BUDGETED BLOCK–TIME CACHE SCHEDULING FOR DIFFUSION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion models have demonstrated remarkable performance in image and video generation, while their multi-step denoising process incurs substantial computational overhead. Feature caching accelerates sampling by reusing intermediate activations across timesteps, making the caching schedule critical to both efficiency and generation quality. Existing scheduling methods primarily operate at the timestep level, imposing a shared execution decision on all Transformer blocks within a timestep and thereby overlooking the heterogeneous cache sensitivity and computational redundancy across network depth. We propose BTCache, which models cacheability jointly across denoising steps and Transformer blocks. By predicting the terminal quality impact of block–time reuse, BTCache identifies critical computations and allocates a prescribed compute budget across the joint block–time space. At each denoising step, a deterministic selector executes the budget-feasible action of lowest predicted terminal risk relative to an equal-budget timestep reference, regularized toward a calibrated reference plan, selectively recomputing sensitive blocks while reusing redundant features within the same timestep; a closed-form feasibility condition guarantees that the budget is spent exactly. On HunyuanVideo, Wan2.1, and FLUX.1-dev, BTCache is the fastest among all compared cache methods, attains the best perceptual quality on both video models, and achieves the highest PSNR and ImageReward on FLUX.1-dev.

1 INTRODUCTION

Diffusion transformers have advanced image and video generation (Ho et al., 2020; Peebles & Xie, 2023), but their inference cost grows with both the number of denoising steps and the depth executed at each step. Feature caching reuses intermediate activations from earlier exact evaluations (Ma et al., 2024; Wimbauer et al., 2024), and its speed–quality trade-off depends on *where* reuse is applied along denoising time and network depth. A cache scheduler therefore solves a fixed-budget allocation problem: it decides which computations remain exact so that local approximation error does not become terminal generation drift. MeanCache places evaluations of interval average velocities with a budget-constrained shortest path (Gao et al., 2026), and DPCache selects a fixed-budget sequence of full-computation timesteps by dynamic programming over a path-aware cost tensor (Cui et al., 2026a). We adopt their two principles: cache error is evaluated along the trajectory, and compute is allocated under an explicit budget.

Both methods, however, decide only *when* the model is evaluated. Each selected timestep remains a whole-stack decision, which restricts the schedule to columns of the block–time grid (Figure 1a). Reuse sensitivity is not uniform over this grid: the shallowest group is the most sensitive at every measured step, the middle groups are the least sensitive at most steps, and the deepest group varies strongly with time (Figure 1c). A column-level policy must then recompute the entire stack, spending budget on redundant groups, or reuse the sensitive group together with the redundant ones, injecting an error that later blocks and steps amplify. Fine-grained methods show that tokens, components, and blocks differ in reuse tolerance (Zou et al., 2025; Cui et al., 2026b; Cao et al., 2026), but independently scored groups do not determine the value of a joint action, because reuse errors interact across groups (Figure 1d). We therefore study how a prescribed budget should be allocated jointly over Transformer depth and denoising time.

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.

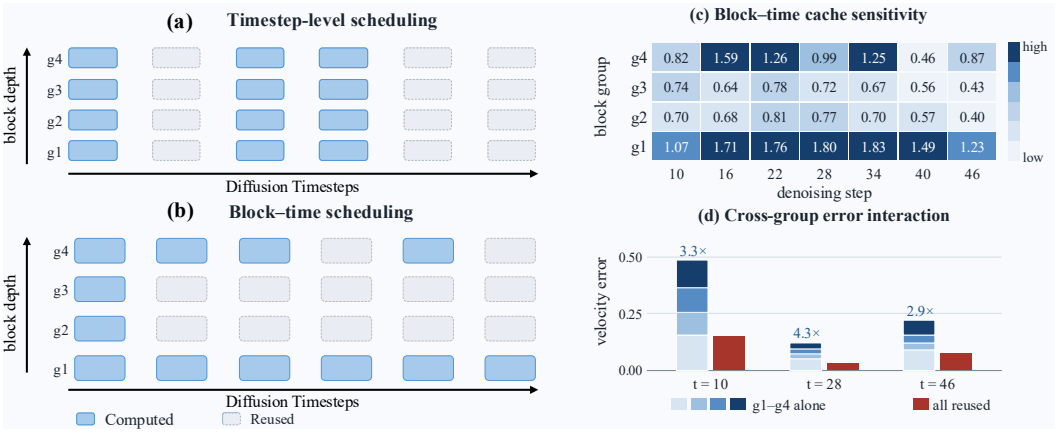


Figure 1: The granularity problem in fixed-budget cache scheduling. (a) Timestep-level scheduling assigns one FULL/CACHE action to the complete stack, so group evaluations can only be spent as whole steps. (b) Block–time scheduling spends the same number of evaluations (12 cells in both grids) over depth as well as time; the timestep-level action set is its depth-uniform restriction. Rows are the four block groups g1–g4, g1 shallowest; the pattern in (b) is illustrative. (c,d) Measured on HunyuanVideo along the all-FULL trajectory: the one-step velocity error $\|v - v_{\mathrm{FULL}}\|/\|v_{\mathrm{FULL}}\|$ when selected groups replay residuals from three steps earlier, averaged over three prompts. (c) Block–time cache sensitivity: the error of reusing a single group, relative to reusing the whole stack at the same step. (d) Cross-group error interaction: the four single-group errors, stacked, exceed the measured whole-stack error by 2.9–4.3×.

The problem is *terminal*, *causal*, and *joint*: a local perturbation is transformed by all later blocks and steps, the fresh residual that reveals the current reuse error exists only after FULL computation, and refreshing one group changes the input of later groups. **BTCache** acts at every step with a joint FULL/CACHE mask over four Transformer groups. A causal predictor scores all 2^4 masks in one batch by their terminal latent, perceptual, and, for video, temporal risk, each the sum of a time–action prior and a state correction. A selector applies a closed-form exact-budget feasibility test and executes the feasible mask of lowest predicted risk relative to an equal-budget timestep comparator, regularized toward a calibrated reference plan. Cached groups replay residuals refreshed by a damped extrapolation that adds no Transformer computation.

Our contributions are threefold:

- We identify the granularity limit of fixed-budget temporal scheduling and formulate diffusion caching as exact-budget allocation over the joint block–time space, in which timestep-level schedules form a strict subset.
- We develop a strictly causal predictor that scores all joint masks through cross-group interaction and predicts terminal latent, perceptual, and, for video, temporal risk without an all-FULL inference pass.
- We derive a closed-form necessary-and-sufficient feasibility condition for the remaining exact budget. On HunyuanVideo, Wan2.1, and FLUX.1-dev, BTCache is the fastest compared cache method. At matched cost it improves PSNR over MagCache by 1.47 dB on FLUX.1-dev and over TeaCache by 0.35 dB on Wan2.1, and controlled ablations attribute the gain to block–time allocation and terminal supervision.

2 Related Work

Cache payloads and adaptive reuse. DeepCache reuses high-level U-Net features (Ma et al., 2024), Block Caching assigns temporal reuse schedules to blocks (Wimbauer et al., 2024), and Pyramid Attention Broadcast applies component-specific broadcast ranges in video Transformers

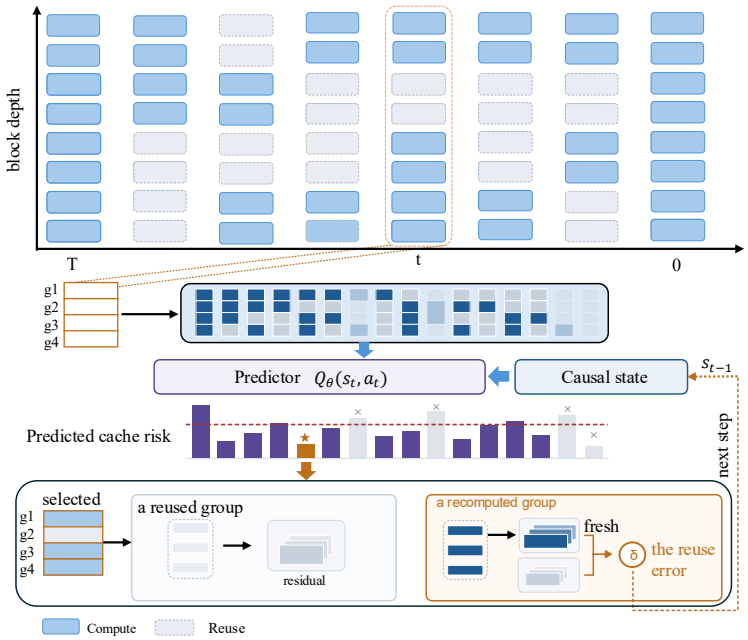


Figure 2: BTCache at denoising step t. The causal state is paired with all 16 group masks, scored jointly by $Q_\theta(s_t, a_t)$. Bars show each mask's predicted excess risk over the timestep comparator (dashed line); $\times$ marks masks that fail the feasibility test, and the selector executes the feasible mask of lowest J_t (Equation (7)). CACHE groups replay stored residuals; FULL groups recompute them and expose the fresh-minus-stale error that updates the next state.

(Zhao et al., 2025). Adaptive methods trigger reuse from trajectory signals: timestep-modulated input change in TeaCache (Liu et al., 2025a), feature dynamics and motion in AdaCache (Kahatapitiya et al., 2025), and calibrated residual magnitudes in MagCache (Ma et al., 2025). TaylorSeer, SpeCa, and ERTACache forecast or rectify features (Liu et al., 2025b;c; Peng et al., 2026), and DiCache, EasyCache, ToCa, and DuCa adapt the reuse decision at runtime (Bu et al., 2025; Zhou et al., 2025; Zou et al., 2025; 2026). These methods define cache values and local reuse signals. BTCache instead learns the terminal effect of a joint reuse action and applies to any payload whose state and cost are defined consistently.

Fixed-budget temporal scheduling. MeanCache (Gao et al., 2026) and DPCache (Cui et al., 2026a) are the closest formulations: both place a prescribed number of full evaluations along the trajectory, by a peak-suppressed shortest path over interval average velocities and by dynamic programming over a path-aware cost tensor. LeMiCa, ReCache, and BudCache study path error, learned schedules, and fixed-NFE masks (Gao et al., 2025; Aliev et al., 2026; Lei et al., 2026). Their decision variable is a timestep or a whole-model evaluation. BTCache keeps the fixed budget and changes the allocation unit to a joint block–time action conditioned on the state of the accelerated trajectory.

Fine-grained and block–time caching. ToCa ranks token-level cache sensitivity (Zou et al., 2025), BWCache triggers block-wise reuse from adjacent-step similarity (Cui et al., 2026b), and ProCache and ECAD combine selective block computation with searched block–time schedules (Cao et al., 2026; Aggarwal et al., 2026). These methods either score blocks independently or fix a schedule offline. BTCache scores the complete multi-group mask online from action-matched terminal outcomes and enforces exact remaining-budget feasibility for every sample.

3 METHOD

A learned predictor estimates the terminal quality consequence of every joint group action, and a deterministic selector converts the estimates into one budget-feasible action (Figure 2; Algorithm 1 in Appendix A). We define the action space and objective (Section 3.1), the causal state and cache operator (Section 3.2), the predictor and its targets (Sections 3.3–3.5), and the exact-budget selector (Section 3.6).

3.1 JOINT BLOCK–TIME ACTIONS

A diffusion Transformer with T steps and L blocks is partitioned into G contiguous, approximately compute-balanced groups, fixed before any prompt is evaluated. At step t, $a_{t,g} = 0$ denotes FULL evaluation of group g and $a_{t,g} = 1$ replay of its cached residuals. A step-level scheduler is restricted to $\mathcal{A}_{\text{step}} = \{\mathbf{0}_G, \mathbf{1}_G\}$, whereas BTCache selects $a_t \in \mathcal{A}^{(G)}_{\text{joint}} = \{0,1\}^G$. For a fixed cache operator and $G \geq 2$ the former is a strict subset of the latter.

Group resolution. G sets the depth resolution and is not part of the formulation: the action space, the predictor, and Proposition 1 hold for any G. Refining a partition only unties bits of the coarse masks, so $\mathcal{A}_{\text{step}} \subset \mathcal{A}^{(G')}_{\text{joint}} \subset \mathcal{A}^{(G)}_{\text{joint}}$ for $G' < G$, while the number of rows scored per step and of same-state rollouts needed as labels doubles with each group. We use $G = 4$, which gives four independently controllable depth regions and 16 masks per batched call (Appendix E compares $G \in \{1, 2, 4, 8\}$).

Budget and objective. One group evaluation is one scheduling unit, $c(a_t) = G - \sum_g a_{t,g}$, and a budget B_0 must be spent exactly, $\sum_t c(a_t) = B_0$. We state budgets in full-model-equivalent evaluations: a budget of n NFE corresponds to $B_0 = nG$ group evaluations, so it bounds the executed Transformer-block computation at n/T of all-FULL inference. The first two steps and the final step are mandatory FULL. These steps and the maximum cache age A are shared with the equal-budget timestep comparator, so the comparison differs only in how the remaining evaluations are distributed. The active risks $\mathcal{K}_\nu$ are latent and perceptual for images ($\nu = 0$), plus temporal for video ($\nu = 1$). Terminal risks $Y_k(\tau) \geq 0$ (Section 3.4) span orders of magnitude, so BTCache works with the standardized log risk $\tilde{Y}_k = (\log(\epsilon + Y_k) - m_k)/s_k$, where m_k and s_k are robust location and scale statistics on the training trajectories. The action value is

$$Q_k(s_t, a) = \mathbb{E}\big[\tilde{Y}_k(\tau) \mid s_t,\, a_t = a,\, a_{t+1:T-1} \sim \pi\big], \tag{1}$$

where π is the continuation policy represented in the training rollouts (Section 3.5). All predictions, comparisons, and scores below use this unit, so the selector compares expected log risks.

3.2 CAUSAL STATE AND CACHE REPRESENTATION

The predictor receives $s_t = (\kappa, g_t, \{c_{t,g}\}_{g=1}^G, m_t)$: the text condition and generation configuration κ, a global summary g_t of the latent, latent drift, time, and remaining budget, the cache state $c_{t,g}$ of each group, and a recurrent memory m_t. All entries exist before a_t is chosen, and fresh current-step residuals, future latents, and reference outputs are excluded.

For block ℓ, BTCache stores the complete-block residual $r_{t,\ell} = x^{\text{out}}_{t,\ell} - x^{\text{in}}_{t,\ell}$. A FULL group recomputes and stores its residuals, and a CACHE group replays them, refreshed from its two most recent evaluations $t_0 < t_1$:

$$x^{\text{out}}_{t,g} = x^{\text{in}}_{t,g} + r_{t_1,g} + \lambda_g \tfrac{t-t_1}{t_1-t_0}\big(r_{t_1,g} - r_{t_0,g}\big), \tag{2}$$

where $r_{t,g}$ sums the residuals of group g and the per-group damping $\lambda_g \in [0,1]$ is calibrated offline, with $\lambda_g = 0$ giving plain reuse. A cached group does not read its input, so replaying the group sum equals per-block replay, and the refresh adds no Transformer computation.

Each $c_{t,g}$ holds sketches and norms of the cached residual and of the last fresh-minus-stale innovation, the age $\alpha_{t,g} = t - t_g^{\text{lastFULL}}$, latent-to-cache distance, recent actions, and uncertainty statistics, with separate conditional and unconditional summaries under classifier-free guidance. A shared encoder with a learned depth identity maps each group to a 128-dimensional token. The memory

$m_t = \mathrm{GRU}(o_{t-1}, m_{t-1}) \in \mathbb{R}^{128}$ accumulates the previous mask, observed innovations, ages, latent drift, and consumed compute, and never receives terminal or reference information (Appendix C).

3.3 Causal Joint Quality Predictor

A learned FULL/CACHE embedding is added to each group token before cross-group mixing, $z_{t,g}(a_t) = E_{\mathrm{group}}(c_{t,g}) + P_{\mathrm{id}} e_g^{\mathrm{id}} + e_{a_{t,g}}^{\mathrm{act}}$, so the value of caching one group depends on which others are refreshed. The four group tokens and a global token pass through a two-layer, width-128 Transformer, whose pooled output is fused with m_t and mask, cost, and next-age metadata by an MLP, and all 16 masks are scored in one batch. Each output decomposes as

$$Q_k(s_t, a_t) = Q_k^{\mathrm{prior}}(p_t, a_t) + \Delta Q_k^{\mathrm{state}}(s_t, a_t), \tag{3}$$

where p_t holds the normalized time, remaining horizon, budget, and consumed compute. The prior, a small network over p_t, the mask bits, their pairwise products, and the cost, captures the time–action structure shared by all prompts. The state correction sees the complete causal representation and has a zero-initialized output layer, so the predictor starts from the prior and the correction models prompt- and trajectory-specific residuals.

3.4 Terminal Vector Risk

Distributional heads $H_k : 128 \to 64 \to (\mu_k, \log \sigma_k^2)$ on a shared trunk predict a Gaussian over $\tilde{Y}_k$ for $k \in \mathcal{K}_\nu$, and μ_k is the estimate $\widehat{Q}_k$ of Equation (1). The raw risks are

$$y_{\mathrm{latent}} = \frac{\|z_{\mathrm{final}}^{\mathrm{cache}} - z_{\mathrm{final}}^{\mathrm{FULL}}\|_2}{\|z_{\mathrm{final}}^{\mathrm{FULL}}\|_2 + \epsilon}, \qquad y_{\mathrm{perc}} = \frac{1}{N_f} \sum_{f=1}^{N_f} \mathrm{LPIPS}(V_f^{\mathrm{cache}}, V_f^{\mathrm{FULL}}), \tag{4}$$

and, for video with frame differences $\Delta V_f = V_{f+1} - V_f$,

$$y_{\mathrm{temp}} = \frac{\sum_{f=1}^{N_f - 1} \|\Delta V_f^{\mathrm{cache}} - \Delta V_f^{\mathrm{FULL}}\|_2^2}{\sum_{f=1}^{N_f - 1} \|\Delta V_f^{\mathrm{FULL}}\|_2^2 + \epsilon}. \tag{5}$$

Images train only the latent and perceptual heads. LPIPS (Zhang et al., 2018), all-FULL latents, and decoded outputs are used only to build labels. Inference neither decodes a reference nor evaluates LPIPS.

The heads are not collapsed into a weighted score. Each is compared with the risk of the *timestep comparator* π_{step}, a frozen equal-budget whole-step schedule with the same cache operator, fixed before the predictor is trained. A separate evaluator predicts its risks $V_k^{\mathrm{step}}(s_{\mathrm{init}})$ once, from the state s_{init} after the mandatory prefix, and holds them fixed, so the reference cannot drift after cache error enters the trajectory. The step-level generator is never executed at deployment.

3.5 Offline Supervision and Calibration

Training uses deterministic action-matched trajectories: all-FULL references, same-prefix branches over joint masks, mixed multi-action rollouts, and equal-budget timestep generations, each continued to termination so that masks exposed from the same state receive their own terminal targets. A bootstrap predictor is fitted first, and its frozen selector then collects policy-visited states and branches, which aligns the continuation policy π of Equation (1) with the deployed selector. Each active head minimizes a heteroscedastic Gaussian negative log likelihood on $\tilde{Y}_k$ plus a same-state pairwise ranking loss. Auxiliary heads predict the local innovation and the next error state for FULL groups. Five independently initialized predictors form an ensemble, and the comparator network predicts $\widehat{V}_k^{\mathrm{step}}$ from the same trajectories. For member i, the paired excess of mask a is $d_k^{(i)}(s_t, a) = \widehat{Q}_k^{(i)}(s_t, a) - \widehat{V}_k^{\mathrm{step},(i)}(s_{\mathrm{init}})$, and $\bar{d}_k$ is its ensemble mean. The ensemble scores 5×16 rows in one batch per free step. The damping λ_g, the reference plan π_0, and the weight β of Section 3.6 are calibrated on the calibration prompts, which are disjoint from the test prompts.

3.6 EXACT-BUDGET SELECTION

Consider a suffix of R free steps with ages $\boldsymbol{\alpha}$, maximum cache age A, and exactly F FULL cells left.

Proposition 1 (exact suffix feasibility). The suffix admits a schedule with exactly F FULL cells and no age above A if and only if

$$\sum_{g=1}^{G} \max\bigl(0, \lceil (R + \alpha_g - A)/(A+1) \rceil\bigr) \le F \le GR. \tag{6}$$

Necessity follows from $R - f_g \le A - \alpha_g + A f_g$ for the f_g refreshes of each group. For sufficiency, each group's lower-bound number of refreshes breaks every CACHE run at length A, and turning any remaining CACHE cell into FULL preserves feasibility (Appendix B).

A candidate a belongs to the feasible set $\mathcal{F}_t$ if, after its cost and age update, every age is at most A and Equation (6) holds for the remaining free steps, FULL cells, and ages. The test costs $O(G2^G)$ per decision and agreed with direct reachability on all 2,994,047 parameterized states we enumerated. The selector executes

$$a_t^\star = \arg\min_{a \in \mathcal{F}_t} J_t(a), \qquad J_t(a) = \max_{k \in \mathcal{K}_\nu} \bar{d}_k(s_t, a) + \beta \left\| a \oplus \pi_0(t) \right\|_1, \tag{7}$$

with ties broken by mask index. The first term is the predicted terminal risk relative to π_{step}. Taking the worst head keeps the latent, perceptual, and temporal risks separate, so a mask ranks well only if none of them exceeds the comparator by more than the alternatives. The reference plan π_0 is the equal-budget timestep schedule with the lowest mean terminal risk on the calibration prompts. It is chosen after training and need not coincide with π_{step}. It fixes the temporal structure shared across prompts, and β, in standardized log-risk units, is the reduction of the first term, from prior and state correction together, that a mask must achieve per deviating group. Because the budget and ages are checked on the whole suffix, $\mathcal{F}_t$ is never empty once the initial state is feasible, and every sample spends exactly the prescribed budget.

4 EXPERIMENTS

4.1 EXPERIMENTAL PROTOCOL

Setup. We evaluate HunyuanVideo (Kong et al., 2024) (544×960, 129 frames, embedded guidance 6.0, flow shift 7.0), Wan2.1-T2V-1.3B (Team Wan, 2025) (832×480, 81 frames, guidance 5), and FLUX.1-dev (Black Forest Labs, 2024) (1024×1024, guidance 3.5), all with $T = 50$ steps, $G = 4$, mandatory FULL steps $\{0, 1, 49\}$, and $A = 10$. The budget is 12 NFE (48 of the 200 group–step cells) on the two video models and 8 NFE (32 cells) on FLUX.1-dev. This bounds the block computation at $1/4.17$ and $1/6.25$ of all-FULL inference. Measured speedups are lower because text encoding, VAE decoding, per-step input and output layers, the residual refresh, and the controller are not cached. Of the 36 (video) and 20 (FLUX) free cells, Equation (6) requires 16 as minimum refreshes, four per group, so 20 and 4 cells can move across groups (Appendix B). Calibration and test prompts are disjoint: three and nine prompts on HunyuanVideo, three prompts and eight prompt–seed pairs on Wan2.1, and ten prompts from a separate fitting set and the 40 DrawBench prompts on FLUX.1-dev. Calibrated quantities are fixed across all test prompts and seeds.

Baselines and metrics. We compare with ToCa (Zou et al., 2025), DuCa (Zou et al., 2026), DiCache (Bu et al., 2025), TeaCache (Liu et al., 2025a), TaylorSeer (Liu et al., 2025b), EasyCache (Zhou et al., 2025), ERTACache (Peng et al., 2026), LeMiCa (Gao et al., 2025), and MagCache (Ma et al., 2025), each at the operating point listed in the tables and with identical prompts, seeds, and generation settings. On FLUX.1-dev, MagCache is reported both at its published setting and at the closest attainable cost to BTCache. Video quality is measured by VBench (Huang et al., 2024) and image quality by ImageReward (Xu et al., 2023). PSNR (Horé & Ziou, 2010), SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) are measured against the same-seed all-FULL output and averaged over frames for video. Latency is the wall-clock generation time on one NVIDIA A800 GPU, including the controller, and speed is its ratio to all-FULL inference. Executed masks are logged to verify the exact budget.

Method	Acceleration		Visual Quality			
	Latency (s) ↓	Speed ↑	VBench ↑	LPIPS ↓	SSIM ↑	PSNR ↑
Original: 50 steps	320.85	1.00×	80.29%	–	–	–
ToCa (Zou et al., 2025) ($\mathcal{N} = 5$)	110.26	2.91×	79.34%	0.459	0.585	15.76
DuCa (Zou et al., 2026) ($\mathcal{N} = 5$)	104.85	3.06×	79.41%	0.458	0.590	15.79
DiCache (Bu et al., 2025) ($\delta = 3.0$)	96.93	3.31×	70.59%	0.591	0.486	**18.87**
TeaCache (Liu et al., 2025a) ($l = 0.39$)	97.22	3.30×	79.32%	0.412	**0.625**	17.09
TaylorSeer (Liu et al., 2025b) ($\mathcal{N} = 7, O = 1$)	95.49	3.36×	79.43%	0.483	0.590	15.11
EasyCache (Zhou et al., 2025) ($\delta = 0.045$)	93.0	3.45×	79.02%	0.497	0.592	15.09
BTCache (12 NFE)	**88.63**	**3.62×**	**79.44%**	**0.410**	0.613	17.23

Table 1: Acceleration and visual-quality comparison on HunyuanVideo. Arrows indicate the preferred direction and bold marks the best accelerated method in each column. All entries are measured in our NVIDIA A800 evaluation.

Method	Acceleration		Visual Quality			
	Latency (s) ↓	Speed ↑	VBench (%) ↑	PSNR ↑	SSIM ↑	LPIPS ↓
Original: 50 steps	179.48	1.00×	81.29	–	–	–
EasyCache (Zhou et al., 2025) ($\delta = 0.13$)	56.26	3.19×	79.44	13.64	0.4711	0.4425
TaylorSeer (Liu et al., 2025b) ($N = 4, O = 2$)	52.02	3.45×	79.12	14.79	0.445	0.4430
TeaCache (Liu et al., 2025a) ($\delta = 0.2$)	50.56	3.55×	79.97	18.02	**0.6799**	0.3011
BTCache (12 NFE)	**50.13**	**3.58×**	**80.22**	**18.37**	0.6798	**0.3002**

Table 2: Acceleration and visual-quality comparison on Wan2.1-T2V-1.3B. All entries are measured in our NVIDIA A800 evaluation.

4.2 Main Results

HunyuanVideo. In Table 1, TeaCache is the strongest baseline in perceptual fidelity (3.30×, 0.412 LPIPS), while the faster TaylorSeer and EasyCache (3.36× and 3.45×) fall to about 15.1 dB PSNR and 0.48–0.50 LPIPS. DiCache attains the highest PSNR (18.87 dB), but its VBench drops to 70.59% and its LPIPS rises to 0.591, which indicates smooth, low-detail frames that stay close in pixel space but differ perceptually. BTCache is the fastest method (3.62×, 88.63 s) and attains the best VBench (79.44%) and LPIPS (0.410). Relative to TeaCache it is 8.59 s faster and improves PSNR by 0.14 dB at 0.012 lower SSIM. Relative to EasyCache, the fastest baseline, it improves VBench by 0.42 points, PSNR by 2.14 dB, and LPIPS by 0.087.

Wan2.1. On Wan2.1 (Table 2), the original model takes 179.48 s with VBench 81.29%. EasyCache runs at 56.26 s (3.19×) with VBench 79.44%, 13.64 dB PSNR, 0.4711 SSIM, and 0.4425 LPIPS. TaylorSeer runs at 52.02 s (3.45×) with 79.12%, 14.79 dB, 0.445, and 0.4430, and TeaCache at 50.56 s (3.55×) with 79.97%, 18.02 dB, 0.6799, and 0.3011. BTCache is the fastest at 50.13 s (3.58×) and reaches 80.22% VBench, 18.37 dB PSNR, 0.6798 SSIM, and 0.3002 LPIPS. At almost the same cost as TeaCache it improves VBench by 0.25 points and PSNR by 0.35 dB while matching its SSIM and LPIPS, and it improves PSNR by 3.58–4.73 dB and LPIPS by about 0.14 over EasyCache and TaylorSeer.

FLUX.1-dev. On FLUX.1-dev (Table 3), BTCache is the fastest method (3.37×, 8.21 s) with the highest PSNR (24.24 dB) and ImageReward (0.987). At the closest matched cost it gains 1.47 dB over MagCache and 6.19 dB over TeaCache, with LPIPS within 0.0003 and SSIM within 0.0006. MagCache at its published setting has the best LPIPS and SSIM but is 25% slower and 3.35 dB lower in PSNR. BTCache also loses the least ImageReward relative to all-FULL inference (0.043), so its fidelity gain does not cost text alignment.

Qualitative results. In Figures 3 and 4, fast baselines alter the panda's pose and blur the wave crest on HunyuanVideo and change the composition of the painting-style scene on Wan2.1. BTCache keeps these fine structures and locally moving regions, where the shallow group is most sensitive (Figure 1c), by refreshing that group without recomputing the full stack.

Method	Efficiency		Visual Quality			
	Latency (s)↓	Speedup↑	LPIPS↓	SSIM↑	PSNR↑	ImageReward↑
Original: 50 steps	27.7	1.00×	–	–	–	1.030
ERTACache (Peng et al., 2026)	9.23	3.00×	0.3213	0.7534	20.34	0.9237
TaylorSeer (Liu et al., 2025b)	10.07	2.75×	0.4498	0.6447	16.09	0.8851
MagCache (Ma et al., 2025) (published)	10.29	2.69×	**0.2505**	**0.7851**	20.89	0.9540
DiCache (Bu et al., 2025)	10.95	2.53×	0.3290	0.7701	21.67	0.8080
LeMiCa (Gao et al., 2025)	11.17	2.48×	0.3145	0.7388	18.41	0.9747
TeaCache (Liu et al., 2025a)	8.445	3.28×	0.3221	0.7356	18.05	0.9703
MagCache (Ma et al., 2025) (matched)	8.47	3.27×	0.3218	0.7364	22.77	0.9710
BTCache (8 NFE)	**8.21**	**3.37**×	0.3218	0.7362	**24.24**	**0.9870**

Table 3: Acceleration and visual-quality comparison on FLUX.1-dev over the 40 DrawBench prompts. MagCache is listed at its published setting and at the closest attainable cost to BTCache. Bold marks the best accelerated method in each column.

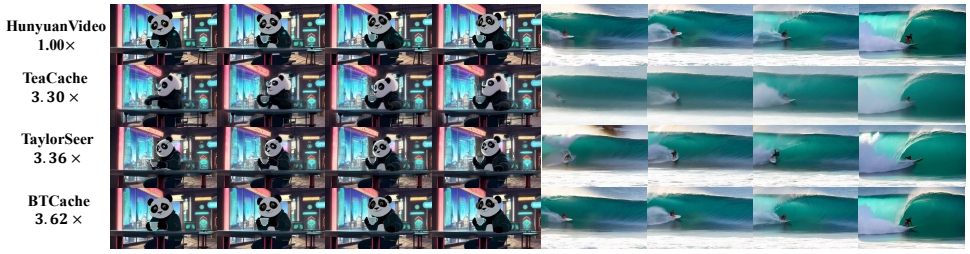


Figure 3: Qualitative comparison on HunyuanVideo. Rows show the all-FULL original and the accelerated methods, and columns are temporally ordered frames from the same prompt and seed. Labels report the measured end-to-end speedup.

4.3 Ablation Study

Each variant in Table 4 changes one component at the same 12-NFE budget and runs within $0.02\times$ of the full method.

Variant	VBench (%) ↑	PSNR ↑	SSIM ↑	LPIPS ↓	Latent err. ↓
BTCache (full)	**79.44**	17.23	**0.613**	**0.410**	0.287
(a) Timestep-level action space	79.40	17.08	0.608	0.415	0.295
(b) Local-error supervision	79.21	16.54	0.589	0.433	0.324
(c) Without state correction	79.31	16.93	0.602	0.421	0.301
(d) Without reference plan ($\beta = 0$)	79.36	17.02	0.606	0.418	0.293
(e) Latent head only in J_t	79.12	**17.26**	0.610	0.423	**0.285**
(f) Plain residual reuse ($\lambda_g = 0$)	79.27	16.71	0.597	0.426	0.312

Table 4: Ablation on HunyuanVideo over the nine test prompts at the 12-NFE budget. Latent err. is the final relative latent deviation from the same-seed all-FULL output. The full method is the operating point of Table 1.

Block–time granularity. Variant (a) restricts the actions to $\mathcal{A}_{\text{step}}$. The 36 free FULL cells can then only be spent as nine all-FULL steps: the variant keeps the choice of *when* to recompute and loses only the choice of *where in depth*. Latent error rises by 2.8% and PSNR drops by 0.15 dB, with the full method ahead on seven of nine prompts. The gap is modest because (a) shares the predictor, plan, and selector and thus places its whole-step refreshes well. It only cannot refresh the shallow group without paying for the other three.

Terminal versus local supervision. Variant (b) trains the heads on the immediate residual discrepancy $\|r_{t,g}^{\text{fresh}} - r_{t,g}^{\text{cache}}\|$ instead of the terminal risk. This is the largest degradation: latent error



Figure 4: Qualitative comparison on Wan2.1-T2V-1.3B with the layout of Figure 3.

rises by 12.9%, PSNR drops by 0.69 dB, and LPIPS rises by 0.023. Across masks at the same state, immediate discrepancy and terminal latent risk have a Spearman correlation of only 0.22, because local error ignores both the amplification by later steps and the cross-group interaction of Figure 1d.

Prior, state correction, and reference plan. Without ΔQ^{state}, variant (c) depends only on time, budget, and consumed compute. It still allocates block–time actions but loses 0.30 dB PSNR, because a fixed time–action pattern cannot follow prompt-specific motion. Variant (d) drops the plan term ($\beta = 0$) and loses 0.21 dB: the predictor is least reliable on masks far from the plan, where few training branches exist, and β admits such masks only when their predicted gain is large.

Worst-head selection and residual refresh. Ranking by the latent head alone (e) gives the lowest latent error and the highest PSNR but worsens LPIPS by 0.013 and VBench by 0.32 points, so the worst-head score trades a small latent loss for better perceived video. Plain reuse (f) loses 0.52 dB. On FLUX.1-dev it lowers PSNR from 24.24 to 23.13 dB while saving only 0.22 s, so the refresh complements the schedule rather than producing its gain.

Allocation. The gain over the timestep-level variant grows from 1.6% to 3.8% in latent error as the budget shrinks from 16 to 10 NFE. BTCache executes a partial mask at two thirds of the free steps, gives the shallowest group the most FULL cells as in Figure 1c, and spends about 2% of the latency on the controller (Appendix D).

5 Limitations and Conclusion

BTCache allocates a prescribed compute budget over the joint block–time space through the feasible joint action of lowest predicted terminal risk. It is the fastest compared cache method on three backbones with the best perceptual quality on both video models, and ablations attribute the gain to block–time allocation and terminal supervision. Its limitations are per-backbone training from terminal rollouts, a hand-fixed partition, and risk control in expectation rather than per trajectory.

AI use statement

We used large language model assistants for language editing, LaTeX formatting, and checking the consistency of notation and reported numbers. The research ideas, method design, implementation, and experiments are the work of the authors. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the content of this paper.

Ethics statement

This work accelerates inference of existing text-to-image and text-to-video models and does not introduce new generative capabilities, datasets, or human subject studies. Faster generation lowers the cost of producing synthetic media, including potentially misleading content. BTCache preserves the outputs of the underlying model and therefore inherits, but does not extend, its safeguards and risks.

Reproducibility statement

Sections 3.1 and 4.1 fix the group partition, budgets, mandatory steps, and age limit. Appendix B proves Proposition 1, Appendix C defines the causal features and predictor hyperparameters, and Algorithm 1 specifies the sampling loop. All baselines are run with the operating points listed in the tables. We will release the controller code, the calibrated quantities, and the prompt splits.

References

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

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. *arXiv preprint arXiv:2508.17356*, 2025.

Fanpu Cao, Yaofo Chen, Zeng You, and Wei Luo. ProCache: Constraint-aware feature caching with selective computation for diffusion transformer acceleration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43632–43642, 2026a.

Hanshuai Cui, Zhiqing Tang, Zhifei Xu, Zhi Yao, Wenyi Zeng, and Weijia Jia. Bwcache: Accelerating video diffusion transformers through block-wise caching. In *International Conference on Learning Representations*, 2026b.

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

Huanlin Gao, Ping Chen, Fuyuan Shi, Ruijia Wu, Yantao Li, Qiang Hui, Yuren You, Ting Lu, Chao Tan, Shaoan Zhao, Zhaoxiang Liu, Fang Zhao, Kai Wang, and Shiguo Lian. Meancache: From instantaneous to average velocity for accelerating flow matching inference. In *International Conference on Learning Representations*, 2026.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Alain Horé and Djemel Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, 2010. doi: 10.1109/ICPR.2010.579.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21807–21818, 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*, 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.

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

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*, 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*, 2025b.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Fei Ren, Shaobo Wang, Kaixin Li, and Linfeng Zhang. SpeCa: Accelerating diffusion transformers with speculative feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025c.

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*, 2024.

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*, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

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.

Team Wan. 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. doi: 10.1109/TIP.2003.819861.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and evaluating human preferences for text-to-image generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 15903–15935, 2023.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

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

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

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In *International Conference on Learning Representations*, 2025.

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.

A Sampling Algorithm

Algorithm 1 summarizes one generation with BTCache. Lines 2, 4–7, and 9 are the additions to standard sampling. Line 2 runs once per generation and the others once per free step, and their cost is included in all reported latencies.

Algorithm 1 BTCache sampling with an exact group-evaluation budget

Require: prompt, budget $B_0 = nG$, mandatory FULL steps, age limit A, predictor ensemble, comparator, plan π_0, weight β

1: run the mandatory prefix with all groups FULL; set $m_t \leftarrow 0$
2: predict $V_k^{\mathrm{step}}(s_{\mathrm{init}})$ once for all $k \in \mathcal{K}_\nu$
3: **for** each free step t **do**
4: build s_t from the latent, cache states, and m_t
5: score all 2^G masks in one batch and form $\bar{d}_k(s_t, a)$
6: $\mathcal{F}_t \leftarrow$ masks whose updated ages are at most A and whose suffix satisfies Equation (6)
7: $a_t^\star \leftarrow \arg\min_{a \in \mathcal{F}_t} J_t(a)$ (Equation (7))
8: recompute FULL groups and replay CACHE groups by Equation (2)
9: update ages, budget, and m_{t+1} from the observed innovations
10: **end for**
11: run the final step with all groups FULL and decode

B Proof of Proposition 1

Fix a group g with current age $\alpha_g \le A$ and let f_g be the number of FULL cells assigned to it in the R free steps. A FULL cell resets the age to zero and every CACHE cell increases it by one. The first CACHE run therefore has length at most $A - \alpha_g$, and each run that follows a refresh has length at most A. The group caches in $R - f_g$ steps, so every valid schedule satisfies

$$R - f_g \le (A - \alpha_g) + A f_g \quad \Longleftrightarrow \quad f_g \ge \frac{R + \alpha_g - A}{A + 1}. \tag{8}$$

Since f_g is a nonnegative integer, $f_g \ge f_g^{\min} = \max(0, \lceil (R + \alpha_g - A)/(A+1) \rceil)$, and summing over groups gives the left inequality of Equation (6). The right inequality holds because the suffix contains GR cells.

For sufficiency, place the $f_g^{\min}$ refreshes of group g at the steps $A - \alpha_g + 1 + j(A+1)$, $j = 0, \ldots, f_g^{\min} - 1$. Every CACHE run then ends before the age exceeds A, and the schedule uses $\sum_g f_g^{\min} \le F$ FULL cells. Because $F \le GR$, the remaining $F - \sum_g f_g^{\min}$ FULL cells can be placed on arbitrary CACHE cells. Turning a CACHE cell into a FULL cell only resets an age, so the schedule stays valid and uses exactly F FULL cells. □

In the main setting, the free segment has $R = 47$ steps and all ages are zero after the mandatory prefix. With $A = 10$, each group needs $\lceil 37/11 \rceil = 4$ refreshes, so at least 16 of the free cells are fixed in number, though not in position. The 12-NFE video budget leaves $F = 36$ free cells, 20 of which may be assigned to any group, and the 8-NFE FLUX budget leaves $F = 20$, four of which may be assigned to any group. The selector decides the positions of all of them.

C Causal Features and Predictor

Group state. BTCache stores each block residual after the complete attention, MLP, and residual paths. Conditional and unconditional guidance branches are summarized separately. For guidance-distilled models such as FLUX.1-dev and HunyuanVideo with embedded guidance, only one branch exists and the second half of the vector is marked unavailable. Per branch, every residual uses a 32-dimensional normalized signed spatial sketch, the log ℓ_2 norm, the log mean absolute value, the log sketch norm, and an availability bit. These 36-dimensional vectors are pooled within a group, and concatenating both branches gives 72 dimensions. The last observed fresh-minus-stale innovation uses the same 72-dimensional summary. Cache age, latent-to-cache distance, five action and

innovation history scalars, and three missingness-aware uncertainty scalars yield a 154-dimensional group state. Continuous features are standardized on the training split, and unavailable innovations are represented by a zero-valued standardized feature plus an explicit missingness indicator.

Predictor. The group encoder is a two-layer MLP ($154 \to 128 \to 128$) shared across groups, followed by the learned depth identity and the FULL/CACHE embedding. The cross-group module is a two-layer Transformer with width 128 and four heads over the four group tokens and one global token. The memory is a single-layer GRU with 128 hidden units. The prior network takes the normalized time, remaining horizon, remaining budget, consumed compute, the mask bits, their six pairwise products, and the cost. Each risk head maps the fused 128-dimensional feature to 64 hidden units and then to a mean and a log variance. The comparator network shares this architecture without the action inputs. Candidate and comparator ensemble members share the same initialization index and data split, so the paired excesses $d_k^{(i)}$ are formed within one member before averaging.

D Allocation Analysis

Budget. Table 5 compares BTCache with the timestep-level variant from 10 to 16 NFE on HunyuanVideo. BTCache attains the lower latent error at every budget, and the relative reduction grows as the budget shrinks, from 1.6% at 16 NFE to 3.8% at 10 NFE. With a generous budget both schedules refresh most sensitive regions. Under a tight budget each whole-step refresh spends three quarters of its cost on groups that do not need it, and placing group evaluations individually matters more. The quality of BTCache degrades smoothly with the budget, so the single deployment control trades speed and fidelity predictably.

Where the budget goes. Table 6 shows that BTCache executes a partial mask at about two thirds of the free steps on both video models, so its schedules are not whole-step schedules in disguise. The shallowest group receives the largest share of the free FULL cells and the middle groups the smallest, and the deepest group receives a large share concentrated in the late steps. This allocation mirrors the measured sensitivity of Figure 1c, although the predictor never observes those measurements: it is learned from terminal outcomes alone.

Controller cost. The ensemble scores 5×16 masks in one batched call per free step and the feasibility test enumerates 16 masks in closed form. Together with state construction this costs 37 ms per free step on HunyuanVideo and 23 ms on Wan2.1, about 2% of the end-to-end latency, and adds less than 0.5 GB of peak memory. All latencies in Tables 1–3 include this cost.

Budget	Speed ↑	Latent err. ↓		Rel.	BTCache	
(NFE)		Timestep	BTCache	change	PSNR ↑	LPIPS ↓
10	4.23×	0.318	0.306	−3.8%	16.78	0.431
12	3.62×	0.295	0.287	−2.7%	17.23	0.410
14	3.16×	0.272	0.266	−2.2%	17.71	0.389
16	2.81×	0.251	0.247	−1.6%	18.20	0.370

Table 5: Budget sweep on HunyuanVideo. "Timestep" is variant (a) of Table 4 at the same budget. Speed is that of BTCache, and the timestep variant runs within $0.02\times$ at every budget.

Statistic	HunyuanVideo	Wan2.1
Free steps executing a partial mask	68.4%	63.1%
Free FULL cells in groups g1/g2/g3/g4	31/22/19/28%	29/24/20/27%
Controller time per free step	37 ms	23 ms
Controller time per generation	1.74 s (2.0%)	1.08 s (2.2%)
Additional peak memory	0.41 GB	0.36 GB

Table 6: Decisions and cost of the controller at the 12-NFE operating point, over the test prompts. Percentages of time are relative to the BTCache latency.

E NUMBER OF GROUPS

Table 7 varies G on HunyuanVideo at the 12-NFE budget. For every G the groups are contiguous and compute-balanced, and $G = 1$ coincides with variant (a) of Table 4. Moving from one to two and four groups lowers latent error from 0.295 to 0.292 and 0.287. Eight groups lower it further to 0.285, but the candidate set grows from 16 to 256 masks, the ensemble must score 5×256 rows per free step, and the number of same-state branches needed as labels grows by the same factor, and the speed falls from $3.62\times$ to $3.47\times$. Four groups retain most of the gain of finer control at a negligible controller cost, which is why we use $G = 4$ for all backbones.

G	Masks	Speed ↑	Latent err. ↓	LPIPS ↓	PSNR ↑
1	2	$3.63\times$	0.295	0.415	17.08
2	4	$3.64\times$	0.292	0.413	17.14
4	16	$3.62\times$	0.287	0.410	17.23
8	256	$3.47\times$	0.285	0.409	17.27

Table 7: Effect of the number of groups on HunyuanVideo at the 12-NFE budget.