EDCACHE: ENDPOINT-DRIVEN TIMESTEP CACHING FOR DIFFUSION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion transformers excel at image and video generation, but iterative denoising incurs high inference costs. Existing caching methods often use local proxy signals to decide what to reuse. However, under matched compute, we find that importance-based token selection does not consistently outperform random selection and incurs additional overhead. We therefore shift our focus from token-level to timestep-level caching: reusing whole-timestep outputs can cause trajectory drift, while the position of reuse strongly affects final generation quality. To address trajectory drift and refresh timing in whole-timestep reuse, we propose EDCache, a two-stage method combining trajectory correction and timestep scheduling. Stage 1 uses recent full-computation outputs to correct drift during reuse without offline calibration. Stage 2 constructs schedule costs from model-specific single-position endpoint-damage profiles and uses dynamic programming to select full-computation timesteps under a fixed budget. Experiments on DiT-XL/2, FLUX.1-dev, and Wan2.1 show that EDCache improves generation fidelity at high denoising speedups. On DiT-XL/2, EDCache achieves more than $5\times$ denoising speedup with FID 3.64, outperforming all evaluated competing methods in the 2.5–$5\times$ speedup range across all five reported distribution metrics.

1 INTRODUCTION

Diffusion transformers (DiTs) (Peebles & Xie, 2023) have become a foundational architecture for high-quality image and video generation (Chen et al., 2024; Black Forest Labs, 2024; Team Wan et al., 2025), yet their iterative inference incurs substantial computation and latency. To accelerate diffusion inference, prior work has explored distillation, quantization, and caching. Caching reuses computation across adjacent timesteps during denoising, reducing redundant computation without retraining and offering a practical route to faster inference.

Existing DiT caching methods operate at different granularities. Some selectively recompute tokens or modules (Zou et al., 2025; 2026), others forecast intermediate features (Liu et al., 2025b), and another line determines which denoising timesteps require a full model evaluation (Liu et al., 2025a; Ma et al., 2025; Cui et al., 2026). Although their execution mechanisms differ, their decisions are commonly driven by proxy signals such as importance scores (Zou et al., 2025), local feature differences (Liu et al., 2025a), residual-magnitude ratios (Ma et al., 2025), or approximate trajectory costs (Cui et al., 2026). These proxies are convenient to compute, but their relationship to the quality of the final generation remains insufficiently understood. This leaves a more fundamental question: **can local proxy signals reliably guide caching decisions that preserve final generation quality?**

We first examine a caching decision strategy based on local proxy signals: selecting tokens to refresh according to importance. We compare importance-based and random selection under the same compute budget and cache implementation, using matched inputs and random seeds. We find that importance-based selectors show no consistent advantage over random selection at matched compute: across DiT-XL/2, PixArt-α, and FLUX.1-dev the paired quality difference stays within about 2% (Fig. 1*a*). Moreover, importance-based selection can introduce online scoring and selection overhead. For ToCa (Fig. 1*b*), token scoring, selection, and reindexing reach $\sim$34% of denoising time at BS=1. These results motivate studying whole-timestep reuse to reduce the overhead of fine-grained scoring and selection. This raises a further question: how much damage does caching at different temporal positions cause to the final output?

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.

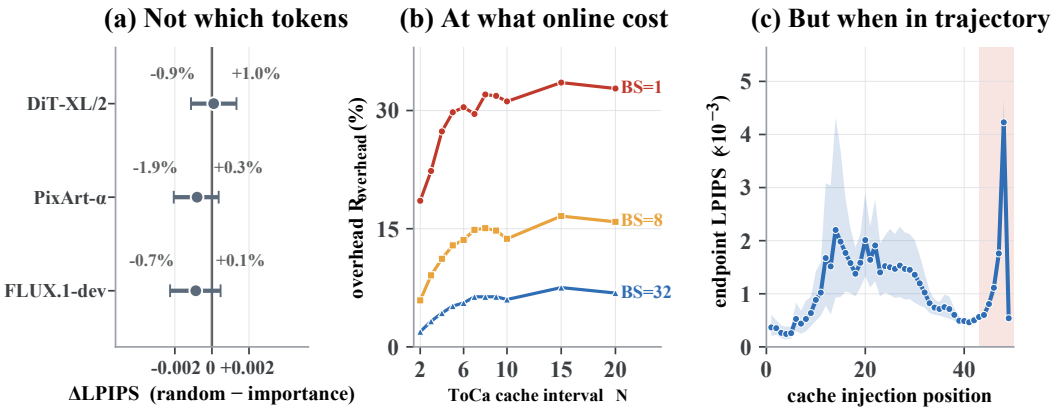

Figure 1: *(a)* Importance-based vs. seeded random token selection at matched compute under a unified executor. *(b)* Online selection overhead of ToCa on DiT-XL/2 vs. cache interval and batch size. *(c)* Endpoint damage vs. cache-injection position on DiT-XL/2 while caching one timestep and running all others in full.

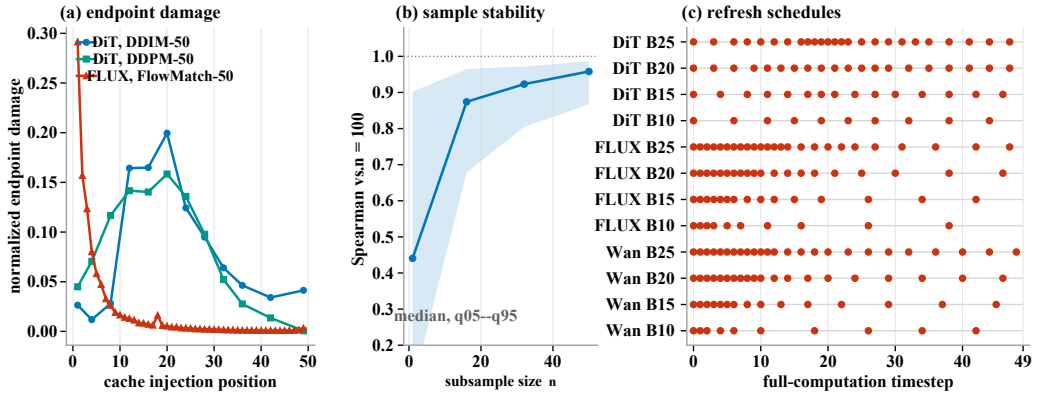

Figure 2: Temporal evidence behind the "when" decision. *(a)* Normalized single-position endpoint-damage profiles differ across DiT–DDIM50, DiT–DDPM50, and FLUX.1-dev–FlowMatch50, so a refresh policy cannot be transferred by timestep index alone. *(b)* Median Spearman agreement of subsampled DiT–DDIM50 profiles with the n=100 reference rises with sample size (shading: q05–q95 over prespecified subsamples). *(c)* Dots mark full-computation positions of representative Stage 2 schedules.

To relate cache position to final generation quality, we design controlled single-position interventions: each injects caching at only one designated timestep while running all other steps in full. Comparing against full inference with matched inputs and randomness measures the resulting damage to the final output. Three findings emerge. First, endpoint damage varies sharply with position—late-step interventions incur roughly $10\times$ the early-step damage in the reported profile (Fig. 1*c*). Second, within a fixed model–sampler configuration, the shape and ordering of the damage profile are consistent across samples, making profile reuse feasible (Fig. 2*b*). Third, the profile is not universal: changing the model or sampler shifts the critical temporal regions (Fig. 2*a*). These findings motivate assessing each timestep's importance to final quality through model- and sampler-specific damage profiles, then selecting refresh times to reduce caching errors.

Beyond selecting refresh times, we must also consider how to reuse cached outputs between full computations. As the denoising state evolves, directly reusing previous outputs introduces errors that cause drift in the subsequent trajectory. Based on these observations, we design EDCache around

trajectory correction and refresh timing. Stage 1 uses the output trend from the two most recent full-compute anchors to correct drift during reuse, without offline calibration. Stage 2 measures a model-specific single-position endpoint-damage profile, converts it to interval costs, and uses dynamic programming to choose full steps under budget and span constraints (Fig. 2*c*). The profile is reused across budgets (Sec. 3.2).

We study DiT-XL/2 (Peebles & Xie, 2023), FLUX.1-dev (Black Forest Labs, 2024), and Wan2.1 (Team Wan et al., 2025) across image and video generation. The same profile-derived cost construction supports multiple full-evaluation budgets, while the damage profiles themselves are measured separately for each model–sampler setting.

Contributions.

- **Endpoint-oriented caching analysis.** We connect token-level caching decisions, controlled timestep-wise interventions, and final-output damage in a unified analysis. We reveal temporal damage profiles that are stable across samples but shift with the model–sampler configuration, and organize these findings with a propagation-based local model, providing direct guidance for timestep-wise caching.
- **Trajectory repair and schedule optimization.** We develop a two-stage method for whole-timestep caching. Stage 1 repairs cache-induced drift through local trend extrapolation from the two most recent full-compute anchors. Stage 2 constructs a reusable interval cost from measured single-position endpoint damage and optimizes refresh times by dynamic programming under fixed budget and span constraints.
- **Extensive cross-model and cross-modal evaluation.** Experiments on DiT-XL/2, FLUX.1-dev, and Wan2.1 demonstrate improved speed–fidelity trade-offs over representative caching baselines across multiple compute budgets for image and video generation.

2 Related Work

Efficient diffusion inference has been pursued through distillation, quantization, pruning, and sampling-step reduction. Caching complements these approaches by reusing computation across denoising steps. We group prior work into token-level and timestep-level caching according to its primary reuse mechanism, while recognizing that some methods combine multiple granularities.

Token-level caching. These methods selectively recompute tokens within a denoising step. ToCa uses importance scores to select tokens for refresh (Zou et al., 2025), while TokenCache learns token importance together with block-wise caching ratios and temporal scheduling (Lou et al., 2024). ClusCa clusters tokens spatially, computes representative tokens, and propagates their updates to other members (Zheng et al., 2025). DuCa alternates aggressive reuse with conservative updates of randomly selected tokens (Zou et al., 2026). Its experiments show that importance-based selection need not outperform random selection. We revisit this observation under matched compute, a common executor, and paired endpoint evaluation.

Timestep-level caching. These methods decide when to refresh or reuse computation along the denoising trajectory. Some reuse intermediate module outputs across steps; others skip full denoiser evaluations at selected steps. FORA periodically reuses attention and MLP outputs (Selvaraju et al., 2024); TaylorSeer predicts future features through Taylor expansions using higher-order finite differences (Liu et al., 2025b); FoCa models hidden-feature evolution as an ODE and combines multistep forecasting with correction (Zheng et al., 2026).

Other methods determine reuse timings from error indicators. TeaCache estimates output changes from modulated-input differences (Liu et al., 2025a), while MagCache models skipping errors using calibrated residual-magnitude ratios (Ma et al., 2025). DiCache uses shallow online probes to customize schedules and aligns feature trajectories to improve reuse (Bu et al., 2026); SeaCache estimates redundancy from spectrally filtered input features (Chung et al., 2026).

Offline search and learning further optimize these decisions. BudCache searches for policies that preserve final-output quality under a fixed budget (Lei et al., 2026); DPCache uses path-dependent costs and dynamic programming to select key timesteps (Cui et al., 2026). ECAD jointly searches

caching decisions across timesteps and network components using evolution (Aggarwal et al., 2026), while ReCache learns budget-conditioned schedules through reinforcement learning (Aliev et al., 2026). ERTACache combines offline residual profiling, error rectification, and integration-interval adjustment (Peng et al., 2026). Our method has two stages: Stage 1 uses local trajectory analysis to correct reused whole-denoiser outputs; Stage 2 uses a measured model-specific endpoint-damage profile and budget-constrained dynamic programming to select full-computation timesteps. Together, the stages reduce caching error through output correction and timestep selection.

3 METHOD

Figure 3 provides an overview of EDCache. We first formalize whole-step output reuse and its constrained schedule, then derive the Stage 1 repair and the profile-guided Stage 2 search.

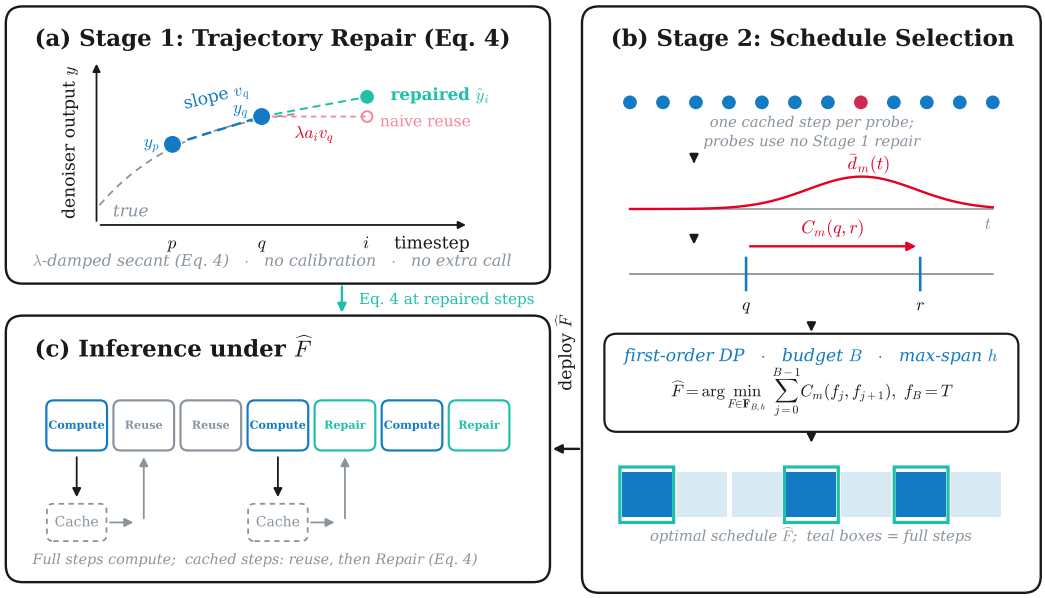

Figure 3: Overview of EDCache. (a) Stage 1 uses two recent full denoiser outputs to repair cached outputs; a_i is the sampler-specific extrapolation distance. (b) Stage 2 measures single-position endpoint damage, constructs interval costs, and selects full-computation steps with dynamic programming. (c) Inference follows the selected schedule and applies repair at reused steps. The damage profile is schematic.

3.1 PROBLEM SETUP

For steps $i = 0, \ldots, T-1$, let y_i denote the denoiser output evaluated at the current sampling state and $\hat{y}_i$ the output supplied to the sampler. At cached positions, the denoiser body is skipped and $\hat{y}_i$ is constructed from previous full outputs. We call this *whole-step output reuse*. At full positions,

$$\hat{y}_i = y_i, \qquad i \in F. \tag{1}$$

We write $F = (f_0, \ldots, f_{B-1})$ for the ordered set of B full-computation positions. Let h be the maximum allowed number of consecutive cached positions. We use $\mathcal{F}_{B,h}$ to denote all schedules with B full positions that satisfy this span constraint. Under this fixed budget, our goal is to reduce the endpoint discrepancy between cached and Full inference. Stage 1 corrects $\hat{y}_i$ at cached positions, whereas Stage 2 selects F by optimizing a profile-derived schedule-cost surrogate. EDCache combines both stages; output correction and timestep selection can also be integrated separately into other caching methods.

3.2 TWO-STAGE WHOLE-STEP CACHING

3.2.1 STAGE 1: TRAJECTORY REPAIR

View denoiser outputs along the current trajectory as samples of a locally smooth function $y(t)$. Around the most recent full anchor q, first-order expansion gives

$$y_i = y_q + (i-q)y'_q + \mathcal{O}\big((i-q)^2\big)\,. \tag{2}$$

Here y'_q is the local derivative at q, and the remainder is second order in the step distance $i-q$. Let (p, y_p) and (q, y_q) be the two most recent full anchors, with $p < q < i$. Estimating y'_q by their secant yields

$$y_i \approx y_q + (i-q)\frac{y_q - y_p}{q-p}, \tag{3}$$

which is exact when $y(t)$ is locally affine. The two-anchor correction is

$$v_q = \frac{y_q - y_p}{q-p}, \qquad \hat{y}_i = y_q + \lambda(i-q)v_q. \tag{4}$$

Here v_q is the secant change per step, λ controls the extrapolation strength, and $i-q$ is the extrapolation distance a_i for the integer-step coordinate. FLUX and Wan use the sampler's continuous noise coordinate for a_i; Appendix A specifies each backbone's coordinate, coefficient, and startup rule. The two-anchor approximation bound appears in Appendix B. The anchors are stored during the current generation; no separate full trajectory or extra denoiser evaluation is required. Coefficients are fixed across samples, with no per-sample search at deployment.

3.2.2 STAGE 2: PROFILE-GUIDED SCHEDULE SEARCH

For each model–sampler configuration, we measure the endpoint damage of caching at one position at a time. At probe position t, the denoiser is skipped once and its preceding full output is reused without Stage 1 repair; all other steps are computed in full. Using paired conditioning and initial noise, the measured mean damage is

$$d_m(t) = \frac{1}{N_m}\sum_{n=1}^{N_m} d\Big(G^{\text{probe}}_{\{t\}}(u_n), G_{\text{full}}(u_n)\Big)\,, \tag{5}$$

Here m denotes the model–sampler setting, N_m is its number of calibration inputs, and u_n is the nth paired conditioning and initial noise. $G^{\text{probe}}_{\{t\}}$ caches only step t without Stage 1 repair, G_{full} computes every step in full, and d is endpoint LPIPS. Linear interpolation between measured probe positions, with the nearest measured value held outside their range, gives the profile $\bar{d}_m(t)$ at every cached position. The profile is measured once for a fixed model–sampler setting and reused when the full-evaluation budget changes.

The damage from multiple cached steps need not equal the sum of isolated damages. We therefore use the profile to construct a *schedule surrogate*, rather than treating it as a measurement of a complete schedule. If q is the most recent full position, the cost of caching position $t > q$ is

$$w_m(t,q) = \bar{d}_m(t)^{1.5}(t-q)^{1.8}\exp\left[-0.1\frac{t-(T-1)/2}{(T-1)/2}\right]. \tag{6}$$

The exponents and temporal tilt are fixed surrogate-design parameters shared across the three models; they are not fitted endpoint-error coefficients. For a full interval starting at q and ending before the next full position r, let

$$C_m(q,r) = \sum_{t=q+1}^{r-1} w_m(t,q). \tag{7}$$

The terminal boundary is $r = T$, so $C_m(q,T)$ includes all cached steps after the last full evaluation.

Let $F = (f_0, \ldots, f_{B-1})$ contain exactly B increasing full positions with $f_0 = 0$, and set $f_B = T$. The selected schedule minimizes

$$\widehat{F} = \arg\min_{F\in\mathcal{F}_{B,h}} \widehat{J}_m(F), \qquad \widehat{J}_m(F) = \sum_{j=0}^{B-1} C_m(f_j, f_{j+1}), \tag{8}$$

where each gap contains at most h cached steps, including the terminal suffix. The final sampling step need not be full.

Because an interval cost depends only on its last full position and next boundary, a first-order dynamic program suffices. If $D_k(q)$ is the minimum surrogate cost of a legal partial schedule with k full positions ending at q, initialize $D_1(0) = 0$ and use

$$D_{k+1}(r) = \min_{\substack{q<r \\ r-q-1\le h}} \big[D_k(q) + C_m(q, r)\big]. \tag{9}$$

After B full positions, choose a feasible terminal q minimizing $D_B(q) + C_m(q, T)$ subject to $T - 1 - q \le h$, then backtrack. With interval costs precomputed, the DP takes $\mathcal{O}(BT(h + 1))$ time. It exactly minimizes the constructed surrogate over the legal schedules; final-output quality is measured separately by executing the selected complete schedule.

Algorithm 1 Profile-guided first-order schedule search

Require: Steps T, full-evaluation budget B, cached-span limit h, interval costs $C_m(q, r)$
Ensure: Full-computation positions $\widehat{F}$

Set $D_k(q) \leftarrow +\infty$ for all states and $D_1(0) \leftarrow 0$
for $k = 1, \dots, B - 1$ **do**
  for $q = 0, \dots, T - 1$ with finite $D_k(q)$ **do**
    for $r = q + 1, \dots, \min(T - 1, q + h + 1)$ **do**
      if $D_k(q) + C_m(q, r) < D_{k+1}(r)$ **then**
        $D_{k+1}(r) \leftarrow D_k(q) + C_m(q, r)$
        $P_{k+1}(r) \leftarrow q$
      end if
    end for
  end for
end for
$q^* \leftarrow \arg\min_{q:T-1-q\le h}[D_B(q) + C_m(q, T)]$
Backtrack from q^* through P to recover $\widehat{F}$
return $\widehat{F}$

4 Experiments

4.1 Setup

Models and data. We evaluate three transformer denoisers with their 50-step samplers: DiT-XL/2 on ImageNet-256 (Deng et al., 2009) with DDIM (Song et al., 2021), FLUX.1-dev on DrawBench-200 (Saharia et al., 2022) with FlowMatch Euler (Lipman et al., 2023), and Wan2.1-T2V-1.3B on 100 fixed prompts with FlowUniPC (Zhao et al., 2023). DiT uses 50K images for FID (Heusel et al., 2017), sFID (Nash et al., 2021), IS (Salimans et al., 2016), and precision/recall (Kynkäänniemi et al., 2019). For FLUX and Wan, accelerated outputs are paired with Full using the same prompt, seed, and initial latent. We measure LPIPS, SSIM, and PSNR against Full (Zhang et al., 2018; Wang et al., 2004), and CLIP text–image alignment for FLUX (Radford et al., 2021).

Baselines and measurement protocol. We compare with ToCa, DuCa, ClusCa, FORA, TaylorSeer, TeaCache, MagCache, and DPCache using their released implementations and native cache decisions under a shared prompt and seed assignments. All reported latencies cover denoising; DiT uses true batch size 64 with five timed batches, and FLUX and Wan use batch size 1 with four timed samples per arm after one warmup. The main tables show the B15 and B10 budgets; B25, B20, and configuration details appear in Appendices C and A. We also report native Full sampling at 10, 15, 20, and 25 total steps, with every step fully computed on its own sampler grid and no caching. These rows are distinct from the 50-step caching budgets: their latency is omitted, and † marks the nominal $50/\text{steps}$ ratio rather than a measured denoising speedup.

Implementation details. Reported latency values are measured on a single NVIDIA L20 GPU per arm. All FLUX speedups use a single official L20 Full reference (43.830 s/image) as the common

denominator. All FLUX arms use one warmup and four timed samples at BS1. Efficient (fused) attention is retained where supported by the method's native implementation; the tables mark this compatibility rather than assuming FlashAttention for every baseline.

4.2 Main quantitative comparison

Table 1 compares DiT at B15/B10; Appendix Table 6 covers B25/B20/B15/B10. The profile-guided Stage 2 schedules are evaluated on 50K images at each budget.

Table 2 reports FLUX at B15/B10. Stage 2 reduces LPIPS by 11.8% and 8.5% relative to the baseline with the lowest LPIPS at each budget, respectively, and improves SSIM and PSNR over both MagCache and DPCache. Although DuCa achieves higher CLIP scores, Fig. 4 illustrates our method's advantage in preserving text details: Stage 2 retains legible "Hello World" text, which DuCa fails to render.

Table 3 reports Wan at B15/B10. Stage 2 achieves lower LPIPS and higher SSIM than MagCache at both budgets, with higher PSNR at B15. Appendix Table 10 and Fig. 11 further report five VBench (Huang et al., 2024) quality dimensions for the paired B15 videos. Appendix D isolates the contributions of trajectory correction and schedule optimization across all three models.

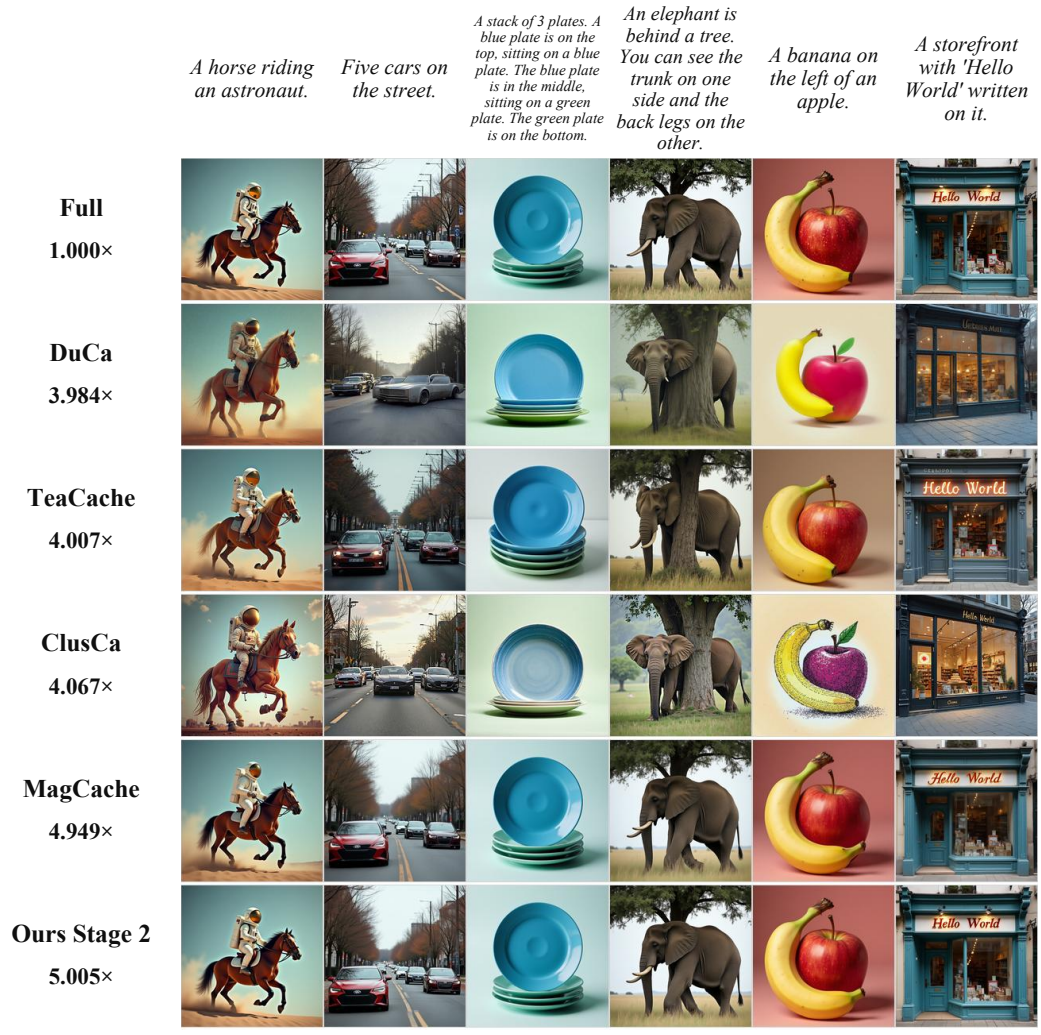

Figure 4: FLUX B10 visual comparison on six DrawBench prompts with paired seeds. Rows show Full, four caching baselines, and Stage 2.

Table 1: Quantitative evaluation of inference efficiency and visual quality on DiT-XL/2 (50K images; caching methods use DDIM-50).

Method	Eff. attn.	Acceleration		Quality				
		Latency (ms/img)↓	Speed↑	FID↓	sFID↓	IS↑	Prec.↑	Rec.↑
Full (50 steps)	✓	381.456	1.000×	2.1915	4.3153	251.18	0.8028	0.5913
B15: $\approx 3.3\times$ *anchor*								
Native Full (15 steps)	✓	–	3.333×†	5.0435	6.1247	214.48	0.7571	0.5551
FORA (N=4)	✓	140.118	2.722×	4.6095	8.4977	222.35	0.7804	0.5254
TaylorSeer (N=9, O=1)	✓	140.172	2.721×	6.0985	10.0942	196.12	0.7424	0.5411
ClusCa (N=10, K=4, O=1)	✓	135.017	2.825×	6.2091	6.9374	180.86	0.7348	0.5475
DuCa (N=6, r=0.1)	✓	126.745	3.010×	6.6504	7.9592	194.98	0.7256	0.5502
Ours Stage 1 (B15)	✓	**115.423**	**3.305**×	**2.5957**	**5.0528**	241.51	0.7990	**0.5795**
Ours Stage 2 (B15)	✓	115.451	3.304×	2.6882	5.6048	**242.11**	**0.8009**	0.5750
B10: $\approx 5\times$ *anchor*								
Native Full (10 steps)	✓	–	5.000×†	12.2573	11.3413	165.65	0.6671	0.5127
FORA (N=5)	✓	125.840	3.031×	6.4496	11.2887	201.25	0.7514	0.4976
DuCa (N=8, r=0.1)	✓	114.916	3.319×	17.0632	15.4127	135.15	0.5834	0.5426
Ours Stage 1 (B10)	✓	76.225	5.004×	**3.6391**	**5.5680**	220.55	0.7762	**0.5675**
Ours Stage 2 (B10)	✓	**76.149**	**5.009**×	3.6426	6.8239	**225.22**	**0.7818**	0.5569

Table 2: Quantitative evaluation of inference efficiency and visual quality on FLUX.1-dev.

Method	Eff. attn.	Acceleration		Quality			
		Latency (s/image)↓	Speed↑	CLIP↑	LPIPS↓	SSIM↑	PSNR↑
Full (RF50)	✓	43.830	1.000×	31.414	–	–	–
B15: $\approx 3.3\times$ *anchor*							
Native Full (15 steps)	✓	–	3.333×†	31.429	0.4176	0.4977	15.24
TaylorSeer (N=3, O=1)	✓	18.538	2.364×	31.443	0.2222	0.7072	19.66
ToCa (N=9)	–	17.369	2.523×	31.551	0.4477	0.5375	16.52
ClusCa (N=4, O=2, K=16)	✓	15.104	2.902×	31.580	0.6759	0.2144	10.39
MagCache (K3)	✓	15.012	2.920×	31.432	0.1377	0.8146	24.24
DuCa (N=5)	✓	14.775	2.967×	**31.828**	0.3516	0.5976	17.41
TaylorSeer (N=4, O=2)	✓	14.455	3.032×	31.464	0.2842	0.6447	18.15
DPCache (k=15)	✓	13.724	3.194×	31.606	0.1568	0.7892	22.66
FORA (N=4)	✓	13.470	3.254×	31.627	0.4457	0.4872	14.77
Ours Stage 1 (B15)	✓	**13.290**	**3.298**×	31.304	0.4260	0.4875	14.82
Ours Stage 2 (B15)	✓	13.298	3.296×	31.556	**0.1215**	**0.8432**	**26.59**
B10: $\approx 5\times$ *anchor*							
Native Full (10 steps)	✓	–	5.000×†	31.179	0.4794	0.4438	14.26
DuCa (N=10)	✓	11.002	3.984×	**31.912**	0.4859	0.5090	16.26
TeaCache (l=0.8)	✓	10.939	4.007×	31.808	0.4071	0.5250	15.87
ClusCa (N=7, O=2, K=16)	✓	10.778	4.067×	31.681	0.7102	0.1909	10.21
TaylorSeer (N=7, O=2)	✓	10.158	4.315×	31.324	0.4630	0.4953	15.47
DPCache (k=10)	✓	9.472	4.627×	31.568	0.2391	0.6999	20.25
FORA (N=8)	✓	8.950	4.897×	31.333	0.5683	0.4033	13.68
MagCache (K11, τ=1.10)	✓	8.856	4.949×	31.561	0.2846	0.6852	20.65
Ours Stage 1 (B10)	✓	8.768	4.999×	31.373	0.4617	0.4601	14.41
Ours Stage 2 (B10)	✓	**8.757**	**5.005**×	31.582	**0.2187**	**0.7444**	**22.49**

Fig. 5 shows the latency–quality operating points across all four budgets for DiT, FLUX, and Wan. Across these settings, Stage 2 combines short measured denoising latency with low FID on DiT and low paired LPIPS on FLUX and Wan, yielding favorable speed–fidelity trade-offs relative to the evaluated caching baselines.

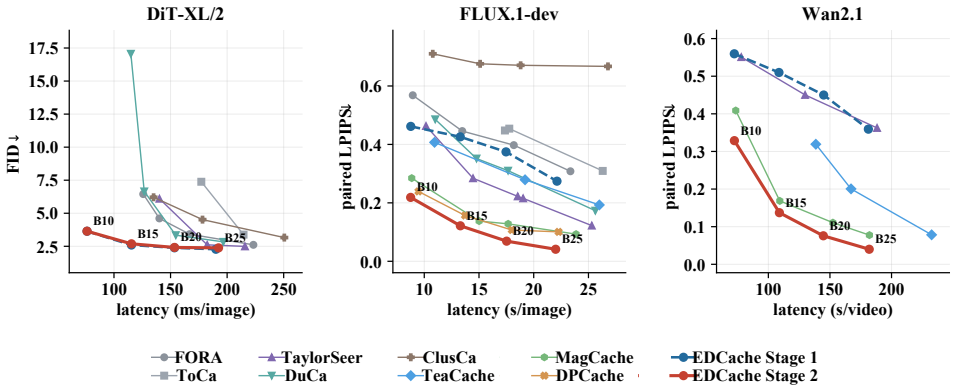

Figure 5: Latency–quality operating points from the detailed results for DiT, FLUX, and Wan; EDCache lines connect its four budgets.

Table 3: Quantitative evaluation of inference efficiency and visual quality on Wan2.1-T2V-1.3B.

Method	Eff. attn.	Acceleration		Quality		
		Latency (s/video)↓	Speed↑	LPIPS↓	SSIM↑	PSNR↑
Full (50 steps)	✓	361.315	1.000×	–	–	–
B15: $\approx 3.3\times$ *anchor*						
Native Full (15 steps)	✓	–	3.333×†	0.4762	0.4879	13.06
TeaCache ($\tau = 0.14$)	✓	138.538	2.608×	0.3192	0.6183	16.13
TaylorSeer (N=3, O=1)	✓	129.824	2.783×	0.4503	0.5054	13.35
MagCache (K8, $\delta = 0.60$)	✓	109.291	3.306×	0.1685	0.7741	21.04
Ours Stage 1 (B15)	✓	**108.622**	**3.326**×	0.5099	0.4772	12.73
Ours Stage 2 (B15)	✓	108.961	3.316×	**0.1372**	**0.8053**	**22.49**
B10: $\approx 5\times$ *anchor*						
Native Full (10 steps)	✓	–	5.000×†	0.5570	0.4471	12.19
TaylorSeer (N=5, O=1)	✓	78.025	4.631×	0.5508	0.4459	12.21
MagCache (K50, $\delta = 9.99$)	✓	73.531	4.914×	0.4088	0.5464	**17.25**
Ours Stage 1 (B10)	✓	**72.360**	**4.993**×	0.5596	0.4555	12.33
Ours Stage 2 (B10)	✓	72.433	4.988×	**0.3287**	**0.6211**	16.41

5 Conclusion

We propose EDCache to improve whole-step output reuse through trajectory correction and full-evaluation timestep selection. Stage 1 corrects reused outputs using recent full computations, while Stage 2 uses measured single-position endpoint damage to select full-evaluation timesteps under a fixed budget. Experiments on DiT-XL/2, FLUX.1-dev, and Wan2.1 show favorable measured speed–fidelity trade-offs across the reported budgets. The damage profile measured for a model–sampler setting can be reused at different budgets. Through single-position interventions and multi-budget evaluation on image and video diffusion models, we show that endpoint damage provides direct evidence for caching decisions. This perspective may also inform future work on efficient inference for diffusion generators and world models.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and polishing, related literature search and discovery, experimental execution, and drafting parts of the manuscript. The authors reviewed the AI-assisted content and take full responsibility for the final text, citations, experimental results, and conclusions of this paper.

REPRODUCIBILITY STATEMENT

Appendix A specifies model and sampler settings, measured profile sizes, search limits, and selected full-step positions; Appendix B.3 gives the cost and DP definitions. Paired evaluations use the same prompt or label, initial latent, and Full reference output.

REFERENCES

Anirud Aggarwal, Abhinav Shrivastava, and Matthew Gwilliam. Evolutionary caching to accelerate your off-the-shelf diffusion model. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2506.15682.

Mishan Aliev, Eva Neudachina, Ilya Bykov, Aleksandr Oganov, Kirill Struminsky, Aibek Alanov, and Denis Rakitin. ReCache: Learning budget-aware caching schedules for diffusion models via REINFORCE. *arXiv preprint arXiv:2606.06060*, 2026.

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

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 (ICLR)*, 2026. arXiv:2508.17356.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.00426.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. arXiv:2602.18993.

Bowen Cui, Yuanbin Wang, Huajiang Xu, Biaolong Chen, Aixi Zhang, Hao Jiang, Zhengzheng Jin, Xu Liu, and Pipei Huang. Denoising as path planning: Training-free acceleration of diffusion models with DPCache. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. arXiv:2602.22654.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2I-CompBench: A comprehensive benchmark for open-world compositional text-to-image generation. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/f8ad010cdd9143dbb0e9308c093aff24-Abstract.html`.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21807–21818, 2024.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Mingkun Lei, Tong Zhao, Liangyu Yuan, and Chi Zhang. Budget-constrained step-level diffusion caching. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2606.13496.

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 (ICLR)*, 2023. arXiv:2210.02747.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025a. arXiv:2411.19108.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with TaylorSeers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025b. arXiv:2503.06923.

Jinming Lou, Wenyang Luo, Yufan Liu, Bing Li, Xinmiao Ding, Weiming Hu, Yuming Li, and Chenguang Ma. Token caching for diffusion transformer acceleration. *arXiv preprint arXiv:2409.18523*, 2024. URL https://arxiv.org/abs/2409.18523v2.

Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. MagCache: Fast video generation with magnitude-aware cache. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2506.09045.

Charlie Nash, Jacob Menick, Sander Dieleman, and Peter Battaglia. Generating images with sparse representations. In *International Conference on Machine Learning (ICML)*, volume 139, pp. 7958–7968, 2021. URL https://proceedings.mlr.press/v139/nash21a.html.

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

Xurui Peng, 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 (ICLR)*, 2026. arXiv:2508.21091.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021. arXiv:2103.00020.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2205.11487.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

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.

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

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

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

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

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 (NeurIPS)*, 2023. arXiv:2302.04867.

Shikang Zheng, Liang Feng, Xinyu Wang, Qinming Zhou, Peiliang Cai, Chang Zou, Jiacheng Liu, Yuqi Lin, Junjie Chen, Yue Ma, and Linfeng Zhang. Forecast then calibrate: Feature caching as ODE for efficient diffusion transformers. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2026. arXiv:2508.16211.

Zhixin Zheng, Xinyu Wang, Chang Zou, Shaobo Wang, and Linfeng Zhang. Compute only 16 tokens in one timestep: Accelerating diffusion transformers with cluster-driven feature caching. In *Proceedings of the ACM International Conference on Multimedia (ACM MM)*, 2025. arXiv:2509.10312.

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 (ICLR)*, 2025. arXiv:2410.05317.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking token-wise feature caching: Accelerating diffusion transformers with dual feature caching. *IEEE Transactions on Image Processing*, 35:6211–6220, 2026. doi: 10.1109/TIP.2026.3698363. arXiv:2412.18911.

APPENDIX GUIDE

The appendix starts with the experimental settings, hardware, and method derivation (Appendices A–B). Complete quantitative results for all four budget tiers and qualitative comparisons follow in Appendix C. Profile collection and reuse are described in Appendix D.

A HYPERPARAMETER SETTINGS

Shared settings. For each backbone, Stage 1 and Stage 2 use the same correction parameters across compute budgets. The budget B specifies the number of full denoiser evaluations; the two stages differ in their placement of these evaluations. Correction parameters are fixed across samples. Both stages retain the two most recent full-computation anchors $p < q < i$ when correcting cached step i. Only outputs from full computations are stored as anchors, and full-compute steps return the original denoiser output.

DiT-XL/2. Both stages use integer denoising-step positions. With the secant velocity $v_q = (y_q - y_p)/(q-p)$ taken over the integer step gap, the extrapolation distance is the step count $a_i = i - q$, and the two-anchor correction of Equation (4) reads

$$\hat{y}_i = y_q + \lambda\,(i-q)\,v_q = y_q + \lambda\,(i-q)\,\frac{y_q - y_p}{q-p}, \qquad \lambda = 1.0.$$

No clipping is applied. Before two full anchors are available, cached steps use the uncorrected cached output.

FLUX.1-dev. Both stages extrapolate in the scheduler's continuous noise coordinate σ (FlowMatch-Euler). Because σ is a nonlinear function of the step index, equal integer steps do not correspond to equal noise-level changes, so the extrapolation distance is measured in σ rather than in steps. Taking the secant slope in σ-space, $(y_q - y_p)/(\sigma_q - \sigma_p)$, and advancing it from σ_q to σ_i gives

$$\hat{y}_i = y_q + \lambda\,(\sigma_i - \sigma_q)\,\frac{y_q - y_p}{\sigma_q - \sigma_p}, \qquad \lambda = 0.5.$$

Equivalently, keeping $v_q = (y_q - y_p)/(q-p)$ over the integer gap as in Equation (4), the extrapolation distance becomes the multiplier $a_i = (\sigma_i - \sigma_q)(q-p)/(\sigma_q - \sigma_p)$, whose $(q-p)$ factor cancels the integer-gap normalization of v_q. No clipping is applied. Before two full anchors are available, cached steps use the uncorrected cached output.

Wan2.1-T2V-1.3B. Both stages use the FlowUniPC native continuous noise coordinate σ. As for FLUX, the secant slope is taken in σ-space, so advancing it from σ_q to σ_i yields

$$\hat{y}_i = y_q + \lambda\,(\sigma_i - \sigma_q)\,\frac{y_q - y_p}{\sigma_q - \sigma_p}, \qquad \lambda = 0.5,$$

i.e. against the integer-gap velocity $v_q = (y_q - y_p)/(q-p)$ the extrapolation distance is the multiplier $a_i = (\sigma_i - \sigma_q)(q-p)/(\sigma_q - \sigma_p)$. No clipping is applied. Before two full anchors are available, cached steps reuse the uncorrected cached output.

Schedule construction and profile measurement. Stage 1 places its B full evaluations approximately uniformly over the sampling trajectory, including the first and last steps. Stage 2 requires the first step to be full and optimizes the remaining positions at the same budget using the profile-cost dynamic program in Equation (9); its last step may be cached. The maximum cached span h counts consecutive cached steps, including a possible cached suffix after the last full evaluation. Table 4 lists the search bound used for each reported Stage 2 configuration.

For Stage 2 schedule construction, the DiT–DDIM50 profile averages 500 identities at 12 isolated cache-injection positions; FLUX–FlowMatch50 and Wan–FlowUniPC50 each average 32 identities at 20 positions. Every probe caches only its designated position without Stage 1 repair, while all other steps are full. The endpoint metric is paired LPIPS. For each backbone, measured mean values are interpolated linearly and held at the nearest measured value outside the probe range. All budgets

use the same profile within a backbone and the shared fixed cost parameters in Equation (6); only B and the search limit h change.

Table 4: Maximum consecutive cached steps h used to construct the reported Stage 2 schedules. All configurations use 50 denoising steps and B full evaluations.

Model	$B = 25$	$B = 20$	$B = 15$	$B = 10$
DiT-XL/2	2	2	3	5
FLUX.1-dev	7	7	7	11
Wan2.1-T2V-1.3B	7	7	7	7

Table 5: Full-computation positions of the profile-guided DP schedules (zero-based execution order).

Model	B	Full positions
DiT	25	$\{0, 3, 6, 8, 10, 12, 14, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 29, 31, 33, 35, 38, 41, 44, 47\}$
DiT	20	$\{0, 3, 6, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 32, 35, 38, 41, 44, 47\}$
DiT	15	$\{0, 4, 8, 11, 14, 17, 19, 21, 24, 27, 30, 34, 38, 42, 46\}$
DiT	10	$\{0, 6, 11, 15, 19, 23, 27, 32, 38, 44\}$
FLUX	25	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18, 20, 22, 24, 27, 31, 36, 42, 47\}$
FLUX	20	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 21, 25, 30, 38, 46\}$
FLUX	15	$\{0, 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 19, 26, 34, 42\}$
FLUX	10	$\{0, 1, 2, 3, 5, 7, 11, 16, 26, 38\}$
Wan	25	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 23, 26, 29, 32, 36, 40, 44, 48\}$
Wan	20	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 17, 20, 24, 29, 34, 40, 46\}$
Wan	15	$\{0, 1, 2, 3, 4, 5, 6, 8, 10, 13, 17, 22, 29, 37, 45\}$
Wan	10	$\{0, 1, 2, 4, 6, 10, 18, 26, 34, 42\}$

Compared-method evaluation settings. The cache settings for each baseline are given in the corresponding result-table rows. Caching methods use 50 sampling steps; native Full baselines use the step counts listed in the tables.

- **DiT-XL/2.** All compared methods use the same ImageNet-256 class-conditional identities, DDIM sampler ($\eta = 0$), and CFG scale 1.5. The quality comparison uses 50,000 images; latency is measured with BS64 on L20.
- **FLUX.1-dev.** The image comparison uses 200 DrawBench prompts, paired initial latents, 1024×1024 resolution, guidance scale 3.5, and the FlowMatch-Euler sampler.
- **Wan2.1-T2V-1.3B.** The video comparison uses 100 fixed VBench prompts and paired initial latents. All methods use FlowUniPC, guidance 6.0, scheduler shift 8.0, 832×480 resolution, and 81 frames.

B Full Derivation of Propagation, Trajectory Repair, and Schedule Search

B.1 Propagation and position dependence of cache errors

We examine how cache errors propagate through denoising and why their final impact can depend on the caching position. The analysis uses a local first-order approximation around the Full trajectory, with identical initial states and random draws for Full and cached inference.

Full computation and cache error. At step i, Full inference computes

$$y_i = f(x_i, i), \qquad x_{i+1} = S_i(x_i, y_i), \tag{10}$$

where f is the denoiser and S_i is the sampler update. Fixed model parameters and conditioning inputs are omitted from the notation. Let $\tilde{x}_i$ be the state under caching and $\hat{y}_i$ the output actually supplied to the sampler. Define

$$\delta x_i = \tilde{x}_i - x_i, \qquad e_i = \hat{y}_i - f(\tilde{x}_i, i). \tag{11}$$

Here $f(\tilde{x}_i, i)$ is the reference output obtained by recomputing at the current state; cached inference need not evaluate it. By definition,

$$\tilde{x}_{i+1} = S_i(\tilde{x}_i, \hat{y}_i) = S_i(\tilde{x}_i, f(\tilde{x}_i, i) + e_i)\,. \tag{12}$$

Thus e_i describes the output approximation error rather than an extra quantity added by the algorithm. At a full step, $e_i = 0$, although an accumulated state deviation δx_i may remain.

One-step propagation. First expand the denoiser around the Full state:

$$f(x_i + \delta x_i, i) \approx y_i + \frac{\partial f}{\partial x}\delta x_i. \tag{13}$$

The change in the output supplied to the sampler is therefore

$$\hat{y}_i - y_i \approx \frac{\partial f}{\partial x}\delta x_i + e_i. \tag{14}$$

Expanding the sampler in both inputs and subtracting the Full update gives

$$\begin{aligned} \delta x_{i+1} &\approx \frac{\partial S_i}{\partial x}\delta x_i + \frac{\partial S_i}{\partial y}(\hat{y}_i - y_i) \\ &\approx \left(\frac{\partial S_i}{\partial x} + \frac{\partial S_i}{\partial y}\frac{\partial f}{\partial x}\right)\delta x_i + \frac{\partial S_i}{\partial y}e_i. \end{aligned} \tag{15}$$

Writing

$$A_i = \frac{\partial S_i}{\partial x} + \frac{\partial S_i}{\partial y}\frac{\partial f}{\partial x}, \qquad B_i = \frac{\partial S_i}{\partial y}, \tag{16}$$

we obtain

$$\delta x_{i+1} \approx A_i \delta x_i + B_i e_i. \tag{17}$$

All derivatives are evaluated on the Full trajectory. The matrix A_i includes both the direct effect of the state on the sampler and its indirect effect through the denoiser output; B_i describes the effect of the current cache error.

Propagation to the endpoint. If an error is introduced only at position i, with no preceding state deviation, then

$$\delta x_{i+1} \approx B_i e_i, \qquad \delta x_{i+2} \approx A_{i+1} B_i e_i. \tag{18}$$

Continuing through the remaining steps yields

$$\delta x_T \approx P_i e_i, \qquad P_i = A_{T-1} \cdots A_{i+1} B_i. \tag{19}$$

For multiple cached positions, the same expansion gives

$$\delta x_T \approx \sum_{i \in \mathcal{C}} P_i e_i, \tag{20}$$

where $\mathcal{C}$ is the set of cached positions. Since P_i varies with position, similar local error magnitudes need not produce similar endpoint deviations.

Interactions between cached positions. The contributions above are vectors with directions. For two endpoint contributions $u = P_i e_i$ and $v = P_j e_j$,

$$\|u + v\|^2 = \|u\|^2 + \|v\|^2 + 2\langle u, v\rangle. \tag{21}$$

A positive cross-term reinforces the two contributions, whereas a negative one partially cancels them. Thus, even when state deviations approximately superpose, endpoint damage need not be additive. The squared norm illustrates this interaction; it does not assume that LPIPS is a squared error.

Caching at preceding steps also changes subsequent states and the available output history, making later errors dependent on the complete schedule. Consequently, the profile-derived sum is an optimization surrogate and need not accurately predict a complete schedule's LPIPS. The quality of each selected complete schedule is evaluated experimentally.

B.2 Derivation of calibration-free trajectory repair

Estimating the local trend. Let $p < q < i$ denote the two most recent full-compute positions and the current cached position. Their stored denoiser outputs are y_p and y_q. Direct reuse supplies y_q at step i. To account for changes since step q, we estimate the local output velocity by

$$v_q = \frac{y_q - y_p}{q - p}. \tag{22}$$

The numerator is the change between two full outputs, and the denominator is their step separation. Thus v_q measures output change per denoising step. Multiplying by the elapsed distance $i - q$ gives the extrapolated change $(i - q)v_q$.

Two-anchor extrapolation. For a twice continuously differentiable local output trajectory $y(t)$, linear interpolation with nodes p and q, evaluated at $i > q$, gives

$$y_i = y_q + (i - q)v_q + r_i, \qquad \|r_i\| \le \frac{(i - q)(i - p)}{2} \sup_{t \in [p,i]} \|y''(t)\|. \tag{23}$$

Here r_i is the extrapolation remainder and $y''(t)$ measures local curvature. The bound applies componentwise or in a compatible tensor norm. For an affine trajectory, $y''(t) = 0$ and the extrapolation is exact. We scale the estimated change by a fixed coefficient λ:

$$\hat{y}_i = y_q + \lambda(i - q)v_q. \tag{24}$$

Setting $\lambda = 0$ gives direct reuse, while $\lambda = 1$ gives the full secant extrapolation. Intermediate values shrink the extrapolated change.

B.3 Profile-guided schedule search

Isolated-position measurement. For each model–sampler setting, the calibration experiment changes exactly one execution position per run: that position reuses the preceding full denoiser output without Stage 1 correction, while the other $T - 1$ positions remain full computations. The same conditioning, seed, and initial noise are used for the probe and its Full reference. Endpoint LPIPS is averaged over the calibration identities at each measured position. The resulting profile is interpolated linearly between measured positions and held at the nearest measured value beyond the measured range. Its values describe isolated raw-cache interventions, not complete-schedule losses under trajectory repair.

From a profile to interval costs. For model m, let $\bar{d}_m(t)$ denote the interpolated profile. Given a previous full position q, Equation (6) increases the cost of a cached step with its age $t - q$ and applies a mild fixed temporal weight. The cost of an interval (q, r) is the sum over its cached positions, $C_m(q, r)$ in Equation (7). The same definition with $r = T$ accounts for the cached suffix. The powers 1.5 and 1.8 and the tilt -0.1 are fixed across models and budgets. They specify an optimization heuristic; they are not estimates of the physical propagation law and do not imply additivity of actual endpoint LPIPS.

Exact search for the surrogate. A legal schedule has $f_0 = 0$, exactly B full positions, strictly increasing positions below T, at most h cached positions between adjacent full positions, and at most h cached positions after the last one. Equation (9) keeps the least surrogate cost for each number of full positions and last full position. It is initialized with $D_1(0) = 0$ and all other states at $+\infty$. The terminal step adds $C_m(q, T)$ and checks $T - 1 - q \le h$. Thus a final full computation is allowed but not required. Since future interval costs depend on the previous schedule only through its last full position q, the first-order state is sufficient. Backpointers recover the selected schedule. Precomputing legal interval costs and searching their successors takes $\mathcal{O}(BT(h + 1))$ DP time. The profile can be reused for a different budget B, while the admissible schedules and their DP solution change with B and h.

The optimizer is exact for $\widehat{J}_m$, not for the unknown true complete-schedule loss. Cache interactions and the use of Stage 1 repair during final inference are assessed by evaluating the selected complete schedules against paired Full outputs.

C Detailed quantitative and qualitative results

C.1 DiT-XL/2

Table 6: Detailed DiT-XL/2 results on ImageNet-256 (50K images; caching methods use DDIM-50).

Method	Eff. attn.	Acceleration		Quality				
		Latency (ms/img)↓	Speed↑	FID↓	sFID↓	IS↑	Prec.↑	Rec.↑
Full (50 steps)	✓	381.456	1.000×	2.1915	4.3153	251.18	0.8028	0.5913
Tier I: B25 anchor (≈ 2×)								
Native Full (25 steps)	✓	–	2.000×†	2.8419	4.5678	239.93	0.7914	0.5717
ClusCa (N=6, K=16, O=1)	✓	250.562	1.522×	3.1595	5.7377	221.58	0.7862	0.5803
FORA (N=2)	✓	223.221	1.709×	2.6112	4.9117	246.64	0.8046	0.5684
TaylorSeer (N=4, O=2)	✓	215.766	1.768×	2.4933	5.2635	245.34	0.8029	0.5773
ToCa (N=4, r=0.1)	–	214.497	1.778×	3.3831	5.1342	236.60	0.7890	0.5599
DuCa (N=3, r=0.15)	✓	196.647	1.940×	2.8348	4.7613	242.21	0.7966	0.5734
Ours Stage 1 (B25)	✓	**190.080**	**2.007**×	**2.2762**	**4.5608**	248.92	0.8042	**0.5859**
Ours Stage 2 (B25)	✓	192.398	1.983×	2.3803	5.1017	**250.86**	**0.8086**	0.5776
Tier II: B20 anchor (≈ 2.5×)								
Native Full (20 steps)	✓	–	2.500×†	3.4248	4.9589	232.18	0.7827	0.5657
TaylorSeer (N=4, O=1)	✓	182.404	2.091×	2.6095	5.5631	244.51	0.8038	0.5678
ClusCa (N=8, K=16, O=1)	✓	178.251	2.140×	4.5180	5.8483	199.18	0.7578	0.5805
ToCa (N=6, r=0.1)	–	177.185	2.153×	7.3818	8.0048	187.73	0.7134	0.5658
FORA (N=3)	✓	166.817	2.287×	3.4169	6.4158	237.15	0.7982	0.5498
DuCa (N=4, r=0.1)	✓	154.641	2.467×	3.3405	4.9214	235.67	0.7895	0.5640
Ours Stage 1 (B20)	✓	**153.480**	**2.485**×	**2.3868**	**4.9067**	246.25	0.8039	**0.5827**
Ours Stage 2 (B20)	✓	153.499	2.485×	2.4140	5.0261	**248.54**	**0.8054**	0.5816
Tier III: B15 anchor (≈ 3.3×)								
Native Full (15 steps)	✓	–	3.333×†	5.0435	6.1247	214.48	0.7571	0.5551
FORA (N=4)	✓	140.118	2.722×	4.6095	8.4977	222.35	0.7804	0.5254
TaylorSeer (N=9, O=1)	✓	140.172	2.721×	6.0985	10.0942	196.12	0.7424	0.5411
ClusCa (N=10, K=4, O=1)	✓	135.017	2.825×	6.2091	6.9374	180.86	0.7348	0.5475
DuCa (N=6, r=0.1)	✓	126.745	3.010×	6.6504	7.9592	194.98	0.7256	0.5502
Ours Stage 1 (B15)	✓	**115.423**	**3.305**×	**2.5957**	**5.0528**	241.51	0.7990	**0.5795**
Ours Stage 2 (B15)	✓	115.451	3.304×	2.6882	5.6048	**242.11**	**0.8009**	0.5750
Tier IV: B10 anchor (≈ 5×)								
Native Full (10 steps)	✓	–	5.000×†	12.2573	11.3413	165.65	0.6671	0.5127
FORA (N=5)	✓	125.840	3.031×	6.4496	11.2887	201.25	0.7514	0.4976
DuCa (N=8, r=0.1)	✓	114.916	3.319×	17.0632	15.4127	135.15	0.5834	0.5426
Ours Stage 1 (B10)	✓	76.225	5.004×	**3.6391**	**5.5680**	220.55	0.7762	**0.5675**
Ours Stage 2 (B10)	✓	**76.149**	**5.009**×	3.6426	6.8239	**225.22**	**0.7818**	0.5569

In Fig. 6, each example pairs the Full output with the cached output. All rows use DDIM-50 and guidance 1.5. Ours, ToCa, DuCa, TaylorSeer (N=9, O=1), and ClusCa (N=10, O=1, K=4) are evaluated on the same inputs.

Figure 7 compares the fastest baselines in Table 6 with our $B = 10$ Stage 1/2 schedules and native 50-step, 15-step, and 10-step DDIM sampling. Caching uses DDIM-50, with measured speedups shown in the row labels. All images use guidance 3.5 and share initial noise within each column. Both stages remain close to Full at approximately 5× speedup, while several baselines show visible deviations at 2.0–3.3×.

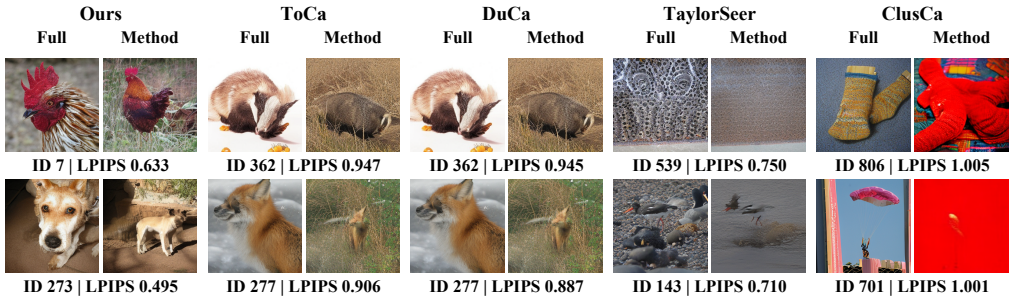

Figure 6: DiT-XL/2 worst cases: for each method, we independently select the two examples with the highest paired LPIPS against their corresponding Full outputs.

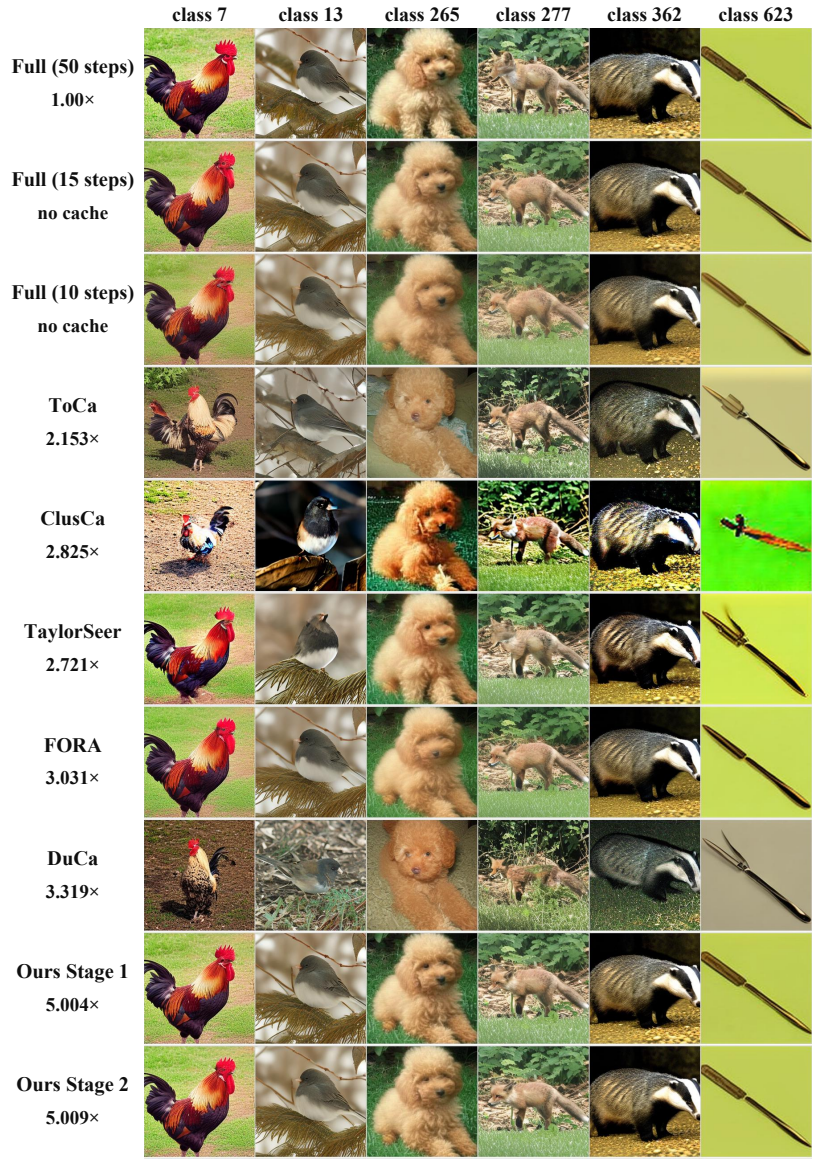

Figure 7: DiT-XL/2 results for six fixed ImageNet-256 classes (columns), using the fastest baseline configurations in Table 6 and our $B = 10$ Stage 1/2 schedules.

C.2 FLUX.1-DEV

Table 7: Detailed FLUX.1-dev results on DrawBench-200.

Method	Eff. attn.	Acceleration		Quality			
		Latency (s/image)↓	Speed↑	CLIP↑	LPIPS↓	SSIM↑	PSNR↑
Full (RF50)	✓	43.830	1.000×	31.414	–	–	–
Tier I: B25 anchor (2.0×)							
Native Full (25 steps)	✓	–	2.000×†	31.517	0.2958	0.6266	17.68
ClusCa (N=2, O=2, K=16)	✓	26.783	1.637×	31.537	0.6671	0.2232	10.41
ToCa (N=4)	–	26.303	1.666×	31.539	0.3093	0.6342	18.26
TeaCache (l=0.2)	✓	25.995	1.686×	31.498	0.1929	0.7495	20.92
DuCa (N=2)	✓	25.630	1.710×	**31.597**	0.1732	0.7711	21.77
TaylorSeer (N=2, O=2)	✓	25.304	1.732×	31.426	0.1222	0.8299	23.62
MagCache (K1)	✓	23.869	1.836×	31.340	0.0929	0.8649	26.05
FORA (N=2)	✓	23.352	1.877×	31.439	0.3081	0.6221	17.33
DPCache (k=25)	✓	22.274	1.968×	31.376	0.1007	0.8597	25.03
Ours Stage 1 (B25)	✓	22.125	1.981×	31.501	0.2742	0.6514	18.18
Ours Stage 2 (B25)	✓	**22.003**	**1.992**×	31.393	**0.0412**	**0.9348**	**32.15**
Tier II: B20 anchor (≈ 2.5×)							
Native Full (20 steps)	✓	–	2.500×†	31.368	0.3528	0.5638	16.41
TeaCache (l=0.375)	✓	19.209	2.282×	31.657	0.2791	0.6529	18.50
TaylorSeer (N=3, O=3)	✓	19.032	2.303×	31.516	0.2155	0.7182	20.02
ClusCa (N=3, O=1, K=4)	✓	18.804	2.331×	31.537	0.6708	0.2197	10.40
FORA (N=3)	✓	18.179	2.411×	31.410	0.3976	0.5297	15.55
DPCache (k=20)	✓	17.999	2.435×	31.422	0.1071	0.8459	24.70
ToCa (N=8)	–	17.767	2.467×	**32.018**	0.4540	0.5182	16.38
MagCache (K2)	✓	17.665	2.481×	31.393	0.1281	0.8233	24.46
DuCa (N=4)	✓	17.659	2.482×	31.526	0.3101	0.6336	18.21
Ours Stage 1 (B20)	✓	**17.469**	**2.509**×	31.277	0.3745	0.5389	15.84
Ours Stage 2 (B20)	✓	17.525	2.501×	31.410	**0.0689**	**0.9020**	**29.79**
Tier III: B15 anchor (≈ 3.3×)							
Native Full (15 steps)	✓	–	3.333×†	31.429	0.4176	0.4977	15.24
TaylorSeer (N=3, O=1)	✓	18.538	2.364×	31.443	0.2222	0.7072	19.66
ToCa (N=9)	–	17.369	2.523×	31.551	0.4477	0.5375	16.52
ClusCa (N=4, O=2, K=16)	✓	15.104	2.902×	31.580	0.6759	0.2144	10.39
MagCache (K3)	✓	15.012	2.920×	31.432	0.1377	0.8146	24.24
DuCa (N=5)	✓	14.775	2.967×	**31.828**	0.3516	0.5976	17.41
TaylorSeer (N=4, O=2)	✓	14.455	3.032×	31.464	0.2842	0.6447	18.15
DPCache (k=15)	✓	13.724	3.194×	31.606	0.1568	0.7892	22.66
FORA (N=4)	✓	13.470	3.254×	31.627	0.4457	0.4872	14.77
Ours Stage 1 (B15)	✓	**13.290**	**3.298**×	31.304	0.4260	0.4875	14.82
Ours Stage 2 (B15)	✓	13.298	3.296×	31.556	**0.1215**	**0.8432**	**26.59**
Tier IV: B10 anchor (10/50 full evaluations)							
Native Full (10 steps)	✓	–	5.000×†	31.179	0.4794	0.4438	14.26
DuCa (N=10)	✓	11.002	3.984×	**31.912**	0.4859	0.5090	16.26
TeaCache (l=0.8)	✓	10.939	4.007×	31.808	0.4071	0.5250	15.87
ClusCa (N=7, O=2, K=16)	✓	10.778	4.067×	31.681	0.7102	0.1909	10.21
TaylorSeer (N=7, O=2)	✓	10.158	4.315×	31.324	0.4630	0.4953	15.47
DPCache (k=10)	✓	9.472	4.627×	31.568	0.2391	0.6999	20.25
FORA (N=8)	✓	8.950	4.897×	31.333	0.5683	0.4033	13.68
MagCache (K11, τ=1.10)	✓	8.856	4.949×	31.561	0.2846	0.6852	20.65
Ours Stage 1 (B10)	✓	8.768	4.999×	31.373	0.4617	0.4601	14.41
Ours Stage 2 (B10)	✓	**8.757**	**5.005**×	31.582	**0.2187**	**0.7444**	**22.49**

Table 8: FLUX.1-dev results for a B10 schedule on 100 external prompts.

Method	CLIP↑	LPIPS↓	SSIM↑	PSNR↑
Full (50 steps)	31.611	0	1.0000	–
Full (25 steps)	31.537	0.2612	0.6789	19.31
Full (10 steps)	31.765	0.4490	0.5029	15.33
DPCache (k=10)	31.771	0.2195	0.7523	21.73
Ours Stage 1 (B10)	**31.848**	0.4392	0.5127	15.44
Ours Stage 2 (B10)	31.658	**0.1864**	**0.7934**	**23.30**



Figure 8: FLUX.1-dev on six DrawBench-200 prompts. Rows show the fastest Table 7 baselines, Stage 2 (B10), and native Full (10 steps), ordered by speedup.

Table 8 evaluates a FLUX B10 schedule on 100 T2I-CompBench prompts (Huang et al., 2023) with no exact text overlap with DrawBench-200; these prompts were not used to measure the damage profile or select the schedule. Methods share paired initial latents and 50-step Full references. Full (25/10 steps) uses native FlowMatch-Euler sampling; DPCache uses its official implementation with $k = 10$; both our stages use $\lambda = 0.5$. Stage 2 attains 0.1864 LPIPS versus Stage 1's 0.4392; relative to DPCache, it improves LPIPS by 0.0331 and SSIM from 0.7523 to 0.7934. Figure 8 shows qualitative results.

C.3 WAN2.1-T2V-1.3B

Table 9: Detailed Wan2.1-T2V-1.3B results on 100 videos (caching methods use FlowUniPC-50). Latency is denoising seconds per video, measured with BS1 on L20.

Method	Eff. attn.	Acceleration		Quality		
		Latency (s/video)↓	Speed↑	LPIPS↓	SSIM↑	PSNR↑
Full (50 steps)	✓	361.315	1.000×	–	–	–
Tier I: B25 anchor (2.0×)						
Native Full (25 steps)	✓	–	2.000×†	0.3619	0.5732	14.94
TeaCache ($\tau = 0.05$)	✓	232.543	1.554×	0.0784	0.8684	24.63
TaylorSeer (N=2, O=1)	✓	188.244	1.919×	0.3625	0.5723	14.92
MagCache (K2, $\delta = 0.12$)	✓	182.040	1.985×	0.0776	0.8734	24.78
Ours Stage 1 (B25)	✓	**181.196**	**1.994**×	0.3592	0.5760	15.07
Ours Stage 2 (B25)	✓	181.888	1.986×	**0.0400**	**0.9318**	**29.11**
Tier II: B20 anchor (≈ 2.5×)						
Native Full (20 steps)	✓	–	2.500×†	0.4122	0.5300	14.09
TeaCache ($\tau = 0.10$)	✓	167.111	2.162×	0.2005	0.7243	19.19
MagCache (K4, $\delta = 0.12$)	✓	152.387	2.371×	0.1107	0.8315	22.93
Ours Stage 1 (B20)	✓	144.914	2.493×	0.4500	0.5138	13.50
Ours Stage 2 (B20)	✓	**144.637**	**2.498**×	**0.0758**	**0.8836**	**26.18**
Tier III: B15 anchor (≈ 3.3×)						
Native Full (15 steps)	✓	–	3.333×†	0.4762	0.4879	13.06
TeaCache ($\tau = 0.14$)	✓	138.538	2.608×	0.3192	0.6183	16.13
TaylorSeer (N=3, O=1)	✓	129.824	2.783×	0.4503	0.5054	13.35
MagCache (K8, $\delta = 0.60$)	✓	109.291	3.306×	0.1685	0.7741	21.04
Ours Stage 1 (B15)	✓	**108.622**	**3.326**×	0.5099	0.4772	12.73
Ours Stage 2 (B15)	✓	108.961	3.316×	**0.1372**	**0.8053**	**22.49**
Tier IV: B10 anchor (5.0×)						
Native Full (10 steps)	✓	–	5.000×†	0.5570	0.4471	12.19
TaylorSeer (N=5, O=1)	✓	78.025	4.631×	0.5508	0.4459	12.21
MagCache (K50, $\delta = 9.99$)	✓	73.531	4.914×	0.4088	0.5464	**17.25**
Ours Stage 1 (B10)	✓	**72.360**	**4.993**×	0.5596	0.4555	12.33
Ours Stage 2 (B10)	✓	72.433	4.988×	**0.3287**	**0.6211**	16.41

Figures 9 and 10 complement Table 9 with frame-level comparisons at the B20 and near-5× (B10) operating points. Their prompts are "a person slicing a vegetable" and "couple dancing with body paint," respectively; within each figure, all rows share the prompt and initial noise. In the near-5× example, Stage 2 B10 preserves the dancers and room layout close to Full; TaylorSeer changes the setting, while MagCache shows more motion blur.

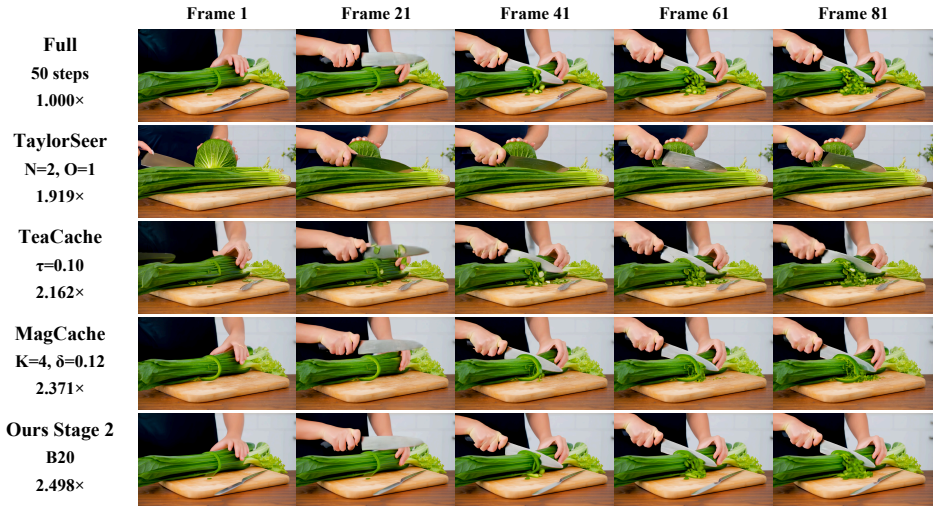

Figure 9: Wan comparison at lower speedups: Stage 2 B20 and the labeled baseline configurations.

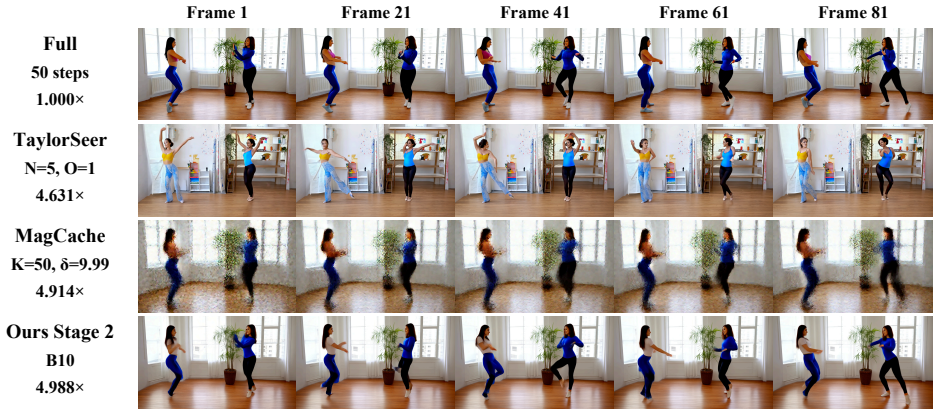

Figure 10: Wan comparison near $5\times$ speedup. All methods share the prompt and initial noise, with Full as the reference.

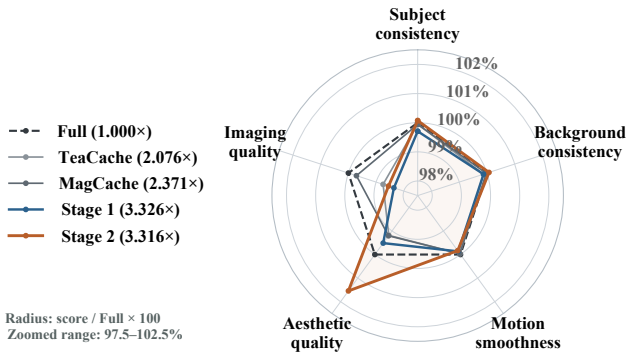

Figure 11: Five VBench scores normalized to Full $= 100\%$ (100 paired Wan videos per method; radial range 97.5–102.5%). Legend entries give L20 denoising speedups against the official Full; Table 10 gives absolute scores.

Table 10: Wan2.1-T2V-1.3B VBench scores on 100 paired videos at each method's operating speed.

Method	Speed↑	Subject↑	Back-ground↑	Smooth-ness↑	Aesthetic↑	Imaging↑
Full (50 steps)	1.000×	0.976625	0.980409	**0.990173**	0.635113	**0.714522**
TeaCache (τ=0.08)	2.076×	0.976154	0.979181	0.989829	0.629944	0.705576
MagCache (K=4, δ=0.12)	2.371×	0.976061	0.980178	0.989786	0.630063	0.712449
Ours Stage 1 (B15)	**3.326**×	0.973759	0.979188	0.988843	0.632034	0.702788
Ours Stage 2 (B15)	3.316×	**0.977341**	**0.981119**	0.988551	**0.644911**	0.704193

Table 10 reports five VBench dimensions on 100 paired Wan videos per method, with each method measured at its own operating speed. Speedups use the same official L20 Full denominator as Table 9; the five quality scores use the paired 100-video set, while latency uses four fixed L20 videos per arm. Full uses 50 FlowUniPC steps. Our method runs at a markedly higher speedup than the caching baselines (Stage 2 B15 at $3.316\times$ and Stage 1 B15 at $3.326\times$, versus $2.076\times$ for TeaCache and $2.371\times$ for MagCache), yet Stage 2 still attains the best subject consistency, background consistency, and aesthetic quality, with all three scores numerically above Full; Full remains best on motion smoothness and imaging quality. Stage 2 is therefore simultaneously faster and higher-fidelity than the compared baselines on these dimensions.

D Component ablation and profile measurement

D.1 Component ablation

Table 11: DiT-XL/2 component ablation at $B = 15$ (128 images; 50 sampling steps). Metrics are means against paired Full outputs with the same inputs and initial noise.

Schedule	Correction	LPIPS↓	SSIM↑	PSNR (dB)↑
Uniform	Off	0.12687	0.80931	25.06
Uniform	On	0.05760	0.90317	29.61
DP	Off	0.11102	0.83218	25.86
DP	On	**0.04740**	**0.92022**	**30.83**

Table 12: FLUX.1-dev component ablation at $B = 15$ (64 DrawBench prompts; 50 sampling steps). Metrics are means against paired Full outputs with the same inputs and initial noise.

Schedule	Correction	LPIPS↓	SSIM↑	PSNR (dB)↑
Uniform	Off	0.40346	0.48918	15.05
Uniform	On	0.39501	0.49384	15.09
DP	Off	0.12363	0.85003	27.35
DP	On	**0.08958**	**0.87856**	**28.33**

Table 13: Wan2.1-T2V-1.3B component ablation at $B = 15$ (16 videos; 50 sampling steps). Metrics are means against paired Full outputs with the same inputs and initial noise.

Schedule	Correction	LPIPS↓	SSIM↑	PSNR (dB)↑
Uniform	Off	0.48391	0.50706	13.91
Uniform	On	0.47328	0.50741	13.88
DP	Off	0.19959	0.76465	24.21
DP	On	**0.11214**	**0.82375**	**26.09**

Trajectory correction and schedule optimization each reduce mean LPIPS across all three models, and their combination achieves the lowest error.

D.2 Profile measurement and reuse

The DiT–DDIM50 profile measures 12 positions on 500 identities (6,000 single-position probes). FLUX–FlowMatch50 and Wan–FlowUniPC50 each measure 20 positions on 32 identities (640 probes per model). A probe caches one step and computes the other 49 steps in full, so obtaining a new profile has a substantial one-time generation cost. Stage 2 reuses each measured profile across the four reported budgets: changing B requires a new DP solve but no new probes for the same model–sampler setting. The selected schedule is then evaluated by complete generation. The measured profiles and reference solver are provided in the supplementary source.