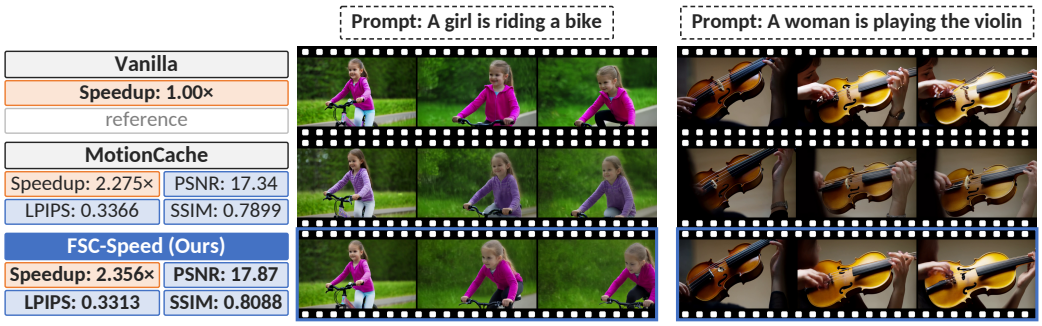
# WHEN TO SYNCHRONIZE, HOW MUCH TO REPAIR: CACHE CONTROL FOR AUTOREGRESSIVE VIDEO DIFFUSION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Autoregressive (AR) video generation makes long-horizon visual synthesis possible, but every generated chunk requires repeated forward passes through a large diffusion transformer (DiT). Training-free caching reduces this cost by reusing intermediate computation. However, existing cache controllers often couple two decisions: *when* the cache is synchronized exactly and *how much* approximate computation is spent in between. Our characterization suggests that these decisions should be separated. For reactive staleness signals, exact synchronization should be scheduled before measured drift accumulates, while measured drift should decide only how much repair to spend between synchronizations. Based on this separation, we introduce **Frequency-Step Cache (FSC)**, a training-free controller that performs exact synchronization at full-computation anchors on a dense-to-sparse schedule and adaptive repair between them. A lightweight latent-space risk combining frequency-resolved drift with token age routes each non-anchor step from whole-step replay to selective token repair or full repair without changing the synchronization schedule. We further show on SkyReels-V2 that whole-step reuse provides the main acceleration, while selective token computation is most effective as fidelity repair. In a controlled ablation, using one reactive signal for both synchronization and repair reduces fidelity at matched latency, with the separation advantage also observed on FramePack and Matrix-Game. Across multiple AR video backbones, FSC provides favorable speed–quality trade-offs. On SkyReels-V2, FSC-Speed reaches a $2.36\times$ speedup, above MotionCache's $2.275\times$, while improving SSIM by $0.019$ and PSNR by $0.5$ dB. At MotionCache's speed, FSC-Quality improves SSIM by $0.031$ and PSNR by $1.0$ dB, while lowering LPIPS by $0.024$. Without retuning, FSC-Quality transfers to the 14B SkyReels-V2 model at $2.12\times$ speedup, compared with $1.79\times$ for MotionCache.




Speedup, PSNR, SSIM and LPIPS are the benchmark averages of Table 1 (fidelity on the 102-prompt same-seed subset), not measurements of the two clips shown; the clips are same-seed examples at identical prompt and sampler settings.

**Visual quality comparison: FSC-Speed exhibits smaller deviations from the vanilla trajectory than MotionCache (Xu et al., 2026).** In particular, FSC better preserves the clothing color in the left example and the violin's appearance in the right example.

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.

# 1 INTRODUCTION

Autoregressive (AR) video generation has emerged as a practical paradigm for long-horizon visual synthesis, generating videos sequentially by conditioning each new chunk on previously generated content (Chen et al., 2024a; Yin et al., 2025; Chen et al., 2025; Sand.ai et al., 2025). This chunk-wise formulation makes long-video generation possible while keeping attention context and memory use bounded. However, each chunk still requires tens of denoising steps, which refine a noisy video representation through repeated evaluations of a large diffusion transformer (Peebles & Xie, 2023). Training-free caching reduces this cost by reusing intermediate outputs across nearby steps (Ma et al., 2024; Selvaraju et al., 2024; Liu et al., 2025a). Yet redundancy varies across steps and video content, making it difficult to decide when reuse remains accurate (Zou et al., 2025; Ma et al., 2026; Xu et al., 2026; Liu et al., 2025b). Reuse introduces approximation errors that can accumulate across steps and chunks (Cui et al., 2026a; Peng et al., 2026; Nan et al., 2026). Existing cache controllers often use reactive reuse signals to govern both when exact computation occurs and how much approximation is allowed in between. These two roles operate differently: synchronization establishes the reference state for subsequent reuse, whereas repair responds to staleness observed after reuse. Cache control therefore involves two distinct decisions. **(1) When should the cache be fully refreshed?** We call this *exact synchronization*: full computation updates cached outputs and establishes a reference for measuring subsequent changes in the video representation. **(2) How much should be recomputed between refreshes?** We call this *adaptive repair*: choosing how much computation to perform based on observed changes that indicate inaccurate reuse.

Our characterization motivates separating these decisions (Sec. 3). Reuse accuracy varies strongly by step, favoring scheduled synchronization (Fig. 1(a)). Approximate synchronization leaves errors in the cached outputs reused by later steps, while change-based measurements report deviations only after they occur (Fig. 1(d)). We therefore schedule exact synchronization independently of these measurements and use them only to allocate repair. In a controlled ablation, a purely reactive variant has lower fidelity at matched latency on all eight development prompts (Sec. 5.3). Between synchronizations, repair must balance accuracy against execution cost. On SkyReels-V2, selectively computing video tokens, the model's units of video representation, yields much less speedup than the computation reduction suggests (Fig. 1(c)). This favors reusing an entire step's cached outputs for acceleration and selective computation for repair. To guide repair, a lightweight score based on changes in the video representation tracks reuse error. The score reaches a Spearman rank correlation of $0.893$ with reuse error measured against full computation, as shown in Fig. 3.

Guided by these findings, **Frequency-Step Cache (FSC)** schedules full-computation steps, called *anchors*, progressively farther apart (Sec. 4.1). Between anchors, it reuses cached outputs for a whole step or repairs them by recomputing varying numbers of tokens or performing full computation (Sec. 4.3). The repair score combines low- and high-frequency changes, representing coarse structure and fine detail respectively, with token age, the steps since each token's last update (Sec. 4.2). These measurements control repair, never the anchor schedule. Our contributions are: (i) separating scheduled synchronization from measurement-driven repair, outperforming coupled control at approximately matched latency on SkyReels-V2, FramePack, and Matrix-Game; (ii) showing on SkyReels-V2 that whole-step reuse supplies acceleration while selective token computation is most effective as fidelity repair; and (iii) FSC, a training-free controller that combines scheduled exact anchors with lightweight frequency- and age-aware adaptive repair.

Experiments across multiple AR video backbones demonstrate favorable speed–quality trade-offs and consistently support the separation principle. On SkyReels-V2, FSC-Speed reaches a $2.36\times$ speedup, above MotionCache's $2.275\times$, while improving same-seed SSIM by $0.019$ and PSNR by $0.5$ dB, with comparable VBench quality. At MotionCache-matched speed, FSC-Quality improves SSIM by $0.031$, PSNR by $1.0$ dB, and lowers LPIPS by $0.024$.

# 2 RELATED WORK

**Autoregressive Video Generation.** Recent AR video models generate long videos as sequences of causally conditioned frames or chunks with diffusion backbones (Chen et al., 2024a; Yin et al., 2025), often trained with flow matching (Chen et al., 2025; Sand.ai et al., 2025). This chunk-wise

formulation enables long-horizon generation under bounded context and memory, but each chunk still requires repeated denoising computation.

**Training-Free Diffusion Caching.** Feature caching reduces repeated denoising computation by reusing activations or residuals across nearby steps, using fixed schedules (FORA (Selvaraju et al., 2024), $\Delta$-DiT (Chen et al., 2024b)), adaptive timing (TeaCache (Liu et al., 2025a), DiCache (Bu et al., 2026)), or forecasting (TaylorSeer (Liu et al., 2025c)). Token-wise methods instead combine reuse with selective recomputation: ToCa (Zou et al., 2025) uses a predetermined refresh cycle with attention- and age-based token ranking, DuCa (Zou et al., 2026) alternates reuse and token recomputation, and ProCache (Cao et al., 2026) uses an offline-searched reuse schedule. ToCa studies token-wise caching in diffusion transformers but not chunk-autoregressive long-video cache control. It also fixes both the refresh cycle and token-recomputation budget a priori; attention and cache age rank which tokens receive that budget, whereas FSC adapts the repair amount online while keeping synchronization scheduled.

**Caching for Autoregressive Video.** FlowCache (Ma et al., 2026) assigns chunk-level reuse policies, MotionCache (Xu et al., 2026) introduces motion-aware token adaptivity, and ARCache (Nan et al., 2026) determines whole-step reuse from history-token changes. SCOPE (Cui et al., 2026b) expands the inter-step action space through prediction and selective frame computation. These methods adapt reuse to temporal redundancy, but do not explicitly formulate exact-synchronization timing and intermediate repair amount as separate control decisions.

**Frequency-Aware Cache Control.** Frequency-aware methods use spectral information to schedule whole-step refreshes (SeaCache (Chung et al., 2026)) or manipulate frequency components (FreqCa (Liu et al., 2025b), SpectralCache (Li, 2026)). PreciseCache (Wang et al., 2026a) uses low-frequency differences to decide when denoising computation can be skipped. FSC instead keeps exact synchronization timing independent of measured staleness: synchronization is scheduled, while frequency-resolved drift and cache age determine only inter-anchor repair.

## 3 Characterization and Observations

We first characterize where redundancy exists and where naive caching fails in controlled profiling experiments (App. A.1). The resulting observations motivate the design; full-scale autoregressive ablations of the deployed controller (Sec. 5) reproduce the relevant orderings.

**Observation 1: Cacheability is strongly step-dependent.** Reusing the previous step's residuals at early steps incurs approximately $2.4\times$ the error of reuse at late steps (Fig. 1(a)). Under a comparable anchor budget, dense-to-sparse placement yields the highest mean SSIM (Fig. 1(b); Sec. 5.3). *Design consequence:* anchors should be dense early and increasingly sparse later (Sec. 4.1).

**Observation 2: Sparse token computation gives limited speedup on SkyReels-V2.** Fig. 1(c) profiles true sparse kernels, which recompute a subset of tokens per step: at $q{=}0.10$ they reach $1.94\times$ against a compute-proportional $2.72\times$, mainly because full-context key/value handling and launch overhead do not shrink with token fraction. Their stale keys and values also cost $\approx 0.024$ SSIM relative to a dense-forward token-refresh quality reference (its $\approx 1.23\times$ is fixed by the full forward; App. A.3). Whole-step dense-to-sparse reuse reaches $2.62\times$. FSC's deployed token repair is a gathered-batch implementation whose cost is measured end to end. *Design consequence:* on this backbone, whole-step reuse should supply the acceleration, and token computation should primarily serve as fidelity repair (Sec. 4.3).

**Observation 3: Approximate anchors seldom improve upon exact anchors.** Only three of the 17 tested variants lie above the exact-anchor frontier (Fig. 1(d); App. A.4). Any approximation inside an anchor is inherited by every step until the next anchor. *Design consequence:* anchors should remain exact full computations (Sec. 4.1).

**Observation 4: Inter-anchor staleness is spatially heterogeneous.** Between anchors, tokens go stale at different rates. High-frequency reuse error grows faster in absolute terms than low-frequency (LF) error. It remains about 2–$2.5\times$ the LF error at cache ages 1–10, up to $2.7\times$ on individual prompts (Fig. 5 in App. A.5). Cache age complements drift: a token with small per-step drift can still become unreliable after many reuse steps. *Design consequence:* repair between anchors should follow measured staleness, proxied by frequency-resolved drift and cache age (Sec. 4.2).

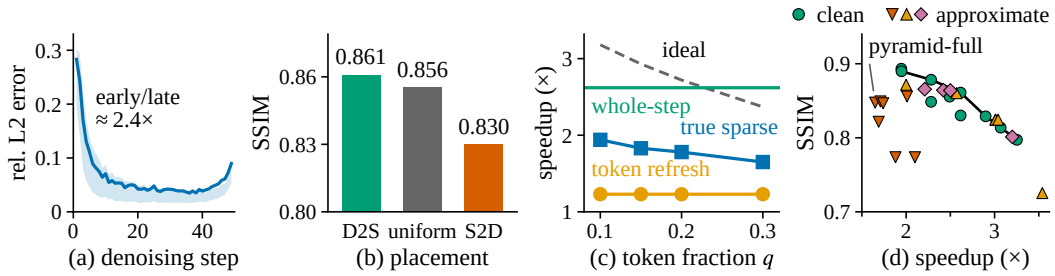



Figure 1: **Three limitations of naive caching: reuse error depends on the step, token-selective repair provides limited speedup, and approximate anchors rarely help (Observations 1–3).** SkyReels-V2, controlled profiling setting (App. A.1): (a) 14B; (b)–(d) 1.3B. Anchor counts differ because each panel is a separate controlled study. **(a)** Relative-$L_2$ error from reusing each block's previous-step residual (band: interquartile range over blocks and prompts). **(b)** SSIM vs. same-seed vanilla for three placements of 7 (8 for uniform) exact anchors: dense-to-sparse (D2S, 0.861) outperforms uniform (0.856) and sparse-to-dense (S2D, 0.830). **(c)** Speedup vs. token fraction $q$. True sparse kernels reach $1.94\times$ at $q$=0.10, far below their compute-proportional $2.72\times$; the dense-forward token-refresh reference stays at $\approx 1.23\times$ by design (dashed: its ideal). Whole-step D2S reuse reaches $2.62\times$. **(d)** SSIM vs. speedup for anchor policies ("clean" circles: exact anchors; "approximate": $\triangledown$ pyramid kernels, $\triangle$ block-skip gates, $\Diamond$ other variants). Of 17 approximate variants, only three block-skip gates near $3.0\times$ lie above the exact-anchor frontier (black line), by at most 0.008 SSIM. The pyramid-full variant has substantially lower SSIM than the clean anchor baseline.

**Summary:** Because reactive signals report drift only after it occurs, these observations motivate the following control principle: schedule exact synchronization in advance (Obs. 1, 3), while using measured drift only to allocate inter-anchor repair (Obs. 2, 4).

## 4 Frequency-Step Cache Design

FSC separates cache control into two decisions: exact synchronization determines *when* the cache is recomputed exactly, while adaptive repair determines *how much* residual computation is spent between synchronizations. Residual caching replays a transformer block's stored residual instead of recomputing it.

After the warmup, FSC recomputes all tokens in two situations that differ in timing. An *anchor* occurs at a step fixed before generation. At non-anchor steps, the staleness risk $\rho_s$ (Sec. 4.2) may select a *full repair*. The risk may trigger a full repair but never removes or delays an anchor. FSC maintains two states. The *residual cache state* stores each token's whole-stack residual, while keys and values are cached per block. The *anchor latent reference* $\mathbf{X}_{\text{ref}}$ holds the latent bands stored at the last anchor (Sec. 4.2). All drift is measured against it. An anchor replaces $\mathbf{X}_{\text{ref}}$; a full repair keeps it. Removing anchors and letting $\rho_s$ trigger every full synchronization after warmup lowers fidelity (Sec. 5.3).

Each step makes up to three decisions (Fig. 2; Algorithm 1 in App. B.7). First, warmup steps and anchors compute all tokens exactly. Second, at any other step, $\rho_s$ chooses how much to compute: replay ($C0$), token repair ($C1$–$C3$), or full repair. Third, if token repair is selected, accumulated drift and token age choose which tokens to recompute.

### 4.1 Dense-to-Sparse Anchor Schedule

Because cacheability grows along the trajectory and approximate anchors seldom improve upon exact anchors (Fig. 1(a,d)), FSC places exact anchors densely early and sparsely late. With warmup length $W$ (steps $s < W$ are fully computed), $n$ post-warmup anchors, and last eligible step $E$, anchor $j$ is placed at

$$a_j = \text{round}[\,W + c_j\,(E - W)\,], \qquad c_j = \frac{\sum_{i<j}(1+i)}{\sum_{i<n-1}(1+i)}, \qquad j = 0, \ldots, n-1. \tag{1}$$

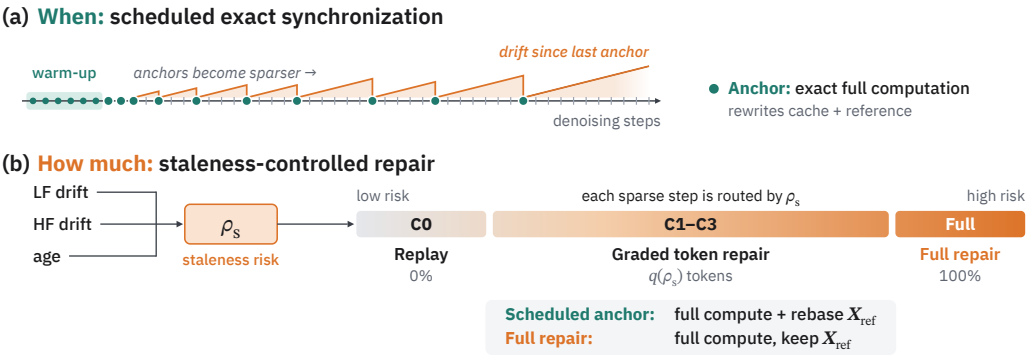



Figure 2: **Anchors are fixed before generation; full repairs are chosen by the risk $\rho_s$ and leave the schedule unchanged.** **(a)** *When*: dense warmup, then dense-to-sparse anchors. **(b)** *How much*: drift from $\mathbf{X}_{\text{ref}}$ and cache age form $\rho_s$, which selects replay ($C0$), token repair ($C1$–$C3$), or full repair. An anchor replaces $\mathbf{X}_{\text{ref}}$; a full repair keeps it.

The nominal anchor gaps grow linearly (1:2:3:$\cdots$; rounding in App. B.1).

### 4.2 Frequency- and Age-Aware Staleness Estimation

Between anchors, staleness is spatially non-uniform; FSC estimates it from the latent alone, without evaluating the transformer. Let $\mathbf{X}_s$ be the latent at step $s$. Its per-frame coarse and detail bands are

$$\mathbf{X}_{s,\text{LF}} = \text{Up}\big(\text{AvgPool}_{2\times 2}(\mathbf{X}_s)\big)\,, \qquad \mathbf{X}_{s,\text{HF}} = \mathbf{X}_s - \mathbf{X}_{s,\text{LF}}. \tag{2}$$

For token $i$ and band $\text{B} \in \{\text{LF}, \text{HF}\}$, drift from $\mathbf{X}_{\text{ref}}$ is

$$d^{\text{B}}_{s,i} = \frac{\|\mathbf{X}_{s,\text{B},i} - \mathbf{X}_{\text{ref},\text{B},i}\|_2}{\|\mathbf{X}_{\text{ref},\text{B},i}\|_2 + \varepsilon}. \tag{3}$$

Each token's age $a_{s,i}$ counts reuse steps since its last recompute. The staleness risk $\rho_s$ is the step-level score that chooses the repair level. It combines the 95th percentile of each band's drift with the mean normalized age,

$$\rho_s = w_{\text{HF}}\, Q_{0.95}\big(d^{\text{HF}}_{s,\cdot}\big) + w_{\text{LF}}\, Q_{0.95}\big(d^{\text{LF}}_{s,\cdot}\big) + w_{\text{age}}\, \frac{1}{N}\sum_{i=1}^{N} \frac{a_{s,i}}{k_{\text{tok}}}, \tag{4}$$

where $N$ is the number of tokens and $k_{\text{tok}}$ the maximum token age. The risk excludes accumulated drift, which only selects tokens (Sec. 4.3). We assess its association with the exact reuse error $E_s$, the 95th-percentile relative error of cached residuals, in an offline diagnostic on the eight development prompts (App. C). Over 4,896 non-anchor decisions, $\rho_s$ reaches Spearman 0.893 with $E_s$ (Fig. 3). As a detector of unsafe reuse, defined as $E_s > 0.05$, it reaches ROC-AUC 0.930 (Fig. 6).

### 4.3 Hierarchical Replay and Selective Repair

A non-anchor step is routed to one of five levels. Replay ($C0$) reuses all cached residuals. Token repair ($C1$–$C3$) recomputes increasing token fractions $q_m$: base fractions $\hat{q}_m$ times a common scale $q_{\text{scale}}$. Full repair (Full) recomputes all residuals. The level rises as $\rho_s$ passes the step thresholds

$$t_m(s) = b_m\, \varphi_m(s)\, r, \qquad m \in \{\text{skip}, 1, 2, \text{full}\}, \tag{5}$$

with base thresholds $b_m$, phase shaping $\varphi_m(s)$, and threshold scale $r$; $C0$ needs $\rho_s < t_{\text{skip}}$. Phase shaping tightens thresholds early and loosens them late. A token is overdue once its age reaches $k_{\text{tok}}$, and any overdue token forbids $C0$ (App. B.3). Token repair ranks tokens by accumulated high-frequency drift $A^{\text{HF}}_{s,i} = \sum_{s'} d^{\text{HF}}_{s',i}$ since the last recompute of token $i$ (App. B.2). The repaired tokens are the top fraction $q_m$ plus every overdue token:

$$\mathcal{S}_s = \text{Top}_{q_m}\big(\{A^{\text{HF}}_{s,i}\}_{i=1}^{N}\big) \;\cup\; \{\, i :\; a_{s,i} \geq k_{\text{tok}} \,\}. \tag{6}$$

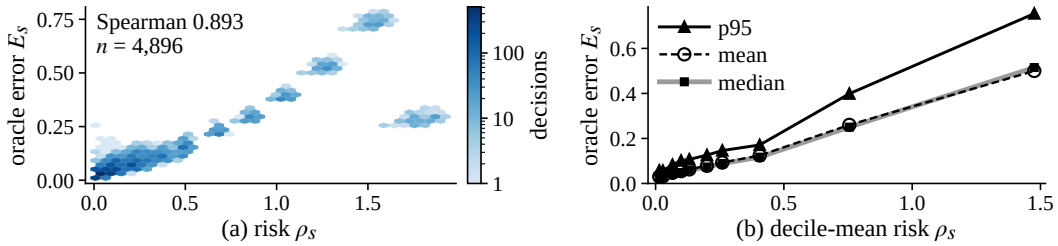



Figure 3: **The staleness risk $\rho_s$ is strongly associated with exact reuse error (Spearman $0.893$) in an offline diagnostic.** SkyReels-V2 1.3B, $4{,}896$ non-anchor decisions on the eight development prompts (App. C). **(a)** $\rho_s$ versus the oracle reuse error $E_s$; within each prompt the Spearman correlation is $0.909$–$0.972$. **(b)** Mean, median, and 95th-percentile $E_s$ per $\rho_s$ decile; mean and median rise monotonically.

At $C3$, an optional LF gate adds the top-$p_{\mathrm{LF}}$ tokens ranked by instantaneous low-frequency drift $d^{\mathrm{LF}}_{s,i}$. The tokens in $\mathcal{S}_s$ run through every block as a gathered batch that attends to all cached keys and values (App. B.4).

All parameter values are listed in App. D. On SkyReels-V2, FSC-Speed differs from FSC-Quality (Sec. 5.1) only in $r$, the primary speed–quality control parameter, and in the warmup $W$. On the full benchmark, FSC-Quality routes $70.3\%$ / $23.8\%$ / $5.8\%$ of non-anchor steps to $C0$ / $C1$–$C3$ / Full (FSC-Speed: $69.1\%$ / $25.2\%$ / $5.7\%$).

### 4.4 Backbone Instantiation and Overhead

On MAGI-1, FSC scales the native token-level reuse threshold, forces exact computation at anchors, and fully recomputes every $k_{\mathrm{tok}}$-th step (App. B.5). MAGI-1 therefore tests transfer through its native cache interface, not the frequency controller. The controller needs no auxiliary network and adds under $0.2\%$ latency (App. B.6).

## 5 Experiments

### 5.1 Experimental Setup

**Base Models and Baselines.** SkyReels-V2, FramePack (Zhang et al., 2025), and Matrix-Game-3.0 (Wang et al., 2026b) use the full FSC controller, whereas MAGI-1 tests transfer through its native cache interface (Sec. 4.4). SkyReels-V2-1.3B (Chen et al., 2025) generates 540P videos of 177 frames (two 97-frame chunks with 17 overlapping frames, 50 steps per temporal group; App. D). MAGI-1-4.5B-distill (Sand.ai et al., 2025) generates $720\times720$ videos of 168 frames (seven 24-frame chunks, 64 steps per chunk). We compare against vanilla, FlowCache (Ma et al., 2026) with chunk-level reuse, MotionCache (Xu et al., 2026) with token-level reuse in its released configuration, and our latency-calibrated SeaCache port. All share one stack, prompts, seed, checkpoints, and settings; SCOPE (Cui et al., 2026b) runs on its official stack. FlowCache runs with its KV-cache compression disabled. In a probe run, the official FlowCache stack without compression runs at $2.21\times$, close to our reproduction ($2.04\times$), at comparable fidelity (App. E). We found no public implementation of ARCache (Nan et al., 2026).

**Operating Points.** We report two operating points of the same controller. On SkyReels-V2, FSC-Quality matches MotionCache's latency and is the single configuration used for all SkyReels-V2 ablations, sensitivity and seed studies, and the 14B no-retuning transfer; FSC-Speed differs only in warmup length and threshold scale and is a faster point on the speed–quality frontier (App. E.1). On MAGI-1, both points are calibrated on its native cache interface and run faster than MotionCache. FramePack and Matrix-Game each use one per-backbone calibrated configuration (App. E.2).

**Evaluation Sets and Metrics.** Quality is measured with VBench-Long (Huang et al., 2024; 2025) on a fixed dimension set chosen with reference to prior autoregressive video caching protocols (Ma et al., 2026; Xu et al., 2026), with the official normalization and weights (App. D). Unless noted,

Table 1: **FSC-Quality matches MotionCache's speed with higher fidelity, and FSC-Speed is faster while keeping higher SSIM and PSNR.** Speed and PFLOPs: Bench (MAGI-1: speedups on the vanilla-clean prompt pool after per-method latency auditing, MotionCache and FSC PFLOPs on a 20-prompt sample); VBench: the 505 Bench prompts of the evaluated dimensions (500 on MAGI-1); PSNR/SSIM/LPIPS: Fid (110 on MAGI-1). PFLOPs come from executed-token counters, warmup forwards included, on a different basis per backbone; Mem: peak memory (both App. D). *Our latency-calibrated SeaCache port (App. E.3). †MAGI-1 tests transfer through its native cache interface (Sec. 4.4). Best accelerated result per backbone in **bold**, second best underlined.

| Model | Method | PFLOPs ↓ | Speedup ↑ | Latency (s) ↓ | Mem (GB) ↓ | VBench ↑ | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
|---|---|---|---|---|---|---|---|---|---|
| SkyReels-V2 | Vanilla | 37.7 | 1.000× | 367 | 53.7 | 0.7977 | – | – | – |
| | FlowCache | 17.4 | 2.037× | 180 | **58.1** | **0.7939** | 17.57 | 0.7946 | **0.3073** |
| | MotionCache | 15.4 | 2.275× | 161 | 58.9 | 0.7898 | 17.34 | 0.7899 | 0.3366 |
| | SeaCache-AR* | 14.9 | 2.320× | 158 | 58.6 | 0.7886 | 17.52 | 0.7919 | 0.3331 |
| | FSC-Quality (ours) | 14.8 | 2.275× | 161 | 58.4 | 0.7900 | **18.38** | **0.8209** | 0.3122 |
| | FSC-Speed (ours) | **14.1** | **2.356**× | **156** | 58.4 | 0.7892 | 17.87 | 0.8088 | 0.3313 |
| MAGI-1 | Vanilla | 140.6 | 1.000× | 684 | 31.8 | 0.7969 | – | – | – |
| | FlowCache | **82.5** | **1.486**× | **461** | 19.7 | **0.7991** | 16.64 | 0.7782 | 0.3559 |
| | MotionCache | 90.4 | 1.357× | 504 | 19.7 | 0.7950 | 18.76 | 0.8468 | 0.2778 |
| | FSC-Quality (ours)† | 86.8 | 1.421× | 482 | 19.7 | 0.7954 | **20.49** | **0.8883** | **0.2297** |
| | FSC-Speed (ours)† | 84.6 | 1.457× | 470 | 19.7 | 0.7896 | 20.00 | 0.8782 | 0.2460 |

all timings use one NVIDIA RTX PRO 6000 GPU per video on otherwise idle devices. Fidelity is PSNR, SSIM, and LPIPS (Zhang et al., 2018) against the same-seed vanilla output. VBench and same-seed fidelity are complementary: VBench measures reference-free video quality, whereas fidelity measures the approximation error that caching introduces relative to the unmodified generator. Same-seed fidelity requires vanilla references and is therefore evaluated on subsets. The evaluation sets are:

| Set | Prompts | Selection | Used for |
|---|---|---|---|
| Bench | 946 | full VBench-Long benchmark | speed, compute, VBench |
| Fid | 102 | first ten of each prompt set, minus Dev | fidelity in Table 1 (MAGI-1: all 110) |
| Paired | 235 | stratified subset, minus Dev | equal latency (Table 2(a), Fig. 4(a)) |
| Dev | 8 | our development set (App. D) | ablations, sensitivity, sweep, Fig. 3 |
| Cross | 40 | per backbone, never used for calibration | cross-family separation test (Sec. 5.3) |
| 14B | 47 | five per prompt set, minus Dev | SkyReels-V2 14B (Table 2(b)) |

### 5.2 Main Results

**Higher fidelity at equal latency.** On SkyReels-V2 (Table 1; FSC-Speed examples in Fig. 7), FSC-Quality matches MotionCache's latency. It raises SSIM from 0.790 to 0.821, improves PSNR by 1.04 dB, and lowers LPIPS by 0.024; all three paired 95% CIs exclude zero. FSC-Speed reaches 2.36×, exceeding all shared-stack baselines while improving SSIM by 0.019 and PSNR by 0.53 dB over MotionCache, both with CIs excluding zero. At both operating points the VBench difference from MotionCache is not statistically resolved. This complements the fidelity result: the methods retain similar reference-free quality but differ in how strongly caching perturbs the same-seed trajectory.

**Baseline operating points.** The MotionCache threshold sweep confirms that its released $\tau$=0.10 is also its latency-matched operating point. On evaluation prompts outside the sweep set, FSC-Quality retains a +0.049 SSIM advantage at matched latency. At the same operating point, FSC-Quality also exceeds matched FlowCache and SeaCache-AR by 0.056 and 0.045 SSIM, respectively (Table 2(a), Fig. 4(a)). The concurrent SCOPE is faster in seconds but 0.015 lower on VBench than FSC-Quality (CI $[-0.024, -0.006]$); stack and fidelity comparability are detailed in App. E.6.

**No-retuning transfer to 14B.** FSC-Quality transfers unchanged to the ∼11× larger SkyReels-V2 14B, reaching 2.12× versus MotionCache's 1.79× with higher SSIM and PSNR, slightly higher

Table 2: **FSC maintains higher fidelity at matched latency and higher SSIM and PSNR on 14B without retuning.** Paired: speedup on all 235 prompts, VBench on 126, fidelity on 21. 14B ($352 \times 640$): VBench is evaluated on 25 of the 47 prompts; all other metrics use all 47. Best in **bold**, second best underlined per block.

| Method | Speedup↑ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|
| *(a) Equal latency, SkyReels-V2 1.3B (Paired)* | | | | | |
| MotionCache | 2.274× | 0.7885 | 17.55 | 0.7789 | 0.3156 |
| FlowCache (threshold 0.123) | 2.305× | **0.7922** | 17.46 | 0.7715 | 0.3125 |
| SeaCache-AR (threshold $\delta$=0.28) | **2.321**× | 0.7908 | 17.68 | 0.7826 | 0.3222 |
| FSC-Quality | 2.275× | 0.7883 | **18.87** | **0.8277** | **0.2840** |
| *(b) SkyReels-V2 14B, 1.3B configuration unchanged (14B)* | | | | | |
| FlowCache | 1.595× | 0.8292 | 16.07 | 0.7692 | **0.3189** |
| MotionCache | 1.792× | 0.8327 | 15.88 | 0.7600 | 0.3289 |
| FSC-Quality | **2.121**× | **0.8406** | **16.18** | **0.7745** | 0.3302 |

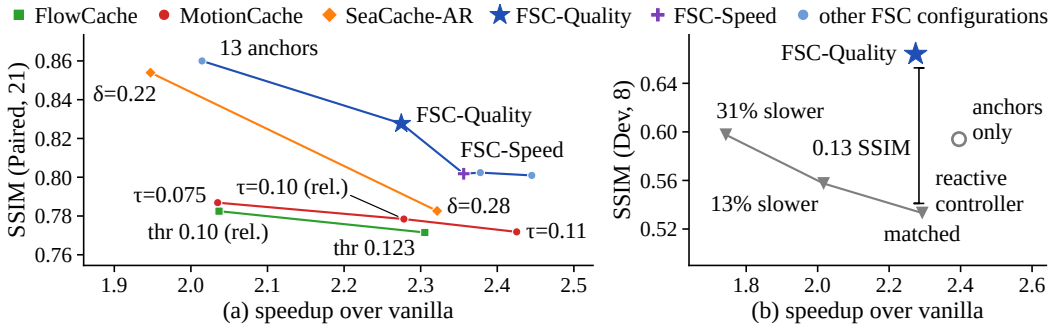



Figure 4: **Operating-point robustness and the effect of separating synchronization from repair. (a)** SkyReels-V2 1.3B; speedup: mean over the 235 Paired prompts; SSIM: their 21 prompts with vanilla references. Outside the MotionCache threshold sweep, FSC retains higher SSIM than the shared-stack baselines at comparable speed, and MotionCache's released $\tau$=0.10 is its latency-matched point. Lines join each method's operating points in speed order ($\tau$, thr, $\delta$: MotionCache, FlowCache, and SeaCache-AR thresholds; rel.: released; FSC: FSC-Quality with 13 anchors and $r$=1.40, FSC-Quality, FSC-Speed, and two other $W$=5 variants of Table 6). **(b)** Dev set (Table 3); speedup: mean over its 8 prompts. The purely reactive variant remains below FSC-Quality at every tested latency; its SSIM gap narrows from 0.13 (matched latency) to 0.07 (31% more latency); hollow: anchors only, no repair.

LPIPS, and comparable VBench quality (Table 2(b); App. E.9). On 1.3B, the fidelity gain replicates across five seeds (App. E.4). The advantage also grows with motion: across vanilla motion terciles, FSC-Quality's SSIM margin over MotionCache increases from +0.018 at low motion to +0.049 at high motion, with the same trend in LPIPS.

**Transfer to MAGI-1.** Both FSC operating points improve speed and same-seed fidelity over MotionCache at comparable VBench quality: FSC-Quality raises SSIM from 0.847 to 0.888, and FSC-Speed reaches 0.878 at higher speed (Table 1). FlowCache is fastest but has substantially lower same-seed fidelity; motion-dependent behavior is analyzed in App. E.8.

### 5.3 Ablation Studies

**Separating Synchronization from Repair.** Removing the anchors and letting $\rho_s$ trigger both repair and full synchronization gives a controlled purely reactive variant of FSC: it uses the same risk signal and repair machinery but has no scheduled anchors. At matched latency, FSC-Quality improves SSIM by 0.131 and lowers LPIPS by 0.186 on all eight development prompts. Giving the reactive variant 13% or 31% more latency narrows but does not eliminate the SSIM gap (Fig. 4(b); Table 3). This controlled result shows that, for FSC's risk signal and repair mechanism, keeping exact synchronization scheduled is more effective than using the same reactive signal to determine both decisions. On the Cross set, FSC also outperforms a latency-matched purely reactive compara-

Table 3: **Separating synchronization from repair: a purely reactive variant reduces SSIM by about $0.13$ at matched latency; increasing its latency by $13\%$ or $31\%$ narrows but does not close the gap.** SkyReels-V2, Dev set; rows modify FSC-Quality (D2S: dense-to-sparse; purely reactive: no anchors, $\rho_s$ triggers repair and full synchronization; $r$: threshold scale); s/video: median wall-clock. Absolute scores are not comparable with Table 1 (different prompts and fidelity harness). The last two columns count the prompts on which FSC-Quality outperforms each variant; further variants in App. E.10. Best in **bold**, second best underlined.

| Variant | SSIM↑ | LPIPS↓ | s/video↓ | Wins (of 8) | |
|---|---|---|---|---|---|
| | | | | SSIM | LPIPS |
| FSC-Quality: D2S anchors + repair | **0.6641** | **0.2925** | 161 | – | – |
| purely reactive, matched latency ($r$=1.076) | 0.5332 | 0.4786 | 160 | 8 | 8 |
| purely reactive, $13\%$ slower ($r$=0.631) | 0.5575 | 0.4285 | 182 | 8 | 8 |
| purely reactive, $31\%$ slower ($r$=0.250) | 0.5976 | 0.3569 | 211 | 8 | 6 |
| anchors only, no repair (13 D2S anchors) | 0.5940 | 0.3337 | **153** | 8 | 8 |
| uniform anchor spacing (with repair) | <u>0.6441</u> | <u>0.3189</u> | 157 | 5 | 5 |
| sparse-to-dense, reversed (with repair) | 0.6033 | 0.3659 | 161 | 8 | 8 |
| no age term, no overdue safeguard | 0.6429 | 0.3369 | <u>155</u> | 8 | 8 |

tor on FramePack / Matrix-Game: SSIM $+0.079$ / $+0.058$, PSNR $+5.18$ / $+2.77$ dB, and LPIPS $-0.092$ / $-0.057$; all 95% CIs exclude zero. These backbones differ from SkyReels-V2 in cache interface and token-execution cost, so the result supports the separation principle independently of SkyReels-specific sparse execution.

**Schedule, Age, and Sensitivity.** A $2 \times 2$ factorial on the Dev set separates the two components: dense-to-sparse placement adds $0.017$ SSIM without repair, adaptive repair adds $0.068$ under uniform anchors, while combining both yields the largest gain of $0.088$. Two results support the growing gaps of Eq. 1: the influence of a single cached step on the final video decays $6.1\times$ from step $8$ to step $39$, and every alternative schedule evaluated end to end, including uniform, reversed, and three surrogate-optimized schedules, yields lower mean SSIM (Tables 3 and 17). Cache age provides an important overdue safeguard: removing the age term and its overdue rule reduces SSIM by $0.021$ on all eight prompts while lowering latency by $4\%$. Around FSC-Quality, six of eight local perturbations change SSIM by $\leq 0.003$ and latency by $\leq 1.8\%$.

## 6 Conclusion

Our results support one principle for cache control in AR video diffusion: for reactive staleness signals, exact synchronization should be scheduled before measured drift accumulates, and measured drift should only decide how much repair to spend between synchronizations. FSC instantiates this principle through scheduled exact anchors and whole-step replay, with a frequency- and age-aware latent risk controlling inter-anchor repair. On SkyReels-V2 (1.3B and, unchanged, 14B), FSC is closer to the vanilla trajectory than MotionCache in SSIM and PSNR at equal or higher speed, and coupling both decisions into one risk signal reduces fidelity at matched latency (Sec. 5.3). On MAGI-1, FSC's native-interface port improves speed and fidelity over MotionCache at comparable VBench, and the separation advantage holds on FramePack and Matrix-Game with per-backbone calibration.

## AI use statement

We used generative AI tools, primarily Claude by Anthropic, as coding and writing assistants. Their uses included implementing author-specified methods and integrations, writing baseline ports and experiment scripts, suggesting ablations and experimental parameters, assisting with experiment planning, drafting and editing text, translating author notes into English, generating code, auditing reported numbers against experiment logs, and finding and summarizing related work.

Generative AI also assisted in brainstorming hypotheses, interpreting experimental outputs, and refining the conceptual presentation of the work. All research questions, methodological choices,

experimental protocols, acceptance or rejection of suggested analyses, and final scientific claims were determined by the authors. We did not use generative AI to generate synthetic datasets.

All AI-assisted outputs were reviewed by the authors. AI-assisted code was validated through consistency tests and reproduction of the reported experiments. Reported numerical results were independently re-derived from the underlying experiment databases and checked by the authors, and descriptions of related work were checked against the cited papers. The authors take full responsibility for the final text, methods, code, analyses, and conclusions.

## REPRODUCIBILITY STATEMENT

Sec. 4 defines the FSC controller, including the dense-to-sparse anchor schedule, the frequency- and age-aware staleness risk, and the repair routing. Algorithm 1 (App. B.7) gives the per-step procedure, while App. B provides the exact schedule construction, routing rules, cache-state updates, and backbone-specific instantiations (App. B.5).

App. D reports the hardware and software environment, generation settings and seeds, timing, FLOPs, and memory measurement procedures, and the final hyperparameters for each backbone. App. E describes baseline configuration and reproduction and reports the evaluated final operating points (Tables 6 and 7). The evaluation protocol is specified in Sec. 5.1.

## REFERENCES

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

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*, volume 40, pp. 19862–19870, 2026. doi: 10.1609/aaai.v40i24.39069.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/2aee1c4159e48407d68fe16ae8e6e49e-Abstract-Conference.html.

Guibin Chen, Dixuan Lin, Jiangping Yang, Chunze Lin, Junchen Zhu, Mingyuan Fan, Hao Zhang, Sheng Chen, Zheng Chen, Chengcheng Ma, Weiming Xiong, Wei Wang, Nuo Pang, Kang Kang, Zhiheng Xu, Yuzhe Jin, Yupeng Liang, Yubing Song, Peng Zhao, Boyuan Xu, Di Qiu, Debang Li, Zhengcong Fei, Yang Li, and Yahui Zhou. SkyReels-V2: Infinite-length film generative model, 2025. URL https://arxiv.org/abs/2504.13074.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. $\Delta$-DiT: A training-free acceleration method tailored for diffusion transformers, 2024b. URL https://arxiv.org/abs/2406.01125.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. SeaCache: Spectral-evolution-aware cache for accelerating diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14283–14294, 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. URL https://openaccess.thecvf.com/content/CVPR2026/html/Cui_Denoising_as_Path_Planning_Training-Free_Acceleration_of_Diffusion_Models_with_CVPR_2026_paper.html.

Hanshuai Cui, Zhiqing Tang, Zhi Yao, Fanshuai Meng, Weijia Jia, and Wei Zhao. Not all frames deserve full computation: Accelerating autoregressive video generation via selective computation and predictive extrapolation, 2026b. URL https://arxiv.org/abs/2604.02979. To appear at ACM Multimedia 2026.

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.

Ziqi Huang, Fan Zhang, Xiaojie Xu, Yinan He, Jiashuo Yu, Ziyue Dong, Qianli Ma, Nattapol Chanpaisit, Chenyang Si, Yuming Jiang, Yaohui Wang, Xinyuan Chen, Ying-Cong Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench++: Comprehensive and versatile benchmark suite for video generative models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025. doi: 10.1109/TPAMI.2025.3633890.

Guandong Li. Frequency-aware error-bounded caching for accelerating diffusion transformers, 2026. URL https://arxiv.org/abs/2603.05315.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7353–7363, 2025a. URL https://openaccess.thecvf.com/content/CVPR2025/html/Liu_Timestep_Embedding_Tells_Its_Time_to_Cache_for_Video_Diffusion_CVPR_2025_paper.html.

Jiacheng Liu, Peiliang Cai, Qinming Zhou, Yuqi Lin, Deyang Kong, Benhao Huang, Yupei Pan, Haowen Xu, Chang Zou, Junshu Tang, Shikang Zheng, and Linfeng Zhang. FreqCa: Accelerating diffusion models via frequency-aware caching, 2025b. URL https://arxiv.org/abs/2510.08669.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with TaylorSeers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15853–15863, 2025c. URL https://openaccess.thecvf.com/content/ICCV2025/html/Liu_From_Reusing_to_Forecasting_Accelerating_Diffusion_Models_with_TaylorSeers_ICCV_2025_paper.html.

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

Yuexiao Ma, Xuzhe Zheng, Jing Xu, Xiwei Xu, Feng Ling, Xiawu Zheng, Huafeng Kuang, Huixia Li, Xing Wang, Xuefeng Xiao, Fei Chao, and Rongrong Ji. Flow caching for autoregressive video generation. In *International Conference on Learning Representations*, 2026.

Kepan Nan, Wangbo Zhao, Penghao Zhou, Jun Li, Zhenheng Yang, Jian Yang, and Ying Tai. Accelerating autoregressive video diffusion via history-guided cache and residual correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43740–43750, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Nan_Accelerating_Autoregressive_Video_Diffusion_via_History-Guided_Cache_and_Residual_Correction_CVPR_2026_paper.html.

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.

Sand.ai, Hansi Teng, Hongyu Jia, Lei Sun, Lingzhi Li, Maolin Li, Mingqiu Tang, Shuai Han, Tianning Zhang, W. Q. Zhang, Weifeng Luo, Xiaoyang Kang, Yuchen Sun, Yue Cao, Yunpeng Huang, Yutong Lin, Yuxin Fang, Zewei Tao, Zheng Zhang, Zhongshu Wang, Zixun Liu, Dai Shi, Guoli Su, Hanwen Sun, Hong Pan, Jie Wang, Jiexin Sheng, Min Cui, Min Hu, Ming Yan, Shucheng Yin, Siran Zhang, Tingting Liu, Xianping Yin, Xiaoyu Yang, Xin Song, Xuan Hu, Yankai Zhang, and Yuqiao Li. MAGI-1: Autoregressive video generation at scale, 2025. URL https://arxiv.org/abs/2505.13211.

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. FORA: Fast-forward caching in diffusion transformer acceleration, 2024. URL https://arxiv.org/abs/2407.01425.

Jiangshan Wang, Kang Zhao, Jiayi Guo, Jiayu Wang, Hang Guo, Chenyang Zhu, Xiu Li, and Xiangyu Yue. PreciseCache: Precise feature caching for efficient and high-fidelity video generation. In *International Conference on Learning Representations*, 2026a.

Zile Wang, Zexiang Liu, Jiaxing Li, Kaichen Huang, Baixin Xu, Fei Kang, Mengyin An, Peiyu Wang, Biao Jiang, Yichen Wei, Yidan Xietian, Jiangbo Pei, Liang Hu, Boyi Jiang, Hua Xue, Zidong Wang, Haofeng Sun, Wei Li, Wanli Ouyang, Xianglong He, Yang Liu, Yangguang Li, and Yahui Zhou. Matrix-Game 3.0: Real-time and streaming interactive world model with long-horizon memory, 2026b. URL https://arxiv.org/abs/2604.08995.

Jing Xu, Yuexiao Ma, Xuzhe Zheng, Xing Wang, Shiwei Liu, Chenqian Yan, Xiawu Zheng, Rongrong Ji, Fei Chao, and Songwei Liu. Motion-aware caching for efficient autoregressive video generation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22963–22974, 2025.

Lvmin Zhang, Shengqu Cai, Muyang Li, Gordon Wetzstein, and Maneesh Agrawala. Frame context packing and drift prevention in next-frame-prediction video diffusion models. In *Advances in Neural Information Processing Systems*, 2025.

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.

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. URL https://arxiv.org/abs/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.

# A Extended Characterization

This appendix details the profiling protocol and the analyses summarized in Sec. 3.

## A.1 Profiling Protocol

Unless noted, reuse error is the relative $L_2$ deviation $\|\hat{\mathbf{y}} - \mathbf{y}\|_2 / \|\mathbf{y}\|_2$ between the output produced with the tested reuse policy ($\hat{\mathbf{y}}$) and the corresponding full-compute output ($\mathbf{y}$) at the same denoising step, prompt, and seed. SSIM is computed against the same-seed vanilla video. Each study uses eight profiling prompts (six for the step-dependence curve of Fig. 1(a) and the age curves of Fig. 5) that are disjoint from the benchmark prompts. All characterization runs use a reduced setting rather than the protocol of App. D: $320 \times 544$ resolution, a

single 81-frame segment generated synchronously (`ar_step = 0`: all frames share one noise level; no autoregressive rollout), 50 steps, seed 0, and the Diffusers releases of the checkpoints (`SkyReels-V2-DF-{14B,1.3B}-540P-Diffusers`).

### A.2 Observation 1: Step Dependence and Schedule Comparison

Along a full-compute trajectory, at every step $s$ and every transformer block we reuse that block's residual from step $s-1$ given the block's exact input and measure the relative error against the exact output, so errors never compound; the curve averages these block-local errors over the 40 blocks and 6 prompts. "Early" and "late" in the $2.4\times$ ratio denote the mean error over the steps 1–10 and 40–49 of the 50-step trajectory, respectively. The placement comparison (Fig. 1(b)) compares three placements of exact refreshes: dense-to-sparse (7 refreshes, $2.62\times$; gaps increasing along the trajectory), uniform (8 refreshes, $2.49\times$; equal gaps), and sparse-to-dense (7 refreshes, $2.62\times$; the mirror image). Dense-to-sparse attains the highest mean SSIM, ahead of uniform on 6 of the 8 paired prompts (one tie) and of sparse-to-dense on 7; the full-controller schedule ablation on the development set is reported in Sec. 5.3.

### A.3 Observation 2: Sparse Execution Details

Two prototypes (Fig. 1(c)) are profiled at token fractions $q \in \{0.10, 0.15, 0.20, 0.30\}$ (Table 4): a dense-forward token-refresh reference and true sparse kernels. *Token refresh* is a dense-forward quality reference: at every step between anchors it runs the full transformer forward for all tokens and keeps the fresh output only for the selected tokens (the others keep their cached output). Its speedup is therefore flat at $1.23\times$ by construction (only the last ten of the 50 steps are free whole-step copies) and does not measure the cost of sparse execution; the speed argument rests on the true sparse kernels. *True sparse kernels* execute attention and FFN only for the selected tokens; the kernel is exact for those tokens, but non-selected tokens contribute keys and values re-projected from their cached, stale hidden states. Its speedup rises to $1.94\times$ at $q = 0.10$, still well below its own compute-proportional projection ($2.72\times$), mainly because costs that do not shrink with $q$ remain, chiefly re-projecting the keys and values of the full context (the largest single item) and per-block launch overhead; the stale keys/values additionally incur an $\approx 0.024$ SSIM reduction relative to the dense-forward token-refresh reference run on the same six-anchor schedule (a separate run from the three-anchor speed measurement of Table 4). Whole-step output reuse with dense-to-sparse refreshes, which skips the transformer entirely at reused steps, reaches $2.62\times$ under the same conditions. The deployed controller (Sec. 4.3) matches neither prototype exactly: repaired tokens run as a gathered batch through every layer and attend to the cached keys and values of all tokens without re-projecting them; its cost enters Sec. 5 only through measured end-to-end latency.

Table 4: **True sparse token repair stays far below its compute-proportional ideal, and whole-step dense-to-sparse (D2S) reuse is faster.** End-to-end speedup of the true sparse kernels and of the dense-forward token-refresh reference (flat by construction) versus their compute-proportional ideals (for token refresh, the speedup its schedule would reach if only the selected tokens were computed; SkyReels-V2 1.3B, profiling setting of App. A.1); the ideals differ because token refresh uses 3 exact anchors and true sparse kernels use 6.

| Token budget $q$ | 0.10 | 0.15 | 0.20 | 0.30 |
|---|---|---|---|---|
| Compute-proportional ideal (token refresh) | 3.18× | 2.93× | 2.72× | 2.37× |
| Compute-proportional ideal (true sparse) | 2.72× | 2.55× | 2.40× | 2.16× |
| Token refresh (dense-forward reference) | 1.23× | 1.23× | 1.23× | 1.23× |
| True sparse kernels | 1.94× | 1.83× | 1.78× | 1.65× |
| Whole-step D2S reuse (reference) | 2.62× | | | |

### A.4 Observation 3: Approximate-Anchor Variants

An anchor step is the reference that all following non-anchor steps replay from; any approximation inside it is inherited by every step until the next anchor. We tested 17 approximate-anchor policies grouped into three families: (i) *pyramid kernels*, which inside refresh steps reuse cached block residuals band by band when per-block gates on input drift, residual stability, and age allow it, and

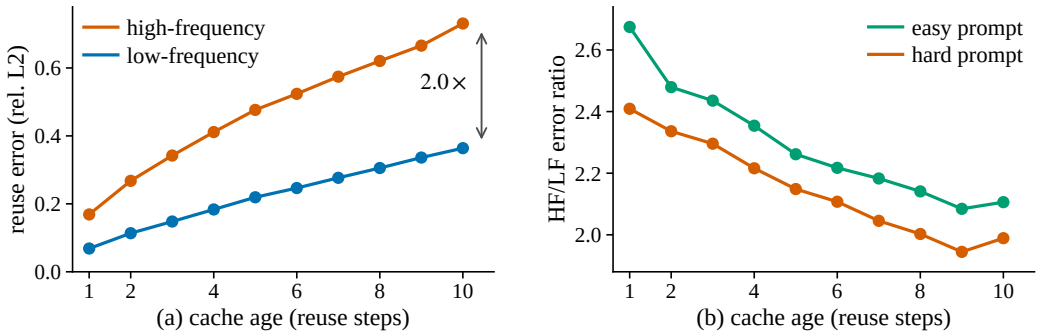



Figure 5: **High-frequency residual content goes stale faster than low-frequency content, but the separation is content-dependent (Observation 4; SkyReels-V2 14B, six profiling prompts). (a)** Relative-$L_2$ reuse error versus cache age (number of reuse steps since the cached state was refreshed) for the low-frequency band of the block residuals (two levels of $2 \times 2$ average pooling) and the high-frequency remainder, each measured against the corresponding full-compute band; HF error grows faster in absolute terms and remains $\approx$ 2–2.5$\times$ the LF error across ages 1–10 (up to 2.7$\times$ for individual prompts; (b)). **(b)** The HF/LF error ratio from (a) for the lowest-error and highest-error profiling prompts ("easy prompt" and "hard prompt" in the legend); it stays above one throughout but is smaller on the highest-error prompt and shrinks with age.

reconstruct skipped blocks from the cached bands (*pyramid-full* is the complete kernel); (ii) *block-skip gates at anchors*, which skip transformer blocks whose input change falls below a threshold even during refresh steps; and (iii) adaptive anchor skipping, which skips a nominal anchor when the measured input risk is low and keeps the whole cache instead. Only three of these points lie above the clean-anchor Pareto frontier of Fig. 1(d): block-skip gates at 3.01–3.04$\times$ with identical SSIM (0.824), up to 0.008 above its linear interpolation and not dominated by any clean schedule. The pyramid-full variant has approximately 0.046 lower SSIM than the clean full-anchor baseline at the same nominal budget. Block-skip gates at anchors provide additional speedup (2.00–3.54$\times$ vs. 1.94$\times$) at lower SSIM than the clean baseline.

### A.5 Observation 4: Frequency-Resolved Staleness

**Measurement.** On a full-compute trace, the exact residual $\mathbf{u}$ of every transformer block is split on the token grid into a low-frequency band (two levels of $2 \times 2$ average pooling, upsampled) and the high-frequency remainder. This probes where reuse error lives in the residuals; the deployed controller instead measures latent drift with a single $2 \times 2$ pool (Eq. 2). For each cache age $k \in \{1, \ldots, 10\}$ we replay the band of the residual cached $k$ steps earlier and measure its relative-$L_2$ deviation from the exact band (Fig. 5(a)). The profiling set contains six prompts; the lowest-error and highest-error prompts, i.e., those with the lowest and highest mean reuse error, are a static museum statue and a texture-rich koi pond, respectively. The HF/LF ratio (Fig. 5(b)) decreases from 2.67 to 2.11 on the lowest-error prompt and from 2.41 to 1.99 on the highest-error prompt over ages 1$\rightarrow$10. Pooling the lower- and higher-error halves of the six prompts preserves this ordering at every age (2.51$\rightarrow$2.04 vs. 2.43$\rightarrow$1.98), and HF stays above LF on all six prompts individually.

**Rationale for Soft Frequency-Aware Control.** Because the HF/LF separation is content-dependent, a design that assigns tokens to two independently fixed anchor schedules according to their frequency content would be mis-calibrated precisely on high-error prompts, where the gap between bands is smallest. FSC therefore uses frequency content as a soft input to a per-step risk score (Sec. 4.2): high-frequency drift detects local detail and texture change, low-frequency drift detects coarse structural shift, and cache age guards against tokens with small per-step but large accumulated drift.

# B FSC Design Details

This appendix expands Sec. 4 with the full construction of the anchor schedule, the complete definition of the staleness estimator, the exact routing rules and cache-state updates, the backbone-specific instantiations, and the per-step algorithm; Appendix C validates the estimator against an oracle.

**Notation.** Throughout, transformer block $\ell$ at denoising step $s$ maps the hidden states $\mathbf{H}_s^\ell$ of all $N$ tokens as $\mathbf{H}_s^{\ell+1} = \mathbf{H}_s^\ell + \mathbf{U}_s^\ell$ with $\mathbf{U}_s^\ell = U_\theta^\ell(\mathbf{H}_s^\ell, s, \mathbf{c})$, whose computation dominates the cost; we write $\mathbf{h}_{s,i}^\ell = [\mathbf{H}_s^\ell]_i$ and $\mathbf{u}_{s,i}^\ell = [\mathbf{U}_s^\ell]_i$ for the hidden state and the residual of token $i$. Residual caching replays a stored $\mathbf{u}_{\mathrm{cache},i}^\ell$ instead, keeping the current hidden state. We write residual replay blockwise for notation; the implementation stores the equivalent whole-stack residual per token, while keys and values are cached per block.

## B.1 Anchor Schedule Construction

For $n$ post-warmup anchors there are $n-1$ inter-anchor gaps, to which FSC assigns linearly increasing weights $\omega_i = 1 + i$, $i = 0, \dots, n-2$. The normalized cumulative weights

$$c_0 = 0, \qquad c_j = \frac{\sum_{i=0}^{j-1} \omega_i}{\sum_{i=0}^{n-2} \omega_i}, \qquad j = 1, \dots, n-1, \tag{7}$$

map the anchors onto $[W, E]$ via Eq. 1. A monotonic integer correction then removes duplicate locations, enforces strict increase, and fixes the endpoints $a_0 = W$ and $a_{n-1} = E$. Degenerate cases ($n \le 1$) reduce to at most one anchor; all released configurations use $n \ge 4$. A step is a clean refresh whenever $\mathrm{anchor}(s) = (s < W) \vee (s \in \mathcal{A})$. Increasing gaps allocate more exact computation to the early, less-cacheable part of the trajectory and exploit greater redundancy later.

## B.2 Staleness Estimator: Definitions

**Token alignment.** Both bands of Eq. 2 are partitioned using the same local spatial layout as the backbone's patch embedding, so each proxy token corresponds to exactly one transformer token; $\mathbf{X}_s \in \mathbb{R}^{C \times F \times H_{\mathrm{lat}} \times W_{\mathrm{lat}}}$ denotes the raw latent, and $\varepsilon$ in Eq. 3 is a small constant for numerical stability.

**Accumulated drift and age.** The accumulated high-frequency drift is updated as $A_{s,i}^{\mathrm{HF}} \leftarrow A_{s-1,i}^{\mathrm{HF}} + d_{s,i}^{\mathrm{HF}}$ and reset to zero whenever token $i$ is recomputed; the age $a_{s,i}$ is incremented at every reuse step and reset to zero on recompute. Because every $d_{s,i}^{\mathrm{HF}}$ is measured against the anchor reference (Eq. 3), $A^{\mathrm{HF}}$ is a time integral of anchor-relative displacement, not the path length $\sum_s \|\mathbf{X}_s - \mathbf{X}_{s-1}\|$: a token displaced from the reference keeps accumulating even if it stops moving. Summing anchor-relative drift over the reuse duration, rather than step-to-step increments, thus penalizes sustained staleness.

**Group-level aggregation.** The instantaneous drifts are summarized by their 95th percentiles, $g_s^{\mathrm{B}} = Q_{0.95}(\{d_{s,i}^{\mathrm{B}}\}_{i=1}^N)$, which emphasizes localized stale regions while being less sensitive than the maximum to isolated noise; the group age is $g_s^{\mathrm{age}} = \frac{1}{N}\sum_i a_{s,i}/k_{\mathrm{tok}}$. The weights $w_{\mathrm{HF}}$, $w_{\mathrm{LF}}$, $w_{\mathrm{age}}$ in Eq. 4 set the relative contribution of instantaneous detail drift, coarse structural drift, and accumulated reuse age.

**Properties of the proxy.** First, $d_{s,i}^{\mathrm{B}}$ measures drift of the evolving latent relative to the anchor latent; it is a proxy, not a direct measurement of residual error (validated in Appendix C). Second, for tokens repaired between anchors, drift is still measured against the anchor reference, so their risk is systematically over-estimated. This makes the routing decision conservative: over-estimation can only move the controller toward more computation at the step level. Third, frequency is used as a soft cache-validity signal rather than a permanent partition of tokens into independently scheduled bands, because the HF/LF separation narrows modestly on higher-error prompts (Fig. 5(b)).

### B.3 Routing Rules and Repair Levels

Let $\mathrm{viol}_s = \exists\, i :\ a_{s,i} \geq k_{\mathrm{tok}}$ indicate that at least one token has reached its reuse-age limit, and let skips count consecutive whole-step reuses. With the thresholds of Eq. 5, a non-anchor step is routed by evaluating the following cases top to bottom and applying the first match:

$$\mathrm{mode}(s) = \begin{cases} \mathrm{Full}, & \rho_s > t_{\mathrm{full}} \ \vee\ \big(\mathrm{viol}_s \wedge \rho_s > \eta\, t_{\mathrm{full}}\big), \\ C0, & \rho_s < t_{\mathrm{skip}} \ \wedge\ \mathrm{skips} < k_{\mathrm{step}} \ \wedge\ \neg \mathrm{viol}_s, \\ C1, & \rho_s < t_1, \\ C2, & \rho_s < t_2, \\ C3, & \text{otherwise.} \end{cases} \tag{8}$$

The relaxation factor $\eta < 1$ lowers the Full bar whenever any token is overdue (we use $\eta = 0.8$ throughout), and $k_{\mathrm{step}}$ prevents indefinitely repeated $C0$ decisions. The phase multipliers $\varphi_m(s)$ use tighter reuse thresholds early in the trajectory and progressively looser thresholds later; exact base thresholds $b_m$ and multipliers are listed in Appendix D.

The level-specific repair fractions are $q_m = q_{\mathrm{scale}}\, \hat{q}_m$ with $\hat{q}_1 < \hat{q}_2 < \hat{q}_3$. At level $C3$ the optional low-frequency structural safeguard augments Eq. 6 as $\mathcal{S}_s \leftarrow \mathcal{S}_s \cup \mathrm{Top}_{p_{\mathrm{LF}}}(\{d^{\mathrm{LF}}_{s,i}\}_{i=1}^{N})$, so that tokens with large coarse structural drift are repaired even when accumulated high-frequency drift alone would not select them. The resulting block output is

$$\widehat{\mathbf{h}}^{\ell+1}_{s,i} = \begin{cases} \mathbf{h}^{\ell}_{s,i} + \mathbf{u}^{\ell}_{s,i}, & i \in \mathcal{S}_s, \\ \mathbf{h}^{\ell}_{s,i} + \mathbf{u}^{\ell}_{\mathrm{cache},i}, & i \notin \mathcal{S}_s, \end{cases} \tag{9}$$

i.e., repaired tokens execute the full block while reused tokens bypass both attention and non-attention residual computation.

### B.4 Cache-State Update

For each compatible controller context, FSC maintains the clean LF/HF references, per-token ages $a_i$, accumulated drift $A^{\mathrm{HF}}_i$, cached transformer residuals, and the consecutive-skip counter. The state evolves with the selected mode:

- **Scheduled clean anchor.** All tokens are recomputed exactly and their residual caches rewritten; the LF/HF reference is replaced by the current exact latent; $a_i \leftarrow 0$, $A^{\mathrm{HF}}_i \leftarrow 0$ for all $i$; $\mathrm{skips} \leftarrow 0$.
- $C0$**.** No residual is recomputed; all ages increase by one; $\mathrm{skips} \leftarrow \mathrm{skips} + 1$.
- $C1$**–**$C3$**.** Tokens in $\mathcal{S}_s$ are gathered and recompute their queries, keys, values, and feed-forward outputs in every block; their new keys/values overwrite their slots in the block's key/value cache and their queries attend to the full cached key/value set of the context window (entries of non-repaired tokens date from their last recompute). Their residuals are overwritten with $a_i \leftarrow 0$, $A^{\mathrm{HF}}_i \leftarrow 0$; reused tokens age by one; the anchor latent reference is unchanged; $\mathrm{skips} \leftarrow 0$.
- **Risk-triggered full repair** (Full)**.** All residuals are recomputed and all age/accumulation variables reset, but the anchor latent reference is unchanged; $\mathrm{skips} \leftarrow 0$.

In the implementation the anchor latent reference is rewritten by exactly one routine, which is invoked only on scheduled anchor steps; the risk-triggered full repair shares the full-recompute path (rewriting the residual cache state and resetting ages and accumulated drift) but never invokes the reference-rewriting routine, so the reference is pinned to the last scheduled anchor by construction. Keeping one drift coordinate per anchor interval means the group risk does not reset after a full repair, so it can re-trigger later in the same anchor interval. On the 946-prompt benchmark FSC-Quality routes 70.3% / 12.0% / 2.9% / 8.9% / 5.8% of non-anchor steps to $C0$ / $C1$ / $C2$ / $C3$ / Full, and FSC-Speed 69.1% / 13.2% / 2.8% / 9.2% / 5.7%.

### B.5 Backbone Instantiations

**SkyReels-V2.** SkyReels-V2 operates over spatial tokens and temporal groups within a diffusion-forcing sliding window, and classifier-free guidance introduces separate conditional and unconditional forward phases. FSC keeps independent controller state for each compatible (phase, group) context, so cache age, accumulated drift, frequency references, and cached residuals are never mixed across incompatible contexts, and group-local denoising steps determine both anchor scheduling and phase-aware routing.

**MAGI-1.** MAGI-1 exposes chunk-level and token-level relative-$L_1$ reuse thresholds together with a token-budget cap. FSC transfers to MAGI-1 through this native cache interface by scaling the token-level threshold,

$$\tau_{\text{token}} = \frac{\beta\, r}{w_{\text{HF}}} \qquad (w_{\text{HF}} = 1), \tag{10}$$

where $\beta$ is the substrate's base reuse threshold and $r$ the threshold scale. The chunk-level threshold is read only in MAGI's chunk-wise-only phase, which every FSC run disables, and the token-budget cap stays at its native value, so the LF weight $w_{\text{LF}}$ and the repair-fraction scale $q_{\text{scale}}$ have no effect on MAGI-1. The dense-to-sparse schedule runs on MAGI's global denoising-step counter; at scheduled anchors the reuse thresholds are set to zero, forcing exact computation, and the age-control principle is realized through a forced full recomputation every $k_{\text{tok}}$ steps of the same counter. Among the two primary Table 1 backbones, the full LF/HF-and-age controller runs on SkyReels-V2, while MAGI-1 tests transfer through its native cache interface. The full controller is additionally evaluated end-to-end on FramePack and Matrix-Game in App. E.2.

### B.6 Complexity

Per denoising step, the SkyReels controller performs one $2 \times 2$ spatial average pool, an upsampling and subtraction to obtain the two bands, and per-token norm and ranking operations. The latent band operations are linear in the latent size, while percentile computation and token ranking add an $\mathcal{O}(N \log N)$ per-group term in the token count $N$. No auxiliary network or additional transformer pass is required. A GPU micro-benchmark with the deployed shapes ($10{,}200$ tokens per group; $612$ routing decisions per video, both guidance phases, weighted by the measured routing shares, plus one band extraction per forward) gives about $0.23$ s per video, under $0.2\%$ of the $161$ s generation latency; band extraction alone is below $0.05\%$. The measured acceleration comes from skipping transformer residual computation through whole-step replay and selective token reuse.

### B.7 Per-Step Algorithm

Algorithm 1 summarizes one FSC denoising step on SkyReels-V2, combining the schedule of Sec. 4.1, the estimator of Sec. 4.2, the routing of Eq. 8, and the state updates of Appendix B.4.

## C Oracle Diagnostic of the FSC Staleness Risk

We measure how strongly the deployed FSC staleness risk $\rho_s$ (Eq. 4) is associated with the error incurred by residual reuse. This controlled profiling experiment uses the SkyReels-V2 1.3B backbone with our development set (eight prompts; App. D) and a development configuration of the same family: the risk weights, $k_{\text{tok}}$, $k_{\text{step}}$, thresholds, repair fractions, and phase shaping of App. D, but $W$=7, $n$=9, $r$=1.75, $q_{\text{scale}}$=1.00, and $p_{\text{LF}}$=0.1. This yields $4{,}896$ eligible non-anchor controller decisions. At each decision, we evaluate the exact residuals for oracle measurement. The probe leaves the anchor reference, token ages, accumulated drift, and routing state unchanged, but overwrites the probed group's K/V cache. Simpler predictors computed from the same decisions are informative too: the number of steps since the last scheduled anchor alone reaches Spearman $0.865$ (ROC-AUC $0.913$), the age term alone $0.715$, and the HF or LF drift term alone $0.859$ / $0.867$; $\rho_s$ adds $0.028$ Spearman over the elapsed-step predictor, and its partial Spearman correlation with $E_s$ controlling for elapsed steps is $0.472$, against $0.174$ for elapsed steps controlling for $\rho_s$. End-to-end quality and acceleration are evaluated separately at benchmark scale in Sec. 5.

**Algorithm 1** Frequency-Step Cache (FSC), one denoising step $s$

**Require:** latent $\mathbf{X}_s$; warmup $W$; anchor set $\mathcal{A}$; clean LF/HF references; cached residuals; token ages $a$; accumulated drift $A^{\mathrm{HF}}$; skip counter skips

**if** $s < W$ **or** $s \in \mathcal{A}$ **then** ▷ clean anchor
  recompute all tokens; store fresh residuals; rewrite LF/HF references
  $a \leftarrow 0$; $A^{\mathrm{HF}} \leftarrow 0$; skips $\leftarrow 0$; **return**
**end if**
compute LF/HF bands (Eq. 2) and drifts $d^{\mathrm{LF}}, d^{\mathrm{HF}}$ w.r.t. the anchor reference (Eq. 3)
$A^{\mathrm{HF}} \leftarrow A^{\mathrm{HF}} + d^{\mathrm{HF}}$; compute $\rho_s$ (Eq. 4) and thresholds $t_m(s)$ (Eq. 5)
route mode $\in \{C0, C1, C2, C3, \mathrm{Full}\}$ (Eq. 8)
**if** mode $= \mathrm{Full}$ **then**
  recompute all tokens; refresh residuals; $a \leftarrow 0$; $A^{\mathrm{HF}} \leftarrow 0$; skips $\leftarrow 0$
**else if** mode $= C0$ **then**
  replay cached residuals for all tokens; $a \leftarrow a + 1$; skips $\leftarrow$ skips $+ 1$
**else**
  build $\mathcal{S}_s$ (Eq. 6); add top-$p_{\mathrm{LF}}$ LF drifters if mode $= C3$ and LF gate enabled
  recompute tokens in $\mathcal{S}_s$ and refresh their residuals; replay the rest
  reset $a, A^{\mathrm{HF}}$ for repaired tokens; $a \leftarrow a + 1$ for reused tokens; skips $\leftarrow 0$
**end if**

For each eligible non-anchor decision $s$, let $\mathbf{u}_{s,i}^{\mathrm{exact}}$ denote the exact current residual of token $i$, and let $\mathbf{u}_{s,i}^{\mathrm{cache}}$ denote the residual currently stored in FSC's cache. We define the token-level oracle reuse error as

$$e_{s,i} = \frac{\left\|\mathbf{u}_{s,i}^{\mathrm{exact}} - \mathbf{u}_{s,i}^{\mathrm{cache}}\right\|_2}{\left\|\mathbf{u}_{s,i}^{\mathrm{exact}}\right\|_2 + \varepsilon}. \tag{11}$$

Because $\rho_s$ is a group-level routing signal, we aggregate token errors using the same high-percentile principle as the FSC controller:

$$E_s = Q_{0.95}\left(\{e_{s,i}\}_{i=1}^{N}\right). \tag{12}$$

The pooled Spearman correlation of Fig. 3 is lower than every per-prompt value because prompts differ in their baseline error: at equal $\rho_s$, $E_s$ differs by up to $2\times$ across prompts, and removing each prompt's median $\log E_s$ raises the pooled correlation to $0.933$. The unsafe-reuse criterion $E_s > 0.05$ (Sec. 4.2) is fixed independently of the ROC analysis and is not selected to maximize AUC; under it, $3{,}216$ of the $4{,}896$ decisions ($65.7\%$) are labeled unsafe (ROC curve in Fig. 6). The monotonic rise of the mean and median oracle error across deciles of $\rho_s$ (Fig. 3(b)) is a calibration-style check that the association is not driven by a few extreme decisions.

## D Implementation Details

**Hardware and Software.** All main-table experiments run on a single server with NVIDIA RTX PRO 6000 Blackwell GPUs (97,887 MiB of reported device memory, corresponding to 102.6 GB; GB denotes $10^9$ bytes throughout; driver 570.211.01), PyTorch 2.11.0 with CUDA 12.8, and bfloat16 inference. Each video is generated on one GPU with batch size 1; timed runs are launched only on GPUs with no other process, and runs later slowed by co-located jobs are flagged post hoc (Timing Procedure below). VBench-Long (Huang et al., 2025) evaluation uses the official repository at commit `45e79ec` in `long_vbench_standard` mode (each video scored on the VBench-Long dimensions of its prompt set with the canonical short prompt); the four GRiT-based semantic dimensions run in a separate detectron2 environment. The reported VBench aggregate uses a fixed subset of eleven dimensions: subject consistency, motion smoothness and dynamic degree (subject-consistency prompts), overall consistency, aesthetic quality and imaging quality (overall-consistency prompts), scene and background consistency (scene prompts), and color, object class and appearance style (their own prompt sets); every dimension is scored on its complete VBench-Long prompt set (505 prompts in total) with the official normalization and dimension weights, and, as in the official implementation, a video in which the detector finds no object of the prompt is excluded from that method's color mean. On MAGI-1, speedups are per-prompt ratios over the vanilla-clean prompt pool after per-method latency auditing; VBench aggregates use the evaluated

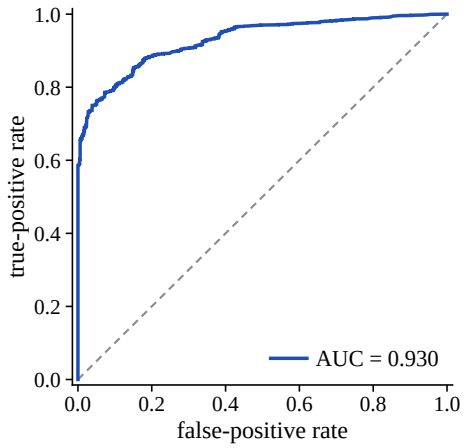



Figure 6: **The FSC risk $\rho_s$ detects unsafe reuse ($E_s > 0.05$) with ROC-AUC $0.930$.** The $4{,}896$ non-anchor decisions of Fig. 3 (our development set, SkyReels-V2 1.3B, development configuration described above). The oracle probe leaves the anchor reference and routing-state variables unchanged; the K/V-cache side effect is described above. Dashed: chance.

dimension subset on $500$ prompts (five color prompts in which the detector finds the object for no method contribute nothing), and fidelity uses the first ten prompts of each prompt set without removing Dev ($110$ prompts). LPIPS uses the AlexNet variant (Zhang et al., 2018); SSIM and PSNR are computed per frame against the same-seed vanilla video and averaged; for every method, including the baselines, the reference is the decoded same-seed vanilla video. Fidelity differences between methods use the same paired, prompt-set-stratified bootstrap ($B$=10,000) as the VBench differences: on the $102$-prompt subset, FSC-Quality$-$MotionCache is SSIM $+0.031$ $[+0.021, +0.042]$, PSNR $+1.04$ dB $[+0.81, +1.26]$, and LPIPS $-0.024$ $[-0.034, -0.015]$, and FSC-Speed$-$MotionCache is SSIM $+0.019$ $[+0.008, +0.030]$, PSNR $+0.53$ dB $[+0.34, +0.73]$, and LPIPS $-0.005$ $[-0.015, +0.004]$. FSC-Quality outperforms MotionCache on $72$ (SSIM), $83$ (PSNR), and $66$ (LPIPS) of the $102$ prompts, and FSC-Speed on $62$, $71$, and $44$. On VBench (all $505$ evaluated Bench prompts) neither difference is resolved: FSC-Quality$-$MotionCache $+0.000$ $[-0.006, +0.006]$ and FSC-Speed$-$MotionCache $-0.001$ $[-0.006, +0.005]$. Our development set (Secs. 4.2 and 5.3) consists of eight prompts, the first prompt of each of eight benchmark prompt sets.

**SkyReels-V2 Generation Settings.** Checkpoint `Skywork/SkyReels-V2-DF-1.3B-540P`; resolution $544 \times 960$ (540P); $177$ frames at $24$ fps; $50$ denoising steps (FlowUniPC scheduler, shift $8.0$); diffusion-forcing settings `ar_step`$=5$, `causal_block_size`$=5$, `base_num_frames`$=97$, `overlap_history`$=17$, `addnoise_condition`$=20$; classifier-free guidance $6.0$; seed $1024$ for every method and prompt.

**MAGI-1 Generation Settings.** Checkpoint MAGI-1 4.5B (distilled); resolution $720 \times 720$; $168$ frames at $24$ fps; $64$ denoising steps; chunk-wise autoregression with the model's native runtime; seed $1024$. Each video runs as a single-GPU subprocess with vendor-recommended offload settings.

**FLOPs Accounting.** On SkyReels-V2 the executed-token trace of every transformer block is recorded for all $946$ prompts after the token-cache warmup; the untraced warmup and overlap-prefix forwards are full computations and are added for every method, vanilla included (five full forwards per temporal group for FlowCache and MotionCache, i.e., their warmup $4$ plus the cache-initializing step that follows it; two for FSC and the SeaCache port). PFLOPs follow from the per-token cost of the block's linear layers and self-attention over the group's own tokens, with the unconditional pass charged as the conditional one; attention to cached context keys and cross-attention are not charged, so SkyReels-V2 PFLOPs are comparable across methods but are not a complete count and are not on the same basis as the MAGI-1 values. The MAGI-1 runtime exposes no such trace, so we instrument its block stack to count executed token-slots and attended query–key pairs per forward and compare them against the vanilla schedule replayed for the same step. The analytic vanilla

cost is 140.6 PFLOPs per video (12,150 tokens per chunk, seven chunks, 64 steps, a four-chunk denoising window with the model's `noise2clean_kvrange`; linear layers 55.9, self-attention 84.2, cross-attention with caption projections 0.5 PFLOPs), 1.2% above the 139 PFLOPs reported by MotionCache for the identical setting, and the instrumented counter reproduces it exactly (1.000) on a vanilla run. Each cached method's PFLOPs is this vanilla cost times its measured executed fraction: FlowCache on the full run, MotionCache and both FSC points on the same instrumented 20-prompt sample (both FSC points were instrumented separately with the same counter; FSC-Quality's per-prompt latencies reproduce its benchmark run within 1%). Sparse forwards are charged for executed query tokens only, matching the SkyReels-V2 convention; charging the full reassembled key/value length instead changes the MAGI-1 fractions by less than 0.03.

**Timing Procedure.** Latency is wall-clock time of the complete generation call, bracketed by `torch.cuda.synchronize()`, measured per prompt. Speedup is computed per prompt as the same-prompt vanilla latency divided by the method latency on the same GPU model, then averaged over prompts; vanilla latencies are measured once per prompt and cached. Because latency is sensitive to co-located jobs on shared servers, all timed runs are launched on otherwise-idle GPUs, and a post-hoc audit flags any run whose latency leaves the per-configuration clean band (the fastest-quintile latency +2%; on idle GPUs the latency of a fixed workload varies by less than 1%). The flagged SkyReels-V2 runs (348 of the 946 runs of the $W{=}5$, $r{=}1.65$ operating point of Table 6, generated while other jobs shared its GPUs, and one FSC-Speed run) were re-measured on idle GPUs, after which every vanilla, FlowCache, and FSC run lies inside its band and all speedups of Table 6 average over all 946 prompts. MotionCache's and SeaCache-AR's executed compute varies more across prompts, which places 95 and 530 of their runs above this band; measured instead against the latency predicted from each run's executed compute, no MotionCache run and five SeaCache-AR runs are more than 2% slower, and excluding those five would change SeaCache-AR's speedup by 0.001×. On MAGI-1, where cached latency depends strongly on the prompt, the band is taken around each prompt's latency predicted from the other cached methods; it flags 8 of 4,730 cached runs, none of them from the two FSC points of Table 1, and without it FlowCache would print 1.485×.

**Peak Memory.** Table 1 reports `torch.cuda.max_memory_allocated` measured in a fresh process per method on an otherwise idle GPU: the pipeline (DiT, text encoder, VAE) is loaded (14.8 GB), the peak counter is reset, and the generation call is bracketed by `cuda.synchronize`; the value is identical on every prompt tested (spread $<$ 0.02 GB). MAGI-1 values come from the benchmark runs themselves (one subprocess per video, counter reset before model load). On SkyReels-V2 1.3B the numbers are 53.7 / 58.1 / 58.9 / 58.4 GB (vanilla / FlowCache / MotionCache / both FSC operating points), and 58.6 GB for the chunk-wise SeaCache port of App. E.3 (recorded inside a benchmark worker process, so an upper bound). The 4.4–5.2 GB above vanilla is primarily shared token-caching state and auxiliary tensors inherited from the common caching stack; FSC's additional controller state is small relative to this total.

**Hardware Portability.** Table 5 repeats the SkyReels-V2 comparison on an H100 80GB under the same generation protocol and seed (speedups against the H100's own vanilla latency and fidelity against the H100's own same-seed vanilla output, since vanilla videos differ across GPUs). Vanilla is 1.22× faster on the H100 (300 vs. 367 s), but the relative ordering is unchanged: at MotionCache-matched latency (2.26–2.29× on both cards) FSC has higher mean SSIM/PSNR and lower mean LPIPS (higher SSIM on $7/10$ and $18/23$ prompts, lower LPIPS on $5/10$ and $15/23$), and its peak memory differs from MotionCache's by at most 0.52 GB on both machines. The H100 columns use two other configurations of the same controller: the 23-prompt column uses FSC-Quality's anchor schedule ($W{=}6$, $n{=}10$) at MotionCache-matched latency with base thresholds $(0.12, 0.20, 0.30, 0.70)$, $r{=}1.9$, $q_{\text{scale}}{=}0.8$ and $p_{\text{LF}}{=}0.1$ (all other values as in FSC-Quality), and the 10-prompt column uses the $W{=}7$, $n{=}9$ configuration of Table 6.

On MAGI-1 all three cached methods peak at 19.7 GB against 31.8 GB for the vanilla runtime: the shared FlowCache/MotionCache runtime on which all three are built drops nearly-clean chunks from the attention window (`discard_nearly_clean_chunk`, its released default), which lowers the footprint for FlowCache, MotionCache, and FSC alike, and the three cache states themselves are indistinguishable at this granularity. FlowCache's KV-cache compression, one of its two components, is disabled in every FlowCache row (the shared SkyReels-V2 runtime does not implement it; on MAGI-1 it is left off, as for every other method), so these rows measure FlowCache's reuse

Table 5: **At MotionCache-matched latency, FSC's fidelity advantage has the same sign on both GPUs.** Same code on two GPUs (SkyReels-V2 1.3B, seed 1024; speedup and fidelity against the local vanilla; paired FSC−MotionCache differences with win counts). RTX columns: the FSC-Quality point of Table 1 (946 prompts; fidelity on its 102-prompt subset) and a separate 8-prompt re-measurement: eight prompts of the 102-prompt fidelity subset, one per MotionCache-protocol dimension and disjoint from the development set; vanilla, released MotionCache ($\tau$=0.10) and FSC-Quality were generated in one session on the same GPU, and fidelity is against that session's vanilla with the ablation-harness metrics of Table 3, so this column is comparable neither with the 946-prompt column nor with the development-set Table 11; RTX peak memory (fresh-process protocol above) was measured for the 946-prompt column only; H100 columns: the $W$=7, $n$=9 configuration (10 prompts) and FSC-Quality's anchor schedule with $q_{\text{scale}}$=0.8 (23 prompts; see text); H100 peak memory comes from the benchmark worker processes, and in the 23-prompt column MotionCache ran after FSC in the same process and inherited FSC's per-window cache-clearing flag, which lowers its resident memory (58.4 vs. 59.0 GB) but not its outputs; $W$: warmup, $n$: anchors, $q_{\text{scale}}$: repair-fraction scale.

| | RTX PRO 6000 (946) | RTX PRO 6000 (8) | H100 (10) | H100 (23) |
|---|---|---|---|---|
| Vanilla latency (s) | 367 | 367 | 300 | 300 |
| MotionCache speedup | 2.275× | 2.285× | 2.278× | 2.259× |
| FSC speedup | 2.275× | 2.274× | 2.280× | 2.277× |
| ΔSSIM (FSC−MC) | +0.031 (72/102) | +0.053 (8/8) | +0.021 (7/10) | +0.028 (18/23) |
| ΔLPIPS (FSC−MC) | −0.024 (66/102) | −0.060 (6/8) | −0.009 (5/10) | −0.021 (15/23) |
| Peak mem. vanilla / MC / FSC (GB) | 53.7 / 58.9 / 58.4 | – | 53.7 / 59.0 / 58.5 | 53.7 / 58.4 / 58.5 |

policy alone. Compression would lower FlowCache's memory below the shared baseline and may also reduce its attention cost; Table 1 credits neither effect.

**FSC Hyperparameters (SkyReels-V2).** *FSC-Quality*: warmup $W = 6$; $n = 10$ dense-to-sparse anchors with last eligible step $E = 39$, giving $\mathcal{A} = \{6, 7, 8, 10, 13, 17, 21, 27, 32, 39\}$ (Eq. 1); threshold scale $r = 1.65$; base thresholds $(b_{\text{skip}}, b_1, b_2, b_{\text{full}}) = (0.16, 0.24, 0.34, 0.80)$ with phase shaping $\varphi$ that scales them by $(0.7, 0.7, 0.8, 0.9)$ for $s < 20$, 1.0 for $20 \le s < 30$, and $(1.4, 1.4, 1.3, 1.2)$ for $s \ge 30$; risk weights $(w_{\text{HF}}, w_{\text{LF}}, w_{\text{age}}) = (1.0, 0.5, 0.25)$; maximum token age $k_{\text{tok}} = 8$; consecutive-replay cap $k_{\text{step}} = 4$; level-specific repair fractions $(\hat{q}_1, \hat{q}_2, \hat{q}_3) = (0.05, 0.10, 0.15)$ with $q_{\text{scale}} = 1.25$; LF gate enabled with $p_{\text{LF}} = 0.2$; overdue relaxation $\eta = 0.8$ (App. B.3). *FSC-Speed*: identical except $W = 5$, hence $\mathcal{A} = \{5, 6, 7, 10, 13, 16, 21, 26, 32, 39\}$, and $r = 1.55$. The LF band uses a $2 \times 2$ average pool (Eq. 2); proxy tokens use the backbone patch layout (2,040 tokens per latent frame; a temporal group of five latent frames yields 10,200 tokens per decision); controller state is per (CFG phase, temporal group).

**FSC Hyperparameters (MAGI-1).** *FSC-Quality*: warmup 4; fifteen dense anchors $\{6, 8, 10, 12, 14, 17, 20, 24, 28, 32, 37, 42, 48, 54, 60\}$ on MAGI's global denoising-step counter (a backbone-specific discrete approximation of the dense-to-sparse schedule rather than Eq. 1 verbatim, which is why $a_0 \neq W$ here); $r = 1.25$ on the MAGI-native base threshold $\beta = 0.0065$ with $w_{\text{HF}} = 1.0$, and $k_{\text{tok}} = 4$. *FSC-Speed*: identical except $k_{\text{tok}} = 8$. The twelve-anchor configuration of Table 7 uses anchors $\{7, 8, 10, 12, 16, 20, 26, 32, 39, 46, 54, 62\}$ (gaps $1, 2, 2, 4, 4, 6, 6, 7, 7, 8, 8$), $r = 1.0$ and $k_{\text{tok}} = 8$. The configuration files also set $w_{\text{LF}}$ (0.5; 0.8 for the twelve-anchor configuration) and $q_{\text{scale}}$ (1.0; 1.25 for FSC-Speed), which have no effect on MAGI-1 (App. B.5). These values map onto MAGI's native reuse thresholds as in App. B.5.

**FSC Hyperparameters (FramePack and Matrix-Game).** FramePack: $W$=2, anchors $\{2, 4, 9, 15, 24\}$, $(k_{\text{tok}}, k_{\text{step}})$=$(8, 4)$, $r$=1.65, $q_{\text{scale}}$=1.25, phase-shaping boundaries at steps 10 and 15 of its 25 steps (instead of 20 and 30); Matrix-Game: $W$=6, anchors $\{6, 7, 8, 9, 11, 14, 17, 20, 24, 29, 34, 39\}$, $(k_{\text{tok}}, k_{\text{step}})$=$(8, 4)$, $r$=2.5, $q_{\text{scale}}$=1.0; all other values as in FSC-Quality on SkyReels-V2 (calibration in App. E.2).

# E BASELINES AND OPERATING POINTS

**Baselines and Their Reproduction.** MotionCache is run from its official repository with the released backbone-specific configuration (SkyReels-V2: threshold 0.1, warmup 4, $K$=6, $\alpha$=0.6; MAGI-1: rel-$L_1$ 0.0065, warmup 5, token-wise reuse on), on the same machine, seed, and prompts as all other rows; the absolute speedups reported in the MotionCache paper correspond to a different hardware and runtime setting and are therefore not comparable to Table 1, whereas the ordering of methods is preserved (verified on an H100 in Table 5). FlowCache is reproduced inside the same shared FlowCache/MotionCache codebase as its chunk-wise special case—the configuration the MotionCache paper identifies as FlowCache ($K$=17 in its sweep, i.e., the coarse full update applied throughout generation, without token-level selectivity): on SkyReels-V2 a group-level reuse/recompute gate with FlowCache's threshold 0.1 and MotionCache's warmup 4; on MAGI-1 the chunk-wise rel-$L_1$ policy with FlowCache's official VBench setting (threshold 0.01, warmup 5) and token-wise reuse disabled. FlowCache's KV-cache compression is disabled (App. D), so these rows reproduce its reuse policy only. The FlowCache paper reports larger speedups (2.38× on MAGI-1, 6.7× on SkyReels-V2) in its own evaluation setting and configuration. In our setting, the official FlowCache repository's SkyReels-V2 stack without compression is 2.21× faster than its own vanilla on a probe prompt (368 vs. 166 s), close to our reproduction (180 s, 2.04×) at comparable fidelity (SSIM 0.758 vs. 0.756); the gap to the published SkyReels-V2 number therefore comes from differences in evaluation setting and configuration, not from the reimplementation. We therefore evaluate all rows on a common stack because the official stack's *vanilla* output diverges from the MotionCache-fork vanilla at the same seed (SSIM 0.92 between the two references), which would make same-seed fidelity incomparable across rows. ARCache (Nan et al., 2026), which evaluates FramePack-F1, SkyReels-V2, and the original Matrix-Game, has no public implementation as of September 2026 and reports results under different settings (e.g., 257 frames in 30 steps on SkyReels-V2 and the GameWorld benchmark on Matrix-Game), so it is not included.

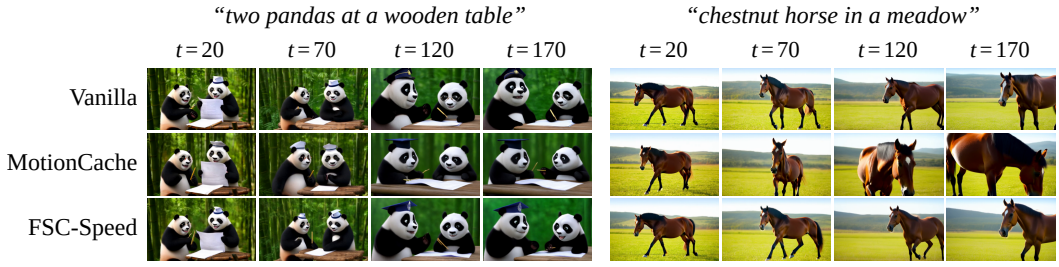



Figure 7: **FSC-Speed exhibits smaller deviation from the vanilla trajectory than MotionCache over time.** SkyReels-V2 1.3B, two prompts, same seed; rows: vanilla, MotionCache, and FSC-Speed; columns: frames at four timestamps. Early frames are nearly identical and deviations accumulate with time: MotionCache drifts in object identity and layout (left: the cap morphs and the paper drifts away) and in framing and color (right: progressive zoom and saturation).

## E.1 OPERATING POINTS AND SPEED–QUALITY FRONTIER

Table 6 lists the SkyReels-V2 FSC operating points used elsewhere in the paper, and Fig. 8 plots FSC-Quality and the three faster $W$=5 points (FSC-Speed with $r$=1.55, $r$=1.65, and $r$=1.65 with two tail anchors at steps 44 and 47) against MotionCache. Every FSC point improves PSNR and SSIM over MotionCache at a VBench aggregate within 0.005 of it while running 0–7.5% faster, and all but the fastest also improve LPIPS. Dense-to-sparse anchors control the *global* refresh frequency and selective token repair recovers *locally* stale content between anchors, so FSC can spend computation continuously along the frontier instead of choosing between full reuse and full recompute.

**MAGI-1 Operating Points.** Table 7 lists the three FSC configurations evaluated on the full MAGI-1 benchmark. The three configurations trace different operating points of MAGI-1's native cache interface. The fifteen-anchor configurations provide the strongest fidelity, while increasing $k_{\text{tok}}$ from 4 to 8 gives a faster operating point; Table 1 reports the two fifteen-anchor points (FSC-Quality with $k_{\text{tok}}$=4; FSC-Speed with $k_{\text{tok}}$=8). On the 110 fidelity prompts, where every

Table 6: **Every SkyReels-V2 FSC operating point improves PSNR and SSIM over MotionCache at a VBench aggregate within $0.005$ of it.** Full $946$-prompt benchmark, ordered by speedup (VBench on the fixed dimension subset of App. D; PSNR/SSIM/LPIPS on the $102$-prompt fidelity subset against the same-seed vanilla output). Configurations are described relative to FSC-Quality ($W$=6, $n$=10 dense-to-sparse (D2S) anchors, $r$=1.65, $q_{\text{scale}}$=1.25, LF gate on with $p_{\text{LF}}$=0.2). The two $W$=5 rows without tail anchors are separate runs that differ only in $r$ (1.55 vs. 1.65); the tail-anchor point adds exact steps $44$ and $47$ to the $r$=1.65 row. Adding tail anchors changes the realized repair routing because anchors reset controller state: risk-triggered full repairs and $C2$ repairs drop from $5.7\%$ and $2.9\%$ to 0, and $C3$ from $8.6\%$ to $3.0\%$, hence the higher speed ($C0$: whole-step replay; $C1$–$C3$: token repair of increasing fraction; Full: full repair; $W$: warmup; $n$: anchors; $r$: threshold scale). Best in **bold**, second best underlined.

| Configuration (relative to FSC-Quality) | Speedup↑ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|
| FSC-Quality (Table 1) | 2.275× | **0.7900** | **18.38** | **0.8209** | **0.3122** |
| $W$=7, $n$=9, $r$=1.75, $q_{\text{scale}}$=1.00, $p_{\text{LF}}$=0.1 | 2.277× | 0.7853 | 18.31 | 0.8202 | 0.3199 |
| $W$=5, $r$=1.55 (FSC-Speed, Table 1) | 2.356× | 0.7892 | 17.87 | 0.8088 | 0.3313 |
| $W$=5 | 2.378× | 0.7892 | 17.88 | 0.8091 | 0.3316 |
| $W$=5, tail anchors at steps 44 and 47 | **2.445**× | 0.7854 | 17.79 | 0.8073 | 0.3464 |
| MotionCache | 2.275× | 0.7898 | 17.34 | 0.7899 | 0.3366 |

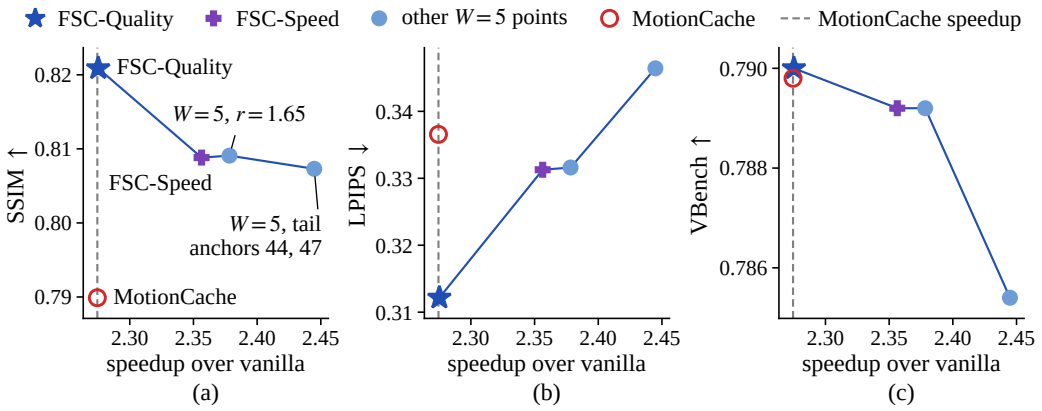



Figure 8: **Every FSC operating point has higher SSIM than MotionCache at equal or higher speed, with VBench within $0.005$.** SkyReels-V2, full benchmark (VBench on the fixed dimension subset of App. D; SSIM and LPIPS on the $102$-prompt fidelity subset); FSC-Quality, FSC-Speed, and the two other $W$=5 points of Table 6 (line: FSC points in speed order); the dashed line marks MotionCache's speedup.

accelerated output has a same-seed vanilla reference, the VBench aggregate of every cached method (computed over the $60$ of these prompts that carry the evaluated dimensions) lies within the vanilla bootstrap CI (paired VBench differences: FSC-Quality−MotionCache $-0.009$ $[-0.030, +0.016]$, FSC-Quality−vanilla $+0.013$ $[-0.005, +0.032]$). On this distilled backbone the opportunity for reuse between scheduled full recomputations is limited by the $16$ denoising steps a chunk spends at each of the four window positions ($64$ steps per chunk in total), which is why the FSC frontier on MAGI-1 spans a narrower speed range than on SkyReels-V2.

### E.2 Cross-Family Validation on FramePack and Matrix-Game

*Backbones.* FramePack-F1 (`FramePack_F1_I2V_HY_20250503`, HunyuanVideo-based next-section prediction; $25$ steps, distilled guidance $10$, $145$ frames at the $640$ bucket, seed $31337$, image-to-video from frame $0$ of the SkyReels-V2 vanilla video) and the Matrix-Game-3.0 base model (Wan2.2-5B based; $50$ steps, CFG $5.0$, shift $5.0$, $704 \times 1280$, $177$ frames, seed $42$; held-out prompts are likewise conditioned on frame $0$ of the SkyReels-V2 vanilla video). *Port.* The full

Table 7: **On MAGI-1, every FSC configuration is faster than MotionCache with higher SSIM and lower LPIPS.** Configurations evaluated over the full benchmark prompt pool; speedup uses the vanilla-clean prompt pool after per-method latency auditing, VBench the 500 prompts of the evaluated subset, fidelity the first ten prompts of each prompt set (110; App. D). ΔVBench is relative to MotionCache (0.7950); its paired bootstrap 95% CI over prompts contains zero for every row ($[-0.010, +0.004]$ for the twelve-anchor configuration, $[-0.007, +0.008]$ for FSC-Quality, $[-0.013, +0.002]$ for FSC-Speed). $k_{\text{tok}}$: forced full-recomputation period; $r$: threshold scale (App. B.5). Best in **bold**, second best underlined; the ΔVBench marks rank the FSC rows only (MotionCache is the reference).

| Configuration (relative to FSC-Quality) | Anchors | $k_{\text{tok}}$ | Speedup↑ | ΔVBench↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|
| $r$=1.0, 12 anchors, $k_{\text{tok}}$=8 | 12 | 8 | 1.393× | −0.0029 | 0.8564 | 0.2722 |
| FSC-Quality (Table 1): dense anchors, $r$=1.25 | 15 | 4 | 1.421× | **+0.0004** | **0.8883** | **0.2297** |
| FSC-Speed (Table 1): $k_{\text{tok}}$=8 | 15 | 8 | **1.457**× | −0.0054 | 0.8782 | 0.2460 |
| MotionCache | – | – | 1.357× | – | 0.8468 | 0.2778 |

controller runs with gathered-token execution as on SkyReels-V2 (selected tokens run every block on a compact batch and attend to the full cached keys and values), with separate controller state per CFG branch. *Calibration.* FSC is calibrated per backbone on eight development prompts over $W$, $n$, $E$, $r$, $q_{\text{scale}}$ and $(k_{\text{tok}}, k_{\text{step}})$; a configuration is eligible if its median latency is within 5% of the transferred SkyReels configuration, its LPIPS is not worse, and its PSNR is at most 0.1 dB lower; the eligible configuration with the highest SSIM is selected; the final configurations are listed in App. D. The coupled comparator is the purely reactive variant of Sec. 5.3: the same controller with warmup but no scheduled anchors, so risk-triggered full repairs also re-base the reference; its threshold scale is calibrated on the development prompts toward FSC's latency ($r$=0.9 and 0.543). *Evaluation.* Each backbone is evaluated on 40 held-out prompts stratified over the benchmark prompt sets and never used for calibration; fidelity is SSIM/PSNR/LPIPS to the same-seed vanilla output (the implementation of Table 1); CIs are stratified bootstrap ($B$=10,000) and wins are per-prompt. *Timing.* Latency ratios and speedups come from a same-GPU re-timing in which each prompt's controllers run back to back on one idle GPU; vanilla, re-timed on five prompts per backbone, agrees with the original timing within 0.2%. The re-timing reproduces the fidelity of the validation runs (mean $|\Delta\text{SSIM}| \leq 0.006$ on FramePack, 0 on Matrix-Game), and its latency ratios agree with those of the validation runs, which used several GPUs, to within 0.001. *Transferred configuration.* Before calibration, the SkyReels-V2 FSC-Quality configuration was transferred by scaling its step-count parameters to the sampler length, without tuning, and validated on the same 40 prompts against a coupled comparator (median latency ratios 1.02 and 0.99). FSC minus coupled was then ΔSSIM $+0.064$ $[+0.047, +0.083]$ and ΔLPIPS $-0.057$ $[-0.069, -0.046]$ on Matrix-Game, and ΔSSIM $+0.026$ $[+0.018, +0.034]$ and ΔPSNR $+1.46$ dB $[+0.82, +2.19]$ with an unresolved ΔLPIPS $-0.006$ $[-0.017, +0.004]$ on FramePack. Without calibration the separation result thus already holds on Matrix-Game in all metrics and on FramePack in SSIM and PSNR but not LPIPS; calibration also improves FSC's own LPIPS (calibrated minus transferred FSC: LPIPS $-0.022$ $[-0.031, -0.012]$ on FramePack and $-0.013$ $[-0.017, -0.009]$ on Matrix-Game). Table 8 reports the results. The coupled comparator is about 3% faster than FSC on both backbones (same-GPU median ratios 0.9685 and 0.9667), inside the pre-registered $[0.90, 1.10]$ band, and FSC is better on both backbones in all three metrics. On these two backbones, gathered token repair is close to compute-proportional (per-step cost divided by the repaired fraction 0.69–0.87 on FramePack and 0.58–0.96 on Matrix-Game), unlike SkyReels-V2 (Observation 2).

### E.3 Matched-Latency Comparison with FlowCache

FlowCache's released operating point (2.04×) is 12% slower than FSC-Quality, whose latency equals MotionCache's. To compare at equal latency in both directions we (i) sped FlowCache up to FSC-Quality's latency by raising its reuse threshold ($0.10 \rightarrow 0.123$, chosen by a three-prompt latency calibration) and (ii) slowed FSC down to FlowCache's latency by adding anchors to FSC-Quality and lowering its threshold scale (13 D2S anchors, $r$=1.40; all other hyperparameters unchanged, same calibration procedure), and evaluated everything on the 235-prompt stratified subset on which all methods have results, giving a paired VBench comparison on the subset's 126

Table 8: **On FramePack and Matrix-Game, FSC outperforms a latency-matched coupled comparator in PSNR, SSIM, and LPIPS, with every 95% CI excluding zero.** Cross-family validation on 40 held-out prompts per backbone: FSC minus comparator, mean with stratified bootstrap 95% CI and prompt wins (FSC/comparator). Latency ratio is the same-GPU median per-prompt comparator/FSC latency (each prompt's controllers run back to back on one idle GPU; bootstrap 95% CI), and speedup is end-to-end relative to the same-seed vanilla run; speedups are means of per-prompt ratios and can differ from the median latency ratio. Pre-registered matching criterion: median ratio in $[0.90, 1.10]$. Coupled: the purely reactive variant of Sec. 5.3 (the same controller without scheduled anchors, one risk signal triggering both repair and full synchronization); $r$: its threshold scale.

| Backbone | Comparator | Speedup FSC/comp.↑ | Latency ratio | ΔPSNR (dB) | ΔSSIM | ΔLPIPS |
|---|---|---|---|---|---|---|
| FramePack | coupled ($r$=0.9) | 2.473×/2.513× | 0.9685 [0.9676, 0.9707] | +5.18 [+4.03, +6.35], 40/0 | +0.079 [+0.065, +0.094], 40/0 | −0.092 [−0.105, −0.080], 39/1 |
| Matrix-Game | coupled ($r$=0.543) | 2.445×/2.535× | 0.9667 [0.9629, 0.9692] | +2.77 [+2.53, +3.00], 40/0 | +0.058 [+0.043, +0.075], 36/4 | −0.057 [−0.067, −0.047], 39/1 |

prompts that carry the evaluated dimensions (124 for the slower block; Table 9). At FSC-Quality's latency FlowCache's VBench aggregate (0.7922) stays within the CI of FSC-Quality's (+0.004 $[-0.003, +0.011]$) and of MotionCache's (+0.004 $[-0.004, +0.012]$), while its fidelity is far below FSC-Quality's (SSIM 0.772 vs. 0.828). The slower FSC point has the highest SSIM and PSNR (0.860, 20.07 dB; SeaCache's LPIPS is lower), yet its VBench aggregate is 0.012 below FlowCache's (bootstrap 95% CI $[-0.022, -0.003]$) and 0.004 below FSC-Quality on the 124 prompts both were scored on (Table 12). Additional repair computation therefore improves same-seed fidelity without a corresponding VBench change: the fast operating points lie within one VBench bootstrap CI of each other, whereas the fidelity ordering is unambiguous. The same table includes SeaCache (Chung et al., 2026), which has no autoregressive implementation; we port its scheduling rule onto the same stack in the chunk-wise form of FlowCache (a group-level reuse/recompute gate driven by the accumulated rel-$L_1$ of the SEA-filtered block-0 input, filter coefficients from each group's own noise level, first two steps and the last step exact, its threshold $\delta$ calibrated to the two latencies). At FSC-Quality's latency SeaCache behaves like MotionCache (SSIM 0.783, VBench 0.791 on this subset; in Table 1 the same $\delta$=0.28 point gives 2.320×, VBench 0.7886, SSIM 0.7919, LPIPS 0.3331); at the slower latency it matches the slow FSC point on fidelity (SSIM 0.854 vs. 0.860, but lower LPIPS, 0.236 vs. 0.244, on the 21 prompts with references) with a higher aggregate (+0.006 $[+0.0001, +0.012]$). FSC's fidelity advantage over the spectral SeaCache-AR scheduler is thus largest at the faster operating point, which is the regime the matched-speed comparison of Table 1 targets; over FlowCache it is larger at the slower point.

Table 9: **At both latencies, the FSC point has the highest PSNR and SSIM.** Paired comparison on the 235-prompt stratified subset (SkyReels-V2, seed 1024; VBench on the 126 subset prompts carrying the evaluated dimensions, 124 for the slower block, with bootstrap 95% CIs over prompts; fidelity on the 21 prompts of the subset that have vanilla references). SeaCache-AR is our chunk-wise port of SeaCache's scheduling rule (see text); thr, $\delta$: FlowCache and SeaCache-AR reuse thresholds; $r$: FSC threshold scale. Best in **bold**, second best underlined within each block.

| Method | Latency (s)↓ | Speedup↑ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|
| *faster operating point* | | | | | | |
| MotionCache | 161 | 2.274× | 0.7885 [0.758, 0.817] | 17.55 | 0.7789 | 0.3156 |
| FlowCache (thr 0.123) | 159 | 2.305× | **0.7922** [0.762, 0.821] | 17.46 | 0.7715 | 0.3125 |
| SeaCache-AR ($\delta$=0.28) | **158** | **2.321**× | 0.7908 [0.761, 0.820] | 17.68 | 0.7826 | 0.3222 |
| FSC-Quality | 161 | 2.275× | 0.7883 [0.757, 0.817] | **18.87** | **0.8277** | **0.2840** |
| *slower operating point* | | | | | | |
| FlowCache (thr 0.10, released) | **180** | **2.037**× | **0.7916** [0.759, 0.821] | 17.80 | 0.7825 | 0.2891 |
| SeaCache-AR ($\delta$=0.22) | 188 | 1.947× | 0.7855 [0.752, 0.816] | 19.82 | 0.8539 | **0.2357** |
| FSC-Quality, 13 anchors, $r$=1.40 | 182 | 2.015× | 0.7796 [0.746, 0.810] | **20.07** | **0.8600** | 0.2435 |

### E.4 SEED ROBUSTNESS

We repeated the SkyReels-V2 comparison with four additional seeds (7, 42, 123, 2026) on a 78-prompt subset stratified over the benchmark prompt sets, generating vanilla, MotionCache, and FSC-Quality for every seed and comparing each accelerated video with the vanilla video of the *same* seed (Table 10). Fidelity margins replicate at every seed and are significant on the 78 paired prompts of each added seed (SSIM $+0.021$ to $+0.044$, PSNR $+0.9$ to $+1.7$ dB, LPIPS $-0.016$ to $-0.043$; paired $t$-test $p \leq 1.6 \times 10^{-5}$ for SSIM and PSNR and $p \leq 0.022$ for LPIPS, the largest values both at seed 42; FSC wins on 52–71 of 78 prompts; on the 13 prompts of seed 1024, $p = 0.044$ / $0.001$ / $0.098$ for SSIM / PSNR / LPIPS); over the five seeds the mean margin is SSIM $+0.035 \pm 0.009$, PSNR $+1.25 \pm 0.31$ dB, LPIPS $-0.031 \pm 0.011$, at identical speed ($2.274 \pm 0.002\times$ for MotionCache, $2.274 \pm 0.001\times$ for FSC; per-seed ratio $1.000 \pm 0.001$). The seed-to-seed standard deviation of the VBench aggregate is 0.003–0.008 per method; on this subset FSC is $0.008 \pm 0.009$ below MotionCache on average, from $-0.019$ (seed 1024) to $+0.003$ (seed 42), whereas on the full benchmark the same difference is $+0.000$ (paired bootstrap 95% CI $[-0.006, +0.006]$)—the 78-prompt subset weights the evaluated dimensions differently from the full benchmark, and MotionCache scores above vanilla on it at two of the five seeds. These results indicate that the fidelity advantage persists across seeds, whereas the VBench ranking between accelerated methods remains sensitive to prompt-subset composition—differences of $\sim 0.01$ change sign and size with the subset. An additional 98-prompt pool (subject-consistency, overall-consistency, and scene prompts) gives the same fidelity ordering: FSC$-$MotionCache SSIM $+0.033$ at seed 7 and $+0.034$ at seed 42.

Table 10: **FSC-Quality's fidelity advantage over MotionCache replicates at all five seeds at identical speed.** Seed robustness on the 78-prompt subset stratified over the benchmark prompt sets (SkyReels-V2). VBench aggregate per seed for all three methods; speedup of MotionCache / FSC-Quality over the same-seed vanilla; same-seed fidelity difference FSC-Quality$-$MotionCache with the number of prompts on which FSC-Quality wins; $n$ = prompts with a same-seed vanilla reference (78 per added seed; for seed 1024 the 13-prompt overlap with the fidelity subset, scored with the in-pipeline metrics rather than the offline metrics of the added seeds). Mean $\pm$ sd rows use the unrounded per-seed values.

| | Speedup | | VBench (41 of the 78 prompts) | | | FSC $-$ MotionCache, same-seed fidelity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Seed | MC | FSC | vanilla | MC | FSC | $n$ | $\Delta$PSNR (wins) | $\Delta$SSIM (wins) | $\Delta$LPIPS (wins) |
| 1024 | 2.274 | 2.274 | 0.7802 | 0.7892 | 0.7698 | 13 | $+1.06$ (11/13) | $+0.040$ (9/13) | $-0.028$ (9/13) |
| 7 | 2.275 | 2.275 | 0.7748 | 0.7829 | 0.7736 | 78 | $+1.71$ (71/78) | $+0.044$ (63/78) | $-0.043$ (57/78) |
| 42 | 2.275 | 2.274 | 0.7875 | 0.7851 | 0.7879 | 78 | $+0.91$ (55/78) | $+0.021$ (55/78) | $-0.016$ (52/78) |
| 123 | 2.271 | 2.273 | 0.7898 | 0.7826 | 0.7835 | 78 | $+1.38$ (65/78) | $+0.036$ (65/78) | $-0.039$ (56/78) |
| 2026 | 2.276 | 2.275 | 0.7855 | 0.7827 | 0.7701 | 78 | $+1.20$ (69/78) | $+0.033$ (67/78) | $-0.028$ (55/78) |
| mean $\pm$ sd | $2.274 \pm 0.002$ | $2.274 \pm 0.001$ | $0.7836 \pm 0.0060$ | $0.7845 \pm 0.0028$ | $0.7770 \pm 0.0082$ | | $+1.25 \pm 0.31$ | $+0.035 \pm 0.009$ | $-0.031 \pm 0.011$ |

### E.5 MOTIONCACHE THRESHOLD SWEEP

Table 1 runs MotionCache with its released configuration. To test whether this favors FSC-Quality, we sweep MotionCache over its reuse threshold $\tau$, the parameter its release documents as controlling reuse aggressiveness (a token is recomputed once its accumulated motion-weighted distance reaches $\tau$); warmup 4, $K$=6, $\alpha$=0.6 and all other released settings are kept. *Development sweep.* On our development set (same stack, seed, GPU and metrics), nine thresholds span 1.96–2.62$\times$ (Table 11, Fig. 9); the released $\tau$=0.10 lies at FSC-Quality's latency, and at equal latency every FSC point of App. E.12 has higher SSIM than the MotionCache sweep ($+0.015$ to $+0.034$). *Operating-Point Selection.* Before the evaluation below, three operating points were frozen from the development latencies alone: the threshold closest to FlowCache's speed (2.04$\times$), the one closest to FSC-Quality's latency (within 3%), and the one 5–8% faster than FSC-Quality, giving $\tau$=0.075, 0.10 and 0.11. The latency-matched point is the released threshold. *Evaluation.* The frozen points are evaluated on the 235-prompt stratified subset of App. E.3 (all outside the threshold-sweep set; VBench on 124: the subset's 126 prompts carrying the evaluated dimensions minus two scene prompts without a 13-anchor score; fidelity on the 21 prompts that have a vanilla reference video and FSC-Quality fidelity), with the paired bootstrap of App. E.3. The regenerated $\tau$=0.10 point matches the benchmark's MotionCache row to within 0.003 on VBench ($+0.0029$ $[+0.0001, +0.0063]$, a resolved but small difference). At matched latency FSC-Quality keeps its fidelity advantage (SSIM $+0.049$ on $16/21$ prompts, PSNR $+1.3$ dB, LPIPS $-0.032$) while the VBench difference is unresolved (MotionCache $-$ FSC-Quality $+0.0030$ $[-0.0070, +0.0136]$); the fidelity advantage also holds against

Table 11: **On the development prompts, MotionCache's released $\tau$=0.10 lies at FSC-Quality's latency, where FSC-Quality has higher SSIM.** MotionCache threshold sweep on our development set (eight prompts; seed 1024, one idle GPU). Paths: share of group decisions that reuse / fully update / selectively update. $^{s,m,f}$: frozen slow, matched and fast points. The FSC-Quality row is run 1 of App. E.12; Table 3 lists the stored reference run (LPIPS 0.2925). s/video: median wall-clock. Best in **bold**, second best underlined.

| $\tau$ | s/video↓ | Speedup↑ | PSNR↑ | SSIM↑ | LPIPS↓ | Paths (%) |
|---|---|---|---|---|---|---|
| 0.07 | 188 | 1.960× | 17.89 | 0.6569 | **0.2806** | 54/33/13 |
| $0.075^{s}$ | 181 | 2.026× | 17.67 | 0.6399 | 0.3040 | 54/28/18 |
| 0.08 | 177 | 2.062× | 17.71 | 0.6433 | 0.3073 | 54/26/20 |
| 0.085 | 175 | 2.100× | 17.89 | 0.6483 | 0.2960 | 54/27/19 |
| 0.09 | 165 | 2.210× | 17.49 | 0.6391 | 0.3058 | 62/27/11 |
| $0.1^{m}$ | 162 | 2.267× | 17.42 | 0.6380 | 0.3113 | 61/23/16 |
| $0.11^{f}$ | 152 | 2.412× | 17.17 | 0.6113 | 0.3540 | 64/22/14 |
| 0.12 | 146 | 2.522× | 17.46 | 0.6174 | 0.3447 | 67/22/11 |
| 0.13 | **140** | **2.616**× | 16.47 | 0.6072 | 0.3564 | 70/21/9 |
| FSC-Quality | 161 | 2.273× | **18.92** | **0.6641** | 0.2926 | – |

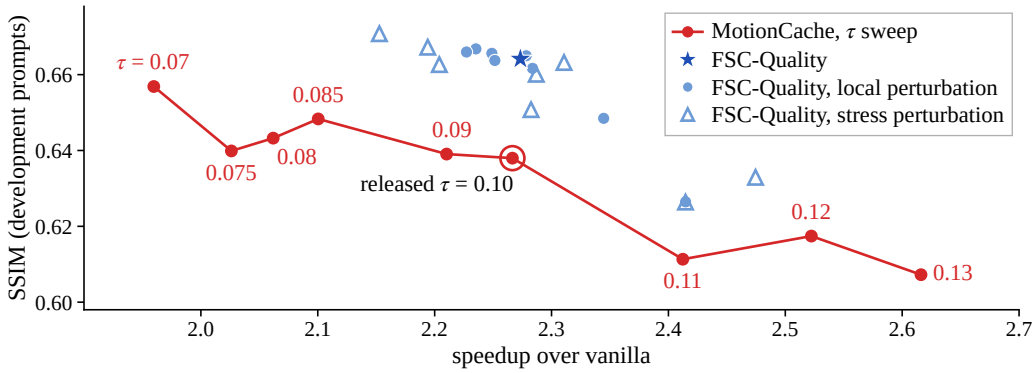



Figure 9: **On the development prompts, every perturbed FSC-Quality point has higher SSIM than the MotionCache threshold sweep at equal latency.** SkyReels-V2, our development set (eight prompts); MotionCache's $\tau$ sweep of Table 11 (released $\tau$=0.10 marked), FSC-Quality, and its local and stress perturbations of App. E.12.

the faster and the slower frozen points (Table 12). On VBench, the faster MotionCache point scores below FSC-Quality ($-0.013$ $[-0.023, -0.004]$), and the slower one has a higher aggregate than the slowed FSC point ($+0.013$ $[+0.004, +0.024]$). The mechanically selected latency-matched point is the released threshold itself, so Table 1 compares MotionCache at its own latency-matched operating point. Fig. 4(a) plots every method on these 21 fidelity prompts, with speedups averaged over the 235 prompts.

### E.6 Comparison with SCOPE on Its Official Stack

SCOPE (Cui et al., 2026b) is concurrent work whose official implementation (commit `05dc4dd`) provides an adapter for SkyReels-V2-DF-1.3B on the upstream SkyReels-V2 runtime (revision `9351d13`, verified with the authors' checker), not on the shared runtime of Table 1. We run it unmodified with the authors' released SkyReels-V2 configuration ($\tau_c$=0.18, $\tau_p$=0.42, second-order prediction, decay 0.75, skip cap 5, selective computation and sensitivity guard on), loaded from the configuration file written by their own command line, and with our benchmark protocol for everything else: the same checkpoint, 946 prompts, seed, resolution, 177 frames, window, overlap, 50

Table 12: **FSC keeps its fidelity advantage over every frozen MotionCache point outside the sweep set.** Frozen MotionCache operating points on the 235 prompts outside the threshold-sweep set (VBench on the 124 carrying the evaluated dimensions, with bootstrap 95% CI; PSNR/SSIM/LPIPS on the 21 paired fidelity prompts). FSC rows are the stored results of App. E.3. Parentheses: fidelity margin of the nearest-latency FSC point (the 13-anchor point for $\tau$=0.075, FSC-Quality otherwise) and FSC wins / MotionCache wins on the 21 prompts; paired VBench differences: $+0.0134$ $[+0.0037, +0.0240]$, $+0.0030$ $[-0.0070, +0.0136]$, $-0.0127$ $[-0.0227, -0.0035]$ (MotionCache − FSC). s/video: median wall-clock; $\tau$: MotionCache reuse threshold; $r$: threshold scale. Best in **bold**, second best underlined.

| Method | s/video↓ | Speedup↑ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|
| MotionCache $\tau$=0.075 (slow) | 180 | 2.035× | **0.7930** [0.761, 0.823] | 17.84 (FSC +2.22) | 0.7869 (FSC +0.073, 17/4) | 0.2911 (FSC −0.048, 13/8) |
| MotionCache $\tau$=0.10 (matched; released value) | 161 | 2.278× | 0.7870 [0.756, 0.816] | 17.55 (FSC +1.32) | 0.7784 (FSC +0.049, 16/5) | 0.3158 (FSC −0.032, 15/6) |
| MotionCache $\tau$=0.11 (fast) | **151** | **2.425**× | 0.7713 [0.740, 0.801] | 17.29 (FSC +1.58) | 0.7718 (FSC +0.056, 19/2) | 0.3389 (FSC −0.055, 20/1) |
| FSC-Quality | 161 | 2.275× | 0.7840 [0.752, 0.814] | 18.87 | 0.8277 | 0.2840 |
| FSC-Quality, 13 anchors, $r$=1.40 | 182 | 2.015× | 0.7796 [0.746, 0.810] | **20.07** | **0.8600** | **0.2435** |

steps, guidance, shift and negative prompt, the same GPU model on idle devices, the same video writer, VBench-Long evaluation and fidelity implementation. No SCOPE parameter was tuned.

*Cross-Stack Comparability.* The two runtimes do not compute the same vanilla video: the shared runtime does not forward frame groups that have finished denoising, the upstream runtime processes the whole valid interval. At the same seed their vanilla outputs are different samples (SSIM 0.48 between them on the 102 fidelity prompts) of equal quality (VBench aggregate difference $+0.004$ $[-0.023, +0.035]$ on those prompts), and the upstream vanilla takes 589 s per video against 367 s. With every step recomputed, SCOPE's pipeline reproduces the upstream vanilla exactly in the first window (checked on one prompt). We therefore (i) measure SCOPE's same-seed fidelity only against its own-stack vanilla, generated for the 102 fidelity prompts, (ii) normalize its speedup only to that vanilla, and (iii) compare across stacks only what does not depend on the reference trajectory: latency in seconds on the same hardware and the VBench aggregate on the same prompts. Fidelity columns of different stacks therefore measure closeness to different references and should not be interpreted as a common ranking.

*Results* (Table 13). SCOPE is faster in seconds (123 vs. 161 s) and reproduces the speedup its paper reports (4.80× here, 4.73× there). Its VBench aggregate is 0.7748, 0.015 below FSC-Quality ($[-0.024, -0.006]$) and 0.015 below MotionCache ($[-0.024, -0.006]$) in the paired bootstrap over the evaluated prompts; against its own vanilla on the fidelity prompts the difference is $-0.022$ $[-0.054, +0.007]$. Its outputs deviate more from its own vanilla (SSIM 0.510, PSNR 12.1 dB, LPIPS 0.570) than FSC's and MotionCache's outputs deviate from theirs. The SCOPE paper reports higher fidelity (SSIM 0.73) under its own protocol (257 frames, a different prompt set); we did not investigate the difference beyond the authors' own ablation: without selective computation (one flag of the released runner, reported in their Sec. 4.5) SCOPE takes 155 s (3.81×; their paper: 3.72×), is closer to its vanilla on 95 of 102 prompts (SSIM 0.684) and changes VBench by $+0.012$ $[-0.005, +0.028]$ on the 235-prompt subset, where its difference to FSC-Quality is $+0.001$ $[-0.015, +0.015]$. Because the protocols differ substantially, we do not interpret this numerical gap as a reproduction failure.

### E.7 High-Dynamics Stress Analysis

Because VBench's per-video dynamic-degree score is coarse (the fraction of clips labelled dynamic), we score the *vanilla* videos with VBench's own RAFT pipeline and use the mean top-5% flow magnitude (relative to VBench's motion threshold) as a continuous motion measure, then split prompts into terciles. On SkyReels-V2 (Table 14; 99 seed-7 prompts with retained videos: the 98-prompt pool of App. E.4 plus one additional subject-consistency prompt; 13 of the 99 are also in the 78-prompt seed subset) FSC-Quality is ahead of MotionCache in every tercile, and the margin *grows* with motion (SSIM $+0.018$, $+0.035$, $+0.049$; LPIPS $-0.004$, $-0.025$, $-0.051$). FSC's motion attenuation appears where vanilla motion is marginal (the mean dynamic-degree score drops by 0.06 on low-motion prompts) and vanishes in the high-motion tercile. On MAGI-1 (110 prompts; 88 static and 22 dynamic by that label) FSC-Quality leads MotionCache on static prompts (SSIM $+0.040$, LPIPS $-0.048$) and on dynamic ones (SSIM $+0.048$, LPIPS $-0.048$).

Table 13: **SCOPE is faster in seconds than FSC-Quality but lower on VBench; fidelity is not comparable across runtime stacks.** SCOPE on its official stack next to the shared-stack rows of Table 1 (SkyReels-V2 1.3B, same prompts, seed, generation settings and GPU model). **Speedup is normalized to each stack's own vanilla and fidelity is measured against each stack's own same-seed vanilla (**$102$ **prompts)**; only latency and VBench are comparable across stacks. †official ablation of the SCOPE paper, run on the $235$-prompt stratified subset and the fidelity prompts ($316$ videos); its VBench is the paired value on the subset's $126$ evaluated prompts, where the released configuration scores $0.7769$ and FSC-Quality $0.7883$. Best in **bold**, second best underlined, only in the columns comparable across stacks (latency, VBench).

| Runtime stack | Method | Own vanilla (s) | Latency (s)↓ | Speedup (own vanilla)↑ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|---|---|
| shared (Table 1) | MotionCache | 367 | 161 | 2.275× | <u>0.7898</u> | 17.34 | 0.7899 | 0.3366 |
| | FSC-Quality (ours) | 367 | 161 | 2.275× | **0.7900** | 18.38 | 0.8209 | 0.3122 |
| official SCOPE (upstream) | SCOPE, released configuration | 589 | **123** | 4.803× | 0.7748 | 12.10 | 0.5099 | 0.5696 |
| | SCOPE, no selective computation† | 589 | <u>155</u> | 3.813× | 0.7889 | 14.37 | 0.6840 | 0.4380 |

Table 14: **FSC-Quality's fidelity margin over MotionCache grows with motion.** SkyReels-V2 (FSC-Quality vs. MotionCache, MC), seed 7, the 99 retained prompts of App. E.7 split using vanilla-motion tercile boundaries defined on the original $100$-prompt pool (motion: mean top-$5\%$ RAFT flow magnitude relative to VBench's motion threshold). Fidelity against the same-seed vanilla (offline metrics of Table 10, not on the scale of Table 1); $\Delta$DD = mean change of the VBench dynamic-degree score (fraction of clips labelled dynamic) vs. vanilla. The better of the two methods in each fidelity metric is in **bold**.

| | | | MotionCache | | | FSC | | | ΔDD | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tercile | $n$ | motion | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | MC | FSC |
| low | 34 | 0.60 | 18.05 | 0.6388 | 0.2864 | **19.13** | **0.6563** | **0.2820** | $+0.01$ | $-0.06$ |
| mid | 32 | 2.26 | 15.51 | 0.6102 | 0.3722 | **16.61** | **0.6448** | **0.3475** | 0.00 | $-0.02$ |
| high | 33 | 6.70 | 14.41 | 0.5701 | 0.4402 | **15.80** | **0.6192** | **0.3893** | $-0.01$ | 0.00 |

### E.8 Motion Attenuation

On MAGI-1, the VBench dimension on which FSC falls furthest behind MotionCache is *dynamic degree*. There, FSC-Quality scores $0.295$ and FSC-Speed $0.236$ versus $0.365$ for MotionCache and $0.434$ for FlowCache: with only $16$ denoising steps per window position on this distilled backbone, reuse between scheduled full recomputations can damp large motions, and this dimension alone accounts for FlowCache's higher VBench aggregate on MAGI-1. On SkyReels-V2, with $50$ denoising steps per temporal group and dense early anchors, the effect vanishes in aggregate (App. E.7: FSC-Quality's dynamic-degree score is $0.06$ below vanilla on the low-motion tercile and unchanged on the high-motion one): FSC-Quality and MotionCache both score $0.719$ and FSC-Speed $0.722$ (vanilla $0.740$). On MAGI-1, highly dynamic prompts are therefore the regime in which more conservative reuse is most important.

### E.9 No-Retuning Transfer to SkyReels-V2 14B

To test cross-scale transfer, we applied the 1.3B FSC-Quality configuration (App. D) unchanged to SkyReels-V2-DF-14B-540P, without additional hyperparameter search, keeping the same autoregressive settings, seed, and baselines. The 14B model's per-window key/value cache alone ($\sim$84 GB at 540P) does not fit on a single RTX PRO 6000 (102.6 GB) alongside the weights, so all four methods are run at $352 \times 640$ with the runtime's CPU-offload setting, which offloads the text encoder and moves the DiT between CPU and GPU across autoregressive windows; everything else (177 frames, 97-frame window, `ar_step` 5, 50 steps) is identical to Table 1. MotionCache and FlowCache use their released settings with the 14B TeaCache polynomial. Table 15 reports 47 prompts (five per benchmark prompt set, excluding the development prompts). The VBench aggregate of FSC-Quality ($0.841$) is within the paired bootstrap CI of every other method (FSC-Quality$-$MotionCache $+0.008$ $[-0.031, +0.036]$, FSC-Quality$-$FlowCache $+0.011$ $[-0.017, +0.042]$, FSC-Quality$-$vanilla $+0.009$ $[-0.014, +0.031]$).

Table 15: **Unchanged from 1.3B, FSC-Quality is the fastest method on SkyReels-V2 14B, with the highest PSNR and SSIM.** SkyReels-V2 14B, no retuning (47 prompts, $352 \times 640$, seed 1024; fidelity against the same-seed 14B vanilla; VBench on the 25 prompts carrying the evaluated dimensions; memory in $10^9$ bytes on a 102.6 GB card). Best accelerated result in **bold**, second best underlined.

| Method | Latency (s)↓ | Speedup↑ | Rel. FLOPs↓ | Mem (GB)↓ | VBench↑ | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|---|---|
| Vanilla | 696 | 1.000× | 1.000 | 95.1 | 0.8313 | – | – | – |
| FlowCache | 437 | 1.595× | 0.581 | **96.7** | 0.8292 | 16.07 | 0.7692 | **0.3189** |
| MotionCache | 389 | 1.792× | 0.504 | 96.8 | 0.8327 | 15.88 | 0.7600 | 0.3289 |
| FSC-Quality (unchanged from 1.3B) | **328** | **2.121**× | **0.392** | 96.8 | **0.8406** | **16.18** | **0.7745** | 0.3302 |

### E.10 Further Separation Variants and $2 \times 2$ Factorial

Table 16 lists the variants of the separation study that are not in Table 3, on the same development set at matched wall-clock. The $2 \times 2$ factorial crosses the anchor schedule (uniform vs. dense-to-sparse) with the repair controller (off vs. on); anchor counts are chosen so that every cell runs within about $5\%$ of FSC-Quality's latency (the other three cells are rows of Table 3). Both levels contribute: dense-to-sparse placement adds $0.017$ SSIM over uniform anchors without repair, the repair controller adds $0.068$ over uniform anchors alone, and the combination adds $0.088$. Phase shaping mainly affects latency ($< 0.002$ SSIM, $3\%$ latency).

Table 16: **Further variants of the separation study: uniform anchors without repair are outperformed on all eight prompts, while removing phase shaping changes SSIM by at most** $0.002$**.** SkyReels-V2, our development set (eight prompts), seed 1024, median wall-clock per video on one idle GPU; rows modify FSC-Quality (D2S: dense-to-sparse); the purely reactive rows and the other factorial cells are in Table 3. "Collapsed repair thresholds" collapses the $C1$–$C3$ risk thresholds to a single level; safeguard-triggered repairs retain their native repair budget (this is not a fixed-budget controller). The last two columns count the prompts on which FSC-Quality outperforms the variant. Best in **bold**, second best underlined.

| Variant | Anchors | s/video ↓ | SSIM ↑ | LPIPS ↓ | FSC-Quality wins (of 8) | |
|---|---|---|---|---|---|---|
| | | | | | SSIM | LPIPS |
| FSC-Quality: repair controller, D2S anchors | 10 D2S | 161 | 0.6641 | 0.2925 | – | – |
| uniform anchors only (no repair) | 13 uniform | **153** | 0.5766 | 0.3496 | 8 | 8 |
| collapsed repair thresholds | 10 D2S | 165 | 0.6596 | 0.3024 | 5 | 4 |
| no phase shaping | 10 D2S | 166 | **0.6658** | **0.2916** | 2 | 2 |

### E.11 Anchor Placement: Influence Curve and Schedule Search

Two experiments assess the empirical basis for the dense-to-sparse schedule of Eq. 1.

**Influence of a single cached step.** For each step $s \in \{6, 8, \ldots, 38, 39\}$ and four profiling prompts we replay step $s$ from the cache with every other step exact (72 generations) and divide the resulting damage in the final video (LPIPS to the same-seed vanilla) by the teacher-forced local reuse error of that step, giving the influence $g(s)$ of a unit local error (Fig. 10(a)). $g(s)$ shows a strong decreasing trend along the trajectory (Spearman $-0.95$ with $s$; exponential-fit half-life 16 steps; $6.1\times$ from step 8 to 39, $3.8\times$ and $2.9\times$ when damage is measured by $1-$SSIM or relative $L_2$). Local errors are therefore not interchangeable: an early one is amplified several times more than a late one, which is the mechanism behind Observation 1 and the reason anchor gaps should grow along the trajectory; it says nothing about *which* growing schedule to use.

**Schedule search.** We next evaluate whether a local surrogate can recover the schedule. From instrumented full-compute traces of eight profiling prompts we measure, for every candidate anchor $i$ and later step $s$, the mismatch between the cached whole-step residual of $i$ and the exact residual of $s$, and minimize the summed mismatch over ten anchors by dynamic programming, with and without weighting each term by $g(s)$; we also draw 500 random ten-anchor schedules. The hand-

crafted schedule sits in the best $10\%$ of the random schedules but is not the surrogate optimum: the unweighted optimum is nearly uniform (gaps $3, 3, 3, 4, 4, 4, 4, 4, 4$), and the $g$-weighted optima are dense-to-sparse in shape (gaps $2, 3, 3, 3, 4, 4, 4, 5, 5$ and $2, 2, 3, 3, 4, 4, 5, 5, 5$). End-to-end, however, every one of them is outperformed by the hand-crafted schedule with the rest of FSC-Quality held fixed on our development set (Table 17; Fig. 10(b)), and the surrogate mis-orders the schedules it can compare (it ranks uniform ahead of dense-to-sparse). The surrogate-optimal schedules run $2.6\%$ faster; this latency difference can account for at most about $0.008$ SSIM of their $0.008$–$0.020$ deficit. The influence curve therefore motivates growing gaps; the anchor locations themselves are selected empirically.

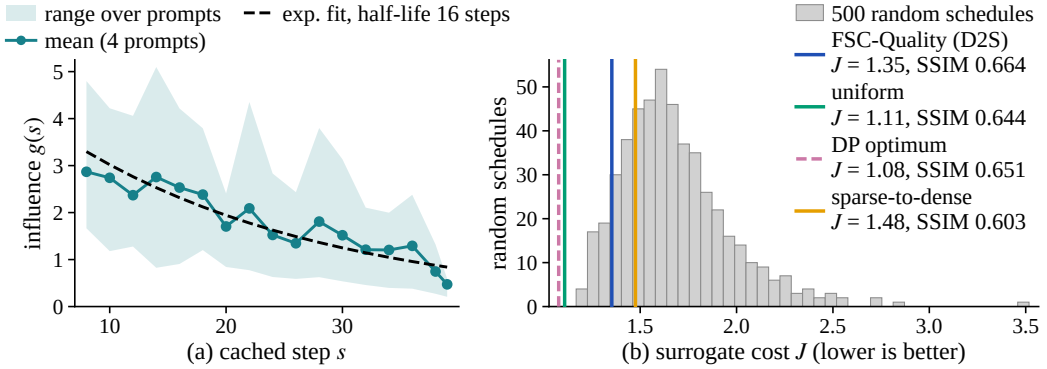



Figure 10: **The influence of a cached step decays along the trajectory, but a local-error surrogate does not select the end-to-end best schedule. (a)** Influence $g(s)$ of a single cached step on the final video (final LPIPS damage per unit local reuse error; mean and range over four profiling prompts, 72 generations): influence decays about $4$–$6\times$ along the trajectory (endpoint ratio $6.1\times$). **(b)** Local-error surrogate cost of 500 random ten-anchor schedules (histogram) and of the hand-crafted, uniform, reversed, and unweighted DP-optimal schedules (lines), each annotated with its end-to-end SSIM on our development set (Table 17); the random schedules are scored under the surrogate only.

Table 17: **Every alternative anchor schedule has lower mean SSIM than FSC-Quality's dense-to-sparse schedule.** Anchor schedules with the rest of FSC-Quality held fixed (SkyReels-V2, our development set of eight prompts, seed 1024, ten anchors each; median s/video). $J$ = local-error surrogate cost (lower is better under the surrogate); DP = surrogate optimum, unweighted or weighted by the measured influence $g(s)$. The last two columns count the prompts on which the hand-crafted schedule wins. The FSC-Quality, uniform, and reversed rows are the runs of Table 3; the DP schedules were generated separately with identical code, seed, and settings, and their win counts are paired against that FSC-Quality run. Best in **bold**, second best underlined.

| Schedule | Gaps | $J\downarrow$ | s/video↓ | SSIM ↑ | LPIPS ↓ | FSC-Quality wins (of 8) | |
|---|---|---|---|---|---|---|---|
| | | | | | | SSIM | LPIPS |
| FSC-Quality: dense-to-sparse (Eq. 1) | 1,1,2,3,4,4,6,5,7 | 1.353 | 161 | **0.6641** | **0.2925** | – | – |
| DP, $g$-weighted (1−SSIM) | 2,3,3,3,4,4,4,5,5 | 1.126 | **157** | 0.6558 | 0.3019 | 5 | 5 |
| DP, unweighted | 3,3,3,4,4,4,4,4,4 | **1.077** | **157** | 0.6510 | 0.3104 | 5 | 5 |
| DP, $g$-weighted (LPIPS) | 2,2,3,3,4,4,5,5,5 | 1.146 | **157** | 0.6442 | 0.3197 | 6 | 6 |
| uniform | 4,3,4,4,3,4,4,3,4 | 1.108 | **157** | 0.6441 | 0.3189 | 5 | 5 |
| sparse-to-dense (reversed) | 7,5,6,4,4,3,2,1,1 | 1.476 | 161 | 0.6033 | 0.3659 | 8 | 8 |

### E.12 Local Hyperparameter Sensitivity

Around FSC-Quality we vary one parameter at a time, with every other parameter held fixed: the threshold scale $r$ ($\pm 15\%$), $q_{\text{scale}}$ ($\pm 20\%$), and the relative weights $w_{\text{LF}}/w_{\text{HF}}$ and $w_{\text{age}}/w_{\text{HF}}$ ($\pm 50\%$, with $w_{\text{HF}}=1$ as the normalization), plus a wider stress endpoint on each axis. All 144 runs use our development set and one idle GPU; FSC-Quality is run twice (both runs take the same routing

Table 18: **Six of the eight local perturbations of FSC-Quality change SSIM by at most** $0.0028$ **and latency by at most** $1.8\%$**.** Local sensitivity around FSC-Quality (SkyReels-V2, our development set of eight prompts): $\Delta$ relative to the mean of the two FSC-Quality runs, routing shares over non-anchor decisions, and W/L = prompts on which the point outperforms/underperforms FSC-Quality; local: $r \pm 15\%$, $q_{\text{scale}} \pm 20\%$, relative weights $\pm 50\%$; stress: a wider endpoint on each axis; routing levels: $C0$ whole-step replay, $C1$–$C3$ token repair of increasing fraction, Full full repair (Sec. 4.3). s/video: median wall-clock. Best perturbed point in **bold**, second best underlined.

| Point | Role | s/video↓ | Δlat. % | PSNR↑ | SSIM↑ | ΔSSIM | LPIPS↓ | ΔLPIPS | C0/C1/C2/C3/Full % | SSIM W/L | LPIPS W/L |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FSC-Quality (run 1) | – | 161 | +0.0 | 18.92 | 0.6641 | +0.0002 | 0.2926 | −0.0002 | 70/12/3/9/6 | 7/1 | 5/3 |
| FSC-Quality (run 2) | – | 161 | +0.0 | 18.90 | 0.6638 | −0.0002 | 0.2931 | +0.0002 | 70/12/3/9/6 | 1/7 | 3/5 |
| $r = 1.4$ | local | 164 | +1.7 | 18.91 | 0.6668 | +0.0028 | 0.2905 | −0.0024 | 68/10/4/12/6 | 7/1 | 5/3 |
| $r = 1.9$ | local | 156 | −3.2 | 18.76 | 0.6485 | −0.0155 | 0.3172 | +0.0243 | 71/12/4/11/3 | 1/7 | 1/7 |
| $q_{\text{scale}} = 1$ | local | 160 | −0.5 | 18.83 | 0.6617 | −0.0023 | 0.2985 | +0.0056 | 70/12/3/9/6 | 2/6 | 2/6 |
| $q_{\text{scale}} = 1.5$ | local | 163 | +1.1 | 18.89 | 0.6656 | +0.0017 | 0.2887 | −0.0042 | 70/12/3/9/6 | 7/1 | 6/2 |
| $w_{\text{LF}} = 0.25$ | local | 152 | −5.6 | 18.43 | 0.6265 | −0.0375 | 0.3432 | +0.0503 | 71/14/3/12/0 | 0/8 | 0/8 |
| $w_{\text{LF}} = 0.75$ | local | 164 | +1.8 | 18.89 | 0.6660 | +0.0020 | 0.2906 | −0.0023 | 67/10/5/12/6 | 7/1 | 6/2 |
| $w_{\text{age}} = 0.125$ | local | 161 | +0.0 | 18.92 | 0.6650 | +0.0010 | 0.2920 | −0.0009 | 71/12/3/9/6 | 6/2 | 5/3 |
| $w_{\text{age}} = 0.375$ | local | 163 | +1.0 | 18.89 | 0.6637 | −0.0002 | 0.2938 | +0.0009 | 68/12/4/10/6 | 3/5 | 3/5 |
| $r = 1.25$ | stress | 167 | +3.3 | 18.89 | 0.6672 | +0.0033 | 0.2844 | −0.0085 | 65/11/6/10/7 | 6/2 | 6/2 |
| $r = 2.05$ | stress | 152 | −5.7 | 18.46 | 0.6264 | −0.0376 | 0.3426 | +0.0497 | 71/13/4/12/0 | 0/8 | 0/8 |
| $q_{\text{scale}} = 0.75$ | stress | 160 | −0.6 | 18.81 | 0.6601 | −0.0039 | 0.3002 | +0.0073 | 70/12/3/9/6 | 1/7 | 0/8 |
| $q_{\text{scale}} = 1.75$ | stress | 161 | −0.4 | 18.65 | 0.6508 | −0.0132 | 0.3268 | +0.0339 | 70/12/3/12/3 | 0/8 | 0/8 |
| $w_{\text{LF}} = 0$ | stress | **148** | −8.2 | 18.53 | 0.6329 | −0.0310 | 0.3368 | +0.0439 | 76/12/5/6/0 | 0/8 | 0/8 |
| $w_{\text{LF}} = 1$ | stress | 170 | +5.6 | **18.94** | **0.6708** | +0.0068 | **0.2757** | −0.0172 | 65/11/5/10/9 | 6/2 | 6/2 |
| $w_{\text{age}} = 0$ | stress | 159 | −1.6 | 18.87 | 0.6632 | −0.0008 | 0.2942 | +0.0013 | 74/10/4/7/6 | 3/5 | 2/6 |
| $w_{\text{age}} = 0.5$ | stress | 167 | +3.6 | 18.84 | 0.6626 | −0.0013 | 0.2952 | +0.0023 | 62/14/6/12/6 | 4/4 | 2/6 |

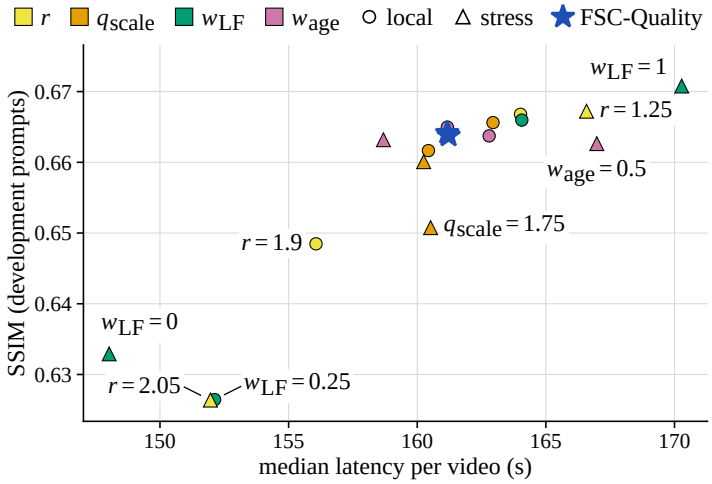



Figure 11: **Perturbations toward more repair are marginally better at slightly higher latency, and the larger deviations toward less repair trade SSIM for lower latency.** SSIM against median latency per video around FSC-Quality (SkyReels-V2, our development set; stars: the two FSC-Quality runs; circles: local perturbations; triangles: stress perturbations; color: perturbed parameter; values in Table 18).

decisions; mean $|\Delta\text{SSIM}|$=0.0008 between the two runs). Table 18 lists every point. Six of the eight local points change SSIM by at most 0.0028, LPIPS by at most 0.0056 and latency by at most $1.8\%$; over all eight local points the maxima are 0.0375, 0.0503 and $5.6\%$. The two larger local deviations ($r$=1.90 and $w_{\text{LF}}$=0.25) trade 0.015–0.038 SSIM for 3–$6\%$ lower latency. Perturbations toward more repair are marginally better at slightly higher latency ($r$=1.40: +0.0028 SSIM at $+1.7\%$ latency; $w_{\text{LF}}$=0.75: +0.0020 at $+1.8\%$; Fig. 11).