A Survey of Feature Caching for Diffusion Model Acceleration

Tong Zhao

Abstract

Diffusion models produce high-quality images and videos, but their iterative denoising makes inference slow, especially for large Diffusion Transformers (DiTs) used in high-resolution video. *Feature caching* reduces this cost by reusing intermediate computations across nearby denoising steps, often without retraining the generator. This review examines fifteen representative papers published between 2023 and 2026. Their methods are compared through three practical design choices: what is cached, when it is recomputed, and how a cached value is reused. The literature has moved from fixed, U-Net-specific schedules toward adaptive and learned policies, while recent work treats reuse as prediction rather than simple copying. Sensitivity analysis and Taylor approximation provide useful connections among these methods, although the empirical evidence remains difficult to compare because baselines, metrics, calibration procedures, hardware, and memory reporting vary substantially across papers.

1 Introduction

Denoising diffusion models [16, 17, 18] are widely used for high-fidelity visual generation. Their quality, however, comes from a reverse process that evaluates a large neural network tens to hundreds of times in sequence, and this iterative structure inhibits parallelisation and dominates inference cost [1, 4]. The cost has increased further with the move from convolutional U-Net backbones to isotropic Diffusion Transformers (DiTs) [19] and to long, high-resolution video: generating a few seconds of video with a modern DiT can take minutes on a high-end GPU [15, 7].

Inference cost is largely determined by the number of network evaluations (NFEs). Each NFE is a full forward pass, the denoising steps are sequential, and the cost of a pass grows with model size, resolution and, for video, clip length. For large image and video DiTs, one sample can take seconds or minutes depending on the hardware and generation settings [15, 7]. Avoiding redundant evaluations can therefore reduce latency directly.

Existing acceleration methods either shorten the sampling trajectory, reduce the cost of each network pass, or avoid repeating computations that have changed little. Better ODE/SDE solvers and step distillation follow the first route [17]; pruning, quantisation and architectural redesign follow the second, often with additional training or calibration. This review concerns the third route, *feature caching*. Caching is usually applied only at inference and can often be combined with the other approaches. These advantages have produced many closely related schemes in a short period, but their claims are hard to compare because they use different terminology, models and evaluation settings.

DeepCache [1] begins from a simple observation: high-level features in a denoising U-Net change slowly between adjacent steps and can therefore be reused. Later work extends this idea to DiTs, videos, individual tokens and learned schedules. The reported speedups now range from roughly $2\times$ to more than $5\times$ depending on the model and setting, large enough to matter in deployment but variable enough that careful comparison is needed. Despite differences in notation and implementation, the reviewed methods can be compared through three questions:

- **What to cache?** The *granularity* of reuse: whole-model U-Net features, individual transformer layers or blocks, residual "deltas," or individual tokens.
- **When to cache?** The *schedule* that decides, for a given step, whether to recompute or reuse: static intervals, adaptive error-driven triggers, globally optimised paths, or learned, budget-aware policies.
- **How to reuse?** The *reuse rule* that reconstructs the skipped features: simple copying (a zeroth-order approximation), first-order trajectory alignment, or higher-order forecasting.

Table 1 uses these questions to compare the fifteen methods and the baselines that connect them. The later sections examine the sensitivity- and Taylor-based interpretations that relate several of the proposed indicators and reuse rules. They also consider weaknesses in the available evidence and the experiments needed to address them.

2 Background and scope

A diffusion model defines a forward process that gradually corrupts data x_0 into noise and a learned reverse process that denoises it back. At inference, a network $\epsilon_\theta(x_t, t, c)$ (with timestep t and optional condition

c) is applied iteratively over a schedule of timesteps; each application is one network evaluation, or NFE. Because adjacent timesteps differ only slightly, the outputs, and the intermediate features that produce them, are strongly correlated across steps. Every caching method exploits this temporal redundancy.

This redundancy is large in practice. DeepCache reported that the high-level features of a denoising U-Net are highly similar between adjacent steps, and the adaptive methods rest on the same property: across most of the trajectory the per-step output changes only slightly, while the larger changes are concentrated in particular phases of sampling that differ by model. Because a standard sampler evaluates the full network at every step, skipping even a fraction of these evaluations removes a large share of the total cost. The practical difficulty, taken up in Section 3, is deciding which steps can be skipped, and how a skipped step should be reconstructed, without visibly changing the result.

Formally, the forward process adds noise as $x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, z_t$ with $z_t \sim \mathcal{N}(0, I)$, and the reverse process draws $x_{t-1} \sim p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\big(\mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\big)$, with the mean produced by the network ϵ_θ [16, 17]. Caching exploits that the per-step outputs O_t change little between neighbours, a change most papers quantify with the relative ℓ_1 distance

$$\mathrm{L1rel}(O, t) = \frac{\|O_t - O_{t+1}\|_1}{\|O_{t+1}\|_1}. \tag{1}$$

Since O_t is unavailable before the step is computed, adaptive methods evaluate Eq. (1) on a *cheap proxy* F (a timestep embedding, a modulated input, a residual magnitude, or a shallow-probe feature) and reuse the cached output over a span $[t_a, t_b)$ while the accumulated, optionally rescaled, indicator stays below a threshold δ:

$$\sum_{t=t_a}^{t_b-1} f\big(\mathrm{L1rel}(F, t)\big) \;\le\; \delta \;<\; \sum_{t=t_a}^{t_b} f\big(\mathrm{L1rel}(F, t)\big), \tag{2}$$

recomputing and resetting the accumulator once the bound is crossed. Equations (1)–(2) form the shared backbone of the adaptive schedulers of Section 3.2, which differ almost entirely in the proxy F and the rescaling f.

Architecture strongly affects what can be cached. A U-Net has an encoder–decoder shape with skip connections, so it naturally separates "high-level" features (the deep, expensive main branch) from "low-level" features (the cheap skip branch); DeepCache exploits this split. A DiT, by contrast, is an isotropic stack of transformer blocks with no comparable high/low-level decomposition, so the U-Net recipe does not transfer directly [3, 2]. Much of the work since 2024 asks what should replace that recipe for transformer backbones.

A second axis of variation is the generative formulation. Classical models predict the added noise ϵ_θ, whereas the recent flow-matching / rectified-flow models (FLUX, Wan, HunyuanVideo) predict a velocity field along a straight-line interpolation between data and noise. This distinction matters for caching because several of the strongest indicators are defined on the model *residual* $r_t = v_\theta(x_t, t) - x_t$, the per-step "update direction", whose magnitude and direction turn out to evolve far more regularly than the raw features. MagCache's magnitude law and DiCache's trajectory alignment both live in this residual space, which is one reason they generalise cleanly across the modern video and image backbones.

Scope. We review fifteen papers (Table 1) that form a tight, citation-connected cluster on training-free or learned feature caching for diffusion inference and together span the field's full chronology, from the originating U-Net method of 2023 to the transformer- and video-oriented work of 2026. We deliberately exclude orthogonal accelerators (fast solvers, step distillation, quantisation and sparse attention) except where a reviewed method composes with them. Each work is cited at its venue of record; the four that remain unpublished at the time of writing (Δ-DiT, DuCa, PromptTea and ReCache) are cited as preprints.

3 The anatomy of a caching method

Every method in Table 1 can be specified by a choice on each of three largely independent axes: *what* unit of computation is cached (granularity), *when* a step recomputes versus reuses (the schedule), and *how* a skipped step's features are reconstructed (the reuse rule). Two methods may share a granularity but use different schedules, or share a schedule but reconstruct features differently. Treating the choices separately therefore provides a useful basis for comparison. The following subsections discuss each axis before Sections 4 and 5 consider their interaction and evaluation.

Table 1: Taxonomy of the reviewed methods along the three axes: *what* is cached, *when*, and *how* it is reused (img = image, vid = video, I+V = both; "Full step" denotes the whole denoiser output or residual). Reported speedups are each paper's own best case and are not directly comparable across rows (Sec. 5).

Method (venue)	**Model**	**What**	**When (schedule)**	**How**	**Speedup**
DeepCache [1] (CVPR'24)	U-Net, img	Features	Static 1:N	Copy	2.3–7×
L2C [2] (NeurIPS'24)	DiT, img	Layers	Learned router	Copy	≤2×
Δ-DiT [3] (arXiv'24)	DiT, img	Deltas	Stage-adaptive	Δ-add	∼1.6×
TeaCache [4] (CVPR'25)	DiT, vid	Full step	Timestep emb.	Copy	≤4.4×
HarmoniCa [6] (ICML'25)	DiT, img	Blocks	Learned (SDT+IEPO)	Copy	∼1.6×
TaylorSeer [5] (ICCV'25)	DiT, I+V	Modules	Static	Forecast	∼5×
MagCache [7] (NeurIPS'25)	DiT, vid	Full step	Magnitude law	Copy	∼2.5×
LeMiCa [8] (NeurIPS'25)	DiT, vid	Full step	Global DAG	Copy	≤2.9×
ClusCa [9] (ACM MM'25)	DiT, I+V	Tokens	Static	Spatio-temp.	5–6×
DuCa [10] (arXiv'25)	DiT, I+V	Tokens	Dual caching	Copy	∼3.5×
DiCache [11] (ICLR'26)	DiT, I+V	Full step	Online probe	1st-order	∼3×
PromptTea [12] (arXiv'25)	DiT, vid	Full step	Prompt-aware	Copy	∼2.8×
ReCache [13] (arXiv'26)	DiT, I+V	Any	Learned budget	Any	—
SeaCache [14] (CVPR'26)	DiT, I+V	Full step	Spectral	Copy	—
SenCache [15] (CVPR'26)	DiT, I+V	Full step	Sensitivity	Copy	∼1.7×

3.1 What and where to cache: the granularity axis

The earliest and coarsest answer is DeepCache [1], which caches the U-Net's high-level main-branch feature at a full-inference step and, for the next $N-1$ steps, recomputes only the shallow skip branch while retrieving the cached high-level feature. Its non-uniform 1:N schedule places more full computations where adjacent-step similarity dips (around 40% into LDM sampling), rather than spacing them evenly. DeepCache reports a 2.3× speedup on Stable Diffusion with a 0.05 CLIP-score drop and up to 7× on LDM-class models. An ablation that replaces the cached feature with zeros collapses quality, confirming that the reused feature carries essential information. DeepCache also matches or outperforms the retrained Diff-Pruning and BK-SDM baselines in its experiments. This result helped establish caching as a competitive inference-time alternative, although the method depends on the U-Net's particular skip geometry and does not transfer directly to a DiT.

DeepCache benefits from the cost imbalance between the two U-Net branches. When the cached high-level main branch accounts for most of the computation, reusing it for $N-1$ of every N steps can substantially reduce the dominant cost, although the end-to-end gain remains below the ideal $N\times$ because the shallow branch is still evaluated. Transformer backbones do not offer the same explicit split: their blocks have more similar roles and costs. DiT methods therefore have to identify cacheable layers, residuals or tokens rather than inherit a caching boundary from the architecture.

Three responses target the DiT. *Layer caching* (Learning-to-Cache, L2C [2]) observes that DiT layers have no high/low-level split and that naively dropping even a few layers is catastrophic; instead it casts a cached step as an interpolation between a cheap evaluation (reuse the previous step's layer output) and an exact one (recompute), and learns a continuous per-layer, per-timestep gate under a sparsity penalty that is discretised at inference into a static, input-invariant computation graph, all while leaving the base weights frozen. Its learned routers indicate that cacheability depends on the backbone. Up to 93.68% of a U-ViT's layers can be skipped in a cache step for under 0.01 FID, against only ≈ 47% for a plain DiT, because U-ViT inherits the U-Net's middle-redundant structure that DiT lacks. L2C is not training-free and, by interleaving one full with one cached step, is capped near 2×. *Delta caching* (Δ-DiT [3]) addresses a different failure: caching a block's raw output can discard the freshly sampled latent x_{t-1}, which causes severe degradation in the few-step regime. Δ-DiT instead caches the input-to-output *offset* (Δ) of a span of blocks and adds it to the current step's real input, preserving x_{t-1} in a skip-free network. Probing block roles, it finds that front blocks govern the outline (lower high-frequency error) and rear blocks the detail, and exploits diffusion's coarse-to-fine schedule with a stage boundary b that caches rear blocks early and front blocks late, cutting FID from 39.0 to 35.9 on PixArt-α and 15.9 to 13.3 on DiT-XL at ≈ 1.6×.

Token caching pushes granularity below the layer, recomputing only a subset of the spatial tokens each step. ToCa [22] scores token "importance" (from attention, position, and how often a token has been cached) and recomputes the important ones. DuCa [10] rethinks this on two fronts. Computing selected tokens at every

cache step may be unnecessary before error has accumulated, so it interleaves cheap "aggressive" steps (skip almost all layers) with corrective "conservative" token-wise steps. In its experiments, importance-based selection is no better than, and sometimes worse than, random selection. The authors attribute this to token diversity; random selection also restores FlashAttention compatibility, which ToCa's attention-map scoring breaks by requiring $O(N^2)$ memory. DuCa reports 3.45× speedup on FLUX with little quality degradation. ClusCa [9] exploits spatial structure differently: tokens cluster tightly (intra-cluster distance $\sim$100× smaller than the global average) and cluster membership is temporally stable (Adjusted Rand Index > 0.8 across adjacent steps), so it runs k-means once per cache cycle, computes a single representative per cluster (≈ 16 tokens), and propagates each representative to its cluster with a blend of the fresh value and the cached history. This cuts the token count by over 90% and reaches $\approx 5\times$ on FLUX and $\approx 6\times$ on HunyuanVideo.

Granularity also determines what must be retained in memory, a cost that becomes important for video. Full-step methods generally retain the most recent denoiser output or residual required by their reuse rule; layer- and block-level methods retain activations of the reused sub-modules; token-level methods retain token features; and higher-order forecasting stores several historical features or finite differences. Long, high-resolution videos make these tensors large. This helps explain why some video-oriented methods, such as DiCache, choose a low prediction order, although the appropriate reuse span still depends on the model and target speedup.

3.2 When to cache: the scheduling axis

Scheduling is the most active of the three axes, and its development is the easiest to follow.

Static schedules. The simplest policies recompute on a fixed timetable. DeepCache's uniform 1:N and FORA [20] reuse whole-block outputs at a constant interval; PAB [21] observes that attention differences follow a "U"-shaped redundancy over timesteps and broadcasts each attention output to a fixed window of later steps, with different broadcast ranges for spatial, temporal and cross-attention. These schemes are simple, require few hyperparameters and serve as common baselines. Their limitation is that output change varies across timesteps and samples, while the schedule does not.

Adaptive, error-driven schedules. Many video-DiT methods compute a cheap per-step *indicator* of output change and recompute when its accumulated value crosses a threshold (Eq. (2)). TeaCache [4] is a widely used example. It reports that per-step output differences are non-uniform and model-specific: Open-Sora has a "U"-shaped curve, whereas Latte and Open-Sora-Plan have an "L" shape. TeaCache estimates this variation from the timestep-embedding-modulated input and rescales it with a calibrated polynomial f. It reaches up to 4.4× on Open-Sora-Plan with little reported quality loss and is a common baseline in later work. Those later methods mainly disagree about the indicator. MagCache [7] argues that TeaCache's polynomial can overfit its calibration prompts and replaces it with a "unified magnitude law": the ratio of successive residual norms $\gamma_t = \mathrm{mean}\big(\|r_t\|_2/\|r_{t-1}\|_2\big)$ (with residual $r_t = \epsilon_\theta(x_t, t) - x_t$) decreases smoothly across the tested prompts and models. The paper reports that one calibration sample is sufficient and that the accumulated skip error $\approx 1 - \prod_i \gamma_i$ matches ground truth to within 10^{-5}. DiCache [11] instead estimates change online for each sample. A shallow-layer probe (as little as one of FLUX's 57 layers) predicts deep-feature evolution with Spearman correlation ≈ 0.8; when recomputation is needed, it continues from the probed layer. SeaCache [14] applies a timestep-conditioned, Wiener-style spectral filter before TeaCache's accumulation rule, separating slowly changing content from faster noise. Its reported FFT cost is below 1% of latency, and FLUX fidelity rises from roughly 21 to 26 dB PSNR at a matched refresh ratio. SenCache [15] relates several of these indicators through the denoiser's local Jacobian sensitivity, $S_t = \|J_x\|\,\|\Delta x_t\| + \|J_t\|\,|\Delta t|$. Under this decomposition, TeaCache mainly reflects the timestep term and MagCache the latent-change term, which offers one explanation for their different behaviour across regimes.

The threshold sets the operating point. In these adaptive schemes the threshold δ controls the speed–quality trade-off: a smaller δ usually triggers recomputation more often, giving higher fidelity at a lower speedup, while a larger δ does the opposite. Full speed–quality curves are therefore more informative than isolated operating points. Comparisons between curves, however, are meaningful only when the model, sampler, hardware and evaluation protocol are matched. Without those controls, different thresholds are only one of several reasons that the same method may be reported at different speedups.

Global and learned schedules. A different critique is that a locally greedy threshold ignores how errors propagate: a small early-step error may affect many later steps, whereas a larger late-step error has less time to spread. LeMiCa [8] formulates scheduling as global optimisation. It builds a directed acyclic graph whose edges are candidate cache segments, weights each edge by its estimated effect on the final sample,

and uses dynamic programming to select a budget-constrained path that lexicographically minimises the largest edge errors. At the same budget, this objective gives VBench 79.27, compared with 76.04 for a shortest sum-of-errors path. The authors argue that the difference arises because segment errors are neither independent nor additive. The schedule is precomputed and adds no runtime overhead. ReCache [13] instead learns a budget-conditioned distribution over which k steps to recompute, using a Plackett–Luce / Gumbel-top-k construction and REINFORCE. Because it learns the schedule rather than changing the reuse mechanism, it can be applied to FORA, Δ-DiT, DiCache and TaylorSeer; on FLUX, for example, it reduces LPIPS by 31% relative to DiCache at a fixed budget. The learned-router methods L2C and HarmoniCa [6] also belong here. HarmoniCa addresses two train/inference mismatches in L2C: training at one sampled timestep ignores accumulated cache error, and per-step noise error does not directly measure final-image error. It trains over the denoising trajectory with an image-error proxy and reports about +6.7 Inception Score and −1.2 FID over L2C at ten steps on DiT-XL.

Content- and prompt-aware schedules. A final refinement observes that the right *threshold*, not just the right indicator, is content-dependent. PromptTea [12] shows that a single fixed δ cannot serve all prompts (a complex, high-motion scene needs a lower threshold to stay faithful, a simple one tolerates a higher threshold for more speed), and so derives δ from the prompt itself: it pre-computes "simple" and "complex" text-embedding prototype sets and maps an input prompt's cosine similarity to them into a complexity coefficient that interpolates between a low and a high threshold. It also sharpens TeaCache's indicator (on full-3D-attention video DiTs the bare timestep embedding correlates with the output better than the modulated input does) and adds a dynamic classifier-free-guidance cache that reuses the unconditional branch adaptively rather than on a fixed interval. The combination reaches 2.8× on Wan2.1 with better fidelity than fixed-threshold TeaCache, the gap widening on the complex prompts where a fixed threshold most badly mis-allocates compute.

A complementary axis: caching the guidance pair. The schedules above all reuse computation across *timesteps*, but classifier-free guidance opens a second, orthogonal source of redundancy within a single step. The denoiser is evaluated twice per step, once conditioned on the prompt and once unconditionally, and the two outputs are strongly correlated. FasterCache [23] caches the difference between them, splitting it into low- and high-frequency parts that are reused with timestep-dependent weights, and PromptTea's DynCFGCache replaces that fixed-interval reuse with the accumulated-difference threshold it already applies to temporal caching. Because the guidance axis is independent of the temporal one, the two compose: a step can be skipped entirely, and on the steps that are computed, the unconditional branch can be reused rather than recomputed. The axis is not universal, since some video models such as HunyuanVideo distil guidance into a single forward pass and so expose no second branch to cache, which is exactly why PromptTea disables this component on those backbones.

3.3 How to reuse: from reusing to forecasting

The reuse rule received less attention in early work. Most methods simply *copy* the cached value, which TaylorSeer [5] describes as a *zeroth-order* (constant) predictor. Because feature similarity decreases with timestep distance, copying becomes less accurate at high acceleration. TaylorSeer instead treats reuse as *forecasting*: if the feature trajectory and its finite differences are sufficiently smooth, a Taylor expansion fitted to fully computed steps can predict the skipped values. From steps spaced N apart, the feature at offset k is forecast as

$$F_{\text{pred},m}\big(x^l_{t-k}\big) = F\big(x^l_t\big) + \sum_{i=1}^{m} \frac{\Delta^i F\big(x^l_t\big)}{i!\, N^i}\,(-k)^i, \tag{3}$$

where $\Delta^i F$ is the i-th forward finite difference of the cached feature; order $m{=}0$ recovers copying, $m{=}1$ a linear trend, and larger m tracks curvature. The Taylor remainder scales as $|k|^{m+1}/(m{+}1)!$ in the offset k. The additional arithmetic does not require more full network evaluations because the finite differences are assembled from previously computed steps. TaylorSeer reports roughly 5× speedup on FLUX and HunyuanVideo with small quality changes. The gains saturate beyond third order as high-order finite differences become noisy, and each order requires another full-resolution state. This memory cost is substantial for long videos. DiCache therefore uses first-order prediction from a shallow probe, although it also describes a k-th-order extension with geometrically decaying weights on older entries. ClusCa [9] combines token-cluster propagation with a Taylor forecast. These methods can be described by prediction order: copying at order zero, linear alignment at order one, and higher-order forecasting.

The reuse rule and the schedule are not independent. A higher prediction order tolerates a longer gap between full evaluations, so a forecasting method can use a sparser schedule than a copying method for the same quality. This is why TaylorSeer reports useful acceleration past the point at which copy-based

methods fail, and why DiCache pairs its online schedule with a first-order rule rather than plain copying. The cost is the extra state each order requires, which is why the long-video methods keep the order low even though, as noted above, higher orders add little arithmetic. Choosing an order is therefore also a choice about how aggressive the schedule can be.

4 Toward a unified view of feature caching

Several connections appear across the methods, although the papers do not use the same terminology. SenCache's sensitivity decomposition treats the TeaCache and MagCache indicators as different terms in a first-order bound on output change; SeaCache and DiCache estimate related changes in spectral and shallow-feature spaces. Scheduling and reuse are also coupled in practice. TaylorSeer and DiCache vary how a feature trajectory is extrapolated from previous evaluations, whereas LeMiCa and ReCache vary where those evaluations are placed. This suggests describing a caching method by the locations of its full evaluations and the approximation used between them. The description is supported by several recurring observations: features often have stable short-range derivatives [5], residual direction changes more slowly than residual magnitude [7], early-step errors can affect global structure [3, 8, 14], and token clusters remain relatively stable across nearby steps [9]. These observations are compatible, but they have not yet been tested under a common experimental protocol.

TeaCache and MagCache both aggregate a per-step signal over a reuse interval, but they do so differently. TeaCache adds rescaled feature differences as in Eq. (2), whereas MagCache derives a multiplicative approximation from successive residual-magnitude ratios, $1 - \prod_i \gamma_i$. The two constructions play a similar operational role—they determine when accumulated change is large enough to refresh the cache—but they are not estimates of an established common error quantity. This distinction matters when comparing their theoretical interpretations.

There is a useful analogy with adaptive numerical integration. TaylorSeer's finite-difference formula (3) is Newton forward interpolation, accumulated indicators such as Eq. (2) play a role similar to local error estimates, and LeMiCa chooses evaluation points globally. The analogy is not exact because caching approximates internal network features rather than directly changing the numerical solver. It nevertheless helps explain a common failure mode: near the end of sampling, high-frequency detail develops quickly and extrapolation becomes less reliable, so methods tend to recompute more densely. Classical solver ideas such as embedded error estimates and automatic order selection may therefore be relevant, but their value for feature caching still needs experimental confirmation.

The sensitivity and Taylor accounts mainly describe temporal change: a feature trajectory is sampled, measured and sometimes extrapolated. They do not directly model the spatial redundancy used by token-level methods, nor do they predict the architecture-dependent caching patterns observed in L2C. Temporal, spatial and architectural redundancy should therefore be treated as related but distinct parts of the design problem.

5 Evaluation: models, benchmarks, and metrics

The experimental substrate has shifted in lock-step with the architectures. Early work evaluated U-Net image models (DDPM, LDM and Stable Diffusion on CIFAR, LSUN, ImageNet and COCO) [1]. DiT-era image work moved to DiT-XL, U-ViT, PixArt and the flow-matching model FLUX [2, 3, 6, 5], while the video wave standardised on Open-Sora, Open-Sora-Plan, Latte, CogVideoX, HunyuanVideo and Wan2.1 [4, 7, 8, 11, 12, 15].

These regimes stress different parts of a caching method, so results do not necessarily transfer between them. U-Net image models expose structural redundancy through their skip geometry. DiT image models such as PixArt and FLUX remove that geometry and shift attention toward layers, tokens and residual-space indicators. Video DiTs add very long token sequences and tighter memory constraints. Caching can save substantial time in this setting because a single clip may take minutes, but storing high-order feature histories is also more expensive. The recent LeMiCa, SenCache and ReCache experiments focus on this video regime. Results on DiT-XL/ImageNet alone therefore provide limited evidence about behaviour on a billion-parameter video model.

The reported operating points give a rough sense of scale. TeaCache reaches about 4× on Open-Sora-Plan, MagCache reports roughly 2.1–2.7× in several video settings, LeMiCa reaches 2.9× on Latte, TaylorSeer reports about 5× on both FLUX and HunyuanVideo, ClusCa reports about 5× on FLUX and 6× on

HunyuanVideo, and DuCa reaches about 3.5× on FLUX. These figures cannot establish which modality tolerates more caching because the models, reuse rules, hardware and quality criteria differ. They do show that aggressive speedups have been reported for both image and video generation. Within a matched method and setting, increasing the interval between full evaluations tends to make direct copying less accurate, which motivates the forecasting methods of Section 3.3.

Metrics fall into two semantically different families, a distinction the literature does not always make explicit. *Distributional / preference* metrics (FID and Inception Score [25], CLIP score, ImageReward [26], and the video benchmark VBench [24]) measure absolute generation quality against a reference distribution or a human-preference model. *Fidelity-to-original* metrics (LPIPS, PSNR and SSIM computed against the *un-accelerated* model's own output) measure how faithfully the cached run reproduces what the full model would have produced. The two can diverge sharply: a method can preserve VBench while degrading PSNR, and several methods even *exceed* the un-accelerated VBench or ImageReward, which complicates any literal reading of "lossless." Efficiency is also reported as both theoretical FLOPs/MACs and wall-clock latency. These measures can disagree, notably for token-importance methods such as ToCa [22], whose attention-score selection is incompatible with FlashAttention and increases real latency even when FLOPs fall [10].

Several papers compare caching with simply reducing the number of sampling steps. TeaCache reports that Open-Sora with a 50% refresh ratio retains a VBench score of 78.88%, whereas reducing the trajectory to fifteen steps gives 77.34%; DeepCache reports a similar result against DDIM/PLMS at matched throughput for U-Net image models. These results suggest that retaining the original noise schedule while skipping selected computations can be preferable to coarsening the schedule everywhere. Caching is therefore usually presented as complementary to fast sampling. This objective also motivates reference-based metrics: LPIPS, PSNR and SSIM against the un-accelerated output measure how closely the cached run follows the original trajectory. For example, SeaCache reports an approximately 8 dB PSNR advantage over earlier methods on HunyuanVideo, while ReCache reports a 31% LPIPS reduction relative to DiCache on FLUX.

VBench decomposes video quality into sixteen dimensions, including subject and background consistency, motion smoothness, temporal flicker, dynamic degree, visual quality and prompt alignment. Its aggregate score can hide trade-offs: aggressive caching may improve consistency or smoothness while reducing motion or fine detail. Reference-based metrics reveal a different part of the comparison. At about 1.5× speedup on Open-Sora, LeMiCa reports LPIPS 0.050 against the un-accelerated output, compared with TeaCache's 0.134, even though both aggregate VBench scores are within 0.1 of the original score of 79.2. MagCache reports a similar LPIPS difference, 0.083 against 0.130. Results of this kind make the evaluation protocol important: an aggregate score alone cannot determine whether two cached outputs remain equally faithful to the original model.

Most of the reported evidence comes from automatic metrics. VBench, ImageReward and CLIP score approximate aspects of human judgement, while reference-based metrics measure similarity to the original model rather than perceived quality. Because few of the reviewed papers include human evaluation, the perceptual significance of a small LPIPS or VBench difference is often unclear, particularly at aggressive operating points.

Caching can be combined with different numerical solvers, but the two components may interact. MagCache reports that a magnitude curve calibrated with one scheduler transfers to another in its tested setting. L2C gives a counterexample to treating the components as fully independent: with a second-order solver, its cache steps must be shifted so that the finite-difference derivative uses freshly evaluated points; otherwise FID degrades from 2.8 to 5.3. Such interactions are likely to be especially important for distilled few-step samplers, where each evaluation carries more weight.

6 Limits of the current evidence

The reported results support the usefulness of caching, but several aspects of the evidence limit direct comparison.

FLOPs do not predict latency. FLOPs reductions routinely overstate real gains. DuCa's central finding is that ToCa's importance scoring breaks memory-efficient attention and roughly doubles latency relative to its FLOPs [10]; HarmoniCa explicitly reports *real* rather than theoretical speedups for this reason. Token-level methods are especially prone to this gap, and not every paper reports latency on the same hardware.

Speedup is defined inconsistently. Headline "$N\times$" figures are not always on the same scale. Papers use different reference samplers, step counts, resolutions, precision settings and hardware, and some report a FLOPs ratio while others emphasise measured latency. Because the result depends on all of these choices,

speedup figures from different papers are often not directly comparable. Each result should state the reference sampler, generation configuration, hardware and whether speedup is based on operations or wall-clock time.

The role of calibration remains unsettled. TeaCache fits its polynomial on 70 prompts; MagCache argues that this procedure can overfit and reports that one random sample is sufficient for its magnitude curve; DiCache replaces an offline prior with an online probe; and HarmoniCa uses image-free training. These approaches make different assumptions about how much calibration is necessary, and they have not been compared under one controlled protocol. When calibration and evaluation prompts come from related sources, data overlap is also a possible concern, although the reviewed evidence does not quantify its effect.

Rankings depend on the operating point. Caching trades speed against quality along a continuous curve governed by δ (or the budget k), yet most comparisons report a single (speedup, quality) pair at a self-chosen threshold, and the full curve is shown only occasionally. Because the curves of competing methods cross, the reported ranking can change with the selected threshold. Comparisons at matched quality or matched speed across the full curve would be more informative than one point chosen separately for each method.

Memory is missing from most comparisons. Higher-order reuse buys accuracy with state. TaylorSeer must store multiple finite differences and runs out of memory on HunyuanVideo in at least one configuration; DiCache deliberately stays first-order to avoid the memory burden of many-frame video; MagCache notes that several competitors need tens of gigabytes of extra memory where it needs about half a gigabyte. Memory is as real a cost as latency but is reported far less consistently.

"Lossless" is used for two different claims. Reference-based LPIPS/PSNR/SSIM reward *mimicking* the un-accelerated model, not absolute quality, while reference-free VBench sometimes *improves* under caching. The community uses "lossless" for both, and the conflation obscures whether a method is faithfully reproducing the original or merely producing a different output of similar benchmark score.

7 Research gaps and directions

The limitations above leave several questions open.

Evaluation on common ground. A shared evaluation would make many current claims easier to assess. It should include a U-Net image model, a DiT or flow-based image model, and at least one large video DiT, because performance in one regime may not transfer to another. Full speed–quality curves are more informative than one threshold chosen separately for each method. Calibration and evaluation prompts should be disjoint, while hardware, precision and software settings should be fixed. Reporting latency and peak memory together with FLOPs would also capture implementation effects such as FlashAttention compatibility. Reference implementations under this protocol would allow claims such as "beats TeaCache" to be checked at matched operating points.

Joint scheduling and prediction. Scheduling, error estimation, reuse order and budget control have mostly been studied separately. LeMiCa addresses global step placement, SenCache studies local sensitivity, TaylorSeer changes the prediction order, and ReCache introduces budget-conditioned schedules. ReCache can be combined with several reuse mechanisms, but the literature has not yet evaluated a method that selects both recomputation points and prediction order under explicit compute and memory constraints.

Error control at the final output. SenCache's sensitivity bound and ReCache's output-consistency objective both suggest that the right target is the *final* generation error, not an intermediate proxy. A predictive theory that bounds final-output error as a function of the schedule and reuse order would convert today's heuristics into principled design.

Caching in few-step and compressed models. ReCache reports little benefit from caching in the 1–4-step regime, which is also where step-distilled models operate. The interaction with distillation, quantisation and sparse attention has received limited systematic study. HarmoniCa combines with the EfficientDM quantiser to increase a 1.18× speedup to 1.85× at a +0.12 FID cost; DiCache and Sparse VideoGen increase speedup from 1.67× to 3.08× on HunyuanVideo; and MagCache is tested on a 4-bit-quantised Wan2.1. These examples show compatibility in particular settings but do not establish when the gains will be additive. Related questions remain open for audio, 3D and autoregressive world models.

Low-overhead sample adaptation. The reviewed experiments generally favour per-sample schedules over fixed schedules, but adaptivity adds work. DiCache runs a shallow probe, SenCache estimates Jacobian terms by finite differences, and PromptTea embeds the prompt. It remains unclear whether quantities

already produced by the sampler can provide a sufficiently accurate indicator without an extra network evaluation or a separate calibration stage.

A concrete direction: budgeted adaptive-order caching. One possible study would make recomputation timing and reuse order a joint online decision. Residual norms and finite differences between recent full evaluations could estimate local change, after which the sampler would choose copying, first-order prediction, higher-order prediction or full recomputation subject to latency and memory limits. This would adapt to the current trajectory without running a separate probe network. The main question is whether the same decision rule transfers across architectures. It could be tuned on one model family and then evaluated without recalibration on U-Net image, DiT image and video models. Complete speed–quality–memory curves and worst-case degradation would show whether trajectory-derived signals are sufficiently reliable for such transfer.

Scope of this review. We cover fifteen closely related methods rather than the whole efficiency literature; orthogonal accelerators and the steady stream of concurrent caching preprints are out of scope, and the synthesis reflects the field as of mid-2026. We also rely on each paper's self-reported numbers, which, for the reasons catalogued in Section 6, are not strictly comparable across sources; the qualitative trends we draw are robust, but we do not claim to certify the precise quantitative rankings.

What this review adds. The what/when/how taxonomy places fifteen methods and their shared baselines in a common structure. The review also connects several scheduling indicators to sensitivity analysis and several reuse rules to Taylor approximation, while identifying where current comparisons remain inconclusive. In practical terms, architecture constrains the useful granularity, the available overhead constrains how adaptive the schedule can be, and aggressive speedups may require prediction rather than direct copying. These are tendencies in the reviewed experiments rather than universal selection rules.

8 Conclusion

Feature caching has expanded from U-Net feature reuse to a broader set of methods for DiTs and video models. Fixed schedules remain useful baselines, but recent papers increasingly adapt recomputation to the timestep, sample or compute budget. Reuse has also moved beyond copying toward first- and higher-order prediction. The what/when/how taxonomy makes these differences explicit, while sensitivity analysis and Taylor approximation provide partial connections between them. The evidence does not yet support a universal caching policy: results depend on the architecture, sampler, target speedup and metric, and memory is often omitted. More comparable speed–quality curves, consistent latency and memory measurements, and stronger links between local indicators and final-output error are needed before the relative merits of these methods can be established reliably.

References

[1] X. Ma, G. Fang, and X. Wang. DeepCache: Accelerating Diffusion Models for Free. In *CVPR*, 2024. arXiv:2312.00858.

[2] X. Ma, G. Fang, M. B. Mi, and X. Wang. Learning-to-Cache: Accelerating Diffusion Transformer via Layer Caching. In *NeurIPS*, 2024. arXiv:2406.01733.

[3] P. Chen, M. Shen, P. Ye, J. Cao, C. Tu, C.-S. Bouganis, Y. Zhao, and T. Chen. Δ-DiT: A Training-Free Acceleration Method Tailored for Diffusion Transformers. arXiv preprint arXiv:2406.01125, 2024.

[4] F. Liu, S. Zhang, X. Wang, Y. Wei, H. Qiu, Y. Zhao, Y. Zhang, Q. Ye, and F. Wan. Timestep Embedding Tells: It's Time to Cache for Video Diffusion Model. In *CVPR*, 2025. arXiv:2411.19108.

[5] J. Liu, C. Zou, Y. Lyu, J. Chen, and L. Zhang. From Reusing to Forecasting: Accelerating Diffusion Models with TaylorSeers. In *ICCV*, 2025. arXiv:2503.06923.

[6] Y. Huang, Z. Wang, R. Gong, J. Liu, X. Zhang, J. Guo, X. Liu, and J. Zhang. HarmoniCa: Harmonizing Training and Inference for Better Feature Caching in Diffusion Transformer Acceleration. In *ICML*, 2025. arXiv:2410.01723.

[7] Z. Ma, L. Wei, F. Wang, S. Zhang, and Q. Tian. MagCache: Fast Video Generation with Magnitude-Aware Cache. In *NeurIPS*, 2025. arXiv:2506.09045.

[8] H. Gao, P. Chen, F. Shi, C. Tan, Z. Liu, F. Zhao, K. Wang, and S. Lian. LeMiCa: Lexicographic Minimax Path Caching for Efficient Diffusion-Based Video Generation. In *NeurIPS* (Spotlight), 2025. arXiv:2511.00090.

[9] Z. Zheng, X. Wang, C. Zou, S. Wang, and L. Zhang. Compute Only 16 Tokens in One Timestep: Accelerating Diffusion Transformers with Cluster-Driven Feature Caching. In *Proc. 33rd ACM International Conference on Multimedia (MM '25)*, 2025. arXiv:2509.10312.

[10] C. Zou, S. Zheng, E. Zhang, R. Guo, H. Xu, Z. Shi, C. He, X. Hu, and L. Zhang. Rethinking Token-wise Feature Caching: Accelerating Diffusion Transformers with Dual Feature Caching. arXiv preprint arXiv:2412.18911, 2025.

[11] J. Bu, P. Ling, Y. Zhou, Y. Wang, Y. Zang, D. Lin, and J. Wang. DiCache: Let Diffusion Model Determine Its Own Cache. In *ICLR*, 2026. arXiv:2508.17356.

[12] Z. Huang, C. Yang, and M. Ren. PromptTea: Let Prompts Tell TeaCache the Optimal Threshold. arXiv preprint arXiv:2507.06739, 2025.

[13] M. Aliev, E. Neudachina, I. Bykov, A. Oganov, K. Struminsky, A. Alanov, and D. Rakitin. ReCache: Learning Budget-Aware Caching Schedules for Diffusion Models via REINFORCE. arXiv preprint arXiv:2606.06060, 2026.

[14] J. Chung, S. Hyun, M. Lee, B. Han, G. Cha, D. Wee, Y. Hong, and J.-P. Heo. SeaCache: Spectral-Evolution-Aware Cache for Accelerating Diffusion Models. In *CVPR* (Oral), 2026. arXiv:2602.18993.

[15] Y. Haghighi and A. Alahi. SenCache: Accelerating Diffusion Model Inference via Sensitivity-Aware Caching. In *CVPR* (Oral), 2026. arXiv:2602.24208.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *NeurIPS*, 2020.

[17] J. Song, C. Meng, and S. Ermon. Denoising Diffusion Implicit Models. In *ICLR*, 2021.

[18] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *CVPR*, 2022.

[19] W. Peebles and S. Xie. Scalable Diffusion Models with Transformers. In *ICCV*, 2023.

[20] P. Selvaraju, T. Ding, T. Chen, I. Zharkov, and L. Liang. FORA: Fast-Forward Caching in Diffusion Transformer Acceleration. arXiv preprint arXiv:2407.01425, 2024.

[21] X. Zhao, X. Jin, K. Wang, and Y. You. Real-Time Video Generation with Pyramid Attention Broadcast. In *ICLR*, 2025. arXiv:2408.12588.

[22] C. Zou et al. Accelerating Diffusion Transformers with Token-wise Feature Caching (ToCa). In *ICLR*, 2025. arXiv:2410.05317.

[23] Z. Lv, C. Si, J. Song, Z. Yang, Y. Qiao, Z. Liu, and K.-Y. K. Wong. FasterCache: Training-Free Video Diffusion Model Acceleration with High Quality. In *ICLR*, 2025. arXiv:2410.19355.

[24] Z. Huang et al. VBench: Comprehensive Benchmark Suite for Video Generative Models. In *CVPR*, 2024.

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *NeurIPS*, 2017.

[26] J. Xu, X. Liu, Y. Wu, Y. Tong, Q. Li, M. Ding, J. Tang, and Y. Dong. ImageReward: Learning and Evaluating Human Preferences for Text-to-Image Generation. In *NeurIPS*, 2023.