TERMINAL-SENSITIVITY-AWARE FEATURE CACHING FOR EFFICIENT DIFFUSION INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Feature caching accelerates diffusion inference by reusing intermediate features across denoising steps, but existing adaptive methods typically make reuse decisions from estimated local denoiser-output differences. The same local difference can produce different terminal errors because its effect depends on the scheduler step size and the sensitivity of the remaining denoising trajectory. To account for these factors, we introduce *Terminal Sensitivity Caching* (TSenCache), a training-free method that estimates the terminal impact of each reuse decision. TSenCache guides reuse decisions by combining a sample-adaptive local-error estimate with the target scheduler step size and a calibrated sensitivity of the remaining trajectory. Keeping the scheduler step size separate also allows the calibrated sensitivity profiles to transfer across sampling settings through noise-level interpolation. Experiments on FLUX.1-dev and Wan2.1 demonstrate that TSenCache preserves higher reconstruction fidelity than the evaluated caching methods at comparable inference speed.

1 INTRODUCTION

Diffusion and flow-based generative models have enabled high-quality image and video synthesis (Ho et al., 2020; Song et al., 2020b; Lipman et al., 2022; Liu et al., 2022). Recently, transformer-based denoisers have further improved generation quality by scaling model capacity (Peebles & Xie, 2023; Esser et al., 2024). Despite these advances, inference remains computationally expensive because generating a single sample requires repeatedly evaluating a large denoiser over multiple timesteps (Ho et al., 2020; Rombach et al., 2022). This cost grows further with the spatial resolution, video duration, and model size required by modern generative systems (Podell et al., 2024; Kong et al., 2024). Feature caching approaches reduce this computational cost without additional training by reusing previously computed outputs or intermediate features across nearby timesteps (Wimbauer et al., 2024; Ma et al., 2024; Chen et al., 2024; Selvaraju et al., 2024).

Early caching approaches specify the reuse timesteps or intervals as fixed hyperparameters (Ma et al., 2024; Chen et al., 2024; Selvaraju et al., 2024). More recent online adaptive methods move beyond these fixed schedules by making sample-specific reuse decisions at each timestep (Liu et al., 2025a; Bu et al., 2026; Chung et al., 2026; Haghighi & Alahi, 2026). These methods base each decision on the local model-output difference at the current timestep, defined as the discrepancy between the outputs produced by cache reuse and by full computation. The cache is reused when this output difference stays below a threshold, whereas full computation is triggered when the difference exceeds it. Since its exact value is available only after a full computation, existing methods have proposed to approximate the output difference using lightweight runtime signals. For example, TeaCache (Liu et al., 2025a) uses rescaled differences between timestep-modulated inputs as a proxy for output differences, assuming that the two are strongly correlated. SenCache (Haghighi & Alahi, 2026) instead predicts the output difference through a first-order sensitivity approximation that accounts for both latent drift and the timestep gap.

However, when reusing the cache, a small local model-output difference at a given timestep does not necessarily imply a small effect on the final generated image or video (Gao et al., 2025). According to the flow matching formulation (Lipman et al., 2022; Liu et al., 2022), the local velocity error (Δv) is scaled by the scheduler step size into a next-latent difference (Δx). This latent difference then propagates through the remaining denoising steps and affects the final-output difference, or terminal

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.

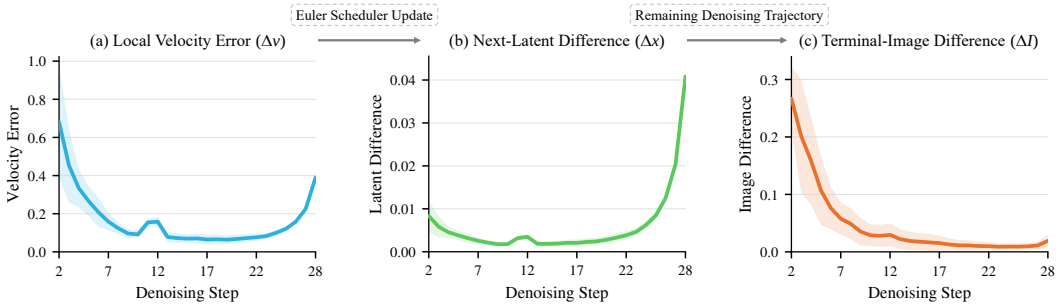

Figure 1: **Cache-induced error propagation through FLUX denoising.** We isolate the effect of a single reuse decision by reusing the preceding step's feature and performing full computation at all other steps. The panels show the resulting magnitudes of (a) the velocity difference Δv, (b) the next-latent difference Δx, and (c) the terminal-image difference ΔI. Although the local velocity error and scheduler-scaled next-latent difference increase toward the end of denoising, the terminal-image difference is concentrated in the early steps. Solid curves and shaded regions denote the mean and standard deviation over 16 prompts, respectively.

impact (ΔI). We empirically trace this error propagation in FLUX (Labs, 2024) by measuring Δv, Δx, and ΔI for a single cache reuse at each tested timestep, with full computation at all other steps. Figure 1(a) shows large local velocity errors at both early and late steps, while the next-latent difference after scheduler scaling is largest near the end (Figure 1(b)). Yet reuse at these late steps produces much smaller terminal-image differences than early reuse (Figure 1(c)). These contrasting patterns suggest that existing timestep-agnostic local-error thresholds can permit reuse too readily in sensitive early steps while triggering full computation in less sensitive late steps. Therefore, we conclude that reuse decisions should be guided by the resulting difference in the terminal decoded output rather than by local model-output difference.

Motivated by this local–terminal mismatch, we propose *Terminal Sensitivity Caching* (TSenCache), a training-free online method that makes sample-adaptive cache-reuse decisions based on their estimated effect on the terminal output. For a reuse decision at any timestep, we define terminal impact as the difference between the terminal decoded outputs produced by cache reuse and those produced by full computation at that step. As measuring this impact exactly would require completing both denoising trajectories, we construct a lightweight estimate from quantities available during inference without a full denoiser evaluation at the current step. Our first-order formulation decomposes terminal impact into the scheduler step size, the timestep-dependent sensitivity of the remaining denoising trajectory, and the local model-output error. We calibrate the trajectory sensitivity offline as a function of noise level and combine it with the scheduler step size during inference to form a timestep-dependent weight on the local-error estimate. Since this weighting is independent of how the local error is estimated, TSenCache supports different lightweight local estimators (Liu et al., 2025a; Bu et al., 2026; Ma et al., 2026; Haghighi & Alahi, 2026) and requires only scalar multiplication to weight their estimates. Keeping the scheduler step size separate also allows TSenCache to change the noise schedule or number of sampling steps without recalibration, reusing the calibrated sensitivity profile through noise-level interpolation.

In summary, our contributions are threefold. First, we empirically analyze how a cache-induced local velocity error is translated into terminal error through the scheduler update and the remaining denoising trajectory. Second, we propose TSenCache, which maps an online estimate of local model-output error to terminal impact by combining the target scheduler's step size with a calibrated sensitivity of the remaining denoising trajectory. Finally, we demonstrate that TSenCache achieves higher reconstruction fidelity than the evaluated caching methods at comparable inference speed on both FLUX.1-dev image generation (Labs, 2024) and Wan2.1 video generation (Wan et al., 2025).

2 Related Work

Efficient diffusion inference. Efficient diffusion inference has been approached by reducing either the number of denoising evaluations or the cost of each evaluation. Improved samplers shorten

the sampling trajectory through deterministic or higher-order updates (Song et al., 2020a; Lu et al., 2022; Zhao et al., 2023), whereas distillation trains a model specifically for few-step generation (Salimans & Ho, 2022; Song et al., 2023; Luo et al., 2023; Yin et al., 2024b;a). Quantization instead reduces the arithmetic and memory cost of individual denoiser evaluations (He et al., 2023; Li et al., 2023; Shang et al., 2023; So et al., 2023). Feature caching reduces inference cost without retraining by reusing intermediate features across nearby timesteps.

Feature reuse and cache scheduling. Block Caching and DeepCache reuse intermediate U-Net features (Wimbauer et al., 2024; Ma et al., 2024), while Δ-DiT and FORA extend reuse to diffusion transformers through block residuals or scheduled attention and MLP features (Chen et al., 2024; Selvaraju et al., 2024). ToCa and related methods vary reuse across tokens, features, or layers (Zou et al., 2024; 2026; Zheng et al., 2025; Liu et al., 2025c), whereas TaylorSeer forecasts features from previous full computations (Liu et al., 2025b). Adaptive methods use lightweight proxies for local output differences to decide when to reuse cached features. These include modulated-input differences in TeaCache, calibrated residual-magnitude ratios in MagCache, and runtime transformation-rate estimates in EasyCache (Liu et al., 2025a; Ma et al., 2026; Zhou et al., 2025). DiCache and SeaCache use an online probe or filtered feature changes (Bu et al., 2026; Chung et al., 2026), while SenCache estimates local output differences from latent and timestep sensitivities (Haghighi & Alahi, 2026). Despite these different proxies, their criteria remain local and do not estimate the effect of reuse on the terminal decoded output. Offline scheduling instead optimizes a static path under a fixed computation budget (Gao et al., 2025), whereas TSenCache makes online, sample-adaptive decisions by mapping local error to the terminal-output scale through the scheduler update and remaining-trajectory sensitivity.

3 METHOD

3.1 PRELIMINARIES

Flow matching. Flow matching defines a continuous-time probability path $\{p_t\}_{t\in[0,1]}$ from a simple prior p_0, typically Gaussian, to a target latent distribution p_1 (Lipman et al., 2022; Liu et al., 2022). Given a conditioning prompt c, a velocity field $f_\theta(x_t, t, c)$ transports samples along this path according to the ODE

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = f_\theta(x_t, t, c), \qquad x_0 \sim p_0. \tag{1}$$

Starting from x_0, inference numerically integrates Equation 1 over a discrete schedule $\{t_k\}_{k=0}^{K}$. Under Euler integration, the latent x_k at time t_k is updated as

$$x_{k+1} = x_k + (t_{k+1} - t_k) f_\theta(x_k, t_k, c). \tag{2}$$

After integration, a decoder $\mathcal{D}$ maps the terminal latent x_K to an image or video $I = \mathcal{D}(x_K)$. As repeated evaluations of f_θ dominate the sampling cost, feature caching reduces inference time by reusing previously computed Transformer features while the scheduler continues to update the latent at every step.

Local-error estimation. When deciding whether to reuse the feature at step k, let $r < k$ denote the most recent full-computation step, with latent x_r and sampling time t_r. A full denoiser evaluation at step k produces $v_k = f_\theta(x_k, t_k, c)$, while reusing the feature cached at step r produces $\widetilde{v}_k$. The local output difference is $e_k = \|v_k - \widetilde{v}_k\|_2$, and evaluating it exactly would require the full computation that caching seeks to avoid. To estimate this difference without a full evaluation, SenCache (Haghighi & Alahi, 2026) uses a first-order Taylor approximation of the denoiser output to weight the latent and timestep differences by their respective sensitivities:

$$\widehat{e}_k = \alpha_x(k)\|x_k - x_r\|_2 + \alpha_t(k)|t_k - t_r|. \tag{3}$$

Here, $\alpha_x(k)$ and $\alpha_t(k)$ estimate how strongly the denoiser output responds to perturbations in the latent and timestep inputs, respectively. SenCache estimates these profiles offline on a small calibration set by perturbing one input at a time and dividing the magnitude of the resulting output difference by the input perturbation size. Averaging these ratios at each timestep gives $\alpha_x(k)$ and $\alpha_t(k)$, which are stored for use during inference. Combining these profiles with the available latent and timestep differences yields $\widehat{e}_k$ without a full denoiser evaluation. The resulting score estimates the local output difference caused by cache reuse.

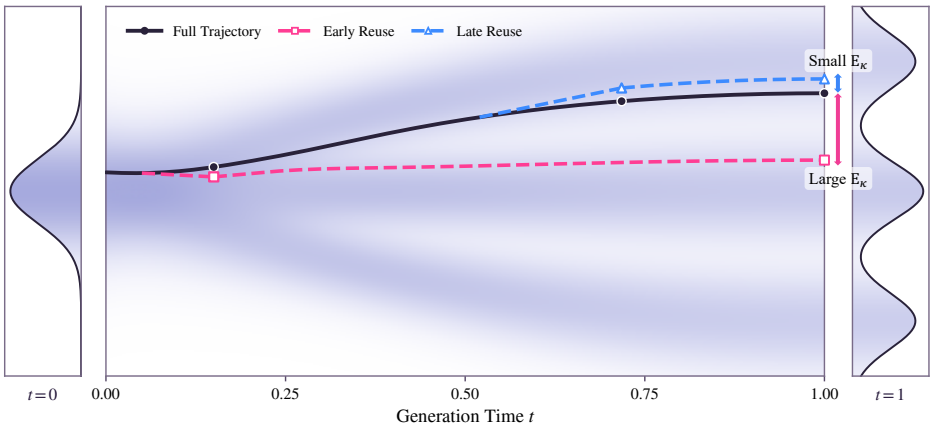

Figure 2: **Timestep-dependent terminal effect of cache reuse in a 1D generative flow.** The black curve shows the full-computation trajectory, while the dashed curves show trajectories obtained by reusing the cache once at an early or late timestep and then resuming full computation. Despite comparable local velocity errors, early reuse produces a larger terminal deviation than late reuse because the remaining-trajectory sensitivity varies with timestep.

3.2 Terminal-Impact Formulation

Figure 2 illustrates why local model-output difference is insufficient as the objective for a reuse decision. Reuse at different timesteps can produce similar local model-output differences, yet those differences may lead to substantially different deviations in the terminal output. To formalize this distinction, consider the reuse decision at step k, given the current latent x_k and the feature cached at step r. Starting from the same x_k, the full-computation branch (F) uses v_k, whereas the cache-reuse branch (C) uses $\widetilde{v}_k$. To isolate the effect of the current decision, we compare their terminal outputs assuming that both branches use full computation at all subsequent steps. Denoting their terminal outputs by I_k^F and I_k^C, we define the terminal impact of this reuse decision as

$$E_k = \|I_k^F - I_k^C\|_2. \tag{4}$$

Although E_k directly measures the quantity that a reuse decision should control, computing it online would require rolling out both branches to the terminal output. To obtain a tractable estimate, we first use Equation 2 to express the perturbation that the local velocity error induces in the next latent as

$$\Delta x_{k+1} := x_{k+1}^C - x_{k+1}^F = (t_{k+1} - t_k)(\widetilde{v}_k - v_k), \qquad \|\Delta x_{k+1}\|_2 = |t_{k+1} - t_k| e_k. \tag{5}$$

The scheduler step size $|t_{k+1} - t_k|$ determines how strongly the local velocity error perturbs the state passed to the remaining trajectory.

The remaining trajectory then propagates this perturbation to the terminal output. Let $F_{k+1}(x; c)$ map a latent x at step $k+1$ to the terminal output by completing the remaining denoising steps without cache reuse and then decoding, so that $I_k^F = F_{k+1}(x_{k+1}^F; c)$ and $I_k^C = F_{k+1}(x_{k+1}^C; c)$. Assuming that the cache-induced perturbation is sufficiently small for a local linearization, we expand F_{k+1} around x_{k+1}^F to obtain

$$J_F := \left.\frac{\partial F_{k+1}(x; c)}{\partial x}\right|_{x = x_{k+1}^F}, \qquad I_k^C - I_k^F \approx J_F \Delta x_{k+1}. \tag{6}$$

Taking the norm of the first-order approximation and applying Equation 5 gives

$$E_k \approx \|J_F \Delta x_{k+1}\|_2 \le \|J_F\| \|\Delta x_{k+1}\|_2 = \|J_F\| |t_{k+1} - t_k| e_k. \tag{7}$$

Here, $\|J_F\|$ denotes the operator norm induced by the Euclidean vector norm. Equation 7 shows that terminal impact depends jointly on the local velocity error e_k, the scheduler step size, and the remaining-trajectory sensitivity $\|J_F\|$. Consequently, the same local error can produce different terminal effects when either the scheduler scaling or the downstream sensitivity varies across timesteps.

3.3 TSENCACHE

Terminal-impact score. The first-order bound in Equation 7 provides a starting point for our score, but its two unknown terms cannot be evaluated efficiently during inference. Computing e_k would require the full denoiser evaluation that caching seeks to avoid, whereas evaluating $\|J_F\|$ would require differentiation through the remaining trajectory. The local-error term e_k can be approximated using different lightweight estimators (Liu et al., 2025a; Bu et al., 2026; Haghighi & Alahi, 2026). In our main implementation, we use SenCache's calibrated local output-difference estimate $\widehat{e}_k$ from Equation 3. To account for the remaining trajectory, we calibrate a timestep-wise sensitivity profile $\alpha_F(k+1)$ that estimates how strongly cache-induced next-latent differences affect the terminal output. Combining this profile with the local-error estimate and the known scheduler step size gives

$$S_k = \underbrace{\alpha_F(k+1)}_{\substack{\text{remaining-trajectory}\\ \text{sensitivity}}} \underbrace{|t_{k+1} - t_k|}_{\substack{\text{scheduler}\\ \text{step size}}} \underbrace{\widehat{e}_k}_{\substack{\text{local output-difference}\\ \text{surrogate}}} . \tag{8}$$

The scheduler step size and α_F weight the local-error estimate $\widehat{e}_k$ across timesteps by accounting for the induced next-latent difference and its effect on the terminal output. By keeping the scheduler step size separate from the calibrated sensitivities, our TSenCache can reuse the profiles without recalibration across noise schedules and sampling-step counts, using the target step sizes and simple noise-level interpolation.

Terminal-sensitivity calibration. TSenCache calibrates α_F on the same calibration set used to estimate the local-sensitivity profiles. At each calibration timestep k on a full-computation trajectory, we reuse the feature from the preceding step $k-1$, producing $\widetilde{v}_k$ and the perturbed next latent x^C_{k+1}. We then complete the remaining denoising steps without further cache reuse to obtain the terminal output I^C_k. For N calibration prompts, the timestep-wise profile is

$$\alpha_F(k+1) = \frac{1}{N} \sum_{n=1}^{N} \frac{\|I^F_{k,n} - I^C_{k,n}\|_2}{\|x^F_{k+1,n} - x^C_{k+1,n}\|_2}. \tag{9}$$

This finite difference measures the sensitivity of the remaining trajectory along perturbation directions induced by actual feature reuse. We average these measurements across calibration prompts to obtain the directional sensitivity profile used in our terminal-impact score.

Online decision. TSenCache performs full computation at the first step and stores its latent x_r, timestep t_r, and resulting feature. At every subsequent step k, TSenCache obtains $\widehat{e}_k$ from the latent and timestep differences relative to (x_r, t_r), and combines it with the current scheduler step size and $\alpha_F(k+1)$ to compute S_k. When $S_k \le \varepsilon$, TSenCache reuses the stored feature and retains (x_r, t_r); otherwise, it performs full computation and updates the reference and cached feature at the current step. Equivalently, this rule applies a timestep-dependent threshold $\varepsilon / [\alpha_F(k+1)|t_{k+1} - t_k|]$ to the local estimate $\widehat{e}_k$. Thus, the tolerance ε governs reuse according to the estimated terminal impact of each decision. Score evaluation at inference requires only tensor differences, norm evaluations, profile lookups, and scalar operations, with no auxiliary prediction network or additional training.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Implementation details. We evaluate TSenCache on FLUX.1-dev (Labs, 2024) for text-to-image generation and Wan2.1-T2V-1.3B (Wan et al., 2025) for text-to-video generation. FLUX generates 1024×1024 images in 28 sampling steps, whereas Wan2.1 generates 81-frame videos at 832×480 resolution in 50 steps. All experiments use NVIDIA RTX A6000 GPUs. For the main experiments, we calibrate FLUX on eight COCO captions (Lin et al., 2014) and Wan2.1 on the descriptions of eight MixKit videos (Lin et al., 2024) under their default sampling configurations. The calibration and evaluation prompts are disjoint, and the resulting sensitivity profiles are reused throughout evaluation. Following SenCache (Haghighi & Alahi, 2026), TSenCache caches and reuses the feature produced by the full Transformer stack. For the main comparisons, we select $\varepsilon = 0.20$ for FLUX and $\varepsilon = 0.068$ for Wan2.1, with score normalization described in Appendix A.

Table 1: **Efficiency and reconstruction fidelity on FLUX.1-dev.** At a comparable full-computation budget, our TSenCache achieves the best fidelity across all three metrics.

Method	NFE ↓	TFLOPs ↓	Latency (s) ↓	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓
Full computation	28	1681.35	29	1.00×	–	–	–
TeaCache	10	610.29	11	2.67×	16.894	0.6747	0.3957
DiCache	10	629.08	11	2.59×	<u>22.800</u>	0.8051	0.2190
MagCache	10	610.28	11	2.67×	22.005	0.8006	0.2127
SeaCache	10	610.29	11	2.67×	22.551	<u>0.8159</u>	<u>0.1954</u>
SenCache	10	610.28	11	2.67×	22.108	0.8028	0.2129
TaylorSeer	11	669.95	13	2.17×	19.735	0.7738	0.2301
TSenCache	10	610.28	11	2.67×	**24.671**	**0.8363**	**0.1695**

Table 2: **Efficiency and reconstruction fidelity on Wan2.1-T2V-1.3B.** Our TSenCache provides the best fidelity among caching methods operating at approximately twice the speed of full computation.

Method	NFE ↓	TFLOPs ↓	Latency (s) ↓	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓
Full computation	50	8496.21	371	1.00×	–	–	–
TeaCache	25	4391.14	193	1.92×	23.020	0.8199	0.1187
DiCache	24	4369.24	193	1.92×	26.844	0.8972	0.0580
MagCache	25	4391.14	193	1.92×	28.189	<u>0.9138</u>	<u>0.0504</u>
SeaCache	24	4226.96	188	1.98×	<u>28.282</u>	0.9077	0.0546
SenCache	25	4391.14	193	1.92×	26.955	0.8977	0.0620
TaylorSeer	25	4391.14	205	1.81×	14.994	0.5500	0.3908
TSenCache	24	4226.94	188	1.98×	**28.933**	**0.9181**	**0.0495**

Baselines. We compare TSenCache with full computation and representative training-free caching methods, including TeaCache (Liu et al., 2025a), DiCache (Bu et al., 2026), MagCache (Ma et al., 2026), SeaCache (Chung et al., 2026), SenCache (Haghighi & Alahi, 2026), and TaylorSeer (Liu et al., 2025b). We use the official implementations for all baselines except SenCache on FLUX, which we implement based on its official video-model code. Reconstruction fidelity is evaluated at comparable full-computation budgets.

Evaluation protocol. We evaluate FLUX on all 200 DrawBench prompts (Saharia et al., 2022) and Wan2.1 on the standard VBench prompts (Huang et al., 2024), using identical prompts, random seeds, and generation settings across methods. We measure reconstruction fidelity against the matching full-computation output using PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018), averaging the video metrics over all frames and prompts. For efficiency, NFE counts the denoising steps that perform full model computation. Since it omits method-specific overhead such as the online probe used by DiCache, we also report FLOPs computed with CalFLOPs (Ye, 2023) and end-to-end generation latency.

4.2 Quantitative Comparison

Tables 1 and 2 compare reconstruction fidelity at comparable inference speeds. On FLUX.1-dev, TSenCache achieves the highest fidelity across all three metrics with a $2.67\times$ speedup over full computation. The improvements in both pixel-level and perceptual metrics indicate closer agreement with the full-computation output. TSenCache shares SenCache's local-error estimator but additionally accounts for the scheduler step size and remaining-trajectory sensitivity. Its gains over SenCache therefore support weighting local error by its estimated terminal effect when deciding which steps require full computation.

On Wan2.1, TSenCache likewise achieves the highest reconstruction fidelity among methods operating at similar speeds, with a $1.98\times$ speedup over full computation. The frame-averaged results show closer agreement with the reference videos in both image structure and perceptual appearance. These gains are obtained with the same score formulation used for FLUX, with sensitivity profiles calibrated for Wan2.1. Together, these results support basing cache-reuse decisions on the estimated effect on the final image or video, rather than on local error alone.

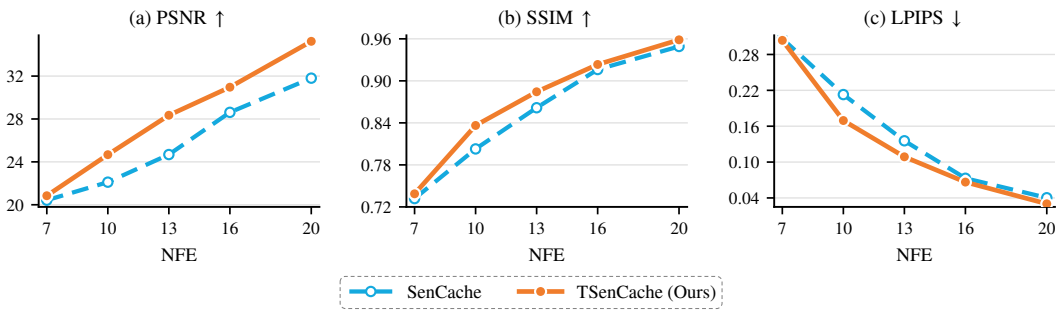

Figure 3: **Speed–quality trade-off on FLUX.1-dev.** We compare SenCache (Haghighi & Alahi, 2026) and our TSenCache over 200 DrawBench prompts at five budgets from 7 to 20 NFE. TSenCache provides better PSNR, SSIM, and LPIPS from 10 NFE onward.

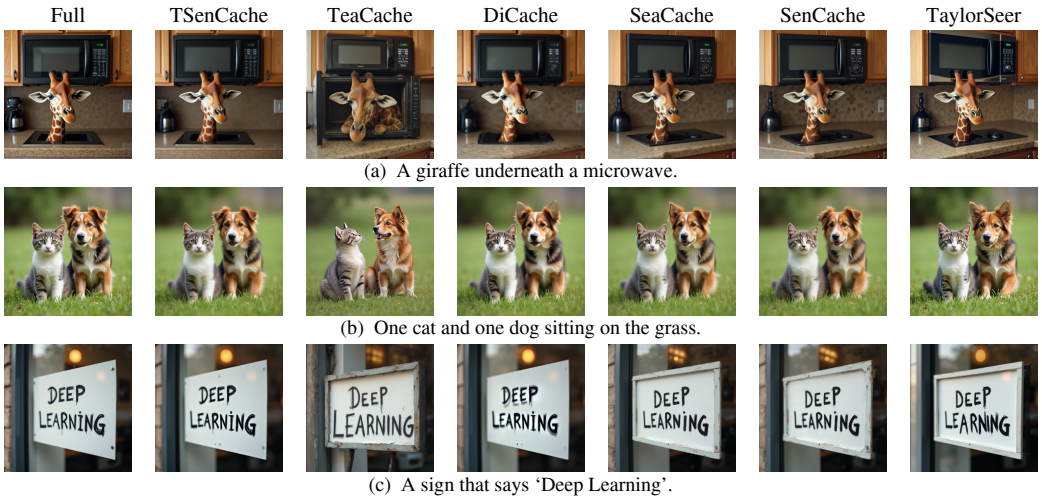

Figure 4: **Qualitative comparison on FLUX.1-dev.** For each DrawBench prompt, we compare full computation with caching methods for a target budget of 10 NFE. The examples compare preservation of scene layout (a), subject appearance (b), and lettering (c).

A comparison at a single operating point does not show whether the advantage persists as the reuse threshold changes. To examine this broader range, we compare SenCache and TSenCache at budgets of 7, 10, 13, 16, and 20 NFE in Figure 3. The two methods remain close under the most aggressive 7-NFE setting, whereas TSenCache consistently provides better PSNR, SSIM, and LPIPS from 10 to 20 NFE. These curves show that the advantage observed in the main comparison extends to higher computation budgets.

4.3 Qualitative Comparison

The reconstruction metrics summarize overall fidelity but do not reveal which parts of the generated content are changed by cache reuse. To examine these differences, Figure 4 compares representative FLUX.1-dev outputs for a target budget of 10 NFE. TSenCache preserves the scene layout in Figure 4(a) and the dog's folded ear in Figure 4(b), whereas several baselines show an upright ear. In Figure 4(c), it retains the reference lettering and sign structure, while several baselines alter the letter shapes or introduce a frame.

For Wan2.1, Figure 5 compares temporally separated frames from SenCache and TSenCache at comparable budgets against the full-computation reference. TSenCache more closely follows the reference poses in the first example and better preserves the bird's appearance and motion in the second. These examples show closer agreement with the reference at multiple points in each video.

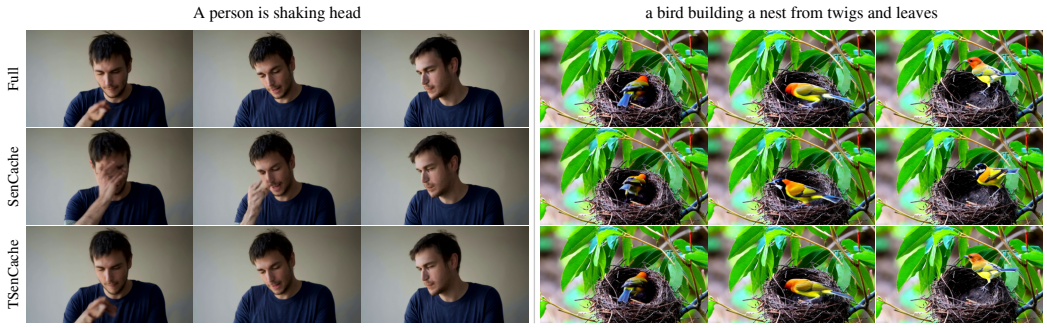

Figure 5: **Qualitative comparison on Wan2.1-T2V-1.3B.** We compare SenCache (Haghighi & Alahi, 2026) and our TSenCache at comparable NFE using temporally separated frames. TSenCache better preserves the poses and subject appearance of the full-computation reference.

Table 3: **Reconstruction fidelity with different score components on FLUX.1-dev.** Results are averaged over 200 DrawBench prompts at a target budget of 10 NFE. Our final score composition leads to the best performance.

Reuse score	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
$\widehat{e}_k$	10	21.921	0.8015	0.2153
$\lvert t_{k+1} - t_k \rvert \widehat{e}_k$	10	15.893	0.6372	0.4572
$\alpha_F(k+1)\widehat{e}_k$	10	23.044	0.7348	0.3338
$\alpha_F(k+1)\lvert t_{k+1} - t_k \rvert \widehat{e}_k$ (Ours)	10	**24.671**	**0.8363**	**0.1695**

4.4 Additional Analysis

Alignment with terminal impact. The central premise of TSenCache is that a reuse score should follow terminal impact more closely than a local-error estimate. Figure 6 compares the SenCache local score and our scores using measured or estimated local error against measured terminal impact over held-out reuse candidates. Since the reuse tolerance can be adjusted to each score's scale, we use Pearson and Spearman correlations across prompts and timesteps to assess how well each score tracks terminal impact. The local score in Figure 6(a) shows weaker alignment with terminal impact than our score in Figure 6(b), which combines measured local error with the scheduler and terminal-sensitivity factors. Using the same local-error estimate, our practical score in Figure 6(c) achieves higher Pearson and Spearman correlations than the SenCache score in Figure 6(a), showing that timestep-dependent weighting improves alignment with terminal impact. Appendix D shows that this advantage persists when reusing features from up to four steps earlier, without recalibration.

Effect of score components. Table 3 examines how the two weighting factors affect reconstruction fidelity relative to the local score. Weighting by the scheduler step size shifts full computation toward later steps, as the default schedule has larger step sizes near the end. This leaves fewer evaluations in the sensitive early region and lowers fidelity across all three metrics. Weighting by the remaining-trajectory sensitivity instead concentrates full computation early and leaves long reuse intervals afterward. PSNR improves over the local score, but SSIM and LPIPS worsen, suggesting that sparse later refreshes can offset the benefit of early-step protection. The complete TSenCache configuration retains dense early computation while refreshing features later and achieves the highest fidelity across all three metrics.

Calibration transfer across sampling configurations. Keeping the scheduler step size separate from the calibrated sensitivity profiles allows TSenCache to use the target scheduler's step sizes directly and interpolate the profiles at the corresponding noise levels. Table 4 tests this property with a 50-step default schedule and a 28-step schedule with uniformly spaced noise levels over the same noise range. We compare target-specific calibration with the default 28-step profiles transferred through noise-level interpolation, using the target scheduler's step sizes in both cases. The transferred profiles closely match the reconstruction fidelity of target-specific calibration at comparable NFE for both changes. These results support reusing a single calibration across the tested step counts and noise schedules.

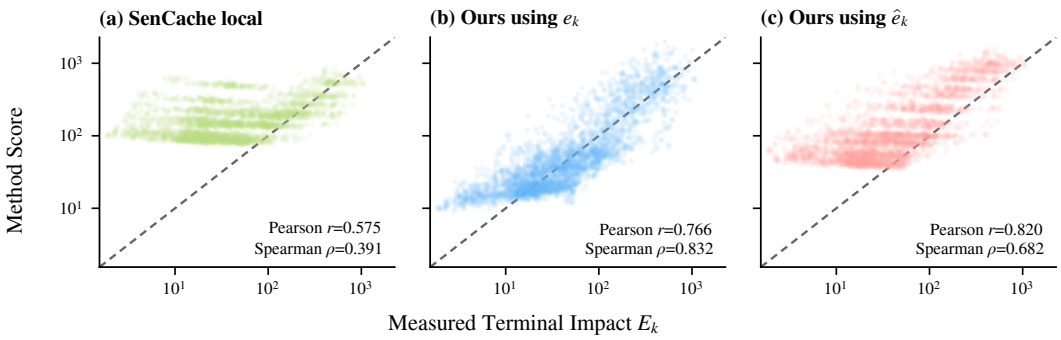

Figure 6: **Alignment of reuse scores with measured terminal impact on FLUX.1-dev.** Using profiles calibrated on eight separate prompts, we evaluate 3,240 candidates from 120 held-out prompts, each reusing the preceding step's feature. The panels compare (a) the SenCache (Haghighi & Alahi, 2026) local score, (b) our score using measured e_k, and (c) our practical score using $\widehat{e}_k$. Dashed lines indicate numerical agreement in panels (b) and (c). Both our scores achieve higher Pearson and Spearman correlations with E_k than the local score.

Table 4: **Sensitivity-profile transfer across FLUX sampling configurations.** Profiles transferred from the default 28-step setting through noise-level interpolation retain fidelity comparable to target-specific calibration.

Inference setting	Sensitivity profile	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Default schedule (50 steps)	Recalibrated for 50 steps	20	29.0109	0.9063	0.0837
	Transferred from default 28 steps	20	29.1106	0.9118	0.0804
Uniform schedule (28 steps)	Recalibrated for uniform schedule	10	28.9675	0.8921	0.1151
	Transferred from default 28 steps	10	29.2888	0.8915	0.1151

Terminal-impact weighting across local estimators. We next examine whether accounting for terminal impact improves reuse decisions across different local estimators. Table 5 combines local estimators from TeaCache, DiCache, MagCache, and SenCache with the scheduler step size and calibrated terminal sensitivity. At 10 NFE, terminal-impact weighting improves all three fidelity metrics for every estimator. These gains support using terminal impact rather than the local signal alone to guide reuse decisions across different local estimators.

Table 5: **Terminal-impact weighting with different local estimators on FLUX.1-dev.** Applying our weighting to each estimator improves reconstruction fidelity at 10 NFE.

Method	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
TeaCache	10	16.894	0.6747	0.3957
+ Ours	10	**22.070**	**0.8044**	**0.2096**
DiCache	10	22.800	0.8051	0.2190
+ Ours	10	**24.213**	**0.8299**	**0.1743**
MagCache	10	22.005	0.8006	0.2127
+ Ours	10	**23.810**	**0.8325**	**0.1745**
SenCache	10	22.108	0.8028	0.2129
+ Ours	10	**24.671**	**0.8363**	**0.1695**

5 Conclusion

We presented TSenCache, a training-free caching method that bases online reuse decisions on estimated terminal impact rather than local denoiser-output difference. Our analysis traced how the scheduler update and remaining denoising trajectory translate local output differences into timestep-dependent terminal effects. TSenCache combines the scheduler step size and calibrated terminal sensitivity with a sample-adaptive local-error estimate to guide online reuse. Keeping the scheduler step size separate from the calibrated sensitivity profiles further supports their reuse across the tested sampling settings through noise-level interpolation. Experiments on FLUX.1-dev and Wan2.1 demonstrated that TSenCache preserves higher reconstruction fidelity than the evaluated caching methods at comparable inference speed. Additional FLUX experiments showed gains across different local estimators and a range of computation budgets.

AI USE STATEMENT

We used generative AI tools to assist with code implementation and debugging, discussions of the methodology and experimental design, interpretation of results, and manuscript editing. The authors take responsibility for the final methodology, results, and content of this paper.

REFERENCES

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. In *International Conference on Learning Representations*, volume 2026, pp. 73778–73803, 2026.

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

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.

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 *Forty-first international conference on machine learning*, 2024.

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. *arXiv preprint arXiv:2511.00090*, 2025.

Yasaman Haghighi and Alexandre Alahi. Sencache: Accelerating diffusion model inference via sensitivity-aware caching. *arXiv preprint arXiv:2602.24208*, 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*, 36:13237–13249, 2023.

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

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)*, pp. 21807–21818. IEEE, 2024.

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.

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

Mingkun Lei, Tong Zhao, Liangyu Yuan, and Chi Zhang. Budget-constrained step-level diffusion caching. *arXiv preprint arXiv:2606.13496*, 2026.

Yanjing Li, Sheng Xu, Xianbin Cao, Xiao Sun, and Baochang Zhang. Q-dm: An efficient low-bit quantized diffusion model. *Advances in neural information processing systems*, 36:76680–76691, 2023.

Bin Lin, Yunyang Ge, Xinhua Cheng, Zongjian Li, Bin Zhu, Shaodong Wang, Xianyi He, Yang Ye, Shenghai Yuan, Liuhan Chen, et al. Open-sora plan: Open-source large video generation model. *arXiv preprint arXiv:2412.00131*, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7353–7363. IEEE, 2025a.

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)*, pp. 15853–15863. IEEE, 2025b.

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*, pp. 10024–10033, 2025c.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in neural information processing systems*, 35:5775–5787, 2022.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15762–15772. IEEE, 2024.

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.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

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. In *International Conference on Learning Representations*, volume 2024, pp. 1862–1874, 2024.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10674–10685. ieee, 2022.

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*, 35:36479–36494, 2022.

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.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1972–1981. IEEE, 2023.

Junhyuk So, Jungwon Lee, Daehyun Ahn, Hyungjun Kim, and Eunhyeok Park. Temporal dynamic quantization for diffusion models. *Advances in neural information processing systems*, 36:48686–48698, 2023.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

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

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.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

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)*, pp. 6211–6220. IEEE, 2024.

Xiaoju Ye. calflops: a flops and params calculate tool for neural networks in pytorch framework, 2023. URL `https://github.com/MrYxJ/calculate-flops.pytorch`.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6613–6623. IEEE, 2024b.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36:49842–49869, 2023.

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, 2025.

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.

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, 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*, 2026.

A CALIBRATION DETAILS

Algorithm 1 summarizes how the three sensitivity profiles are obtained from a shared set of calibration trajectories. For each prompt, we first generate a reference trajectory without cache reuse and store the latent, velocity, and reusable feature at every step. We estimate the local profiles by separately replacing the latent and timestep inputs with their values from the preceding step. For the terminal profile, we test reuse of the preceding step's feature separately at each step after the first, completing the remaining denoising steps without further reuse. We then average all three profiles across the calibration prompts.

In the algorithm, $h^F_{k,n}$ denotes the reusable feature stored during full computation at step k. DENOISEWITHFEATURE evaluates the denoiser with the feature cached at the preceding step, while FULLSUFFIX completes the remaining denoising steps without cache reuse. The resulting profiles are indexed by the noise levels of the calibration schedule and can be interpolated at the noise levels of a target schedule.

Algorithm 1 TSenCache sensitivity calibration

Require: Calibration prompts $\mathcal{C} = \{c_n\}_{n=1}^{N}$, schedule $\{t_k\}_{k=0}^{K}$, denoiser f_θ, decoder $\mathcal{D}$
Initialize $A_x(k), A_t(k), A_F(k+1) \leftarrow 0$
for $n = 1, \dots, N$ **do**
  Run full sampling for c_n, storing $\{x^F_{k,n}, v^F_{k,n}, h^F_{k,n}\}$ and terminal output I^F_n, with $I^F_{k,n} = I^F_n$ for every step k
  for $k = 1, \dots, K-1$ **do**
    $\Delta x_{k,n} \leftarrow x^F_{k-1,n} - x^F_{k,n}$, $\Delta t_k \leftarrow t_{k-1} - t_k$
    $v^x_{k,n} \leftarrow f_\theta(x^F_{k-1,n}, t_k, c_n)$
    $v^t_{k,n} \leftarrow f_\theta(x^F_{k,n}, t_{k-1}, c_n)$
    $A_x(k) \leftarrow A_x(k) + \|v^x_{k,n} - v^F_{k,n}\|_2 / \|\Delta x_{k,n}\|_2$
    $A_t(k) \leftarrow A_t(k) + \|v^t_{k,n} - v^F_{k,n}\|_2 / |\Delta t_k|$
    $\widetilde{v}_{k,n} \leftarrow \text{DENOISEWITHFEATURE}(x^F_{k,n}, t_k, c_n, h^F_{k-1,n})$
    $x^C_{k+1,n} \leftarrow x^F_{k,n} + (t_{k+1} - t_k)\widetilde{v}_{k,n}$
    $I^C_{k,n} \leftarrow \mathcal{D}(\text{FULLSUFFIX}(x^C_{k+1,n}, k+1, c_n))$
    $A_F(k+1) \leftarrow A_F(k+1) + \|I^F_n - I^C_{k,n}\|_2 / \|x^F_{k+1,n} - x^C_{k+1,n}\|_2$
  end for
end for
return $\alpha_x = A_x/N, \quad \alpha_t = A_t/N, \quad \alpha_F = A_F/N$

Offline cost and reuse. TSenCache calibrates its sensitivity profiles using local probes and full suffix rollouts, which require only forward passes and can be evaluated independently across prompts. For N calibration prompts and K sampling steps, this procedure requires $O(NK^2)$ full denoiser evaluations, as summarized in Algorithm 1. Under our default settings, the cost per prompt is 433 full-computation steps for FLUX and 1,324 for Wan2.1, corresponding to approximately 15.5 and 26.5 standard generations, respectively. With these settings, the recorded calibration time per prompt is approximately 7 minutes for FLUX and 2 hours 50 minutes for Wan2.1 on a single NVIDIA RTX A6000. As in prior calibration-based caching methods (Haghighi & Alahi, 2026; Gao et al., 2025), this cost is incurred offline and amortized over subsequent generations by reusing the profiles across prompts and reuse tolerances. Noise-level interpolation further allows the profiles to be transferred across the step counts and noise schedules tested in Table 4, avoiding recalibration for these settings.

Score normalization. The derivation and score-validation figures use unnormalized Euclidean differences. The implementation tolerances reported in Section 4.1 are applied after dividing S_k by the square root of the number of scalar elements in the decoded RGB image or video. This fixed rescaling within each output configuration changes the numerical tolerance but not the ordering of reuse scores.

B Effect of Calibration-Set Size

Figure 7 evaluates whether the sensitivity profiles can be estimated reliably from the eight prompts used in our main experiments. For both FLUX.1-dev and Wan2.1, the eight-prompt profiles capture the main timestep-dependent trends of the 128-prompt references across the latent, timestep, and terminal components. Although their magnitudes differ at some timesteps, the overall profile shapes remain similar, supporting the use of a small calibration set to capture the sensitivity variation used by TSenCache.

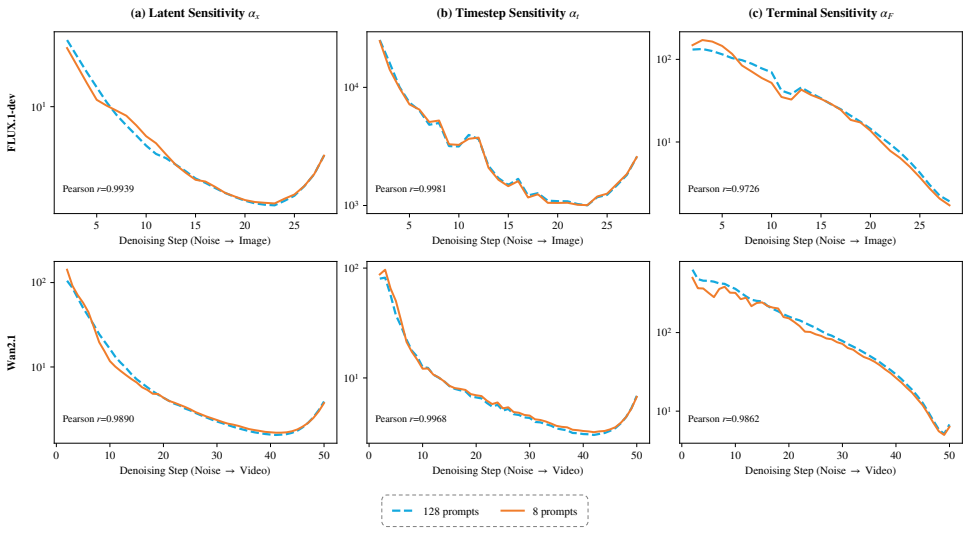

Figure 7: **Effect of calibration-set size on the sensitivity profiles.** Across FLUX.1-dev and Wan2.1, eight prompts capture the main timestep-dependent trends of the latent, timestep, and terminal profiles estimated from 128 prompts.

C Validation of the Local-Error Estimator

We separately examine the SenCache local-error estimator used by TSenCache on the same 3,240 held-out reuse candidates as in Figure 6. Figure 8 compares the online estimate $\widehat{e}_k$ with the measured local output difference e_k. Their positive correlation indicates that $\widehat{e}_k$ tracks the relative size of local errors, supporting its use as a thresholding signal even when its magnitude differs from e_k.

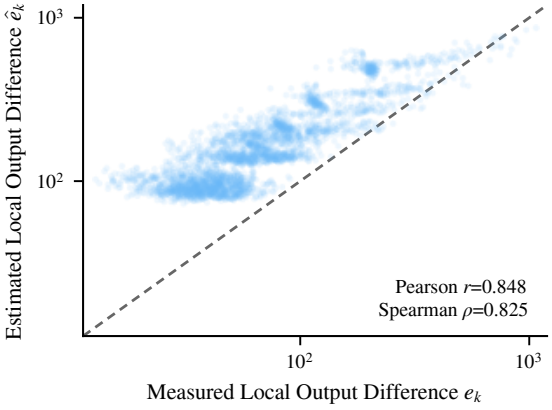

Figure 8: **Validation of the local-error estimator on FLUX.1-dev.** We compare $\widehat{e}_k$ with measured e_k over 3,240 candidates from 120 held-out prompts, using profiles calibrated on eight separate prompts. The dashed line indicates numerical agreement, and the annotation reports Pearson and Spearman correlations.

D SENSITIVITY TO CACHE AGE

The terminal-sensitivity profiles are calibrated using features from the preceding step, whereas inference can reuse features from several steps earlier. We examine whether these profiles remain informative as the cache gap increases by evaluating gaps of one to four steps on 120 held-out FLUX prompts, keeping the eight-prompt calibration unchanged.

Table 6 compares the same three scores as Figure 6 using Pearson and Spearman correlations with measured terminal impact. Both terminal-impact scores achieve higher correlations than the SenCache local score across all tested gaps. The practical score retains this advantage with the online local-error estimate, suggesting that the calibrated terminal sensitivity remains useful for assessing the impact of older cached features without recalibration.

Table 6: **Alignment with terminal impact across cache gaps on FLUX.1-dev.** Each gap is evaluated on 2,880 candidates using the same profiles calibrated at gap 1. For a matched comparison across gaps, we restrict evaluation to the shared step range 4–27. Both terminal-impact scores retain higher correlations than the local score across all tested gaps.

Cache gap	SenCache local		Ours using e_k		Ours using $\widehat{e}_k$	
	Pearson	Spearman	Pearson	Spearman	Pearson	Spearman
1	0.095	0.185	0.662	0.769	0.604	0.555
2	0.119	0.210	0.693	0.771	0.664	0.583
3	0.132	0.222	0.690	0.761	0.688	0.600
4	0.145	0.234	0.684	0.749	0.698	0.609

E EFFECT OF EARLY FULL COMPUTATION

Table 7 examines whether caching methods can preserve reconstruction fidelity without separate rules that force full computation in the early steps. Disabling these rules lowers the fidelity of DiCache, MagCache, and SenCache at the same target NFE budget, showing that these methods benefit from early-step protection beyond their reuse scores. In contrast, TSenCache achieves higher fidelity without an early-retention rule than these baselines achieve with their early-retention rules enabled. Its terminal-impact score accounts for early-step sensitivity and directs computation toward those steps.

Table 7: **Effect of the early full-computation rule on FLUX.1-dev.** Disabling the rule reduces baseline fidelity, whereas TSenCache achieves the best results without using it.

Method	Early full computation	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
DiCache	Enabled	10	22.800	0.8051	0.2190
DiCache	Disabled	10	17.191	0.6924	0.3648
MagCache	Enabled	10	22.005	0.8006	0.2127
MagCache	Disabled	10	20.165	0.7744	0.2526
SenCache	Enabled	10	22.108	0.8028	0.2129
SenCache	Disabled	10	21.068	0.7738	0.2516
TSenCache	Not used	10	**24.671**	**0.8363**	**0.1695**

F FULL-COMPUTATION PATTERNS

Figure 9 compares how each method allocates full model evaluations across the denoising trajectory. Several baselines produce deterministic refresh bands or explicitly retain dense computation in the early steps before relying on their respective cache signals. TSenCache instead assigns a high full-computation ratio to the sensitive early region and then performs sparser, prompt-dependent refreshes over the remaining steps without an explicit early-retention rule. This pattern is consistent with the calibrated terminal sensitivity and shows how the proposed score translates timestep-dependent terminal effect into an adaptive computation schedule.

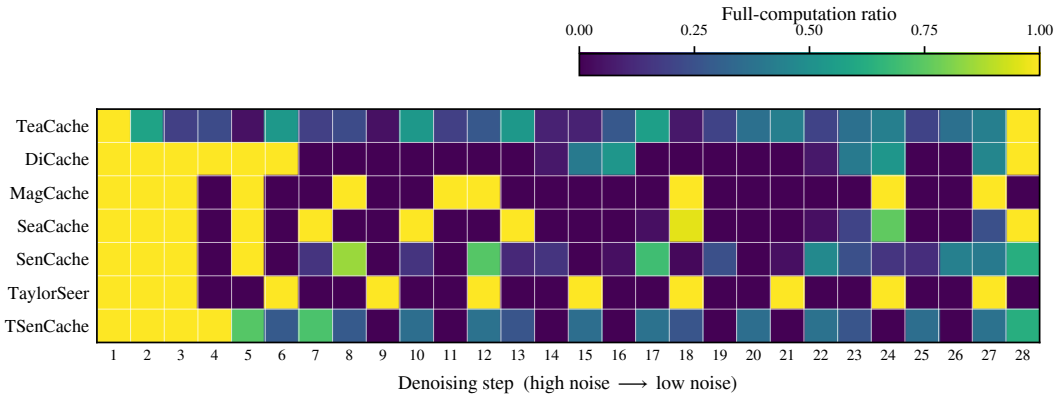

Figure 9: **Full-computation patterns across FLUX denoising steps.** Each cell shows the fraction of 200 DrawBench prompts that perform full computation at the corresponding step. TSenCache allocates more computation to the sensitive early steps without an explicit early-retention rule.

G Comparison with Offline Scheduling Methods

We extend the FLUX comparison to LeMiCa (Gao et al., 2025) and BudCache (Lei et al., 2026), which optimize a fixed reuse schedule offline under a specified computation budget. Like TSenCache, these methods consider how cache reuse affects the terminal output rather than relying on local error alone. Their optimized schedules are shared across samples, whereas TSenCache estimates the terminal impact of the current reuse decision during inference to determine whether full computation is needed.

Table 8: **Comparison with offline scheduling methods on FLUX.1-dev.** TSenCache achieves the best PSNR, SSIM, and LPIPS at comparable full-computation budgets.

Method	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
LeMiCa	10	<u>24.166</u>	<u>0.8264</u>	0.1866
BudCache	10	24.065	0.8253	<u>0.1759</u>
TSenCache	10	**24.671**	**0.8363**	**0.1695**

Table 8 compares these approaches on 200 DrawBench prompts under the 28-step FLUX setting, using the TSenCache result from Table 1. At comparable full-computation budgets, TSenCache achieves higher PSNR and SSIM and lower LPIPS than both baselines. These results show that TSenCache's fidelity advantage extends to methods that also target terminal-output preservation, supporting terminal-impact estimation as an online alternative to offline schedule optimization.

H Limitations

Although TSenCache improves fidelity to the full-computation output, it has several limitations. First, more aggressive reuse can increase deviations from that output when cached features no longer reflect the current denoising state. Second, the terminal-impact score relies on a local-error proxy and calibrated trajectory sensitivity, so it may not capture the exact effect of every reuse decision. This can lead to suboptimal decisions for individual samples. Finally, TSenCache is designed to reproduce the full-computation generator's output. Consequently, prompt mismatches or visual artifacts already present in that output may remain even when reconstruction fidelity is high.

I Additional Qualitative Results

We provide enlarged comparisons with SenCache on FLUX.1-dev and Wan2.1 using the settings of the main experiments, with full computation as the reference.

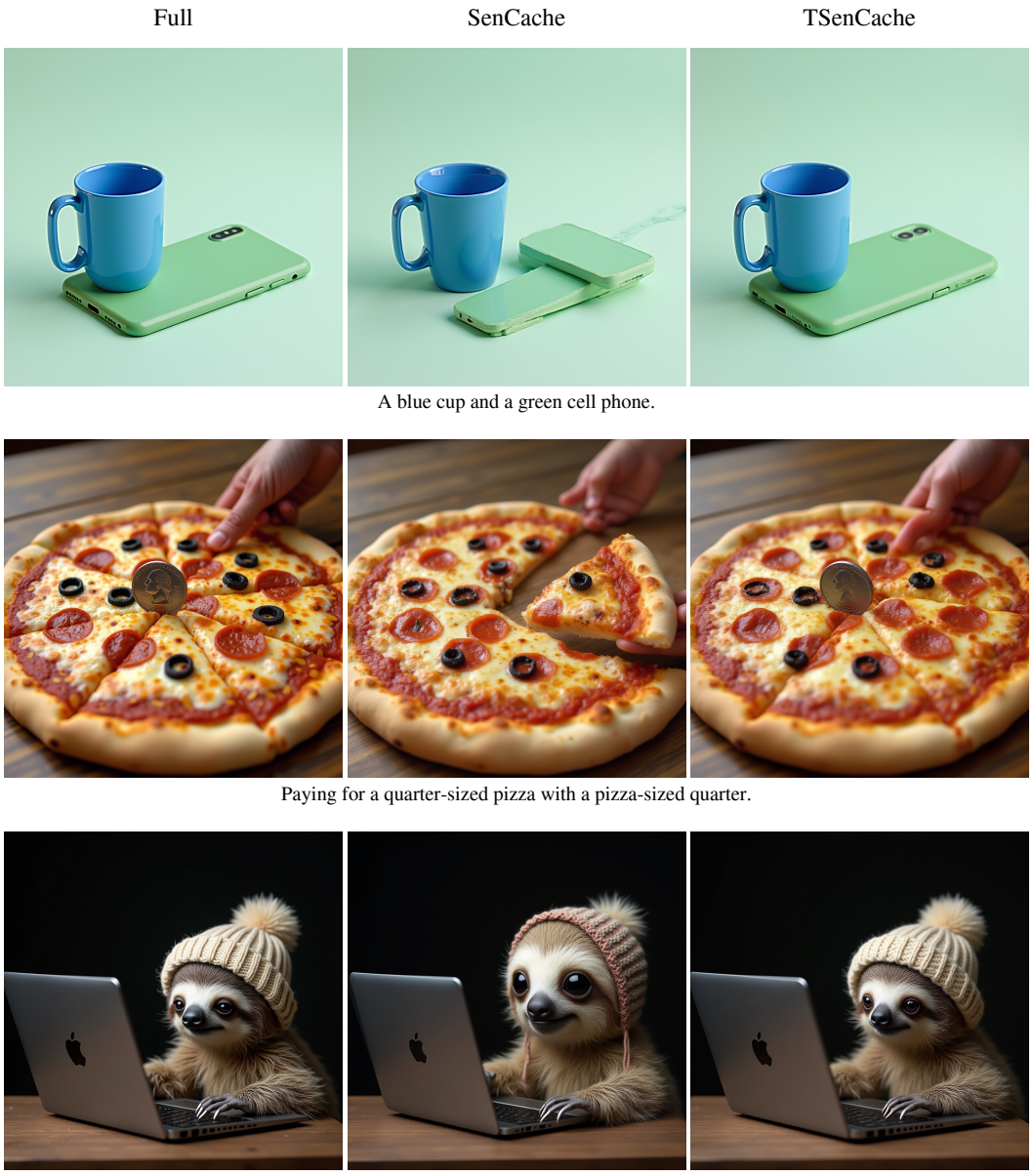

Figure 10: **Additional qualitative comparisons on FLUX.1-dev.** TSenCache retains the coin on the pizza and more closely preserves the phone shape and the sloth's appearance in the full-computation reference, whereas SenCache (Haghighi & Alahi, 2026) changes these features.

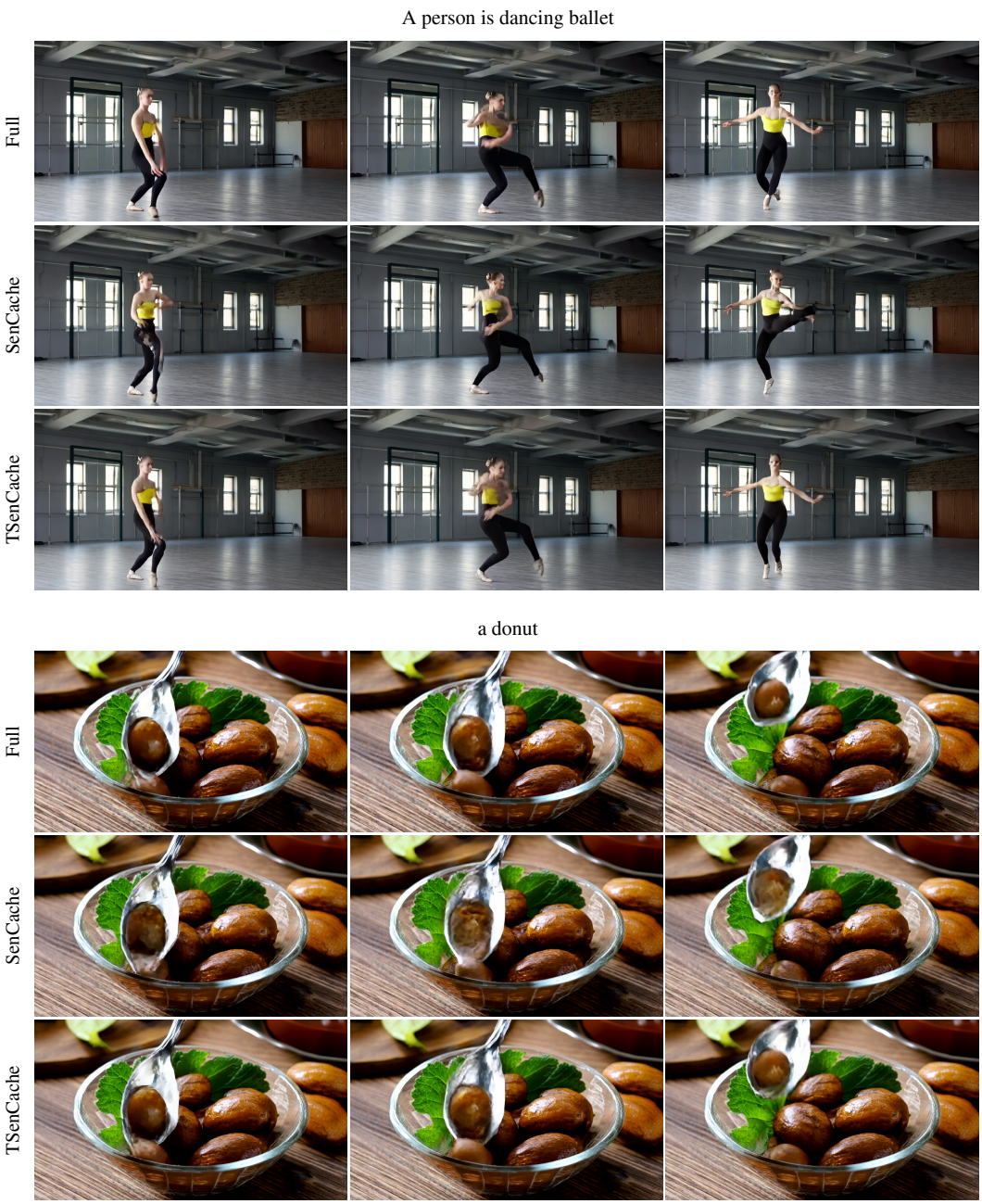

Figure 11: **Additional qualitative comparisons on Wan2.1.** Each video is shown at three matching frame indices in chronological order. TSenCache more closely preserves the ballet poses and the shape of the food on the spoon in the full-computation reference than SenCache (Haghighi & Alahi, 2026).