ONE THRESHOLD IS NOT ENOUGH: PROMPT-INVARIANT CACHING SCHEDULES FOR VIDEO DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Video diffusion models remain expensive because denoising repeatedly evaluates a costly model backbone. Dynamic caching methods like TeaCache, EasyCache, and DiCache reduce this cost by reusing prior outputs while an accumulated drift signal remains below a threshold τ. Although existing methods improve the drift signal, they hold τ fixed throughout denoising. We show that fixed thresholds are structurally suboptimal: varying τ across denoising achieves quality–latency trade-offs unattainable by any fixed threshold across three caching methods with distinct drift signals. We further show that, surprisingly, each method's drift trajectory is nearly prompt-invariant across seven method–model combinations. The shape of the drift trajectory is determined by the model and caching method; therefore, a threshold schedule can be calibrated offline and reused across generations. Based on these findings, we present ACID (Adaptive Caching for vIDeo generation), which uses a low threshold during critical regions where drift changes rapidly and a high threshold across stable regions. ACID identifies these regions offline from the drift signal's second derivative, requires no training, and adds negligible runtime overhead. Across TeaCache, EasyCache, and DiCache on HunyuanVideo, Wan 2.1, and CogVideoX, ACID pushes the speed–quality Pareto frontier beyond fixed thresholds. On TeaCache with HunyuanVideo, it achieves $2.16\times$ speedup over no caching and 38% additional speedup over a conservative fixed threshold, with less than 0.3 dB PSNR, 0.01 SSIM, and 0.01 LPIPS degradation relative to that fixed-threshold configuration.

1 INTRODUCTION

Video generation models (Zheng et al., 2024; Ma et al., 2024a; Brooks et al., 2024; Wang et al., 2025; Hong et al., 2023; Kong et al., 2024) are rapidly being used to generate realistic videos (Zheng et al., 2025); however, their slow inference speed and high compute requirements (3 to 23 minutes for $\sim$5-second videos on a single A100 GPU) pose an impediment to their wide adoption (Li et al., 2024b). As these models scale to generate even higher resolutions, the challenges associated with slow inference speed and high computation requirements will be further exacerbated. The primary reason for these slow inference speeds and high computation needs is the core inference loop of these video generation models (Zheng et al., 2024; Ma et al., 2024a; Brooks et al., 2024; Wang et al., 2025; Hong et al., 2023; Kong et al., 2024): a computationally expensive diffusion backbone that runs a sequential denoising procedure, dominating generation time.

Training-free methods accelerate video generation by reducing the cost of denoising (Zhang et al., 2025; Xi et al., 2025; Ren et al., 2025; Lu et al., 2022; Liu et al., 2025a). Caching methods are appealing because they are lightweight and easy to deploy. These methods exploit similarity across diffusion steps to reuse previously computed representations. Early caching methods reuse representations at fixed intervals. Recent methods instead adapt their caching decisions at runtime (Kahatapitiya et al., 2024; Liu et al., 2025a; Zhou et al., 2025; Liu et al., 2025b; Bu et al., 2025). At each step, they estimate the change in model state and use it to decide whether to reuse a cached representation or recompute it.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

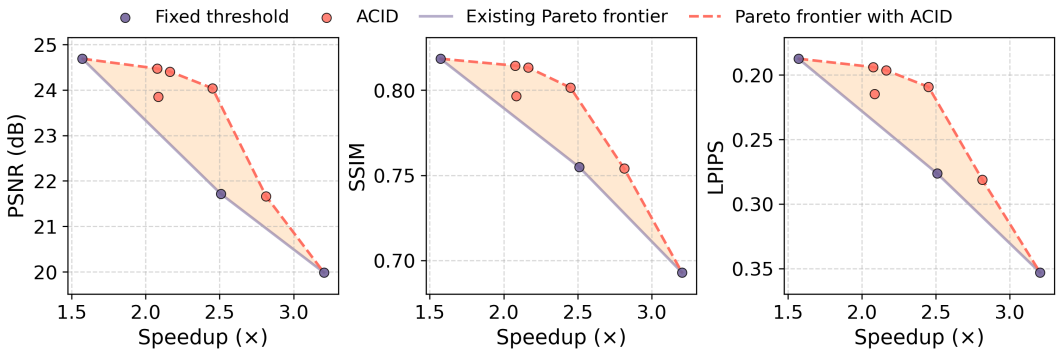


Figure 1: TeaCache on HunyuanVideo. ACID reaches quality–latency operating points beyond those attainable by any fixed-threshold configuration.

Popular caching methods, including TeaCache (Liu et al., 2025a), EasyCache (Zhou et al., 2025), and DiCache (Bu et al., 2025), share the same basic structure. Each accumulates an estimate of the change in model output. When the accumulated drift remains below a user-defined threshold τ, the method skips the expensive model evaluation and reuses the previous output with a correction. When the drift exceeds τ, the method runs the full model and resets the accumulator.

Within this shared structure, successive methods differ primarily in how they estimate drift. TeaCache uses changes in the model input. EasyCache instead estimates the transformation rate from input change to output change. DiCache uses shallow features from the model's early layers. Despite these different drift signals, all three compare the accumulated drift against a single user-defined threshold τ. This threshold remains fixed throughout denoising.

Choosing τ exposes a tradeoff between output quality and inference latency. A high threshold caches aggressively, reducing latency at the cost of output quality. A low threshold preserves quality but provides less acceleration. This raises a natural question: *Do fixed thresholds achieve the best possible quality–latency tradeoff?* We show that they do not. Varying τ across denoising reaches operating points unattainable by any fixed threshold, as Figure 1 shows. This result holds for TeaCache, EasyCache, and DiCache despite their different drift signals. The limitation therefore lies not only in how drift is estimated, but also in applying a single threshold throughout denoising. Threshold scheduling provides a complementary axis for improving the quality–latency tradeoff.

Varying τ requires deciding when to change it. At first glance, this appears to require a runtime policy that adapts the threshold to each prompt. However, across 98 prompts and seven method–model combinations, each method's drift signal follows a nearly identical trajectory across prompts (Figures 2 and 3). Because the trajectory depends on the model and caching method, both fixed at deployment, the threshold schedule can be computed once offline and reused across generations.

We attribute the gains from threshold scheduling to *critical steps*, regions of denoising where the drift signal changes rapidly. Aggressive caching during these regions causes disproportionate quality loss. We therefore use a low threshold at critical steps and a high threshold elsewhere. Because critical steps occur in consistent regions of the trajectory, their boundaries can be identified offline and reused across generations.

We demonstrate these findings with ACID (Adaptive Caching for vIDeo generation), a minimal threshold schedule. ACID applies a low threshold τ_{low} at critical steps and a high threshold τ_{high} across the stable interior. It identifies these regions offline using the second derivative of the caching method's drift signal. This simple design is training-free, signal-agnostic, and adds negligible runtime overhead.

Across three caching methods and three video diffusion models, ACID pushes the speed–quality Pareto frontier beyond fixed thresholds. On TeaCache with HunyuanVideo, it achieves a $2.16\times$ speedup over no caching and 38% additional speedup over a conservative fixed threshold, with less than 0.3 dB PSNR, 0.01 SSIM, and 0.01 LPIPS degradation relative to the latter.

We summarize our contributions below:

- We show that the caching threshold τ, held fixed by every dynamic caching method for video diffusion, is an independent axis of improvement from the drift signal those methods have competed on. Varying τ across denoising reaches quality–latency operating points that no fixed τ attains, and it does so for TeaCache, EasyCache, and DiCache alike, despite their unrelated drift signals. This indicates the limitation is structural rather than a deficiency of any single signal.
- We show, across 98 prompts and all seven method–model combinations we evaluate, that each method's drift signal traces essentially the same trajectory regardless of the prompt, and that its shape is instead characteristic of the model and the caching method. Since both are fixed at deployment, the threshold schedule can be computed once offline rather than inferred at runtime.
- We identify *critical steps* in the sequential denoising procedure, timesteps at which the drift signal changes rapidly and aggressive caching disproportionately degrades quality, and show that they occupy consistent regions of the trajectory that can be located offline from the second derivative of the drift signal.
- We demonstrate both findings with ACID, a minimal threshold schedule requiring no training, no changes to the underlying caching method, and negligible runtime overhead. Across three caching methods and three video diffusion models, ACID pushes the achievable speed–quality Pareto frontier beyond any fixed threshold, with the largest gains on TeaCache and HunyuanVideo.

2 Related Work

Diffusion Models. While U-Net-based diffusion models like Stable Diffusion (Rombach et al., 2022) and SDXL (Podell et al., 2023) successfully catalyzed high-quality image and video generation (Guo et al., 2023; Chen et al., 2024a; Blattmann et al., 2023), their limited architectural capacity bottlenecks the training and deployment of massive models. To eliminate this constraint, recent frameworks have rapidly shifted toward the Diffusion Transformer (DiT) backbone, exploiting its structural flexibility to support next-generation foundation models (Esser et al., 2024; Black Forest Labs, 2024; Chen et al., 2023; Peebles & Xie, 2023). However, despite their theoretical promise, scaling these DiT backbones introduces substantial computational overhead.

Diffusion Model Acceleration. As DiT backbones scale in capacity and complexity, their runtime increases proportionately, exacerbating the notoriously slow inference speeds of diffusion models and impeding real-world deployment. To alleviate this latency bottleneck, several acceleration paradigms have emerged. Recent work has extensively explored efficient attention (Dao et al., 2022), sparse attention (Xi et al., 2025; Ren et al., 2025), distillation (Meng et al., 2023; Sauer et al., 2024; Wang et al., 2023), quantization (Li et al., 2024a; Chen et al., 2025), and improved SDE/ODE solvers (Lu et al., 2022). Alongside these methods, caching-based feature reuse strategies offer a particularly compelling, orthogonal path forward, which we detail next.

Caching-based feature reuse strategies. Caching methods have gained attention recently due to their lightweight nature. DeepCache (Ma et al., 2024c) and Faster Diffusion (Li et al., 2023) reuse U-Net features across timesteps to reduce computational redundancy. FORA (Selvaraju et al., 2024), Δ-DiT (Chen et al., 2024b), and Learning-to-Cache (Ma et al., 2024b) extend this idea to transformer-based architectures. PAB (Zhao et al., 2024) broadcasts attention features to later timesteps in a pyramid-style method based on various block characteristics. AdaCache (Kahatapitiya et al., 2024) modifies residual reuse methods based on content complexity. FasterCache (Lv et al., 2024) proposes to cache for both the conditional and unconditional branch of Classifier Free Guidance (Ho & Salimans, 2022). TeaCache (Liu et al., 2025a), EasyCache (Zhou et al., 2025), and DiCache (Bu et al., 2025) represent the most recent state-of-the-art dynamic caching approaches; we examine their shared structure and its key limitation in the paragraphs below.

The Fixed Threshold of Dynamic Feature Caching Methods. Most recent dynamic caching-based feature reuse strategies make caching decisions based on the diffusion process's state at runtime. Methods like TeaCache, EasyCache, and DiCache share a common algorithmic skeleton: a running accumulated drift variable is initialized to zero, an estimate of output change is added at each step,

and this accumulated value is compared against a fixed threshold τ. When the accumulated drift is below τ, the expensive model forward pass is skipped and the previous output is reused with some correction. When the accumulated drift exceeds τ, the full model is run and the accumulator is reset. The methods differ in how they estimate drift and how they reconstruct the output on skipped steps, which we delve into below.

Warmup Phases. Each of these methods runs the full model for a hardcoded number of initial steps before caching begins. This is effectively a coarse threshold schedule, with $\tau = 0$ early and $\tau > 0$ afterward, where the phase boundary is fixed per model and never varied per prompt. The practice is treated as an implementation detail rather than a design principle, and the threshold is held constant everywhere outside this prefix.

TeaCache. TeaCache estimates drift using the L1 distance between consecutive values of the Timestep Embedding Modulated Noisy Input (TEMNI), a combination of the text embedding, timestep embedding, and current noisy input. Rather than measuring how much the model output changed, which would require running the model, TeaCache measures how much the model's inputs changed, exploiting the empirical observation that large input changes correlate with large output changes. On skipped steps, TeaCache reuses the previous output with a stored residual correction.

EasyCache. EasyCache estimates drift using the model's transformation rate k, defined as the ratio of output change to input change at the most recently computed step. The authors observe that k is unstable early in the denoising process but quickly stabilizes, reflecting the model settling into a roughly linear input-output regime. Under this regime, EasyCache estimates the current step's output change as $k \times \Delta\text{input}/\|v_{t-1}\|$, where Δinput is the current input change, $\|v_{t-1}\|$ is the magnitude of the previous output, and k is the transformation rate of the last fully computed step. This normalized estimate is accumulated against τ. On skipped steps, EasyCache reconstructs the output as the current input plus the offset $\Delta = \text{output} - \text{input}$ from the last fully computed step, which is justified because a stable k implies the model's additive contribution is approximately constant.

DiCache. DiCache replaces the input-level proxy used by TeaCache and EasyCache with a feature-level proxy. At each step, DiCache runs only the first few transformer blocks of the model as a shallow probe and measures how much those features changed. Because these early layers are cheap to compute, this provides a direct estimate of whether the full model's output would change significantly, without paying the cost of a full forward pass. On skipped steps, DiCache uses the trajectory of shallow probe features across multiple recent steps to interpolate a better estimate of the current deep output.

Despite these differences, all three methods share a critical limitation: the fixed user-provided threshold is held constant throughout the entire denoising process. This means the caching schedule cannot respond to the diffusion process's varying dynamics. Note that all three improve the drift estimate while leaving the threshold untouched; our work shows that the threshold is itself an independent axis of improvement, and that because the drift signal is consistent across prompts, a schedule for varying it can be computed once offline rather than inferred at runtime.

3 ACID

In this section, we introduce ACID, a method for varying caching frequency across the denoising process.

3.1 Analysis

Two properties of caching signals. To understand the limitations of fixed-threshold caching, we analyze how caching signals evolve throughout the denoising process. Figures 2 and 3 show each method's drift signal measured across 98 T2V-CompBench prompts, for every method–model combination we evaluate. Two properties are apparent.

First, the signal is consistent across prompts. The 98 prompts are drawn evenly from the seven compositional categories of T2V-CompBench (Sun et al., 2025). Within any single panel, the

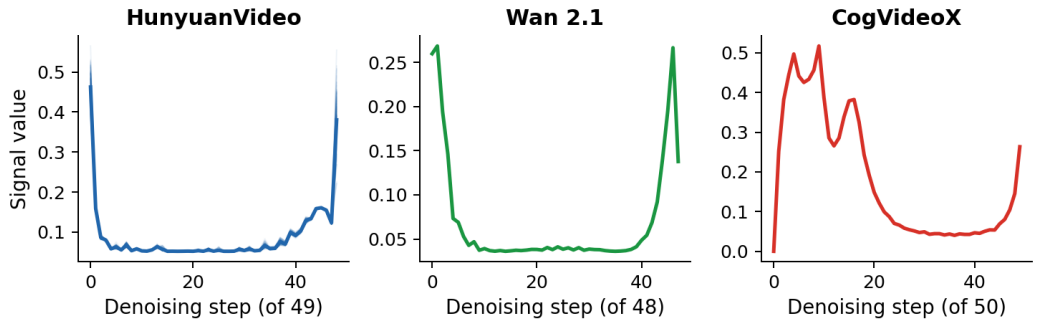


Figure 2: TeaCache drift signal across 98 T2V-CompBench prompts; faint lines are individual prompts, the bold line is the mean. Trajectories are nearly indistinguishable within a panel but differ across panels, showing the signal reflects the model rather than the prompt. EasyCache and DiCache appear in Figure 3.

trajectory produced by a visually complex, fast-moving scene is nearly indistinguishable from that of a static, minimal one.

Second, the shape of the signal is characteristic of the model and the caching method. HunyuanVideo and Wan 2.1 under TeaCache exhibit a U-shaped trajectory, falling steeply from a high initial value, remaining flat through the middle of denoising, and rising again near the end. CogVideoX under TeaCache shows a visibly different structure, with an extended high-signal region early in denoising. The same model produces different shapes under different caching methods, since each method measures a different quantity.

Taken together, these properties mean the trajectory of the drift signal is fixed once the model and the caching method are chosen, both of which are decided at deployment time rather than per generation. Anything we derive from the shape of this trajectory can therefore be computed ahead of time.

We note that the drift signal is a proxy: each method measures something cheap in place of the model output change it cannot observe without running the model. Figure 4 in Appendix A compares TeaCache's ΔTEMNI against the actual relative change in model output on HunyuanVideo, confirming that the proxy tracks the quantity it is meant to estimate.

Why one threshold is not enough. As Figure 2 shows, there are certain parts of the diffusion process where the drift signal is relatively high, and other parts where it is low. Dynamic feature caching methods already take this into account: when the predicted Δoutput is greater, they cache less often, because they hit the fixed threshold more often, and when the predicted Δoutput is smaller, they cache more often. But while dynamic caching methods do find a better speed–quality tradeoff than the no-caching baseline, we find that this tradeoff is not optimal: the degree to which they cache more in stable zones or cache less in unstable zones is suboptimal, a limitation imposed by the single fixed threshold. We find that using different thresholds for different parts of the diffusion process, such as low thresholds for unstable regions and high thresholds for stable regions, leads to a better speed–quality tradeoff.

We could attribute the gains from varying the threshold to a suboptimal metric for TeaCache, but we importantly find the same gains for EasyCache and DiCache as well, which use entirely different caching signals. This suggests the limitation is structural: any method that applies a single fixed threshold throughout the denoising process will face this suboptimality, regardless of the signal it uses.

Locating the transition points. The previous observations establish that multiple thresholds help, but not where along the trajectory the threshold should change. We address this in two stages: first establishing that some transition points outperform every fixed threshold, then asking what property of the signal predicts where those points fall.

Algorithm 1 Offline Schedule Calibration

Require: Model f_θ, base method $\mathscr{M}$, calibration prompts P, sampling steps T
1: **for** each prompt $p \in P$ **do**
2: Run f_θ without caching, recording $s_t \leftarrow \mathscr{M}.\text{signal}(x_t, t, c_p)$ for $t = T, \ldots, 0$
3: **end for**
4: $\bar{s} \leftarrow$ mean signal trajectory over P *// prompt-invariant by Figure 2*
5: $\bar{s}'' \leftarrow$ second difference of $\bar{s}$
6: $R_{\text{crit}} \leftarrow \{t : \bar{s}''$ has not yet stabilized$\}$ *// critical steps at trajectory ends*
7: **return** R_{crit}

For the first stage, we ran a calibration study using TeaCache on HunyuanVideo, Wan 2.1, and CogVideoX. For each model we evaluated a low fixed threshold, a high fixed threshold, and a range of multi-threshold configurations in which we varied the steps at which the threshold changes. Figure 5 in Appendix A shows the ΔTEMNI signal for each model, with dashed lines marking the transition points at which a multi-threshold configuration achieved a better speed–quality tradeoff than both fixed-threshold baselines. Such points exist for all three models, confirming that the gains are not specific to a particular model's signal shape.

For the second stage, we asked what distinguishes these transition points from any other step. Examining the second derivative of the signal (Figure 6), we found that they coincide with the steps at which the second derivative stabilizes, marking where the signal passes from a rapidly changing regime into a stable one. This gives a criterion for placing the transitions that depends only on the shape of the drift signal, which the previous properties established can be measured offline.

This criterion was derived from TeaCache alone. We applied it unchanged to EasyCache and DiCache, whose drift signals measure entirely different quantities, and obtained schedules that expand the quality–latency frontier in every method–model combination we evaluate (Appendix B). That a criterion calibrated on one signal transfers to two unrelated ones supports the view that it is capturing a property of the denoising process rather than an artifact of any single metric.

3.2 Threshold Schedules

ACID extends the single-threshold framework of methods like TeaCache, EasyCache, and DiCache with a threshold schedule. Rather than applying a single fixed threshold τ at every step, we partition the denoising trajectory into regions and assign a threshold to each. We use two threshold values: a low threshold τ_{low}, which caches conservatively and preserves fidelity, and a high threshold τ_{high}, which caches aggressively and reduces latency.

The regions follow the structure of the drift signal. Steps at which the signal changes rapidly are *critical steps*, where approximating the model output is most damaging to quality, and these receive τ_{low}. Steps in the stable interior of the trajectory, where the signal is flat and successive outputs differ little, receive τ_{high}. For the U-shaped signals of HunyuanVideo and Wan 2.1 this yields a low-high-low schedule, since the signal changes rapidly at both ends of denoising. For signals with a different structure the schedule follows accordingly.

Because the drift signal is fixed once the model and caching method are chosen, the boundaries between these regions are determined once, offline, by the calibration procedure of Algorithm 1, and reused for every subsequent generation. At inference time (Algorithm 2) the underlying method's accumulate-and-compare loop is unchanged except that the threshold it compares against is read from the schedule rather than fixed. ACID therefore requires no retraining, no modification to how a method estimates drift or reconstructs skipped outputs, and adds negligible runtime overhead.

Algorithm 2 Inference with a Threshold Schedule

Require: Sampling steps T, base method $\mathscr{M}$, thresholds τ_{low}, τ_{high}, critical step set R_{crit}

Initialize accumulated drift $\Sigma \leftarrow 0$
for $t = T$ **to** 0 **do**
  if $\mathscr{M}$.warmup(t) **then**
    $y_t \leftarrow f_\theta(x_t, t, c)$; $\mathscr{M}$.update(y_t) *// run full model during warmup*
  else
    $s_t \leftarrow \mathscr{M}$.signal$(x_t, t, c)$ *// compute drift signal via $\mathscr{M}$*
    $\Sigma \leftarrow \Sigma + s_t$
    if $t \in R_{\text{crit}}$ **then**
      $\tau \leftarrow \tau_{\text{low}}$ *// critical step: cache conservatively*
    else
      $\tau \leftarrow \tau_{\text{high}}$ *// stable interior: cache aggressively*
    end if
    if $\Sigma \leq \tau$ **then**
      $y_t \leftarrow \mathscr{M}$.reuse$(x_t, t)$ *// skip full model; reconstruct via $\mathscr{M}$*
    else
      $y_t \leftarrow f_\theta(x_t, t, c)$; $\mathscr{M}$.update(y_t); $\Sigma \leftarrow 0$ *// run full model, reset*
    end if
  end if
end for

4 Experiments

4.1 Experimental Setup

Base models. We evaluate on three open-source video generation diffusion models: HunyuanVideo T2V 720P (Kong et al., 2024), Wan 2.1 T2V 1.3B (Wang et al., 2025), and CogVideoX 1.5-5B (Yang et al., 2024).

Compared methods. We apply our threshold schedule framework on top of three base caching methods: TeaCache (evaluated on HunyuanVideo, Wan 2.1, CogVideoX), EasyCache (evaluated on HunyuanVideo, Wan 2.1), and DiCache (evaluated on HunyuanVideo, Wan 2.1). EasyCache and DiCache are not evaluated on CogVideoX because neither method provides an official implementation for that model; we restrict our evaluation to officially supported configurations to avoid confounding results from self-implemented ports. For each method–model combination, we compare a no-caching baseline, the fixed-threshold method with several threshold values spanning a low-to-high range, and our scheduled-threshold variant ACID with several low/high threshold values.

Evaluation metrics. Following TeaCache, we evaluate along two dimensions: inference efficiency and visual quality. For inference efficiency, we report inference latency and speedup, defined as the ratio of the no-caching baseline latency to the method latency for the same model. For visual quality, we use LPIPS (perceptual similarity), PSNR (peak signal-to-noise ratio), and SSIM (structural similarity), computed by comparing cached-method outputs against the no-caching baseline.

Implementation details. All experiments were carried out on NVIDIA A100 80GB PCIe GPUs with PyTorch. We selected 33 prompts from VBench's (Huang et al., 2024) prompt set: 3 prompts per group across 11 groups, where each group corresponds to one or more VBench evaluation dimensions, together spanning all 16 VBench dimensions. For each method–model combination, we generated 33 videos per configuration (one per prompt) across all evaluated threshold settings, using a fixed seed throughout. Separately, for the drift signal analysis, we recorded signals over 98 T2V-CompBench (Sun et al., 2025) prompts for each of the seven method–model combinations, giving 686 signal-recording runs.

Table 1: Quantitative evaluation across caching methods and models. Per method–model, rows are: baseline (no caching), fixed low threshold, fixed high threshold, and our scheduled-threshold method (best Pareto-optimal configuration); the fixed thresholds are the low/high values used by the scheduled mode. Fidelity metrics are computed against the uncached baseline, and "—" marks the self-referential baseline. ↑/↓: higher/lower is better.

Method	Model	Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
TeaCache	HunyuanVideo	Baseline	1359s	1.00×	—	—	—
		Fixed (lo, 0.1)	864s	1.57×	24.69	0.819	0.187
		Fixed (hi, 0.3)	424s	3.20×	19.98	0.693	0.353
		Scheduled	628s	2.16×	24.41	0.813	0.196
	Wan 2.1	Baseline	212s	1.00×	—	—	—
		Fixed (lo, 0.05)	150s	1.42×	24.93	0.845	0.083
		Fixed (hi, 0.2)	74s	2.86×	17.10	0.538	0.335
		Scheduled	108s	1.96×	22.10	0.757	0.150
	CogVideoX 1.5	Baseline	1019s	1.00×	—	—	—
		Fixed (lo, 0.1)	723s	1.41×	37.32	0.958	0.029
		Fixed (hi, 0.3)	465s	2.19×	16.35	0.580	0.463
		Scheduled	545s	1.87×	32.02	0.930	0.058
EasyCache	HunyuanVideo	Baseline	1357s	1.00×	—	—	—
		Fixed (lo, 0.025)	659s	2.06×	33.51	0.926	0.058
		Fixed (hi, 0.045)	531s	2.55×	28.99	0.851	0.153
		Scheduled	568s	2.39×	30.62	0.894	0.094
	Wan 2.1	Baseline	216s	1.00×	—	—	—
		Fixed (lo, 0.020)	125s	1.73×	25.58	0.852	0.087
		Fixed (hi, 0.040)	102s	2.12×	22.02	0.750	0.148
		Scheduled	113s	1.92×	24.90	0.833	0.100
DiCache	HunyuanVideo	Baseline	1265s	1.00×	—	—	—
		Fixed (lo, 0.05)	659s	1.92×	30.96	0.904	0.088
		Fixed (hi, 0.10)	419s	3.02×	20.15	0.720	0.328
		Scheduled	620s	2.04×	29.42	0.888	0.105
	Wan 2.1	Baseline	224s	1.00×	—	—	—
		Fixed (lo, 0.07)	99s	2.27×	13.41	0.356	0.508
		Fixed (hi, 0.225)	56s	3.97×	11.79	0.282	0.644
		Scheduled	82s	2.73×	13.29	0.355	0.541

4.2 Main Results

The Pareto plots in Appendix B show the efficiency and visual quality tradeoff offered by fixed-threshold methods, and how our scheduled-threshold method expands the fixed-threshold Pareto frontier for each of the seven caching method and diffusion model combinations.

Table 1 shows the efficiency and visual quality of notable scheduled mode points for each feature caching method and diffusion model combination, comparing them to the baseline mode, the fixed-threshold mode using the low threshold of the scheduled mode, and the fixed-threshold mode using the high threshold of the scheduled mode.

TeaCache. For TeaCache and HunyuanVideo, the scheduled mode achieves a 2.16× speedup, between the low-threshold fixed mode's speedup of 1.57× and the high-threshold fixed mode's speedup of 3.20×, while offering fidelity scores very similar to those of the low-threshold mode.

For TeaCache and Wan 2.1, the scheduled mode achieves a speedup of 1.96×, between the low-threshold fixed mode's speedup of 1.42× and the high-threshold fixed mode's speedup of 2.86×, while retaining quality far closer to the low-threshold mode than to the high-threshold mode.

For TeaCache and CogVideoX, the scheduled mode achieves a speedup of 1.87×, between the low-threshold fixed mode's speedup of 1.41× and the high-threshold fixed mode's speedup of 2.19×, again retaining quality far closer to the low-threshold mode than to the high-threshold mode.

Figure 7 shows how ACID's scheduled modes surpass the Pareto frontier curves for all three diffusion models when used with TeaCache.

EasyCache. For EasyCache and HunyuanVideo, our scheduled mode achieves a speedup of 2.39×, much closer to the high-threshold fixed mode's speedup of 2.55× than to the low-threshold fixed mode's speedup of 2.06×, with quality metrics falling between those of the two fixed modes.

For EasyCache and Wan 2.1, our scheduled mode achieves a speedup of 1.92×, falling between the low- and high-threshold fixed mode speedups of 1.73× and 2.12×, while retaining quality far closer to the low-threshold mode than to the high-threshold mode.

Figure 8 shows how several of ACID's scheduled threshold modes surpass the Pareto frontier curves for both diffusion models when used with EasyCache.

We also note that EasyCache, by running the full model for the first few diffusion steps, already supports our main idea: varying the threshold across the diffusion process, in this case using a threshold of 0 (full computation) at the beginning and a higher threshold later, leads to a better quality–efficiency tradeoff.

DiCache. For DiCache and HunyuanVideo, the scheduled mode achieves a speedup of 2.04×, falling between the low- and high-threshold speedups of 1.92× and 3.02×, while achieving visual quality very close to the low-threshold mode and much better than the high-threshold mode.

For DiCache and Wan 2.1, the scheduled mode achieves a speedup of 2.73×, falling between the low- and high-threshold speedups of 2.27× and 3.97×, while achieving visual quality very close to the low-threshold mode and much better than the high-threshold mode.

Figure 9 in Appendix B shows how several of ACID's scheduled modes surpass the Pareto frontier curve for both diffusion models when used with DiCache.

5 Conclusion

We showed two findings. First, the fixed caching threshold used by prior dynamic caching methods for video diffusion is a source of structural suboptimality. Second, the drift signal is a function of the model and the feature caching method rather than the prompt, so the schedule for varying the threshold can be computed once offline. We exploit both findings with ACID, a minimal threshold schedule that caches conservatively at critical steps and aggressively across the stable interior. Experiments across three open-source video generation models and three caching methods confirm that ACID achieves near-low-threshold visual quality with higher inference speedup.

AI Use Statement

We used large language models to aid and polish the writing of this paper. All research contributions, including the experimental design, implementation, and analysis, are our own.

References

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

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

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators. *OpenAI Blog*, 1:8, 2024.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. *arXiv preprint arXiv:2508.17356*, 2025.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv preprint arXiv:2310.00426*, 2023.

Haoxin Chen, Yong Zhang, Xiaodong Cun, Menghan Xia, Xintao Wang, Chao Weng, and Ying Shan. Videocrafter2: Overcoming data limitations for high-quality video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7310–7320, 2024.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Δ-dit: A training-free acceleration method tailored for diffusion transformers. *arXiv preprint arXiv:2406.01125*, 2024.

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

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

Yuwei Guo, Ceyuan Yang, Anyi Rao, Zhengyang Liang, Yaohui Wang, Yu Qiao, Maneesh Agrawala, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning. *arXiv preprint arXiv:2307.04725*, 2023.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Wenyi Hong, Ming Ding, Wendi Zheng, Xinghao Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via transformers. In *Proceedings of the International Conference on Learning Representations*, 2023.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21807–21818, 2024.

Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S. Ryoo, and Tian Xie. Adaptive caching for faster video generation with diffusion transformers. *arXiv preprint arXiv:2411.02397*, 2024.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *CoRR*, 2023.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024.

Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36, 2024.

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 Computer Vision and Pattern Recognition Conference*, pp. 7353–7363, 2025.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. *arXiv preprint arXiv:2503.06923*, 2025.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. Fastercache: Training-free video diffusion model acceleration with high quality. *arXiv preprint arXiv:2410.19355*, 2024.

Xin Ma, Yaohui Wang, Gengyun Jia, Xinyuan Chen, Ziwei Liu, Yuan-Fang Li, Cunjian Chen, and Yu Qiao. Latte: Latent diffusion transformer for video generation. *Transactions on Machine Learning Research*, 2024.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. *Advances in Neural Information Processing Systems*, 37:133282–133304, 2024.

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.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14297–14306, 2023.

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.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Sucheng Ren, Qihang Yu, Ju He, Alan Yuille, and Liang-Chieh Chen. Grouping first, attending smartly: Training-free acceleration for diffusion transformers. *arXiv preprint arXiv:2505.14687*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, pp. 87–103. Springer, 2024.

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*, 2024.

Kaiyue Sun, Kaiyi Huang, Xian Liu, Yue Wu, Zihan Xu, Zhenguo Li, and Xihui Liu. T2v-compbench: A comprehensive benchmark for compositional text-to-video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8406–8416, 2025.

Xiang Wang, Shiwei Zhang, Han Zhang, Yu Liu, Yingya Zhang, Changxin Gao, and Nong Sang. Videolcm: Video latent consistency model. *arXiv preprint arXiv:2312.09109*, 2023.

Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. *arXiv preprint arXiv:2502.18137*, 2025.

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. *arXiv preprint arXiv:2408.12588*, 2024.

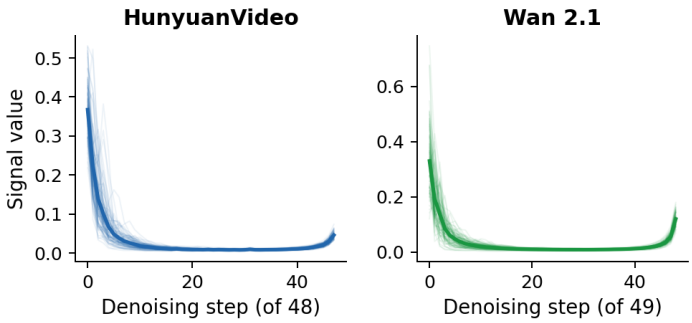
Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

Dian Zheng, Ziqi Huang, Hongbo Liu, Kai Zou, Yinan He, Fan Zhang, Lulu Gu, Yuanhan Zhang, Jingwen He, Wei-Shi Zheng, Yu Qiao, and Ziwei Liu. Vbench-2.0: Advancing video generation benchmark suite for intrinsic faithfulness. *arXiv preprint arXiv:2503.21755*, 2025.

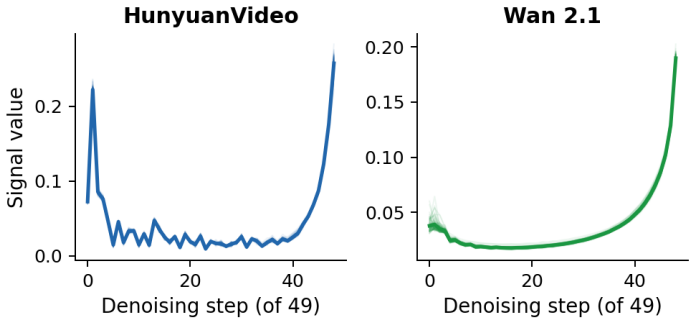
Xin Zhou, Dingkang Liang, Kaijin Chen, Tianrui Feng, Xiwu Chen, Hongkai Lin, Yikang Ding, Feiyang Tan, Hengshuang Zhao, and Xiang Bai. Less is enough: Training-free video diffusion acceleration via runtime-adaptive caching. *arXiv preprint arXiv:2507.02860*, 2025.

A Drift Signal Analysis

Figures 3–6 support the analysis in Section 3.1.



(a) EasyCache



(b) DiCache

Figure 3: Drift signal for EasyCache and DiCache across 98 T2V-CompBench prompts, plotted as in Figure 2. The shape differs across methods, showing it is characteristic of the caching method as well as the model.

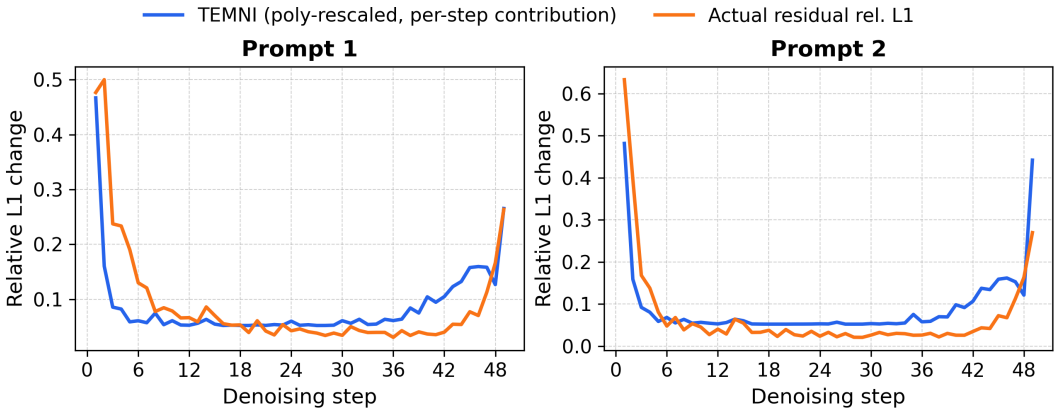


Figure 4: TeaCache's ΔTEMNI proxy compared against the actual relative $L1$ change in model output on HunyuanVideo, for two prompts. **Prompt 1:** "Two anthropomorphic cats in comfy boxing gear and bright gloves fight intensely on a spotlighted stage." **Prompt 2:** "Static black and white shot of a single bare tree against an overcast sky, no movement, high contrast, minimalist composition."

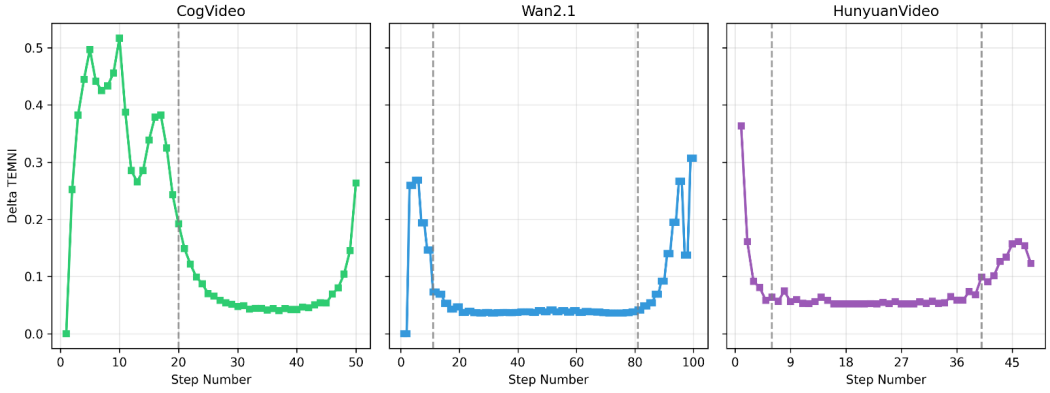


Figure 5: ΔTEMNI signal across all three models throughout the denoising process. Vertical dashed lines mark the transition points identified in the calibration study.

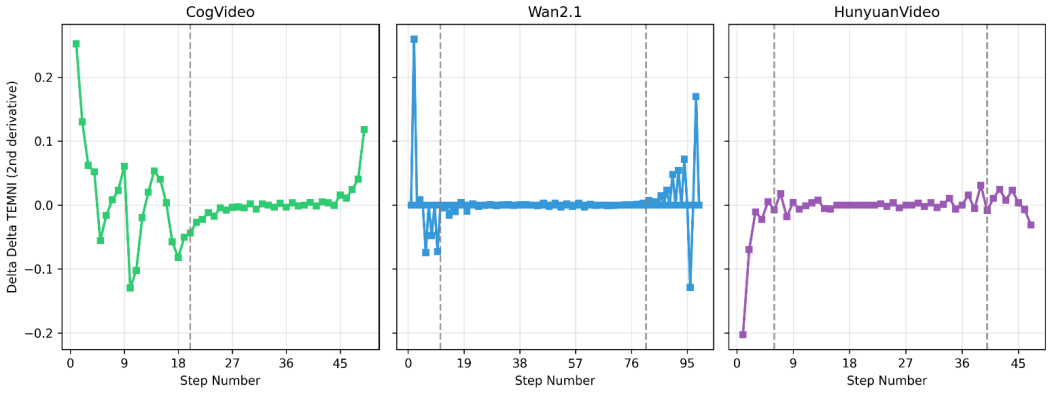
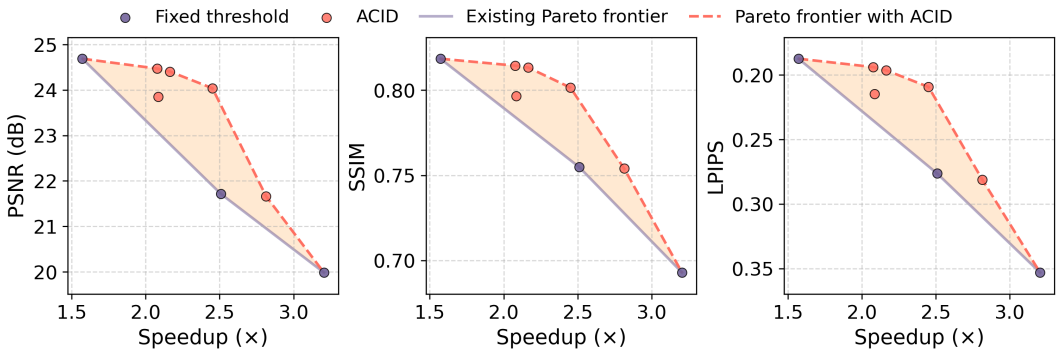


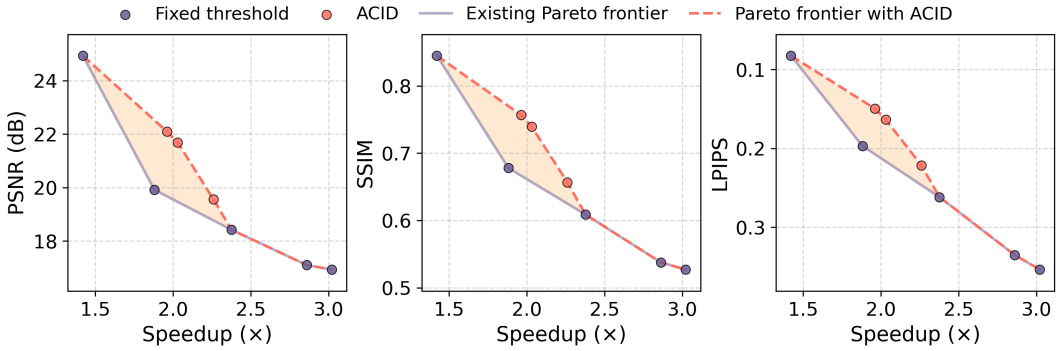
Figure 6: Second derivative of the ΔTEMNI signal across all three models, with the same transition markers. The transition points align with where the second derivative stabilizes, motivating its use as the criterion for placing threshold transitions.

B PARETO FRONTIER PLOTS

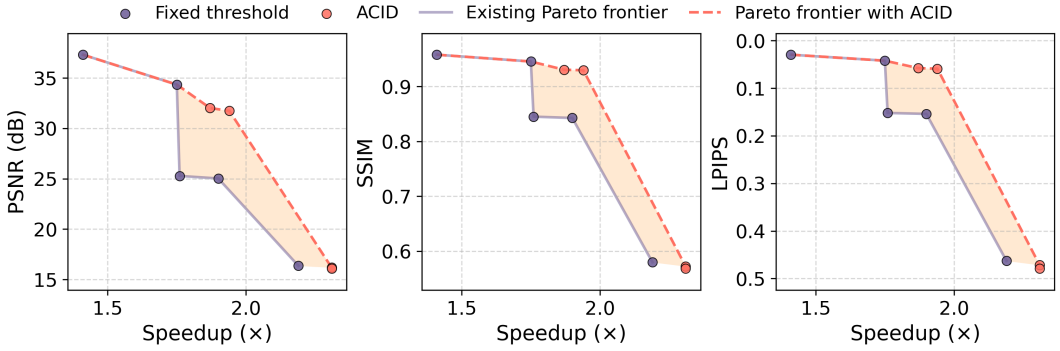
Figures 7–9 show Pareto frontier plots for all evaluated method–model combinations. Each plot compares fixed-threshold configurations across a range of thresholds against our scheduled-threshold variants. In each case, scheduled points expand the frontier beyond what any fixed threshold achieves.



(a) HunyuanVideo

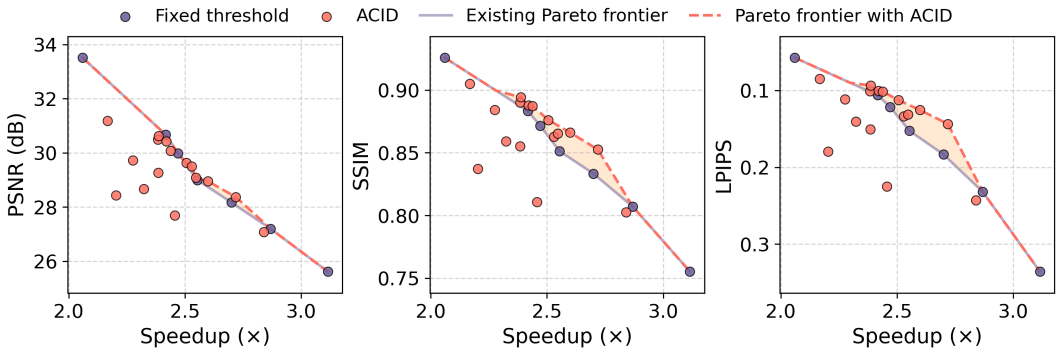


(b) Wan 2.1

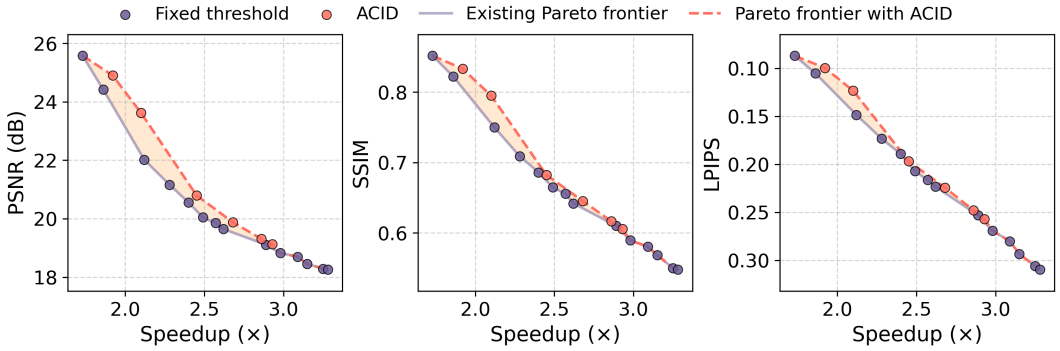


(c) CogVideoX 1.5

Figure 7: TeaCache Pareto frontiers across all three models. ACID (red markers) expands the frontier beyond any fixed-threshold configuration (purple markers).

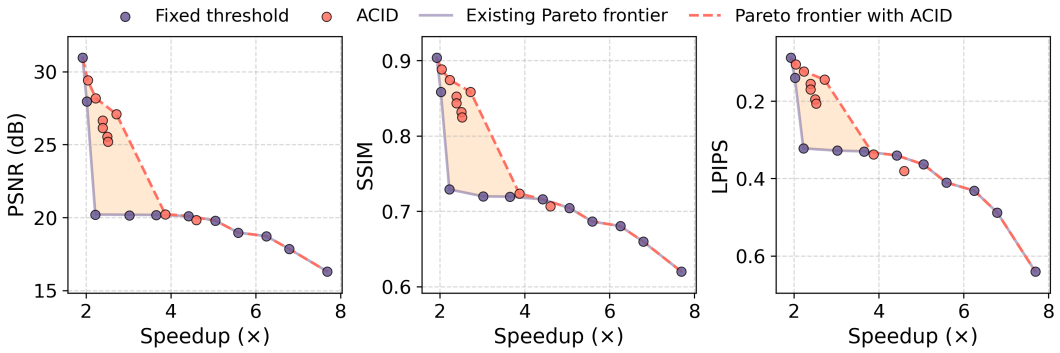


(a) HunyuanVideo

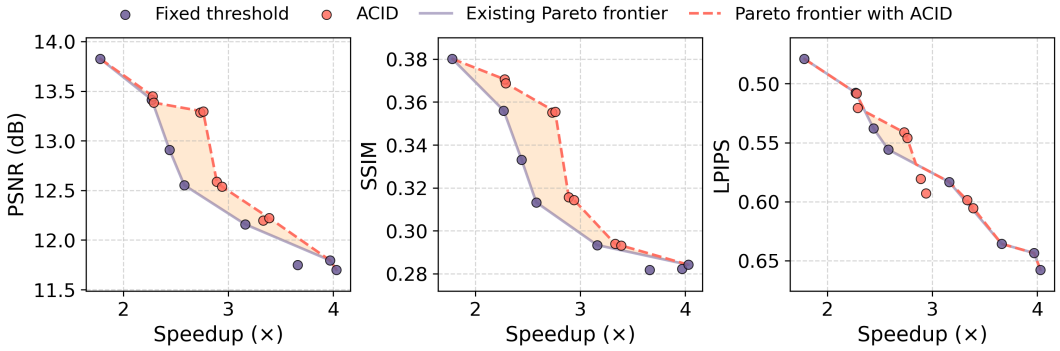


(b) Wan 2.1

Figure 8: EasyCache Pareto frontiers. ACID expands the frontier beyond any fixed-threshold configuration.



(a) HunyuanVideo



(b) Wan 2.1

Figure 9: DiCache Pareto frontiers. ACID expands the frontier beyond any fixed-threshold configuration.

C Detailed Per-Combo Results

All configurations evaluated for each method × model combination are listed below, sorted by increasing speedup. **Bold** rows are scheduled modes. Fidelity metrics (PSNR, SSIM, LPIPS) are computed against the uncached baseline. τ denotes the caching threshold for all methods (cache-reuse residual for TeaCache/EasyCache; feature-difference score for DiCache). Subscripts ℓ/h mark the low/high threshold in scheduled modes. Scheduled rows that share the same threshold pair differ in where the threshold transitions are placed.

Table 2: TeaCache on HunyuanVideo: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.1$)	864s	1.57×	24.69	0.819	0.187
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.25$)	654s	2.08×	24.47	0.814	0.194
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.2$)	652s	2.09×	23.85	0.796	0.215
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.3$)	628s	2.16×	24.41	0.813	0.196
Scheduled ($\tau_\ell = 0.15$, $\tau_h = 0.3$)	554s	2.45×	24.04	0.802	0.209
Fixed ($\tau = 0.2$)	542s	2.51×	21.71	0.755	0.276
Scheduled ($\tau_\ell = 0.2$, $\tau_h = 0.3$)	483s	2.81×	21.66	0.754	0.281
Fixed ($\tau = 0.3$)	424s	3.20×	19.98	0.693	0.353

Table 3: TeaCache on Wan 2.1: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.05$)	150s	1.42×	24.93	0.845	0.083
Fixed ($\tau = 0.1$)	113s	1.88×	19.92	0.678	0.197
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.2$)	108s	1.96×	22.10	0.757	0.150
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.25$)	105s	2.03×	21.69	0.739	0.164
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.2$)	94s	2.26×	19.56	0.657	0.222
Fixed ($\tau = 0.15$)	89s	2.38×	18.42	0.609	0.262
Fixed ($\tau = 0.2$)	74s	2.86×	17.10	0.538	0.335
Fixed ($\tau = 0.25$)	70s	3.02×	16.94	0.527	0.354

Table 4: TeaCache on CogVideoX 1.5: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.1$)	723s	1.41×	37.32	0.958	0.029
Fixed ($\tau = 0.2$)	582s	1.75×	34.35	0.945	0.042
Fixed ($\tau = 0.22$)	579s	1.76×	25.28	0.845	0.152
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.3$)	545s	1.87×	32.02	0.930	0.058
Fixed ($\tau = 0.25$)	536s	1.90×	25.03	0.843	0.154
Scheduled ($\tau_\ell = 0.1$, $\tau_h = 0.3$)	525s	1.94×	31.74	0.929	0.059
Fixed ($\tau = 0.3$)	465s	2.19×	16.35	0.580	0.463
Scheduled ($\tau_\ell = 0.30$, $\tau_h = 0.35$)	441s	2.31×	16.18	0.572	0.472
Scheduled ($\tau_\ell = 0.30$, $\tau_h = 0.40$)	441s	2.31×	16.08	0.568	0.479

Table 5: EasyCache on HunyuanVideo: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.025$)	659s	2.06×	33.51	0.926	0.058
Scheduled ($\tau_\ell = 0.010$, $\tau_h = 0.050$)	626s	2.17×	31.18	0.905	0.085
Scheduled ($\tau_\ell = 0.010$, $\tau_h = 0.120$)	616s	2.20×	28.43	0.837	0.179
Scheduled ($\tau_\ell = 0.010$, $\tau_h = 0.075$)	596s	2.28×	29.72	0.884	0.112
Scheduled ($\tau_\ell = 0.010$, $\tau_h = 0.100$)	584s	2.32×	28.67	0.859	0.141
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.075$)	569s	2.38×	30.49	0.890	0.101
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.090$)	569s	2.38×	29.27	0.855	0.151
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.045$)	568s	2.39×	30.62	0.894	0.094
Fixed ($\tau = 0.0375$)	561s	2.42×	30.67	0.883	0.106
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.050$)	561s	2.42×	30.43	0.888	0.101
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.050$)	557s	2.44×	30.08	0.887	0.102
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.150$)	552s	2.46×	27.69	0.811	0.225
Fixed ($\tau = 0.040$)	549s	2.47×	29.98	0.872	0.122
Scheduled ($\tau_\ell = 0.030$, $\tau_h = 0.050$)	542s	2.51×	29.63	0.876	0.113
Scheduled ($\tau_\ell = 0.0375$, $\tau_h = 0.050$)	537s	2.53×	29.50	0.863	0.134
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.075$)	533s	2.55×	29.08	0.865	0.132
Fixed ($\tau = 0.045$)	531s	2.55×	28.99	0.851	0.153
Scheduled ($\tau_\ell = 0.030$, $\tau_h = 0.060$)	522s	2.60×	28.95	0.866	0.126
Fixed ($\tau = 0.050$)	503s	2.70×	28.17	0.833	0.183
Scheduled ($\tau_\ell = 0.035$, $\tau_h = 0.065$)	499s	2.72×	28.37	0.853	0.144
Scheduled ($\tau_\ell = 0.050$, $\tau_h = 0.120$)	478s	2.84×	27.08	0.803	0.243
Fixed ($\tau = 0.060$)	473s	2.87×	27.20	0.807	0.232
Fixed ($\tau = 0.075$)	436s	3.12×	25.62	0.755	0.336

Table 6: EasyCache on Wan 2.1: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.020$)	125s	1.73×	25.58	0.852	0.087
Fixed ($\tau = 0.025$)	116s	1.86×	24.42	0.822	0.105
Scheduled ($\tau_\ell = 0.020$, $\tau_h = 0.040$)	112s	1.92×	24.90	0.833	0.100
Scheduled ($\tau_\ell = 0.025$, $\tau_h = 0.050$)	103s	2.10×	23.62	0.795	0.123
Fixed ($\tau = 0.040$)	102s	2.12×	22.02	0.750	0.148
Fixed ($\tau = 0.050$)	95s	2.28×	21.16	0.709	0.173
Fixed ($\tau = 0.060$)	90s	2.40×	20.55	0.686	0.189
Scheduled ($\tau_\ell = 0.020$, $\tau_h = 0.025$)	88s	2.45×	20.80	0.682	0.197
Fixed ($\tau = 0.070$)	87s	2.49×	20.04	0.665	0.207
Fixed ($\tau = 0.080$)	84s	2.57×	19.85	0.656	0.216
Fixed ($\tau = 0.090$)	82s	2.62×	19.64	0.642	0.223
Scheduled ($\tau_\ell = 0.020$, $\tau_h = 0.050$)	81s	2.68×	19.88	0.645	0.224
Scheduled ($\tau_\ell = 0.020$, $\tau_h = 0.075$)	76s	2.86×	19.31	0.617	0.248
Fixed ($\tau = 0.125$)	75s	2.89×	19.11	0.610	0.253
Scheduled ($\tau_\ell = 0.020$, $\tau_h = 0.1$)	74s	2.93×	19.13	0.605	0.257
Fixed ($\tau = 0.150$)	72s	2.98×	18.83	0.589	0.269
Fixed ($\tau = 0.175$)	70s	3.09×	18.70	0.581	0.280
Fixed ($\tau = 0.200$)	69s	3.15×	18.45	0.568	0.293
Fixed ($\tau = 0.225$)	66s	3.25×	18.29	0.550	0.306
Fixed ($\tau = 0.250$)	66s	3.28×	18.26	0.548	0.310

Table 7: DiCache on HunyuanVideo: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.05$)	659s	1.92×	30.96	0.904	0.088
Fixed ($\tau = 0.07$)	626s	2.02×	27.95	0.858	0.140
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.10$)	620s	2.04×	29.42	0.888	0.105
Fixed ($\tau = 0.08$)	570s	2.22×	20.21	0.729	0.322
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.15$)	567s	2.23×	28.19	0.874	0.123
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.25$)	529s	2.39×	26.66	0.852	0.155
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.30$)	529s	2.39×	26.15	0.844	0.169
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.35$)	506s	2.50×	25.55	0.832	0.195
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.40$)	502s	2.52×	25.20	0.824	0.206
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.20$)	465s	2.72×	27.08	0.859	0.144
Fixed ($\tau = 0.10$)	419s	3.02×	20.15	0.720	0.328
Fixed ($\tau = 0.15$)	347s	3.65×	20.18	0.719	0.330
Scheduled ($\tau_\ell = 0.10$, $\tau_h = 0.30$)	327s	3.87×	20.23	0.724	0.338
Fixed ($\tau = 0.20$)	286s	4.42×	20.11	0.716	0.340
Scheduled ($\tau_\ell = 0.15$, $\tau_h = 0.40$)	275s	4.60×	19.83	0.707	0.380
Fixed ($\tau = 0.25$)	250s	5.05×	19.79	0.704	0.364
Fixed ($\tau = 0.30$)	226s	5.59×	18.97	0.686	0.410
Fixed ($\tau = 0.35$)	202s	6.25×	18.73	0.680	0.431
Fixed ($\tau = 0.40$)	186s	6.79×	17.85	0.660	0.488
Fixed ($\tau = 0.60$)	164s	7.69×	16.30	0.620	0.641

Table 8: DiCache on Wan 2.1: all evaluated configurations

Mode	Latency	Speedup	PSNR ↑	SSIM ↑	LPIPS ↓
Fixed ($\tau = 0.05$)	126s	1.78×	13.83	0.380	0.479
Fixed ($\tau = 0.07$)	99s	2.27×	13.41	0.356	0.508
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.225$)	98s	2.28×	13.45	0.371	0.508
Scheduled ($\tau_\ell = 0.05$, $\tau_h = 0.25$)	98s	2.29×	13.38	0.369	0.520
Fixed ($\tau = 0.08$)	92s	2.44×	12.91	0.333	0.538
Fixed ($\tau = 0.10$)	87s	2.58×	12.55	0.313	0.556
Scheduled ($\tau_\ell = 0.07$, $\tau_h = 0.225$)	82s	2.73×	13.29	0.355	0.541
Scheduled ($\tau_\ell = 0.07$, $\tau_h = 0.25$)	81s	2.76×	13.30	0.356	0.546
Scheduled ($\tau_\ell = 0.10$, $\tau_h = 0.225$)	78s	2.89×	12.59	0.316	0.581
Scheduled ($\tau_\ell = 0.10$, $\tau_h = 0.25$)	76s	2.94×	12.54	0.314	0.593
Fixed ($\tau = 0.15$)	71s	3.16×	12.16	0.293	0.583
Scheduled ($\tau_\ell = 0.15$, $\tau_h = 0.225$)	67s	3.33×	12.20	0.294	0.599
Scheduled ($\tau_\ell = 0.15$, $\tau_h = 0.25$)	66s	3.39×	12.22	0.293	0.605
Fixed ($\tau = 0.20$)	61s	3.66×	11.75	0.282	0.636
Fixed ($\tau = 0.225$)	56s	3.97×	11.79	0.282	0.644
Fixed ($\tau = 0.25$)	56s	4.03×	11.70	0.284	0.658