Freeu: Free lunch in diffusion u-net, 2023. URL https://arxiv.org/abs/2309.11497.

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

[38] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models, 2023. URL https://arxiv.org/abs/2303.01469.

[39] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016. URL https://api.semanticscholar.org/CorpusID:715122.

[40] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861. URL https://doi.org/10.1109/TIP.2003.819861.

[41] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025. URL https://arxiv.org/abs/2508.02324.

[42] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and Evaluating Human Preferences for Text-to-Image Generation, December 2023. URL http://arxiv.org/abs/2304.05977. arXiv:2304.05977 [cs].

[43] Zhengan Yan, Shikang Zheng, Haoran Qin, Xiaobing Tu, Yinggui Wang, Jiacheng Liu, Jiaxuan Ren, Yuqi Lin, Peiliang Cai, Jinkui Ren, Xiantao Zhang, and Linfeng Zhang. Specedit: Training-free acceleration for diffusion based image editing via semantic locking, 2026. URL https://arxiv.org/abs/2605.02152.

[44] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. URL https://arxiv.org/abs/1801.03924.

[45] Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast, 2025. URL https://arxiv.org/abs/2408.12588.

[46] Shikang Zheng, Guantao Chen, Qinming Zhou, Yuqi Lin, Lixuan He, Chang Zou, Peiliang Cai, Jiacheng Liu, and Linfeng Zhang. Let features decide their own solvers: Hybrid feature caching for diffusion transformers, 2025. URL https://arxiv.org/abs/2510.04188.

[47] Shikang Zheng, Liang Feng, Xinyu Wang, Qinming Zhou, Peiliang Cai, Chang Zou, Jiacheng Liu, Yuqi Lin, Junjie Chen, Yue Ma, and Linfeng Zhang. Forecast then calibrate: Feature caching as ode for efficient diffusion transformers, 2025. URL https://arxiv.org/abs/2508.16211.

[48] Shikang Zheng, Guantao Chen, Lixuan He, Jiacheng Liu, Yuqi Lin, Chang Zou, and Linfeng Zhang. From sketch to fresco: Efficient diffusion transformer with progressive resolution, 2026. URL https://arxiv.org/abs/2601.07462.

[49] Shikang Zheng, Jingkai Huang, Jiacheng Liu, Guantao Chen, Lixuan, Yuqi Lin, Peiliang Cai, and Linfeng Zhang. Dynamic video generation: Shaping video generation across time and space, 2026. URL https://arxiv.org/abs/2605.21042.

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

[51] Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In *Proceedings of the 13th International Conference on Learning Representations (ICLR 2025)*. ICLR, 2025. URL https://openreview.net/forum?id=yYZbZGo4ei. accepted to ICLR 2025.

[52] Chang Zou, Changlin Li, Yang Li, Patrol Li, Jianbing Wu, Xiao He, Songtao Liu, Zhao Zhong, Kailin Huang, and Linfeng Zhang. Disca: Accelerating video diffusion transformers with distillation-compatible learnable feature caching, 2026. URL https://arxiv.org/abs/2602.05449.

A Additional Visualizations

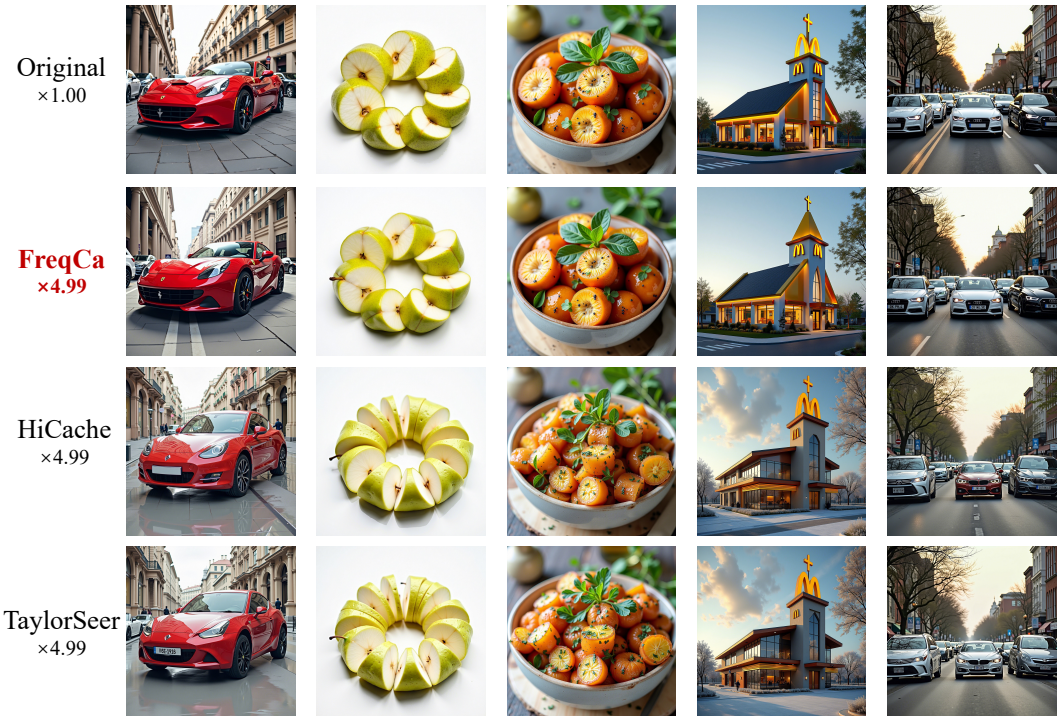


Figure 9: Qualitative comparison on FLUX under a nearly $5\times$ speedup ratio. FreqCa better preserves image structure, style consistency, and fine-grained details compared with the original outputs.

We provide additional visualizations on FLUX.1-dev. As shown in Fig. 9, FreqCa achieves visual results that are more consistent with the original outputs in terms of image structure, style, and fine-grained details, while maintaining a nearly $5\times$ acceleration ratio.

B Experimental Setup

This section provides comprehensive technical details for all experimental configurations. We emphasize that the specific choices of frequency decomposition (DCT or FFT) reported here are provided for **reproducibility**. As shown in Sec. 4.4, FFT and DCT generally provide larger benefits over the no-decomposition baseline at longer caching intervals in the tested settings. *This indicates that the effectiveness of FreqCa does not rely on a particular decomposition or reconstruction method, but rather on the underlying frequency-aware feature separation.*

B.1 Model and Task Specifications

FLUX.1-dev and Qwen-Image: The images generated by the FLUX.1-dev, FLUX.1-schnell, and Qwen-Image-Lightning models are obtained at 1024×1024 resolution using 200 high-quality prompts sourced from the DrawBench benchmark, while those generated by Qwen-Image were obtained at 1328×1328 resolutions. Quality assessment is performed using ImageReward, a robust perceptual metric for text-to-image alignment.

FLUX.1-Kontext-dev and Qwen-Image-Edit: We employ the FLUX.1-Kontext-dev and Qwen-Image-Edit model for image editing synthesis. Images editing and quality assessment is performed using GEdit benchmark, which grounded in real-world usages is developed to support more authentic and comprehensive evaluation of image editing models.

B.2 Hardware and Computational Resources

All experiments are conducted on enterprise-grade GPU infrastructure:

- FLUX.1-dev experiments: NVIDIA A100 GPU
- FLUX.1-Kontext-dev experiments: NVIDIA A100 GPU
- Qwen-Image experiments: NVIDIA H20 GPU
- Qwen-Image-Edit experiments: NVIDIA H20 GPU

B.3 FreqCa Implementation Parameters

- FLUX.1-dev experiments: DCT-based frequency decomposition was adopted.
- FLUX.1-Kontext-dev experiments: DCT-based frequency decomposition was adopted.
- Qwen-Image experiments: FFT-based frequency decomposition was adopted.
- Qwen-Image-Edit experiments: FFT-based frequency decomposition was adopted.

C Algorithmic Details

C.1 Caching Strategy and Interaction with Hyperparameter N

The hyperparameter N serves as the core control for the **caching and reuse strategy** within FreqCa. To address the instability of predicted noise in the early stages of multi-step diffusion, we implement a `first_enhance` stage, where the initial few steps are computed entirely by the main model. In subsequent stages, the model operates with an interval of $\mathcal{N}$: one full-computation step (denoted as C) is followed by $\mathcal{N}-1$ prediction steps (denoted as P), which are handled by the FreqCa predictor without executing the diffusion backbone.

For example, in a 10-step diffusion process with `first_enhance` $= 2$ and an interval $N = 3$, the execution sequence is `CCPPCPPCPP`. During prediction steps, the diffusion model is bypassed. Instead, we reuse the low-frequency component from the most recent full computation step and synthesize the high-frequency component using a Hermite-based predictor.

Full Computation Steps (Generation Logic): At these steps, we decompose the model output into frequency bands and compute finite-difference residuals for the high-frequency data. Let F_0, F_1, F_2 be the high-frequency data from the latest three full computation steps at times t_0, t_1, t_2. The specific difference calculations are:

$$\begin{aligned} \Delta F^{(1)}_{\text{curr}} &= \frac{F_0 - F_1}{t_0 - t_1}, \\ \Delta F^{(2)}_{\text{curr}} &= \frac{\Delta F^{(1)}_{\text{curr}} - \Delta F^{(1)}_{\text{prev}}}{t_0 - t_1} = \frac{\frac{F_0 - F_1}{t_0 - t_1} - \frac{F_1 - F_2}{t_1 - t_2}}{t_0 - t_1}. \end{aligned} \tag{11}$$

Prediction Steps (Usage Logic): We reuse the low-frequency data from the previous full computation step and predict the high-frequency data $\widehat{F}_{\text{high}}$ as follows:

$$\widehat{F}_{\text{high}}(d) = F_0 + \alpha_1(d)\Delta F^{(1)} + \alpha_2(d)\Delta F^{(2)}. \tag{12}$$

Here, F_0 is the high-frequency data from the previous full computation step, and $\Delta F^{(1)}, \Delta F^{(2)}$ are the derivatives computed at that step. The coefficients $\alpha_k(d)$ are derived from Hermite polynomials:

$$\begin{aligned} \alpha_k(d) &= \frac{H_k(x)}{k!} s^k, \\ H_1(x) = 2x, \quad H_2(x) &= 4x^2 - 2, \quad x = sd. \end{aligned} \tag{13}$$

where s is a tuning factor and d is the distance to the previous full computation step. The final output is the sum of the cached low-frequency data and the predicted high-frequency data.

C.2 Spectral Feature Decomposition Details

In our diffusion transformer, latent features $X_{\text{tok}} \in \mathbb{R}^{B\times L\times D}$ are reshaped into spatial feature maps $X \in \mathbb{R}^{B\times C\times H\times W}$ via the transformation $[B, L, D] \to [B, H, W, D] \to [B, C, H, W]$. We utilize a

2D Type-II Discrete Cosine Transform (DCT) rather than FFT to avoid boundary ringing artifacts caused by periodic extension. The transform is defined as:

$$Y_{b,c}(u,v) = \alpha_u \alpha_v \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} X_{b,c}(x,y) \cdot \cos\left[\frac{\pi}{H}\left(x+\tfrac{1}{2}\right)u\right] \cos\left[\frac{\pi}{W}\left(y+\tfrac{1}{2}\right)v\right]. \tag{14}$$

Using a radial mask with cutoff τ, we separate the spectrum into Y_{low} and Y_{high}. By linearity of the inverse transform $\mathcal{T}^{-1}$, we obtain the decomposed features:

$$\begin{aligned} X_{\text{low}} &= \mathcal{T}^{-1}(Y \odot M_{\text{low}}), \\ X_{\text{high}} &= \mathcal{T}^{-1}(Y \odot M_{\text{high}}). \end{aligned} \tag{15}$$

X_{low} represents the stable structural component suitable for caching, while X_{high} captures volatile details handled by the predictor. For implementation, we use `torch_dct.dct_2d` with orthogonal normalization.

C.3 Second-Order Hermite Predictor and Efficient Computation

To evaluate the trajectory efficiently, we employ a forward-difference operator $\Delta g(d) = g(d+1) - g(d)$. The second-order Hermite interpolator used in FreqCa corresponds to:

$$\widehat{F}(d) = F_0 + 2s^2 d\, \Delta F^{(1)} + \left(2s^4 d^2 - s^2\right) \Delta F^{(2)}. \tag{16}$$

Since $\widehat{F}(d)$ is a quadratic polynomial in d, all differences of order $m \geq 3$ vanish. We can thus simplify the computation using recursive forward differences:

$$\begin{aligned} \Delta \widehat{F}(d) &= 2s^2\, \Delta F^{(1)} + \left(4s^4 d + 2s^4\right) \Delta F^{(2)}, \\ \Delta^2 \widehat{F}(d) &= 4s^4\, \Delta F^{(2)}, \\ \Delta^m \widehat{F}(d) &= 0, \quad \text{for } m \geq 3. \end{aligned} \tag{17}$$

In implementation, $\Delta^2 \widehat{F}(d)$ is a constant computed once per interval, while $\Delta \widehat{F}(d)$ is updated recursively. This closed-form scheme ensures the predictor's overhead remains negligible while accurately modeling the high-frequency dynamics.

D Computing Efficiency

Table 8: **Comparison of methods in Cache Memory, MACs, Latency, and FLOPS on FLUX**

Method	VRAM Overhead (GB)↓	MACs (T)↓	Latency (s)↓	FLOPs (T)↓	Image Reward ↑
[dev]: 50 steps	0.62	1859.62	23.24	3726.87	0.99 (+0.0%)
ToCa($\mathcal{N}$=8, $\mathcal{R}$=75%)	11.93(+11.31GB)	414.88	12.39	829.86	0.95 (-4.0%)
DuCa($\mathcal{N}$=8, $\mathcal{R}$=70%)	3.65(+3.03GB)	428.86	9.40	858.27	0.94 (-5.1%)
TeaCache(l=1.0)	**0.69**(+0.07GB)	409.43	7.07	820.55	0.84 (-15.2%)
TaylorSeer($\mathcal{N}$=6, O=2)	7.66(+7.04GB)	<u>372.38</u>	<u>6.73</u>	<u>746.28</u>	<u>1.00</u> (+1.0%)
FreqCa	<u>0.80</u> (+0.18GB)	**372.25**	**5.19**	**746.03**	**1.01** (+2.0%)

- **Note:** All methods inherit baseline memory optimizations (e.g., FlashAttention). ToCa is incompatible with these, hence its higher reported cache usage. **Actual cache memory usage** for each method should be computed as: `VRAM Overhead − 0.62 GB`.

To complement the empirical results, we provide a concise theoretical analysis of the computational and memory efficiency of FreqCa. Specifically, we derive the space and time complexity of both conventional layer-wise caching and our proposed method, aiming to offer intuition and supporting evidence for the observations in Tab. 8, rather than introducing additional algorithmic contributions.

Notation Definition. Let $\mathcal{M}$ denote the memory size of a single feature map tensor, defined as $\mathcal{M} = B \times S \times D \times P$, where B is batch size, S is sequence length, D is hidden dimension, and P is precision (bytes). Let L be the network depth (number of blocks), and m be the order of the prediction solver.

D.1 Memory Complexity: $\mathcal{O}(L)$ vs. $\mathcal{O}(1)$.

Layer-wise Caching (e.g., TaylorSeer, ToCa). Conventional methods necessitate caching feature maps at each layer to compute local derivatives or reuse historical states. Typically, outputs from both the Attention and MLP modules are cached ($N = 2$). To support an m-th order predictor, the system must retain $m + 1$ historical states per module. The total memory overhead $\mathcal{K}_{\text{layer}}$ is derived as:

$$\mathcal{K}_{\text{layer}} = \sum_{l=1}^{L} N \cdot (m+1) \cdot \mathcal{M} = L \cdot N \cdot (m+1) \cdot \mathcal{M}. \tag{18}$$

For FLUX.1-dev ($L = 57, N = 2, m = 2$), this requires storing $57 \times 2 \times 3 = \mathbf{342}$ feature map units. This linear scaling $\mathcal{O}(L)$ explains the prohibitive VRAM overheads observed in Tab. 8 for methods like ToCa (+11.31 GB) and TaylorSeer (+7.04 GB).

Cross-layer Caching (FreqCa). Our approach operates on the Cumulative Residual Feature (CRF), which aggregates transformations across all layers. We apply a frequency-decoupled strategy: reused low-frequency components directly, requiring **1 cache unit**. perform polynomial prediction on high-frequency components, requires $m + 1$ **cache units**.

The total memory overhead $\mathcal{K}_{\text{FreqCa}}$ is:

$$\mathcal{K}_{\text{FreqCa}} = (1 + (m+1)) \cdot \mathcal{M}. \tag{19}$$

For the same setting ($m = 2$), FreqCa requires only $1 + 3 = \mathbf{4}$ cache units. Crucially, this equation is independent of L, achieving constant space complexity $\mathcal{O}(1)$. The relative cache-memory footprint is::

$$R = \frac{\mathcal{K}_{\text{FreqCa}}}{\mathcal{K}_{\text{layer}}} = \frac{m+2}{L \cdot N \cdot (m+1)} \approx \frac{4}{342} \approx 1.17\%. \tag{20}$$

This derivation perfectly aligns with the empirical measurement where FreqCa requires only +0.18GB VRAM.

D.2 Memory Access Cost (MAC) and Latency.

Theoretical FLOPs often fail to predict actual latency due to the "Memory Wall" in modern GPUs. We analyze the Memory Access Cost (MAC) during a skipped step.

Layer-wise Caching. Must read and write cache tensors at every layer l.

$$\text{MAC}_{\text{layer}} \propto 2 \cdot L \cdot \mathcal{M} \quad (\text{Read + Write per layer}). \tag{21}$$

This high IO frequency saturates memory bandwidth, causing the GPU compute units to idle waiting for data. This explains why TaylorSeer (6.73s) is significantly slower than FreqCa (5.19s) despite having comparable FLOPs.

Cross-layer Caching (FreqCa). Reads the input $\mathbf{x}_t$ once and writes the predicted $\mathbf{z}_t$ once at the network level.

$$\text{MAC}_{\text{FreqCa}} \propto 2 \cdot \mathcal{M} \quad (\text{Read + Write once}). \tag{22}$$

By reducing memory transactions by a factor of L, FreqCa shifts the workload from IO-bound to Compute-bound, allowing for near-linear latency scaling with respect to FLOPs reduction.

D.3 Computational Complexity: FLOPs Analysis.

We define $\mathcal{C}$ as the number of floating-point operations required for a single time step. To understand the efficiency gains, we decompose the costs for the full model, conventional layer-wise acceleration, and our FreqCa.

Full Forward Pass ($\mathcal{C}_{\text{full}}$). A standard DiT step involves passing features through L transformer blocks. Each block relies heavily on General Matrix Multiplications (GEMMs) in Self-Attention

and MLP layers. Let S be the sequence length and D be the hidden dimension. The complexity is dominated by the quadratic scaling with respect to D:

$$\begin{aligned} \mathcal{C}_{\text{full}} &\approx L \cdot \left(\underbrace{4SD^2}_{\text{Attention Proj.}} + \underbrace{2S^2D}_{\text{Attn Score}} + \underbrace{8SD^2}_{\text{MLP (SwiGLU)}} \right) \\ &\approx \mathcal{O}(L \cdot S \cdot D^2). \end{aligned} \tag{23}$$

For FLUX.1-dev ($D = 3072, L = 57$), these GEMM operations constitute the vast majority of the computational load.

Conventional Layer-wise Prediction ($\mathcal{C}_{\text{layer}}$). Methods like TaylorSeer or standard caching skip the GEMMs during accelerated steps. However, they must still perform prediction (extrapolation) at *every* layer to generate the input for the next block or to update the cache. Assuming an m-th order polynomial predictor (element-wise ops), the cost is:

$$\begin{aligned} \mathcal{C}_{\text{layer}} &= \sum_{l=1}^{L} (m+1) \cdot S \cdot D = L \cdot (m+1) \cdot S \cdot D \\ &\approx \mathcal{O}(L \cdot S \cdot D). \end{aligned} \tag{24}$$

While $\mathcal{C}_{\text{layer}} \ll \mathcal{C}_{\text{full}}$ (because $D \ll D^2$), the cost still scales **linearly with network depth** L. For deep models (large L), this cumulative overhead is non-negligible and, more importantly, incurs significant memory I/O overhead as analyzed in the previous section.

FreqCa Prediction ($\mathcal{C}_{\text{FreqCa}}$). In contrast, FreqCa performs prediction only *once* at the network level (on the accumulated CRF). It does not iterate through layers.

$$\mathcal{C}_{\text{FreqCa}} = 1 \cdot (m+1) \cdot S \cdot D \approx \mathcal{O}(S \cdot D). \tag{25}$$

Crucially, this complexity is **independent of depth** L and scales linearly with D.

Comparative Ratio. Comparing the prediction overheads directly:

$$\frac{\mathcal{C}_{\text{FreqCa}}}{\mathcal{C}_{\text{layer}}} = \frac{(m+1) \cdot S \cdot D}{L \cdot (m+1) \cdot S \cdot D} = \frac{1}{L}. \tag{26}$$

FreqCa reduces the prediction computation by a factor of L (e.g., $57\times$ reduction for FLUX.1). Furthermore, comparing against the full pass:

$$\frac{\mathcal{C}_{\text{FreqCa}}}{\mathcal{C}_{\text{full}}} \approx \frac{m+1}{L \cdot D} \approx 1.7 \times 10^{-5} \ll 0.01\%. \tag{27}$$

This rigorous derivation proves that FreqCa minimizes the computational overhead to the theoretical limit ($\mathcal{O}(1)$ w.r.t depth), ensuring that the speedup is bounded strictly by the skipping interval $\mathcal{N}$.

D.4 Theoretical Speedup Comparison.

Finally, we translate the computational costs into theoretical speedup. We define the skipping interval $\mathcal{N}$ as the period of the acceleration cycle: for every $\mathcal{N}$ consecutive timesteps, the model performs one full forward pass to update the cache, followed by $\mathcal{N} - 1$ prediction steps using cached features.

The average computational cost per timestep $\bar{\mathcal{C}}$ is thus a weighted sum:

$$\bar{\mathcal{C}} = \frac{1}{\mathcal{N}} \mathcal{C}_{\text{full}} + \left(1 - \frac{1}{\mathcal{N}}\right) \mathcal{C}_{\text{pred}}. \tag{28}$$

We define the theoretical speedup $\mathcal{S}$ as the ratio between the baseline cost (full computation at every step) and the amortized cost:

$$\mathcal{S} = \frac{\mathcal{C}_{\text{full}}}{\bar{\mathcal{C}}} = \frac{\mathcal{N}}{1 + (\mathcal{N} - 1) \cdot \underbrace{\frac{\mathcal{C}_{\text{pred}}}{\mathcal{C}_{\text{full}}}}_{\text{Overhead Ratio}}}. \tag{29}$$

This formulation shows that the speedup is upper-bounded by $\mathcal{N}$, and the deviation from the ideal linear scaling is entirely determined by the *overhead ratio*.

Layer-wise Speedup ($\mathcal{S}_{\text{layer}}$). Let S denote the token sequence length and $\mathcal{N}$ the caching interval. For an m-th order layer-wise predictor, the prediction cost scales as $\mathcal{C}_{\text{layer}} = \mathcal{O}(L(m+1)SD)$. Under the simplified cost model in which projection and MLP operations dominate the full forward pass, $\mathcal{C}_{\text{full}} = \mathcal{O}(LSD^2)$. The prediction-to-full-computation cost ratio therefore scales as

$$r_{\text{layer}} \equiv \frac{\mathcal{C}_{\text{layer}}}{\mathcal{C}_{\text{full}}} = \mathcal{O}\left(\frac{m+1}{D}\right). \tag{30}$$

Using one full-computation step followed by $\mathcal{N}-1$ prediction steps, the corresponding FLOPs-based speedup is approximately

$$\mathcal{S}_{\text{layer}}^{\text{FLOPs}} \approx \frac{\mathcal{N}}{1+(\mathcal{N}-1)r_{\text{layer}}}. \tag{31}$$

Although the prediction cost is small relative to a full forward pass, layer-wise prediction still accesses cache tensors and executes operations throughout the network. The resulting memory traffic and kernel-launch overhead can limit latency gains. Consequently, the measured latency speedup can be substantially lower than the FLOPs-based speedup, as observed in our experiments.

FreqCa Speedup ($\mathcal{S}_{\text{FreqCa}}$). FreqCa performs prediction once on the CRF rather than separately at each layer. Its prediction cost scales as $\mathcal{C}_{\text{FreqCa}} = \mathcal{O}((m+1)SD)$, giving

$$r_{\text{FreqCa}} \equiv \frac{\mathcal{C}_{\text{FreqCa}}}{\mathcal{C}_{\text{full}}} = \mathcal{O}\left(\frac{m+1}{LD}\right). \tag{32}$$

Under the same per-feature prediction-cost model, this removes the factor of L associated with layer-wise prediction:

$$\frac{\mathcal{C}_{\text{FreqCa}}}{\mathcal{C}_{\text{layer}}} \approx \frac{1}{L}. \tag{33}$$

The corresponding FLOPs-based speedup is approximately

$$\mathcal{S}_{\text{FreqCa}}^{\text{FLOPs}} \approx \frac{\mathcal{N}}{1+(\mathcal{N}-1)r_{\text{FreqCa}}}. \tag{34}$$

When $(\mathcal{N}-1)r_{\text{FreqCa}} \ll 1$, this approaches

$$\mathcal{S}_{\text{FreqCa}}^{\text{FLOPs}} \approx \mathcal{N}. \tag{35}$$

This analysis explains how CRF-level prediction reduces computational overhead and supports near-$\mathcal{N}$ FLOPs-based speedup when that overhead is negligible. Actual latency gains additionally depend on memory traffic, implementation overhead, warm-up steps, and computation outside the accelerated backbone.

E More Supplementary Experiment Results

E.1 U-Net architectures

We further include a supplementary evaluation on a U-Net-based model (SDXL) to assess the generalization of FreqCa beyond Transformers. As shown in Tab. 9, FreqCa consistently outperforms DeepCache in both semantic alignment (ImageReward) and perceptual quality (LPIPS). At a standard interval of $N = 2$, FreqCa achieves higher ImageReward (0.47 vs. lower baseline) and lower LPIPS (0.21), and the advantage becomes more pronounced at higher acceleration ratios. In particular, at $N = 4$, while DeepCache degrades significantly (ImageReward drops to 0.33), FreqCa maintains relatively stable performance (0.42). These results suggest that FreqCa generalizes well to U-Net architectures by leveraging frequency-aware feature dynamics.

Table 9: Performance comparison across different skip intervals using ImageReward and LPIPS.

Method	Interval (N)	ImageReward ↑	LPIPS ↓
Original	1	0.52	–
DeepCache	2	0.45	0.26
FreqCa	2	**0.47**	**0.21**
DeepCache	3	0.42	0.31
FreqCa	3	**0.47**	**0.27**
DeepCache	4	0.33	0.36
FreqCa	4	**0.42**	**0.29**



Figure 10: Qualitative visualization results on HunyuanVideo under 5.56× acceleration.

E.2 Video Diffusion Models

While the main experiments of FreqCa focus on image generation and image editing, we further conduct a supplementary evaluation on HunyuanVideo to preliminarily explore its applicability to video generation. Under a 480p, 65-frame, and $5.56\times$ acceleration setting, as shown in Tab. 10, FreqCa achieves performance comparable to TaylorSeer across VBench, PSNR, and LPIPS, while substantially reducing cache memory consumption (0.47 GB vs. 28.37 GB). These results suggest that FreqCa can maintain similar generation quality with significantly lower memory cost in video generation scenarios.

Table 10: **Video acceleration results on HunyuanVideo** under $5.56\times$ acceleration.

Method	VBench↑	PSNR↑	LPIPS↓	Cache Mem. (GB)↓
TeaCache	78.32	16.07	0.44	**0.35**
TaylorSeer	**79.43**	16.24	**0.30**	28.37
FreqCa	79.40	**16.38**	0.31	0.47

E.3 Few-Step Diffusion Models

We further conduct supplementary experiments to examine the compatibility of FreqCa with few-step diffusion settings. Few-step inference reduces the number of sampling steps and thus changes the sampling trajectory, whereas FreqCa reduces the denoising cost per step along a given trajectory. Therefore, the two strategies are complementary rather than interchangeable.

Qwen-Image-Lightning. We first evaluate FreqCa on Qwen-Image-Lightning, an 8-step distilled model. As shown in Tab. 11, FreqCa consistently outperforms TaylorSeer under the same cache intervals. At $N = 4$, FreqCa maintains an ImageReward of 1.29, matching the original Lightning model, while TaylorSeer drops to 1.20. FreqCa also achieves lower LPIPS across all tested intervals, suggesting better preservation of perceptual quality under few-step inference.

Table 11: **Evaluation on Qwen-Image-Lightning** under different cache intervals.

Method	Interval	ImageReward↑	LPIPS↓
Lightning (8 steps)	1	1.29	–
TaylorSeer	2	1.27	0.28
FreqCa	2	**1.29**	**0.12**
TaylorSeer	3	1.24	0.35
FreqCa	3	**1.28**	**0.18**
TaylorSeer	4	1.20	0.42
FreqCa	4	**1.29**	**0.36**

Direct Step Compression. We also evaluate direct step compression by reducing Qwen-Image from 50 to 20 steps. As shown in Tab. 12, FreqCa remains effective under this compressed trajectory, achieving an ImageReward of 1.09 and LPIPS of 0.22 at interval $N = 5$, while TaylorSeer degrades substantially. This suggests that frequency-aware caching is more robust than uniform feature extrapolation under aggressive step reduction.

Table 12: **Evaluation under direct step compression** from 50 to 20 steps.

Method	Interval	ImageReward↑	LPIPS↓
Qwen-Image (20 steps)	1	1.19	0.00
TaylorSeer	5	-1.16	0.61
FreqCa	5	**1.09**	**0.22**

Comparison with Distilled Few-Step Models. Finally, we compare FreqCa with distilled few-step inference on Qwen-Image. As shown in Tab. 13, under the same $5.0\times$ speedup, applying FreqCa to the original 50-step model achieves higher ImageReward than the 10-step distilled model. The distilled baselines are based on alternative (non-Lightning) distillation pipelines. These results indicate that FreqCa serves as a complementary acceleration strategy that better preserves the behavior of the original model.

E.3.1 Pixel Diffusion Model

We include a supplementary evaluation on pixel diffusion using JiT-B/16 on a single NVIDIA A100 GPU. As shown in Tab. 14, FreqCa significantly reduces latency (35.77s → 14.06s) while maintaining

Table 13: **Comparison with distilled few-step inference** on Qwen-Image.

Method	Speed↑	ImageReward↑
Qwen-Image (50 steps)	1.0×	**1.25**
Distill (Default Setting 20 steps)	2.5×	1.23
Distill (10 steps)	5.0×	1.02
Qwen-Image (50 steps) + **FreqCa**	5.0×	<u>1.20</u>

competitive FID and improving IS. Compared with prior methods under similar FLOPs, FreqCa achieves a favorable speed-quality trade-off. Latency and FLOPs are averaged over one batch of 256 images. This experiment is provided as an additional validation.

Table 14: Pixel diffusion results on JiT-B/16.

Method	Latency (s)↓	FLOPs (T)↓	FID↓	IS↑
Baseline	35.77	2413.21	**3.62**	270.09
FORA	26.62	531.54	7.05	243.71
TaylorSeer	31.18	531.54	31.40	90.51
FreqCa	**14.06**	**524.12**	<u>4.85</u>	**278.08**

Table 15: **Solver and schedule compatibility** on Qwen-Image with 50 steps and $N = 10$.

Solver / Schedule	Interval	ImageReward↑
Euler (default)	1	**1.25**
Euler (default) + FreqCa	10	1.02
Euler (cosine) + FreqCa	10	0.93
Euler (shift = 0.5) + FreqCa	10	0.96
Euler (shift = 2.0) + FreqCa	10	<u>1.05</u>
Heun (2nd-order) + FreqCa	10	<u>1.05</u>

E.3.2 Compatibility with Solvers.

We provide a supplementary study on solver and schedule variants to examine whether FreqCa is sensitive to the sampling configuration. Since FreqCa reduces the per-step denoising cost while the solver controls the sampling trajectory, these two components are largely complementary. As shown in Tab. 15, under Qwen-Image with 50 steps and interval $N = 10$, FreqCa remains effective across the tested variants, with Heun and Euler (shift = 2.0) both achieving an ImageReward of 1.05. These results suggest that FreqCa is not tightly coupled to the default Euler setting, although a broader solver study is left for future work.

E.3.3 Comparison with Frequency-Related Caching Methods

FreqCa is not the first work to exploit frequency-related observations for diffusion acceleration. A closely related method is FasterCache [24], which also introduces frequency-domain analysis in its caching design. However, FasterCache should not be viewed as a unified frequency-aware caching framework for general DiT features. Instead, it addresses different redundancies separately. For cross-timestep attention features, FasterCache mainly operates in the original feature space and corrects direct reuse using a temporal bias term:

$$F_{t-1} \approx F_t + \big(F_t - F_{t+2}\big)w(t), \tag{36}$$

which can be interpreted as a time-domain feature extrapolation. Its frequency-domain treatment is introduced primarily in the CFG branch, where low- and high-frequency biases between conditional and unconditional outputs are modeled to approximate the unconditional branch at later timesteps. Thus, the role of frequency in FasterCache is local and branch-specific, serving mainly as a correction mechanism for CFG redundancy.

In contrast, FreqCa takes frequency decomposition as the core principle of the caching mechanism. Given a cached feature z_t, we decompose it into low- and high-frequency components:

$$z_t^{\text{low/high}} = T^{-1}\big(T(z_t) \odot M_{\text{low/high}}\big)\,, \tag{37}$$

where T denotes the frequency transform and $M_{\text{low/high}}$ denotes the corresponding frequency masks. We then apply different update rules to the two components: low-frequency features are directly reused due to their higher temporal similarity, while high-frequency features are predicted to better preserve rapidly changing details:

$$\hat{z}_t^{\text{low}} = z_{t_{\text{prev}}}^{\text{low}}, \qquad \hat{z}_t^{\text{high}} = z_{t_{\text{prev}}}^{\text{high}} + \alpha_1 \Delta F^{(1)} + \alpha_2 \Delta F^{(2)}. \tag{38}$$

The final cached feature is reconstructed as

$$\hat{z}_t = \hat{z}_t^{\text{low}} + \hat{z}_t^{\text{high}}. \tag{39}$$

This design unifies the cache-then-reuse and cache-then-forecast paradigms within a single frequency-aware framework. Moreover, by caching the Cumulative Residual Feature (CRF) instead of layer-wise features, FreqCa reduces cache complexity from $\mathcal{O}(L)$ to $\mathcal{O}(1)$, making it more scalable in high-resolution and high-speedup settings.

We further compare FasterCache and FreqCa on Qwen-Image. As shown in Tab. 16, FasterCache improves efficiency over the default baseline but suffers a substantial drop in ImageReward under tested settings. In contrast, FreqCa achieves higher speedup while preserving much stronger generation quality, highlighting the benefit of using frequency separation as a general caching principle rather than a branch-specific correction.

Table 16: **Comparison with FasterCache** on Qwen-Image.

Method	**FLOPs (T)**↓	**Speedup**↑	**ImageReward**↑
Qwen-Image	12917.56	1.00×	**1.25**
FasterCache (s=1, i=3, α=0.3)	4334.78	2.98×	0.44
FasterCache (s=3, i=5, α=0.3)	3996.38	3.23×	0.54
FasterCache (s=3, i=8, α=0.3)	3796.14	3.40×	0.39
FasterCache (s=15, i=2, α=0.3)	5337.40	2.42×	0.68
FreqCa (N=6)	**2584.70**	5.00×	<u>1.20</u>
FreqCa (N=10)	<u>1809.38</u>	7.14×	1.02

F More Ablation Study

F.1 Decomposition Method and Prediction Order

We further study the impact of prediction order for low- and high-frequency components. As shown in Fig. 11, different order combinations lead to noticeable quality differences. Empirically, using zeroth-order prediction (direct reuse) for low-frequency components and second-order prediction for high-frequency components achieves the best balance between structural consistency and detail fidelity. Applying higher-order prediction to low-frequency components tends to introduce instability, while lower-order prediction is insufficient to capture rapidly varying high-frequency details.

We further fix zeroth-order prediction for low-frequency components and compare different Hermite orders for high-frequency prediction. As shown in Tab. 17, second-order consistently outperforms first-order under the interval $N = 10$. Based on these observations, we adopt the combination of zeroth-order reuse for low-frequency components and second-order prediction for high-frequency components as the default setting.

F.2 Frequency Split Threshold

Sensitivity to the Fixed Cutoff. We study the sensitivity of the cutoff ratio τ for frequency separation, with $\tau = 0.1$ as the default without per-sample or per-task tuning. As shown in Tab. 18, performance varies moderately over $\tau \in [0.1, 0.4]$, whereas a cutoff of 0.05 reduces quality. A smaller cutoff assigns more low- and mid-frequency components to extrapolation, potentially affecting structural fidelity; a larger cutoff assigns more components to direct reuse, potentially affecting fine details. The results support a broad operating range rather than a uniquely optimal cutoff.



Figure 11: QwenImage ablation results showing image quality under different frequency prediction configurations and acceleration ratios. $(x, y) = (low, high)$ prediction orders.

Table 17: Comparison of Hermite predictor orders under interval $N = 10$.

Order	Clip Score↑	Image Reward↑	PSNR↑	SSIM↑	LPIPS↓
1	34.68	0.99	28.90	0.66	0.42
2	34.84	1.04	28.95	0.67	0.41

Adaptive Cutoff Implementation. We additionally evaluate an adaptive strategy based on cumulative spectral energy. At each full-computation step, we compute the spectrum of the CRF feature independently for each sample, accumulate its energy in increasing order of radial frequency, and select the smallest frequency radius covering 92% of the total energy. The selected cutoff is clipped to $[0.1, 0.4]$, using the same radius normalization as the fixed-cutoff mask. Subsequent cache steps reuse the latest selected cutoff until the next full-computation step.

Comparison with the Fixed Cutoff. Table 19 reports the fixed-versus-adaptive comparison. The adaptive strategy obtains an ImageReward of 0.952, compared with 0.993 for the fixed cutoff, although the difference is not statistically significant ($p = 0.077$). Across sample-level full-computation

Table 18: Ablation of cutoff ratio for frequency separation.

Cutoff Ratio	CLIPScore↑	ImageReward↑	LPIPS↓
0.05	34.54	0.94	0.43
0.1	34.84	1.04	0.41
0.2	35.05	1.04	0.39
0.3	35.06	1.05	0.39
0.4	35.26	1.06	0.40

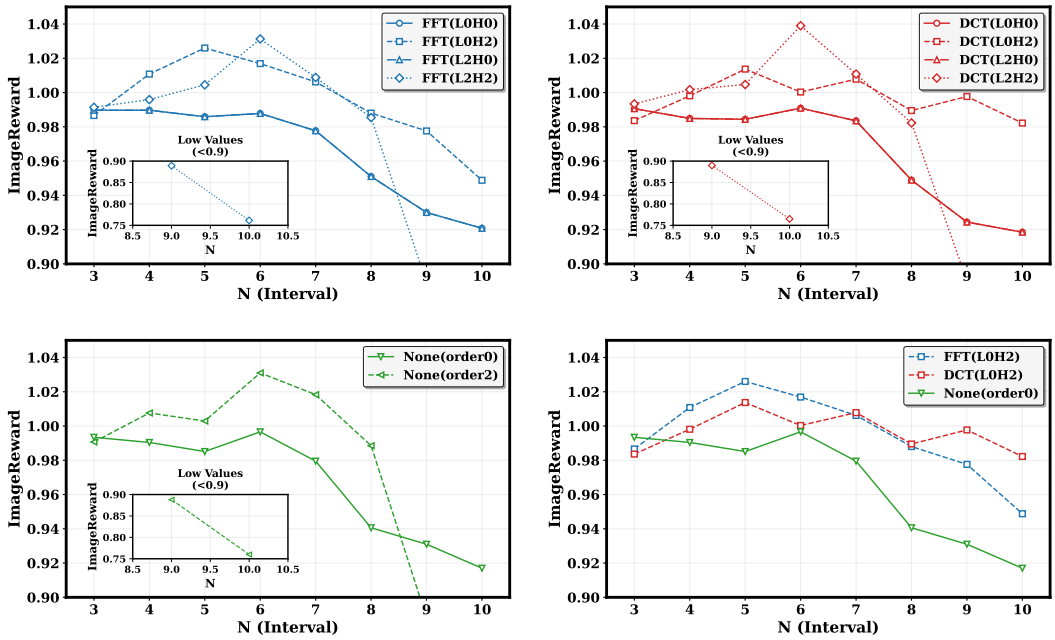


Figure 12: The ImageReward scores of FLUX.1-dev's decomposition strategies (FFT, DCT, no-decomposition) paired with different frequency prediction approaches are presented here. This content includes the optimal prediction method for each decomposition strategy where low-frequency reuse and high-frequency prediction apply to FFT and DCT, and direct reuse applies to the no-decomposition (None) strategy.

updates, 57.8% of selected cutoffs equal 0.1 and 79.0% fall within $[0.1, 0.15)$, showing that this adaptive rule usually remains close to the fixed default.

Table 19: Comparison of fixed and energy-based adaptive cutoffs.

Method	**Cutoff Selection**	**ImageReward↑**
Fixed	$\tau = 0.1$	**0.993**
Adaptive	92% energy, clipped to $[0.1, 0.4]$	0.952

One possible explanation is that spectral energy does not directly measure cache-reuse error or Hermite-prediction error. Increasing the cutoff may assign perceptually important components, including edges, textures, and text, to direct reuse, potentially degrading detail fidelity. The adaptive implementation requires energy computation, sorting, and cumulative summation, adding approximately 0.34 ms per decomposition call. Given the additional overhead and the absence of an improvement in this comparison, we retain the fixed cutoff for simplicity and efficiency. This result concerns the evaluated energy-based rule and does not rule out benefits from other adaptive criteria.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction describe frequency-aware caching and Cumulative Residual Feature (CRF) caching. Section 3 explains the method, while Section 4 and Appendices E–F evaluate its effectiveness and scope across the tested models and tasks.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Appendix E describes the scope of the supplementary evaluations, including the preliminary video experiments and the limited range of tested solvers. Appendix F.2 discusses the limitations of the evaluated energy-based adaptive cutoff and explicitly limits the conclusion to that rule.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [No]

Justification: Appendix D provides simplified memory, computational-cost, and speedup analyses. However, the current presentation does not fully specify the assumptions needed for all complexity estimates and their interpretation in terms of measured latency.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: Section 4 and Appendices B–C and F describe the models, benchmarks, evaluation metrics, caching strategy, and ablations. The public repository at `https://github.com/Tammytcl/FreqCa` provides the implementation, installation instructions, inference commands, and evaluation entry points for the main image experiments.

Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: The implementation is publicly available at `https://github.com/Tammytcl/FreqCa`, with installation instructions and commands for inference and evaluation. The experiments use existing models and benchmarks, which are obtained separately from their upstream providers subject to their access terms.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 4 and Appendices B–C and F specify the model configurations, benchmarks, image resolutions, frequency transforms, prediction strategy, and cutoff settings; the repository documents additional runtime parameters. The method is training-free, so optimizer and training-data settings are not applicable.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

Justification: The main experimental comparisons are reported without confidence intervals or repeated-run uncertainty estimates. Appendix F.2 reports a significance test for the fixed-versus-adaptive cutoff comparison, but this limited analysis does not establish statistical significance for the main results.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [No]

   Justification: Appendix B.2 reports the GPU types, and the experimental tables report inference latency and selected memory measurements. The paper does not provide a complete accounting of per-experiment compute usage or the total computational cost of the research project.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [No]

    Justification: The paper focuses on inference efficiency and does not provide a dedicated discussion of positive and negative societal impacts. Relevant considerations include lower computational requirements and the potential misuse of more efficient image generation and editing.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The work releases acceleration code rather than new pretrained model weights or a newly collected dataset. The evaluated models and benchmarks are obtained separately from their upstream providers.

    Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [No]

Justification: The paper credits the principal models, benchmarks, and methods, and the repository includes a `THIRD_PARTY_NOTICES.md` file for upstream code. However, the paper and accompanying documentation do not explicitly enumerate the licenses and terms of every external model, dataset, and software asset used in the experiments.

Guidelines:

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

13. **New assets**

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

Answer: [Yes]

Justification: The newly released asset is the FreqCa implementation at `https://github.com/Tammytcl/FreqCa`. Its README documents installation, supported models, inference and evaluation commands, and cache parameters; the repository also includes a source-code license and third-party notices.

Guidelines:

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

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

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

Answer: [N/A]

Justification: The study does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: The study does not involve research with human subjects; IRB approval is therefore not applicable.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs are not used as an important, original, or non-standard component of the core method. Assistance limited to writing, editing, or formatting does not fall within the scope of this question.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.