LITECA: MEMORY-EFFICIENT FEATURE CACHING FOR DIFFUSION ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Feature caching accelerates Diffusion Transformers (DiTs) by reusing intermediate features across timesteps. Recently Forecasting-based methods show that module features evolve predictably across timesteps, enabling temporal extrapolation for feature forecasting. However, storing module-level features and their historical states introduces substantial cache memory overhead. To solve this problem, we propose **LiteCa**, a training-free and plug-and-play memory-efficient feature caching framework. We observe that the Adaptive Layer Normalization (AdaLN) modulation parameters exhibit a low-rank structure, enabling LiteCa to build a more compact cache representation with an adjustable rank for different memory budgets. We further introduce Test-Time Correction to improve generation quality without changing the original forecasting method. Experiments on recent DiT models show that LiteCa reduces feature-cache memory by **90–99%** compared with module-level caching, while improving both inference speed and generation quality and achieving state-of-the-art performance.

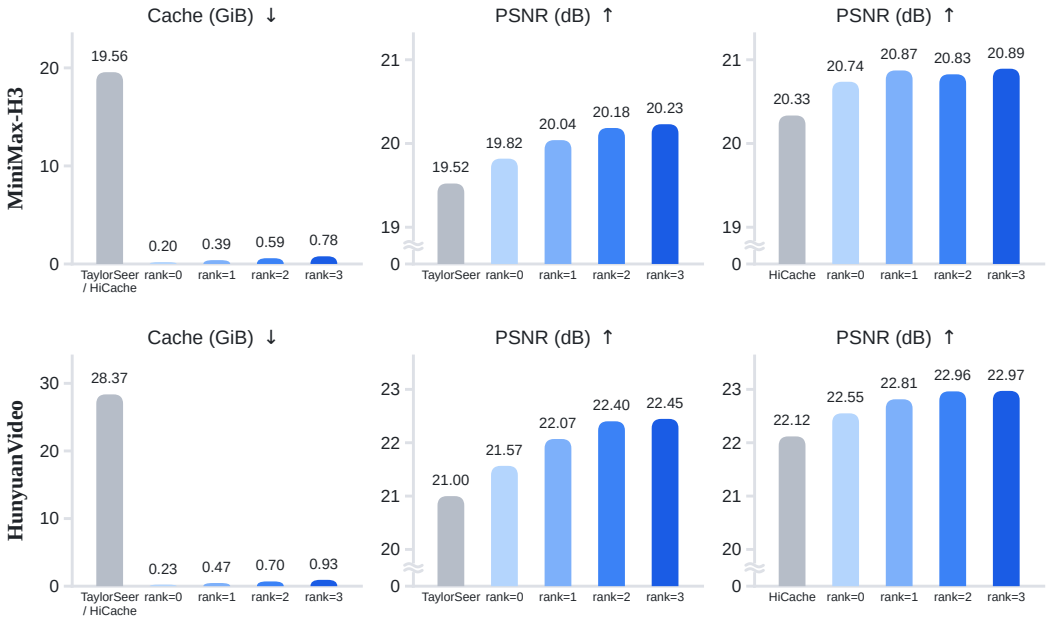
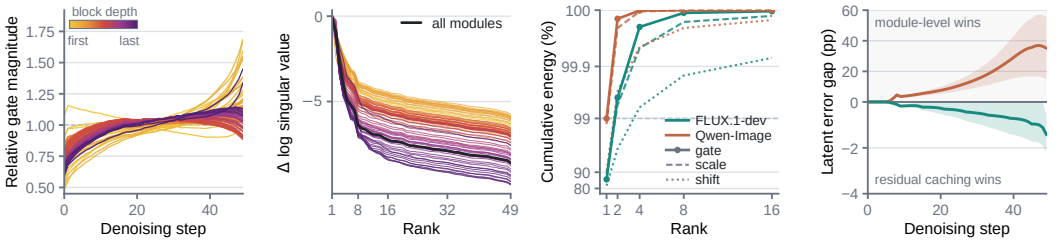


Figure 1: **LiteCa substantially reduces feature-cache memory while improving generation quality.** LiteCa is applied to TaylorSeer and HiCache on MiniMax-H3 and HunyuanVideo without changing their original prediction mechanisms. Since TaylorSeer and HiCache use the same cached feature representation, cache memory is reported once for each model. Across all settings, LiteCa reduces feature-cache memory by **96–99%** while achieving higher generation quality at every evaluated rank. Higher ranks provide further quality gains with only a small linear increase in cache memory.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) AdaLN gate trajectories **(b) Per-module spectrum** **(c) Cumulative energy** **(d) Caching granularity**

Figure 2: **AdaLN modulation exhibits a strong low-rank structure, and the error difference between residual-level and module-level caching varies across models.** (a) The gates in AdaLN follow highly consistent trajectories over denoising timesteps, each curve shown relative to its own time average. (b) In addition, the singular-value spectra of individual modules also decay rapidly, indicating a strong low-rank structure in the trajectory matrix formed across modules/channels and timesteps. (c) We further analyze the effective rank of AdaLN modulation parameters on FLUX.1-dev and Qwen-Image. The cumulative energy results show that the gate, scale, and shift parameters retain most of their energy with only a few ranks, with consistent behavior across both models, suggesting that low-rank structure is a common property of AdaLN modulation across architectures. (d) Meanwhile, the relative error between residual-level and module-level caching shows different trends across models, indicating distinct advantages of the two caching granularities and motivating a unified representation that supports different caching granularities, including both residual-level and module-level caching.

1 Introduction

Diffusion models (Ho et al., 2020; Rombach et al., 2022) have achieved remarkable success in generative tasks, including text-to-image and video generation. Diffusion Transformers (DiTs) (Peebles & Xie, 2023) have further demonstrated strong generation performance. However, their iterative denoising process requires repeated execution of large neural networks across multiple sampling timesteps, resulting in substantial computational cost and inference latency. Existing acceleration methods include knowledge distillation (Salimans & Ho, 2022; Song et al., 2023), quantization (Li et al., 2024b), sparsification, efficient attention (Xi et al., 2025), and token reduction (Bolya & Hoffman, 2023b).

Feature caching methods (Ma et al., 2024b; Selvaraju et al., 2024; Chen et al., 2024b; Zhao et al., 2025b; Liu et al., 2025b; Ma et al., 2026) exploit the high redundancy of intermediate features across adjacent timesteps by reusing previously computed intermediate features. Further studies (Liu et al., 2025e; Feng et al., 2026; Chen et al., 2026b) show that module features follow predictable temporal evolution, allowing future features to be extrapolated from historical information.

Despite reducing the computation of expensive attention and feed-forward modules, module-level caching requires storing the intermediate features of individual modules. Moreover, higher-order extrapolation requires retaining additional historical features, causing the cache memory overhead to increase substantially with the extrapolation order. This memory-for-computation trade-off becomes a major challenge in practical deployment, where GPU memory is often limited. This issue is particularly severe for high-resolution image generation, long-term video generation, and recent DiT models with dozens of Transformer layers. In large video generation models, the cache memory overhead can reach tens of gigabytes, easily leading to out-of-memory (OOM) errors and may even require multi-GPU deployment, limiting the practical use of these methods on resource-constrained hardware. Recent methods (Liu et al., 2025c; Guan et al., 2025) reduce cache memory by merging residual features across modules into a single cache. While this design substantially reduces the number of cached features and the memory overhead, it also removes the fine-grained control of module-level caching over individual modules. This raises an important question: *Can we preserve the fine-grained flexibility of module-level caching without storing the intermediate features of each module separately?*

To address this question, we revisit the role of Adaptive Layer Normalization (AdaLN) in DiTs. Despite recent studies investigating AdaLN from various perspectives (Chen et al., 2024a; Zhu et al., 2026; Gan et al., 2026), its structure across diffusion timesteps remains underexplored. To the best of our knowledge, we are the first to reveal that AdaLN modulation parameters exhibit a strong low-rank temporal structure shared across modules and channels: only a few temporal components capture nearly all of the variation in their gate trajectories, both within individual modules and when all modules are jointly represented using a shared basis (Figure 2(b,c)). This low-rank structure enables a more compact representation of the gates in AdaLN. Based on this observation, we propose LiteCa, a plug-and-play, memory-efficient feature caching framework for existing linear extrapolation-based caching methods. LiteCa uses the low-rank structure of AdaLN to combine module features into a few aggregated feature caches and one residual feature cache. This representation unifies residual caching and module-level caching, with only a few aggregated feature caches needed to closely approximate module-level caching.

However, better approximating module-level caching does not necessarily improve generation quality. To address this, LiteCa introduces Test-Time Correction, which uses the existing full-computation steps to estimate a correction coefficient and adjust the contribution of the aggregated features. This allows LiteCa to go beyond the interpolation between residual caching and module-level caching and further improve generation quality without changing the original forecasting method.

LiteCa separately controls cache capacity and feature contribution, enabling flexible rank adjustment for different memory budgets.

- **A Unified Memory-Efficient Feature Caching Framework.** We propose LiteCa, a training-free and plug-and-play feature caching framework that unifies residual caching and module-level caching within a compact representation. LiteCa substantially reduces cache memory overhead and supports flexible memory budgets.
- **Low-Rank AdaLN with Test-Time Correction.** We reveal a low-rank structure in the AdaLN modulation parameters, which forms the basis of LiteCa's compact cache representation. We further introduce **Test-Time Correction** to improve generation quality without changing the original forecasting method.
- **General Applicability and Superior Performance.** Extensive experiments across six diverse image and video generation models demonstrate the general applicability of LiteCa. LiteCa reduces cache memory overhead by **90–99%** while also improving generation quality and inference speed.

2 Related Work

Accelerating diffusion sampling. Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Rombach et al., 2022) and their transformer backbones (Peebles & Xie, 2023; Esser et al., 2024) generate high-quality images and videos (Saharia et al., 2022; Kong et al., 2024; Wan et al., 2025) at the price of many evaluations of a large network per sample. Fewer evaluations are obtained with better solvers and schedules (Song et al., 2021; Lu et al., 2022; Zheng et al., 2023; Li et al., 2023; Shih et al., 2024), straighter trajectories (Lipman et al., 2023; Liu et al., 2023) or distillation (Salimans & Ho, 2022; Song et al., 2023; Sauer et al., 2024); cheaper evaluations through quantization (Shang et al., 2023; He et al., 2024; Li et al., 2024b), pruning (Fang et al., 2023; Zhu et al., 2024), token reduction (Bolya & Hoffman, 2023a; Kim et al., 2024), sparse or low-bit attention (Xi et al., 2025; Zhang et al., 2025c; Yang et al., 2026; Zhang et al., 2025b) and multi-device parallelism (Li et al., 2024a; Fang et al., 2024). These techniques modify the sampler or the weights, or act *within* a step; feature caching leaves both untouched and skips work *across* steps, so it composes with all of them. LiteCa keeps this property and targets the memory that caching itself introduces.

Feature caching: from reuse to forecasting. Feature caching exploits the redundancy of intermediate features across adjacent timesteps (see Liu et al., 2025d, for a survey). Reuse-based methods store the features of one step and read them back at the following ones, first for U-Nets (Ma et al., 2024b; Li et al., 2024c; So et al., 2024; Wimbauer et al., 2024) and then per module or block (Selvaraju et al., 2024; Chen et al., 2024b; Zhao et al., 2025b; Ma et al., 2024a; Liu et al., 2024b;a; Kim et al., 2025; Zhang et al., 2025a) or per token (Zou et al., 2024; 2026b; Zheng et al., 2025b; Sun et al., 2025; Liu et al., 2026c; Silveria et al., 2025; Fang et al., 2026; Zhang et al., 2024; Saghatchian et al., 2025) of a DiT, and have been carried over to editing (Yu et al., 2024; Yan et al., 2025). Much

of the subsequent work decides *when* to cache, replacing fixed intervals by schedules that track the change of inputs or outputs (Kahatapitiya et al., 2024; Lv et al., 2024; Zhou et al., 2025; Liu et al., 2025b; Ma et al., 2026; Chung et al., 2026; Bu et al., 2026; Gao et al., 2025; Gu et al., 2026; Son et al., 2026; Chu et al., 2025; Adnan et al., 2025; Shen et al., 2025; Cao et al., 2026). Forecasting-based methods change *what* is read back: they extrapolate each module's history to the current step with Taylor (Liu et al., 2025e), Hermite (Feng et al., 2026), multistep or higher-order difference (Yu et al., 2025; Cui et al., 2026; Liu et al., 2026b), ODE (Zheng et al., 2026), per-dimension (Zheng et al., 2025a), speculative (Liu et al., 2025f) or learned (Zou et al., 2026a) rules, and keep quality at much larger skip intervals. LiteCa proposes neither a schedule nor a forecasting rule: it attaches to any host whose rule is *linear* in the cached history, reproduces the host's schedule and predictions, and therefore adds to progress on either axis.

Cache memory and caching granularity. Cache memory is set by its *granularity*. Module-level methods store every attention and feed-forward feature, and forecasting multiplies this by the retained history, so the cache grows with the module count and the extrapolation order (Guan et al., 2025; Peng et al., 2025; Lu et al., 2026); on the video DiTs we evaluate it reaches tens of gigabytes. Residual caching keeps one aggregated residual for the whole stack (Liu et al., 2025b; Ma et al., 2026; Chung et al., 2026; Zhou et al., 2025): its memory does not grow with the module count, but it cannot treat modules differently. Memory-aware forecasting caches a cumulative residual (Liu et al., 2025c), forecasts only the last block (Guan et al., 2025), extrapolates from a few full-step outputs (Lu et al., 2026) or caches hidden states (Liu et al., 2025a); SVD-Cache (Chen et al., 2026a) and LearniBridge (Huang et al., 2026) compress the cached features into a low-rank subspace. LiteCa keeps module-level forecasting but replaces the L per-module caches by one residual cache and a few gate-weighted aggregated caches, so residual and module-level caching are the two ends of one representation. The low-rank structure lies in the AdaLN *gate trajectories*, known before denoising, rather than in the features.

Correcting cached features. Several methods correct or rescale cached features: a learned scale-and-shift (Wimbauer et al., 2024), offline per-block scaling (Gu et al., 2026), low-rank parameters derived from the weights (Chen et al., 2025), error-optimized updates (Qiu et al., 2025), a Heun corrector (Zheng et al., 2026), closed-form residual rectification (Peng et al., 2025), probe-based trajectory weights (Bu et al., 2026), on-the-fly offset correction (Liu et al., 2026a) or learned calibration modules (Huang et al., 2026; Xie et al., 2026). These corrections are fitted offline, learned from data, or applied in feature space. Test-Time Correction instead fits, *online* and only at the anchor steps the host already computes in full, a coefficient on a *gate-derived* direction by linearizing the readout; the coefficient interpolates between residual and module-level caching and can extrapolate beyond both, without calibration data or training.

AdaLN modulation in DiTs. AdaLN-Zero (Peebles & Xie, 2023) injects the timestep through per-module scale, shift and gate vectors, whose projections account for a large share of DiT parameters. This redundancy has been reduced by sharing one modulation across blocks (Chen et al., 2024a; Wan et al., 2025; HaCohen et al., 2026) or adding a low-rank per-block adjustment (Gupta et al., 2024; Agarwal et al., 2025; Zhao et al., 2025a), and studied through the gate's design (Zhu et al., 2026), its role in massive activations (Gan et al., 2026) and the timestep dependence of modulation and activation statistics, which quantization (Huang et al., 2024a; Wu et al., 2024) and caching policies (Liu et al., 2025b) exploit. Closest to our observation, DiMR (Liu et al., 2024c) finds that the scale and shift of a class-conditional DiT move along two principal components over the sampling steps and replaces the modulation network with a time-dependent layer norm, and MTEO (Cai et al., 2026) shows that the timestep embedding itself is nearly two-dimensional. LiteCa shows that the *joint* gate trajectory over all modules and channels of large text-to-image and text-to-video DiTs is captured by a few components, and uses this structure *at inference* to aggregate feature caches.

3 METHOD

3.1 PRELIMINARIES

Diffusion Transformer Architecture. DiT (Peebles & Xie, 2023) consists of L Transformer modules, denoted by $\mathcal{F}_\ell$. AdaLN modulates each module using timestep-dependent scale $\boldsymbol{\gamma}_\ell(t)$, shift $\boldsymbol{\beta}_\ell(t)$, and gate $\mathbf{g}_\ell(t)$ (α in Peebles & Xie, 2023), all in $\mathbb{R}^C$. Specifically,

$$\mathbf{f}_\ell(t) = \mathcal{F}_\ell\big((1+\boldsymbol{\gamma}_\ell(t)) \odot \mathrm{LN}(\mathbf{h}_{\ell-1}) + \boldsymbol{\beta}_\ell(t)\big), \qquad \mathbf{h}_\ell = \mathbf{h}_{\ell-1} + \mathbf{g}_\ell(t) \odot \mathbf{f}_\ell(t), \tag{1}$$

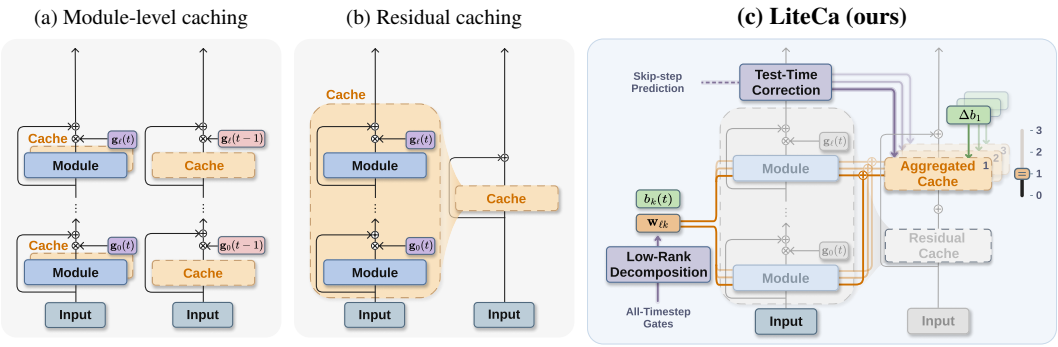


Figure 3: **Caching granularities and LiteCa.** Module-level caching (a) stores one cache per module, whereas residual caching (b) collapses them into one. LiteCa (c) retains one residual cache and K gate-derived aggregated caches, whose contributions are calibrated online by Test-Time Correction.

where C is the channel width and $\odot$ denotes per-channel multiplication broadcast over tokens. Summing the residual updates across all modules gives

$$\mathbf{h}(t) = \mathbf{h}_{\text{in}}(t) + \mathbf{r}(t), \qquad \mathbf{r}(t) = \sum_{\ell=1}^{L} \mathbf{g}_\ell(t) \odot \mathbf{f}_\ell(t). \tag{2}$$

Feature Caching. Feature caching computes the network only at sparse *anchor steps* $\mathcal{A} = \{a_1 < a_2 < \cdots\}$ and estimates *skip-step* features by reuse or Taylor-, Hermite-, or ODE-based extrapolation (Liu et al., 2025e; Feng et al., 2026; Chen et al., 2026b). We denote the host extrapolator by $P_t[\cdot]$ and the most recent anchor before t by a; applying P_t independently to all L modules requires storing every module's history.

LiteCa assumes that the AdaLN trajectories are available before denoising and admit a low-rank approximation, and that P_t is linear in its cached features. Appendix B states these conditions formally and discusses supported streams and classifier-free guidance.

3.2 LiteCa

LiteCa uses low-rank AdaLN gate trajectories to replace L module caches with $K+1$ aggregated caches and calibrates their contributions online at existing anchor steps.

3.2.1 Low-Rank Cache Aggregation

Module-level caching vs. Residual caching. Let $\mathbf{f}_\ell(a)$ denote the output feature of module ℓ at anchor step a, $\mathbf{g}_\ell(t)$ its AdaLN gate at skip step t, and $P_t[\cdot]$ the linear temporal extrapolator. Module-level caching extrapolates each module feature separately and applies the current gate, while residual caching extrapolates the aggregated residual:

$$\hat{\mathbf{r}}_{\text{M}}(t) = \sum_{\ell=1}^{L} \mathbf{g}_\ell(t) \odot P_t\big[\mathbf{f}_\ell(a)\big], \qquad \hat{\mathbf{r}}_{\text{R}}(t) = P_t\Big[\sum_{\ell=1}^{L} \mathbf{g}_\ell(a) \odot \mathbf{f}_\ell(a)\Big]. \tag{3}$$

By the linearity of P_t, their relation can be written as

$$\hat{\mathbf{r}}_{\text{M}}(t) = \hat{\mathbf{r}}_{\text{R}}(t) + P_t\Big[\sum_{\ell} \big(\mathbf{g}_\ell(t) - \mathbf{g}_\ell(a)\big) \odot \mathbf{f}_\ell(a)\Big]. \tag{4}$$

Equation 4 shows that the difference between the two caching forms comes from the change of the AdaLN gates between the anchor and skip steps. Module-level caching preserves these module-wise gate variations but requires storing individual module features, while residual caching stores only the aggregated residual and therefore loses this fine-grained information.

Low-Rank Gate Decomposition. We observe that AdaLN gate trajectories are low-rank across timesteps. We center the gate trajectories and apply a rank-K truncated SVD:

$$G = \big[\, \mathbf{g}_\ell(t) - \bar{\mathbf{g}}_\ell \,\big]_{\ell,t} \approx U_K \Sigma_K V_K^\top, \qquad \mathbf{g}_\ell(t) \approx \bar{\mathbf{g}}_\ell + \sum_{k=1}^{K} b_k(t)\, \mathbf{w}_{\ell k}, \tag{5}$$

where $\mathbf{w}_{\ell k}$ is the module-wise weight and $b_k(t)$ is the timestep-dependent basis. Therefore,

$$\mathbf{g}_\ell(t) - \mathbf{g}_\ell(a) \approx \sum_{k=1}^{K} \big[b_k(t) - b_k(a)\big]\, \mathbf{w}_{\ell k}. \tag{6}$$

Aggregated Cache Construction. Substituting Equation 6 into Equation 4 allows the module features to be aggregated into one residual feature cache and K aggregated feature caches:

$$\mathbf{C}_{\text{res}}(a) = \sum_{\ell} \mathbf{g}_\ell(a) \odot \mathbf{f}_\ell(a), \qquad \mathbf{C}_k(a) = \sum_{\ell} \mathbf{w}_{\ell k} \odot \mathbf{f}_\ell(a). \tag{7}$$

Since $\hat{\mathbf{r}}_{\text{R}}(t) = P_t\big[\mathbf{C}_{\text{res}}(a)\big]$, LiteCa predicts

$$\hat{\mathbf{r}}_{\text{LiteCa}}(t) = \hat{\mathbf{r}}_{\text{R}}(t) + \sum_{k=1}^{K} P_t\Big[\big[b_k(t) - b_k(a)\big]\, \mathbf{C}_k(a)\Big] \;\approx\; \hat{\mathbf{r}}_{\text{M}}(t). \tag{8}$$

LiteCa therefore replaces L module-level caches with only $K+1$ caches per history node. When $K = 0$, LiteCa corresponds to residual caching, while increasing K provides a closer approximation to module-level caching.

3.2.2 Test-Time Correction

Although the low-rank representation approximates module-level caching, a closer approximation does not necessarily lead to better generation quality. We therefore introduce *Test-Time Correction* (TTC) to adjust the contribution of each aggregated feature component without changing the cached features or the original extrapolation rule.

Let the k-th aggregated correction component be

$$\mathbf{r}_k(t) = P_t\Big[\big[b_k(t) - b_k(a)\big]\, \mathbf{C}_k(a)\Big]. \tag{9}$$

For each component, LiteCa introduces a correction coefficient λ_k. At a full-computation anchor a, the not-yet-updated history produces the counterfactual skip state $\hat{\mathbf{h}}(a; \boldsymbol{\lambda})$; with $\mathbf{h}_{\text{F}}(a)$ the full-computation state and J_a the readout Jacobian there, we solve

$$\hat{\boldsymbol{\lambda}}_a = \arg\min_{\boldsymbol{\lambda}} \Big\| J_a\big[\hat{\mathbf{h}}(a; \boldsymbol{\lambda}) - \mathbf{h}_{\text{F}}(a)\big] \Big\|_2^2. \tag{10}$$

The coefficients are reused until the next anchor; Appendix B gives the full derivation and implementation details. The complete LiteCa prediction is

$$\boxed{\hat{\mathbf{r}}_{\text{LiteCa}}(t; \boldsymbol{\lambda}) = \hat{\mathbf{r}}_{\text{R}}(t) + \sum_{k=1}^{K} \lambda_k\, \mathbf{r}_k(t)} \tag{11}$$

where $\lambda_k = 0$ removes the corresponding aggregated component and $\lambda_k = 1$ preserves its original low-rank contribution. The coefficients are not restricted to $[0, 1]$, allowing LiteCa to go beyond a simple interpolation between residual caching and module-level caching.

Table 1: **Quantitative comparison in text-to-image generation** on FLUX.1-dev and Qwen-Image-2512. Z-Image is reported in Table 7.

Model	Method	Efficiency				Visual Retention		
		Cache (GiB)↓	Reduction (%)↑	Latency (s)↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
FLUX.1-dev	Full-step (100% steps)	—	—	16.305	1.00×	—	—	—
	TaylorSeer (FE=3, N=6, O=2)	6.012	—	4.263	3.83×	16.0339	0.6501	0.3899
	LiteCa-TS (rank 1)	0.141	97.66	3.345	4.88×	**16.9191**	**0.6797**	**0.3625**
	HiCache (FE=3, N=6, O=2)	6.012	—	4.277	3.81×	16.8502	0.6773	**0.3479**
	LiteCa-Hi (rank 1)	0.141	97.66	3.343	4.88×	**17.2545**	**0.6917**	0.3544
Qwen-Image 2512	Full-step (100% steps)	—	—	31.526	1.00×	—	—	—
	TaylorSeer (FE=5, N=6, O=2)	16.926	—	11.566	2.73×	14.655	0.556	0.424
	LiteCa-TS (rank 2)	0.425	97.49	10.039	3.14×	**18.194**	**0.694**	**0.282**
	HiCache (FE=5, N=6, O=2)	16.926	—	12.959	2.43×	18.063	0.650	0.284
	LiteCa-Hi (rank 1)	0.284	98.32	9.892	3.19×	**20.017**	**0.753**	**0.249**

Table 2: **Quantitative comparison in text-to-video generation** on HunyuanVideo, Wan2.1 and MiniMax-H3.

Model	Method	Efficiency				Visual Retention		
		Cache (GiB)↓	Reduction (%)↑	Latency (s)↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
HunyuanVideo	Full-step (100% steps)	—	—	134.023	1.00×	—	—	—
	TaylorSeer (FE=5, N=5, O=1)	28.367	—	42.987	3.12×	20.9992	0.7440	0.2187
	LiteCa-TS (rank 3)	0.934	96.71	39.023	3.43×	**22.4468**	**0.7807**	**0.1907**
	HiCache (FE=5, N=5, O=1)	28.367	—	41.736	3.21×	22.1183	0.7633	0.1969
	LiteCa-Hi (rank 3)	0.934	96.71	39.019	3.43×	**22.9715**	**0.7876**	**0.1923**
Wan2.1	Full-step (100% steps)	—	—	131.113	1.00×	—	—	—
	TaylorSeer (FE=8, N=5, O=1)	33.742	—	43.620	3.01×	21.9269	0.8083	0.1332
	LiteCa-TS (rank 1)	1.500	95.56	42.687	3.07×	**21.9881**	**0.8121**	**0.1319**
	HiCache (FE=8, N=5, O=1)	33.742	—	46.066	2.85×	23.7717	0.8381	**0.1057**
	LiteCa-Hi (rank 0)	0.750	97.78	42.369	3.09×	**23.8067**	**0.8437**	0.1076
MiniMax-H3	Full-step (100% steps)	—	—	123.70	1.00×	—	—	—
	TaylorSeer (FE=8, N=5, O=1)	19.560	—	45.05	2.75×	19.522	0.6428	0.2453
	LiteCa-TS (rank 3)	0.782	96.00	41.91	2.95×	**20.230**	**0.6586**	**0.2371**
	HiCache (FE=8, N=5, O=1)	19.560	—	43.32	2.86×	20.334	0.6635	0.2245
	LiteCa-Hi (rank 1)	0.391	98.00	40.97	3.02×	**20.872**	**0.6758**	**0.2227**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Model Configurations. We evaluate LiteCa across five state-of-the-art diffusion models spanning two representative generative tasks: text-to-image generation with FLUX.1-dev (Black Forest Labs, 2024), Qwen-Image-2512 (Wu et al., 2025), and text-to-video generation with HunyuanVideo (Kong et al., 2024), Wan2.1 (Wan et al., 2025), and MiniMax-H3 (MiniMax, 2026), demonstrating that LiteCa provides a general solution for reducing cache memory overhead and improving overall inference efficiency across different models and tasks. Benchmarks, resolutions, frame counts, seeds, metrics and hardware are listed in Appendix A; throughout, a cached run is compared against *full-step inference*: the same model with every denoising step computed and no feature cache (the FULL-STEP row of every table).

4.2 EXPERIMENT RESULTS

Text-to-Image Generation. Table 1 compares LiteCa with its two module-level hosts on two image models. Across the evaluated ranks, LiteCa reduces the persistent feature cache by 90.6–99.98%. The selected configurations improve PSNR and SSIM in all six model–host pairs of the three models, and LPIPS in five, while achieving 2.80–$4.88\times$ speedup over full-step inference. Figure 4 shows the corresponding qualitative comparisons on FLUX.1-dev and Qwen-Image.

Text-to-Video Generation. Table 2 extends the comparison to three video models. LiteCa reduces the feature cache by 91.1–99.2% and is faster than both hosts on every model. It improves all three

Table 3: **Comparison with existing memory-efficient caching methods** on FLUX.1-dev over DrawBench-200. Full calls is the average number of fully computed steps, and cache includes the gate basis. CG-Taylor and FreqCa timings were collected separately from the other rows, so latency is compared only within paired runs. **Bold** and underline mark the best and second-best quality results within each family.

Family	Method	Efficiency			Visual Retention		
		Full calls↓	Cache (GiB)↓	Latency (s)↓	PSNR↑	SSIM↑	LPIPS↓
—	Full-step (100% steps)	50.000	—	16.305	—	—	—
Taylor	TaylorSeer (FE=3, N=6, O=2)	10.000	6.012	4.263	16.0339	0.6501	0.3899
	+ LiteCa (R=1)	10.000	0.141	3.345	16.9191	0.6797	0.3625
	CG-Taylor (τ=0.30)	10.225	0.094	4.078	17.5287	0.6460	0.4228
	+ LiteCa (R=1)	10.145	0.142	4.036	**19.1485**	**0.7156**	**0.3329**
Hermite	HiCache (FE=3, N=6, O=2)	10.000	6.012	4.277	16.8502	0.6773	0.3479
	+ LiteCa (R=1)	10.000	0.141	3.343	**17.2545**	**0.6917**	0.3544
	FreqCa (own schedule)	10.000	0.141	3.291	17.1586	0.6800	**0.3469**
	+ LiteCa (R=1)	10.000	0.282	3.451	17.2247	0.6837	0.3475

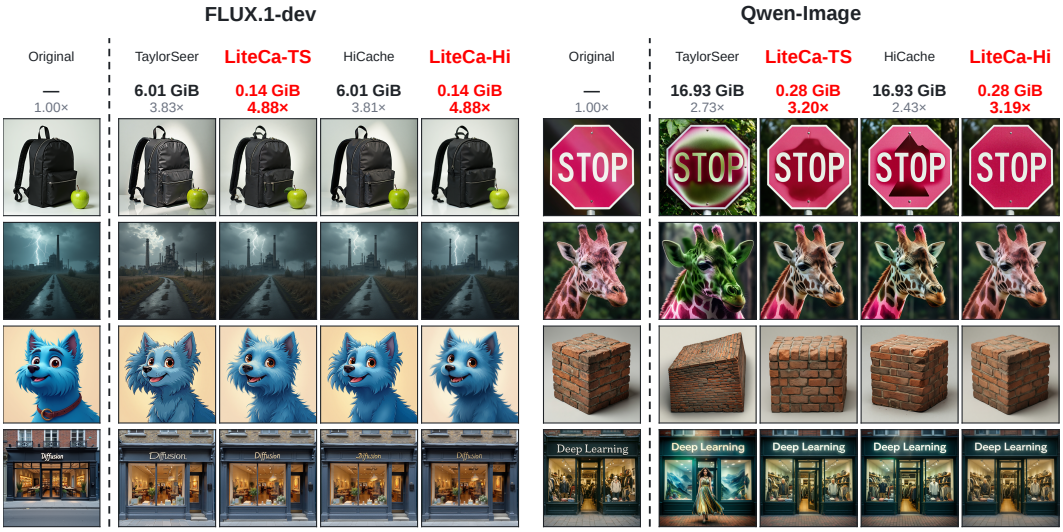


Figure 4: **Qualitative comparison for text-to-image generation on DrawBench (Saharia et al., 2022) prompts using FLUX.1-dev (Black Forest Labs, 2024) and Qwen-Image (Wu et al., 2025).** FLUX.1-dev runs FE=3 and Qwen-Image FE=5, both with N=6 and O=2, and both LiteCa columns are R=1. The two lines under each column are that setting's feature cache and its speedup over full-step inference, taken from Table 7.

quality metrics on HunyuanVideo and MiniMax-H3, while remaining within 0.2 dB of the hosts on Wan2.1. Figure 5 further shows that LiteCa remains visually closer to the full-step MiniMax-H3 output under an aggressive skip setting where the hosts degrade visibly.

4.3 Ablation Study

Comparison with Existing Memory-Efficient Methods. We compare LiteCa with CG-Taylor (Guan et al., 2025), which improves efficiency by adapting the schedule for Taylor-based forecasting, and FreqCa (Liu et al., 2025c), which adopts frequency-aware Hermite forecasting. Unlike these methods, LiteCa compresses the cache representation without modifying the host schedule or predictor. When applied only to the original Taylor and Hermite hosts, LiteCa already achieves performance comparable to these specialized methods. To demonstrate LiteCa's plug-and-play capability, we further integrate it into both methods. Its TTC uses only a few additional low-rank cache components to improve performance while preserving the original schedule and predictor (Table 3). The complete rank sweep is reported in Appendix C.

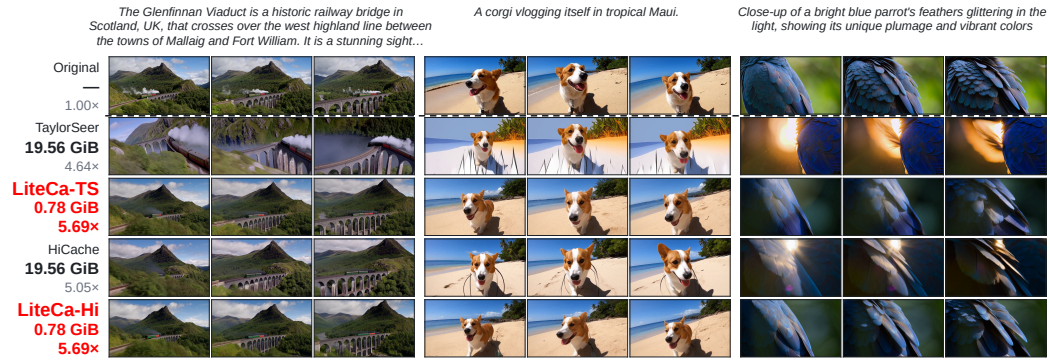
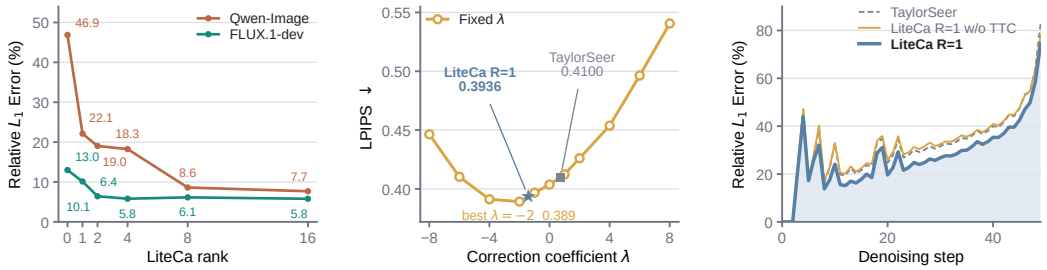


Figure 5: **Qualitative comparison for text-to-video generation on MovieGen (Polyak et al., 2024) prompts using MiniMax-H3 (MiniMax, 2026).**



(a) Number of low-rank components **(b) Fixed λ vs. Test-Time Correction** **(c) Per-step error**

Figure 6: **Ablation studies.** (a) Final-latent relative L_1 error to module-level TaylorSeer versus LiteCa rank, with Test-Time Correction disabled, on FLUX.1-dev and Qwen-Image. (b) LPIPS against full-step FLUX.1-dev under fixed correction coefficients λ, with TaylorSeer and online Test-Time Correction shown as references. (c) Per-step relative L_1 output error on FLUX.1-dev for TaylorSeer and LiteCa with and without Test-Time Correction. Lower is better.

Number of Low-Rank Components. We disable Test-Time Correction, so its online update remains zero, and vary only the LiteCa rank. Figure 6a shows that the error relative to module-level TaylorSeer falls rapidly with only a few components, confirming that the low-rank structure in Figure 2c persists during inference.

Effect of the Correction Coefficient. We observe that the best performance does not necessarily occur at a correction coefficient of 0 or 1. Here, $\lambda = 0$ corresponds to residual caching, while $\lambda = 1$ simply adds the low-rank component to approximate module-level caching. We therefore compare different fixed correction coefficients and find that the best setting can lie away from both endpoints (Figure 6b). Test-Time Correction estimates the correction coefficient online without offline search or calibration. As shown in Figure 6c, and per prompt in Figure 11, Test-Time Correction consistently reduces the prediction error at skipped steps.

Further analysis. Appendix E reports LiteCa's GPU memory usage under each model's default generation setting.

5 Conclusion

LiteCa turns the gate structure of DiTs into a memory knob for feature caching. Because the gates are affine images of a shared timestep embedding, known before denoising and spanning far fewer components than there are steps, the gate variation separating module-level from aggregated-residual caching needs only a few shared components, not many per-module caches, with a calibrated λ setting its strength. Across six image and video DiTs it matches or improves on both hosts, cutting the audited feature caches by 90–99%. *Limitations:* no principled cross-stream rank-allocation rule; Assumptions 1–2 exclude per-token noise levels and non-AdaLN models; offloading and low-precision storage are not benchmarked.

AI USE STATEMENT

In this work, we used generative AI tools for literature retrieval, for drafting and editing the manuscript text, and for writing figure-plotting and result-aggregation scripts. We have not used generative AI tools for generating research ideas, for designing or running experiments, or for producing experimental results. We have reviewed all AI-assisted work: every cited reference was checked against its original source by the authors, all LLM-edited text was read and corrected by the authors, and all AI-assisted scripts were verified against independently computed values. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies inference efficiency of existing open-weight generative models and introduces no new training data, human-subject study, or personally identifiable information. Faster and less memory-hungry inference lowers the barrier to deploying generative models, which carries the same dual-use considerations as the underlying models; LiteCa does not alter their content or safety behavior.

REPRODUCIBILITY STATEMENT

Appendix A reports the evaluation protocol, while the assumptions behind the exactness claims are stated in Section 3. Code and configuration files will be released at [URL] (anonymous link in the supplementary material at submission).

REFERENCES

Muhammad Adnan, Nithesh Kurella, Akhil Arunkumar, and Prashant J. Nair. Foresight: Adaptive layer reuse for accelerated and high-quality text-to-video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2506.00329.

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

Black Forest Labs. FLUX.1 [dev]. `https://github.com/black-forest-labs/flux`, 2024.

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *2023 IEEE/CVF conference on computer vision and pattern recognition workshops (CVPRW)*, pp. 4599–4603. IEEE, 2023a.

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023b. arXiv:2303.17604.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Tong Wu, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. In *International Conference on Learning Representations*, 2026.

Yuren Cai, Guangyi Wang, Zongqing Li, Li Li, Zhihui Liu, and Songzhi Su. Three creates all: You only sample 3 steps. *arXiv preprint arXiv:2603.22375*, 2026.

F. Cao, Y. Chen, Z. You, and W. Luo. Procache: Constraint-aware feature caching with selective computation for diffusion transformer acceleration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. arXiv:2512.17298; full first names to verify.

Guantao Chen, Shikang Zheng, Yuqi Lin, and Linfeng Zhang. Forecast the principal, stabilize the residual: Subspace-aware feature caching for efficient diffusion transformers. *arXiv preprint arXiv:2601.07396*, 2026a.

Jiawei Chen et al. Forecast then calibrate: Feature caching as ODE for efficient diffusion transformers. In *AAAI Conference on Artificial Intelligence*, 2026b. arXiv:2508.16211.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International conference on learning representations*, 2024a.

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*, 2024b.

Zhiyuan Chen, Keyi Li, Yifan Jia, Le Ye, and Yufei Ma. Accelerating diffusion transformer via increment-calibrated caching with channel-aware singular value decomposition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. arXiv:2505.05829.

Huanpeng Chu, Wei Wu, Guanyu Feng, and Yutao Zhang. Omnicache: A trajectory-oriented global perspective on training-free cache reuse for diffusion transformer models. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 16302–16312. IEEE, 2025.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. SeaCache: Spectral-evolution-aware cache for accelerating diffusion models. *arXiv preprint arXiv:2602.18993*, 2026.

Hanshuai Cui, Zhiqing Tang, Qianli Ma, Zhi Yao, Weijia Jia, and Wei Zhao. Predict to skip: Linear multistep feature forecasting for efficient diffusion transformers. *arXiv preprint arXiv:2602.18093*, 2026.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2403.03206.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

Haipeng Fang, Yu Li, Fan Tang, Yixing Lu, Juan Cao, and Sheng Tang. Resca: Residual caching for diffusion transformers acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 32957–32966, 2026.

Jiarui Fang, Jinzhe Pan, Xibo Sun, Aoyu Li, and Jiannan Wang. xdit: an inference engine for diffusion transformers (dits) with massive parallelism. *arXiv preprint arXiv:2411.01738*, 2024.

Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, et al. Hicache: A plug-in scaled-hermite upgrade for taylor-style cache-then-forecast diffusion acceleration. In *International Conference on Learning Representations*, 2026.

Chaofan Gan, Yuanpeng Tu, Xi Chen, Tieyuan Chen, Yuxi Li, Mehrtash Harandi, and Weiyao Lin. Unleashing diffusion transformers for visual correspondence by modulating massive activations. *Advances in Neural Information Processing Systems*, 38:114432–114462, 2026.

Huanlin Gao, Ping Chen, Fuyuan Shi, Chao Tan, Zhaoxiang Liu, Fang Zhao, Kai Wang, and Shiguo Lian. Lemica: Lexicographic minimax path caching for efficient diffusion-based video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2511.00090.

Lihui Gu, Jingbin He, Lianghao Su, Kang He, Wenxiao Wang, and Yuliang Liu. Scalingcache: Extreme acceleration of dits through difference scaling and dynamic interval caching. In *The Fourteenth International Conference on Learning Representations*, 2026.

Xiaoliu Guan, Lielin Jiang, Hanqi Chen, Xu Zhang, Jiaxing Yan, Guanzhong Wang, Yi Liu, Zetao Zhang, and Yu Wu. Forecasting when to forecast: Accelerating diffusion models with confidence-gated taylor. *Knowledge-Based Systems*, 2025.

Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Fei-Fei Li, Irfan Essa, Lu Jiang, and José Lezama. Photorealistic video generation with diffusion models. In *European Conference on Computer Vision*. Springer, 2024.

Yoav HaCohen, Benny Brazowski, Nisan Chiprut, Yaki Bitterman, Andrew Kvochko, Avishai Berkowitz, Daniel Shalem, Daphna Lifschitz, Dudu Moshe, Eitan Porat, et al. Ltx-2: Efficient joint audio-visual foundation model. *arXiv preprint arXiv:2601.03233*, 2026.

Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 2024.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2006.11239.

Xuyue Huang, Zhe Chen, Wang Shen, and Xiao-Ping Zhang. Learnibridge: Learnable calibration of feature caching for diffusion models acceleration. *arXiv preprint arXiv:2606.26778*, 2026.

Yushi Huang, Ruihao Gong, Jing Liu, Tianlong Chen, and Xianglong Liu. Tfmq-dm: Temporal feature maintenance quantization for diffusion models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7362–7371. IEEE, 2024a.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S. Ryoo, and Tian Xie. Adaptive caching for faster video generation with diffusion transformers. *arXiv preprint arXiv:2411.02397*, 2024.

Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *WACV*, 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)*, 2025.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Lijiang Li, Huixia Li, Xiawu Zheng, Jie Wu, Xuefeng Xiao, Rui Wang, Min Zheng, Xin Pan, Fei Chao, and Rongrong Ji. Autodiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration. In *ICCV*, 2023.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Ming-Yu Liu, Kai Li, and Song Han. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024a.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024b.

Senmao Li, Taihang Hu, Joost van de Weijer, Fahad Shahbaz Khan, Tao Liu, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of the encoder for diffusion model inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024c. arXiv:2312.09608.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Dong Liu, Jiayi Zhang, Yifan Li, Yanxuan Yu, Ben Lengerich, and Ying Nian Wu. Fastcache: Fast caching for diffusion transformer through learnable linear approximation. *arXiv preprint arXiv:2505.20353*, 2025a.

Dong Liu, Yanxuan Yu, Ben Lengerich, and Ying Nian Wu. Adacorrection: Adaptive offset cache correction for accurate diffusion transformers. *arXiv preprint arXiv:2602.13357*, 2026a.

Enhuai Liu, Yunke Wang, Changming Sun, and Chang Xu. D2cache: Second-order delta caching for higher video diffusion acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 43589–43599, 2026b.

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. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025b.

Haozhe Liu, Wentian Zhang, Jinheng Xie, Francesco Faccio, Mengmeng Xu, Tao Xiang, Mike Zheng Shou, Juan-Manuel Perez-Rua, and Jürgen Schmidhuber. Faster diffusion via temporal attention decomposition. *arXiv preprint arXiv:2404.02747*, 2024a. Accepted to Transactions on Machine Learning Research.

Jiacheng Liu, Peiliang Cai, Qinming Zhou, Yuqi Lin, Deyang Kong, Benhao Huang, Yupei Pan, Haowen Xu, Chang Zou, Junshu Tang, et al. Freqca: Accelerating diffusion models via frequency-aware caching. *arXiv preprint arXiv:2510.08669*, 2025c.

Jiacheng Liu, Xinyu Wang, Yuqi Lin, Zhikai Wang, Peiru Wang, Peiliang Cai, Qinming Zhou, Zhengan Yan, Zexuan Yan, Zhengyi Shi, et al. A survey on cache methods in diffusion models: Toward efficient multi-modal generation. *arXiv preprint arXiv:2510.19755*, 2025d.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2025e.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Fei Ren, Shaobo Wang, Kaixin Li, and Linfeng Zhang. SpeCa: Accelerating diffusion transformers with speculative feature caching. In *ACM International Conference on Multimedia (MM)*, 2025f. arXiv:2509.11628.

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

Qihao Liu, Zhanpeng Zeng, Ju He, Qihang Yu, Xiaohui Shen, and Liang-Chieh Chen. Alleviating distortion in image generation via multi-resolution diffusion models and time-dependent layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024c. arXiv:2406.09416.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2209.03003.

Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026c. arXiv: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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2206.00927.

Chengjie Lu, Tianchi Deng, Zhengqi He, Zhijian Gao, Huisi Wu, and Xueliang Li. Memory-efficient training-free acceleration of diffusion transformers with barycache. *arXiv preprint arXiv:2608.28670*, 2026.

Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. FasterCache: Training-free video diffusion model acceleration with high quality. *arXiv preprint arXiv:2410.19355*, 2024.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a. arXiv:2406.01733.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b. arXiv:2312.00858.

Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. Magcache: Fast video generation with magnitude-aware cache. *Advances in Neural Information Processing Systems*, 38: 34348–34380, 2026.

MiniMax. MiniMax-H3 model card. `https://huggingface.co/MiniMaxAI/MiniMax-H3`, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. arXiv:2212.09748.

Xurui Peng, Hong Liu, Chenqian Yan, Rui Ma, Fangmin Chen, Xing Wang, Zhihua Wu, Songwei Liu, and Mingbao Lin. Ertacache: Error rectification and timesteps adjustment for efficient diffusion. *arXiv preprint arXiv:2508.21091*, 2025.

Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, et al. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

Junxiang Qiu, Shuo Wang, Jinda Lu, Lin Liu, Houcheng Jiang, Xingyu Zhu, and Yanbin Hao. Accelerating diffusion transformer via error-optimized cache. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 9588–9597, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. arXiv:2112.10752.

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

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2202.00512.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, 2024.

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.

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*, 2023.

Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Yanyu Li, Yifan Gong, Kai Zhang, Hao Tan, Jason Kuen, Henghui Ding, Zhihao Shu, Wei Niu, Pu Zhao, Yanzhi Wang, and Jiuxiang Gu. Lazydit: Lazy learning for the acceleration of diffusion transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 20409–20417, 2025. doi: 10.1609/AAAI.V39I19.34248. arXiv:2412.12444.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 2024.

Austin Silveria, Soham V. Govande, and Daniel Y. Fu. Chipmunk: Training-free acceleration of diffusion transformers with dynamic column-sparse deltas. *arXiv preprint arXiv:2506.03275*, 2025.

Junhyuk So, Jungwon Lee, and Eunhyeok Park. FRDiff: Feature reuse for universal training-free acceleration of diffusion models. In *European Conference on Computer Vision (ECCV)*, 2024. arXiv:2312.03517.

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

Hojae Son, Md Ashraful Islam, Huy Gia Cao, Hui Guan, and Marco Serafini. Kalypso: Relational llm serving. *arXiv preprint arXiv:2607.23815*, 2026.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2010.02502.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning (ICML)*, 2023. arXiv:2303.01469.

Wenzhang Sun, Qirui Hou, Donglin Di, Jiahui Yang, Yongjia Ma, and Jianxun Cui. Unicp: A unified caching and pruning framework for efficient video generation. In *Proceedings of the 7th ACM International Conference on Multimedia in Asia*, pp. 1–7, 2025.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2024.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Junyi Wu, Haoxuan Wang, Yuzhang Shang, Mubarak Shah, and Yan Yan. Ptq4dit: Post-training quantization for diffusion transformers. *Advances in neural information processing systems*, 37: 62732–62755, 2024.

Haocheng Xi et al. Sparse VideoGen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

Zhikang Xie, Xichen Ye, Yifan Wu, Haoshen Yu, Peizhu Gong, Weizhong Zhang, Cheng Jin, et al. Onlinecache: Learning dynamic caching policies with error correction for efficient diffusion inference. *arXiv preprint arXiv:2607.29398*, 2026.

Zexuan Yan, Yue Ma, Chang Zou, Wenteng Chen, Qifeng Chen, and Linfeng Zhang. Eedit: Rethinking the spatial and temporal redundancy for efficient image editing. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17474–17484. IEEE, 2025.

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. *Advances in Neural Information Processing Systems*, 2026.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. *arXiv preprint arXiv:2412.07772*, 2024.

Zichao Yu, Zhen Zou, Guojiang Shao, Chengwei Zhang, Shengze Xu, Jie Huang, Feng Zhao, Xiaodong Cun, and Wenyi Zhang. Ab-cache: Training-free acceleration of diffusion models via adams-bashforth cached feature reuse. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025. doi: 10.1145/3746027.3755650. arXiv:2504.10540.

Zihao Yu, Haoyang Li, Fangcheng Fu, Xupeng Miao, and Bin Cui. Accelerating text-to-image editing via cache-enabled sparse diffusion inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

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. *arXiv preprint arXiv:2501.00375*, 2024.

Hui Zhang, Tingwei Gao, Jie Shao, and Zuxuan Wu. Blockdance: Reuse structurally similar spatio-temporal features to accelerate diffusion transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12891–12900, 2025a. arXiv:2503.15927.

Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *Proc. Int. Conf. Learn. Representations*, 2025b.

Peiyuan Zhang, Yongqi Chen, Runlong Su, Hangliang Ding, Ion Stoica, Zhengzhong Liu, and Hao Zhang. Fast video generation with sliding tile attention. In *International Conference on Machine Learning (ICML)*, 2025c. arXiv:2502.04507.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. arXiv:1801.03924.

Wangbo Zhao, Yizeng Han, Jiasheng Tang, Kai Wang, Yibing Song, Gao Huang, Fan Wang, and Yang You. Dynamic diffusion transformer. In *International Conference on Learning Representations*, 2025a.

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. In *International Conference on Learning Representations*, 2025b.

Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *NeurIPS*, 2023.

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. *arXiv preprint arXiv:2510.04188*, 2025a.

Shikang Zheng, Liang Feng, Xinyu Wang, Qinming Zhou, Peiliang Cai, Chang Zou, Jiacheng Liu, Yuqi Lin, Junjie Chen, Yue Ma, et al. Forecast then calibrate: Feature caching as ode for efficient diffusion transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Zhixin Zheng, Xinyu Wang, Chang Zou, Shaobo Wang, and Linfeng Zhang. Compute only 16 tokens in one timestep: Accelerating diffusion transformers with cluster-driven feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10181–10189, 2025b.

Xin Zhou, Dingkang Liang, Kaijin Chen, Tianrui Feng, Xiwu Chen, Hongkai Lin, Yikang Ding, Feiyang Tan, Hengshuang Zhao, and Xiang Bai. Less is enough: Training-free video diffusion acceleration via runtime-adaptive caching. *arXiv preprint arXiv:2507.02860*, 2025.

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. *NeurIPS*, 2024.

Jie Zhu, Mingyu Ding, Boqiang Duan, Leye Wang, and Jingdong Wang. Unveiling the secret of adaln-zero in diffusion transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026. arXiv:2608.09438.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024.

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. *arXiv preprint arXiv:2602.05449*, 2026a.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking token-wise feature caching: Accelerating diffusion transformers with dual feature caching. *IEEE Transactions on Image Processing*, 2026b.

A Experimental Setup

Evaluation protocol. Following previous works (Chung et al., 2026), for text-to-image generation, we evaluate on 200 prompts from DrawBench (Saharia et al., 2022) and generate images at a resolution of 1024×1024. For text-to-video generation, we use the 944 prompts from VBench (Huang et al., 2024b). HunyuanVideo generates 65-frame videos at a resolution of 480×640, while Wan2.1 generates 81-frame videos at 480×832. In addition, following the evaluation setting of Yin et al. (2024), we evaluate MiniMax using the first 128 prompts from the MovieGen benchmark (Polyak et al., 2024) and generate 124-frame videos at a resolution of 672×384. For each experimental setting, we use the outputs of the original model with full timesteps as the reference and compute PSNR, LPIPS (Zhang et al., 2018), and SSIM between each cached sample and its corresponding reference, followed by averaging over all samples. We further measure the GPU memory occupied by the additional feature tensors stored for caching during inference and report it in GB. For the i-th prompt, we use a fixed seed of base seed $+ i$, and use the same prompt-specific seed across all methods. All experiments run on NVIDIA Blackwell Pro 6000 GPUs.

B Additional method details

B.1 Formal assumptions

Assumption 1 (Precomputable Low-Rank AdaLN Parameters)**.** *The AdaLN parameters* $\{\boldsymbol{\gamma}_\ell(t), \boldsymbol{\beta}_\ell(t), \mathbf{g}_\ell(t)\}$ *depend only on the timestep and conditioning known before sampling, allowing their full trajectories to be precomputed before denoising. We observe that the shift, scale and gate trajectories are all low-rank when stacked across modules and channels (Figure 2; Section 4.3); the LiteCa cache construction specifically uses a rank-K approximation of the gate trajectory, with* $K \ll \min(LC, T)$.

Assumption 2 (Linear Temporal Extrapolation)**.** *We consider extrapolators that form feature-independent scalar linear combinations of cached features. Thus, for any fixed* $\mathbf{w} \in \mathbb{R}^C$ *and any cached quantities* $\mathbf{u}$ *and* $\mathbf{v}$*,*

$$\begin{aligned} P_t\big[\{\mathbf{w} \odot \mathbf{u}(a_j)\}_j\big] &= \mathbf{w} \odot P_t\big[\{\mathbf{u}(a_j)\}_j\big], \\ P_t\big[\{\mathbf{u}(a_j) + \mathbf{v}(a_j)\}_j\big] &= P_t\big[\{\mathbf{u}(a_j)\}_j\big] + P_t\big[\{\mathbf{v}(a_j)\}_j\big]. \end{aligned} \tag{12}$$

B.2 Implementation and calibration details

Online calibration. At a full-computation anchor a, the not-yet-updated cache history produces the residual-caching state

$$\mathbf{h}_{\mathrm{R1}}(a) = \mathbf{h}_{\mathrm{in}}(a) + \hat{\mathbf{r}}_{\mathrm{R}}(a)$$

and the correction components $\{\mathbf{r}_k(a)\}_{k=1}^K$, while the full forward pass produces $\mathbf{h}_{\mathrm{F}}(a)$. The model's native readout is

$$\Phi_t(\mathbf{h}) = \mathrm{Proj}_{\mathrm{out}}(\mathrm{Norm}_{\mathrm{out},t}(\mathbf{h}))\,, \qquad J_a = \left.\frac{\partial \Phi_a(\mathbf{h})}{\partial \mathbf{h}}\right|_{\mathbf{h}=\mathbf{h}_{\mathrm{F}}(a)}. \tag{13}$$

Directly fitting the coefficients through the nonlinear readout would require iterative nonlinear optimisation. We instead linearise Φ_a once at the full state and analytically compute the required

Jacobian–vector products, without evaluating the native readout separately for every perturbed hidden state. The resulting target and correction directions are

$$\mathbf{T}_a = J_a[\mathbf{h}_{\mathrm{F}}(a) - \mathbf{h}_{\mathrm{R1}}(a)], \qquad \mathbf{X}_{k,a} = J_a \mathbf{r}_k(a), \tag{14}$$

and estimate the correction coefficients by

$$\hat{\boldsymbol{\lambda}}_a = \arg\min_{\boldsymbol{\lambda}} \left\| \sum_{k=1}^{K} \lambda_k \mathbf{X}_{k,a} - \mathbf{T}_a \right\|_2^2. \tag{15}$$

All coefficients are solved jointly within the fixed interval $[\lambda_{\min}, \lambda_{\max}]$ to avoid unstable extreme values. The fit uses only the current full-computation anchor before its caches are added to the history, and therefore requires neither future targets nor additional denoising-network evaluations.

Application at skipped steps. With a the most recent calibrated anchor, a skipped step t reconstructs the hidden state and then applies the native readout:

$$\hat{\mathbf{h}}_t = \mathbf{h}_{\mathrm{R1}}(t) + \sum_{k=1}^{K} \hat{\lambda}_{a,k} \mathbf{r}_k(t), \qquad \hat{\mathbf{y}}_t = \Phi_t\left(\hat{\mathbf{h}}_t\right). \tag{16}$$

The Jacobian approximation is used only to estimate the coefficients at full-computation anchors; generation always uses the native readout.

Algorithm 1 LiteCa with Test-Time Correction for one denoising run.

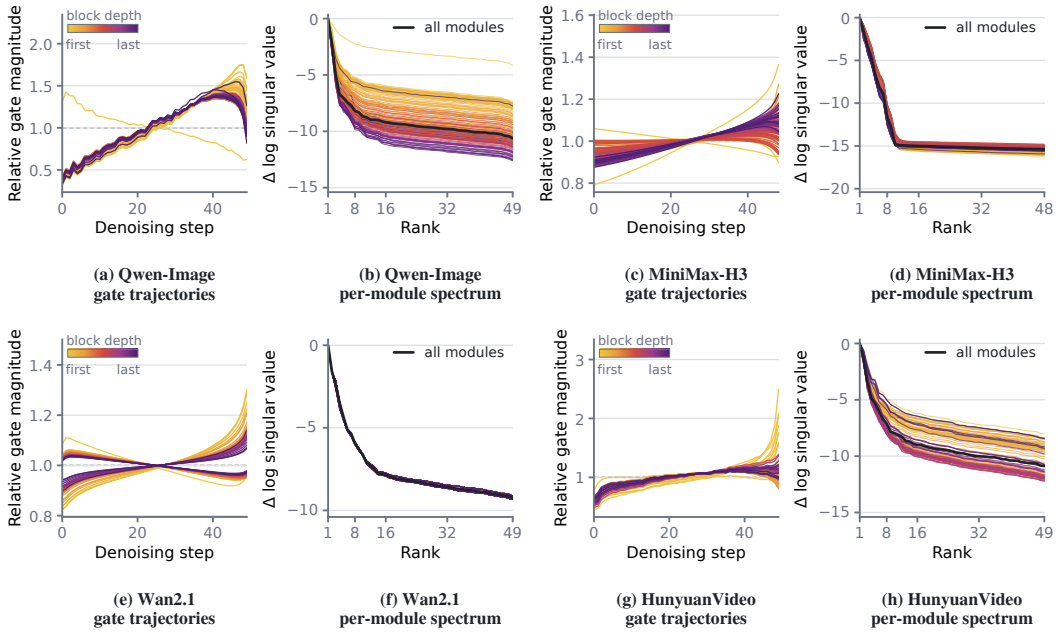
Require: host schedule with anchor set $\mathcal{A}$, rank K, and coefficient bounds $[\lambda_{\min}, \lambda_{\max}]$
precompute $\{\mathbf{g}_\ell(t)\}_{t=1}^{T}$; SVD of the centred gate matrix $\rightarrow \{b_k(t)\}, \{\mathbf{w}_{\ell k}\}$
for $t = 1, \ldots, T$ **do**
  if $t \in \mathcal{A}$ **then**
    construct $\mathbf{h}_{\mathrm{R1}}(t)$ and $\{\mathbf{r}_k(t)\}$ from the current cache history
    compute $\mathbf{h}_{\mathrm{F}}(t)$ and aggregated caches $\mathbf{C}_{\mathrm{res}}(t), \mathbf{C}_{1..K}(t)$
    compute $\mathbf{T}_t$ and $\{\mathbf{X}_{k,t}\}$; update $\hat{\boldsymbol{\lambda}}$
    update the cache history; $\mathbf{h} \leftarrow \mathbf{h}_{\mathrm{F}}(t)$
  else
    $\mathbf{h} \leftarrow \mathbf{h}_{\mathrm{R1}}(t) + \sum_k \hat{\lambda}_k \mathbf{r}_k(t)$
  end if
  $\mathbf{y}_t \leftarrow \Phi_t(\mathbf{h})$ native readout
end for

C Low-Rank Analysis

C.1 Gate trajectories across models

Figure 2(a–c) analyzes the AdaLN modulation on FLUX.1-dev and Qwen-Image. We apply the same analysis to the remaining image and video architectures in Figure 7. The singular spectra decay rapidly across models and modules: the gates change throughout sampling while remaining concentrated in a small temporal subspace, so temporal variation does not by itself imply a high-dimensional trajectory. The same structure has been reported at smaller scale: DiMR (Liu et al., 2024c) finds the scale and shift of a class-conditional DiT moving along two principal components over the sampling steps, and MTEO (Cai et al., 2026) finds the timestep embedding itself nearly two-dimensional. The measurement here extends that to the joint gate trajectory of large text-to-image and text-to-video DiTs.

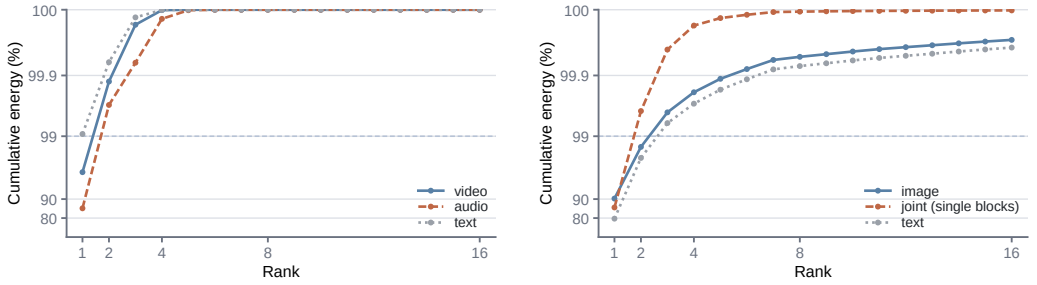
Different residual streams need not have identical spectra. Figure 8 therefore measures each stream independently. The differences are most visible at rank one, while a small number of components captures most of the gate variation in each stream. LiteCa also supports assigning different ranks to different streams when needed.



(a) Qwen-Image gate trajectories (b) Qwen-Image per-module spectrum (c) MiniMax-H3 gate trajectories (d) MiniMax-H3 per-module spectrum

(e) Wan2.1 gate trajectories (f) Wan2.1 per-module spectrum (g) HunyuanVideo gate trajectories (h) HunyuanVideo per-module spectrum

Figure 7: **Gate trajectories and per-module singular spectra across models.** Each pair shows the module-normalized gate magnitude over denoising timesteps and the corresponding centered spectrum. Panels cover Qwen-Image, MiniMax-H3, Wan2.1, and HunyuanVideo; the pooled content-stream spectrum is shown in black.



(a) MiniMax-H3 (b) FLUX.1-dev

Figure 8: **Gate spectrum for individual residual streams.** (a) The text, video, and audio streams of MiniMax-H3. (b) The image and text streams of FLUX.1-dev double blocks and its joint single blocks. The dashed line marks 99% cumulative energy.

C.2 Rank sensitivity with memory-efficient methods

Table 3 reports the lowest-rank LiteCa configuration for CG-Taylor and FreqCa, while Table 4 provides the complete sweep. For CG-Taylor at τ=0.30, R=1 gives the best SSIM and LPIPS; higher ranks yield only a small PSNR increase and reduce the other two metrics. FreqCa changes by less than 0.008 dB across all evaluated ranks. Since cache memory grows linearly with rank, the lowest-rank setting provides the clearest quality–memory trade-off for both methods.

C.3 Closed-loop rank analysis

Figure 6a shows that a small rank is already effective during inference. Here, we extend this analysis to both TaylorSeer and HiCache and evaluate the complete denoising trajectory. This closed-loop setting captures the accumulation of approximation errors across skipped steps and therefore directly measures how closely LiteCa reproduces its module-level host.

Table 4: **Rank sweep with existing memory-efficient methods** on FLUX.1-dev over DrawBench-200. Cache and gate-basis memory are reported separately and combined in Total; Δ is relative to the corresponding host. Latency is the median over prompts for the τ=0.30 and FreqCa blocks, which were timed on a different machine; the τ=0.13 block is the mean of three timed runs after three warm-ups on an idle GPU, timing the diffusion loop only. Seconds are comparable within a block, not across blocks. **Bold** and underline mark the best and second-best visual-retention results within each block. The R=1 rows are reported in Table 3.

Host	LiteCa	Memory				Visual Retention			
		Cache (MiB)	Gate basis (MiB)	Total (GiB)	Δ (%)	Latency↓ (s)	PSNR↑	SSIM↑	LPIPS↓
CG-Taylor $\tau = 0.13$	— (host)	96	—	0.0938	—	4.447	21.5973	0.8199	**0.1771**
	R=1	144	0.891	0.1415	+50.9	4.534	22.0593	0.8207	<u>0.1848</u>
	R=2	192	1.782	0.1892	+101.7	4.581	22.1366	**0.8212**	0.1863
	R=3	240	2.672	0.2370	+152.7	4.646	**22.1588**	<u>0.8212</u>	0.1871
	R=4	288	3.563	0.2847	+203.5	4.717	<u>22.1554</u>	0.8209	0.1875
CG-Taylor $\tau = 0.30$	— (host)	96	—	0.0938	—	4.078	17.5287	0.6460	0.4228
	R=1	144	0.891	0.1415	+50.9	4.036	19.1485	**0.7156**	**0.3329**
	R=2	192	1.782	0.1892	+101.7	4.009	19.2309	<u>0.7087</u>	<u>0.3644</u>
	R=3	240	2.672	0.2370	+152.7	4.154	<u>19.2694</u>	0.7054	0.3741
	R=4	288	3.563	0.2847	+203.5	4.215	**19.2700**	0.7021	0.3821
FreqCa	— (host)	144	—	0.1406	—	3.291	17.1586	0.6800	**0.3469**
	R=1	288	0.891	0.2821	+100.6	3.451	17.2247	0.6837	<u>0.3475</u>
	R=2	360	1.782	0.3533	+151.3	3.596	17.2317	0.6840	0.3477
	R=3	432	2.672	0.4245	+201.9	3.681	**17.2326**	<u>0.6840</u>	0.3477
	R=4	504	3.563	0.4957	+252.6	3.740	<u>17.2323</u>	**0.6841**	0.3477

Protocol. We pair each LiteCa variant with the host from which it is constructed: LiteCa-TS with TaylorSeer and LiteCa-Hi with HiCache. All runs use closed-loop generation with Test-Time Correction disabled, so its online update remains zero. Within each comparison, the seed, resolution, and anchor schedule are fixed, and only the rank is changed. We report the relative L_1 error of the final latent with respect to both the corresponding module-level host and full-step inference.

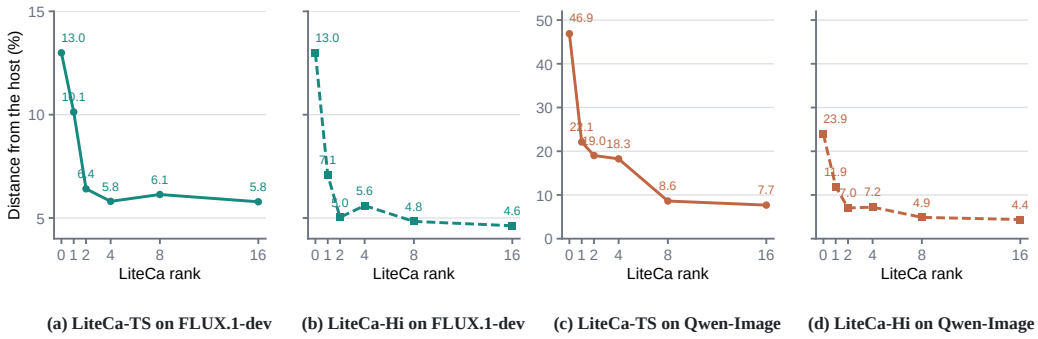
Table 5: **Rank ladder on FLUX.1-dev.** Final-latent relative L_1 error over 11 prompts, to each LiteCa variant's own module-level host and to full-step inference; Test-Time Correction is disabled, so the rank is the only thing that changes down a column. The last row is the host by itself: a ladder converges *towards* module-level caching when the second column of a pair approaches it. Bold marks the best value in each column; the two minima of a pair do not fall on the same rank.

Rank	LiteCa-TS (host: TaylorSeer)		LiteCa-Hi (host: HiCache)	
	vs. host (%)	vs. full step (%)	vs. host (%)	vs. full step (%)
0	12.99	**38.28**	12.98	**35.27**
1	10.13	39.46	7.08	36.53
2	6.41	39.95	5.03	37.20
4	5.81	40.05	5.60	37.24
8	6.14	39.95	4.83	37.23
16	**5.79**	39.86	**4.63**	37.24
host alone	—	39.88	—	37.27

Results. As shown in Tables 5 and 6 and Figure 9, FLUX.1-dev approaches both hosts rapidly: increasing the rank from zero to two reduces the host error from 13.0% to 6.4% for TaylorSeer and to 5.0% for HiCache, with little additional improvement at higher ranks. Qwen-Image exhibits a stronger host-dependent trend. LiteCa-Hi again reaches a low error at rank two (7.0%), whereas

Table 6: **Rank ladder on Qwen-Image.** Final-latent relative L_1 error over 11 prompts, to each LiteCa variant's own module-level host and to full-step inference; Test-Time Correction is disabled, so the rank is the only thing that changes down a column. The last row is the host by itself: a ladder converges *towards* module-level caching when the second column of a pair approaches it. Bold marks the best value in each column; the two minima of a pair do not fall on the same rank.

Rank	LiteCa-TS (host: TaylorSeer)		LiteCa-Hi (host: HiCache)	
	vs. host (%)	vs. full step (%)	vs. host (%)	vs. full step (%)
0	46.87	88.43	23.91	44.80
1	22.12	59.25	11.90	33.16
2	19.02	**52.22**	7.02	**33.05**
4	18.26	64.48	7.22	36.20
8	8.61	59.31	4.87	35.12
16	**7.67**	58.38	**4.38**	34.72
host alone	—	58.51	—	34.82



(a) LiteCa-TS on FLUX.1-dev (b) LiteCa-Hi on FLUX.1-dev (c) LiteCa-TS on Qwen-Image (d) LiteCa-Hi on Qwen-Image

Figure 9: **Closed-loop rank ladders.** Final-latent relative L_1 error to each LiteCa variant's module-level host, with Test-Time Correction disabled. The two panels of a model share a vertical range, but FLUX.1-dev and Qwen-Image do not, and the FLUX.1-dev panels start at 4% rather than zero: that ladder lives entirely between 4.6% and 13.0%, which a common axis anchored at zero would flatten. Compare panels within a model, and read the labelled values across models.

LiteCa-TS continues to improve and reaches a single-digit error at rank eight. These results show that a small number of components captures most of the module-level behavior, while the most suitable rank can depend on both the model and the underlying caching method. This supports using rank as a simple memory–accuracy control rather than fixing one value for every setting. Closeness to the host is not the same as closeness to full-step inference: on FLUX.1-dev the smallest distance to full-step inference is reached at rank zero, before any gate component is added, and raising the rank moves each arm towards the host's own distance rather than below it; on Qwen-Image the two minima again fall on different ranks. The second column of each pair is therefore a convergence check on module-level caching, not a quality ranking.

D Test-Time Correction Analysis

D.1 Fixed-λ sweep

Figure 10 complements the aggregate correction-coefficient ablation in Figure 6b with prompt-level outputs. It retains every fixed coefficient evaluated in the DrawBench-55 sweep, together with the full-step reference, the TaylorSeer host and LiteCa with online Test-Time Correction. These examples are diagnostic views of the sweep rather than the basis for the aggregate conclusion, which is computed over all 55 prompts.

D.2 TEACHER-FORCED PER-STEP ANALYSIS

Figure 6c shows the per-step error of an ordinary generation, in which every step starts from the latent the previous step produced. That measurement cannot localise the gain: an improvement at an early step is carried into the latent of every later step, so a lower curve at step 40 may be entirely inherited from step 10. To ask where the gain is actually produced, we repeat it *teacher-forced* — at every step the latent is replaced by full-step inference's, so each step is evaluated from the same state for all arms and the errors cannot compound.

Results. On FLUX.1-dev the gain is not uniform over the run: Test-Time Correction is most useful in the early and middle intervals. What remains in the late phase is small and not consistent from prompt to prompt, so we do not claim it.

E EFFICIENCY AND PRACTICAL MEMORY

E.1 PEAK MEMORY UNDER PRACTICAL GENERATION SETTINGS

We further compare the GPU memory usage of Wan, HunyuanVideo, and MiniMax under their default model settings. For long-video and high-resolution generation, we measure the total GPU memory usage, including both model inference and cache memory overhead, and compare it against the available GPU memory limit. Figure 12 reports the result. Under memory-intensive settings, the total memory usage of the original caching methods can approach or exceed the hardware limit. In contrast, LiteCa substantially reduces the additional memory overhead introduced by caching, allowing the models to support longer videos or higher-resolution generation under the same GPU memory budget.

F FULL QUANTITATIVE TABLES

F.1 TEXT-TO-IMAGE FULL RANK SWEEP

Tables 1 and 2 show a single LiteCa row per host block, the rank that wins the most of PSNR, SSIM and LPIPS inside that block. The two tables below are the same runs with every measured rank kept, so the whole trade-off is visible rather than its argmax: the cache column grows linearly with the rank while the quality columns saturate, and on several blocks the best rank on one metric is not the best on another. Nothing is recomputed here; these are the rows the main tables were reduced from.

F.2 TEXT-TO-VIDEO FULL RANK SWEEP

Table 8 reports the complete rank sweep for all text-to-video models under the same settings and anchor schedules as Table 2. Cache memory increases linearly with rank, yet LiteCa retains a 91.1–99.2% reduction across all evaluated configurations. HunyuanVideo consistently benefits from additional components, whereas Wan2.1 and MiniMax-H3 favor lower or intermediate ranks, indicating that the optimal rank depends on both the model and the host method. Nevertheless, compact LiteCa configurations achieve competitive or improved generation quality while preserving the inference speedup.

Table 7: **Full rank sweep, text-to-image.** Every LiteCa rank we measured, on the same runs and the same anchor replay as Table 1, which keeps one row per block and leaves Z-Image entirely to this table. Bold is the best value in a host block and underline the second best.

Model	Method	Efficiency				Visual Retention		
		Cache↓ (GiB)	Reduction↑ (%)	Latency↓ (s)	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
FLUX.1-dev	Full-step (100% steps)	—	—	16.305	1.00×	—	—	—
	TaylorSeer (FE=3, N=6, O=2)	6.012	—	4.263	3.83×	16.0339	0.6501	0.3899
	LiteCa-TS (rank 0)	0.070	98.83	3.262	5.00×	16.1928	0.6634	0.3792
	LiteCa-TS (rank 1)	0.141	97.66	3.345	4.88×	16.9191	**0.6797**	**0.3625**
	LiteCa-TS (rank 2)	0.211	96.49	3.391	4.81×	16.8284	0.6733	0.3655
	LiteCa-TS (rank 3)	0.281	95.32	3.455	4.72×	16.8769	0.6714	0.3660
	LiteCa-TS (rank 7)	0.562	90.64	3.807	4.28×	**16.9959**	0.6709	0.3652
	HiCache (FE=3, N=6, O=2)	6.012	—	4.277	3.81×	16.8502	0.6773	**0.3479**
	LiteCa-Hi (rank 0)	0.070	98.83	3.262	5.00×	16.9983	0.6881	**0.3479**
	LiteCa-Hi (rank 1)	0.141	97.66	3.343	4.88×	**17.2545**	**0.6917**	0.3544
	LiteCa-Hi (rank 2)	0.211	96.49	3.393	4.81×	17.0573	0.6847	0.3550
	LiteCa-Hi (rank 3)	0.281	95.32	3.453	4.72×	17.0926	0.6807	0.3578
	LiteCa-Hi (rank 7)	0.562	90.64	3.807	4.28×	17.1030	0.6769	0.3588
Qwen-Image 2512	Full-step (100% steps)	—	—	31.526	1.00×	—	—	—
	TaylorSeer (FE=5, N=6, O=2)	16.926	—	11.566	2.73×	14.655	0.556	0.424
	LiteCa-TS (rank 0)	0.003	99.98	9.611	3.28×	11.732	0.413	0.579
	LiteCa-TS (rank 1)	0.284	98.32	9.843	3.20×	17.827	0.608	0.319
	LiteCa-TS (rank 2)	0.425	97.49	10.039	3.14×	**18.194**	**0.694**	**0.282**
	HiCache (FE=5, N=6, O=2)	16.926	—	12.959	2.43×	18.063	0.650	0.284
	LiteCa-Hi (rank 0)	0.003	99.98	9.632	3.27×	16.207	0.601	0.382
	LiteCa-Hi (rank 1)	0.284	98.32	9.892	3.19×	20.017	**0.753**	**0.249**
	LiteCa-Hi (rank 2)	0.425	97.49	10.205	3.09×	**20.019**	0.728	0.255
Z-Image	Full-step (100% steps)	—	—	32.086	1.00×	—	—	—
	TaylorSeer (FE=5, N=6, O=2)	10.646	—	11.305	2.84×	15.202	0.6063	0.4576
	LiteCa-TS (rank 0)	0.003	99.97	9.089	3.53×	15.578	0.6238	0.4335
	LiteCa-TS (rank 1)	0.358	96.64	9.763	3.29×	15.691	0.6255	**0.4312**
	LiteCa-TS (rank 2)	0.535	94.97	11.466	2.80×	**15.701**	**0.6261**	0.4342
	LiteCa-TS (rank 3)	0.713	93.31	10.350	3.10×	15.648	0.6236	0.4363
	HiCache (FE=5, N=6, O=2)	10.646	—	10.894	2.95×	16.184	0.6228	0.4076
	LiteCa-Hi (rank 0)	0.003	99.97	9.226	3.48×	16.825	0.6477	**0.3943**
	LiteCa-Hi (rank 1)	0.358	96.64	9.753	3.29×	**16.850**	0.6457	0.3971
	LiteCa-Hi (rank 2)	0.535	94.97	11.220	2.86×	16.820	**0.6495**	0.3956
	LiteCa-Hi (rank 3)	0.713	93.31	10.360	3.10×	16.795	0.6490	0.3956

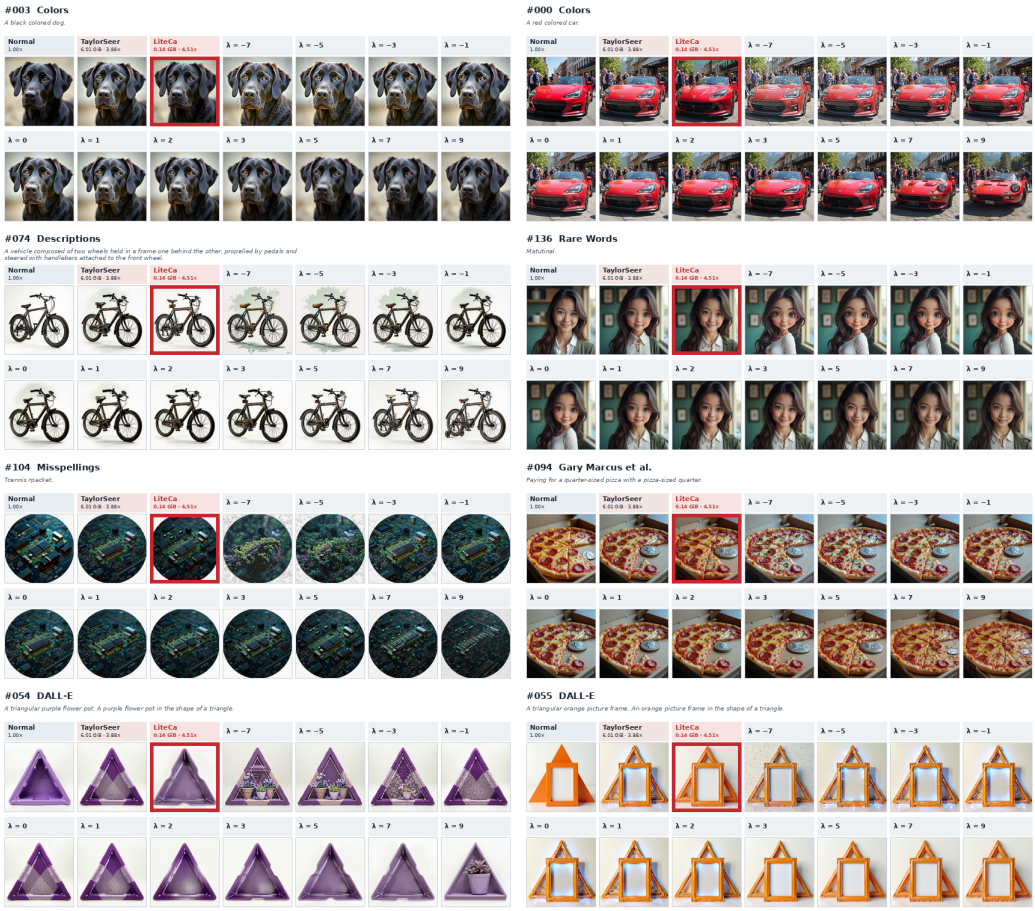
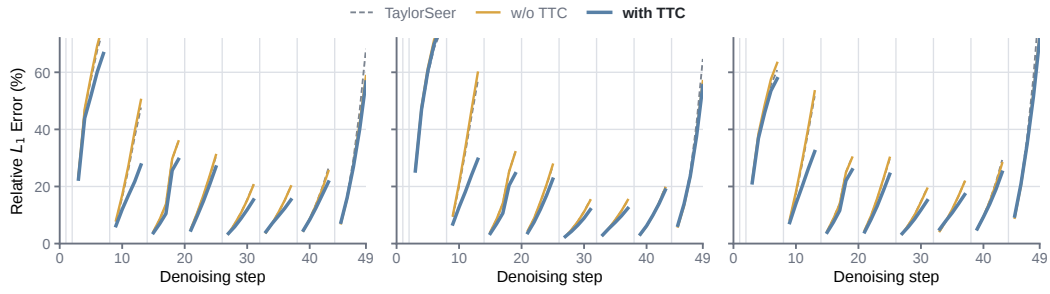


Figure 10: **Fixed correction-coefficient sweep on FLUX.1-dev.** Eight diagnostic prompts are arranged in two columns to keep the full comparison on one page. Each prompt shows the full-step reference, TaylorSeer, LiteCa, and all eleven fixed λ values under the same seed and FE$= 3$, $N = 6$, $O = 2$ anchor schedule. The thick red keyline identifies LiteCa. Efficiency annotations follow Table 1: TaylorSeer uses 6.01 GiB of persistent feature cache and runs at $3.88\times$ the full-step baseline, whereas LiteCa uses 0.14 GiB and runs at $4.51\times$. The labels use the paper's λ convention ($\lambda = 0$ is aggregated-residual caching and $\lambda = 1$ restores the unscaled low-rank correction); the archived runner parameter is $\beta = 1 - \lambda$.



(a) DrawBench prompt 0
early −7.7 pts, late −1.6 pts

(b) DrawBench prompt 25
early −7.9 pts, late −0.5 pts

(c) DrawBench prompt 35
early −5.6 pts, late −1.3 pts

Figure 11: **Teacher-forced per-step error, one panel for each prompt.** We evaluate LiteCa Rank $= 1$ on FLUX.1-dev with FE$= 3$, N=6 and O=2. The light vertical lines show the full-computation anchors, which have no error under teacher forcing and are not included in the comparison. The panels show the first three DrawBench category prompts, not selected examples. Within each anchor interval, the error increases as the step moves farther from the anchor. The gap between the two LiteCa settings also becomes larger, especially in the early and middle intervals. Without Test-Time Correction, LiteCa Rank $= 1$ closely follows module-level TaylorSeer because the low-rank component only restores the module-level gates. The two curves almost overlap, with the gray dashed line under the orange curve. The gap between the two LiteCa curves therefore shows the improvement from Test-Time Correction.

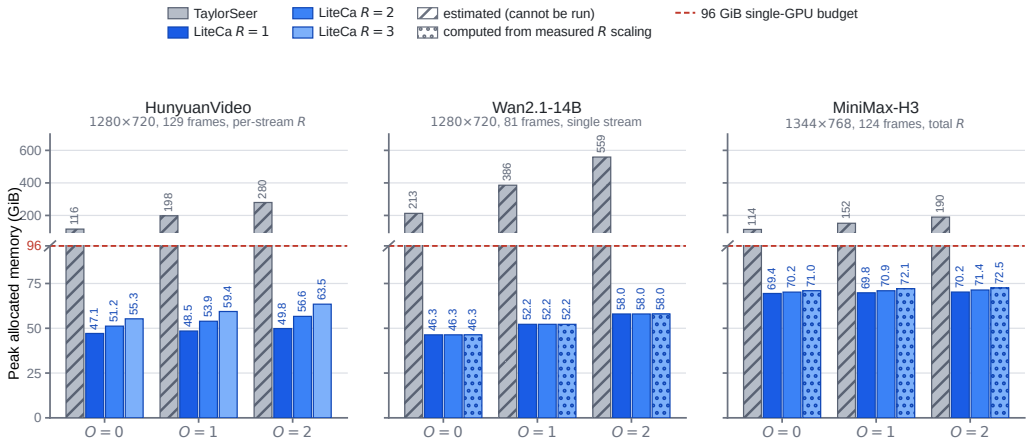


Figure 12: **Peak GPU memory at each model's native generation setting,** as `max_memory_allocated` measured after model load over the diffusion loop, so each bar is the whole footprint of weights, activations and caches. The dashed line is the 96 GiB budget of the single GPU used for these runs, and the panels are cut there: a bar that crosses it does not fit on the card. Module-level TaylorSeer crosses it at every order on all three models and therefore cannot be run natively at all, so its bars (diagonal hatch) are estimates backed by the native out-of-memory traces and reduced-shape scaling probes rather than measurements. The two LiteCa R=3 bars for Wan2.1-14B and MiniMax-H3 (dots) are computed rather than estimated: cache grows exactly linearly in the rank, so R=3 is R=1 + 2(R=2 − R=1), and on HunyuanVideo, where R=3 was run, that identity reproduces the measured peak to within 6 MB. Every other bar is a direct measurement. Rank follows each model's own run: per-stream over the three streams of HunyuanVideo, a single stream on Wan2.1-14B, and all-stream (total) rank on MiniMax-H3.

Table 8: **Full rank sweep, text-to-video.** Every LiteCa rank we measured, on the same runs and the same anchor replay as Table 2. Bold is the best value in a host block and underline the second best.

Model	Method	Efficiency				Visual Retention		
		Cache↓ (GiB)	Reduction↑ (%)	Latency↓ (s)	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
HunyuanVideo	Full-step (100% steps)	—	—	134.023	1.00×	—	—	—
	TaylorSeer (FE=5, N=5, O=1)	28.367	—	42.987	3.12×	20.9992	0.7440	0.2187
	LiteCa-TS (rank 0)	0.233	99.18	38.373	3.49×	21.5661	0.7604	0.2076
	LiteCa-TS (rank 1)	0.467	98.35	38.717	3.46×	22.0697	0.7722	0.1975
	LiteCa-TS (rank 2)	0.700	97.53	39.06	3.43×	<u>22.4016</u>	<u>0.7799</u>	<u>0.1917</u>
	LiteCa-TS (rank 3)	0.934	96.71	39.023	3.43×	**22.4468**	**0.7807**	**0.1907**
	HiCache (FE=5, N=5, O=1)	28.367	—	41.736	3.21×	22.1183	0.7633	0.1969
	LiteCa-Hi (rank 0)	0.233	99.18	38.314	3.50×	22.5507	0.7766	0.1936
	LiteCa-Hi (rank 1)	0.467	98.35	38.889	3.45×	22.8142	0.7832	0.1930
	LiteCa-Hi (rank 2)	0.700	97.53	39.16	3.42×	<u>22.9626</u>	<u>0.7875</u>	**0.1922**
	LiteCa-Hi (rank 3)	0.934	96.71	39.019	3.43×	**22.9715**	**0.7876**	<u>0.1923</u>
Wan2.1	Full-step (100% steps)	—	—	131.113	1.00×	—	—	—
	TaylorSeer (FE=8, N=5, O=1)	33.742	—	43.620	3.01×	<u>21.9269</u>	0.8083	<u>0.1332</u>
	LiteCa-TS (rank 0)	0.750	97.78	42.343	3.10×	21.9233	<u>0.8100</u>	0.1343
	LiteCa-TS (rank 1)	1.500	95.56	42.687	3.07×	**21.9881**	**0.8121**	**0.1319**
	LiteCa-TS (rank 2)	2.249	93.33	42.976	3.05×	21.7610	0.8070	0.1355
	LiteCa-TS (rank 3)	2.999	91.11	43.124	3.04×	21.8496	0.8089	0.1334
	HiCache (FE=8, N=5, O=1)	33.742	—	46.066	2.85×	<u>23.7717</u>	0.8381	**0.1057**
	LiteCa-Hi (rank 0)	0.750	97.78	42.369	3.09×	**23.8067**	**0.8437**	<u>0.1076</u>
	LiteCa-Hi (rank 1)	1.500	95.56	42.742	3.07×	23.7575	<u>0.8416</u>	0.1081
	LiteCa-Hi (rank 2)	2.249	93.33	43.040	3.05×	23.5721	0.8412	0.1079
	LiteCa-Hi (rank 3)	2.999	91.11	43.279	3.03×	23.5689	0.8410	0.1077
MiniMax-H3	Full-step (100% steps)	—	—	123.70	1.00×	—	—	—
	TaylorSeer (FE=8, N=5, O=1)	19.560	—	45.05	2.75×	19.522	0.6428	0.2453
	LiteCa-TS (rank 0)	0.196	99.00	40.48	3.06×	19.819	0.6485	0.2439
	LiteCa-TS (rank 1)	0.391	98.00	41.23	3.00×	20.040	0.6544	0.2396
	LiteCa-TS (rank 2)	0.587	97.00	41.56	2.98×	<u>20.185</u>	<u>0.6583</u>	<u>0.2376</u>
	LiteCa-TS (rank 3)	0.782	96.00	41.91	2.95×	**20.230**	**0.6586**	**0.2371**
	HiCache (FE=8, N=5, O=1)	19.560	—	43.32	2.86×	20.334	0.6635	0.2245
	LiteCa-Hi (rank 0)	0.196	99.00	40.73	3.04×	20.737	0.6728	<u>0.2228</u>
	LiteCa-Hi (rank 1)	0.391	98.00	40.97	3.02×	<u>20.872</u>	**0.6758**	**0.2227**
	LiteCa-Hi (rank 2)	0.587	97.00	41.36	2.99×	20.826	0.6738	0.2238
	LiteCa-Hi (rank 3)	0.782	96.00	42.39	2.92×	**20.893**	<u>0.6754</u>	<u>0.2228</u>