PEACACHE: PROGRESSIVE ERROR ALIGNMENT FOR DIFFUSION TRANSFORMER CACHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers (DiTs) exhibit strong generative capabilities but require repeated and computationally expensive denoiser evaluations. Feature caching reduces this cost by reusing intermediate computations, but inevitably introduces reuse-induced errors. In this work, we propose **PEACache**, a Progressive Error Alignment framework that controls caching errors at three progressive levels. At the *branch level*, instead of modeling the Transformer residual at a cached step as in most existing methods, we directly model the error caused by residual reuse, i.e., the missing residual change. Through analysis, we show that this error can be more effectively corrected using historical residual changes, motivating a historical residual change subspace with adaptive spectral rank and update magnitude. At the *classifier-free guidance (CFG) level*, we show that independently optimizing branch-level errors can still lead to larger CFG output errors, since the guidance scale can amplify the mismatch between conditional and unconditional branch. We therefore supervise cache prediction directly on the CFG-composed output. At the *trajectory level*, we estimate the impact of caching each individual step on the final latent and use their accumulated impact as a scheduling surrogate, which is optimized via dynamic programming under a computation budget. On Wan2.1-1.3B, Open-Sora 1.2, and CogVideoX, our method substantially improves output fidelity at comparable acceleration. On Wan2.1-1.3B, among compared methods achieving over $3\times$ acceleration, PEACache improves the previous best SSIM/PSNR/LPIPS from $0.629/17.160/0.290$ to $0.730/21.543/0.188$.

1 INTRODUCTION

Diffusion models have become a dominant paradigm for visual generation (Sohl-Dickstein et al., 2015; Song & Ermon, 2019; Ho et al., 2020; Lipman et al., 2023), supporting recent advances in both image (Rombach et al., 2022; Esser et al., 2024; Black Forest Labs, 2024) and video generation (Zheng et al., 2024; Kong et al., 2024; HaCohen et al., 2024; Wan et al., 2025; Yang et al., 2025). Their backbones have also evolved from convolutional U-Nets (Ronneberger et al., 2015) to Diffusion Transformers (DiTs) (Vaswani et al., 2017; Peebles & Xie, 2023), substantially increasing modeling capacity and generation quality. These gains come at a high inference cost: producing one sample requires repeatedly evaluating a large denoiser over many sampling steps, making generation latency a major bottleneck.

Existing DiT acceleration methods can be broadly divided into training-based and training-free approaches. Training-based methods reduce sampling steps or compress the model (Salimans & Ho, 2022; Song et al., 2023; Xie et al., 2024; Lu et al., 2024), but typically require additional fine-tuning. Training-free methods instead accelerate pretrained models through more efficient computation or fewer model evaluations (Xi et al., 2025; Zhang et al., 2024; Song et al., 2021; Lu et al., 2022; Zhao et al., 2023), making them easier to apply to rapidly evolving image and video generators.

Feature caching has recently gained increasing attention as a practical approach to diffusion acceleration, owing to its training-free nature and ease of deployment in existing inference pipelines (Ma et al., 2024; Selvaraju et al., 2024; Chen et al., 2024; Liu et al., 2025a; Lv et al., 2024). Early caching methods exploit temporal redundancy between neighboring denoising steps to directly reuse intermediate features (Wimbauer et al., 2024; Zhao et al., 2025; Liu et al., 2025a), with reuse decisions typically determined by local, step-wise feature discrepancies. More recent methods have extended

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.

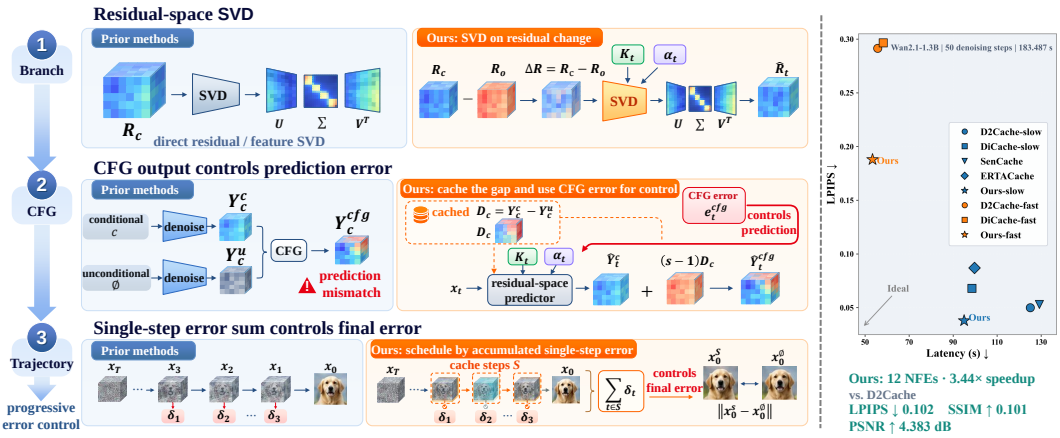

Figure 1: **Overview of Progressive Error Alignment.** We progressively align cache-induced errors at three increasingly global levels: at the *branch level*, we correct the residual-reuse error by predicting the missing residual change; at the *CFG level*, we align cache correction with the CFG-composed output rather than individual branch errors; and at the *trajectory level*, we schedule cached steps according to their impact on the final latent. On Wan2.1-1.3B, our method achieves a favorable quality-efficiency trade-off, reaching 3.44× speedup with only 12 NFEs while substantially improving fidelity over comparable cache baselines.

this paradigm along two directions: improving the reused features by forecasting their current values from historical observations (Liu et al., 2025b; Chen et al., 2026), and improving the caching schedule by accounting for the global impact of individual caching decisions on the final output (Gao et al., 2025; Lei et al., 2026).

In this paper, we revisit feature caching from a more fundamental, error-centric perspective: rather than directly approximating the features to be reused, we explicitly model and control the errors introduced by cache reuse. We trace caching errors along the generation process and progressively align them at increasingly global levels. Based on this principle, we propose **PEACache**, a Progressive Error Alignment framework that aligns caching errors at three levels: the residual-reuse error at the branch level, the CFG-composed error at the guidance level, and the final-latent error at the output level.

At the branch level, we shift the modeling target from the Transformer residual itself to the error introduced by residual reuse, i.e., the missing residual change. Through SVD analysis, we find that the channel-side subspace of historical residual changes is more stable across timesteps, while the token-side structure varies more substantially. This motivates reusing the stable channel-side structure while adaptively selecting the transferable spectral support and scaling the update magnitude. Unlike SVD-Cache (Chen et al., 2026), which applies SVD directly to residual features, we model the reuse-induced error; Appendix A.3 further shows why residual changes provide a more effective representation for cache correction.

At the guidance level, we align cache errors with what is actually used by the sampler. Since CFG combines conditional and unconditional predictions with a guidance scale (Ho & Salimans, 2022), reducing the two branch errors independently does not necessarily reduce the CFG error, as their mismatch can be amplified by guidance. Unlike FasterCache (Lv et al., 2024), which exploits conditional–unconditional redundancy to reduce CFG computation, we focus on their error interaction after CFG composition and directly supervise cache prediction with the CFG-composed output.

At the trajectory level, we align caching errors with their impact on the final generation. The final-latent error of jointly cached steps is not additive, i.e., it is not equal to the sum of their individual errors. Nevertheless, our two-step cache-schedule analysis suggests that the sum of per-step final-latent impacts can provide a useful signal for ranking candidate schedules, motivating its use as a

scheduling surrogate. Based on this surrogate, a single dynamic programming procedure can obtain cache schedules for different numbers of cached steps.

Together, these three designs instantiate Progressive Error Alignment at progressively broader scopes: correcting reuse-induced residual changes within each branch, aligning errors after CFG composition, and optimizing cache schedules according to their impact on the final latent.

Our contributions can be summarized as follows:

- We propose **PEACache**, a Progressive Error Alignment framework that controls cache-induced errors at the branch, CFG, and final-generation levels.
- For cache prediction, we directly model the error caused by residual reuse using historical residual changes with adaptive spectral rank and update magnitude. More importantly, to our knowledge, we are the first to supervise cache prediction directly with the CFG-composed output, rather than optimizing the conditional and unconditional branches independently.
- For cache scheduling, we use the accumulated final-latent impact of cached steps as a scheduling surrogate and optimize the schedule via dynamic programming. Experiments on three video diffusion models show substantially improved fidelity at comparable acceleration.

2 Related Work

Diffusion Models Acceleration. Diffusion sampling repeatedly evaluates a denoising network, making acceleration especially important for high-resolution image and video generation. Existing methods reduce either the number or the cost of denoiser evaluations. DDIM enables shorter deterministic sampling trajectories (Song et al., 2021), while DPM-Solver and UniPC use higher-order numerical updates to maintain quality with fewer steps (Lu et al., 2022; Zhao et al., 2023). Training-based approaches include progressive distillation and consistency models for few-step generation (Salimans & Ho, 2022; Song et al., 2023), as well as efficient architectures such as SANA (Xie et al., 2024) and quantized models such as TerDiT (Lu et al., 2024). Training-free methods instead accelerate pretrained models without additional fine-tuning. For example, Sparse VideoGen exploits spatiotemporal attention sparsity (Xi et al., 2025), while SageAttention uses low-precision attention computation (Zhang et al., 2024). Among training-free approaches, cache-based methods further exploit temporal redundancy across denoising steps by reusing intermediate computations.

Cache-based Acceleration. Cache-based acceleration methods are attractive in practice due to their simplicity and training-free nature. Early methods such as PAB (Zhao et al., 2025) and TeaCache (Liu et al., 2025a) exploit the temporal similarity of intermediate features across nearby timesteps and reuse previously computed attention outputs or transformer residuals. FasterCache (Lv et al., 2024) further explores redundancy within classifier-free guidance (CFG), reusing information between the conditional and unconditional branches to reduce redundant computation. TaylorSeer (Liu et al., 2025b) introduces history-based prediction, using temporal derivatives to estimate features at cached timesteps instead of directly reusing the previous result. Along this direction, SVD-Cache (Chen et al., 2026) observes that different components of historical features exhibit different temporal predictability and uses SVD to decompose them into subspaces, forecasting the dominant principal components while directly reusing the less predictable residual components. D2Cache (Liu et al., 2026) leverages second-order residual variations for more accurate extrapolation, DiCache (Bu et al., 2026) uses shallow-feature probes and multiple historical caches for dynamic error correction, and ERTACache (Peng et al., 2026) mitigates accumulated caching errors through residual correction and trajectory adjustment. Beyond improving cached-step prediction, LeMiCa (Gao et al., 2025) and BudCache (Lei et al., 2026) emphasize global cache scheduling, optimizing cache decisions over the entire denoising trajectory under a given computation budget rather than relying only on local per-step error estimates.

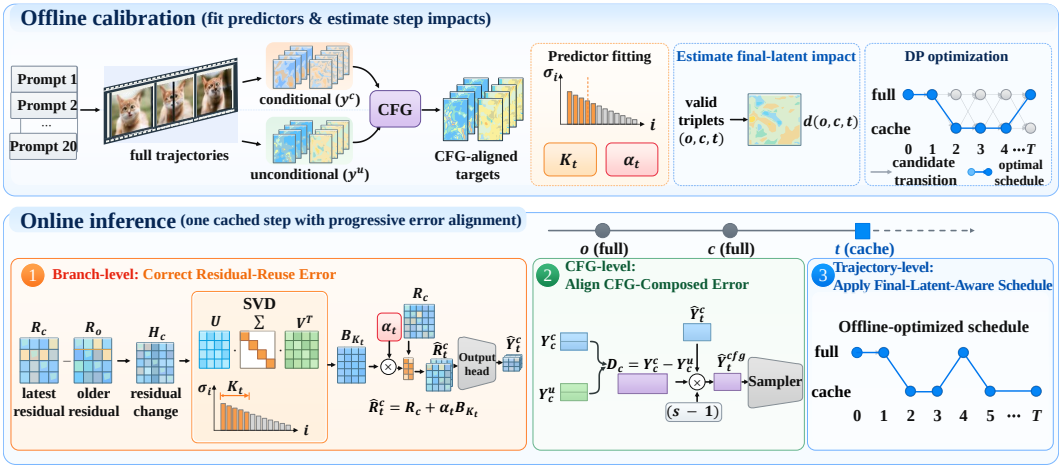

Figure 2: **Overview of Progressive Error Alignment.** Offline, we fit rank/scale predictors with CFG-aligned targets and estimate cache-induced final-latent impacts for DP scheduling. Online, **(1)** branch-level alignment corrects residual reuse using historical motion, **(2)** CFG-level alignment applies CFG-aligned predictors and reuses the cached branch gap to construct the CFG-composed output, and **(3)** trajectory-level alignment applies the offline-optimized final-latent-aware cache schedule.

3 METHOD

Fig. 2 illustrates the overall pipeline of our method. Following the principle of Progressive Error Alignment, we progressively trace and align cache-induced errors from local feature reuse to their global effect on the final generation.

3.1 BRANCH-LEVEL ALIGNMENT VIA RESIDUAL-CHANGE PREDICTION

Let $o < c < t$ denote the two most recent full-computation steps and a target cached step in sampling order. For $j \in \{o, c, t\}$, let R_j denote the token-by-channel Transformer residual under full computation: the output of the Transformer block stack before the final projection minus its input. Directly reusing R_c at step t misses the change $H_t = R_t - R_c$, where R_t is the full-computation reference. We use the observed historical change $H_c = R_c - R_o$ to predict this missing residual change. We decompose H_c into singular components:

$$H_c = U\Sigma V^\top, \qquad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r), \tag{1}$$

where U and V span the token and channel directions, respectively, r is the rank, and the singular values are ordered in descending magnitude. Our primary interest is the temporal consistency of these singular components. As shown in Appendix A.1, the channel-side structure is more stable across timesteps, while the token-side structure varies more and its reusability changes along the denoising trajectory. This motivates adapting the transferred spectral support to each target timestep.

Accordingly, we retain the top-K_t singular modes of H_c, $B_{K_t} = H_c V_{K_t} V_{K_t}^\top = U_{K_t}\Sigma_{K_t} V_{K_t}^\top$, to approximate the future residual change. Since the transferable spectral support varies across diffusion phases and prediction horizons, we predict a continuous spectral threshold and retain the singular values above it to determine K_t. We further introduce a temporal scaling factor α_t to adjust the update magnitude according to the temporal geometry and the retained spectral energy.

Combining the adaptive spectral selection and temporal scaling, at the branch level, we use $\alpha_t B_{K_t}$ to approximate the missing residual change H_t, and reconstruct the target residual as

$$\widehat{R}_t = R_c + \alpha_t B_{K_t}. \tag{2}$$

Here, K_t determines how many dominant singular modes of the historical residual change are retained, while α_t controls the magnitude of the predicted change. We fit both predictors offline using

Table 1: Oracle RMSE comparison of different CFG optimization formulations. Lower is better.

Method	$\mathrm{RMSE}(e^{\mathrm{c}})$	$\mathrm{RMSE}(e^{\mathrm{u}})$	$\mathrm{RMSE}(e^{\mathrm{c}} - e^{\mathrm{u}})$	$\mathrm{RMSE}(e^{\mathrm{cfg}})$
Direct reuse	0.085	0.086	0.013	0.103
Branch-wise $(Y^{\mathrm{c}}, Y^{\mathrm{u}})$	0.044	0.046	0.022	0.106
CFG $(Y^{\mathrm{c}}, Y^{\mathrm{u}})$	0.083	0.099	0.023	**0.076**
CFG $(Y^{\mathrm{c}}, D_c$ reuse$)$	0.046	0.046	0.014	0.077
CFG $(Y^{\mathrm{u}}, D_c$ reuse$)$	0.047	0.048	0.014	0.077

calibration prompts. At inference time, given H_c, we recover its singular spectrum and dominant components through eigendecomposition of the corresponding Gram matrix Golub & Van Loan (2013). The fitted models first predict a continuous spectral threshold to determine K_t, and then predict α_t from the temporal geometry and the retained spectral energy. The construction and supervision of both predictors are detailed in Appendix A.2.

3.2 Guidance-Level Alignment with CFG Output

The predictors in Section 3.1 determine the rank and update magnitude for cached residual reconstruction. A natural choice, as in most existing methods (Liu et al., 2025b; Bu et al., 2026; Peng et al., 2026), is to predict the conditional and unconditional branches independently. **However, this can be misleading, because the sampler ultimately consumes the CFG-composed output rather than either branch alone.**

Let Y_t^{c} and Y_t^{u} denote the exact conditional and unconditional denoiser outputs at step t, and $\widehat{Y}_t^{\mathrm{c}}$ and $\widehat{Y}_t^{\mathrm{u}}$ their cached approximations. With guidance scale $s > 1$, CFG produces (Ho & Salimans, 2022)

$$Y_t^{\mathrm{cfg}} = Y_t^{\mathrm{u}} + s\,(Y_t^{\mathrm{c}} - Y_t^{\mathrm{u}}) = Y_t^{\mathrm{c}} + (s-1)\,(Y_t^{\mathrm{c}} - Y_t^{\mathrm{u}})\,. \tag{3}$$

Defining the branch errors as $e_t^{\mathrm{c}} = \widehat{Y}_t^{\mathrm{c}} - Y_t^{\mathrm{c}}$ and $e_t^{\mathrm{u}} = \widehat{Y}_t^{\mathrm{u}} - Y_t^{\mathrm{u}}$, the resulting CFG error is

$$e_t^{\mathrm{cfg}} = s e_t^{\mathrm{c}} + (1-s) e_t^{\mathrm{u}} = e_t^{\mathrm{c}} + (s-1)(e_t^{\mathrm{c}} - e_t^{\mathrm{u}}). \tag{4}$$

This formulation makes the issue clear. Even if $\|e_t^{\mathrm{c}}\|$ and $\|e_t^{\mathrm{u}}\|$ are reduced, the discrepancy $\|e_t^{\mathrm{c}} - e_t^{\mathrm{u}}\|$ may still increase. Since CFG amplifies this term by $s-1$, it can contribute substantially to $\|e_t^{\mathrm{cfg}}\|$.

Based on equation 4, a natural solution is to jointly optimize the two branches with respect to the CFG output error. However, explicitly correcting both branches may not be necessary. Define the branch gap as $D_t = Y_t^{\mathrm{c}} - Y_t^{\mathrm{u}}$, such that $Y_t^{\mathrm{cfg}} = Y_t^{\mathrm{c}} + (s-1)D_t = Y_t^{\mathrm{u}} + sD_t$.

For a cached target step t, the exact gap D_t is unavailable without evaluating both branches. We therefore reuse the branch gap from the most recent full-computation step, $\widehat{D}_t = D_c = Y_c^{\mathrm{c}} - Y_c^{\mathrm{u}}$. This allows us to approximate the CFG output by correcting only one branch: $\widehat{Y}_t^{\mathrm{cfg}} = \widehat{Y}_t^{\mathrm{c}} + (s-1)D_c$, or alternatively $\widehat{Y}_t^{\mathrm{cfg}} = \widehat{Y}_t^{\mathrm{u}} + sD_c$ when correcting the unconditional branch.

We compare these two formulations with jointly correcting both conditional and unconditional branches on Wan2.1-1.3B. We randomly sample target steps and two preceding full-computation steps from several calibration prompts, and perform oracle optimization over the rank and update magnitude for each formulation. This isolates the best achievable error under each objective from predictor estimation errors. We additionally include independent branch-wise optimization and direct residual reuse as baselines. For each formulation, we report the RMSE of the conditional error, unconditional error, branch-discrepancy error, and CFG error, i.e., $\mathrm{RMSE}(e_t^{\mathrm{c}})$, $\mathrm{RMSE}(e_t^{\mathrm{u}})$, $\mathrm{RMSE}(e_t^{\mathrm{c}} - e_t^{\mathrm{u}})$, and $\mathrm{RMSE}(e_t^{\mathrm{cfg}})$, in Table 1.

From the table, Branch-wise $(Y^{\mathrm{c}}, Y^{\mathrm{u}})$ substantially reduces the RMSE of both individual branches compared with Direct reuse, but increases the branch-discrepancy RMSE from 0.013 to 0.022, causing the final CFG RMSE to increase from 0.103 to 0.106. This shows that independently improving the two branches can be counterproductive after CFG composition. Direct reuse, in contrast, preserves the branch gap particularly well, motivating the reuse of D_c as an approximation to D_t. Joint

CFG optimization of $(Y^{\mathrm{c}}, Y^{\mathrm{u}})$ achieves the lowest oracle CFG RMSE of 0.076. However, correcting only one branch while reusing D_c achieves a nearly identical CFG RMSE of 0.077, with a simpler prediction target. We therefore adopt conditional-branch correction with cached-gap reuse.

For a candidate rank K and scale α from Section 3.1, let $\widehat{Y}_t^{\mathrm{c}}(K, \alpha)$ denote the conditional output produced with the reconstructed residual. We construct the supervision targets using

$$\mathcal{L}_{\mathrm{cfg}}(K, \alpha) = \left\| \widehat{Y}_t^{\mathrm{c}}(K, \alpha) + (s-1)D_c - Y_t^{\mathrm{cfg}} \right\|_F^2 . \tag{5}$$

Thus, the conditional branch is corrected by our residual-change predictor, while the conditional–unconditional gap is reused from the most recent full-computation step. The inference equations in Section 3.1 remain unchanged; Appendix A.2 details the corresponding target construction.

3.3 Trajectory-Level Alignment via Cache Scheduling

At the trajectory level, we control the final generation error through cache scheduling. Let T denote the number of sampling steps, indexed in execution order by $1, \ldots, T$, and let $\mathcal{S} \subseteq \{1, \ldots, T\}$ denote a cache schedule. For a calibration sample z, we denote the final latent obtained from the full-computation trajectory by $x_0^{\varnothing}(z)$ and that obtained under schedule $\mathcal{S}$ by $x_0^{\mathcal{S}}(z)$. The final-latent error of a schedule is measured as

$$E(\mathcal{S}) = \mathbb{E}_z \left[\frac{\left\| x_0^{\mathcal{S}}(z) - x_0^{\varnothing}(z) \right\|_2}{\left\| x_0^{\varnothing}(z) \right\|_2} \right]. \tag{6}$$

Ideally, we would like to estimate $E(\mathcal{S})$ for every candidate cache schedule and directly select the one with the smallest final-latent error under a given computation budget. Exhaustively evaluating all schedules, however, is prohibitively expensive. A more tractable alternative is to estimate the final-latent impact of caching each individual step and combine these per-step impacts to approximate the error of a multi-step schedule.

However, the difficulty here is that final-latent errors are generally not additive. For example,

$$E(\{i, j\}) \neq E(\{i\}) + E(\{j\}), \tag{7}$$

because caching step i changes the latent passed to subsequent steps, which in turn changes the effect of caching step j. Therefore, the impact of multiple cached steps cannot be represented exactly as the sum of their individual impacts.

This limitation, however, does not prevent us from using such an additive approximation for scheduling. The key is that cache scheduling does not require exact error estimation. Our goal is not to accurately predict the absolute final-latent error of every schedule, but rather to identify schedules with smaller errors among all candidates. Therefore, an additive surrogate can still be useful as long as it approximately preserves the relative ordering of candidate schedules. Concretely, if two schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfy

$$E(\mathcal{S}_1) < E(\mathcal{S}_2), \tag{8}$$

we would like their surrogate scores $\widetilde{E}$ to satisfy

$$\widetilde{E}(\mathcal{S}_1) < \widetilde{E}(\mathcal{S}_2) \tag{9}$$

as often as possible, even if $\widetilde{E}(\mathcal{S})$ does not numerically match the true error $E(\mathcal{S})$.

This observation shifts the question from exact additivity to whether accumulated single-step impacts preserve the ranking of cache schedules. We examine this through a small empirical study using two-step cache schedules on Wan2.1 1.3B. For each step i, we define its individual final-latent impact as

$$\delta_i = E(\{i\}), \tag{10}$$

and approximate the error of caching steps i and j jointly by

$$\widetilde{E}(\{i, j\}) = \delta_i + \delta_j. \tag{11}$$

We then compare the ranking induced by $\widetilde{E}(\{i, j\})$ with that of the true final-latent errors $E(\{i, j\})$ over candidate step pairs.

Table 2: Video-generation results on the VBench prompt set. δ denotes the cache threshold used by D2Cache and DiCache.

Method	Efficiency		Visual Quality			
	Speedup↑	Latency (s)↓	VBench↑	LPIPS↓	SSIM↑	PSNR↑
Wan2.1-1.3B (81 frames, 480P)						
Wan2.1-1.3B ($T = 50$)	1×	183.487	81.20	–	–	–
SenCache	1.42×	129.210	81.08	0.053	0.906	27.773
D2Cache ($\delta = 0.05$)	1.47×	125.050	**81.19**	0.050	0.908	28.307
ERTACache	1.84×	99.692	81.04	0.087	0.854	24.949
DiCache ($\delta = 0.08$)	1.86×	98.528	81.06	0.068	0.879	26.150
Ours-slow (24 NFEs)	**1.93×**	**94.991**	81.04	**0.038**	**0.924**	**30.525**
DiCache ($\delta = 0.20$)	3.19×	57.510	79.30	0.298	0.626	17.035
D2Cache ($\delta = 0.24$)	3.25×	56.384	80.28	0.290	0.629	17.160
Ours-fast (12 NFEs)	**3.44×**	**53.331**	**80.10**	**0.188**	**0.730**	**21.543**
Open-Sora 1.2 (51 frames, 480P)						
Open-Sora 1.2 ($T = 30$)	1×	47.676	79.70	–	–	–
D2Cache ($\delta = 0.20$)	2.22×	21.458	**78.91**	0.273	0.710	17.654
ERTACache	2.26×	21.128	78.72	0.166	0.808	22.275
Ours (12 NFEs)	2.24×	21.317	78.36	**0.100**	**0.862**	**26.373**
Ours (11 NFEs)	**2.40×**	**19.887**	78.17	0.117	0.846	25.288
CogVideoX (48 frames, 480P)						
CogVideoX ($T = 50$)	1×	80.337	76.62	–	–	–
ERTACache	2.61×	30.813	75.36	0.159	0.802	23.192
Ours (16 NFEs)	**2.64×**	**30.468**	**75.65**	**0.085**	**0.895**	**28.260**

Despite the lack of strict additivity, the additive surrogate preserves the schedule ranking well: it achieves a Spearman correlation of 0.983, a Kendall correlation of 0.908, and correctly orders 95.4% of schedule pairs. Moreover, for incorrectly ordered pairs, the average gap in their true final-latent errors is only 1.35×10^{-4}, suggesting that most ranking violations occur between schedules with very similar final-latent errors.

The above experiment suggests that, although cache-induced final-latent errors are not additive in general, accumulated single-step impacts can still provide a useful ranking signal for cache scheduling. We therefore estimate these impacts offline on a small calibration set and use their sum as a tractable surrogate for the error of a cache schedule. Since the impact of caching a step depends on its two preceding full-computation anchors, the surrogate accounts for this local trajectory context.

Given a cache-count budget, we optimize the surrogate objective using dynamic programming. Each decision either performs full computation, which updates the two anchors, or caches the current step, which keeps the anchors unchanged and incurs the corresponding calibrated cost. This yields a minimum-cost schedule for each feasible computation budget. Details of the offline calibration, surrogate construction, and dynamic-programming procedure are provided in Appendix A.4.

4 Experiments

4.1 Experimental Setup

Models and evaluation prompts. We evaluate three text-to-video backbones: Wan2.1-1.3B (Wan et al., 2025) generates 81 frames in 50 denoising steps, Open-Sora 1.2 (Zheng et al., 2024) generates 51 frames in 30 steps, and CogVideoX (Yang et al., 2025) generates 48 frames in 50 steps. All videos are generated at 480p. Following TeaCache (Liu et al., 2025a), we use the VBench prompt set (Huang et al., 2024) for evaluation. We randomly reserve 20 prompts for offline calibration of our predictors and cache scheduler, and exclude them from the final evaluation.

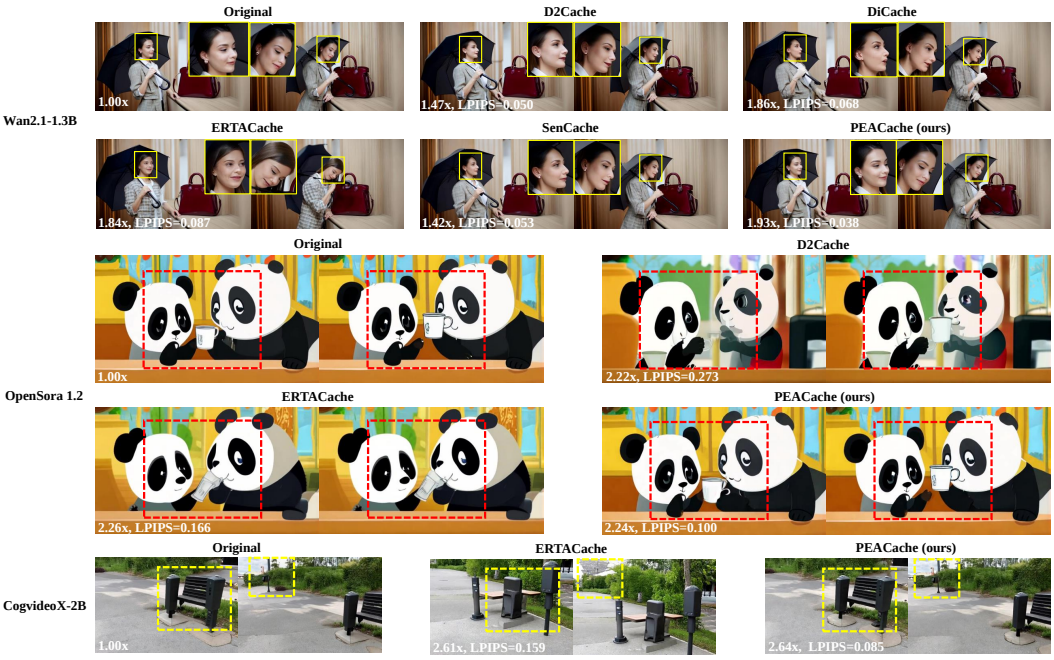

Figure 3: **Qualitative comparison with cache-based acceleration methods.** Representative frames under comparable acceleration ratios, with highlighted regions showing differences in visual details and structural fidelity. Speedup and LPIPS are reported for each method.

Baselines and measurements. We compare against D2Cache (Liu et al., 2026), DiCache (Bu et al., 2026), SenCache (Haghighi & Alahi, 2026), and ERTACache (Peng et al., 2026), using their official open-source implementations. Since ERTACache also requires offline calibration, it uses the same 20 calibration prompts as our method. All methods are evaluated on the same remaining VBench prompts. Following TeaCache (Liu et al., 2025a), we report VBench, SSIM (Wang et al., 2004), LPIPS (Zhang et al., 2018), and PSNR against paired full-computation videos, together with inference latency and the corresponding speedup over full computation.

Implementation details. Inference latency is measured on an NVIDIA A100 GPU. Since the runtime of cache-based methods such as D2Cache can vary across prompts even under the same threshold, we randomly sample a fixed set of 20 VBench prompts and use this same set to benchmark all methods, reporting the average latency. Batched video generation for quality evaluation is conducted on PPU-ZW810E accelerators. Following the official TeaCache evaluation protocol (Liu et al., 2025a), we generate five videos for each prompt using seeds $0, 1, 2, 3, 4$ and evaluate them with VBench, SSIM, LPIPS, and PSNR.

4.2 Experimental Results

Quantitative Comparison. Across all three backbones, our method consistently achieves substantially better fidelity than competing caching methods at comparable or even lower latency. On Wan2.1-1.3B, our 24-step setting takes 94.991 seconds, which is faster than DiCache (98.528 s) and ERTACache (99.692 s), while achieving markedly better LPIPS/SSIM/PSNR of $0.038/0.924/30.525$, compared with $0.068/0.879/26.150$ for DiCache and $0.087/0.854/24.949$ for ERTACache. Under the more aggressive 12-step setting, our method reaches a $3.44\times$ speedup with 53.331 seconds of latency, while still substantially outperforming D2Cache and DiCache at similar speedups on all three fidelity metrics. Similar trends are observed on Open-Sora 1.2 and CogVideoX, where our method consistently provides a better quality–efficiency trade-off than the compared baselines. Meanwhile, the VBench scores remain close across methods, indicating only minor differences in overall perceptual quality under this metric.

Table 3: Ablation of cache scheduling and cache prediction on Wan2.1-1.3B.

Method	SSIM ↑	PSNR ↑	LPIPS ↓
TeaCache	0.542	15.583	0.412
TeaCache + Our schedule	0.689	21.146	0.214
TeaCache + Our schedule + Our prediction	**0.730**	**21.543**	**0.188**

Qualitative Comparison. Fig. 3 further compares the visual fidelity of different caching methods. On Wan2.1-1.3B, under relatively moderate acceleration, PEACache preserves fine-grained details more faithfully than competing methods, particularly in facial regions, where other methods exhibit noticeable appearance distortions. Under more aggressive acceleration, the advantage becomes more evident in structural preservation: competing methods may introduce visible deformation or inconsistency in object shapes and spatial layouts, whereas PEACache better maintains the structure of the full-computation output while still retaining fine visual details. These qualitative observations are consistent with the improvements in LPIPS, SSIM, and PSNR reported in Table 2.

4.3 Ablation Study

We evaluate the contributions of cache scheduling and cache prediction on Wan2.1-1.3B by progressively incorporating them into TeaCache (Liu et al., 2025a). TeaCache controls the number of cached steps through a feature-difference threshold; empirically, setting this threshold to 0.26 results in approximately 12-NFE across prompts. We therefore use TeaCache with $\delta = 0.26$ as the baseline and compare it with our 12-NFE schedule, ensuring a similar computation budget. We first replace the TeaCache threshold-based schedule with our offline optimized 12-step schedule from Section 3.3 while retaining direct residual reuse, and then further introduce our residual-change prediction and CFG-aligned supervision from Sections 3.1 and 3.2. All settings follow the same evaluation protocol described above.

As shown in Table 3, replacing TeaCache's threshold-based schedule with our schedule improves all three fidelity metrics when cached residuals are directly reused. This intermediate setting evaluates the schedule under direct reuse, although the schedule itself was calibrated with our cache-prediction formulation. Adding our cache prediction yields the complete method and further improves all three metrics.

4.4 Generalization to Unseen Prompts

Since our method relies on offline calibration for predictor fitting and cache-schedule construction, we further evaluate whether the resulting configuration generalizes to prompts outside the calibration and VBench distributions. We use GPT-6 Sol via Codex (OpenAI, 2026) to generate 192 additional text prompts with diverse content, and independently sample one random seed for each prompt. Without any further calibration or schedule adjustment, we directly apply the 12-NFE configuration obtained on the original calibration set to Wan2.1-1.3B. On these unseen prompts, our method achieves an SSIM of **0.733**, PSNR of **21.978**, and LPIPS of **0.191**, closely matching the **0.730/21.543/0.188** obtained on the VBench evaluation set under the same 12-NFE configuration. The comparable performance across the two prompt distributions suggests that the offline-calibrated cache prediction and schedule generalize well beyond the calibration and VBench prompts, rather than overfitting to a particular prompt set.

5 Conclusion

We presented PEACache, a Progressive Error Alignment framework for Diffusion Transformer caching. PEACache controls cache-induced errors at the branch, CFG, and trajectory levels through residual-change prediction, CFG-aligned supervision, and final-latent-aware scheduling. The strong performance across multiple video diffusion backbones validates the effectiveness of this framework. We hope this error-centric perspective can inspire further research on more accurate and efficient diffusion caching.

AI USE STATEMENT

We used generative AI tools to polish the manuscript's language and to review the clarity and consistency of its method and experimental descriptions. We also used GPT-6 Sol via Codex to generate the 192 additional text prompts used in the unseen-prompt evaluation. We take responsibility for the final content of this work, including its text, analyses, and results.

REFERENCES

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

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. In *International Conference on Learning Representations*, volume 2026, pp. 73778–73803, 2026.

Guantao Chen, Shikang Zheng, Yuqi Lin, and Linfeng Zhang. Forecast the principal, stabilize the residual: Subspace-aware feature caching for efficient diffusion transformers. *arXiv preprint arXiv:2601.07396*, 2026.

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. URL https://arxiv.org/abs/2406.01125.

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.

Huanlin Gao, Ping Chen, Fuyuan Shi, Chao Tan, Zhaoxiang Liu, Fang Zhao, Kai Wang, and Shiguo Lian. Lemica: Lexicographic minimax path caching for efficient diffusion-based video generation. *arXiv preprint arXiv:2511.00090*, 2025.

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, et al. Ltx-video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2024.

Yasaman Haghighi and Alexandre Alahi. Sencache: Accelerating diffusion model inference via sensitivity-aware caching. *arXiv preprint arXiv:2602.24208*, 2026.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 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.

Mingkun Lei, Tong Zhao, Liangyu Yuan, and Chi Zhang. Budget-constrained step-level diffusion caching. *arXiv preprint arXiv:2606.13496*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Enhuai Liu, Yunke Wang, Changming Sun, and Chang Xu. D2cache: Second-order delta caching for higher video diffusion acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43589–43599, 2026.

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

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15853–15863. IEEE, 2025b.

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. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2206.00927.

Xudong Lu, Aojun Zhou, Ziyi Lin, Qi Liu, Yuhui Xu, Renrui Zhang, Xue Yang, Junchi Yan, Peng Gao, and Hongsheng Li. TerDiT: Ternary diffusion models with transformers. *arXiv preprint arXiv:2405.14854*, 2024. URL https://arxiv.org/abs/2405.14854.

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. URL https://arxiv.org/abs/2410.19355.

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. URL https://openaccess.thecvf.com/content/CVPR2024/html/Ma_DeepCache_Accelerating_Diffusion_Models_for_Free_CVPR_2024_paper.html.

OpenAI. Gpt-6 sol. https://developers.openai.com/api/docs/models/gpt-6-sol, 2026.

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

Xurui Peng, Chenqian Yan, Hong Liu, Rui Ma, Fangmin Chen, Xing Wang, Zhihua Wu, Songwei Liu, and Mingbao Lin. Ertacache: Error rectification and timesteps adjustment for efficient diffusion. In *International Conference on Learning Representations*, volume 2026, pp. 82104–82120, 2026.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. ieee, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, 2015. doi: 10.1007/978-3-319-24574-4_28.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2202.00512.

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. URL https://arxiv.org/abs/2407.01425.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32211–32252. PMLR, 2023. URL https://proceedings.mlr.press/v202/song23a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. doi: 10.1109/TIP.2003.819861.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6211–6220. IEEE, 2024.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, Jianfei Chen, Ion Stoica, Kurt Keutzer, and Song Han. Sparse VideoGen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025. URL https://arxiv.org/abs/2502.01776.

Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, and Song Han. SANA: Efficient high-resolution image synthesis with linear diffusion transformers. *arXiv preprint arXiv:2410.10629*, 2024. URL https://arxiv.org/abs/2410.10629.

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. In *International Conference on Learning Representations*, volume 2025, pp. 83048–83077, 2025.

Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. SageAttention: Accurate 8-bit attention for plug-and-play inference acceleration. *arXiv preprint arXiv:2410.02367*, 2024. URL https://arxiv.org/abs/2410.02367.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018. doi: 10.1109/CVPR.2018.00068.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. UniPC: A unified predictor-corrector framework for fast sampling of diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2170.

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

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.

A Additional Details

A.1 Temporal Consistency of Residual-Change Subspaces

We analyze the temporal consistency of residual-change subspaces to determine which spectral structures can be reliably transferred across denoising steps. Let $H_c = U_c \Sigma_c V_c^\top$ and $H_t = U_t \Sigma_t V_t^\top$ denote the historical and subsequent residual changes, respectively. Here, U and V correspond to the token-side and channel-side singular subspaces.

For the top-k subspaces, we measure their temporal overlap as

$$\mathrm{Overlap}_k(Q_c, Q_t) = \frac{1}{k} \left\| Q_c^\top Q_t \right\|_F^2, \tag{12}$$

where Q denotes either U or V. To account for the overlap expected between random k-dimensional subspaces, we report the chance-adjusted overlap

$$\mathrm{AdjOverlap}_k = \frac{\mathrm{Overlap}_k - k/d}{1 - k/d}, \tag{13}$$

where d is the corresponding ambient dimension.

As shown in Fig. 4, the channel-side subspace exhibits consistently higher temporal overlap than the token-side subspace, indicating that the channel basis is more persistent across neighboring residual changes. In contrast, token-side overlap varies substantially along the denoising trajectory, showing that its reusability is strongly timestep-dependent. These observations support reusing the more stable channel-side structure while adapting the amount of transferred spectral information to the target timestep.

A.2 Training the Adaptive Rank and Scaling Predictors

We use two lightweight regression models to determine the adaptive spectral support K_t and temporal scaling factor α_t. Both are fitted offline from fully computed calibration trajectories and remain fixed during inference. The rank predictor first estimates a continuous spectral threshold, which is converted into K_t, while the scaling predictor subsequently predicts α_t conditioned on the selected rank through its retained spectral energy.

Adaptive spectral-threshold predictor. For the historical residual motion

$$H_c = R_c - R_o = U \Sigma V^\top, \qquad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r), \tag{14}$$

we define the rank-k transferable motion as

$$B_k = U_k \Sigma_k V_k^\top = H_c V_k V_k^\top = U_k U_k^\top H_c. \tag{15}$$

To construct supervision, we evaluate a coarse ascending grid of candidate ranks,

$$\mathcal{K} = \{k_1, k_2, \ldots, k_J\}, \qquad k_1 < k_2 < \cdots < k_J = K_{\max}, \tag{16}$$

where $K_{\max}$ is the maximum admissible spectral rank of each backbone. We use $K_{\max} = 1536$ for Wan2.1-1.3B (Wan et al., 2025), $K_{\max} = 2048$ for CogVideoX (Yang et al., 2025), and $K_{\max} = 1152$ for Open-Sora 1.2 (Zheng et al., 2024). The candidate ranks increase from low rank to the full available dimension.

For each candidate k, we independently optimize the motion scaling coefficient under our CFG-aligned objective,

$$\alpha_{t,k}^\star = \arg\min_a \left\| \widehat{Y}_t^{\mathrm{cfg}}(k, a) - Y_t^{\mathrm{cfg}} \right\|_F^2, \tag{17}$$

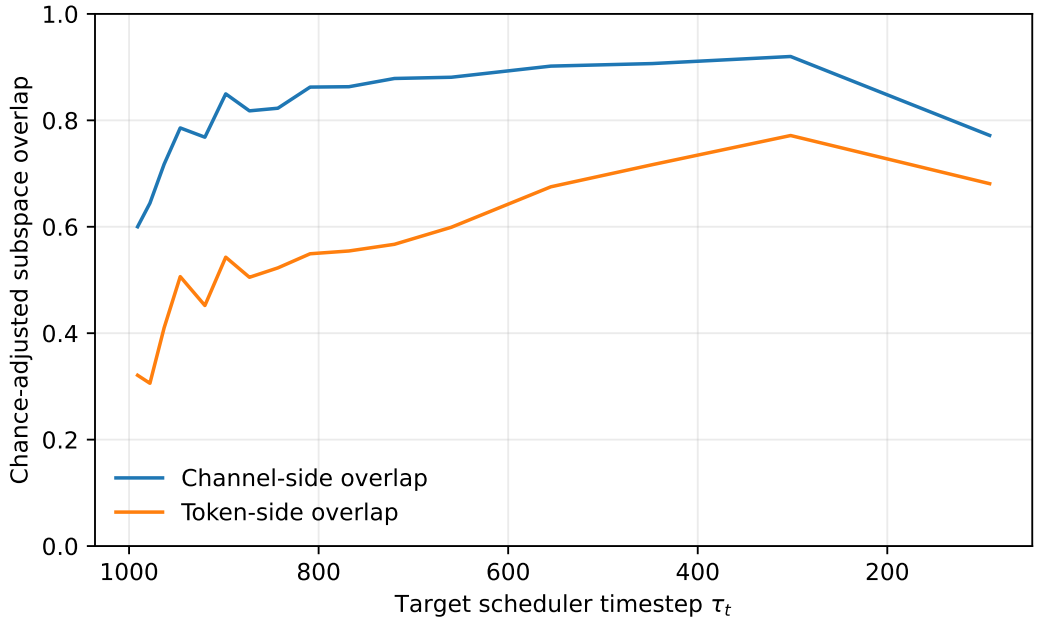

Figure 4: **Temporal consistency of residual-change subspaces.** Chance-adjusted overlap between historical and future residual-change subspaces along the denoising trajectory, averaged across calibration examples and evaluated spectral ranks. The channel-side structure remains more persistent, while token-side reusability varies more with timestep.

where

$$\widehat{Y}_t^{\mathrm{cfg}}(k,a) = \widehat{Y}_t^{\mathrm{c}}(k,a) + (s-1)D_c. \tag{18}$$

We then evaluate the corresponding generation-facing one-step latent error, denoted by $\mathcal{L}_t(k)$.

Rather than supervising the strict minimum-error rank, we use the smallest rank whose error lies within a tolerance of the optimum,

$$K_{\epsilon,t} = \min\left\{k \in \mathcal{K} : \mathcal{L}_t(k) \le \min_{j\in\mathcal{K}} \mathcal{L}_t(j) + \epsilon\right\}. \tag{19}$$

This favors lower ranks when the error–rank curve is nearly flat.

Instead of directly regressing the discrete $K_{\epsilon,t}$, we convert it into an admissible interval for a continuous spectral threshold:

$$\sigma_{K_{\epsilon,t}+1} < \Theta_t \le \sigma_{K_{\epsilon,t}}. \tag{20}$$

When $K_{\epsilon,t} = K_{\max}$, only the upper bound is required.

Concretely, the threshold predictor combines diffusion-time geometry with summary statistics of the singular spectrum of H_c. We define the historical and future timestep gaps as

$$\Delta\tau_{\mathrm{hist}} = \tau_o - \tau_c, \qquad \Delta\tau_{\mathrm{fut}} = \tau_c - \tau_t. \tag{21}$$

Let r denote the number of available singular values of H_c, and define the normalized spectral-energy distribution as

$$p_i = \frac{\sigma_i^2}{\sum_{j=1}^{r}\sigma_j^2}. \tag{22}$$

We characterize the concentration of the spectrum using the entropy-based effective rank

$$r_{\mathrm{eff}} = \exp\left(-\sum_{i:p_i>0} p_i \log p_i\right), \tag{23}$$

and the stable rank

$$r_{\text{stable}} = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sigma_1^2} = \frac{\|H_c\|_F^2}{\sigma_1^2}. \tag{24}$$

We further define K_q, for $q \in \{90, 95, 99\}$, as the smallest rank retaining at least $q\%$ of the total spectral energy:

$$K_q = \min \left\{ k : \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq \frac{q}{100} \right\}. \tag{25}$$

The resulting feature vector is

$$\mathbf{x}_t^K = \left[\frac{\tau_c}{1000}, \frac{\Delta\tau_{\text{hist}}}{1000}, \frac{\Delta\tau_{\text{fut}}}{1000}, \log\sigma_1, \log\|H_c\|_F, \log r_{\text{eff}}, \log r_{\text{stable}}, \frac{K_{90}}{r}, \frac{K_{95}}{r}, \frac{K_{99}}{r} \right]. \tag{26}$$

The first three features encode the current diffusion phase and the temporal geometry between the two preceding full-computation steps and the target step. The remaining features describe the scale, concentration, and cumulative-energy profile of the historical residual-change spectrum.

We expand these features to all first- and second-order polynomial terms, standardize the expanded features using statistics from the calibration set, and predict the log-threshold with a linear regressor:

$$z_t = \log \widehat{\Theta}_t = \beta_0 + \boldsymbol{\beta}^\top \overline{\psi_2(\mathbf{x}_t^K)}, \tag{27}$$

where $\psi_2(\cdot)$ denotes the degree-two polynomial expansion and the bar denotes feature standardization.

The regressor is fitted with an interval loss rather than a point target:

$$\mathcal{L}_\Theta = [\ell_t - z_t]_+^2 + [z_t - u_t]_+^2 + \lambda_K \|\boldsymbol{\beta}\|_2^2, \tag{28}$$

where

$$\ell_t = \log\sigma_{K_{\epsilon,t}+1}, \qquad u_t = \log\sigma_{K_{\epsilon,t}}, \tag{29}$$

and the lower-bound term is omitted at the maximum rank. We use $\lambda_K = 10^{-4}$. Thus, any prediction inside the oracle threshold interval incurs no interval penalty.

At inference time, we exponentiate the predicted log-threshold and count the singular values above it:

$$\widehat{\Theta}_t = \exp(z_t), \qquad K_t = \text{clip}\left(\#\{i : \sigma_i \geq \widehat{\Theta}_t\}, K_{\min}, K_{\max} \right). \tag{30}$$

Hence, although the supervision is obtained from a coarse rank grid, the deployed predictor can select any intermediate rank supported by the spectrum rather than being restricted to the calibration candidates.

Temporal-scaling predictor. After fitting the spectral-threshold predictor, we freeze it and use its predicted K_t to construct supervision for the scaling predictor. This ensures that the scaling model is trained under the same rank selection that will be available at inference time, rather than conditioning on an oracle rank.

For each calibration instance, we obtain the oracle scaling coefficient at the predicted rank as

$$\alpha_t^\star = \arg\min_a \left\| \widehat{Y}_t^{\text{c}}(K_t, a) + (s-1)D_c - Y_t^{\text{cfg}} \right\|_F^2. \tag{31}$$

We characterize the selected spectral support by its retained energy

$$E_{K_t} = \frac{\sum_{i=1}^{K_t} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}. \tag{32}$$

The scaling predictor uses the normalized time geometry together with this retained energy. Let

$$p_c = \frac{\tau_c - \tau_{\min}}{\tau_{\max} - \tau_{\min}}, \tag{33}$$

$$\delta_{\text{hist}} = \frac{\tau_o - \tau_c}{\tau_{\max} - \tau_{\min}}, \qquad \delta_{\text{fut}} = \frac{\tau_c - \tau_t}{\tau_{\max} - \tau_{\min}}, \tag{34}$$

Table 4: Comparison of residual and residual-change subspaces on Wan2.1-1.3B. Higher error capture is better.

Representation	Energy@64	Capture@64	Capture@256	Capture@512
Residual R_c	**0.970**	0.619	0.704	0.741
Residual change H_c	0.791	**0.675**	**0.746**	**0.780**

and

$$r_\tau = \frac{\tau_c - \tau_t}{\tau_o - \tau_c}. \tag{35}$$

Its input is therefore

$$\mathbf{x}_t^\alpha = [p_c,\, \delta_{\text{hist}},\, \delta_{\text{fut}},\, r_\tau,\, E_{K_t}]\,. \tag{36}$$

We again use a degree-two polynomial regressor,

$$\alpha_t = \boldsymbol{\gamma}^\top \psi_2^+(\mathbf{x}_t^\alpha), \tag{37}$$

where $\psi_2^+(\cdot)$ contains an intercept together with all first- and second-order polynomial terms. The coefficients are fitted to the oracle targets $\alpha_t^\star$ with ridge regularization,

$$\min_{\boldsymbol{\gamma}} \sum_t \left(\boldsymbol{\gamma}^\top \psi_2^+(\mathbf{x}_t^\alpha) - \alpha_t^\star\right)^2 + \lambda_\alpha \|\boldsymbol{\gamma}\|_2^2, \tag{38}$$

with $\lambda_\alpha = 10^{-6}$.

The scaling predictor therefore depends only on the diffusion-time geometry and the fraction of historical spectral energy retained by the predicted K_t. At inference, the complete order is to first predict $\widehat{\Theta}_t$, convert it to K_t, compute E_{K_t}, and finally predict α_t. The reconstructed residual is then

$$\widehat{R}_t = R_c + \alpha_t H_c V_{K_t} V_{K_t}^\top. \tag{39}$$

A.3 Comparison with Residual-Space Prediction

Our design differs from SVD-Cache (Chen et al., 2026), which performs spectral decomposition directly in the residual space. To understand which representation is more suitable for correcting cache-reuse error, we compare the dominant subspaces extracted from the cached residual R_c and the historical residual change $H_c = R_c - R_o$ on Wan2.1-1.3B.

While the channel-side subspace is more temporally stable and is therefore suitable for transfer, the token-side structure varies more substantially across timesteps. We therefore use the token-side subspace here as a stricter diagnostic of how well each representation captures the future reuse-induced error.

For each representation $S \in \{R_c, H_c\}$, let $U_k(S)$ contain its top-k left singular vectors, corresponding to the token-side subspace. We measure how much of the true reuse-induced error $H_t = R_t - R_c$ lies in this subspace:

$$\text{Capture}_k(S) = \frac{\left\|U_k(S)U_k(S)^\top H_t\right\|_F^2}{\|H_t\|_F^2}. \tag{40}$$

For reference, we also report the fraction of the source representation's own spectral energy retained by the same top-k subspace:

$$\text{Energy}_k(S) = \frac{\left\|U_k(S)U_k(S)^\top S\right\|_F^2}{\|S\|_F^2} = \frac{\sum_{i=1}^k \sigma_i(S)^2}{\sum_i \sigma_i(S)^2}. \tag{41}$$

Table 4 reveals an important distinction between low-rank structure and error alignment. The residual itself is highly concentrated spectrally: its top 64 components already retain 97.0% of its own energy. However, its dominant token-side subspace captures only 61.9% of the missing residual change. In contrast, although the residual-change representation is less spectrally concentrated, its

Algorithm 1 Dynamic-Programming Cache Scheduler

Require: Number of sampling steps T; calibrated cache costs $d(o, c, t)$; maximum intervals $L_{\text{hist}}, L_{\text{fut}}$
Ensure: Cache schedule $\mathcal{S}_b$ for each feasible cache count b

Initialize $\mathrm{DP}(t, b, o, c) \leftarrow \infty$
$\mathrm{DP}(3, 0, 1, 2) \leftarrow 0$
for $t = 3, \ldots, T$ **do**
  for each reachable state (b, o, c) **do**
    ▷ Full computation at step t
    if $\mathrm{DP}(t, b, o, c) < \mathrm{DP}(t+1, b, c, t)$ **then**
      $\mathrm{DP}(t+1, b, c, t) \leftarrow \mathrm{DP}(t, b, o, c)$
      Store predecessor and action FULL
    end if
    ▷ Cache step t if (o, c, t) is calibrated
    if $1 \le c - o \le L_{\text{hist}}$ **and** $1 \le t - c \le L_{\text{fut}}$ **then**
      $v \leftarrow \mathrm{DP}(t, b, o, c) + d(o, c, t)$
      if $v < \mathrm{DP}(t+1, b+1, o, c)$ **then**
        $\mathrm{DP}(t+1, b+1, o, c) \leftarrow v$
        Store predecessor and action CACHE
      end if
    end if
  end for
end for
$\mathcal{B} \leftarrow \{b : \min_{o<c} \mathrm{DP}(T+1, b, o, c) < \infty\}$
for each $b \in \mathcal{B}$ **do**
  $(o^\star, c^\star) \leftarrow \arg\min_{o<c} \mathrm{DP}(T+1, b, o, c)$
  $\mathcal{S}_b \leftarrow \textsc{Backtrack}(T+1, b, o^\star, c^\star)$
end for
return $\{\mathcal{S}_b\}_{b \in \mathcal{B}}$

dominant token-side subspace captures substantially more of the cache-reuse error, reaching 67.5%, 74.6%, and 78.0% at ranks 64, 256, and 512, respectively.

This distinction is important for cache correction. The dominant components of the raw residual primarily characterize information already present in the cached residual, whereas the residual-change representation is more aligned with the missing update that must be predicted at the target step. The improved error capture of H_c, even on the less temporally stable token side, therefore supports using historical residual changes rather than raw residuals as the representation for cache correction.

A.4 Calibration and Dynamic-Programming Cache Scheduler

We provide the details of the offline calibration and dynamic-programming scheduler used for trajectory-level alignment.

Offline calibration. Since the cache error of a test prompt is unavailable at inference time, we estimate cache costs offline using $M = 20$ calibration prompts. Under our cache formulation, the effect of caching a target step t depends on the two most recent full-computation steps (o, c). We therefore calibrate each valid triplet (o, c, t) satisfying

$$1 \le c - o \le L_{\text{hist}}, \qquad 1 \le t - c \le L_{\text{fut}}, \tag{42}$$

where L_{hist} and L_{fut} denote the maximum historical and future intervals, respectively. We use $(L_{\text{hist}}, L_{\text{fut}}) = (7, 10)$ for Wan2.1-1.3B and CogVideoX, and $(5, 5)$ for Open-Sora 1.2.

Let $d_z(o, c, t)$ denote the measured final-latent error caused by caching step t with anchors (o, c) for calibration prompt z. Because the error scale varies across prompts, we normalize the measurements within each prompt and average them across the calibration set:

$$d(o, c, t) = \frac{1}{M} \sum_{z=1}^{M} \frac{d_z(o, c, t)}{\max_{(o', c', t')} d_z(o', c', t')}. \tag{43}$$

For a cache schedule $\mathcal{S}$, we define the additive surrogate

$$\widetilde{E}(\mathcal{S}) = \sum_{t \in \mathcal{S}} d(o_t, c_t, t), \tag{44}$$

where (o_t, c_t) are the two most recent full-computation steps preceding cached step t.

Dynamic-programming scheduler. We minimize equation 44 under a cache-count budget using dynamic programming. Let $\mathrm{DP}(t, b, o, c)$ denote the minimum accumulated surrogate cost before processing step t, where b steps have been cached and (o, c) are the two most recent full-computation anchors. The first two steps are always fully computed to initialize the historical residual change, giving

$$\mathrm{DP}(3, 0, 1, 2) = 0. \tag{45}$$

At each step, the scheduler considers two transitions. A full-computation action incurs no additional surrogate cost and updates the anchors from (o, c) to (c, t). A cache action keeps (o, c) unchanged and adds the calibrated cost $d(o, c, t)$, provided that the corresponding triplet lies within the calibrated range. Backtracking from the minimum-cost terminal states yields the surrogate-optimal schedule for each feasible cache count. The complete procedure is summarized in Algorithm 1.