RACE: REFRESH-ANCHORED CACHE FOR DIFFUSION ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow matching methods have made advanced progress in image and video generation, yet their iterative sampling remains computationally expensive. Cache-based acceleration explores computational redundancy and approximates velocity in certain skipped steps. However, velocity approximation errors accumulate through sampler updates, resulting in the perturbed state for velocity prediction in cache refresh. To improve velocity prediction in cache refresh, the state displacement in the preceding updates should be adjusted. Moreover, we propose that the distribution of state and velocity updates is related along local generation trajectory. This distribution constraint can be easily applied on the estimation of velocity at reuse steps. Building on these insights, we propose **RACE** (Refresh-Anchored Cache), a training-free framework that combines state-guided velocity extrapolation with refresh-anchored integral correction. RACE uses the evolving state to guide velocity prediction and refresh-endpoint errors to compensate for state deviations accumulated during reuse steps. Each refresh step thus serves as an anchor both for subsequent velocity approximation and preceding state correction. RACE is compatible with existing cache schedulers and requires no additional network computation or hyper-parameters tuning. Extensive experiments on FLUX.1-dev, Wan 2.1, HunyuanVideo, and MiniMax-H3 demonstrate that RACE achieves state-of-the-art latency-fidelity trade-offs. We will release our code soon.

1 INTRODUCTION

Diffusion (Ho et al., 2020; Song et al., 2021) and flow matching models (Lipman et al., 2023; Tong et al., 2026) have advanced high-quality image generation (Rombach et al., 2022; Wu et al., 2025; Esser et al., 2024) and video generation (Yang et al., 2025; Kong et al., 2024; Blattmann et al., 2023). Iterative network computations during multi-step sampling, however, remain a major inference bottleneck (Lu et al., 2022). Cache-based acceleration explores computational redundancy across adjacent sampling steps, replacing certain network computations with information from previous steps to reduce inference cost (Ma et al., 2024; Zhao et al., 2025; Lv et al., 2025).

Existing works (Zou et al., 2025; Kahatapitiya et al., 2025) mainly address two foundational questions: when to refresh the cache, and how to use cached information to approximate the current output (Bu et al., 2026). Cache scheduling allocates computation by identifying redundancy across steps. TeaCache (Liu et al., 2025a) estimates accumulated error from the discrepancy of inputs modulated by timestep embedding. SeaCache (Chung et al., 2026) improves the feature-distance measurement through spectral alignment, adapting refresh decisions to the dynamics of generation. Methods of cache reuse focus on improving approximation accuracy at reuse steps. TaylorSeer (Liu et al., 2025b) improves feature prediction based on the temporal evolution of historical features, and DiCache (Bu et al., 2026) uses online observations from shallow layers to align historical caches with the current feature trajectory. Together, adaptive cache schedules and more accurate cache utilization improve speed–quality performance.

However, it should be noted that when network computations are skipped, the intermediate state remains evolving (Chu et al., 2025). In the generation process driven by the velocity field, errors in the estimated velocity propagate through sampler updates to subsequent states and affect full network computation at later refresh steps (Peng et al., 2026; Haghighi & Alahi, 2026). Specifically, a refresh step predicts the "precise" velocity with current state, while the input state is already contaminated

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.

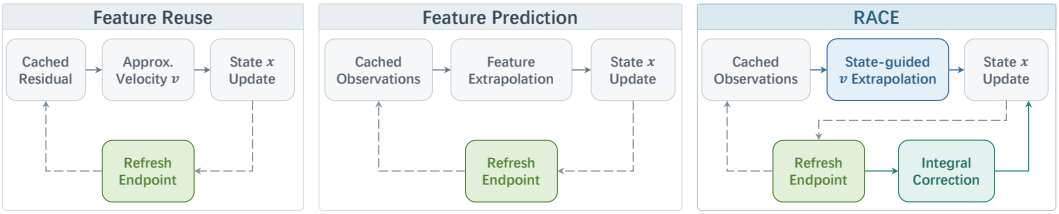


Figure 1: Comparison of cache utilization mechanisms. Feature reuse approximates outputs from cached residual, while feature prediction extrapolates features from cached observations. RACE uses the current state to guide velocity extrapolation and refresh endpoints to correct state deviations.

by the displacement of estimated velocity at preceding reuse steps. Mitigating the impact of accumulated velocity errors becomes a key task while rarely mentioned for caching acceleration. This motivates us to improve the accuracy of velocity estimation in future reuse steps and simultaneously correct the accumulated historical errors.

Our main idea is that **refresh steps characterize local distributions that can be used to model the state-velocity relations and velocity displacement**. We observe that the updates of latent state and velocity exhibit a linear relation over a local interval of the generation trajectory. Using the state and velocity of refresh steps as endpoints, the velocity at subsequent reuse steps is extrapolated under the constraint of linear relation. Similarly, the velocity displacement at reuse steps can be linearly interpolated within two endpoints of refresh steps. The displacement of velocity within two refresh steps is compensated to the state after the refresh step, mitigating the impact of estimation errors.

Motivated by the above insights, we propose **Refresh-Anchored Cache (RACE)**, which combines state-guided velocity extrapolation with state correction. As shown in Fig. 1, RACE uses previous refresh endpoints and the current state to guide velocity extrapolation, and corrects the state after each refresh step using the integrated velocity error over the reuse interval that has just gone through.

Our main contributions are as follows: (1) We analyze the linear relation of state-velocity updates and emphasize the importance of state information for velocity estimation. Multiple steps' information are involved in the relation regression and thus improve the velocity estimation. (2) We study the impact of accumulated displacements in historical velocity estimation, and propose a correction strategy to adjust the displacement in state update. This correction strategy complements the linear relation modeling, leading to our Refresh-Anchored Cache (RACE) without additional network computation or hyper-parameters tuning. (3) RACE achieves state-of-the-art performance on FLUX (Black Forest Labs, 2024), Wan (Wan et al., 2025), HunyuanVideo (Kong et al., 2024), and MiniMax-H3 (MiniMax, 2026), demonstrating its effectiveness for image and video generation.

2 Related Work

Diffusion Model Inference Acceleration. Diffusion inference acceleration mainly includes sampling-step reduction (Zhao et al., 2023), low-precision computation (Shang et al., 2023), and cross-step computation reuse (Selvaraju et al., 2024). For sampling-step reduction, efficient solvers such as DPM-Solver (Lu et al., 2022; 2025) reduce neural function evaluations through improved numerical integration, while Progressive Distillation (Salimans & Ho, 2022; Frans et al., 2025) transfers a multi-step teacher trajectory to a few-step student model. These approaches substantially improve sampling efficiency, although reducing the number of denoising steps may introduce a trade-off between acceleration and generation fidelity. Quantization methods, such as Q-Diffusion (Li et al., 2023; Chen et al., 2025), instead reduce the computational cost of individual denoising steps using low-precision weights and activations, with their practical efficiency depending partly on hardware support for low-precision operators. Orthogonal to these approaches, cache-based methods exploit redundancy across adjacent denoising steps by reusing or predicting intermediate computations. In this work, we focus on this cache-based acceleration paradigm.

Cache-Based Diffusion Acceleration. Cache-based methods exploit cross-step redundancy in diffusion models by reusing intermediate features across neighboring denoising steps (Zou et al., 2025; Li et al., 2024; Chen et al., 2024). Early methods such as DeepCache (Ma et al., 2024)

reuse high-level U-Net features, while later approaches, including TeaCache (Liu et al., 2025a) and SeaCache (Chung et al., 2026), adapt refresh schedules according to temporal or spectral feature variation. Recent approaches improve skipped-computation approximations through feature prediction (Chen et al., 2026): TaylorSeer (Liu et al., 2025b) uses Taylor expansion, DiCache (Bu et al., 2026) employs lightweight online probes for adaptive scheduling and cache-trajectory alignment, and FoCa (Zheng et al., 2026) combines multi-step prediction with Heun-style calibration. Although these methods improve the accuracy–efficiency trade-off, they primarily focus on refresh scheduling or per-step approximation rather than the latent-state deviations that accumulate during recursive sampling. In contrast, RACE uses state-conditioned velocity extrapolation for reuse-step prediction and refresh-endpoint velocity residuals to estimate and correct accumulated state errors, enabling trajectory-aware correction beyond local approximation.

3 PRELIMINARIES

Flow matching and sampling. Given a data latent $x_1 \sim p_{data}$ and independent Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$, the linear interpolation path in flow matching is

$$x_t = (1-t)\epsilon + tx_1, \qquad t \in [0,1]. \tag{1}$$

Thus, $t = 0$ corresponds to pure noise, and $t = 1$ corresponds to clean data. The target velocity field along this path is $u_t = dx_t/dt = x_1 - \epsilon$. A neural network $v_\theta(x_t, t)$ is trained to fit this target velocity field using the flow matching objective

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t,\epsilon,x_1}\left[\|v_\theta(x_t,t) - u_t\|_2^2\right]. \tag{2}$$

Given an increasing time grid $\mathcal{T} = \{0 = t_0 < \cdots < t_N = 1\}$, let $\Delta t_i = t_{i+1} - t_i > 0$ denotes the step size from t_i to t_{i+1}, where $i \in \{0, \ldots, N-1\}$. The sampling process can be written as

$$x_{t_{i+1}} = x_{t_i} + \Delta t_i \, v_{t_i}, \qquad v_{t_i} := v_\theta(x_{t_i}, t_i). \tag{3}$$

Cache refresh and reuse. Cache scheduling determines when to use the full network computation and when to approximate its output using cached observations (Liu et al., 2025a; Chung et al., 2026). In particular, the full network computation steps are referred to as *refresh steps*, and the intermediate steps that use approximations are *reuse steps*. In the residual-reuse framework, Transformer blocks $\mathcal{F}$ take the hidden features z of state x as input, and $\mathcal{P}$ projects the output of Transformer blocks into velocity. At refresh step a, the residual $r_a = \mathcal{F}(z_a) - z_a$ is calculated and cached. At the subsequent reuse step i, the output of Transformer blocks is approximated as the sum of the cached residual r_a and hidden features z_i, where z_i is simply the embedding of states x_i. The approximated output is passed through the projection head $\mathcal{P}$ to obtain the velocity

$$\hat{v}_i^{\mathrm{r}} = \mathcal{P}(z_i + r_a). \tag{4}$$

This framework assumes that the feature residual r_a is consistent among certain reuse steps, and thus accelerates the prediction of velocity by approximating the output of Transformer blocks with the fixed feature residual and the current hidden features.

4 METHOD: RACE

Our RACE framework studies the task of state-guided velocity estimation and state correction in reuse steps. As illustrated in Fig. 2, we explore the relations of state-velocity pairs in local time intervals and propose the *State-Guided Velocity Extrapolation* (SGVE) to estimate velocity $\hat{v}_i$ based on information of preceding refresh steps a and b (Sec. 4.1). To mitigate the historical errors of velocity estimation, we adjust the state of the first reuse step after each refresh step by *Refresh-Anchored Integral Correction* (RAIC) strategy (Sec. 4.2). Overall, the refresh step acts as the anchor for future velocity estimation and a basis for past state correction.

4.1 STATE-GUIDED VELOCITY EXTRAPOLATION

The residual-reuse framework estimates the output of Transformer blocks under the assumption of fixed feature residual among local time intervals. However, the estimated velocities at the reuse steps

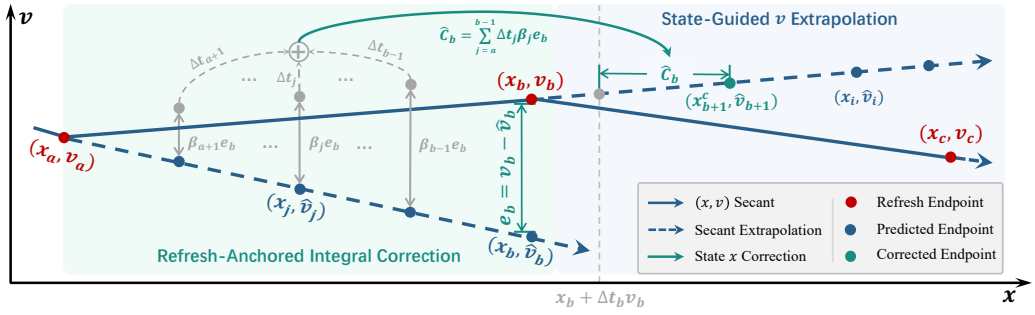


Figure 2: **Overview of RACE**. Left: At refresh step b, the velocity v_b is obtained from the full network computation. The discrepancy of v_b and the approximated velocity $\hat{v}_b$ is defined as the velocity error e_b. Thus, the intermediate errors $e_j \approx \beta_j e_b$, where $j \in [a, b)$, can be estimated to adjust the state deviation. Right: the observed state–velocity pairs at refresh steps a and b define a secant in (x, v) space, along which $\hat{v}_i$ is extrapolated using the current state x_i.

gradually diverge from the trajectory of full network computations, as illustrated in the top row of Fig. 3. To analyze this observation, let $s_i = \mathcal{F}_i(z_i)$ denotes the output features after full computation for reuse step $i \in (b, c)$, and we can get the "real" residual $r_i = s_i - z_i$. When accelerating, the residual-reuse approach approximates this output feature as $\tilde{s}_i = z_i + r_b$, which introduces an error $\Delta s_i = r_b - r_i$ before being input into the output head $\mathcal{P}$. Therefore, the local first-order expansion of the output head gives

$$\hat{v}_i^{\mathrm{r}} - v_i = \mathcal{P}(\hat{s}_i) - \mathcal{P}(s_i) \approx J_{\mathcal{P}_i}(s_i)\,\Delta s_i, \tag{5}$$

where $J_{\mathcal{P}_i}$ is the Jacobian of the output head *w.r.t.* its hidden input features, and $\hat{v}_i^r$ is the approximated velocity. The robustness of output head to mismatched features may impact the magnitude of this error. This mismatch along high-gain directions of $J_{\mathcal{P}_i}$ may be amplified and affect the next state. See Appendix B for details.

We claim that the residual consistency is a loose constraint to caching acceleration for two reasons. First of all, the hidden features at refresh step are contaminated by the accumulated errors in the previous velocity estimation. Second, a single refresh step cannot provide enough information to identify a straight trajectory. In our method, we explore the relations of multiple endpoints in the generation trajectory for velocity estimation and reveal the linear relation of state-velocity pairs. In the top row of Fig. 3, we extract and present the first principal component of states and velocities along the generation trajectory (refer to Appendix E for more results). The outcome demonstrates that the secant of two adjacent refresh steps can near-linearly approximate the distribution of subsequent state-velocity points. We therefore propose to extrapolate the velocity according to the displacement of local states. Using the calculated state–velocity pairs at refresh steps a and b where $b > a$, we define the anchored state and velocity displacement as

$$\Delta x_{ba} = x_b - x_a, \qquad \Delta v_{ba} = v_b - v_a. \tag{6}$$

Motivated by the property of local linearity, the displacements of state x_i and velocity v_i at reuse step i to preceding refresh step b can be approximated by the same factor λ_i. For one-dimensional linear extrapolation, we have

$$\lambda_i = \frac{x_i - x_b}{\Delta x_{ba}} = \frac{v_i - v_b}{\Delta v_{ba}}.$$

At the high-dimensional manifold, the state-velocity pairs in the generation trajectory approximately follow the same linear relation. We therefore fit $x_i - x_b \approx \lambda_i \Delta x_{ba}$ to obtain λ_i by least squares:

$$\begin{aligned} \lambda_i &= \arg\min_{\lambda \in \mathbb{R}} \|(x_i - x_b) - \lambda \Delta x_{ba}\|_2^2 \\ &= \frac{\langle \Delta x_{ba}, x_i - x_b \rangle}{\|\Delta x_{ba}\|_2^2}. \end{aligned} \tag{7}$$

In this way, the estimated velocity at reuse step i is obtained via:

$$\begin{aligned} \hat{v}_i &= v_b + \lambda_i \Delta v_{ba} \\ &= v_b + \frac{\langle \Delta x_{ba}, x_i - x_b \rangle}{\|\Delta x_{ba}\|_2^2} \Delta v_{ba}. \end{aligned} \tag{8}$$

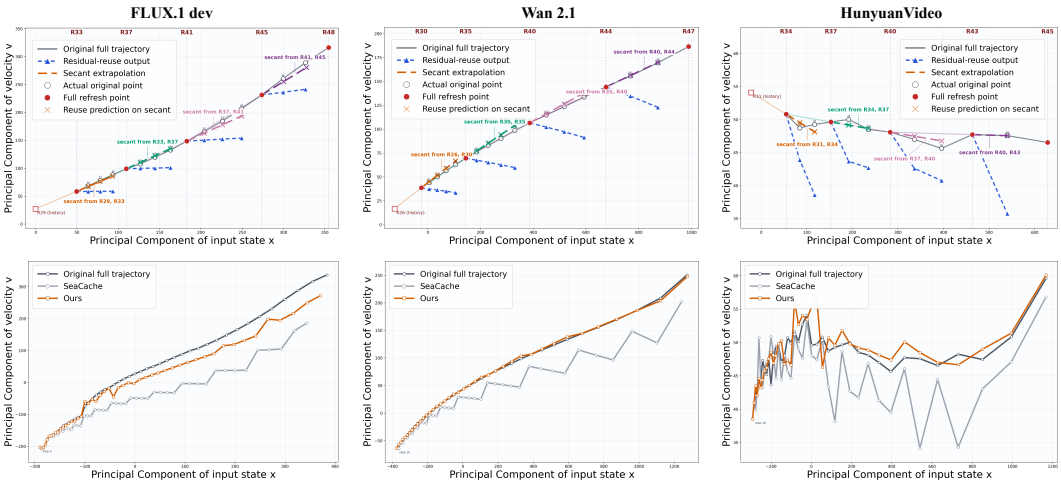


Figure 3: Local state–velocity relationships and trajectory comparisons. Top: the velocity outputs of residual reuse gradually deviate from full computation within reuse intervals, whereas secants constructed from previous refresh points better track subsequent velocity changes. Bottom: sampling trajectories of full computation, SeaCache with residual reuse, and RACE.

Under the same sampling and refresh paradigm, TaylorSeer (Liu et al., 2025b) uses time intervals shared among all entries to extrapolate velocities. In our framework, λ_i is determined by the current state of each entry, allowing prediction at reuse step adapting to the sample-specific local trajectories without any network calculation and cumbersome hyper-parameter tuning. In implementation, when only one previous refresh point a is available, the predictor returns v_a for subsequent reuse steps.

4.2 Refresh-Anchored Integral Correction

As caching acceleration inevitably incurs approximation error, the input state at a refresh step is disrupted by the displacement from preceding reuse steps. Therefore, we propose to estimate the integral compensation at refresh step to mitigate the state error accumulated over the preceding reuse interval. This compensation is then used to adjust the state update at refresh step.

Specifically, we correct for the state error accumulated over (a, b) at refresh step b. Let $\hat{v}_j$ denote the estimated velocity at reuse step j where $j \in (a, b)$, and $e_j = v_\theta(x_j, t_j) - \hat{v}_j$ is the velocity error, where $\hat{v}_j$ is approximately estimated. Therefore, the state displacement at step j is $\Delta t_j e_j$, and these displacements accumulate through subsequent updates.

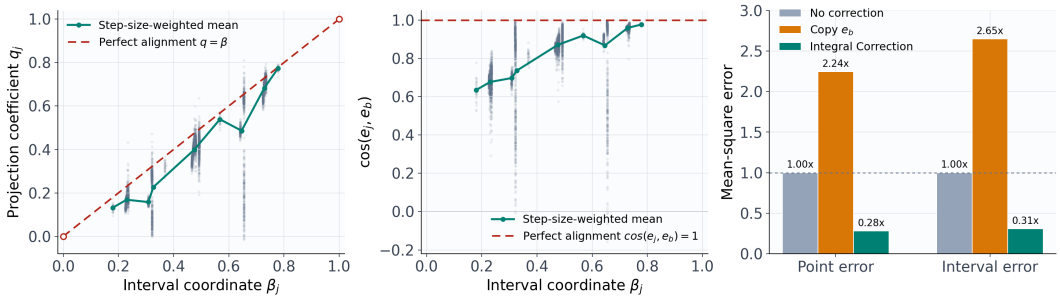
As illustrated on the left of Fig. 2, let x_j^c denote the state after incorporating the error-induced displacements within the current interval. The estimated secant is consistent with the full network computation at the starting point of local interval $[a, b]$, that is, $x_a^c = x_a$ and $e_a = 0$. Starting from $x_a^c = x_a$, the discrete sampling update in Eq. (3) accumulates these displacements on top of the original states:

$$\begin{aligned} x_{j+1}^c &= x_j^c + \Delta t_j\,(\hat{v}_j + e_j), \\ &= x_j^c + (x_{j+1} - x_j) + \Delta t_j\, e_j, \\ x_{j+1}^c - x_{j+1} &= (x_j^c - x_j) + \Delta t_j\, e_j, \qquad j = a, \ldots, b-1. \end{aligned} \tag{9}$$

Expanding this recurrence equation to refresh step b yields cumulative correction:

$$C_b := x_b^c - x_b = \sum_{j=a}^{b-1} \Delta t_j\, e_j \tag{10}$$

However, velocity errors within the interval cannot be directly observed during reuse steps. We design a strategy of error estimation under the constraints of two consecutive refresh steps. First, we calculate the velocity error of the refresh step $e_b = v_b - \hat{v}_b$, which indicates the deviation between



(a) Error magnitude (b) Error direction (c) Error remaining

Figure 4: The errors of velocity extrapolation within reuse intervals for FLUX.1-dev on DrawBench: (a) the projection coefficient of the current error e_j onto the direction of the refresh-endpoint error e_b versus interval coordinate β_j; (b) the direction between current error e_j and the refresh-endpoint error e_b; (c) mean-square errors for no correction, direct reuse of the refresh-endpoint error, and the Integral Correction, evaluated for both pointwise and interval accumulation.

full network computation and velocity extrapolation. Then, to estimate the velocity error within the interval using this refresh endpoint error, we define the normalized interval coordinate as

$$\beta_j = \frac{t_j - t_a}{t_b - t_a}, \qquad \beta_a = 0, \quad \beta_b = 1. \tag{11}$$

Fig. 4(a) shows that the projection coefficient $q_j = \langle e_j, e_b \rangle / \|e_b\|_2^2$ generally equals β_j. Fig. 4(b) shows directional alignment between interior errors e_j and refresh-endpoint error e_b. Therefore, we approximate $e_j \approx \beta_j e_b$ by scaling the refresh-endpoint error e_b with the normalized interval coordinate β_j, which captures the growth of errors within the reuse interval. Substituting this approximation into Eq. (10) gives

$$C_b \approx \hat{C}_b = \left(\sum_{j=a}^{b-1} \Delta t_j \beta_j \right) e_b. \tag{12}$$

The integral covers only steps $a, \ldots, b-1$ since refresh step b already uses the velocity v_b from a full network evaluation. Fig. 4(c) presents the effectiveness of our proposed error estimation. Specifically, the mean-square error of the integral correction from this linear endpoint error approximation is 0.31 times that of the no-correction baseline, while directly reusing the refresh-endpoint error yields 2.65 times the baseline error. Appendix G provides more details about error analysis.

Following Eq. (3) and Eq. (10), we add $\hat{C}_b$ to the original refresh-step update:

$$x_{b+1}^c = x_b^c + \Delta t_b v_b \approx x_b + \Delta t_b v_b + \hat{C}_b. \tag{13}$$

Importantly, v_b is evaluated from x_b before the correction. After e_b is observed, the refresh step b acts as an anchor for displacement correction. Therefore, this strategy is named Refresh-Anchored Integral Correction (RAIC). The bottom row of Fig. 3 shows that, for the illustrated trajectories, SeaCache with residual reuse exhibits noticeable accumulated deviations, while RACE armed with SGVE and RAIC, generates better estimated result closer to the full-computation trajectory.

5 Experiments

5.1 Experimental Setup

Models. We evaluate RACE on various generation models. For text-to-image generation, we use FLUX.1-dev(Black Forest Labs, 2024) at a resolution of 1024×1024. For text-to-video generation, we evaluate Wan 2.1(Wan et al., 2025) with 65 frames at 480×832, HunyuanVideo(Kong et al., 2024) with 65 frames at 480×640, and MiniMax-H3(MiniMax, 2026) with 124 frames at 960×544.

Table 1: Quantitative comparison on FLUX.1-dev and MiniMax-H3.

Method	FLUX.1-dev Param.	Latency (s)	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑	MiniMax-H3 Param.	Latency (s)	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
Original	–	7.8	1861	–	–	–	–	132.5	31294	–	–	–
25 steps	–	3.9	930	17.54	0.302	0.740	–	81.0	16215	17.78	0.340	0.643
15 steps	–	2.4	558	15.36	0.414	0.663	–	61.3	9925	15.70	0.435	0.583
TeaCache	$\delta = 0.3$	4.2	969	21.18	0.184	0.819	$\delta = 0.23$	68.9	15123	18.66	0.302	0.667
TaylorSeer	$\mathcal{S} = 3$	3.7	831	24.65	0.126	0.875	$\mathcal{S} = 2$	**73.3**	16966	18.90	0.291	0.650
SenCache	$\varepsilon = 0.26$	3.8	807	27.49	0.079	0.913	$\varepsilon = 0.37$	74.6	15436	23.47	0.157	0.808
DiCache	$\delta = 0.1$	3.8	866	27.62	0.059	0.932	$\delta = 0.07$	74.4	15484	22.28	0.190	0.761
SeaCache	$\delta = 0.3$	**3.6**	**777**	27.03	0.086	0.906	$\delta = 0.28$	74.9	15584	23.34	0.173	0.782
RACE	$\delta = 0.3$	**3.6**	**777**	**30.26**	**0.051**	**0.938**	$\delta = 0.28$	73.4	**15329**	**25.16**	**0.129**	**0.820**
TeaCache	$\delta = 0.6$	2.5	559	17.24	0.339	0.701	$\delta = 0.42$	56.2	10177	17.34	0.357	0.630
TaylorSeer	$\mathcal{S} = 5$	2.7	521	19.85	0.222	0.777	$\mathcal{S} = 4$	56.1	10177	13.95	0.584	0.509
SenCache	$\varepsilon = 0.88$	2.5	570	19.53	0.251	0.766	$\varepsilon = 2.20$	55.9	10176	17.74	0.336	0.640
DiCache	$\delta = 0.4$	2.5	572	21.58	0.167	0.827	$\delta = 0.31$	53.2	**8942**	17.87	0.314	0.655
SeaCache	$\delta = 0.6$	**2.4**	**484**	21.26	0.213	0.807	$\delta = 0.55$	53.1	9392	17.69	0.348	0.642
RACE	$\delta = 0.6$	**2.4**	**484**	**22.92**	**0.150**	**0.848**	$\delta = 0.55$	**52.1**	9281	**19.72**	**0.254**	**0.695**

Baselines. We compare RACE with representative cache-based acceleration methods, including TeaCache (Liu et al., 2025a), SeaCache (Chung et al., 2026), SenCache (Haghighi & Alahi, 2026), TaylorSeer (Liu et al., 2025b), and DiCache (Bu et al., 2026). For a fair comparison, when a baseline does not provide hyper-parameters for the target budgets, we perform grid search.

Evaluation protocol and metrics. We perform inference on DrawBench (Saharia et al., 2022) for image generation and Vbench (Huang et al., 2024) for video generation under different computation budgets ($\approx 50\%$ and $\approx 30\%$). We use identical prompts and seeds across methods on a single NVIDIA B200 GPU. PSNR, LPIPS and SSIM are computed with references from the original model. Efficiency is evaluated using TFLOPs and latency in seconds without text encoder and VAE.

5.2 Quantitative Results

Image Generation. Tab. 1 (left) shows that RACE achieves the best reference fidelity on FLUX.1-dev at both budgets. Compared with SeaCache, RACE improves all three fidelity metrics at the same computational cost and latency, highlighting the benefit of combining state-guided prediction with accumulated-error correction.

Video Generation. On MiniMax-H3 (Tab. 1, right), RACE improves all metrics over SeaCache (Chung et al., 2026) at both budgets. On Wan 2.1 and HunyuanVideo (Tab. 2), RACE outperforms the reported baselines across all three fidelity metrics in both settings. For both models, these gains over SeaCache come at the same computational cost and nearly unchanged latency. Together, the results support RACE's applicability across different video resolutions and sequence lengths.



Figure 5: Qualitative comparisons of various methods on FLUX.1-dev.

Table 2: Quantitative comparison on Wan 2.1 and HunyuanVideo.

Method	Wan 2.1 1.3B Param.	Latency (s)	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑	HunyuanVideo Param.	Latency (s)	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
Original	–	54.5	9934	–	–	–	–	78.9	14703	–	–	–
25 steps	–	27.8	4967	14.78	0.379	0.583	–	42.1	7351	20.12	0.255	0.717
15 steps	–	17.2	2980	13.28	0.476	0.511	–	27.4	4411	17.25	0.375	0.633
TeaCache	$\delta=0.09$	31.5	4967	23.53	0.112	0.839	$\delta=0.12$	43.1	7317	23.55	0.166	0.808
TaylorSeer	$\mathcal{S}=2$	32.6	5166	18.38	0.241	0.712	$\mathcal{S}=2$	46.9	7645	24.93	0.137	0.836
SenCache	$\varepsilon=0.01$	31.7	4959	29.62	0.048	0.936	$\varepsilon=0.28$	41.0	7272	30.19	0.067	0.907
DiCache	$\delta=0.10$	28.2	4808	27.23	0.056	0.905	$\delta=0.065$	40.8	7221	32.40	0.047	0.930
SeaCache	$\delta=0.20$	**27.2**	**4769**	29.38	0.043	0.931	$\delta=0.26$	40.9	7060	32.44	0.048	0.930
RACE	$\delta=0.20$	27.3	**4769**	**31.15**	**0.022**	**0.951**	$\delta=0.26$	**40.7**	**7060**	**35.71**	**0.027**	**0.952**
TeaCache	$\delta=0.20$	22.0	3581	19.16	0.244	0.720	$\delta=0.16$	35.1	5986	20.74	0.245	0.739
TaylorSeer	$\mathcal{S}=4$	21.8	3580	14.20	0.440	0.561	$\mathcal{S}=3$	35.7	**5293**	21.01	0.225	0.740
SenCache	$\varepsilon=0.10$	23.7	**3167**	21.66	0.194	0.782	$\varepsilon=0.45$	35.1	5986	27.52	0.104	0.871
DiCache	$\delta=0.20$	23.0	3203	23.63	0.101	0.846	$\delta=0.13$	35.2	6029	27.84	0.097	0.863
SeaCache	$\delta=0.35$	**19.8**	3378	23.53	0.119	0.828	$\delta=0.35$	35.1	5884	28.16	0.091	0.879
RACE	$\delta=0.35$	**19.8**	3378	**24.40**	**0.088**	**0.854**	$\delta=0.35$	**35.0**	5884	**31.18**	**0.060**	**0.912**

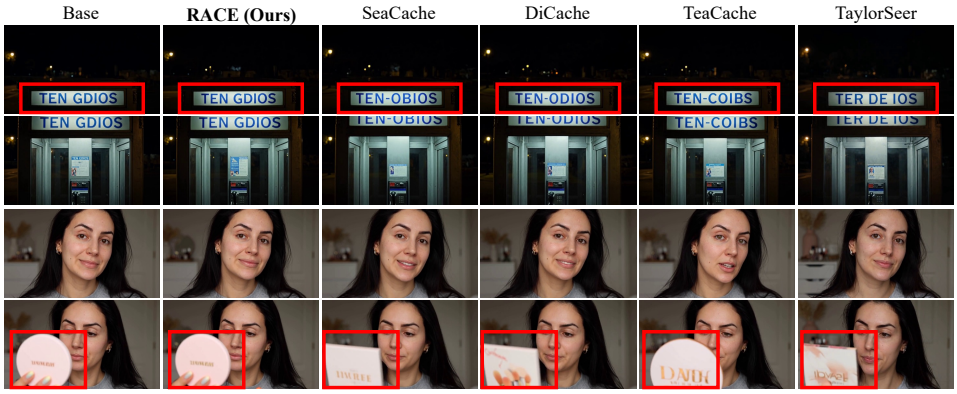


Figure 6: Qualitative comparisons of various methods on MiniMax-H3.

5.3 Qualitative Results

On FLUX.1-dev (Fig. 5), RACE better retains details, while competing methods exhibit local artifacts. MiniMax-H3 frames in Fig. 6 show that RACE better preserves lettering and object structure, while other methods alter text or introduce geometric distortions. Together, these examples complement the quantitative results, demonstrating closer agreement with the full-computation reference across image and video generation; see Appendix L for additional visualizations.

5.4 Ablation Studies

We evaluate RACE on DrawBench with FLUX.1-dev, using 50 sampling steps, 1024×1024 resolution, BF16 precision, and a guidance scale of 3.5.

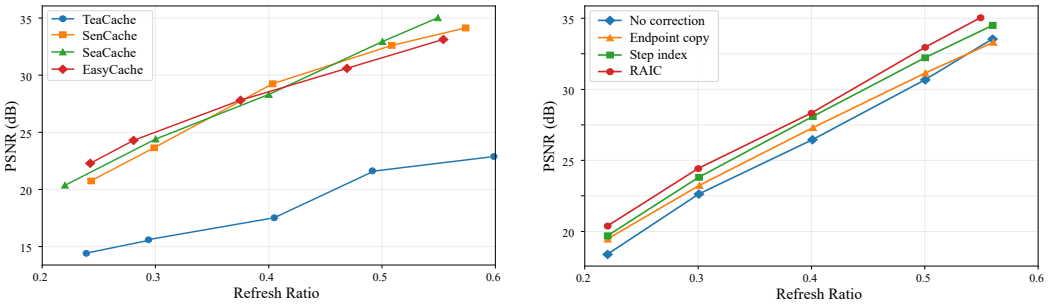
Table 3: Adaptable to different cache schedulers.

Scheduler	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
TeaCache	914	21.87	0.176	0.824
EasyCache	873	30.61	0.039	0.951
SenCache	946	32.62	0.031	0.959
SeaCache	931	**32.96**	**0.029**	**0.962**

Table 4: Effect of different approximations.

Variant	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
Residual reuse	931	31.85	0.036	0.954
TaylorSeer	931	31.69	0.031	0.912
DCTA	941	32.28	0.030	0.960
SGVE	931	**32.96**	**0.029**	**0.962**

Compatibility with cache scheduling. We combine RACE with TeaCache (Liu et al., 2025a), EasyCache (Zhou et al., 2025), SenCache (Haghighi & Alahi, 2026), and SeaCache (Chung et al.,



(a) Plug different cache schedulers into RACE.

(b) Effect of the correction coordinate.

Figure 7: PSNR-refresh ratio trade-off of various cache schedulers and correction coordinates.

2026). At comparable compute, SeaCache achieves the best fidelity at the operating points in Tab. 3, while EasyCache and SenCache remain competitive at lower refresh ratios (Fig. 7(a)). Therefore, we use SeaCache as the default cache scheduler.

SGVE versus alternative approximations. With RAIC fixed, we compare SGVE with residual reuse (Liu et al., 2025a), TaylorSeer's (Liu et al., 2025b) first-order rule adapted to final velocities, and DiCache's (Bu et al., 2026) Dynamic Cache Trajectory Alignment (DCTA). As demonstrated in Tab. 4, SGVE achieves the best fidelity across all three metrics, improving PSNR by 1.27 dB over the adapted TaylorSeer and 0.68 dB over DCTA. These results support conditioning the extrapolation scale on each sample's evolving state, rather than determining it solely from sampling-time intervals. SGVE obtains the guidance directly from the sampler state without additional network computation.

Effect of the correction coordinate. With SGVE fixed, we compare no correction, endpoint copy ($\widehat{e}_j = e_b$), step-index scaling ($\widehat{e}_j = \frac{j-a}{b-a} e_b$), and RAIC in Tab. 5. Endpoint copy and step-index scaling improve over no correction, but RAIC further improves PSNR by 1.03 dB over step-index scaling. This gain supports modeling residual growth according to progress in sampling time, rather than assuming equal growth per sampling step. Fig. 7(b) compares these correction variants across refresh ratios. On nonuniform grids, Eq. (11) measures progress in integration time rather than step count, aligning the residual model with the state displacement being corrected.

Table 5: Effect of the correction coordinate.

Variant	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
No correction	931	31.38	0.036	0.952
Endpoint copy	931	31.85	0.035	0.955
Step index	931	31.93	0.031	0.958
RAIC	931	**32.96**	**0.029**	**0.962**

Table 6: Components ablation.

SGVE	RAIC	TFLOPs	PSNR ↑	LPIPS ↓	SSIM ↑
×	×	931	29.63	0.055	0.935
✓	×	931	31.38	0.036	0.952
×	✓	931	31.85	0.036	0.954
✓	✓	931	**32.96**	**0.029**	**0.962**

Effect of SGVE and RAIC. As shown in Tab. 6, SGVE and RAIC individually improve PSNR by 1.75 and 2.22 dB over residual reuse at the same compute. RAIC remains beneficial with SGVE enabled, adding another 1.58 dB and further improving LPIPS and SSIM. This persistent gain supports the distinction between predicting the current velocity and compensating errors already integrated into the state: more accurate extrapolation does not make accumulated-error correction redundant.

6 Conclusion

We presented **RACE** (Refresh-Anchored Cache), a training-free framework for cache-based diffusion acceleration that jointly addresses velocity prediction and accumulated state errors. SGVE uses the evolving state to guide velocity prediction, while RAIC uses refresh-endpoint velocity errors to estimate and compensate for accumulated state deviations. Each refresh endpoint thus serves as dual role: supporting velocity prediction and correction of errors from the preceding interval. RACE integrates with different cache scheduling strategies without introducing additional networks or tunable hyper-parameters. Extensive experiments present the importance of state-guided velocity approximation and integral error correction on the cache-based acceleration.

AI USE STATEMENT

Generative AI tools were used for language polishing and to assist with the development of simple utility code and tests. All AI-assisted text was carefully reread by the authors, and all AI-assisted code was reviewed before being incorporated into the manuscript or Git repository. Such outputs were used only after the authors had verified them and identified no issues. The research design, interpretation of results, and all mathematical reasoning and proofs were carried out by the authors without assistance from generative AI. The authors take full responsibility for the final manuscript, code, claims, and results.

ETHICS STATEMENT

Our work focuses on training-free inference acceleration and does not involve collecting new datasets or conducting human-subject experiments. RACE does not modify the underlying generative model or its training data, and therefore generally inherits the capabilities, limitations, and potential risks of the corresponding base models.

REPRODUCIBILITY STATEMENT

We provide the formulation and implementation details of RACE, including the complete pseudocode, together with the corresponding error analysis and online accumulation procedure. We report the evaluated models, resolutions, sampling settings, cache schedulers, computational budgets, hardware, evaluation datasets, and metrics. We will release our implementation to facilitate reproduction of the reported results and further research.

REFERENCES

Black Forest Labs. FLUX. Official model and inference-code release, 2024. URL `https://github.com/black-forest-labs/flux`.

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

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.

Guantao Chen, Shikang Zheng, Yuqi Lin, and Linfeng Zhang. Forecast the principal, stabilize the residual: Subspace-aware feature caching for diffusion transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23632–23641, June 2026.

Lei Chen, Yuan Meng, Chen Tang, Xinzhu Ma, Jingyan Jiang, Xin Wang, Zhi Wang, and Wenwu Zhu. Q-dit: Accurate post-training quantization for diffusion transformers. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 28306–28315. IEEE, 2025.

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.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16302–16312, October 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14283–14294, June 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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12606–12633, 2024.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations*, volume 2025, pp. 34668–34684, 2025.

Yasaman Haghighi and Alexandre Alahi. SenCache: Accelerating diffusion model inference via sensitivity-aware caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14295–14304, 2026.

Jonathan Ho, Ajay N. Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 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.

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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15240–15252, October 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.

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*, volume 37, 2024.

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

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

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7353–7363, June 2025a.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with TaylorSeers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15853–15863, October 2025b.

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*, volume 35, pp. 5775–5787, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 22 (4):730–751, 2025.

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. In *International Conference on Learning Representations*, volume 2025, pp. 33132–33156, 2025.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15762–15772, June 2024.

MiniMax. MiniMax H3: An open model breaking the boundaries between tasks and modalities. Official model release, July 2026. URL `https://github.com/MiniMax-AI/MiniMax-H3`.

Xurui Peng, Chenqian Yan, Hong Liu, Rui Ma, Fangmin Chen, Xing Wang, Zhihua Wu, Songwei Liu, and Mingbao Lin. ERTACache: Error rectification and timesteps adjustment for efficient diffusion. In *International Conference on Learning Representations*, 2026.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, June 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1972–1981, June 2023.

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

Haoyang Tong, Yu He, Fang Li, Lichen Ma, Jingling Fu, Dong Chen, Zhen Chen, Junshi Huang, and Jie Cao. Energy-guided flow matching. *arXiv preprint arXiv:2608.05811*, 2026.

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.

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.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. CogVideoX: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, 2025.

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

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. In *International Conference on Learning Representations*, volume 2025, pp. 3296–3319, 2025.

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*, volume 40, pp. 13449–13457, 2026.

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. In *International Conference on Learning Representations*, 2025.

Supplementary Material

A Pseudocode

Algorithm 1 RACE: Refresh-Anchored Cache

Require: Velocity model v_θ, time grid $0 = t_0 < \cdots < t_N = 1$, cache scheduler $\mathcal{S}$
Ensure: Generated latent x_N

$x_0 \sim \mathcal{N}(0, I)$; $\mathcal{H} \leftarrow \emptyset$; $a \leftarrow 0$
for $i = 0, \ldots, N-1$ **do**
  $\Delta t_i \leftarrow t_{i+1} - t_i$
  if $i \in \{0, N-1\}$ or $\mathcal{S}$ requests refresh at (x_i, t_i) **then**
    $v_i \leftarrow v_\theta(x_i, t_i)$; $\widehat{C}_i \leftarrow 0$
    if $i > a + 1$ **then** ▷ RAIC: correct the preceding reuse interval
      $\widehat{v}_i \leftarrow \mathrm{SGVE}(x_i, \mathcal{H})$ ▷ Use the previous predictor
      $e_i \leftarrow v_i - \widehat{v}_i$
      $\beta_j \leftarrow (t_j - t_a)/(t_i - t_a)$ for $j = a, \ldots, i-1$
      $\widehat{C}_i \leftarrow \left(\sum_{j=a}^{i-1} \Delta t_j \beta_j\right) e_i$
    end if
    Append (i, x_i, v_i) to $\mathcal{H}$, retaining the latest two observations
    $a \leftarrow i$
    $x_{i+1} \leftarrow x_i + \Delta t_i v_i + \widehat{C}_i$
  else
    $\widehat{v}_i \leftarrow \mathrm{SGVE}(x_i, \mathcal{H})$ ▷ Predict the reuse velocity
    $x_{i+1} \leftarrow x_i + \Delta t_i \widehat{v}_i$
  end if
end for
return x_N

function $\mathrm{SGVE}(x_i, \mathcal{H})$ ▷ State-Guided Velocity Extrapolation
  Let (b, x_b, v_b) be the latest observation in $\mathcal{H}$
  if $|\mathcal{H}| = 1$ **then**
    return v_b
  end if
  Let (a, x_a, v_a) be the preceding observation in $\mathcal{H}$
  $\Delta x_b \leftarrow x_b - x_a$; $\Delta v_b \leftarrow v_b - v_a$
  $\lambda_i \leftarrow \langle \Delta x_b, x_i - x_b \rangle / \|\Delta x_b\|_2^2$ ▷ Use 0 if $\Delta x_b = 0$
  return $v_b + \lambda_i \Delta v_b$
end function

B Error Propagation in Residual Reuse

We follow the notation in the main text, with $x_i := x_{t_i}$ and $v_i := v_\theta(x_i, t_i)$; the indices i, j, a, b denote discrete sampling steps. We treat the state, hidden features, and velocity tensors of each sample as flattened vectors, and use $\|\cdot\|_2$ for the Euclidean norm. The state updates and error integrals below use the discrete sampling process in Eq. (3), with $\Delta t_i = t_{i+1} - t_i > 0$.

For a fixed state x_i and time condition t_i, the inputs to the output head under full computation and residual reuse are, respectively,

$$s_i = z_i + r_i, \qquad \tilde{s}_i = z_i + r_b = s_i + \Delta s_i, \qquad \Delta s_i = r_b - r_i. \tag{14}$$

The corresponding velocity difference is

$$\hat{v}_i^{\mathrm{r}} - v_i = \mathcal{P}_i(s_i + \Delta s_i) - \mathcal{P}_i(s_i). \tag{15}$$

If the output head $\mathcal{P}_i$ is continuously differentiable in a neighborhood of the line segment joining s_i and $s_i + \Delta s_i$, the fundamental theorem of calculus gives

$$\begin{aligned} \hat{v}_i^{\mathrm{r}} - v_i &= J_{\mathcal{P}_i}(s_i)\Delta s_i + R_i, \\ R_i &= \int_0^1 \left[J_{\mathcal{P}_i}(s_i + \tau \Delta s_i) - J_{\mathcal{P}_i}(s_i)\right] \Delta s_i \, d\tau. \end{aligned} \tag{16}$$

If the Jacobian also satisfies the Lipschitz condition $\|J_{\mathcal{P}_i}(y) - J_{\mathcal{P}_i}(z)\|_{\mathrm{op}} \le L_i\|y - z\|_2$ in this neighborhood, where $\|\cdot\|_{\mathrm{op}}$ is the operator norm induced by the Euclidean norm, then

$$\begin{aligned} \|R_i\|_2 &\le \int_0^1 L_i \tau \|\Delta s_i\|_2^2 \, d\tau = \frac{L_i}{2}\|\Delta s_i\|_2^2, \\ \|\hat{v}_i^{\mathrm{r}} - v_i\|_2 &\le \|J_{\mathcal{P}_i}(s_i)\Delta s_i\|_2 + \frac{L_i}{2}\|\Delta s_i\|_2^2. \end{aligned} \tag{17}$$

This bounds the remainder of the first-order approximation in Eq. (5) under the stated assumptions. The first-order velocity error is determined by $J_{\mathcal{P}_i}(s_i)\Delta s_i$: feature mismatch along high-gain directions can be amplified, and output normalization does not guarantee its removal. Starting from the same state, one Euler update with the two velocities produces a state difference of $\Delta t_i(\hat{v}_i^{\mathrm{r}} - v_i)$, which changes subsequent network inputs.

Relation to the training feature distribution. During standard training, the output head receives hidden features computed by the full network from training inputs and their timestep conditioning. Residual reuse combines the current input features z_i with a previous residual r_b, potentially changing the feature distribution presented to the head at a given timestep. The training objective does not explicitly require reused features and their full-computation counterparts to yield the same velocity, so it provides no guarantee that the head eliminates the velocity error introduced by reuse.

C Comparison with Time-guided Extrapolation

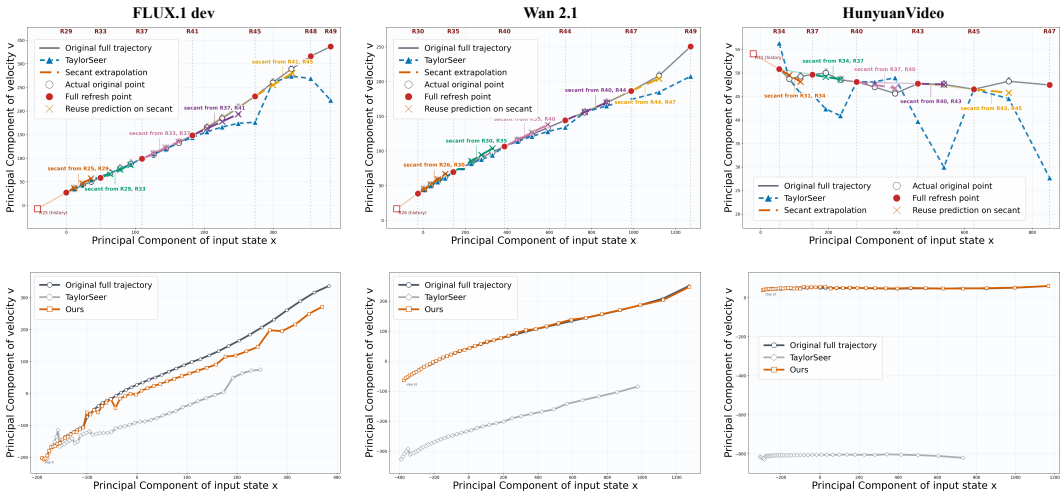


Figure 8: Comparison with TaylorSeer in PCA-projected state–velocity space on FLUX.1-dev, Wan2.1, and HunyuanVideo. Top: velocity predictions from TaylorSeer and SGVE relative to full computation. Bottom: sampling trajectories of full computation, TaylorSeer, and RACE.

TaylorSeer (Liu et al., 2025b) extrapolates cached features using a Taylor expansion in the sampling-step interval since the latest refresh. SGVE extrapolates the final velocity with a coefficient determined by the current state displacement (Eq. (7)), allowing its extrapolation scale to vary across samples even under identical sampling and refresh grids. In the top row of Fig. 8, SGVE follows the local full-computation trend more closely in the displayed intervals, while TaylorSeer shows larger deviations, particularly on HunyuanVideo.

In the bottom row, RACE remains closer to the full-computation reference, with particularly close alignment on Wan2.1 and HunyuanVideo. The FLUX.1-dev example still shows a visible gap, indicating that RACE reduces rather than eliminates trajectory deviations. The quantitative comparison of velocity approximation rules on FLUX.1-dev is reported in Tab. 4.

D More Ablation

We evaluate all variants on the 200 DrawBench prompts with FLUX.1-dev, using 50 sampling steps, 1024×1024 resolution, BF16 precision, and matching random seeds across variants. With the SeaCache threshold set to 0.225, we replace only SGVE's two-point secant with a quadratic least-squares fit to the latest 4, 6, or 8 refresh observations, constrained to pass through the latest observation. All other components, including RAIC, remain unchanged.

Table 7: State-guided multi-point extrapolation on FLUX.1-dev and DrawBench.

# Refresh points	Latency (s) ↓	PSNR ↑	LPIPS ↓	SSIM ↑
2 (SGVE)	**4.145**	**32.962**	**0.02890**	**0.96226**
4	4.155	32.220	0.03323	0.95737
6	4.154	31.892	0.03420	0.95676
8	4.157	31.548	0.03581	0.95515

In Tab. 7, the 4-, 6-, and 8-point variants reduce PSNR by 0.742, 1.070, and 1.414 dB, respectively, relative to two-point SGVE, with higher LPIPS and lower SSIM. Latency remains nearly unchanged at 4.145–4.157 s. Among these variants, more refresh points degrade all three fidelity metrics.

We explain that more observations span a longer history, over which the local state–velocity relationship can change. Fitting older observations may compromise the slope and curvature estimates near the current state, even when the curve passes through the latest observation. The resulting extrapolation errors affect subsequent sampler states, and endpoint correction does not guarantee recovery of the full-computation trajectory. These results favor the two-point secant for more accurate velocity extrapolation.

E Visualization of More Principal Components

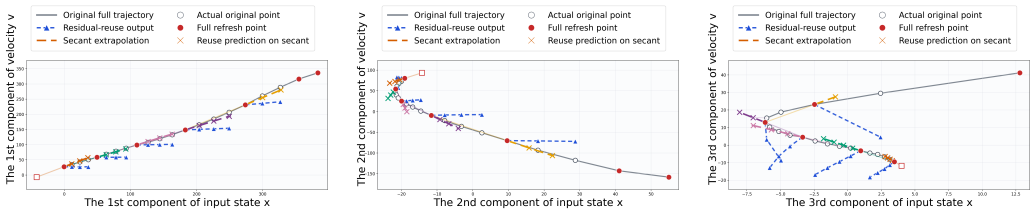


Figure 9: State–velocity relationships in the first (left), second (middle), and third (right) principal components, following the same comparison protocol used in the top row of Fig. 3.

PCA is fitted separately to states and velocities from the full-computation run. Residual-reuse outputs are evaluated at the refresh endpoints without changing the sampling trajectory. The second and third components reveal more curved state–velocity relationships than the first. Secant extrapolation generally remains close to the original full trajectory, whereas residual reuse shows larger deviations, particularly in the third component. These additional projections provide qualitative evidence that the observed local linearity extends beyond the first principal component.

F Secant Form of SGVE

Let $\Delta x_b = x_b - x_a$ and $\Delta v_b = v_b - v_a$. For $\Delta x_b \neq 0$, differentiating the least-squares objective in Eq. (7) gives

$$\frac{d}{d\lambda} \|(x_i - x_b) - \lambda \Delta x_b\|_2^2 = -2\langle \Delta x_b, x_i - x_b \rangle + 2\lambda \|\Delta x_b\|_2^2. \tag{18}$$

Setting the derivative to zero yields the unique least-squares solution

$$\lambda_i = \frac{\langle \Delta x_b, x_i - x_b \rangle}{\|\Delta x_b\|_2^2}. \tag{19}$$

Substituting this coefficient into Eq. (8) and writing the inner product as a matrix product gives

$$\begin{aligned} \hat{v}_i - v_b &= \Delta v_b \frac{\Delta x_b^\top (x_i - x_b)}{\|\Delta x_b\|_2^2} \\ &= \left(\frac{\Delta v_b \Delta x_b^\top}{\|\Delta x_b\|_2^2} \right) (x_i - x_b). \end{aligned} \tag{20}$$

Thus, for fixed refresh points a, b, the predicted velocity increment is a linear map of the current state displacement. Writing its matrix as B_b, we obtain

$$B_b := \frac{\Delta v_b \Delta x_b^\top}{\|\Delta x_b\|_2^2}, \qquad \hat{v}_i = v_b + B_b(x_i - x_b). \tag{21}$$

This is an equivalent matrix form of the existing extrapolation rule and requires no additional fitting. For D-dimensional states and velocities, $\Delta v_b \Delta x_b^\top$ is a $D \times D$ outer-product matrix. Each column is a multiple of Δv_b, so its column space has dimension at most one. Equivalently, for any state displacement h,

$$B_b h = \frac{\Delta x_b^\top h}{\|\Delta x_b\|_2^2} \Delta v_b \in \mathrm{span}\{\Delta v_b\}, \qquad \mathrm{rank}(B_b) \le 1. \tag{22}$$

The map also satisfies the secant condition

$$B_b \Delta x_b = \frac{\Delta v_b \, \Delta x_b^\top \Delta x_b}{\|\Delta x_b\|_2^2} = \Delta v_b, \tag{23}$$

mapping the previous state increment exactly to the corresponding velocity increment. Given $\Delta x_b \neq 0$, B_b has rank one if $\Delta v_b \neq 0$, and is the zero matrix with rank zero if $\Delta v_b = 0$.

The predictor therefore uses only the projection of the current displacement onto the previous direction Δx_b and converts it into a velocity increment along Δv_b. Displacements orthogonal to Δx_b are mapped to zero. The implementation requires only inner products and vector operations, without explicitly constructing B_b. If $\Delta x_b = 0$, the closed-form solution is undefined and the least-squares objective is independent of λ; setting $\lambda_i = 0$ recovers $\hat{v}_i = v_b$. When only one previous refresh point is available, we use its velocity directly, as described in Sec. 4.1.

G Error Analysis of RAIC

By the triangle inequality, Eqs. (10) and (12) give a bound that requires no smoothness of the interval errors:

$$\begin{aligned} C_b - \hat{C}_b &= \sum_{j=a}^{b-1} \Delta t_j (e_j - \beta_j e_b), \\ \|C_b - \hat{C}_b\|_2 &\le \sum_{j=a}^{b-1} \Delta t_j \|e_j - \beta_j e_b\|_2. \end{aligned} \tag{24}$$

Thus, correction accuracy depends on the weighted error of the linear endpoint model within the interval.

Let $T_{ab} = t_b - t_a$. Assume that the errors of the same previous predictor admit a twice continuously differentiable representation $e(t)$ on $[t_a, t_b]$, with $e(t_j) = e_j$ and $\|e''(t)\|_2 \le M$. The linear interpolation is $\ell(t) = (t - t_a)e_b/T_{ab}$. Applying the scalar linear interpolation remainder bound to $\langle d, e(t) \rangle$ for any unit vector d gives

$$|\langle d, e(t) - \ell(t) \rangle| \le \frac{M}{2}(t - t_a)(t_b - t). \tag{25}$$

Taking the supremum over all unit vectors and substituting into Eq. (24) gives

$$\begin{aligned} \|C_b - \hat{C}_b\|_2 &\le \frac{M}{2} \sum_{j=a}^{b-1} \Delta t_j (t_j - t_a)(t_b - t_j) \\ &\le \frac{M T_{ab}^3}{8}. \end{aligned} \tag{26}$$

The last step follows from $(t_j - t_a)(t_b - t_j) \le T_{ab}^2/4$ and $\sum_{j=a}^{b-1} \Delta t_j = T_{ab}$, and also holds on nonuniform increasing time grids. If the error is affine in time, $e_j = \beta_j e_b$ and $C_b = \hat{C}_b$. Twice continuous differentiability is an additional assumption; discrete sampling does not guarantee it.

H Online Accumulation of RAIC

Starting at refresh step a, maintain the running sums

$$S_0 = \sum_{j=a}^{b-1} \Delta t_j, \qquad S_1 = \sum_{j=a}^{b-1} \Delta t_j t_j. \tag{27}$$

When the scheduler selects refresh at step b, compute

$$w_b = \frac{S_1 - t_a S_0}{t_b - t_a} = \sum_{j=a}^{b-1} \Delta t_j \beta_j, \qquad \hat{C}_b = w_b e_b. \tag{28}$$

This requires two scalar accumulators and no storage of intermediate velocity error tensors. If $b = a + 1$, there are no reuse steps, so $w_b = 0$ and no correction is needed.

To avoid zeroing the endpoint error or counting a step twice, process refresh step b in the following order:

1. Retain the previous predictor and evaluate $\hat{v}_b$ and $v_b = v_\theta(x_b, t_b)$ at the same state x_b before correction, obtaining $e_b = v_b - \hat{v}_b$.
2. Compute w_b and $\hat{C}_b$ from accumulators covering only steps $a, \dots, b-1$.
3. Store the original observation (x_b, v_b), update the predictor from the latest two refresh points, and apply $x_{b+1} = x_b + \Delta t_b v_b + \hat{C}_b$. Do not pair corrected states with old velocities or replace cached v_b with an effective velocity that includes the displacement correction.
4. Reset the accumulators for the new interval starting at b, then add Δt_b and $\Delta t_b t_b$ to the new S_0 and S_1, respectively. At subsequent reuse steps, accumulate these quantities after each state update.

Apply the correction once to the refresh-step update, without replaying earlier sampling steps. If no refresh follows the final reuse interval, its endpoint error is unobserved and no endpoint correction is applied.

I Construction Details for Figures 3 and 4

Construction of Figure 3. For each model, we record the input state x_i and output velocity v_i at all 50 sampling steps. FLUX.1-dev uses DrawBench and both video models use VBench. We fit one-dimensional PCA separately to the flattened states and velocities of the full-computation run. All curves for that model use these same fixed means and principal directions, with state on the horizontal axis and velocity on the vertical axis.

Top row: same-state comparison. Sampling always uses full-computation velocities. SeaCache's gate determines refresh steps on this original trajectory, while its residual-reuse outputs are recorded at the same states without affecting sampling. For each reuse step i, the two preceding refreshes $a < b < i$ define the SGVE coefficient λ_i in the original high-dimensional state space. We plot the projections of

$$\widehat{x}_i = x_b + \lambda_i (x_b - x_a), \qquad \widehat{v}_i = v_b + \lambda_i (v_b - v_a).$$

Thus, prediction crosses lie on the projected historical secant by construction; open circles show the actual (x_i, v_i), and triangles show SeaCache outputs at x_i. Red points and vertical dashed lines mark refreshes. The displayed windows are steps 33–48, 30–47, and 34–45 for FLUX, Wan, and HunyuanVideo, respectively, with one preceding refresh retained to define the first secant.

Bottom row: independent sampling trajectories. Full computation, SeaCache, and RACE start from identical noise and conditioning within each model, but each method feeds its own outputs back into sampling. We project the recorded states and sampler-facing velocities onto the same PCA axes. FLUX is shown from step 0, and both video models from step 10.

Construction of Figure 4. We run FLUX.1-dev on all 200 DrawBench prompts using 50 Euler steps, 1024×1024 resolution, guidance 3.5, and SeaCache threshold 0.3. Reuse steps use SGVE, with RAIC disabled. At each reuse state, an additional full evaluation measures $e_j = v_\theta(x_j, t_j) - \widehat{v}_j$ without changing the cache or sampling trajectory. At the next refresh b, we compute e_b using the previous predictor, before updating its history. All measurements use full-dimensional tensors, not PCA, and cover 5,830 reuse points in 2,606 intervals.

In (a), each point plots $q_j = \langle e_j, e_b \rangle / \|e_b\|_2^2$ against $\beta_j = (t_j - t_a)/(t_b - t_a)$; in (b), it plots $\cos(e_j, e_b)$ against β_j. Green curves are step-size-weighted means in 16 equal-width β bins; red dashed lines indicate $q_j = \beta_j$ and cosine 1. The endpoints $(0, 0)$ and $(1, 1)$ in (a) are boundary constraints, not additional observations.

In (c), we compare three error estimates at reuse steps: $\widetilde{e}_j = 0$, e_b, and $\beta_j e_b$. The point-error bars report $\sum |\Delta t_j| \|e_j - \widetilde{e}_j\|_2^2 / \sum |\Delta t_j| \|e_j\|_2^2$, pooled over all reuse points. For each interval (a, b), we also form

$$C_{a,b} = \sum_{j=a+1}^{b-1} \Delta t_j e_j, \qquad \widetilde{C}_{a,b} = \sum_{j=a+1}^{b-1} \Delta t_j \widetilde{e}_j.$$

The interval-error bars report $\sum_{(a,b)} \|C_{a,b} - \widetilde{C}_{a,b}\|_2^2 / \sum_{(a,b)} \|C_{a,b}\|_2^2$. Both ratios sum errors before normalization, rather than averaging per-prompt ratios. In the order above, point errors are $1.00/2.24/0.28$, and interval errors are $1.00/2.65/0.31$.

J Comparison with ERTACache and FoCa

ERTACache. ERTACache (Peng et al., 2026) explicitly addresses cache-error accumulation through an offline-selected reuse schedule, integration-step adjustment, and cached-output rectification. Its error model uses coefficients precomputed from calibration data through residual linearization and applies the estimated correction during sampling.

FoCa. FoCa (Zheng et al., 2026) models hidden-feature evolution as a feature ODE. It combines BDF2-inspired forecasting from recent features and feature-derivative estimates with Heun-style calibration anchored to the latest fully computed feature. This calibration refines the predicted hidden feature at reuse steps to reduce overshoot.

RACE. SGVE instead extrapolates the final velocity using the current latent state displacement and two online state–velocity observations. RAIC uses a newly observed refresh-endpoint velocity residual to estimate the time-weighted error accumulated over the preceding reuse interval and adds this estimate directly to the latent state. Thus, its correction is neither offline-calibrated output rectification nor reuse-step feature calibration. Neither SGVE nor RAIC requires offline error fitting or changes the sampling-time grid. Our contribution is this coupling of state-guided velocity prediction and refresh-anchored integral correction, not the general idea of cache-error correction or predictor–corrector design.

Comparison Experiment. Table 8 compares ERTACache and RACE on Wan 2.1 1.3B. At the same 4,769 TFLOPs, RACE improves PSNR by 5.72 dB and reduces LPIPS from 0.069 to 0.022, with 0.2 s higher latency. We do not report FoCa results because its code was not publicly available at the time of our experiments.

Table 8: Comparison on Wan 2.1 1.3B at matched computational cost.

Method	Param.	Latency (s) ↓	TFLOPs ↓	PSNR ↑	LPIPS ↓	SSIM ↑
ERTACache	$\lambda = 0.08$	**27.1**	4769	25.43	0.069	0.885
RACE	$\delta = 0.20$	27.3	4769	**31.15**	**0.022**	**0.951**

K Complete Quantitative Results

Tables 9–12 report the complete results for each model. Latency speedup is the Original latency divided by the latency of the current method; FLOPs speedup is defined analogously. Both ratios are computed from the displayed values and rounded to two decimal places. Parameters, latency, TFLOPs, PSNR, LPIPS, and SSIM match the corresponding main-text tables.

Table 9: Complete quantitative results on Wan 2.1 1.3B. Speedups are relative to the Original row.

Method	Param.	Latency (s)↓	Latency speedup↑	TFLOPs↓	FLOPs speedup↑	PSNR↑	LPIPS↓	SSIM↑
Original	—	54.5	1.00×	9934	1.00×	—	—	—
25 steps	—	27.8	1.96×	4967	2.00×	14.78	0.379	0.583
15 steps	—	17.2	3.17×	2980	3.33×	13.28	0.476	0.511
TeaCache	$\delta=0.09$	31.5	1.73×	4967	2.00×	23.53	0.112	0.839
TaylorSeer	$\mathcal{S}=2$	32.6	1.67×	5166	1.92×	18.38	0.241	0.712
SenCache	$\varepsilon=0.01$	31.7	1.72×	4959	2.00×	29.62	0.048	0.936
DiCache	$\delta=0.10$	28.2	1.93×	4808	2.07×	27.23	0.056	0.905
SeaCache	$\delta=0.20$	**27.2**	2.00×	**4769**	2.08×	29.38	0.043	0.931
RACE	$\delta=0.20$	27.3	2.00×	**4769**	2.08×	**31.15**	**0.022**	**0.951**
TeaCache	$\delta=0.20$	22.0	2.48×	3581	2.77×	19.16	0.244	0.720
TaylorSeer	$\mathcal{S}=4$	21.8	2.50×	3580	2.77×	14.20	0.440	0.561
SenCache	$\varepsilon=0.10$	23.7	2.30×	**3167**	3.14×	21.66	0.194	0.782
DiCache	$\delta=0.20$	23.0	2.37×	3203	3.10×	23.63	0.101	0.846
SeaCache	$\delta=0.35$	**19.8**	2.75×	3378	2.94×	23.53	0.119	0.828
RACE	$\delta=0.35$	**19.8**	2.75×	3378	2.94×	**24.40**	**0.088**	**0.854**

Table 10: Complete quantitative results on HunyuanVideo. Speedups are relative to the Original row.

Method	Param.	Latency (s)↓	Latency speedup↑	TFLOPs↓	FLOPs speedup↑	PSNR↑	LPIPS↓	SSIM↑
Original	—	78.9	1.00×	14703	1.00×	—	—	—
25 steps	—	42.1	1.87×	7351	2.00×	20.12	0.255	0.717
15 steps	—	27.4	2.88×	4411	3.33×	17.25	0.375	0.633
TeaCache	$\delta=0.12$	43.1	1.83×	7317	2.01×	23.55	0.166	0.808
TaylorSeer	$\mathcal{S}=2$	46.9	1.68×	7645	1.92×	24.93	0.137	0.836
SenCache	$\varepsilon=0.28$	41.0	1.92×	7272	2.02×	30.19	0.067	0.907
DiCache	$\delta=0.065$	40.8	1.93×	7221	2.04×	32.40	0.047	0.930
SeaCache	$\delta=0.26$	40.9	1.93×	7060	2.08×	32.44	0.048	0.930
RACE	$\delta=0.26$	**40.7**	1.94×	**7060**	2.08×	**35.71**	**0.027**	**0.952**
TeaCache	$\delta=0.16$	35.1	2.25×	5986	2.46×	20.74	0.245	0.739
TaylorSeer	$\mathcal{S}=3$	35.7	2.21×	**5293**	2.78×	21.01	0.225	0.740
SenCache	$\varepsilon=0.45$	35.1	2.25×	5986	2.46×	27.52	0.104	0.871
DiCache	$\delta=0.13$	35.2	2.24×	6029	2.44×	27.84	0.097	0.863
SeaCache	$\delta=0.35$	35.1	2.25×	5884	2.50×	28.16	0.091	0.879
RACE	$\delta=0.35$	**35.0**	2.25×	5884	2.50×	**31.18**	**0.060**	**0.912**

Table 11: Complete quantitative results on FLUX.1-dev. Speedups are relative to the Original row.

Method	Latency (s)↓	Latency speedup↑	TFLOPs↓	FLOPs speedup↑	PSNR↑	LPIPS↓	SSIM↑
Original	7.8	1.00×	1861	1.00×	—	—	—
25 steps	3.9	2.00×	930	2.00×	17.54	0.302	0.740
15 steps	2.4	3.25×	558	3.34×	15.36	0.414	0.663
TeaCache ($\delta = 0.3$)	4.2	1.86×	969	1.92×	21.18	0.184	0.819
TaylorSeer ($\mathcal{S} = 3$)	3.7	2.11×	831	2.24×	24.65	0.126	0.875
SenCache ($\varepsilon = 0.26$)	3.8	2.05×	807	2.31×	27.49	0.079	0.913
DiCache ($\delta = 0.1$)	3.8	2.05×	866	2.15×	27.62	0.059	0.932
SeaCache ($\delta = 0.3$)	**3.6**	2.17×	**777**	2.40×	27.03	0.086	0.906
RACE ($\delta = 0.3$)	**3.6**	2.17×	**777**	2.40×	**30.26**	**0.051**	**0.938**
TeaCache ($\delta = 0.6$)	2.5	3.12×	559	3.33×	17.24	0.339	0.701
TaylorSeer ($\mathcal{S} = 5$)	2.7	2.89×	521	3.57×	19.85	0.222	0.777
SenCache ($\varepsilon = 0.88$)	2.5	3.12×	570	3.26×	19.53	0.251	0.766
DiCache ($\delta = 0.4$)	2.5	3.12×	572	3.25×	21.58	0.167	0.827
SeaCache ($\delta = 0.6$)	**2.4**	3.25×	**484**	3.85×	21.26	0.213	0.807
RACE ($\delta = 0.6$)	**2.4**	3.25×	**484**	3.85×	**22.92**	**0.150**	**0.848**

Table 12: Complete quantitative results on MiniMax-H3. Speedups are relative to the Original row.

Method	Latency (s)↓	Latency speedup↑	TFLOPs↓	FLOPs speedup↑	PSNR↑	LPIPS↓	SSIM↑
Original	132.5	1.00×	31294	1.00×	—	—	—
25 steps	81.0	1.64×	16215	1.93×	17.78	0.340	0.643
15 steps	61.3	2.16×	9925	3.15×	15.7	0.435	0.583
TeaCache ($\delta = 0.23$)	68.9	1.92×	15123	2.07×	18.66	0.302	0.667
TaylorSeer ($\mathcal{S} = 2$)	**73.3**	1.81×	16966	1.84×	18.90	0.291	0.650
SenCache ($\varepsilon = 0.37$)	74.6	1.78×	15436	2.03×	23.47	0.157	0.808
DiCache ($\delta = 0.07$)	74.4	1.78×	15484	2.02×	22.28	0.190	0.761
SeaCache ($\delta = 0.28$)	74.9	1.77×	15584	2.01×	23.34	0.173	0.782
RACE ($\delta = 0.28$)	73.4	1.81×	**15329**	2.04×	**25.16**	**0.129**	**0.820**
TeaCache ($\delta = 0.42$)	56.2	2.36×	10177	3.07×	17.34	0.357	0.630
TaylorSeer ($\mathcal{S} = 4$)	56.1	2.36×	10177	3.07×	13.95	0.584	0.509
SenCache ($\varepsilon = 2.20$)	55.9	2.37×	10176	3.08×	17.74	0.336	0.640
DiCache ($\delta = 0.31$)	53.2	2.49×	**8942**	3.50×	17.87	0.314	0.655
SeaCache ($\delta = 0.55$)	53.1	2.50×	9392	3.33×	17.69	0.348	0.642
RACE ($\delta = 0.55$)	**52.1**	2.54×	9281	3.37×	**19.72**	**0.254**	**0.695**

L Additional Visualizations

Figures 10 and Figures 11 provide additional qualitative comparisons on FLUX.1-dev at refresh ratios of approximately 30% and 50%. Figures 13 and Figures 12 provide additional qualitative comparisons on Wan2.1 1.3B and HunyuanVideoat refresh ratios of approximately 30% and 50%. Figures 14 provide additional qualitative comparisons on MiniMax-H3 at refresh ratios of approximately 30%. These examples complement the main-text results, comparing RACE and other caching methods with the full-computation reference across different prompts.

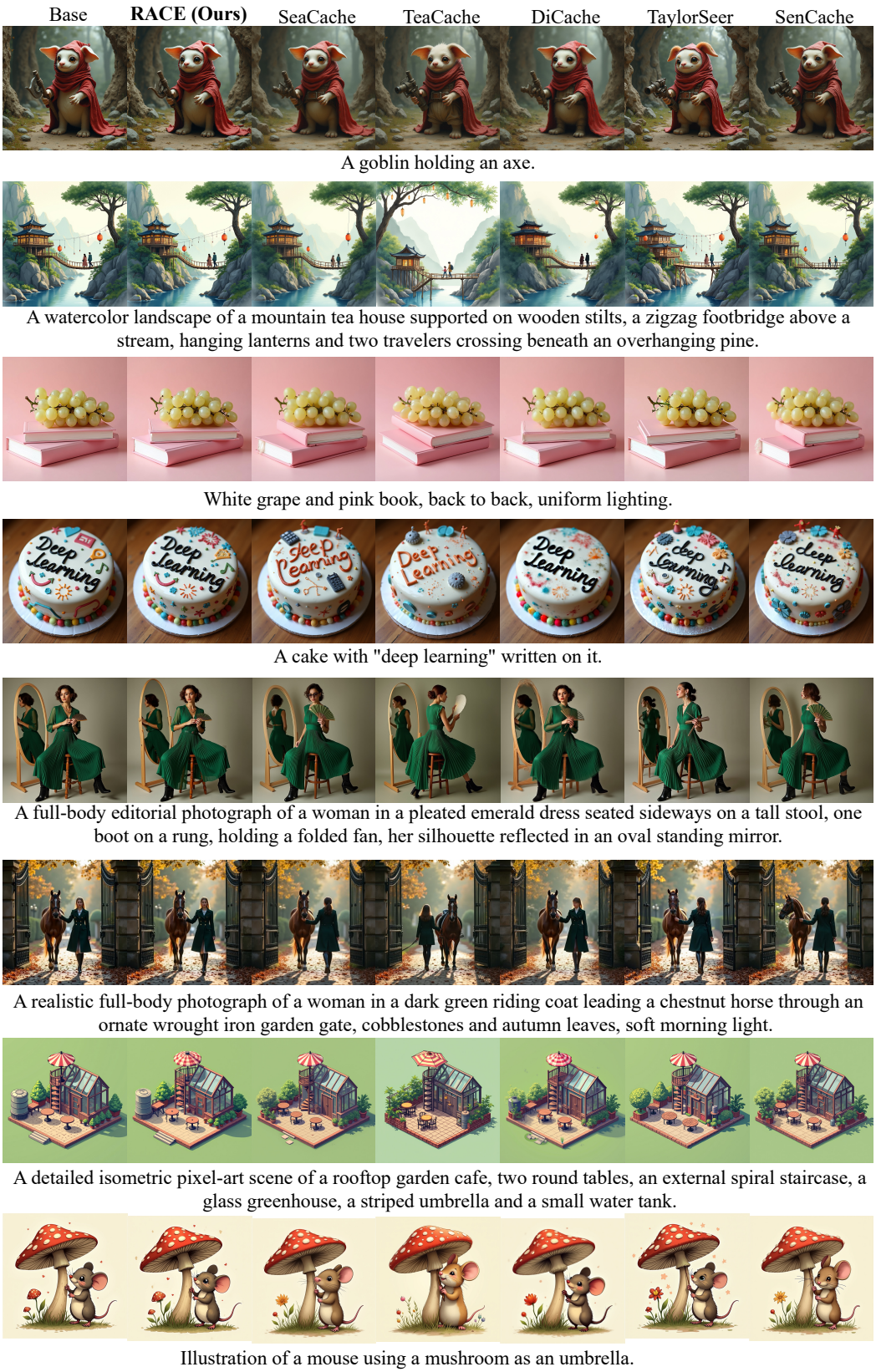


Figure 10: More qualitative comparison results on Flux.1-dev at refresh ratios of approximately 30%.

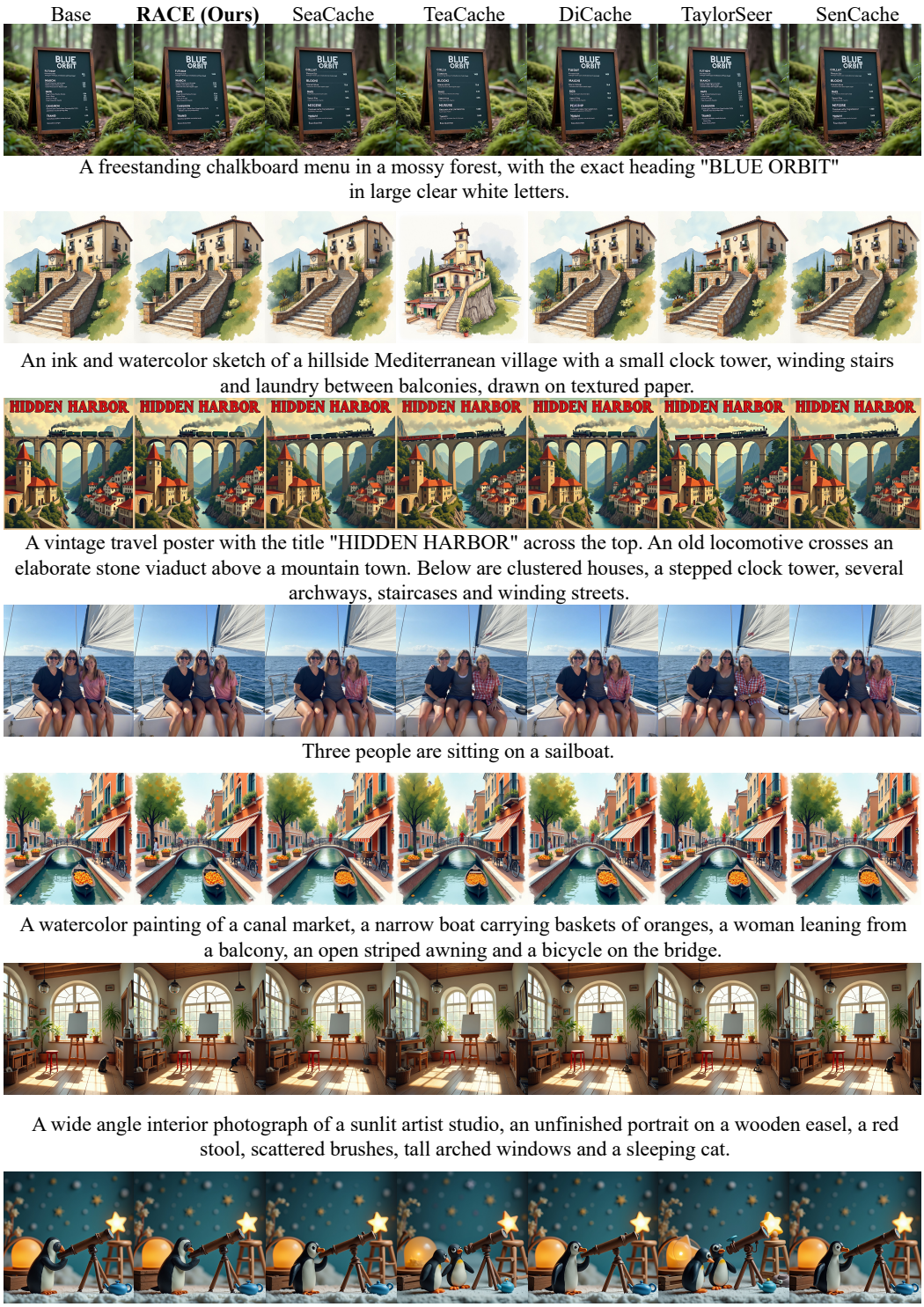


Figure 11: More qualitative comparison results on Flux.1-dev at refresh ratios of approximately 50%.



Figure 12: More qualitative comparison results on HunyuanVideo at refresh ratios of approximately 50% (left) and 30% (right).

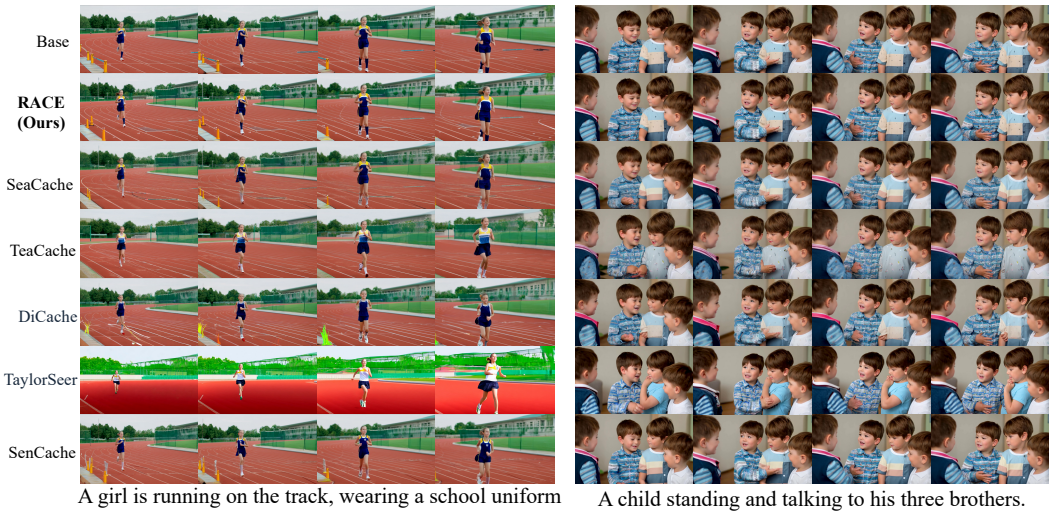


Figure 13: More qualitative comparison results on Wan 2.1 1.3B at refresh ratios of approximately 50% (left) and 30% (right).



A tranquil tableau of in the Arizona desert, a massive stone bridge arched across a rugged canyon



A couple in formal evening wear going home get caught in a heavy downpour with umbrellas, pixel art

Figure 14: More qualitative comparison results on MiniMax-H3 at refresh ratios of approximately 30% .