Z-Cache: Accelerating Diffusion Transformers via Self-Reflection

Zegang Cheng[1,*] **Zhikai Wang**[2,*] **Jiacheng Liu**[1] **Xiaobing Tu**[3]
Chang Zhou[1,§] **Junjie Chen**[1,†] **Yue Ma**[1] **Zhiyuan Ma**[4]
Jinkui Ren[3] **Likai Zou**[3] **Linfeng Zhang**[1,‡]

[1]Shanghai Jiao Tong University [2]Southern University of Science and Technology
[3]Alibaba Cloud [4]Huazhong University of Science and Technology

[*]Equal contribution. [†]Work done during an internship. [‡]Corresponding author.
[§]The correct name is Chang Zou; Chang Zhou was entered in error.



Figure 1: Images sampled by 6.22× accelerated FLUX.1-dev with *Z-Cache*.

Abstract

Diffusion transformers have become the most powerful models for visual generation, but still suffer from massive computation costs. To solve this problem, feature caching has been proposed to cache the features of diffusion models in the previous computation steps and then reuse them in the following caching steps, which brings significant acceleration but also degradation in generation quality. To address this problem, this paper proposes Z-cache as a feature caching method that can maintain high-quality generation through self-reflection. Concretely, we observe that *the error from feature caching tends to be sharply reduced after each full computation.* Based on this observation, *Z-Cache* is designed to first predict the features in the future caching steps and then perform a full computation. After that, *Z-Cache* returns to the caching steps and re-predicts them based on the previous and the current computation steps, which brings correction in features. Experiments demonstrate that with *Z-Cache*, diffusion transformers achieve comparable generation quality to the original model but with faster inference speed, for instance, **6.22**× acceleration on FLUX-dev for text-to-image generation. *Our code is released on GitHub.*

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

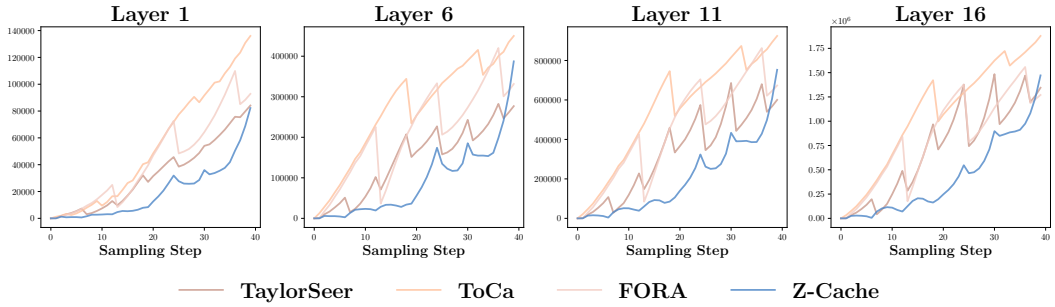


Figure 2: **Visualization of the per-layer L1 error from feature caching.** It is observed that there exists a periodic decline in L1 error, which is caused by the full-computation step in feature caching. The proposed *Z-Cache* utilizes this phenomenon and derives fewer cumulative errors.

1 Introduction

Diffusion models [Ho et al., 2020, Song et al., 2021, Liu et al., 2023] have become a dominant paradigm for high-quality visual generation, but their iterative denoising process remains computationally expensive, especially for high-resolution image and video synthesis. This cost limits practical deployment and motivates a broad set of acceleration methods. Among them, feature caching [Selvaraju et al., 2024, Qiu et al., 2025, Zou et al., 2024b, Liu et al., 2024, 2025c, Feng et al., 2025, Zheng et al., 2025b, Liu et al., 2025d] is particularly attractive because it is training-free: instead of running the full denoising network at every timestep, it reuses or predicts intermediate hidden states across nearby timesteps.

The key assumption behind feature caching is temporal redundancy. Hidden states at adjacent denoising steps are often similar, allowing expensive transformer blocks to be skipped with minor quality loss. However, this assumption becomes fragile under aggressive acceleration. As the cache interval grows, single-anchor reuse or forecasting accumulates hidden-state errors, which can lead to semantic drift, local artifacts, and degraded prompt fidelity. This creates a central trade-off: larger cache windows improve efficiency but increasingly deviate from the true denoising trajectory.

We start from a simple empirical observation. As shown in Figure 2, cache-induced feature error tends to grow within each cache window, but drops sharply when a scheduled full-computation step is performed. This suggests that full-computation anchors partially pull the accelerated trajectory back toward the original denoising path. Rather than only using a full step as the starting point for the next cache window, we ask whether the newly obtained future anchor can also be used to correct the provisional states inside the previous active window.

Based on this observation, we propose *Z-Cache*, a training-free and predictor-agnostic reflection framework for diffusion feature caching. For each active cache window, *Z-Cache* first uses a base cache predictor to forecast intermediate hidden states from a past full-computation anchor and runs the corresponding provisional denoising transitions. After reaching the next scheduled full-computation anchor, *Z-Cache* revisits the same window, estimates each intermediate hidden state with both the past and future anchors, and re-samples the window trajectory before committing the reflected window-end latent. Therefore, *Z-Cache* is not a post-hoc smoothing operation over already generated samples. Instead, it is an in-loop trajectory correction mechanism that changes the subsequent denoising path. Extensive experiments demonstrate the effectiveness of Z-Cache in text-to-image, text-to-video, and image editing tasks, as well as its compatibility with other feature caching methods and other diffusion acceleration methods such as step distillation and quantization.

Our contributions are summarized as follows:

- We identify a cache-induced error pattern in diffusion feature caching: hidden-state errors tend to accumulate within a cache window but are sharply reduced after scheduled full-computation anchors, which motivates using future anchors to correct the previous active cache window.
- We propose *Z-Cache*, a training-free cache-then-check framework that performs dual-anchor reflection and active-window re-sampling. Unlike post-hoc smoothing, *Z-Cache* revises the latent trajectory inside the denoising loop and continues sampling from the reflected window-end state.

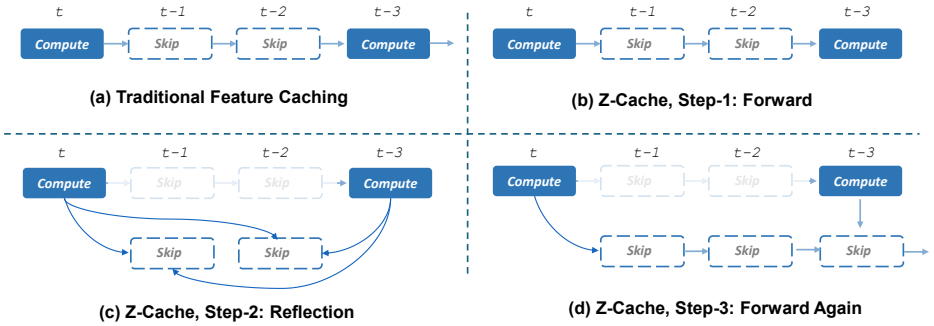


Figure 3: **Pipeline of Z-Cache.** The figure uses the reverse-diffusion notation $t, t-1, \ldots$ for visualization, while the text uses a sampling-order index that increases along denoising. Traditional feature caching forecasts an active window from a past full-computation anchor and directly moves to the next window. In contrast, Z-Cache treats the window as provisional: after obtaining the next scheduled full-computation anchor, it revisits the same window, constructs dual-anchor hidden estimates, re-samples the window trajectory, and commits the reflected window-end latent. Z-Cache therefore performs in-loop trajectory correction without adding extra full-computation steps.

- We establish *Z-Cache* as a plug-and-play and composable acceleration module. It can be attached to compatible feature-cache predictors, generalizes beyond free-form generation to image editing and video synthesis, and remains compatible with orthogonal acceleration techniques such as few-step distillation and low-bit quantization.

2 Related Works

Feature caching Feature caching accelerates diffusion sampling by reusing or predicting intermediate activations. It spans U-Net models [Li et al., 2023a, Ma et al., 2024], DiT representation reuse [Selvaraju et al., 2024, Chen et al., 2024c], adaptive timestep caching [Liu et al., 2024], token-wise or dual feature caching [Zou et al., 2024a,b], and predictors such as TaylorSeer [Liu et al., 2025c], SpeCa [Liu et al., 2025d], and MagCache [Ma et al., 2025]. SenCache [Haghighi and Alahi, 2026] adapts reuse using sensitivity estimates, D2Cache [Liu et al., 2026] improves temporal prediction with second-order deltas, and ResCa [Fang et al., 2026] updates tokens using proxy-guided residuals.

Positioning of *Z-Cache* *Z-Cache* performs *cache-then-check* correction: after observing the next scheduled full-computation anchor on the provisional trajectory, it combines past- and future-anchor predictions, replays the closed provisional window, and continues from the reflected endpoint. This complements compatible cache policies and predictors while preserving the full-computation schedule; composition requires compatible cached representations and state updates. Appendix A.1 provides a detailed taxonomy and comparisons.

3 Methodology

3.1 Preliminary

3.1.1 Diffusion sampling.

Diffusion models [Ho et al., 2020, Song et al., 2021] generate a clean sample by iteratively denoising a noisy latent. In a standard DDPM-style parameterization, one reverse step is written as

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\tau_\theta(x_t, t)\right), \beta_t \mathbf{I}\right), \tag{1}$$

where τ_θ is the denoising network and $\alpha_t, \bar{\alpha}_t, \beta_t$ are scheduler coefficients. Modern samplers may use different update rules, but they share the same iterative structure. To avoid confusion between the diffusion timestep t and the order in which the sampler is executed, we use $i \in \{0, \ldots, S-1\}$ to denote the sampling-order index in the rest of this section, with larger i corresponding to later denoising. Let τ_i be the diffusion timestep used at sampling index i. We write one denoising update abstractly as

$$x_{i+1} = F_i(x_i, \mathcal{H}_i), \tag{2}$$

where $\mathcal{H}_i$ denotes the hidden features produced by the denoising network at x_i and τ_i.

3.1.2 Diffusion transformer hidden states.

A Diffusion Transformer (DiT) [Peebles and Xie, 2023] is composed of L transformer blocks, $\mathcal{G} = g_L \circ \cdots \circ g_1$, where each block contains attention and MLP submodules with timestep- and condition-dependent normalization. At sampling index i, the full denoising network produces a collection of block features

$$\mathcal{H}_i = \{h_i^\ell\}_{\ell=1}^L. \tag{3}$$

Feature caching accelerates inference by avoiding full recomputation of $\mathcal{H}_i$ at selected sampling indices and replacing it with a cached or predicted estimate.

3.2 Feature Caching

Let $\mathcal{A}_i$ be the set of sampling indices whose full hidden states have been cached before evaluating index i. A generic cache predictor $\mathcal{C}$ estimates the hidden features at a cached step by

$$\hat{\mathcal{H}}_i = \mathcal{C}(\{(j, \mathcal{H}_j) : j \in \mathcal{A}_i\}, i)\,. \tag{4}$$

This abstraction covers simple feature reuse, timestep-adaptive caching, polynomial forecasting, speculative prediction, and calibration-style predictors. For example, a naive fixed-interval cache method computes full features at an anchor a and reuses them for the following cached steps,

$$\hat{\mathcal{H}}_i = \mathcal{H}_a, \qquad a < i < b, \tag{5}$$

where $[a, b]$ is one cache window and $b - a$ is the cache interval. Larger intervals reduce the number of full denoising-network evaluations, but the approximation error $\|\mathcal{H}_i - \hat{\mathcal{H}}_i\|$ can accumulate inside the window and lead to semantic drift or local artifacts.

3.3 Z-Cache

Z-Cache is a training-free reflection wrapper for feature caching. It does not prescribe a specific cache predictor; instead, it can be applied on top of a base predictor $\mathcal{C}$ such as TaylorSeer, SpeCa, or MagCache. The key idea is to treat the states inside the current cache window as provisional until the next scheduled full-computation anchor is available. After obtaining that future anchor, *Z-Cache* revisits the active window, constructs dual-anchor hidden estimates, re-samples the window trajectory, and then continues denoising from the reflected state.

Active cache window. Consider one active cache window $[a, b]$ in sampling-order indices. Full hidden states are computed at the endpoint anchors a and b, while the intermediate indices $a < i < b$ are cached. Let $\mathcal{A}_a$ denote the anchor set available before entering this window, which includes the past anchor a. The window proceeds in three stages.

Stage 1: forecast from the past anchor. Before the future endpoint is known, the base cache predictor estimates each intermediate hidden state from the currently available anchors:

$$\hat{\mathcal{H}}_i^p = \mathcal{C}(\{(j, \mathcal{H}_j) : j \in \mathcal{A}_a\}, i)\,, \qquad a < i < b, \tag{6}$$

where the superscript p denotes the past-anchor prediction. These estimates define a provisional trajectory,

$$\bar{x}_a = x_a, \qquad \bar{x}_{i+1} = F_i(\bar{x}_i, \bar{\mathcal{H}}_i), \qquad a \le i < b, \tag{7}$$

where $\bar{\mathcal{H}}_a = \mathcal{H}_a$ and $\bar{\mathcal{H}}_i = \hat{\mathcal{H}}_i^p$ for $a < i < b$. After the provisional trajectory reaches the endpoint, the sampler performs the next scheduled full computation and obtains the future anchor $\mathcal{H}_b$. This full computation is part of the original cache schedule, so *Z-Cache* does not add extra scheduled full-computation steps.

Stage 2: dual-anchor reflection. Once $\mathcal{H}_b$ is available, *Z-Cache* constructs a future-anchor prediction for each intermediate index by reusing the same base predictor with the extended anchor set:

$$\hat{\mathcal{H}}_i^f = \mathcal{C}(\{(j, \mathcal{H}_j) : j \in \mathcal{A}_a \cup \{b\}\}, i), \qquad a < i < b, \tag{8}$$

where the superscript f denotes the future-anchor prediction. The reflected hidden state is the convex dual-anchor estimate

$$\tilde{\mathcal{H}}_i = (1 - w_i)\hat{\mathcal{H}}_i^p + w_i \hat{\mathcal{H}}_i^f, \qquad a < i < b. \tag{9}$$

The weight w_i is defined by the normalized progress inside the active window,

$$w_i = \frac{i - a}{b - a}, \qquad w_i \in [0, 1], \tag{10}$$

This schedule is hyperparameter-free once the cache interval is fixed. It trusts the past-anchor prediction near the beginning of the window and gradually shifts trust toward the future-anchor prediction near the endpoint.

Stage 3: active-window re-sampling. The reflection step is performed inside the denoising loop, not after a sample has already been finalized. The provisional latent states $\bar{x}_{a+1}, \ldots, \bar{x}_b$ are discarded, and the active window is re-sampled with the reflected hidden states:

$$\tilde{x}_a = x_a, \qquad \tilde{x}_{i+1} = F_i(\tilde{x}_i, \tilde{\mathcal{H}}_i), \qquad a \le i < b, \tag{11}$$

where $\tilde{\mathcal{H}}_a = \mathcal{H}_a$ and $\tilde{\mathcal{H}}_i$ is given by Eq. (9) for $a < i < b$. The transition at a is reused. After re-sampling, *Z-Cache* commits $\tilde{x}_{b+1} = F_b(\tilde{x}_b, \mathcal{H}_b)$ using the retained right-anchor features and updates the cache accordingly (Appendix A.4). States before a have already been committed and are not modified, while states after $b + 1$ have not yet been generated. Thus, *Z-Cache* changes the subsequent denoising trajectory without creating inconsistency with finalized states outside the current window. This distinguishes *Z-Cache* from post-hoc hidden-state smoothing: the reflected hidden estimates affect later outputs because they are used to re-run the latent transitions inside the active window.

Local error propagation. We next give a local analysis explaining why active-window reflection can reduce trajectory error. Let the full, non-accelerated trajectory satisfy

$$x_{i+1}^\star = F_i(x_i^\star, \mathcal{H}_i^\star), \tag{12}$$

and let an accelerated method use a hidden estimate $\bar{\mathcal{H}}_i$ and latent $\bar{x}_i$. Define the latent error and hidden approximation error as

$$\delta_i = \|\bar{x}_i - x_i^\star\|, \qquad \epsilon_i = \|\bar{\mathcal{H}}_i - \mathcal{H}_i^\star\|. \tag{13}$$

Assume each local update is Lipschitz in both the latent state and the hidden state:

$$\|F_i(x, \mathcal{H}) - F_i(x', \mathcal{H}')\| \le L_i^x \|x - x'\| + L_i^H \|\mathcal{H} - \mathcal{H}'\|. \tag{14}$$

Then

$$\delta_{i+1} \le L_i^x \delta_i + L_i^H \epsilon_i. \tag{15}$$

Unrolling this recursion over one active window gives

$$\delta_b \le \left(\prod_{i=a}^{b-1} L_i^x \right) \delta_a + \sum_{i=a}^{b-1} \left(\prod_{j=i+1}^{b-1} L_j^x \right) L_i^H \epsilon_i, \tag{16}$$

where an empty product equals 1. Therefore, the error δ_b depends on the hidden-state errors accumulated at all earlier indices in the same window. This explains why correcting intermediate cached steps changes later latents: the sampler recursively propagates hidden-state errors through Eq. (2).

For the no-reflection base predictor, the hidden error at an intermediate index is

$$\epsilon_i^p = \|\hat{\mathcal{H}}_i^p - \mathcal{H}_i^\star\|. \tag{17}$$

Table 1: **Quantitative comparison on class-to-image generation** on ImageNet with DiT-XL/2. F-Speed denotes the FLOPs reduction relative to DDIM with 50 steps. Metrics marked with "-" are left unreported, as they were not provided in the original publications.

Method	Efficient Attention	Latency(s) ↓	FLOPs(T) ↓	F-Speed ↑	FID ↓	sFID ↓	Inception Score ↑	PSNR ↑	SSIM ↑	LPIPS ↓
DDIM-50 steps	✔	0.428	23.74	1.00×	2.32	4.32	241.25	∞	1.00	0.000
DDIM-12 steps	✔	0.128	5.70	4.17×	7.80	8.03	184.50	22.70	0.742	0.196
L2C(NFE=30)	✔	0.281	11.55	2.05×	2.61	-	237.83	-	-	-
SmoothCache($\alpha = 0.22$)	✔	0.251	8.57	2.77×	4.15	-	231.71	-	-	-
DDIM-10 steps	✔	0.115	4.75	5.00×	12.15	11.33	159.13	21.52	0.687	0.251
FORA ($\mathcal{N} = 5$)	✔	0.149	5.24	4.53×	6.58	11.29	193.01	21.85	0.692	0.235
ToCa ($\mathcal{N} = 6$)	✘	0.163	6.34	3.75×	6.55	7.10	189.53	17.66	0.572	0.360
DuCa ($\mathcal{N} = 5$)	✔	0.154	6.27	3.78×	6.06	6.72	198.46	16.65	0.543	0.390
TaylorSeer ($\mathcal{N} = 5$)	✔	0.195	5.24	4.53×	2.65	5.36	231.59	28.50	0.901	0.065
Z-Cache ($\mathcal{N} = 5$)	✔	0.213	5.25	4.53×	**2.46**	**4.71**	**232.34**	**29.24**	**0.908**	**0.052**
FORA ($\mathcal{N} = 6$)	✔	0.135	4.29	5.53×	9.24	14.84	171.33	20.69	0.638	0.290
ToCa ($\mathcal{N} = 9$)	✘	0.127	4.54	5.23×	12.86	12.82	151.37	15.71	0.485	0.473
DuCa ($\mathcal{N} = 9$)	✔	0.105	4.30	5.52×	12.05	11.82	156.20	15.85	0.483	0.460
TaylorSeer ($\mathcal{N} = 6$)	✔	0.191	4.76	4.98×	3.09	6.50	225.16	25.48	0.831	0.107
Z-Cache ($\mathcal{N} = 6$)	✔	0.207	4.78	4.96×	**2.68**	**4.84**	**228.26**	**27.16**	**0.880**	**0.078**
FORA ($\mathcal{N} = 7$)	✔	0.123	3.82	6.22×	12.55	18.63	148.44	19.87	0.598	0.335
ToCa ($\mathcal{N} = 13$)	✘	0.118	4.03	5.90×	21.24	19.93	116.08	15.06	0.449	0.540
DuCa ($\mathcal{N} = 12$)	✔	0.081	3.94	6.02×	31.97	27.26	87.94	14.23	0.396	0.593
TaylorSeer ($\mathcal{N} = 7$)	✔	0.177	3.82	6.22×	3.60	7.07	217.79	**24.09**	0.794	0.136
Z-Cache ($\mathcal{N} = 7$)	✔	0.185	3.82	6.22×	**3.25**	**5.52**	**219.30**	23.96	**0.795**	**0.129**

For *Z-Cache*, Eq. (9) and the triangle inequality yield

$$\begin{aligned} \epsilon_i^z &= \|\tilde{\mathcal{H}}_i - \mathcal{H}_i^\star\| \\ &\leq (1 - w_i)\epsilon_i^p + w_i\epsilon_i^f, \end{aligned} \tag{18}$$

where $\epsilon_i^f = \|\hat{\mathcal{H}}_i^f - \mathcal{H}_i^\star\|$ is the future-anchor prediction error. Combining Eq. (18) with Eq. (16) shows that no-reflection caching is the special case $w_i = 0$, while *Z-Cache* replaces the single-anchor error term ϵ_i^p with the reflected error term ϵ_i^z. Thus, whenever the dual-anchor estimate reduces hidden error over the active window, *Z-Cache* gives a tighter local upper bound on δ_b.

4 Experiments

4.1 Class-to-Image Generation

For comprehensive evaluation of class-conditional generation on ImageNet [Russakovsky et al., 2015], all experiments employed a DDIM-50 sampling protocol at 256×256 resolution with 50,000 images generated in total. Methods implementing activation period optimization are explicitly annotated with the operator $\mathcal{N}$. All experiments were executed on NVIDIA H20 GPUs with standardized controls.

Quantitative evaluations on ImageNet class-to-image generation using the DiT-XL/2 model confirm that *Z-Cache* consistently outperforms state-of-the-art caching baselines across key metrics. As Table 1 demonstrates, *Z-Cache* achieves the best performance at all tested acceleration levels. Specifically, at $6.22\times$ acceleration, *Z-Cache* maintains both the lowest FID (3.25) and sFID (5.52) score, and also the highest Inception Score (219.30). On the contrary, FORA, ToCa and DuCa are all suffering from catastrophic degradation (FID > 10, sFID > 10, and Inception Score < 200). This signifies *Z-Cache*'s superior preservation of image fidelity and diversity.

Furthermore, as detailed in Table 1, *Z-Cache* achieved a comparable performance with Taylorseer while surpassing other sampling results on the $6.22\times$ acceleration level. This comprehensive performance advantage underscores that *Z-Cache* generates sampling trajectories closely aligned with the original model, minimizing semantic shifts in the output data.

To better illustrate *Z-Cache*'s mechanism, we gathered latent hidden states generated during sampling and compared their trajectories. Visualized in Figure 4, other methods often produce "bumps" during cache time as they do not perceive future states, while *Z-Cache* repetitively reflects past hidden states and therefore lies close to the ground truth trajectories.

4.2 Text-to-Image Generation

The paired comparisons in Table 2 show a consistent pattern: *Z-Cache* mainly acts as a quality-recovery module for aggressive feature caching. Across TeaCache, TaylorSeer, SpeCa, and MagCache,

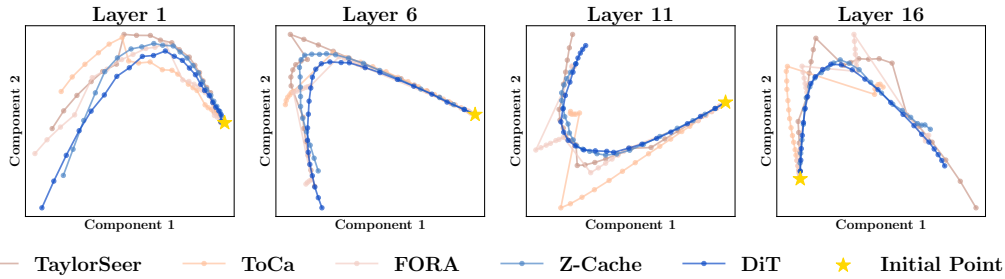


Figure 4: The Trajectories of PCA components of hidden features. *Z-Cache* has the minimum divergence to the ground-truth comparing with other methods.

Table 2: **Full paired results for cache predictors on FLUX.1-dev with DrawBench.** Rows are grouped by the equivalent cache interval $\mathcal{N}$. Bold quality metrics indicate the better value within each Base/+Z-Cache pair; gray rows identify +Z-Cache. Parameters are abbreviated in parentheses: TeaCache uses (L), where L = `rel_l1_thresh`; SpeCa uses $(B, D, m, M) = (\texttt{base}, \texttt{decay}, \texttt{min}, \texttt{max})$; MagCache uses $(T, K, R) = (\texttt{th}, K, \texttt{retention})$.

Method (FLUX.1-dev)	Peak Mem (GiB) ↓	Latency(s) ↓	L-Speed ↑	FLOPs(T) ↓	F-Speed ↑	PSNR ↑	SSIM ↑	LPIPS ↓	IR ↑
Equivalent cache interval $\mathcal{N} = 6$									
TeaCache ($L = .8$)	35.77	2.53	3.67×	891.97	4.17×	28.47	0.6084	0.4691	**1.0246**
TeaCache + Z-Cache ($L = .8$)	39.99	5.07	1.83×	896.71	4.15×	**28.65**	**0.6367**	**0.4328**	1.0147
TaylorSeer	39.96	3.10	3.00×	745.54	4.99×	28.84	0.6302	0.4359	0.9737
TaylorSeer + Z-Cache	39.96	3.29	2.82×	746.47	4.98×	**29.22**	**0.6539**	**0.3914**	**0.9882**
SpeCa $(100, 1, 4, 20)$	40.02	3.24	2.86×	756.87	4.91×	28.78	0.6617	0.4034	0.9517
SpeCa + Z-Cache $(100, 1, 4, 20)$	40.02	3.46	2.69×	757.96	4.91×	**29.18**	**0.6806**	**0.3675**	**0.9844**
MagCache $(.45, 7, .02)$	33.85	1.99	4.67×	746.03	4.99×	29.44	0.7119	0.3556	0.9316
MagCache + Z-Cache $(.45, 7, .02)$	37.98	3.37	2.75×	747.27	4.98×	**29.73**	**0.7253**	**0.3236**	**0.9674**
Equivalent cache interval $\mathcal{N} = 7$									
TeaCache ($L = 1.2$)	35.77	1.99	4.67×	668.98	5.56×	28.30	0.5577	0.5466	0.8715
TeaCache + Z-Cache ($L = 1.2$)	39.99	4.75	1.95×	673.17	5.53×	**28.41**	**0.5875**	**0.5085**	**0.8802**
TaylorSeer	39.96	2.94	3.16×	671.11	5.54×	28.61	0.6032	0.4797	0.9455
TaylorSeer + Z-Cache	39.96	3.12	2.97×	672.02	5.53×	**28.93**	**0.6310**	**0.4306**	**0.9753**
SpeCa $(100, 1, 5, 20)$	40.02	2.97	3.12×	679.74	5.47×	28.54	0.6353	0.4527	0.9305
SpeCa + Z-Cache $(100, 1, 5, 20)$	40.02	3.39	2.74×	680.83	5.46×	**28.87**	**0.6581**	**0.4059**	**0.9698**
MagCache $(.45, 9, .02)$	33.85	1.94	4.78×	671.50	5.54×	29.41	0.7020	0.3689	0.9252
MagCache + Z-Cache $(.45, 9, .02)$	37.98	3.12	2.97×	672.70	5.53×	**29.68**	**0.7164**	**0.3356**	**0.9682**
Equivalent cache interval $\mathcal{N} = 8$									
TeaCache ($L = 1.4$)	35.77	1.87	4.96×	594.65	6.25×	28.24	0.5353	0.5770	0.7680
TeaCache + Z-Cache ($L = 1.4$)	39.99	4.64	2.00×	598.66	6.21×	**28.33**	**0.5702**	**0.5321**	**0.8288**
TaylorSeer	39.96	2.83	3.28×	596.69	6.23×	28.46	0.5703	0.5190	0.9292
TaylorSeer + Z-Cache	39.96	3.17	2.93×	597.52	6.22×	**28.71**	**0.6002**	**0.4692**	**0.9451**
SpeCa $(100, 1, 6, 20)$	40.02	2.84	3.27×	605.23	6.15×	28.43	0.6111	0.4867	0.8874
SpeCa + Z-Cache $(100, 1, 6, 20)$	40.02	3.04	3.05×	606.27	6.14×	**28.67**	**0.6327**	**0.4420**	**0.9004**
MagCache $(.80, 7, .02)$	33.85	1.86	5.00×	596.98	6.23×	28.88	0.6582	0.4275	0.9415
MagCache + Z-Cache $(.80, 7, .02)$	37.98	2.78	3.34×	598.12	6.22×	**29.05**	**0.6710**	**0.3994**	**0.9511**

reflection improves PSNR, SSIM, and LPIPS in every matched setting, while ImageReward improves in all but one case. These results indicate *Z-Cache* can be directly applied to any existing feature caching methods.

Table 2 also separates arithmetic cost from practical wall-clock cost. Since *Z-Cache* reuses the next scheduled full-computation anchor rather than inserting extra full steps, the additional FLOPs are small. The FLOPs overhead is below 0.7% for all paired settings and below 0.2% for TaylorSeer, SpeCa, and MagCache, so the FLOPs speedup is nearly unchanged after reflection. However, latency and peak memory are more predictor-dependent. For TaylorSeer and SpeCa, *Z-Cache* introduces modest latency overhead and no measured increase in peak memory. For TeaCache and MagCache, the quality gain remains consistent, but the current implementation incurs a larger wall-clock cost and higher peak memory. This suggests that *Z-Cache* should be viewed as a FLOPs-light quality-recovery wrapper rather than a universally latency-free module.

Figure 5 further shows an interval sweep where TaylorSeer progressively loses semantic details as $\mathcal{N}$ increases, while *Z-Cache* preserves the object layout and visual identity over larger cache intervals. Together, the prompt-level comparison and the interval sweep are consistent with the quantitative trend in Table 2: revisiting the active cache window with a future anchor reduces cache-induced semantic drift under high acceleration.

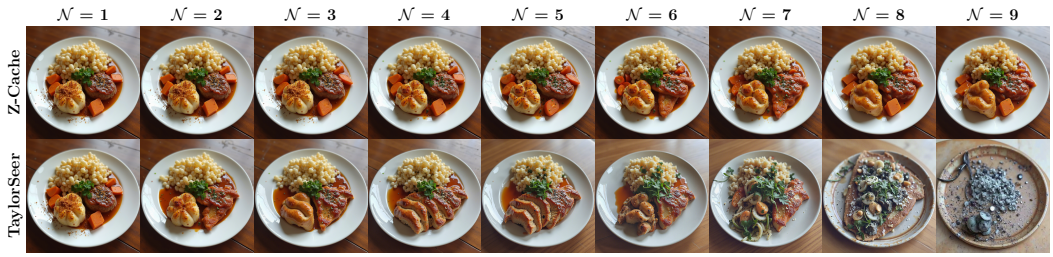


Figure 5: **Semantic stability under increasing cache interval $\mathcal{N}$.** As the cache interval increases, TaylorSeer gradually accumulates semantic drift, while Z-Cache maintains more stable object layout and visual identity.

4.3 Text-to-Video Generation

Table 3: **Quantitative comparison on text-to-video generation** with HunyuanVideo on VBench. L-Speed and F-Speed are the latency and FLOPs speedups relative to the original 50-step model. Within each caching block, bold numbers indicate the best values, including ties.

Method (on HunyuanVideo)	Efficient Attention	Acceleration				VBench ↑
		Latency(s) ↓	L-Speed ↑	FLOPs(T) ↓	F-Speed ↑	Score(%)
Original: 50 steps	✔	145.00	1.00×	29773.0	1.00×	80.66
22% **steps**	✔	31.87	4.55×	6550.1	4.55×	78.74
TeaCache($l = 0.4$)	✔	26.61	5.45×	6550.1	4.55×	79.36
FORA($N = 5$)	✔	34.39	4.22×	**5960.4**	**5.00**×	78.83
ToCa ($\mathcal{N} = 5, R = 90\%$)	✘	38.52	3.76×	7006.2	4.25×	78.86
DuCa ($\mathcal{N} = 5, R = 90\%$)	✔	31.69	4.58×	6483.2	4.59×	78.72
TaylorSeer ($\mathcal{N} = 5, O = 1$)	✔	**25.34**	**5.72**×	**5960.4**	**5.00**×	79.93
Z-Cache ($\mathcal{N} = 5, O = 1$)	✔	28.69	5.05×	5997.2	4.96×	**80.10**
TeaCache($l = 0.5$)	✔	26.61	5.45×	5359.1	5.56×	78.32
TaylorSeer ($\mathcal{N} = 7, O = 1$)	✔	**21.01**	**6.90**×	**4794.5**	**6.21**×	79.09
Z-Cache ($\mathcal{N} = 7, O = 1$)	✔	23.08	6.28×	4799.6	6.20×	**79.60**

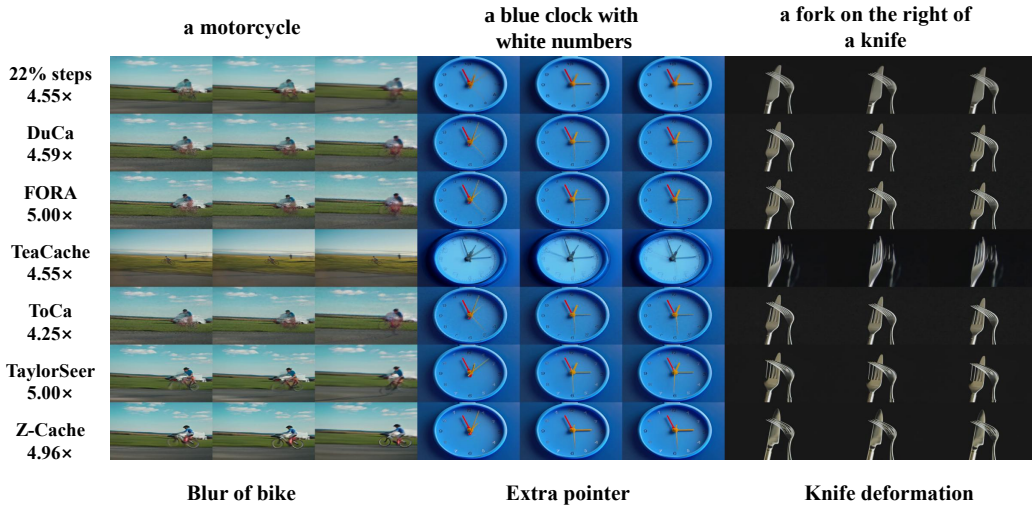


Figure 6: Visualization of videos generated by Hunyuan Video with different feature caching methods. The annotated speedups denote FLOPs reductions relative to 50-step sampling.

Extending to video synthesis on HunyuanVideo, *Z-Cache* maintains a consistent and comparable performance with other competing methods on the VBench metric. As Table 3 shows, our method achieves the highest VBench scores within the two caching groups (80.10 at 4.96× FLOPs speedup and 79.60 at 6.20× FLOPs speedup). This demonstrates *Z-Cache*'s advantage for video diffu-

sion acceleration, where existing caching strategies struggle to balance efficiency with temporal consistency.

Beyond quantitative metrics, we present qualitative comparisons of these methods on identical frame sequences in Figure 6. For instance, in the "a motorcycle" video, *Z-Cache* generates a coherent depiction of the rider's face, whereas methods like TaylorSeer produce distorted facial features or fragmented figures. In the "a blue clock with white numbers" video, several baselines generate extra or distorted clock hands, whereas Z-Cache better preserves the intended clock structure. Finally, in the *"a fork on the right of a knife"* video, all baseline methods either produce malformed or misshapen knives or completely disregard the fork, whereas *Z-Cache* accurately captures both objects in their correct spatial arrangement.

4.4 Miscellaneous

To test whether *Z-Cache* remains effective in the few-step regime, we additionally evaluate it on a distilled model, as reported in Table 4a. Specifically, we compare *Z-Cache* with TaylorSeer and ToCa on Qwen-Image-Lightning using an 8-step sampler. *Z-Cache* achieves the highest PSNR and the lowest LPIPS among the methods, while maintaining comparable SSIM. These results show that, even under aggressive distillation with very few denoising steps, *Z-Cache* still delivers a clear improvement over prior cache-based baselines, indicating that cache-based acceleration can remain beneficial beyond standard multi-step diffusion settings.

Table 4: Additional quantitative results on few-step generation, image editing, U-Net backbones, and non-cache acceleration settings. All **Z-Cache** variants use the TaylorSeer predictor, and the number in parentheses is the cache interval.

(a) Few-step distilled model Qwen-Image-Lightning on DrawBench.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
TaylorSeer (2)	30.222	0.6734	0.2984
ToCa (2)	29.546	0.6780	0.2938
Z-Cache (2)	**30.440**	**0.6785**	**0.2910**

(b) Comparison with other acceleration methods on DrawBench.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	ImageReward ↑	Accel. ↑
ToMA [Lu et al., 2025]	29.271	0.6984	0.4258	0.7812	1.21×
8bit	32.331	**0.8334**	0.1770	**1.1971**	2.10×
8bit TaylorSeer (2)	33.085	0.8197	0.1413	0.8773	3.12×
8bit Z-Cache (2)	**33.600**	0.8289	**0.1337**	0.8877	3.12×
Flux-mini	28.101	0.5186	0.6449	0.2522	**3.80**×
Z-Cache (4)	30.570	0.7758	0.2406	0.9890	2.17×

(c) FLUX-Kontext-dev [Labs et al., 2025] on Gedit-bench [Liu et al., 2025e].

Method	PSNR ↑	SSIM ↑	LPIPS ↓
TaylorSeer (8)	29.014	0.4338	0.4448
Z-Cache (8)	**29.511**	**0.5114**	**0.3694**
TaylorSeer (7)	29.675	0.5013	0.3658
Z-Cache (7)	**30.279**	**0.5667**	**0.3015**

(d) Experiments with the U-Net model SDXL [Podell et al., 2023] on DrawBench.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	ImageReward ↑	Accel. ↑
TaylorSeer (5)	29.028	0.6969	0.3294	0.3350	2.61×
Z-Cache (5)	**29.399**	**0.7112**	**0.3100**	**0.3788**	2.61×
TaylorSeer (7)	28.251	0.6041	0.4528	0.1024	4.00×
Z-Cache (7)	**28.589**	**0.6373**	**0.4162**	**0.1802**	4.00×

In addition to free-form generation, we also evaluate *Z-Cache* in an image-editing setting. Specifically, we run FLUX-Kontext-dev on the Gedit-bench benchmark and compare *Z-Cache* with TaylorSeer, as summarized in Table 4c. At the same acceleration ratio, *Z-Cache* achieves higher PSNR and SSIM while substantially reducing LPIPS. These improvements indicate that *Z-Cache* better preserves the source image's structure and perceptual details, which are crucial for maintaining subject identity and keeping non-edited background regions consistent.

We further compare *Z-Cache* with non-caching acceleration methods in Table 4b. The results show that *Z-Cache* is complementary to quantization: adding *Z-Cache* on top of 8-bit quantization (8bit *Z-Cache*) increases the speedup from 2.10× (8bit only) to 3.12×, while also improving PSNR and reducing LPIPS. Under the same 3.12× acceleration budget, 8bit *Z-Cache* consistently outperforms 8bit TaylorSeer across all four metrics (PSNR, SSIM, LPIPS, and ImageReward), indicating that the reflection step provides non-trivial gains even when the backbone is already compressed. In contrast, a distilled model such as Flux-mini achieves a comparable speedup (3.80×) but with substantially worse fidelity and preference scores, suggesting that cache-based acceleration remains competitive in the high-speed regime. Overall, these results support the view that Z-Cache is not a replacement for quantization/pruning, but a plug-in that can be layered on top of them.

We additionally evaluate *Z-Cache* on **SDXL**, a U-Net-based diffusion model, and directly compare it with TaylorSeer under matched step budgets (i.e., the same cache-step setting). As summarized in Table 4d, *Z-Cache* consistently improves PSNR and SSIM, reduces LPIPS, and achieves higher ImageReward than TaylorSeer across both acceleration ratios, while using the same number of cache steps. These results indicate that the proposed cache-then-check mechanism generalizes beyond DiT-style backbones and naturally transfers to U-Net architectures.

Additional trajectory controls, fusion and replay ablations, and recent-predictor compositions are reported in Appendix A.3.

5 Conclusion

This paper introduces *Z-Cache*, a novel caching framework that redefines the efficiency-quality trade-off for diffusion transformers. Central to its innovation is a self-reflective mechanism, which periodically refines intermediate features through "future-aware" feature integration—enabling substantial acceleration without sacrificing output fidelity. By integrating this mechanism, *Z-Cache* achieves exceptional acceleration (up to 6.22× faster inference on FLUX-dev.1) while preserving high output quality. Beyond static image generation, Z-Cache demonstrates remarkable adaptability to video tasks, attaining a state-of-the-art 6.20× acceleration rate for videos, all while maintaining high output quality.

Acknowledgment

This paper was partially sponsored by WUYING - Alibaba Cloud.

References

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

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4599–4603, 2023.

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.

Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation, 2024a.

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

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*, 2024c. URL https://arxiv.org/abs/2406.01125.

Xinle Cheng, Zhuoming Chen, and Zhihao Jia. Cat pruning: Cluster-aware token pruning for text-to-image diffusion models, 2025. URL https://arxiv.org/abs/2502.00433.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. *arXiv preprint arXiv:2305.10924*, 2023.

Haipeng Fang, Yu Li, Fan Tang, Yixing Lu, Juan Cao, and Sheng Tang. ResCa: Residual caching for diffusion transformers acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 32957–32966, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Fang_ResCa_Residual_Caching_for_Diffusion_Transformers_Acceleration_CVPR_2026_paper.html.

Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, and Linfeng Zhang. Hicache: Training-free acceleration of diffusion models via hermite polynomial-based feature caching, 2025. URL https://arxiv.org/abs/2508.16984v1.

Yasaman Haghighi and Alexandre Alahi. SenCache: Accelerating diffusion model inference via sensitivity-aware caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14295–14304, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Haghighi_SenCache_Accelerating_Diffusion_Model_Inference_via_Sensitivity-Aware_Caching_CVPR_2026_paper.html.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1383–1392, 2024.

Sungbin Kim, Hyunwuk Lee, Wonho Cho, Mincheol Park, and Won Woo Ro. Ditto: Accelerating diffusion model via temporal value similarity. In *Proceedings of the 2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2025.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Junkun Yuan, Yanxin Long, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Dax Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. HunyuanVideo: A systematic framework for large video generative models, 2024. URL https://arxiv.org/abs/2412.03603.

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.

Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *arXiv preprint arXiv:2312.09608v1*, 2023a. URL https://arxiv.org/abs/2312.09608v1.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17489–17499, 2023b. doi: 10.1109/ICCV51070.2023.01608.

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, 2023c.

Enhuai Liu, Yunke Wang, Changming Sun, and Chang Xu. D2Cache: Second-order delta caching for higher video diffusion acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 43589–43599, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Liu_D2Cache_Second-Order_Delta_Caching_for_Higher_Video_Diffusion_Acceleration_CVPR_2026_paper.html.

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.

Jiacheng Liu, Peiliang Cai, Qinming Zhou, Yuqi Lin, Deyang Kong, Benhao Huang, Yupei Pan, Haowen Xu, Chang Zou, Junshu Tang, Shikang Zheng, and Linfeng Zhang. FreqCa: Accelerating diffusion models via frequency-aware caching, 2025a. URL https://arxiv.org/abs/2510.08669.

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, 2025b. URL https://arxiv.org/abs/2510.19755.

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

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 2025d. doi: 10.1145/3746027.3755331. URL http://dx.doi.org/10.1145/3746027.3755331.

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, 2025e. URL https://arxiv.org/abs/2504.17761.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://iclr.cc/virtual/2023/poster/11266.

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

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, 2022a.

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*, 2022b.

Wenbo Lu, Shaoyi Zheng, Yuxuan Xia, and Shengjie Wang. Toma: Token merge with attention for diffusion models, 2025. URL https://arxiv.org/abs/2509.10918.

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. URL https://openaccess.thecvf.com/content/CVPR2024/html/Ma_DeepCache_Accelerating_Diffusion_Models_for_Free_CVPR_2024_paper.html.

Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. Magcache: Fast video generation with magnitude-aware cache, 2025. URL https://arxiv.org/abs/2506.09045.

Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *NeurIPS 2022 Workshop on Score-Based Methods*, 2022. URL https://arxiv.org/abs/2210.03142v1.

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. doi: 10.1109/ICCV51070.2023.00387. URL https://openaccess.thecvf.com/content/ICCV2023/html/Peebles_Scalable_Diffusion_Models_with_Transformers_ICCV_2023_paper.html.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023. URL https://arxiv.org/abs/2307.01952.

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.

Junxiang Qiu, Shuo Wang, Jinda Lu, Lin Liu, Houcheng Jiang, and Yanbin Hao. Accelerating diffusion transformer via error-optimized cache, 2025. URL https://arxiv.org/abs/2501.19243v1.

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.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3): 211–252, December 2015. doi: 10.1007/s11263-015-0816-y. URL https://link.springer.com/article/10.1007/s11263-015-0816-y.

Omid Saghatchian, Atiyeh Gh. Moghadam, and Ahmad Nickabadi. Cached adaptive token merging: Dynamic token reduction and redundant computation elimination in diffusion model, 2025.

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

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. URL https://arxiv.org/abs/2407.01425.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1972–1981, 2023.

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, 2026a. URL https://arxiv.org/abs/2604.26365.

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, 2026b. URL https://arxiv.org/abs/2609.05981.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL https://iclr.cc/virtual/2021/poster/2804.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning*, pages 32211–32252. PMLR, 2023.

Wenzhang Sun, Qirui Hou, Donglin Di, Jiahui Yang, Yongjia Ma, and Jianxun Cui. Unicp: A unified caching and pruning framework for efficient video generation, 2025. URL https://arxiv.org/abs/2502.04393.

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.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, Da Yin, Yuxuan Zhang, Weihan Wang, Yean Cheng, Bin Xu, Xiaotao Gu, Yuxiao Dong, and Jie Tang. Cogvideox: Text-to-video diffusion models with an expert transformer. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ce31378e9f41d8907e97dab172b6c559-Abstract-Conference.html.

Evelyn Zhang, Bang Xiao, Jiayi Tang, Qianli Ma, Chang Zou, Xuefei Ning, Xuming Hu, and Linfeng Zhang. Token pruning for caching better: 9 times acceleration on stable diffusion for free, 2024. URL https://arxiv.org/abs/2501.00375.

Evelyn Zhang, Jiayi Tang, Xuefei Ning, and Linfeng Zhang. Training-free and hardware-friendly acceleration for diffusion models via similarity-based token pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. DPM-solver-v3: Improved diffusion ODE solver with empirical model statistics. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/ada8de994b46571bdcd7eeff2d3f9cff-Abstract-Conference.html.

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, 2025a. URL https://arxiv.org/abs/2510.04188.

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, 2025b. URL https://arxiv.org/abs/2508.16211.

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, 2026a. URL https://arxiv.org/abs/2601.07462.

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

Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all, 2024. URL https://arxiv.org/abs/2412.20404.

Haowei Zhu, Dehua Tang, Ji Liu, Mingjie Lu, Jintu Zheng, Jinzhang Peng, Dong Li, Yu Wang, Fan Jiang, Lu Tian, Spandan Tiwari, Ashish Sirasao, Jun-Hai Yong, Bin Wang, and Emad Barsoum. Dip-go: A diffusion pruner via few-step gradient optimization, 2024.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024a. URL https://arxiv.org/abs/2410.05317.

Chang Zou, Evelyn Zhang, Runlin Guo, Haohang Xu, Conghui He, Xuming Hu, and Linfeng Zhang. Accelerating diffusion transformers with dual feature caching, 2024b. URL https://arxiv.org/abs/2412.18911v1.

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 Appendix

A.1 Additional Related Works

Diffusion models and diffusion transformers. Diffusion models [Sohl-Dickstein et al., 2015, Ho et al., 2020] generate data through iterative denoising and have become a dominant framework for visual generation. Early systems were largely based on U-Net architectures [Ronneberger et al., 2015], while recent large-scale models increasingly adopt diffusion transformers (DiTs) [Peebles and Xie, 2023]. This architectural shift has enabled strong image and video generation performance across modern systems [Chen et al., 2024b,a, Zheng et al., 2024, Yang et al., 2025]. However, the iterative sampling process remains computationally expensive, motivating two broad acceleration directions: reducing the number of sampling timesteps and reducing the per-step cost of the denoising network.

Sampling timestep reduction. Timestep reduction methods aim to preserve generation quality with fewer denoising iterations. DDIM [Song et al., 2021] introduces deterministic sampling for faster inference, while DPM-Solver and its variants [Lu et al., 2022a,b, Zheng et al., 2023] use high-order ODE solvers to reduce the number of function evaluations. Rectified Flow [Liu et al., 2023] learns straighter paths between noise and data, and distillation-based approaches compress multi-step generation into fewer steps [Salimans and Ho, 2022, Meng et al., 2022]. Consistency Models [Song et al., 2023] further enable one-step or few-step generation by directly mapping noisy inputs to clean outputs. These methods reduce sampling length and are orthogonal to our goal of improving feature caching within a given sampling schedule.

Denoising network acceleration. Another line of work accelerates the denoising network at each timestep. Compression-based methods include pruning [Fang et al., 2023, Zhu et al., 2024], quantization [Li et al., 2023b, Shang et al., 2023, Kim et al., 2025], distillation [Li et al., 2023c], and token reduction or token merging [Bolya and Hoffman, 2023, Kim et al., 2024, Zhang et al., 2024, 2025, Cheng et al., 2025, Saghatchian et al., 2025], alongside dynamic-resolution approaches [Zheng et al., 2026a, Yan et al., 2026, Zheng et al., 2026b, Qin et al., 2026]. These approaches reduce computation by simplifying model weights, activations, or token sequences. Although effective, they often require tuning, calibration, or careful engineering to maintain the quality–efficiency balance. Z-Cache is complementary to these methods because it is training-free and modifies the inference-time feature-caching trajectory rather than the model parameters.

Feature caching-based acceleration. Feature caching exploits temporal redundancy across diffusion timesteps to accelerate inference [Liu et al., 2025b,a]. Early caching methods for U-Net diffusion models include Faster Diffusion [Li et al., 2023a] and DeepCache [Ma et al., 2024]. Recent methods extend this idea to transformer-based diffusion models. FORA [Selvaraju et al., 2024] and Δ-DiT [Chen et al., 2024c] reuse attention or representation states across nearby steps. TeaCache [Liu et al., 2024] uses timestep-adaptive caching, while ToCa and DuCa [Zou et al., 2024a,b] cache token-wise or dual feature representations. Other methods improve efficiency through error-aware caching, adaptive caching-pruning combinations, or spatially varying sampling strategies [Qiu et al., 2025, Sun et al., 2025, Liu et al., 2025f].

Forecasting and predictor-oriented caching. More recent feature-caching methods improve the base cache predictor itself [Zheng et al., 2025a, Cai et al., 2026, Shen et al., 2026b, Zou et al., 2026, Shen et al., 2026a]. TaylorSeer [Liu et al., 2025c] predicts intermediate features using polynomial extrapolation from neighboring cached activations, which establishes the *cache-then-forecast* paradigm. SpeCa [Liu et al., 2025d] introduces speculative feature caching, and MagCache [Ma et al., 2025] exploits magnitude-aware caching for efficient generation. These works primarily focus on improving feature prediction or cache scheduling.

Sensitivity-aware, second-order, and spatial residual caching. Recent methods address different sources of caching error. SenCache [Haghighi and Alahi, 2026] estimates how denoising outputs respond to changes in the latent input and timestep, and uses these sensitivities to decide whether cached outputs can be reused for each sample. Its sensitivity estimates are obtained using a small calibration set. D2Cache [Liu et al., 2026] augments a cached first-order residual with a second-order residual difference and scales this correction using error estimates derived from timestep embeddings.

ResCa [Fang et al., 2026] exploits similarity among token denoising trajectories: it computes a proxy token for each cluster and uses multi-order residuals to guide simulated updates of the remaining tokens. Its updates combine a token's historical residuals with current residual information from the proxy. These approaches respectively emphasize sensitivity-guided cache decisions, temporal residual prediction, and spatially shared denoising information.

Relation to Z-Cache. Z-Cache performs retrospective trajectory correction within an active cache window. Once the next scheduled full-computation anchor is observed on the provisional trajectory, Z-Cache combines past-anchor and future-anchor predictions, replays the window, and commits the reflected endpoint before continuing denoising. This differs from SenCache's reuse decisions, D2Cache's temporal extrapolation, and ResCa's current-step proxy-guided token updates. Z-Cache can therefore complement compatible cache policies and predictors while preserving the base full-computation schedule, provided their cached representations and state updates support dual-anchor prediction and window replay.

A.2 Experiment Settings

A.2.1 DiT

We conducted experiments mainly on three state-of-the-art diffusion architectures: DiT-XL/2 [Peebles and Xie, 2023], FLUX.1-dev [Black Forest Labs, 2024], and HunyuanVideo [Kong et al., 2024]. A per-model subset of baselines was systematically selected from the literature: FORA [Selvaraju et al., 2024], ToCa [Zou et al., 2024a], DuCa [Zou et al., 2024b], TeaCache Liu et al. [2024], and TaylorSeer [Liu et al., 2025c].

A.2.2 FLUX-dev.1

We evaluate text-guided image generation on FLUX.1-dev with DrawBench. Each method generates 200 images at 1024×1024 resolution using the default sampling configuration. To make the efficiency comparison explicit, we report end-to-end H100 latency, peak CUDA memory, and FLOPs under the same implementation. FLOPs speed is normalized by the original 50-step FLUX.1-dev cost. PSNR, SSIM, and LPIPS are computed against the corresponding full 50-step FLUX.1-dev outputs under the same prompts and seeds, while ImageReward is computed on generated image–prompt pairs.

A.2.3 HunyuanVideo

For video generation tasks, all experiments utilized the model's default configuration. We generated a total of 4720 video sequences (65 frames per sequence at 640×480 resolution) from VBench with 4 samples each. Evaluation employed the comprehensive VBench framework. Since all methods produce noticeable semantic shifts, we skipped all pixel-level evaluation for the best interpretation.

A.3 Supplementary Experiments

A.3.1 Global trajectory comparison

We compare Full sampling, TaylorSeer without reflection, and canonical Z-Cache on FLUX.1-dev using 32 frozen DrawBench prompts, two seeds per prompt, and cache intervals $N \in \{6, 8\}$. Each paired sample shares its initial latent and the 50-step scheduler. We save all 50 latent tensors. For Z-Cache, the last replayed value replaces the provisional value at each revisited step; this reconciled trajectory is the primary comparison.

For method M, define the relative latent error and its normalized trapezoidal area under the curve (AUC) as

$$d_i^M = \frac{\|x_{i+1}^M - x_{i+1}^{\text{Full}}\|_2}{\max(\|x_{i+1}^{\text{Full}}\|_2, 10^{-12})}, \qquad A^M = \frac{1}{T-1} \sum_{i=0}^{T-2} \frac{d_i^M + d_{i+1}^M}{2}, \quad T = 50. \tag{19}$$

Table 5 reports sample means.

Table 5: Global latent trajectories relative to the shared Full reference.

N	Metric	TaylorSeer	Z-Cache
6	AUC ↓	0.212977	**0.198117**
6	Final rel.-L_2 ↓	0.574011	**0.546836**
6	Final cosine ↑	0.823749	**0.841404**
8	AUC ↓	0.256287	**0.235652**
8	Final rel.-L_2 ↓	0.669091	**0.630887**
8	Final cosine ↑	0.762938	**0.792403**

Reflection reduces mean trajectory AUC by 6.98% and 8.05% at $N = 6$ and $N = 8$, respectively. All three prompt-averaged metrics improve for all 32 prompts at both settings. This does not imply uniform sample-wise improvement: final relative L_2 improves for 60/64 and 63/64 samples, respectively. All 52,736 checked trajectory values are finite and no rank fails. These are latent-trajectory measurements, not additional image-quality or latency scores, and they do not establish a universal stability or convergence guarantee.

A.3.2 Same-start local reflection analysis

To separate the effect of reflection from previously accumulated drift, we fork three branches from the exact same committed canonical Z-Cache left-anchor state x_{a+1}: a local Full branch, the provisional TaylorSeer branch, and canonical reflected replay. The local Full branch evaluates the full network at every post-anchor step. It supplies neither features nor the future anchor to the cached branches and is a diagnostic reference, not an equal-compute baseline. The experiment uses the same 32 prompts and two seeds as Section A.3.1, with seven windows per sample at $N = 6$ and five at $N = 8$, giving 768 matched windows.

We first average window metrics within each sample. For the endpoint-displacement intervals reported below, we use 95% percentile bootstrap intervals from 20,000 prompt-cluster resamples, retaining both seeds of each sampled prompt. Local AUC is the normalized trapezoidal relative-L_2 error from the common start through the window endpoint x_{b+1}. Table 6 uses the conditional local Full trajectory as its reference; it must not be pooled with the global Full comparison.

Table 6: Same-start local trajectories relative to the conditional local Full reference.

N	Metric	TaylorSeer	Z-Cache
6	AUC ↓	0.012688	**0.008928**
6	Endpoint rel.-L_2 ↓	0.026057	**0.016922**
6	Endpoint cosine ↑	0.999620	**0.999828**
8	AUC ↓	0.020822	**0.015519**
8	Endpoint rel.-L_2 ↓	0.045693	**0.031364**
8	Endpoint cosine ↑	0.998882	**0.999430**

All three sample-aggregated local metrics favor reflection for 64/64 samples and 32/32 prompts at both intervals. The common starts match exactly and all 20,992 checked local relative-L_2/cosine values are finite.

The replayed endpoint intentionally differs from the provisional endpoint. Its direct relative-L_2 displacement, normalized by the provisional TaylorSeer endpoint norm, is 0.014184 [0.013653, 0.014733] at $N = 6$ and 0.023060 [0.022209, 0.023909] at $N = 8$ (sample means and prompt-cluster 95% intervals). The corresponding window-level 95th percentiles are 0.025632 and 0.037227. Thus, endpoint changes are measurable; the same-start control shows that they move the tested local trajectories closer to conditional Full, without asserting that every possible trajectory or cache predictor behaves similarly.

A.3.3 Fusion rules and replay implementation

We evaluate four reflection functions on 200 paired DrawBench prompts with FLUX.1-dev, TaylorSeer $N = 6$, and the same frozen Base schedule. With Taylor order K, distances $d_L = i - a$, $d_R = b - i$, and $r = d_L/(d_L + d_R)$, the weights on the future-conditioned feature prediction are

$$w_{\text{linear}} = r, \qquad w_{\text{remainder}} = \frac{d_L^{K+1}}{d_L^{K+1} + d_R^{K+1}},$$
$$w_{\text{variance}} = \frac{d_L^{2(K+1)}}{d_L^{2(K+1)} + d_R^{2(K+1)}}, \qquad w_{\text{smoothstep}} = r^2(3 - 2r). \tag{20}$$

The "variance-optimal" name refers to an independent-error model using squared Taylor-remainder scales; it does not use oracle errors.

Image-evaluation protocol. Tables 7–9 report SSIM computed on grayscale images; LPIPS and ImageReward use the same frozen evaluation configuration.

Table 7: Fusion-function ablation. Bold marks the best point estimate in each SSIM, LPIPS, and IR column; no confidence intervals or human-preference study establish quality significance.

Fusion rule	SSIM↑	LPIPS↓	IR↑
Original linear	0.6462	0.3914	**0.9908**
Remainder ratio	0.6474	0.3888	0.9763
Variance-optimal	**0.6476**	**0.3883**	0.9863
Jet smoothstep	0.6471	0.3898	0.9821

All four rules use the same Base schedule, 747.501934 TFLOPs, and approximately 41.652 GiB peak allocated memory. Variance-optimal leads SSIM and LPIPS, whereas linear leads ImageReward. Linear remains a simple parameter-free default.

Table 8: Two replay backends under the same TaylorSeer $N = 6$ schedule. Image quality uses DrawBench200; efficiency is a separate frozen-prompt A100 diagnostic.

Backend	SSIM↑	LPIPS↓	IR↑
Block-wise linear	0.6462	0.3914	0.9908
Root-residual state graft	**0.6505**	**0.3797**	**0.9916**
Backend	**Latency (s)↓**	**TFLOPs↓**	**Peak GiB↓**
Block-wise linear	6.7656	747.502	41.652
Root-residual state graft	**5.4130**	**747.095**	**40.011**

The graft reflects root residuals and bypasses the 57 Transformer blocks during replay instead of fusing 95 internal feature sites. Both backends retain 10 primary FULL steps, 40 primary PREDICT steps, seven closed windows, and zero extra endpoint FULL calls. Latency is the median of seven balanced rounds after one excluded warm-up on one A100 and frozen prompt 0, including hooks, denoising, and replay but excluding model loading, prompt encoding, latent preparation, and VAE decoding. Memory is maximum CUDA allocated memory; FLOPs are a separate exact-trajectory Transformer-forward profile. The graft reduces latency by 19.99% and peak memory by 3.94%, while changing the approximation rather than preserving identical outputs. This is not an H100 reproduction or evidence that overhead is eliminated for other Bases, intervals, or models.

A.3.4 Composition with recent cache predictors

We evaluate audited clean-room FLUX ports of SenCache and D2Cache on all 200 DrawBench prompts. Within each Base/+Z pair, prompts, seeds, initial latents, and the Base FULL schedule are identical. These controlled ports are additional composability evidence, not measurements of the authors' official implementations. Table 9 uses the image-evaluation protocol described in

Section A.3.3; bold indicates the better SSIM, LPIPS, or ImageReward point estimate within each matched pair.

Table 9: Matched recent-predictor results. FULL is the mean number of primary FULL evaluations across 200 samples. F-Speed is measured on one frozen prompt-0 trajectory; it is not a mean over 200 prompts or a latency speedup.

Base	N	Arm	FULL	F-Speed ↑	SSIM ↑	LPIPS ↓	IR ↑
SenCache	6	Base	10.000	4.996	0.6115	0.5126	0.7210
	6	+Z	10.000	4.987	**0.6149**	**0.4917**	**0.7985**
SenCache	7/8	Base	8.000	6.243	0.5943	0.5559	0.5486
	7/8	+Z	8.000	6.230	**0.5982**	**0.5289**	**0.6256**
D2Cache	6	Base	10.045	4.996	0.6488	0.4211	**0.9536**
	6	+Z	10.045	4.987	**0.6515**	**0.4158**	0.9415
D2Cache	7	Base	9.015	5.550	0.6346	0.4416	0.9411
	7	+Z	9.015	5.539	**0.6358**	**0.4390**	**0.9554**
D2Cache	8	Base	8.140	6.243	**0.6146**	**0.4775**	0.8706
	8	+Z	8.140	6.229	0.6134	0.4778	**0.9107**

SenCache improves SSIM, LPIPS, and ImageReward at each distinct operating point. Its $N = 7$ calibration target realizes eight FULL evaluations, yielding the same schedules and pixel-identical outputs as $N = 8$ for each arm; we combine these rows rather than count them as independent replications. D2Cache is mixed: at $N = 6$, SSIM/LPIPS improve while ImageReward decreases; at $N = 8$, only ImageReward improves among these three metrics. No quality confidence intervals or human-preference evaluation support a significance or uniform-dominance claim.

F-Speed is the 50-step Full Transformer FLOPs divided by each arm's whole-trajectory Transformer FLOPs, including reflection forwards. The measured additional compute is 0.17–0.21% for these five distinct operating points. The separate A100 latency measurements are preliminary execution diagnostics and are omitted because they do not match the original H100 hardware and optimized timing protocol. Low additional Transformer FLOPs therefore do not imply universally low wall-clock or memory overhead.

A.4 Pseudocode and State Updates

Algorithm 1 specifies the TaylorSeer-based block-feature instantiation with linear fusion used in the trajectory experiments, with a deterministic step-indexed scheduler. Let $a < b$ be consecutive indices selected for FULL computation by the base schedule. The saved left boundary is x_{a+1}. The notation x_i, F_i, and $\mathcal{H}_i$ follows the main text.

Let $\mathcal{B}^p$ be the frozen predictor state at the left boundary and $\mathcal{C}(\mathcal{B}, i)$ its feature prediction. $F_i(x, \mathcal{H})$ evaluates the uncached network components and the scheduler using the supplied cache-site features. $\mathrm{FullStep}_i(x)$ performs one full network evaluation and its scheduler update, returning the output latent and the observed feature bank. Both operators use the original timestep and step index i; replay does not advance a shared scheduler cursor.

Algorithm 1 Z-Cache: forecast, reflect, and commit one window

Require: Scheduled FULL indices $a < b$; saved latent x_{a+1}; frozen history $\mathcal{B}^p$; predictor $\mathcal{C}$
Ensure: Committed latent $\tilde{x}_{b+1}$ and predictor state $\mathcal{B}_b$

$\bar{x}_{a+1} \leftarrow x_{a+1}$ {provisional trajectory}
for $i = a+1, \ldots, b-1$ **do**
  $\hat{\mathcal{H}}_i^p \leftarrow \mathcal{C}(\mathcal{B}^p, i)$
  $\bar{x}_{i+1} \leftarrow F_i(\bar{x}_i, \hat{\mathcal{H}}_i^p)$
end for
$(\bar{x}_{b+1}, \mathcal{H}_b) \leftarrow \mathrm{FullStep}_b(\bar{x}_b)$ {the scheduled right FULL}
$\mathcal{B}^f \leftarrow \mathrm{Update}(\mathcal{B}^p, b, \mathcal{H}_b)$
if $b = a+1$ **then**
  return $(\bar{x}_{b+1}, \mathcal{B}^f)$ {no cached interior}
end if
$\tilde{x}_{a+1} \leftarrow x_{a+1}$ {restore the left boundary}
for $i = a+1, \ldots, b$ **do**
  $w_i \leftarrow (i-a)/(b-a)$
  $\tilde{\mathcal{H}}_i \leftarrow (1-w_i)\mathcal{C}(\mathcal{B}^p, i) + w_i \mathcal{C}(\mathcal{B}^f, i)$
  $\tilde{x}_{i+1} \leftarrow F_i(\tilde{x}_i, \tilde{\mathcal{H}}_i)$ {cached replay, including step b}
end for
$\mathcal{B}_b \leftarrow \mathrm{Update}(\mathcal{B}^p, b, \tilde{\mathcal{H}}_b)$ {reconcile the closing cache}
return $(\tilde{x}_{b+1}, \mathcal{B}_b)$

Cache reconciliation. $\mathrm{Update}(\mathcal{B}^p, b, \mathcal{H})$ rebuilds the predictor state from the frozen past state and the supplied endpoint features; for TaylorSeer, it rebuilds the finite-difference coefficients at b. At each cache site, let $D_a^0, \ldots, D_a^{K_a}$ be the available coefficients and q the maximum order. The update is

$$D_b^0 = \mathcal{H}, \qquad D_b^k = \frac{D_b^{k-1} - D_a^{k-1}}{b-a}, \quad k = 1, \ldots, \min(q, K_a + 1).$$

Both predictor states remain fixed during replay. With the linear weight and Taylor prediction, $w_b = 1$ and $\tilde{\mathcal{H}}_b = \mathcal{H}_b$. This bank was observed on the provisional input to step b; it is not a new FULL evaluation on the replayed input. The replayed output latent can therefore differ from $\bar{x}_{b+1}$.

Sampling-loop integration. Execute the initial FULL step before opening a window, then use each returned $(\tilde{x}_{b+1}, \mathcal{B}_b)$ as the next left boundary. Warm-up FULL steps follow the base schedule. Every scheduled closing step is completed even if no later closed window remains. A cached tail with no scheduled right FULL follows the base predictor without reflection; no extra endpoint FULL is inserted. The procedure preserves the FULL schedule while adding cached replay and scheduler operations.

A.5 Impact Statement

This work improves the efficiency of diffusion-model inference via a training-free caching-and-reflection strategy (*Z-Cache*). By reducing the compute and latency required per generated sample, it can lower energy use and broaden access to high-quality generative models for research and practical applications. However, cheaper and faster generation may also enable misuse (e.g., large-scale misinformation, deceptive or non-consensual synthetic media, and spam) and can amplify existing concerns such as bias or copyright infringement. *Z-Cache* does not change the underlying model's training data or safety properties, so responsible deployment should rely on standard safeguards such as access controls, rate limiting, moderation/filtering, and provenance mechanisms where appropriate.

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: main claims made in the abstract and introduction accurately reflect the paper's contributions.

   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: discussed in experiments.

   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: [Yes]

Justification: provided in method.

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: provided in appendix.

   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: will open on github.

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: provided in appendix.

   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: [Yes]

   Justification: discussed in experiments.

   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: [Yes]

   Justification: provided in appendix.

   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: confirmed.

   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: [Yes]

    Justification: discussed in impact statement.

    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: not applicable.

    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: [Yes]

    Justification: will ship along with code on github.

    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: documented in appendix.

    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: not applicable.

    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: 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: No LLM was used as an important, original, or non-standard component of the method, experiments, or evaluation.

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.