OriCache: Orientation-Guided Feature Caching for DiT Acceleration

Anonymous Authors[1]

Abstract

Diffusion Transformers (DiTs) deliver high-quality generation but require repeated Transformer evaluations over many denoising steps, making inference expensive. Feature caching reduces this cost by reusing intermediate computations across denoising steps, where cache scheduling determines when cached computations can be reused or refreshed. Existing training-free scheduling methods have explored lightweight trajectory cues such as feature distance, update magnitude, and spectral variation. However, the orientation consistency of local updates remains underexplored, despite its potential to reflect the stability of local denoising trajectories. In this work, we propose OriCache, a training-free feature caching method that explicitly incorporates local update orientation into cache scheduling. OriCache computes a cache score from lightweight block inputs by combining update-scale variation with directional alignment between consecutive local updates. By jointly considering how much local updates change and whether their directions remain consistent, OriCache captures complementary aspects of denoising trajectory evolution for recomputation decisions. Experiments on FLUX.1-[dev] and Stable Diffusion 3.5-Large show that OriCache improves over representative static and adaptive caching baselines, and ablations confirm that combining magnitude and orientation yields more reliable cache decisions than either cue alone.

1. Introduction

Diffusion Transformers (DiTs) (Peebles & Xie, 2023; Black Forest Labs, 2024; Ma et al., 2024b; Yang et al., 2024; Esser et al., 2024) have emerged as a powerful backbone for modern generative models, achieving strong scalability and high visual quality in image and video generation. Compared with U-Net-based diffusion models (Ho et al., 2020; Rombach et al., 2022; Podell et al., 2023), Transformer-based architectures provide expressive token interactions and are increasingly used in large-scale diffusion foundation models. However, DiT inference remains expensive because generation requires many iterative denoising steps, each involving repeated evaluations of large Transformer blocks. This repeated computation becomes a major bottleneck when deploying DiT-based models under practical latency or resource constraints.

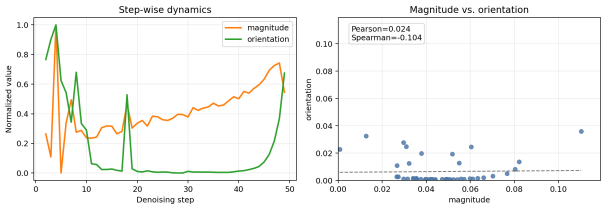

Figure 1. Step-wise dynamics of two aspects of local trajectory evolution on FLUX.1-[dev]: update magnitude and update orientation. These two aspects show different behaviors along the denoising trajectory and have low correlation, motivating orientation consistency as an additional cue for cache scheduling.

A common approach (Selvaraju et al., 2024; Ma et al., 2024a; Zhao et al., 2024; Kahatapitiya et al., 2025; Bu et al.; Zhou et al., 2025) to reducing this cost is to exploit temporal redundancy across neighboring denoising steps. Since intermediate representations often evolve smoothly along the sampling trajectory, feature caching methods reuse previously computed features or module outputs across denoising steps instead of recomputing them at every step. A central problem in feature caching is cache scheduling, which determines when cached computations can be safely reused and when block outputs should be recomputed. Early methods typically use a static refresh schedule, periodically recomputing features after a fixed number of denoising steps. More recent studies have shifted toward adaptive cache scheduling, where reuse decisions are made online using signals from the denoising process.

Adaptive cache scheduling depends on how well the chosen cue reflects the local dynamics of the denoising trajectory. Prior training-free caching methods have explored different proxies for cache reliability. Feature-distance-based approaches (Liu et al., 2025) measure how far intermediate representations move across denoising steps, while magnitude-aware methods (Ma et al., 2025) track the scale of feature or residual updates to identify steps that require recomputation. More recently, spectral-aware methods (Chung et al., 2026)

Figure 2. Qualitative comparison on FLUX.1-[dev] with four configurations: 50-step baseline, FORA, TeaCache, and OriCache. Compared with fixed-schedule and adaptive feature-change-based caching, OriCache better preserves the context and visual appearance of the reference generation, suggesting that orientation consistency provides a meaningful cue for cache scheduling.

have used frequency-domain behavior as an additional signal for detecting changes in the denoising trajectory. These cues provide useful signals for estimating when cached computations become less representative of the current denoising step. In this work, we investigate another aspect of local trajectory evolution: the orientation consistency of update directions.

A denoising trajectory can be understood as a sequence of local updates that gradually transform a noisy latent representation into a clean sample. From this perspective, each step is characterized not only by how far the representation moves, but also by the direction in which it moves. We therefore describe local trajectory dynamics from two complementary perspectives: update magnitude and update orientation. Update magnitude reflects the amount of movement in representation space, whereas update orientation reflects the local direction of trajectory evolution.

As shown in Figure 1, update magnitude and update orientation exhibit different step-wise behaviors during denoising. This difference is important because diffusion inference proceeds by repeatedly applying local updates along the denoising trajectory. While magnitude indicates how far the representation moves between steps, orientation indicates the local direction in which the trajectory continues. Therefore, a change in orientation can signal a change in the local trajectory direction, even when the accumulated feature distance evolves smoothly. This suggests that orientation provides meaningful trajectory information that is not fully reflected by magnitude-based signals. Since cache scheduling decides where cached computations should approximate later points on the trajectory, such directional information can help identify recomputation points more reliably. This motivates using orientation consistency as an additional trajectory cue for cache scheduling.

Based on this observation, we propose OriCache, an orientation-guided feature caching method for DiT acceleration. OriCache estimates whether the current denoising step remains locally consistent with the recent trajectory before executing expensive block computation. To this end, OriCache computes a cache score from block inputs using two factors: an amplitude ratio and directional alignment. The amplitude ratio captures update-scale variation, while directional alignment measures whether the local update direction remains consistent. By combining these two factors, OriCache accounts for both how much the trajectory changes and whether it continues in a similar direction. The resulting score determines whether the cached computation can be reused or the block should be recomputed.

We evaluate OriCache on FLUX.1-[dev] and Stable Diffusion 3.5-Large under various acceleration conditions. Experiments show that OriCache improves the latency-fidelity trade-off over representative static and adaptive caching baselines. At comparable inference cost, OriCache better preserves the reference generation in terms of reconstruction and perceptual quality across both models. These results suggest that orientation consistency provides a meaningful complementary cue for cache scheduling in DiT inference.

In summary, our contributions are as follows:

- We analyze cache scheduling from the perspective of denoising trajectory evolution, showing that feature-change signals alone may not fully capture local trajectory dynamics.
- We propose OriCache, an orientation-guided feature caching method that jointly considers update magnitude and update orientation through a unified cache score computed from block inputs.
- We demonstrate that orientation consistency provides a complementary signal for cache scheduling, improving the latency-fidelity trade-off over representative caching baselines on FLUX.1-[dev] and Stable Diffusion 3.5-Large.

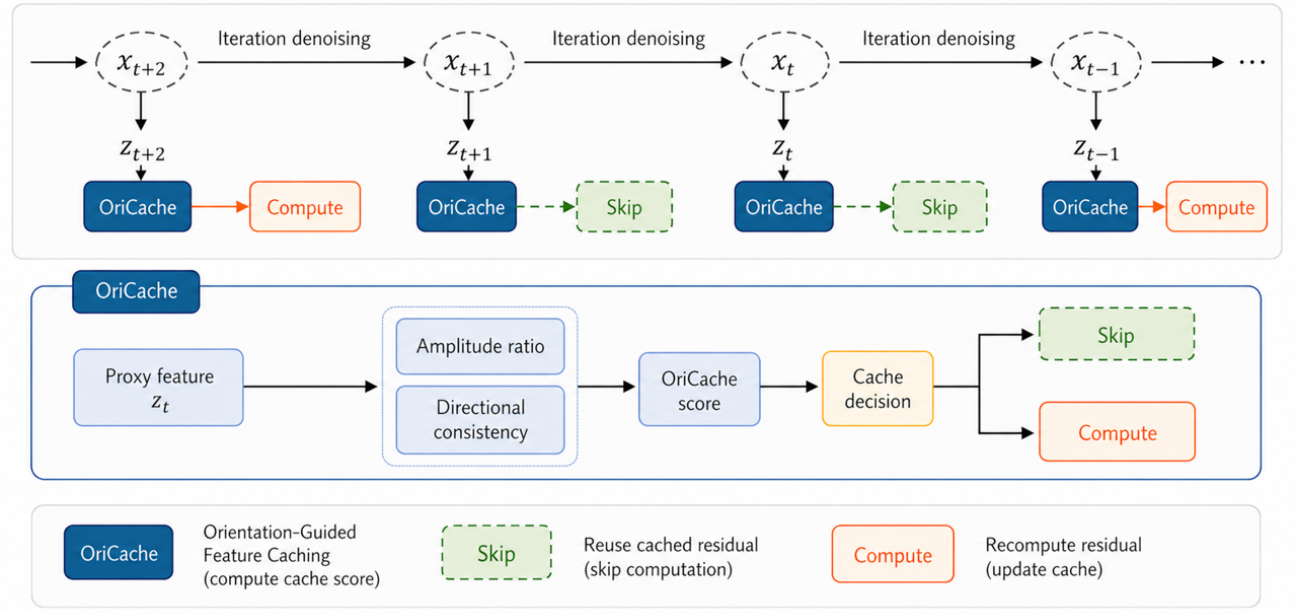

Figure 3. Overview of OriCache. Before executing a target DiT block, OriCache estimates a cache score from lightweight block inputs using update-scale variation and directional alignment. The score determines whether to reuse the cached output or recompute the block.

2. Related Work

2.1. Feature Caching for Diffusion Acceleration

Feature caching accelerates diffusion inference by exploiting temporal redundancy across neighboring denoising steps. Since intermediate representations often evolve smoothly during sampling, cached features or block outputs can be reused for nearby steps instead of being recomputed every time. Early cache-based methods typically rely on static refresh schedules. For example, DeepCache (Ma et al., 2024a) reuses high-level features in U-Net-based diffusion models, while FORA (Selvaraju et al., 2024) applies periodic feature reuse to Diffusion Transformers by caching intermediate Transformer computations.

Recent training-free adaptive methods improve cache scheduling by selecting reuse steps online based on the observed dynamics of the denoising trajectory. TeaCache uses feature-based signals to estimate cache reuse decisions (Liu et al., 2025), while MagCache (Ma et al., 2025) studies magnitude-aware changes for cache scheduling. SeaCache (Chung et al., 2026) further explores spectral behavior during denoising and uses spectral-aware cues to guide cache decisions. AdaCache (Kahatapitiya et al., 2025), DiCache (Bu et al.), and EasyCache (Zhou et al., 2025) also study adaptive cache decisions from sample-specific or runtime-adaptive perspectives. These studies suggest that cache scheduling can benefit from cues that capture different properties of the denoising trajectory. OriCache follows this direction by investigating orientation consistency as another trajectory cue for cache scheduling.

3. Method

3.1. Overview

OriCache accelerates DiT inference by reusing intermediate block outputs across denoising steps. For each target block, OriCache computes a cache score from the block input before executing the expensive block computation. The score measures how the local input update changes between consecutive denoising steps in terms of both update scale and update direction. Specifically, OriCache uses an update amplitude ratio to capture magnitude variation and cosine alignment to capture directional consistency. If the score indicates that the local trajectory remains stable, the cached block output is reused. Otherwise, the block is recomputed and the cache is refreshed. This design enables training-free cache scheduling with small overhead compared with Transformer block computation.

3.2. Orientation-Guided Feature Caching

Following TeaCache (Liu et al., 2025), we use a block-level input z_t at denoising step t as a lightweight proxy signal for cache scheduling. We define the step-wise update vector as

$$\Delta z_t = z_t - z_{t-1}. \tag{1}$$

This update provides a local signal for estimating the evolution of the denoising trajectory. To jointly capture scale

variation and directional evolution, OriCache compares consecutive input updates through their relative magnitude and orientation.

Specifically, we first define the update amplitude ratio as

$$\rho_t = \frac{\|\Delta z_t\|_2}{\|\Delta z_{t-1}\|_2 + \epsilon}, \tag{2}$$

where ϵ is a small constant for numerical stability. The amplitude ratio measures how much the current update scale changes relative to the previous update.

We then define the relative orientation between consecutive updates by

$$\cos\theta_t = \frac{\langle \Delta z_t, \Delta z_{t-1} \rangle}{\|\Delta z_t\|_2 \|\Delta z_{t-1}\|_2 + \epsilon}, \tag{3}$$

where θ_t denotes the orientation difference between Δz_t and Δz_{t-1}. A small θ_t indicates that consecutive updates follow a similar local direction, while a large θ_t indicates a directional change along the denoising trajectory.

Using these two quantities, we measure how far the current local update deviates from the previous one in both scale and orientation. The ideal case for cache reuse corresponds to $\rho_t \approx 1$ and $\theta_t \approx 0$, where the current update has both a similar magnitude and a similar direction to the previous update. We therefore define the cache score as

$$s_t = \left\| \begin{bmatrix} \rho_t \cos\theta_t \\ \rho_t \sin\theta_t \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|_2^2 = \rho_t^2 + 1 - 2\rho_t \cos\theta_t. \tag{4}$$

This score measures the squared mismatch between the current relative update and the unit reference update in an jointly penalizes deviations. As a result, s_t remains small when the local trajectory evolves consistently in both scale and orientation, and becomes large when either the update magnitude or direction changes.

3.3. OriCache Scheduling

OriCache uses the accumulated cache score since the most recent refresh to decide whether to reuse or recompute the target block. Let r denote the most recent denoising step at which the target block was recomputed. We define the accumulated score at step t as

$$A_t = \sum_{i=r+1}^{t} s_i. \tag{5}$$

If A_t remains below the threshold τ, OriCache reuses the cached output from step r. Otherwise, it recomputes the target block and refreshes the cache:

$$\hat{y}_t = \begin{cases} y_r, & A_t < \tau, \\ F(z_t), & A_t \geq \tau, \end{cases} \tag{6}$$

where $F(\cdot)$ denotes the target DiT block, y_r is the cached block output from the most recent recomputed step, and τ is the cache threshold. When recomputation occurs, OriCache sets the refresh step to the current step, $r \leftarrow t$, stores the newly computed output as the cache, $y_r \leftarrow F(z_t)$, and resets the accumulated score.

The threshold τ controls the operating point between generation quality and inference efficiency. A larger threshold allows more cache reuse and stronger acceleration, while a smaller threshold triggers more frequent recomputation and better preserves fidelity. Because the score is computed from block inputs before the target block execution, OriCache requires neither auxiliary networks nor model retraining. This provides a simple way to study orientation consistency as a cache scheduling cue in DiT inference.

4. Experiments

4.1. Experimental Setup

We evaluate OriCache on two text-to-image diffusion models: FLUX.1-[dev] (Black Forest Labs, 2024) and Stable Diffusion 3.5-Large (Esser et al., 2024). We use the DrawBench prompt set (Saharia et al., 2022) and generate images at a resolution of 1024×1024. For each model, we use the 50-step generation as the reference output and evaluate accelerated results under various acceleration conditions to verify the effectiveness of the proposed orientation cue across different cache reuse levels. For efficiency, we report end-to-end latency and executed denoising FLOPs after cache decisions. For visual quality, we compute PSNR, SSIM, and LPIPS against the 50-step reference outputs. To contextualize the results, we compare OriCache with representative cache scheduling baselines, including FORA (Selvaraju et al., 2024) as a static scheduler and TeaCache (Liu et al., 2025) as an adaptive feature-change-based scheduler. All experiments are conducted on 2 NVIDIA RTX 4090 GPU.

4.2. Comparison Results

Table 1 compares direct step reduction and cache-based acceleration on FLUX.1-[dev] and Stable Diffusion 3.5-Large. Directly reducing the number of denoising steps lowers latency, but it also noticeably degrades visual fidelity. On FLUX.1-[dev], reducing the sampling budget from 50 steps to 25 steps gives a 1.97× latency speedup, but the quality drops to 17.81 PSNR, 0.7512 SSIM, and 0.2884 LPIPS. The degradation becomes more severe at 17 steps, where the speedup increases to 2.82× but the quality further decreases to 15.75 PSNR, 0.6834 SSIM, and 0.3909 LPIPS. A similar trend is observed on Stable Diffusion 3.5-Large: the 25-step and 17-step baselines improve latency, but progressively reduce reconstruction and perceptual fidelity. These results

Table 1. Quantitative comparison on FLUX.1-[dev] and Stable Diffusion 3.5-Large. We compare direct step reduction and cache-based acceleration under different acceleration settings. OriCache improves visual fidelity over representative cache scheduling baselines at comparable latency, supporting orientation consistency as a transferable cache-scheduling cue.

Method	FLUX.1-[dev]					Stable Diffusion 3.5-Large				
	Efficiency		Visual Quality			Efficiency		Visual Quality		
	Latency↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓	Latency↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
50 steps	36.79	1.00	–	–	–	30.81	1.00	–	–	–
25 steps	18.66	1.97	17.81	0.7512	0.2884	15.90	1.94	16.32	0.7274	0.3489
17 steps	13.07	2.82	15.75	0.6834	0.3909	10.78	2.86	14.57	0.6372	0.4784
FORA ($N = 4$)	10.68	3.44	14.73	0.6524	0.4499	11.68	2.64	14.61	0.6433	0.4710
TeaCache ($\ell = 0.6$)	11.55	3.19	16.71	0.7053	0.3595	11.61	2.65	17.46	0.7371	0.3317
OriCache ($\tau = 1.5$)	11.83	3.11	**18.34**	**0.7379**	**0.3034**	11.62	2.65	**19.15**	**0.8089**	**0.2400**
FORA ($N = 6$)	7.81	4.71	14.04	0.6197	0.5169	7.23	4.26	13.53	0.5296	0.6331
TeaCache ($\ell = 1.0$)	8.32	4.42	15.32	0.6403	0.4662	8.72	3.53	15.72	0.6421	0.4671
OriCache ($\tau = 3.0$)	7.80	4.72	**15.99**	**0.6538**	**0.4403**	7.48	4.12	**16.57**	**0.7217**	**0.3752**

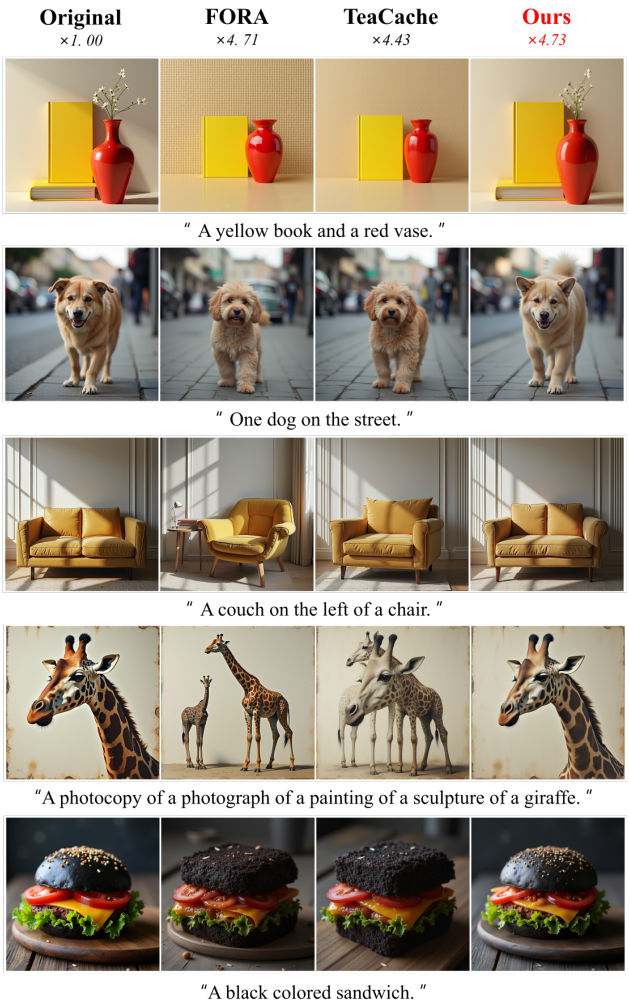

Figure 4. Qualitative comparison with the 50-step reference on FLUX.1-[dev] under high acceleration conditions. OriCache better preserves the original visual context and overall scene structure.

indicate that reducing the sampling budget alone changes the denoising trajectory and cannot reliably preserve the 50-step reference generation.

We next compare cache-based acceleration methods. In the moderate acceleration setting, FORA provides strong latency reduction but suffers from substantial quality degradation due to fixed periodic reuse. TeaCache improves fidelity by using adaptive cache decisions. OriCache further improves reference similarity at comparable latency. On FLUX.1-[dev], OriCache with $\tau = 1.5$ achieves 18.34 PSNR, 0.7379 SSIM, and 0.3034 LPIPS, outperforming TeaCache with $\ell = 0.6$. On Stable Diffusion 3.5-Large, OriCache achieves 19.15 PSNR, 0.8089 SSIM, and 0.2400 LPIPS, while maintaining nearly the same latency as TeaCache. This suggests that orientation consistency provides useful information for selecting recomputation points beyond feature-change-based adaptive scheduling.

The same tendency appears in the higher acceleration setting. FORA achieves the highest or comparable speedup, but its visual quality drops significantly on both models. TeaCache partially recovers fidelity, while OriCache consistently improves all visual quality metrics over TeaCache. On FLUX.1-[dev], OriCache with $\tau = 3.0$ improves PSNR from 15.32 to 15.99 and LPIPS from 0.4662 to 0.4403 compared with TeaCache. On Stable Diffusion 3.5-Large, OriCache improves PSNR from 15.72 to 16.57, SSIM from 0.6421 to 0.7217, and LPIPS from 0.4671 to 0.3752, while also achieving lower latency than TeaCache. These results show that the proposed orientation cue is not limited to FLUX.1-[dev], but also provides useful cache-scheduling information on another large-scale diffusion Transformer model.

Figure 4 shows qualitative comparisons with the 50-step reference. Consistent with the quantitative results, OriCache

better preserves the visual context of the reference generation under cache-based acceleration. Compared with fixed-schedule caching and feature-difference-based adaptive caching, OriCache maintains more faithful object structure, spatial layout, and local appearance. This qualitative behavior supports our hypothesis that orientation consistency captures trajectory information that is not fully reflected by feature-change magnitude alone.

We interpret this result from the perspective of denoising trajectory approximation. Cached block outputs are computed at previous denoising steps and reused to approximate later computations. When the local trajectory changes direction, reusing cached computations without considering directional consistency can lead to context drift or structural changes in the generated image. By using orientation as an additional cache-scheduling cue, OriCache can refresh the cache at steps where the local trajectory direction becomes less consistent. This helps the accelerated generation remain closer to the 50-step reference in both visual structure and perceptual appearance. Overall, the quantitative and qualitative results suggest that orientation consistency is a meaningful cue for cache scheduling.

4.3. Ablation Study

To verify whether orientation consistency provides a meaningful cue for cache scheduling, we conduct a component-wise ablation study on FLUX.1-[dev]. We compare three variants: a magnitude-only score based on update-scale variation, an orientation-only score based on directional consistency between consecutive updates, and the full OriCache score combining both cues. The 50-step generation is used as the reference for visual quality evaluation.

The magnitude-only criterion achieves 16.71 PSNR, 0.7053 SSIM, and 0.3595 LPIPS with 903.10T FLOPs, while the orientation-only criterion achieves higher PSNR and lower LPIPS, reaching 17.01 PSNR and 0.3534 LPIPS with a similar FLOPs budget of 919.38T. This suggests that orientation consistency is not redundant with magnitude-based change, but provides additional trajectory information for recomputation decisions under comparable computation.

Figure 5 shows a similar trend qualitatively. The orientation-only variant better preserves the overall composition and visual context of the 50-step reference, indicating that update direction is related to the structural evolution of the denoising trajectory. However, orientation alone does not fully capture update-scale variation, leaving differences in local appearance and fidelity.

Combining magnitude and orientation yields the best visual quality across all metrics. Compared with the magnitude-only criterion, the full score improves PSNR from 16.71 to 18.34, SSIM from 0.7053 to 0.7379, and LPIPS from 0.3595 to 0.3034, while using only a slightly larger FLOPs budget of 923.35T. These results indicate that the improvement is not simply due to substantially more computation; rather, orientation provides complementary information about directional changes in the denoising trajectory. This supports its use as a meaningful cue for cache scheduling.

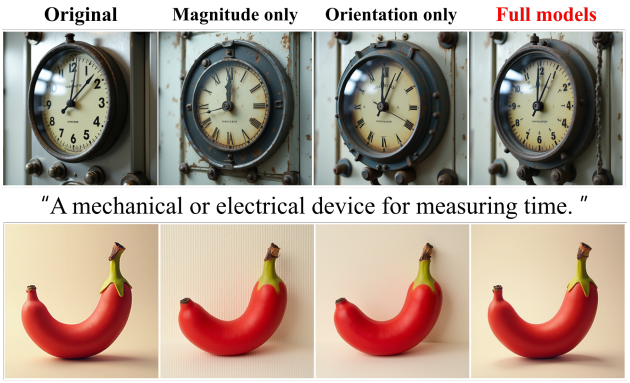

Figure 5. Qualitative ablation on FLUX.1-[dev]. The combined score better preserves local details and overall fidelity.

Table 2. Ablation on cache score components on FLUX.1-[dev]. The combined score improves fidelity at a similar FLOPs budget.

Criterion	FLOPs↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
Baseline	3710.50	1.00	–	–	–
Magnitude only	903.10	4.11	16.71	0.7053	0.3595
Orientation only	919.38	4.04	17.01	0.7047	0.3534
Mag. + Ori.	923.35	4.02	**18.34**	**0.7379**	**0.3034**

5. Conclusion

We presented OriCache, a training-free feature caching method that uses orientation consistency as a cue for DiT cache scheduling. OriCache is motivated by the view that cache scheduling can benefit from trajectory cues that describe not only how much the representation changes, but also how the local denoising direction evolves. To this end, OriCache computes a cache score from block inputs using update-scale variation and directional alignment. Experiments on FLUX.1-[dev] and component-wise ablations show that OriCache improves the latency-fidelity trade-off over representative static and adaptive caching baselines, while demonstrating that update orientation provides complementary information for cache scheduling. Qualitative results further indicate that orientation-aware scheduling helps preserve the visual context and structure of the reference generation under cache reuse. These findings suggest that update orientation is a meaningful trajectory cue for cache scheduling and may serve as a useful component for future training-free acceleration methods for DiTs.

References

Black Forest Labs. Announcing black forest labs. https://bfl.ai/blog/24-08-01-bfl, 2024. Accessed: 2026-04-28.

Bu, J., Ling, P., Zhou, Y., Wang, Y., Zang, Y., Lin, D., and Wang, J. Dicache: Let diffusion model determine its own cache. In *The Fourteenth International Conference on Learning Representations*.

Chung, J., Hyun, S., Lee, M., Han, B., Cha, G., Wee, D., Hong, Y., and Heo, J.-P. Seacache: Spectral-evolution-aware cache for accelerating diffusion models. *arXiv preprint arXiv:2602.18993*, 2026.

Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

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

Kahatapitiya, K., Liu, H., He, S., Liu, D., Jia, M., Zhang, C., Ryoo, M. S., and Xie, T. Adaptive caching for faster video generation with diffusion transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15240–15252, 2025.

Liu, F., Zhang, S., Wang, X., Wei, Y., Qiu, H., Zhao, Y., Zhang, Y., Ye, Q., and Wan, F. Timestep embedding tells: It's time to cache for video diffusion model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 7353–7363, 2025.

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

Ma, X., Wang, Y., Chen, X., Jia, G., Liu, Z., Li, Y.-F., Chen, C., and Qiao, Y. Latte: Latent diffusion transformer for video generation. *arXiv preprint arXiv:2401.03048*, 2024b.

Ma, Z., Wei, L., Wang, F., Zhang, S., and Tian, Q. Magcache: Fast video generation with magnitude-aware cache. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=KZn7TDOL4J.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

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

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35: 36479–36494, 2022.

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

Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

Zhao, X., Jin, X., Wang, K., and You, Y. Real-time video generation with pyramid attention broadcast. *arXiv preprint arXiv:2408.12588*, 2024.

Zhou, X., Liang, D., Chen, K., Feng, T., Chen, X., Lin, H., Ding, Y., Tan, F., Zhao, H., and Bai, X. Less is enough: Training-free video diffusion acceleration via runtime-adaptive caching. *arXiv preprint arXiv:2507.02860*, 2025.