CACO: COST-AWARE DYNAMIC CACHING WITH ON-POLICY DISTILLATION FOR ACCELERATING DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers (DiTs) achieve strong performance in high-resolution visual generation, but their iterative denoising requires repeated backbone computation, resulting in substantial inference cost. Feature caching accelerates sampling by reusing intermediate features and skipping backbone computation at selected steps. However, their practical effectiveness is limited by scheduling overhead and the accuracy of approximated features, with existing feature predictors further suffering from train–test mismatch caused by teacher-forcing training. In this paper, we propose CACO, a global–local framework that integrates global cache scheduling with local feature prediction to improve both scheduling efficiency and feature estimation accuracy. Specifically, we propose a Cost-Aware Global Scheduler (CAGS) that first allocates a fixed budget of refresh anchors and dynamically relocates refresh positions through sparse probing while preserving the refresh budget, thereby reducing the overhead of dynamic scheduling. For feature estimation, we present Time-Conditioned Feature Predictor (TCFP), which is first trained with supervised fine-tuning (SFT) on full-computation reference trajectories and subsequently refined with On-Policy Distillation (OPD) on cached trajectories to mitigate train–test mismatch during inference. Extensive experiments on image and video generation demonstrate that CACO substantially reduces inference cost while maintaining generation quality, enabling efficient practical deployment.

1 INTRODUCTION

Diffusion transformers (DiTs) (Peebles & Xie, 2023) have emerged as a dominant architecture for high-resolution image and video generation, owing to their strong capability in modeling long-range dependencies and fine-grained textures Black Forest Labs (2024); Esser et al. (2024); Wan et al. (2025). However, their inference cost grows rapidly with model scale, generation resolution, and the number of sampling steps. Among these, the iterative forward passes through the deep Transformer backbone become a major bottleneck in both end-to-end latency and memory consumption. Recently, feature caching (Ma et al., 2024; Zheng et al., 2026b; Liu et al., 2025a; Chu et al., 2025; Liu et al., 2025b) has emerged as a practical paradigm for DiT acceleration by reusing intermediate representations across adjacent denoising steps to skip some backbone computations. In practice, the backbone computation is performed only at refresh steps selected by scheduling. The intermediate features within refresh computation are reused to estimate the model output in the remaining steps, substantially reducing their inference overhead without modifying the pretrained model.

Existing cache-based acceleration methods (Bhaskar et al., 2025; Liu et al., 2025c; Zou et al., 2026; Peng et al., 2026; Xie et al., 2026; Zheng et al., 2025; Chen et al., 2025; Zhao et al., 2023) mainly focus on two tasks: scheduling and feature reconstruction. Scheduling-based methods, such as TeaCache (Liu et al., 2025a), AdaCache (Kahatapitiya et al., 2025), Dicache (Bu et al., 2026), SmoothCache (Liu et al., 2025d), Sencache (Haghighi & Alahi, 2026) and SeaCache (Chung et al., 2026), determine when to perform a refresh process based on feature variation scores or frequency-domain signals across adjacent steps. Prediction-based methods, including TaylorSeer (Liu et al., 2025b), D2Cache (Jiang et al., 2026), and FoCa (Zheng et al., 2026b), study to reconstruct features at reuse steps through analytical extrapolation or learnable predictors. Despite substantial gains in generation efficiency, existing approaches still confront two fundamental limitations. First, scheduling methods

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.

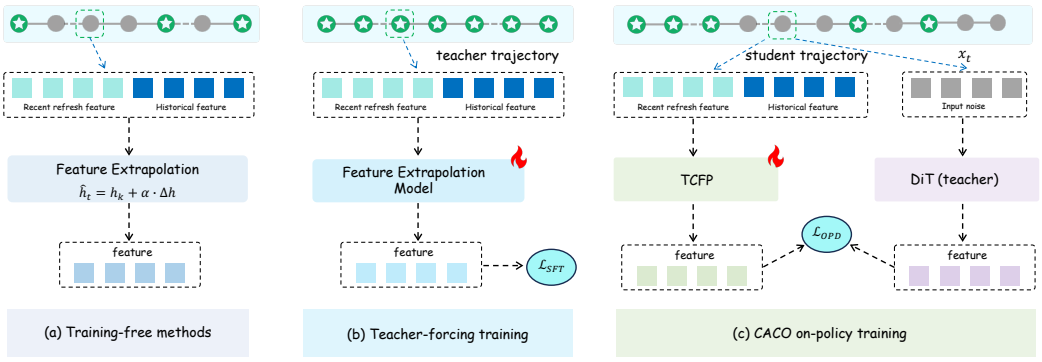


Figure 1: Comparison paradigms for feature reconstruction: training-free extrapolation and teacher-forced training both overlook autoregressive error accumulation during cache rollouts.

often overlook decision overhead, including feature comparison, frequency-domain filtering, and device synchronization, which may adversely impact the speedup in practice. Second, feature approximation errors propagate through denoising processes, altering subsequent inputs, as shown in Figure 1. Althrough some existing methods (Liu et al., 2026; Qiu et al., 2025) fine-tune feature predictors on full-computation trajectories, they still suffer from train–test mismatch between reference states and altered states during cached inference. These limitations reveal an intrinsic problem of how to achieve low-overhead scheduling while reducing error accumulation during feature reuse.

To address these challenges, we propose Cost-Aware Dynamic Caching with On-Policy Distillation (CACO) for the DiT acceleration based on the comprehensive exploration of global scheduling with local feature prediction. Specifically, we introduce Cost-Aware Global Scheduler (CAGS), a training-free scheduler that allocates a fixed budget of refresh anchors along the denoising trajectory and uses sparse probing to adaptively relocate them. By advancing a refresh while canceling its original position, CAGS improves refresh utility while accounting for scheduling overhead. Based on the refresh trajectory established by CAGS, we further introduce Time-Conditioned Feature Predictor (TCFP) to estimate features at reuse steps from the latest refresh state. TCFP is first initialized through supervised fine-tuning (SFT) on full-model reference trajectories. It is then further refined through On-Policy Distillation (OPD) on student-induced cached trajectories, with a frozen teacher providing supervision for both local feature accuracy and trajectory fidelity. This progressive training strategy enables the predictor to better adapt to error accumulation during cache rollouts.

The main contributions are summarized as follows:

1. We propose CACO, a global–local diffusion caching framework that decomposes cache acceleration into global scheduling and local feature prediction, with CAGS allocating refresh steps and TCFP reconstructing features at reuse steps.
2. We introduce a two-stage progressive learning strategy for TCFP, where SFT learns from CAGS-aligned rollouts and OPD further adapts the predictor to student-induced cache states, mitigating train–test mismatch and error accumulation.
3. Extensive experiments on visual generation demonstrate that CACO substantially reduces DiT inference cost while maintaining generation quality comparable to full denoising.

2 Related Work

2.1 Cache Scheduling for Diffusion Acceleration

Diffusion caching methods mainly focus on determining which denoising steps should perform full-model computation. Early methods such as FORA (Selvaraju et al., 2024) and Δ-DiT (Chen et al., 2024) typically use fixed cache intervals or predefined computation patterns. Later approaches, including TeaCache (Liu et al., 2025a), AdaCache (Kahatapitiya et al., 2025), and SeaCache (Chung et al., 2026), dynamically schedule refreshes based on feature or residual variations during inference, adapting across samples and denoising stages. SeaCache (Chung et al., 2026) further uses

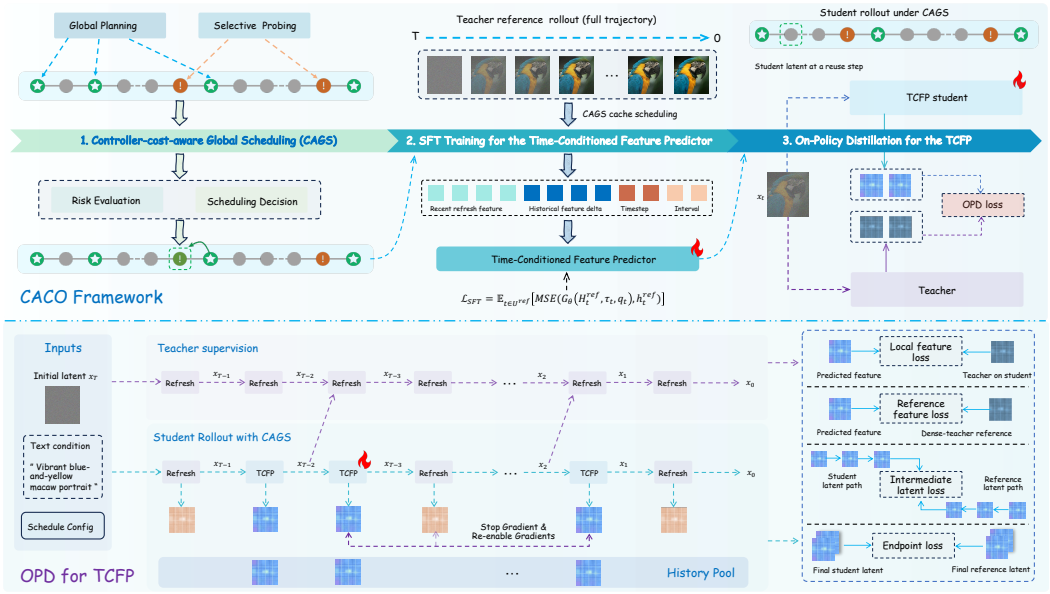


Figure 2: Overview of the proposed CACO framework. CACO combines global scheduling and local feature prediction for efficient diffusion transformer inference. CAGS adaptively allocates refresh steps under a fixed budget, while TCFP predicts features at reuse steps based on the latest refresh feature and reuse history. TCFP is first trained with SFT and further refined with OPD on student cache rollouts, using local and trajectory-level supervision to improve feature prediction.

spectrum-aware filtering to measure inter-step variation more reliably. Recent methods explicitly incorporate computation budgets: BudCache (Lei et al., 2026) globally optimizes refresh-step selection under a fixed number of function evaluations (NFE), while BAG (Zhao et al., 2026) combines the remaining budget and local trajectory feedback through a learned gate for budget-aware online scheduling. However, existing methods mainly make per-step scheduling decisions without separating global computation allocation from local feature reconstruction. Their adaptive decisions also introduce additional costs for feature processing or gating that are not fully reflected by NFE, reducing practical speedup. CACO addresses these issues by separating global scheduling from local feature reconstruction: the scheduler allocates and adjusts refresh steps under a fixed budget with low observation overhead, while a lightweight predictor reconstructs features at reuse steps.

2.2 Feature Reconstruction for Cached Diffusion

Feature reconstruction methods aim to approximate intermediate representations at denoising steps where full Transformer computation is skipped. Early training-free methods mainly reuse historical representations directly: DeepCache (Ma et al., 2024), FORA, and Δ-DiT exploit feature redundancy across neighboring denoising steps, while FasterCache (Lyu et al., 2025) develops dynamic feature reuse for video diffusion. Recent training-free approaches move from direct reuse to explicit feature prediction. TaylorSeer (Liu et al., 2025b) extrapolates future representations from higher-order trends in historical features; FoCa (Zheng et al., 2026b) models feature evolution as an ordinary differential equation (ODE) with a *forecast-then-calibrate* strategy; HyCa (Zheng et al., 2026a) applies different prediction schemes to heterogeneous feature dynamics; and SpeCa uses *forecast-then-verify* to validate predictions. By modeling feature evolution, these methods improve approximation accuracy at reuse steps without additional training.

Learnable approaches further improve local feature reconstruction. LinCa (Liu et al., 2026) uses learnable feature decomposition and prediction to improve local reconstruction. However, predictors trained on full-model reference histories can face a state-distribution mismatch: during cached inference, prediction errors propagate through latent updates and alter subsequent features and scheduling observations. CACO addresses this mismatch with On-Policy Distillation (OPD), supervising the predictor on its own cached trajectories to improve approximation accuracy under the states and refresh schedules encountered at inference.

3 METHOD

3.1 GLOBAL–LOCAL CACHING FRAMEWORK

Overview. CACO couples training-free global scheduling with learned local feature prediction for DiTs, as illustrated in Figure 2. At the global level, CAGS places refresh anchors $\{b_1, \ldots, b_B\}$ along the denoising trajectory under a fixed budget B, while keeping controller overhead small relative to the resulting speedup. Based on the refresh schedule established by CAGS, TCFP performs local feature prediction through a lightweight predictor G_ϕ, which maps the most recent refresh feature, the reuse length ℓ_t, and the denoising progress p_t to an approximate feature $\widehat{\mathbf{h}}_t$.

Problem Formulation. Given a diffusion process with T steps, the input latent at step t is defined as $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1$, where $t \in 0, \ldots, T-1$, and $\mathbf{x}_0$ and $\mathbf{x}_1$ denote pure noise and the clean image, respectively. Let F denote the Transformer backbone, P the output projection, and S_t the scheduler at step t. The complete denoising process is given by

$$\mathbf{h}_t = F(\mathbf{x}_t, \mathbf{c}, \tau_t), \qquad \mathbf{x}_{t+1} = S_t(\mathbf{x}_t, P(\mathbf{h}_t)), \tag{1}$$

where $\mathbf{h}_t$ is the pre-projection feature, $\mathbf{c}$ is the condition, and τ_t is the timestep embedding. We seek to design a cache-based acceleration framework that preserves the generation trajectory and quality of the full model under a refresh budget K. Specifically, the framework performs a full backbone forward only at refresh steps $\mathcal{R}$, with $|\mathcal{R}| \leq K$, and approximates the backbone output at reuse steps $\mathcal{U} = \{0, \ldots, T-1\} \setminus \mathcal{R}$. This formulation gives rise to two coupled problems: how to allocate full computation across the denoising trajectory, and how to estimate the features at reuse steps while controlling the approximation error and associated computational cost.

In this paper, we study these problems in a global-local paradigm, where CAGS determines the refresh opportunity for full-model computation and TCFP leverages a lightweight predictor G_θ to estimate the skipped features from historical refresh and temporal conditions.

Coupled Global–Local Inference. In the CACO framework, CAGS and TCFP jointly advance the generation trajectory by selecting appropriate refresh steps $\mathcal{R}$ and estimating precise hidden feature

$$\widehat{\mathbf{h}}_t = \begin{cases} F(\widehat{\mathbf{x}}_t, \mathbf{c}, \tau_t), & t \in \mathcal{R}, \\ G_\theta(\mathcal{H}_t, \tau_t), & t \in \mathcal{U}, \end{cases} \qquad \widehat{\mathbf{x}}_{t+1} = S_t\Big(\widehat{\mathbf{x}}_t, P(\widehat{\mathbf{h}}_t)\Big), \tag{2}$$

where $\widehat{x}_t$ is the latent in approximated trajectory and $\widehat{x}_0 = x_0$. $\mathcal{H}_t$ is a set of historical feature from preceding refresh steps. Starting from a global set of refresh steps $\mathcal{R}$ and initial feature of refresh step, TCFP estimates the local features of reuse steps, according to which the CAGS adaptively updates the refresh step set. These two modules iteratively refine the refresh schedule and reuse features, allowing global scheduling and local prediction to adapt to each other along the generation trajectory. Accordingly, the practical inference cost of the cache-based acceleration consists of three components: refresh computation, reuse estimation, and scheduling overhead:

$$C_{\text{total}} = C_{\text{refresh}} + C_{\text{reuse}} + C_{\text{scheduling}}. \tag{3}$$

3.2 COST-AWARE GLOBAL SCHEDULER

The traditional step scheduling process includes state extraction, frequency filtering, and device synchronization. *Cost-Aware Global Scheduler* (CAGS) updates this process with global anchors and local sparse probes, leading to adaptive refresh relocation with affordable scheduling cost.

Global Anchor Placement. The CAGS pre-defines the initial anchors for generation model by fitting the distribution of refresh steps via existing cache-based acceleration. Denoting the initial set of anchored steps $\mathcal{A} = \{a_0, \ldots, a_{K-1}\}$ with $|\mathcal{A}| = K$ and $2 \leq K \leq T$. The endpoints $a_0 = 0$ and $a_{K-1} = T-1$ are fixed to stabilize the dynamic relocation process.

Reuse Risk and Relocation. Given the anchor steps a_j, $j \in \{1, \ldots, K-2\}$, some preceding steps are considered as alternative candidates for the investigation and relocation of new refresh step to a_j. Denoting $p_j \in \{a_j - l, \ldots, a_j - 1\}$ the candidates to be probed and l is a pre-defined probing interval, CAGS compares modulated $\mathbf{m}_{p_j}$ from the cached state $\widehat{\mathbf{x}}_{p_j}$ with $\mathbf{m}_{a_j}$ stored at the latest

actual refresh a_j. In this way, we apply the SEA filter Φ_{σ_p} introduced in SeaCache (Chung et al., 2026) on $\mathbf{m}_{p_j}$ and $\mathbf{m}_{a_j}$ to estimate the discrepancy d_p between steps p_j and a_j

$$d_p = \frac{\text{mean}\left|\Phi_{\sigma_p}(\mathbf{m}_{p_j}) - \Phi_{\sigma_p}(\mathbf{m}_{a_j})\right|}{\text{mean}\left|\Phi_{\sigma_p}(\mathbf{m}_{a_j})\right| + \epsilon}, \tag{4}$$

where ϵ is a small number to ensure the numerical stability. Therefore, the risk of keeping current step p_j as reuse step, denoted by $\widehat{d}_a$, is obtained as follow

$$\widehat{d}_{p_j} = d_p \frac{a_j - p_j}{a_j - a_{j-1}}. \tag{5}$$

If $\widehat{d}_{p_j}$ exceeds the threshold δ, CAGS moves the refresh from a_j to p_j and switches to refresh process. Otherwise, p_j remains a reuse step. As the evaluation is ranged over a small interval around anchors, the relocation cost is smaller than that of fully scheduling, *e.g.,* SeaCache. In our implementation, we find that $l = 1$ achieves a comprising trade-off between scheduling efficiency and performance.

3.3 On-Policy Distillation for Local Feature Prediction

In the reuse steps, we introduce a lightweight time-conditioned predictor to estimate the local features. The predictor is progressively trained through an SFT stage followed by an OPD stage, with the goal of mitigating the train-test mismatch between training and deployment.

3.3.1 Time-Conditioned Feature Predictor and Supervised Initialization

Lightweight Prediction from Refresh History. We instantiate G_θ as a *Time-Conditioned Feature Predictor* (TCFP). Given two adjacent refresh steps p and r with $p < r$, let $\widehat{\mathbf{h}}_p = F(\widehat{\mathbf{x}}_p, \mathbf{c}, \tau_p)$ denote the refreshed feature at step p. The feature variation between the two refresh steps is then given by $\mathbf{\Delta}_r = \widehat{\mathbf{h}}_r - \widehat{\mathbf{h}}_p$. For a reuse step $t > r$, the feature estimation can be formulated as

$$\widehat{\mathbf{h}}_t = \widehat{\mathbf{h}}_r + \boldsymbol{\alpha}_{\theta,t} \odot \mathbf{\Delta}_r, \qquad \boldsymbol{\alpha}_{\theta,t} = G_\theta\Big((\widehat{\mathbf{h}}_r, \mathbf{\Delta}_r, k_t), \tau_t\Big), \tag{6}$$

where $k_t = t - r$. Accordingly, the feature estimation $G_\theta(\mathcal{H}_t, \tau_t) = G_\theta\Big((\widehat{\mathbf{h}}_r, \mathbf{\Delta}_r, k_t), \tau_t\Big)$ is modeled with temporal conditions τ_t and k_t. When there is only one single refresh step, we set $\mathbf{\Delta}_r = \mathbf{0}$ as the latest-feature reuse. Although the historical features are computed by the backbone, their corresponding student inputs still reflect errors accumulated from preceding reuse steps.

Supervised Initialization. Supervised fine-tuning (SFT) replays CAGS on full-model reference trajectories to pair reuse-step targets $\mathbf{h}_t^{\text{ref}}$, $t \in \mathcal{U}^{\text{ref}}$, with prior refresh histories $\mathcal{H}_t^{\text{ref}}$, minimizing

$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_{t \in \mathcal{U}^{\text{ref}}}\left[\text{MSE}\big(G_\theta(\mathcal{H}_t^{\text{ref}}, \tau_t, q_t), \mathbf{h}_t^{\text{ref}}\big)\right]. \tag{7}$$

3.3.2 On-Policy Distillation on Student-Induced Cache Trajectories

From Teacher-Forcing to On-Policy Supervision. SFT trains G_θ with teacher-forced reference trajectories, while feature caching during inference introduces approximation errors that deviate the generated trajectory from the full-model reference, resulting in train–test mismatch. To address this mismatch, OPD initializes from the SFT-trained G_θ, freezes F and P, and applies fixed CAGS rules to generate student cache trajectories $\mathcal{T}_\theta$ under the same initial noise, text condition, and scheduler configuration. CAGS maintains $|\mathcal{R}| = K$, while student-induced states determine subsequent refresh positions and features. OPD then provides frozen-teacher supervision on these student-induced states, thereby capturing inference-time error propagation. Here, *on-policy* refers to student-induced states rather than learned scheduling or policy gradients.

Online Distillation Supervision. For each reuse step $t \in \mathcal{U}$ along the student rollout, we obtain local and reference targets:

$$\mathbf{h}_t^{\text{loc}} = \text{sg}[F(\widehat{\mathbf{x}}_t, \mathbf{c}, \tau_t)], \qquad \mathbf{h}_t^{\text{ref}} = \text{sg}\big[F(\mathbf{x}_t^{\text{ref}}, \mathbf{c}, \tau_t)\big], \tag{8}$$

Table 1: Quantitative comparison on FLUX.1-dev with DrawBench. Best and second-best scores are **bold** and underlined, respectively. † denotes the official BudCache implementation.

Method	NFE	Latency (s)	Speed	TFLOPs	TFLOPs Spd.	PSNR↑	SSIM↑	LPIPS↓
Original	50.0	8.05	1.00×	2976	1.00×	$+\infty$	1	0
25 steps	25.0	4.04	1.99×	1487	2.00×	17.543	0.740	0.302
15 steps	15.0	2.51	3.21×	892	3.34×	15.357	0.663	0.414
$\sim 2.3\times$ *acceleration*								
BudCache†	21.0	3.47	2.32×	1250	2.38×	28.02	0.910	0.078
MagCache ($\alpha = 0.07$)	22.0	3.63	2.22×	1310	2.27×	27.28	0.905	0.085
SeaCache ($\delta = 0.3$)	20.9	3.67	2.19×	1242	2.40×	26.98	0.899	0.090
TaylorSeer	20.0	3.85	2.09×	1191	2.50×	23.30	0.856	0.134
LinCa (N=2)	26.0	4.44	1.81×	1554	1.92×	22.44	0.839	0.153
TeaCache ($\delta = 0.38$)	21.0	3.50	2.30×	1248	2.38×	18.53	0.744	0.275
CACO ($\delta = 0.3$)	21.0	3.51	2.29×	1249	2.38×	**32.28**	**0.937**	**0.048**
$\sim 3.3\times$ *acceleration*								
BudCache†	15.0	2.52	3.19×	893	3.33×	22.14	0.820	0.181
MagCache ($\alpha = 0.3$)	15.0	2.52	3.19×	893	3.33×	22.83	0.826	0.172
SeaCache ($\delta = 0.55$)	14.0	2.49	3.23×	834	3.57×	22.78	0.823	0.180
TaylorSeer	10.0	2.41	3.34×	596	4.99×	17.35	0.688	0.323
LinCa ($N = 4$)	14.0	2.62	3.08×	839	3.55×	18.62	0.733	0.281
TeaCache ($\delta = 0.72$)	14.0	2.41	3.34×	836	3.56×	16.80	0.688	0.357
CACO ($\delta = 0.6$)	14.0	2.43	3.31×	833	3.57×	**25.99**	**0.862**	**0.123**

where sg denotes stop-gradient. The local target is queried at the current student state to supervise the instantaneous prediction error, while the reference target is obtained from an independent full-model rollout to constrain accumulated trajectory drift. These two targets provide complementary supervision: the local target adapts to student-induced states, whereas the reference target anchors the prediction to the full-model trajectory. Teacher targets are used only for loss computation and never replace predictions, enter the cache, or reset latents. The additional teacher queries are performed only during training and do not count as student refreshes or affect the inference budget.

Local and Trajectory-Level Supervision. The OPD objective jointly promotes local prediction accuracy and consistency with the full-model trajectory through four complementary supervision terms. The local feature loss is evaluated on student-induced states along the current rollout, enabling the predictor to adapt to inference-time approximation errors. In contrast, the reference feature loss aligns predicted features with the corresponding full-model features, while the latent-state loss constrains deviations in intermediate states and the terminal loss preserves the final denoising state. Together, these terms provide supervision at both the feature and trajectory levels, balancing adaptation to student-induced states with fidelity to the full-model reference. These terms are combined into the following OPD objective:

$$\mathcal{L}_{\text{local}} = \mathbb{E}_{t\in\mathcal{U}}\left[\text{MSE}\left(\widehat{\mathbf{h}}_t, \mathbf{h}_t^{\text{loc}}\right)\right], \qquad \mathcal{L}_{\text{ref-feat}} = \mathbb{E}_{t\in\mathcal{U}}\left[\text{MSE}\left(\widehat{\mathbf{h}}_t, \mathbf{h}_t^{\text{ref}}\right)\right], \tag{9}$$

$$\mathcal{L}_{\text{ref-lat}} = \mathbb{E}_{t\in\mathcal{U}}\left[\text{MSE}\left(\widehat{\mathbf{x}}_{t+1}, \mathbf{x}_{t+1}^{\text{ref}}\right)\right], \qquad \mathcal{L}_{\text{end}} = \text{MSE}\left(\widehat{\mathbf{x}}_T, \mathbf{x}_T^{\text{ref}}\right), \tag{10}$$

$$\mathcal{L}_{\text{OPD}} = \lambda_{\text{local}}\mathcal{L}_{\text{local}} + \lambda_{\text{feat}}\mathcal{L}_{\text{ref-feat}} + \lambda_{\text{lat}}\mathcal{L}_{\text{ref-lat}} + \lambda_{\text{end}}\mathcal{L}_{\text{end}}, \tag{11}$$

where all loss weights are set to 1.0 by default.

On-Policy Rollouts with Truncated Backpropagation. During OPD training, we generate complete student rollouts to capture the actual error propagation induced by feature caching, while restricting gradient computation to a terminal suffix of each trajectory. The suffix is initialized from the detached prefix state, and gradients propagate through the projection, scheduler, and frozen refresh backbone to the preceding predictions within this window. Teacher targets remain detached, and CAGS decisions are treated as non-differentiable. Only G_θ is updated, after which fresh student trajectories are generated. More details are provided in Appendix C.

Table 2: Quantitative comparison on Wan 2.1 and HunyuanVideo with VBench. Best and second-best quality scores within each group are **bold** and underlined, respectively.

	HunyuanVideo					Wan 2.1 1.3B				
Method	Latency (s)	Speedup	PSNR ↑	LPIPS ↓	SSIM ↑	Latency (s)	Speedup	PSNR ↑	LPIPS ↓	SSIM ↑
Original	78.8	1.00×	+∞	0	1	56.1	1.00×	+∞	0	1
25 steps	42.1	1.87×	20.12	0.255	0.717	28.6	1.96×	14.78	0.379	0.583
15 steps	27.4	2.88×	17.25	0.375	0.633	18.4	3.05×	13.28	0.476	0.511
TeaCache	41.3	1.91×	23.25	0.179	0.814	32.3	1.74×	23.42	0.117	0.833
TaylorSeer	43.3	1.82×	23.12	0.167	0.821	33.9	1.65×	18.51	0.239	0.719
SenCache	39.2	2.01×	29.56	0.072	0.921	32.8	1.71×	<u>29.70</u>	<u>0.042</u>	<u>0.934</u>
DiCache	41.3	1.91×	30.89	0.061	<u>0.925</u>	29.5	1.90×	29.21	0.047	0.929
SeaCache	38.4	2.05×	<u>31.25</u>	<u>0.053</u>	0.924	28.5	1.97×	29.10	0.054	0.924
CACO	**36.5**	**2.16×**	**35.34**	**0.028**	**0.957**	**27.8**	**2.02×**	**30.69**	**0.027**	**0.948**
TeaCache	31.6	2.49×	20.62	0.254	0.729	22.9	2.45×	19.10	0.251	0.712
TaylorSeer	29.1	2.71×	19.92	0.273	0.717	22.6	2.48×	14.65	0.429	0.569
SenCache	30.8	2.50×	<u>26.80</u>	<u>0.092</u>	<u>0.841</u>	24.3	2.31×	<u>23.69</u>	<u>0.096</u>	<u>0.848</u>
DiCache	28.8	2.73×	25.78	0.112	0.823	24.1	2.33×	21.68	0.145	0.804
SeaCache	28.3	2.78×	26.62	0.101	0.824	20.2	2.78×	23.57	0.101	0.842
CACO	**27.7**	**2.84×**	**29.23**	**0.071**	**0.909**	**19.9**	**2.82×**	**24.52**	**0.089**	**0.868**

4 EXPERIMENTS

Experimental Setting. To comprehensively evaluate our method, we compare against representative open-source generative models, including FLUX.1-dev (Black Forest Labs, 2024) for image generation and Wan2.1 (Wan et al., 2025) and HunyuanVideo (Kong et al., 2024) for video generation. We use the full DrawBench (Saharia et al., 2022) benchmark with 200 prompts for image generation and the complete VBench Huang et al. (2024) evaluation set with 946 prompts for video generation. For each benchmark, all methods are evaluated using identical prompts and a consistent evaluation protocol. Under comparable inference acceleration ratios, we assess reconstruction quality using PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018).

4.1 EXPERIMENTAL RESULTS

Results on Image Generation. As shown in Table 1, CACO achieves the best PSNR, SSIM, and LPIPS on FLUX.1-dev with DrawBench under both acceleration regimes. At $2.3\times$ acceleration, CACO improves PSNR by 5.30,dB and reduces LPIPS by 46.7% compared with SeaCache, while reducing latency from 3.67 s to 3.51 s with comparable NFE. At $3.3\times$ acceleration, CACO further improves PSNR by 3.16 dB and reduces LPIPS by 28.5% compared with MagCache, which achieves the highest fidelity among methods at this acceleration level, while reducing latency from 2.52 s to 2.43 s. These results show that CACO maintains its advantage under the higher acceleration setting. As illustrated in Figure 3, qualitative comparisons further show that CACO preserves finer details more faithfully to the original model at both 50% and 30% refresh ratios, without noticeable visual quality degradation. Notably, compared with the 15-step sampling baseline, CACO achieves a similar speedup ($3.31\times$ vs. $3.21\times$), while improving PSNR from 24.51 dB to 32.28 dB and reducing LPIPS from 0.414 to 0.123. This indicates that simply reducing the number of sampling steps cannot replace the combination of refresh and feature prediction used by CACO.

Results on Video Generation. Table 2 summarizes the results on Wan2.1 and HunyuanVideo under two acceleration regimes of approximately $2.0\times$ and $2.8\times$. CACO achieves the best PSNR, SSIM, and LPIPS scores, as well as the lowest latency, across both models and acceleration regimes. At approximately $2\times$ acceleration, CACO achieves measured speedups of $2.16\times$ and $2.02\times$ on HunyuanVideo and Wan2.1, respectively. Compared with SeaCache, CACO improves PSNR by 4.09 dB and 1.59 dB, and reduces LPIPS by 47.2% and 50.0%, respectively, without increasing latency. At approximately $2.8\times$ acceleration, CACO further achieves speedups of $2.84\times$ and $2.82\times$, with additional PSNR improvements of 2.61 dB and 0.95 dB over SeaCache on HunyuanVideo and Wan2.1,

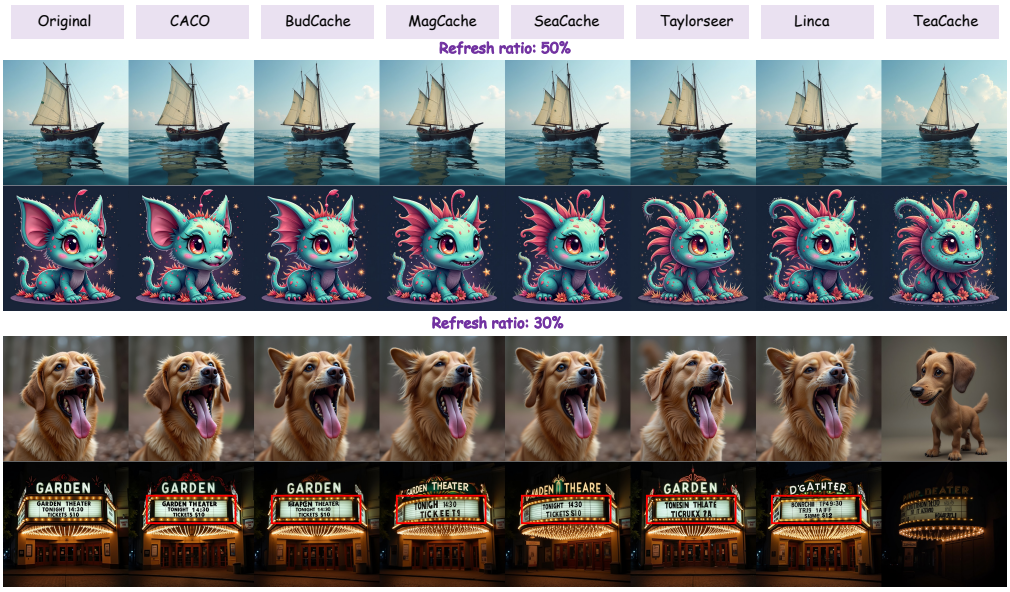


Figure 3: Visual comparison of different methods on FLUX.1-dev with DrawBench.

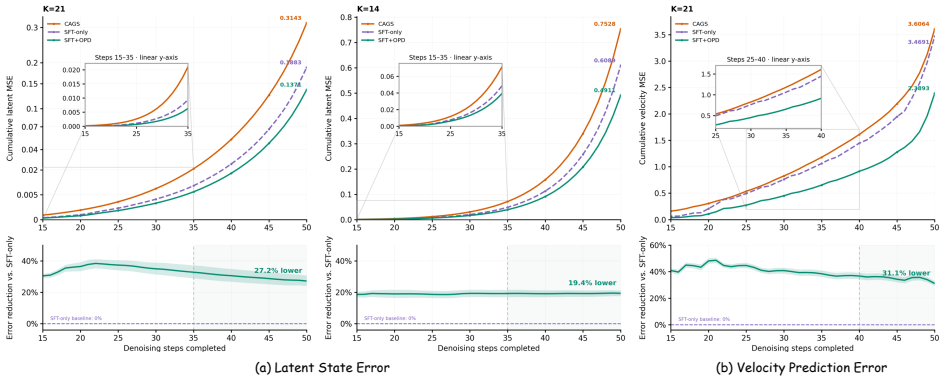


Figure 4: Quantitative ablation of TCFP training strategies on FLUX.1-dev across different refresh ratios, showing prediction errors during the denoising process.

respectively. In addition, CACO maintains its advantage under the higher acceleration setting, consistent with the results on image generation. When NFE is comparable, the latency gap between CACO and the compared methods is also more pronounced for video generation. This observation reflects the higher costs of decoding and frequent scheduler operations in video models, making the reduction in controller overhead provided by CAGS more beneficial for video generation.

4.2 Ablation Study

We conduct ablations on FLUX.1-dev to analyze the contributions of proposed components. All ablations are performed under the same refresh ratio to ensure comparable NFE and latency.

Analysis of Component Contributions. Table 3 compares the effects of progressively adding SFT and OPD in terms of PSNR, SSIM, and LPIPS. CAGS alone already outperforms SeaCache, improving PSNR from 26.98 to 28.75 under comparable NFE and latency. Adding SFT further improves generation quality, while OPD provides additional gains by adapting to student-induced states. With both components enabled, CACO achieves the best performance across all three metrics. Figure 4 compares CAGS, SFT-only, and SFT+OPD under a fixed refresh budget. SFT consistently reduces prediction error over direct feature reuse in CAGS, indicating that the time-conditioned predictor learns local feature evolution at reuse steps. OPD further reduces the error, with larger gains at later denoising steps, reflecting its ability to mitigate error accumulation along long reuse chains. Under

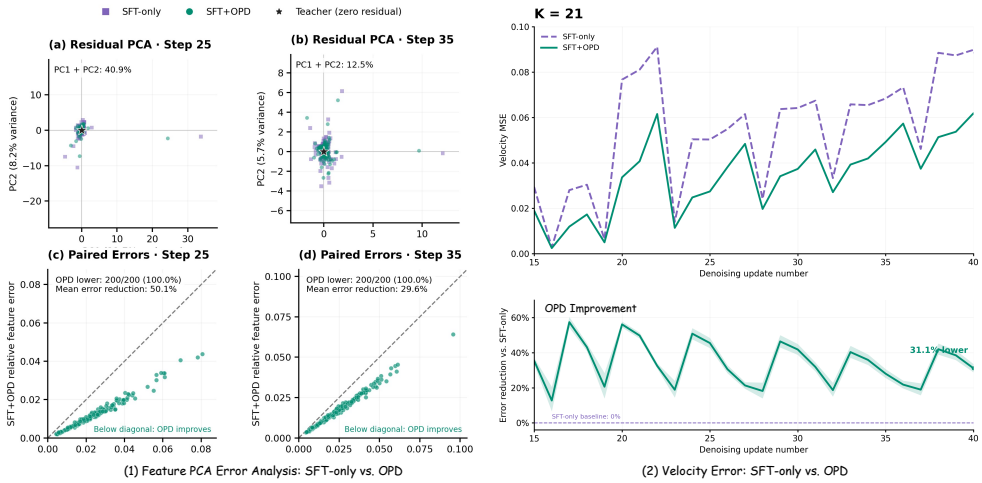


Figure 5: Quantitative analysis of the impact of OPD on feature residuals and velocity estimation.

Table 3: Ablation study of each component on FLUX.1-dev. The best scores are highlighted in **bold**.

Method	NFE	Latency (s)	Speedup	PSNR↑	SSIM↑	LPIPS↓
Original	50.0	8.1	1.00×	$+\infty$	1	0
CAGS	21.0	3.5	2.31×	28.75	0.911	0.0783
CAGS + SFT	21.0	3.6	2.31×	30.53	0.926	0.0577
CAGS + OPD	21.0	3.6	2.25×	31.50	0.931	0.0492
CAGS + SFT + OPD	21.0	3.6	2.25×	**32.28**	**0.937**	**0.0484**

$K = 21$, SFT+OPD further reduces the final cumulative latent error by 27.2% and cumulative velocity prediction error by 31.1% relative to SFT-only. Under the higher compression $K = 14$ setting, it still reduces cumulative latent error by 19.4%. These results demonstrate that OPD consistently mitigates error amplification caused by train–test mismatch across different compression levels.

Analysis of OPD at Sample and Trajectory Levels. Figure 5 provides complementary evidence at both the sample and trajectory levels. The PCA projections of feature residuals show that, at denoising steps 25 and 35, SFT+OPD produces residuals that are more concentrated around the teacher's zero-residual reference than SFT-only, indicating that OPD effectively shifts the predicted features toward the desired feature space rather than merely making small numerical adjustments. The paired error plots further show that OPD reduces the relative feature error at both steps, with mean reductions of 50.1% and 29.6%, respectively, without increasing the error for any sample. For velocity MSE, SFT+OPD consistently achieves lower error than SFT-only throughout the entire denoising process, retaining a 31.1% reduction at the final update. Overall, the ablation results demonstrate that SFT and OPD address complementary sources of prediction error: SFT learns feature evolution along the reference teacher trajectory, while OPD adapts the predictor to states encountered along its own cached trajectories. Combining the two training stages therefore substantially reduces approximation errors at both the sample and trajectory levels.

5 Conclusion

In this paper, we presented CACO, a global–local framework for efficient Diffusion Transformer inference through cost-aware cache scheduling and local feature prediction. CACO introduces CAGS, which adaptively allocates and relocates refresh steps under a fixed budget while explicitly accounting for scheduling overhead. Based on the resulting refresh trajectory, TCFP reconstructs features at reuse steps and is progressively trained with SFT and OPD to better adapt to student-induced cached states and mitigate error accumulation. Extensive experiments on image and video generation demonstrate that CACO substantially reduces inference cost while maintaining generation quality close to full denoising. These results highlight the effectiveness of jointly optimizing global refresh allocation and local feature reconstruction for practical diffusion acceleration.

REFERENCES

Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. Cache me if you can: How many kvs do you need for effective long-context lms? *arXiv preprint arXiv:2506.17121*, 2025.

Black Forest Labs. FLUX. Official model and inference-code release, 2024. URL `https://github.com/black-forest-labs/flux`.

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.

Guanjie Chen, Xinyu Zhao, Yucheng Zhou, Xiaoye Qu, Tianlong Chen, and Yu Cheng. Towards stabilized and efficient diffusion transformers through long-skip-connections with spectral constraints. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17708–17718. IEEE, 2025.

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.

Huanpeng Chu, Wei Wu, Guanyu Feng, and Yutao Zhang. Omnicache: A trajectory-oriented global perspective on training-free cache reuse for diffusion transformer models. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 16302–16312. IEEE, 2025.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. Seacache: Spectral-evolution-aware cache for accelerating diffusion models. *arXiv preprint arXiv:2602.18993*, 2026.

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.

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

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

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, et al. d^2 cache: Accelerating diffusion-based llms via dual adaptive caching. In *International Conference on Learning Representations*, volume 2026, pp. 137511–137531, 2026.

Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S Ryoo, and Tian Xie. Adaptive caching for faster video generation with diffusion transformers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15240–15252. IEEE, 2025.

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

Mingkun Lei, Tong Zhao, Liangyu Yuan, and Chi Zhang. Budget-constrained step-level diffusion caching. *arXiv preprint arXiv:2606.13496*, 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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7353–7363. IEEE, 2025a.

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.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Fei Ren, Shaobo Wang, Kaixin Li, and Linfeng Zhang. Speca: Accelerating diffusion transformers with speculative feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10024–10033, 2025c.

Jinshan Liu, Haoran Qin, Xiaobing Tu, Jiahui Hu, Yukun Xie, Kerui Shen, Jinkui Ren, Yuqi Lin, Xiantao Zhang, and Linfeng Zhang. Linca: Accelerating diffusion models via learnable decomposed feature caching. In *European Conference on Computer Vision*, pp. 463. Springer Nature, 2026.

Joseph Liu, Joshua Geddes, Ziyu Guo, Haomiao Jiang, and Mahesh Kumar Nandwana. Smooth cache: A universal inference acceleration technique for diffusion transformers. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3220–3229. IEEE, 2025d.

Zhengyao Lyu, 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. In *International Conference on Learning Representations*, volume 2025, pp. 33132–33156, 2025.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15762–15772. IEEE, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. 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.

Junxiang Qiu, Lin Liu, Shuo Wang, Jinda Lu, Kezhou Chen, and Yanbin Hao. Accelerating diffusion transformer via gradient-optimized cache. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17608–17617. IEEE, 2025.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

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.

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.

Zhikang Xie, Xichen Ye, Yifan Wu, Haoshen Yu, Peizhu Gong, Weizhong Zhang, Cheng Jin, et al. Onlinecache: Learning dynamic caching policies with error correction for efficient diffusion inference. *arXiv preprint arXiv:2607.29398*, 2026.

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/CVF Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

Tong Zhao, Mingkun Lei, Yucheng Han, and Chi Zhang. Bag: Budget-aware gating for diffusion caching. *arXiv preprint arXiv:2608.09231*, 2026.

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, pp. 49842–49869, 2023.

Shikang Zheng, Guantao Chen, Qinming Zhou, Yuqi Lin, Lixuan He, Chang Zou, Peiliang Cai, Jiacheng Liu, and Linfeng Zhang. Let features decide their own solvers: Hybrid feature caching for diffusion transformers. In *International Conference on Learning Representations*, volume 2026, pp. 62767–62790, 2026a.

Shikang Zheng, Liang Feng, Xinyu Wang, Qinming Zhou, Peiliang Cai, Chang Zou, Jiacheng Liu, Yuqi Lin, Junjie Chen, Yue Ma, et al. Forecast then calibrate: Feature caching as ode for efficient diffusion transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 13449–13457, 2026b.

Zhixin Zheng, Xinyu Wang, Chang Zou, Shaobo Wang, and Linfeng Zhang. Compute only 16 tokens in one timestep: Accelerating diffusion transformers with cluster-driven feature caching. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10181–10189, 2025.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking token-wise feature caching: Accelerating diffusion transformers with dual feature caching. *IEEE Transactions on Image Processing*, 2026.

A AI USE STATEMENT

Generative AI tools were used to assist with language editing and translation of the manuscript, as well as the development of simple utility code and tests. The authors reviewed all AI-assisted text and inspected all AI-assisted code, incorporating these outputs into the manuscript or Git repository only after checking them and finding no issues. The research design, interpretation of results, mathematical reasoning, and proofs were carried out by the authors without generative AI assistance. The authors take full responsibility for the final manuscript, code, claims, and results.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we describe the CACO framework and its training and inference procedures in Section 3. Appendix B details the architecture, input and output dimensions, conditioning mechanisms, initialization, and history-update rules of the Time-Conditioned Feature Predictor. The SFT and OPD objectives, rollout procedures, gradient-flow rules, and optimization settings are provided in Appendix C.

B ARCHITECTURE AND IMPLEMENTATION OF THE TIME-CONDITIONED FEATURE PREDICTOR

This section details the architecture, inputs and outputs, initialization, and history management of the Time-Conditioned Feature Predictor (TCFP) in CACO. Given features from the two most recent full backbone evaluations, the current sampling time, and the reuse interval, TCFP predicts an extrapolation gain for each sample and feature channel. These gains reconstruct the features at the current step, which are then processed by the original model's output head.

We describe the feature-space implementation of TCFP. FLUX.1-dev and Wan2.1-T2V-1.3B use the same predictor architecture, differing only in their feature dimensions: $D = 3072$ and $D = 1536$, respectively.

B.1 REFRESH HISTORY AND INPUT REPRESENTATION

Let $p < r < t$ denote the two most recent full-computation (*refresh*) steps and the current feature-reuse (*reuse*) step. These discrete indices increase in sampling execution order and are distinct from the raw timesteps returned by the scheduler. Let B be the batch size, N the number of tokens per sample, and D the feature dimension. The two refresh features and their difference are

$$H_p, H_r \in \mathbb{R}^{B\times N\times D}, \qquad \Delta H_r = H_r - H_p. \tag{12}$$

The discrete interval since the latest refresh is

$$k_t = t - r, \qquad k_t \geq 1. \tag{13}$$

Here, k_t is the current reuse interval and should be distinguished from the refresh budget K for the entire sampling trajectory.

Both H_p and H_r are obtained by evaluating the full backbone at the corresponding *student-trajectory latents*. They are therefore backbone-computed features rather than cached predictor outputs. Nevertheless, their inputs may already reflect trajectory deviations introduced by earlier reuse steps; these features should not be interpreted as features on an error-free teacher trajectory.

Instead of flattening all token features into a multilayer perceptron (MLP), TCFP separately averages the latest features and their difference over the token dimension:

$$\overline{H}_r = \frac{1}{N}\sum_{n=1}^{N} H_r[:, n, :], \qquad \overline{\Delta H}_r = \frac{1}{N}\sum_{n=1}^{N} \Delta H_r[:, n, :], \tag{14}$$

where $\overline{H}_r, \overline{\Delta H}_r \in \mathbb{R}^{B\times D}$. Two independent content branches encode these pooled inputs and combine them with time and interval conditioning to predict the extrapolation gains.

B.2 Channel-Wise Gains and Feature Reconstruction

Given the pooled history, normalized timestep τ_t, and reuse interval k_t, the gain predictor outputs

$$\alpha_{\theta,t} = f_\theta\big(\overline{H}_r, \overline{\Delta H}_r, \tau_t, k_t\big) \in \mathbb{R}^{B\times D}. \tag{15}$$

The gains are broadcast over the token dimension to reconstruct the current features:

$$\widehat{H}_t = H_r + \alpha_{\theta,t}[:, \mathrm{None}, :] \odot \Delta H_r, \tag{16}$$

where $\odot$ denotes element-wise multiplication. Equivalently, for sample b, token n, and channel d,

$$\widehat{H}_t[b,n,d] = H_r[b,n,d] + \alpha_{\theta,t}[b,d]\,\Delta H_r[b,n,d]. \tag{17}$$

Thus, TCFP predicts one gain per sample and channel and shares it across all tokens within that sample. Mean pooling is used only to estimate the gains and does not change the token layout of the reconstructed features. Each token retains its own historical direction and magnitude of feature change.

Table 4: Inputs, outputs, and tensor shapes of TCFP.

Tensor	Shape	Description
H_p, H_r	$B \times N \times D$	Features from the two latest actual refreshes
ΔH_r	$B \times N \times D$	Feature difference between these refreshes
$\overline{H}_r, \overline{\Delta H}_r$	$B \times D$	Inputs to the two content branches
τ_t	B	Normalized time condition
k_t	B	Discrete steps since the latest refresh
$\alpha_{\theta,t}$	$B \times D$	Per-sample, per-channel extrapolation gains
$\alpha_{\theta,t}[:, \mathrm{None}, :]$	$B \times 1 \times D$	Gains broadcast over the token dimension
$\widehat{H}_t$	$B \times N \times D$	Reconstructed features for the original output head

When all samples share the same sampling step and reuse interval, scalar inputs for τ_t and k_t may be broadcast across the batch. Table 4 summarizes the tensor shapes.

The gain prediction head ends with a linear layer, without a sigmoid, softmax, or clipping operation. Consequently, $\alpha_{\theta,t}$ may be negative or exceed 1. It specifies a channel-wise adjustment along the historical feature-change direction, rather than a refresh probability or a convex-combination weight. TCFP neither predicts independent gains for individual tokens nor directly outputs diffusion latents or velocities.

B.3 Time Conditioning and Dual-Branch Architecture

Time and interval encoding. Let s_t denote the raw timestep returned by the scheduler at the current sampling step. We normalize it as

$$\tau_t = \begin{cases} s_t/1000, & \text{FLUX.1-dev}, \\ s_t/T_{\mathrm{train}}, & \text{Wan2.1-T2V-1.3B}, \end{cases} \tag{18}$$

where T_{train} is the video scheduler's `num_train_timesteps`. We first apply a 64-dimensional sinusoidal encoding to τ_t. For $j = 0, \ldots, 31$, define

$$\omega_j = \exp\left(-\frac{j \log 10000}{31}\right), \qquad a_j = 1000\tau_t\omega_j. \tag{19}$$

The time encoding is

$$\mathrm{TE}_{64}(\tau_t) = [\sin(a_0), \ldots, \sin(a_{31}), \cos(a_0), \ldots, \cos(a_{31})]\,. \tag{20}$$

This encoding has no learnable parameters. A time MLP then maps it to a 96-dimensional representation:

$$e_t = \mathrm{MLP}_{\mathrm{time}}(\mathrm{TE}_{64}(\tau_t)) \in \mathbb{R}^{B\times 96}. \tag{21}$$

The reuse interval is encoded by a learnable embedding table:

$$e_k = \mathrm{Emb}_{256\times 32}(\min(k_t, 255)) \in \mathbb{R}^{B\times 32}. \tag{22}$$

The time condition τ_t describes the current sampling position, whereas k_t specifies the discrete distance from the latest refresh. They are provided as separate predictor inputs.

Content encoding and gain prediction. Two content branches separately encode the latest features and their historical difference. Independent bias-free linear projections first map the pooled inputs to 96-dimensional representations:

$$c_H = P_H(\overline{H}_r), \qquad c_\Delta = P_\Delta(\overline{\Delta H}_r), \qquad c_H, c_\Delta \in \mathbb{R}^{B\times 96}. \tag{23}$$

Each content representation is concatenated with the same time representation e_t and interval representation e_k:

$$z_H = \mathrm{MLP}_H([c_H, e_t, e_k]), \qquad z_\Delta = \mathrm{MLP}_\Delta([c_\Delta, e_t, e_k]), \tag{24}$$

where $[\cdot]$ denotes concatenation along the last dimension. Each branch has input dimension $96 + 96 + 32 = 224$ and output dimension 128. The two branch outputs are concatenated and passed to the prediction head:

$$\alpha_{\theta,t} = \mathrm{MLP}_{\mathrm{head}}([z_H, z_\Delta]). \tag{25}$$

The content projections and their subsequent MLPs have separate parameters. Only the time MLP and interval embedding are shared across the two branches.

Table 5: Layer-by-layer architecture of TCFP. Linear layers include biases unless marked as bias-free.

Module	Layer sequence	Output shape
Sinusoidal time encoding	TE_{64} (parameter-free)	$B \times 64$
Time MLP	$\mathrm{Linear}(64, 96) \to \mathrm{SiLU}$ $\to \mathrm{Linear}(96, 96)$	$B \times 96$
Interval embedding	$\mathrm{Embedding}(256, 32)$	$B \times 32$
Latest-feature projection P_H	$\mathrm{Linear}(D, 96)$, bias-free	$B \times 96$
Feature-difference projection P_Δ	$\mathrm{Linear}(D, 96)$, bias-free	$B \times 96$
Latest-feature branch MLP_H	$\mathrm{Linear}(224, 128) \to \mathrm{SiLU}$ $\to \mathrm{Linear}(128, 128)$	$B \times 128$
Feature-difference branch MLP_Δ	$\mathrm{Linear}(224, 128) \to \mathrm{SiLU}$ $\to \mathrm{Linear}(128, 128)$	$B \times 128$
Joint prediction head	$\mathrm{Linear}(256, 96) \to \mathrm{SiLU}$ $\to \mathrm{Linear}(96, D)$	$B \times D$

Table 5 gives the complete layer structure. Except for the two content projections, all linear layers include biases. No activation follows the final layer of either content-branch MLP or the joint prediction head.

B.4 Initialization and Transition Between Training Stages

TCFP is initialized to produce a constant channel gain of 1. Specifically, we zero the weights of both content projections and the final-layer weights of both content-branch MLPs. The prediction head's final-layer weights are set to zero and its bias to one. All other parameters retain the implementation's default initialization.

Because the prediction head's final weights are zero and its bias is one, the untrained predictor satisfies

$$\alpha_{\theta_0,t} = \mathbf{1}_{B\times D} \tag{26}$$

for every input. The resulting initial feature prediction is

$$\widehat{H}_t = H_r + \Delta H_r = 2H_r - H_p. \tag{27}$$

This initialization *repeats the feature increment between the two latest refreshes*. It does not explicitly scale that increment by the ratio of the current extrapolation interval to the historical refresh interval, and hence is generally distinct from time-scaled linear extrapolation. For example, linear extrapolation in discrete step indices uses gain $(t - r)/(r - p)$, which equals 1 only when the two intervals are equal.

The final-layer biases of the two content-branch MLPs are not all set to zero. Accordingly, this scheme should not be described as zero-initializing the entire predictor. The constant-gain property applies only before training: SFT learns input-dependent channel gains, and OPD loads the SFT checkpoint and continues optimization without repeating the constant-gain initialization.

B.5 History Updates and Boundary Cases

The predictor history is updated only by actual refresh operations. A new feature is written to history only after a full backbone evaluation at the current student latent. Reuse predictions $\widehat{H}_t$ are never written back as refresh features, and teacher features computed solely for training supervision are also excluded from this history.

Consecutive reuse steps therefore use the same H_r and ΔH_r defined by the two latest actual refreshes, rather than recursively treating the previous prediction as a new extrapolation anchor. Each reuse step obtains its own gains through its time condition τ_t and reuse interval k_t.

When only one valid refresh is available, we set

$$H_p = H_r, \qquad \Delta H_r = 0. \tag{28}$$

Feature reconstruction then reduces to

$$\widehat{H}_t = H_r, \tag{29}$$

regardless of the predicted gains, directly reusing the latest features. If the history is empty, the prediction path cannot be used; a full backbone evaluation must first establish an initial state. TCFP uses only the two latest valid refreshes and does not access older history entries.

B.6 Parameter Count and Computational Cost

The architecture in Table 5 has

$$\begin{aligned} P_{\mathrm{TCFP}}(D) = {} & \underbrace{192D}_{\text{content projections}} + \underbrace{15\,552}_{\text{time MLP}} + \underbrace{8\,192}_{\text{interval embedding}} \\ & + \underbrace{90\,624}_{\text{content branches}} + \underbrace{24\,672 + 97D}_{\text{prediction head}} \\ = {} & 289D + 139\,040 \end{aligned} \tag{30}$$

parameters. Table 6 reports the counts for the two backbone configurations.

Table 6: TCFP parameter counts, including weights and biases. The backbone, original output head, and training-only auxiliary networks are excluded.

Backbone	Feature dimension D	TCFP parameters
FLUX.1-dev	3072	1,026,848
HunyuanVideo	3072	1,026,848
Wan2.1-T2V-1.3B	1536	582,944

Because content encoding follows token-wise mean pooling, neither the parameter count nor the post-pooling MLP computation grows with the number of tokens N. However, computing feature differences, pooling, and reconstructing features still operate on full $B \times N \times D$ tensors and require $\mathcal{O}(BND)$ computation. Independence of the parameter count from N therefore does not imply that the full prediction path has token-independent computation or memory cost.

C OPD Training Details

Rollout Initialization and Teacher Targets. We initialize G_θ from SFT and keep the backbone F, output projection P, and CAGS control rules fixed. For each training example, the student and full-model reference use the same initial noise, text condition, and sampler configuration. The reference

Algorithm 1 OPD training (on-policy distillation)

Given: frozen backbone F, projection P, scheduler S
conditions $\mathcal{D}$, refresh budget K, steps T, gradient horizon H
loss weights $\boldsymbol{\lambda}$ and supervision sets $\mathcal{V}_{\text{feat}}, \mathcal{V}_{\text{lat}}$
Initialize: $\theta \leftarrow \theta_{\text{SFT}}$; fix CAGS rules
while training **do**
  sample $(\mathbf{c}, \mathbf{x}_0) \sim \mathcal{D}$; load or run dense reference $\mathcal{T}^{\text{ref}}$
  $\widehat{\mathbf{x}}_0 \leftarrow \mathbf{x}_0$; $\mathcal{H} \leftarrow \varnothing$; $\mathcal{U} \leftarrow \varnothing$
  reset scheduler, CAGS, loss records, and optimizer gradients
  choose suffix start $b \in [\max(0, T-H), T-1]$ ▷ configured rule
  for $t = 0, \dots, T-1$ **do**
    record gradients iff $t \geq b$; detach prefix state at $t = b$
    $d_t \leftarrow \text{CAGS}(\widehat{\mathbf{x}}_t; K)$ ▷ discrete decision
    if $d_t =$ refresh **then**
      $\widetilde{\mathbf{h}}_t \leftarrow F(\widehat{\mathbf{x}}_t, \mathbf{c}, \tau_t)$ ▷ input gradients in suffix
      update $\mathcal{H}$ with $(t, \widetilde{\mathbf{h}}_t)$ ▷ refresh history only
    else
      $\widehat{\mathbf{h}}_t \leftarrow G_\theta(\mathcal{H}, \tau_t, q_t)$; $\widetilde{\mathbf{h}}_t \leftarrow \widehat{\mathbf{h}}_t$
      $\mathbf{h}_t^{\text{loc}} \leftarrow \text{sg}[F(\widehat{\mathbf{x}}_t, \mathbf{c}, \tau_t)]$; $\mathcal{U} \leftarrow \mathcal{U} \cup \{t\}$
    end if
    $\widehat{\mathbf{x}}_{t+1} \leftarrow S_t(\widehat{\mathbf{x}}_t, P(\widetilde{\mathbf{h}}_t))$ ▷ student output
    retain local and selected reference loss records
  end for
  $\mathcal{L}_{\text{OPD}} \leftarrow \lambda_{\text{local}}\mathcal{L}_{\text{local}} + \lambda_{\text{feat}}\mathcal{L}_{\text{ref-feat}} + \lambda_{\text{lat}}\mathcal{L}_{\text{ref-lat}} + \lambda_{\text{end}}\mathcal{L}_{\text{end}}$
  if $\mathcal{L}_{\text{OPD}}$ is disconnected from θ: **continue**
  backpropagate $\mathcal{L}_{\text{OPD}}$ + enabled regularizers; clip gradients; update θ
end while
Return: predictor G_θ

executes the full backbone at every step; its features $\mathbf{h}_t^{\text{ref}}$ and latents $\mathbf{x}_t^{\text{ref}}$ may be precomputed and are treated as detached targets. Each student rollout starts with an empty refresh history and reset sampler and controller states. CAGS preserves the refresh budget K, but determines the actual refresh/reuse positions from the evolving student states rather than replaying the reference schedule used during SFT.

At actual reuse steps, the frozen backbone is queried at the current student input to obtain $\mathbf{h}_t^{\text{loc}}$ as defined in Eq. 8. Unlike the reference targets, these local targets must be obtained anew for the current rollout. Only student refresh features enter $\mathcal{H}_t$; reuse predictions drive latent updates without being stored as history anchors. Neither local nor reference targets replace student predictions, enter the cache, or reset the student's state. Additional teacher queries are training-only supervision operations, not student refreshes, and therefore do not alter the inference budget.

Loss Configuration and Temporal Weighting. The objective in Eq. 11 combines local supervision with reference feature, intermediate-latent, and terminal constraints. Reference feature and intermediate-latent terms can be disabled by setting their weights to zero. In particular, terminal supervision can serve as the sole trajectory constraint while retaining the local teacher loss; this does not reduce OPD to terminal-only supervision. Disabled terms and terms with empty supervision sets are defined as zero. The sets $\mathcal{V}_{\text{feat}}$ and $\mathcal{V}_{\text{lat}}$ select the positions at which reference constraints are evaluated. Selected positions contribute uniformly by default; the formulation also permits normalized temporal weights to emphasize later deviations. These choices affect loss aggregation, not student state generation: reference quantities remain comparison targets, and local supervision continues to use the actual student inputs.

Gradient Window. Given an integer maximum backpropagation horizon $H \geq 1$, we retain the computation graph for a terminal suffix

$$\mathcal{W}_b = \{b, \dots, T-1\}, \qquad b \in \{\max(0, T-H), \dots, T-1\}. \tag{31}$$

The prefix executes the same refresh/reuse process as deployment without retaining a computation graph. Its latent, refresh history, and controller state initialize the suffix, with differentiable quanti-

ties detached at the boundary. No reference state is substituted at this boundary, so the suffix retains the effects of all preceding cache decisions and prediction errors. The truncation therefore limits gradient propagation rather than the generation of on-policy training states.

Gradient Paths and Loss Evaluation. Losses may be evaluated throughout the rollout, but prefix losses do not contribute gradients. Within the suffix, losses backpropagate through the output projection, sampler updates, and subsequent state transitions. Although the backbone parameters are frozen, student refresh outputs and the features written to history retain their derivatives with respect to input latents. Disabling these paths would prevent later losses from supervising earlier reuse predictions. In contrast, additional teacher targets are always detached, and no gradient passes through discrete CAGS decisions. Only losses connected to predictor evaluations within the suffix contribute to its update.

Predictor Updates. We estimate the task objective in Eq. 11 using the current student rollouts, backpropagate through the retained graph, apply gradient clipping, and update only θ. If the task loss is disconnected from the predictor, the update is skipped. Each subsequent iteration resets the student cache and controller state and generates fresh trajectories and local teacher targets using the updated predictor, while keeping the backbone, output projection, refresh budget, and CAGS control rules fixed.

OPD Training Iteration. The following pseudocode summarizes one iteration. Its inputs are the current predictor, paired sampling conditions, a fixed refresh budget, a maximum gradient horizon H, and the chosen supervision sets and loss weights. SFT initialization is performed once before these iterations, rather than repeated at each update.

1. **Prepare reference targets.** Obtain the detached full-model reference trajectory for the current noise, condition, and sampler configuration, reusing precomputed reference quantities when available.

2. **Initialize the student.** Set $\widehat{\mathbf{x}}_0 = \mathbf{x}_0^{\mathrm{ref}}$, clear the refresh history, and reset the sampler and CAGS states. Set the suffix start b subject to Eq. 31, and initialize loss accumulators.

3. **For $t = 0, \ldots, T-1$, advance the student.** Execute steps $t < b$ without retaining a computation graph. At $t = b$, detach the prefix outputs and carry the resulting latent, history, and controller state into the differentiable suffix. At each step:

   (a) Apply CAGS to the current student state to decide refresh or reuse, without differentiating through this decision.

   (b) If refresh, compute $\widetilde{\mathbf{h}}_t = F(\widehat{\mathbf{x}}_t, \mathbf{c}, \tau_t)$ and store the feature and its sampling position in the history. Within the suffix, retain its input-latent derivatives despite the frozen backbone parameters.

   (c) Otherwise, compute the reuse gap $k_t = t - r_t$ and predict $\widehat{\mathbf{h}}_t = G_\theta(\mathcal{H}_t, \tau_t, k_t)$, set $\widetilde{\mathbf{h}}_t = \widehat{\mathbf{h}}_t$, and query the detached local teacher target at $\widehat{\mathbf{x}}_t$. Accumulate local and enabled reference-feature losses without writing the prediction or teacher target into the history.

   (d) Advance the latent using $\widehat{\mathbf{x}}_{t+1} = S_t(\widehat{\mathbf{x}}_t, P(\widetilde{\mathbf{h}}_t))$. Accumulate enabled intermediate-latent losses at the selected positions. Teacher targets never replace this student update.

4. **Aggregate losses.** Evaluate the terminal loss and aggregate enabled terms using the selected averaging scheme and Eq. 11. Prefix losses may be recorded, but only losses connected to suffix predictor evaluations contribute gradients.

5. **Update the predictor.** If the task loss is disconnected from G_θ, skip the update. Otherwise, backpropagate the task objective through the retained graph, clip gradients, and update only θ. Keep all pretrained parameters and CAGS control rules unchanged.

6. **Repeat with the updated predictor.** Start the next iteration with reset student states and generate a new cache trajectory and corresponding local teacher targets.

D Additional Experiments

D.1 OPD Loss Ablation

We ablate the OPD loss components on DrawBench200 using FLUX.1-dev with CAGS and a refresh budget of $K = 21$. Starting from the same SFT checkpoint, we train each variant for the same number of steps and remove one loss term at a time. Table 7 compares the full objective with these variants.

Table 7: OPD loss ablation on DrawBench200 with FLUX and CAGS $K = 21$. All variants are trained for the same number of steps from the same SFT checkpoint.

Training objective	PSNR ↑	SSIM ↑	LPIPS ↓
Full objective	32.28	0.937	0.048
w/o local feature loss	32.04	0.933	0.058
w/o reference feature loss	31.79	0.931	0.062
w/o reference latent loss	32.12	0.935	0.053
w/o endpoint loss	32.15	0.935	0.053
w/o innovation regularization	32.02	0.933	0.059

Effect of individual loss components. The full objective achieves the best PSNR, SSIM, and LPIPS. Removing the reference feature loss yields the largest degradation across all three metrics: PSNR decreases by 0.49 dB, SSIM drops from 0.937 to 0.931, and LPIPS increases from 0.048 to 0.062. This suggests that reference-feature supervision is particularly important for preserving fidelity to the original model's trajectory in this setting.

Removing the local feature loss reduces PSNR by 0.24 dB and increases LPIPS to 0.058, supporting the value of teacher supervision at student-induced states. The reference latent and endpoint losses provide additional benefits: omitting them reduces PSNR by 0.16 dB and 0.13 dB, respectively, with LPIPS increasing to 0.053 in both cases. Removing innovation regularization likewise lowers PSNR to 32.02 dB and increases LPIPS to 0.059. Together, these ablations support retaining all five components in the evaluated configuration, with feature supervision, trajectory constraints, and auxiliary regularization contributing to the final reconstruction quality.

D.2 Cumulative Error Across Denoising Stages

Figure 6 provides a closer view of cumulative latent mean squared error (MSE) for SeaCache, SFT-only, and SFT+OPD under refresh budgets $K = 21$ and $K = 15$. We divide the displayed trajectory into three windows to make differences visible at early, middle, and late denoising stages. Each panel shows the original cumulative error rather than an error recomputed within that window.

Across both budgets, SFT-only reduces cumulative error relative to SeaCache, and SFT+OPD provides a further reduction. The separation is visible in the early window and becomes larger in absolute magnitude toward the end of denoising. All three methods incur greater error with the smaller refresh budget, while SFT+OPD retains the lowest cumulative error. These trends support the complementary roles of supervised initialization and on-policy refinement: SFT improves feature prediction, and OPD further limits error accumulation along student-induced trajectories. Because the panels use different vertical scales, cross-window comparisons should account for the axis values rather than visual slope alone.

D.3 Qualitative Comparison

Figure 7 compares CACO with other caching methods on FLUX.1-dev under two acceleration settings, using refresh ratios of 50% and 30%. Across both settings, CACO produces images that closely match the original model outputs, with no apparent loss of visual quality in the illustrated examples. At a 50% refresh ratio, CACO preserves the statue's pose, the adjacent object's appearance, and the arrangement of the people in the kitchen. At the more aggressive 30% setting, it retains the dogs' appearance and relative positions, as well as the woman's facial expression and

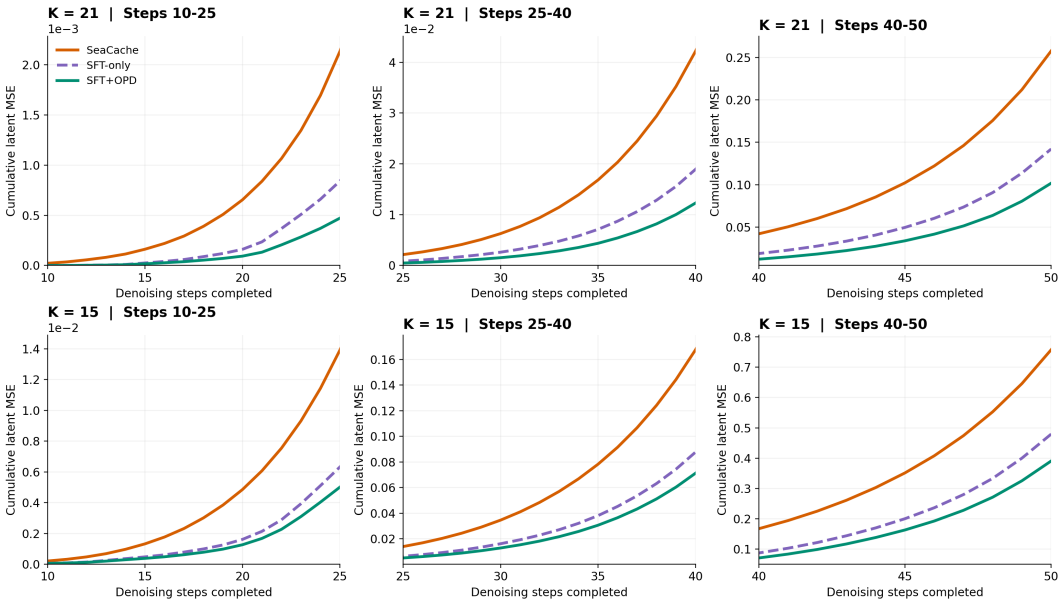


Figure 6: Cumulative latent error across denoising stages. The top and bottom rows use refresh budgets $K = 21$ and $K = 15$, respectively; columns show steps 10–25, 25–40, and 40–50. Curves report mean cumulative latent MSE for SeaCache, SFT-only, and SFT+OPD. Each panel uses an independent linear y-axis range; cumulative errors are not reset at window boundaries. Lower is better.

the arrangement of the books. Competing methods exhibit more visible changes in some of these structures and details. These examples suggest that CACO maintains greater fidelity to the original outputs across different acceleration settings, preserving both overall composition and local visual details.

D.4 Qualitative Comparison on Video Generation Models

Figure 8 compares frames from the original and accelerated outputs of Wan2.1 and HunyuanVideo at refresh ratios of 30% and 50%, respectively. Across both models, CACO closely preserves the original scene composition, subject appearance, and local structures. The highlighted regions illustrate that its outputs remain closer to the original frames than those of competing methods in the presented examples, supporting its applicability to multiple video generation backbones.

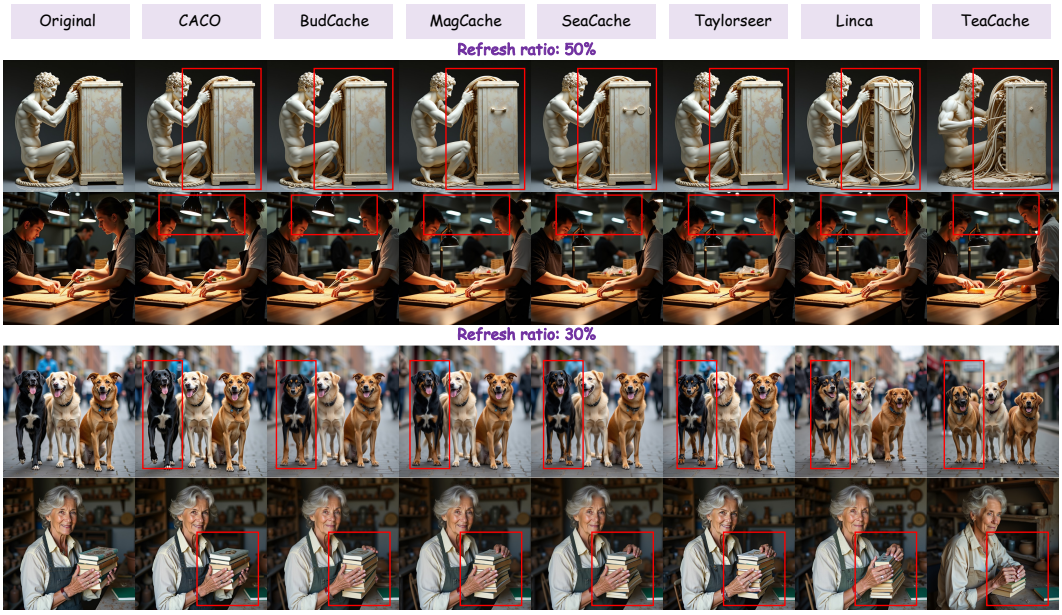


Figure 7: Qualitative comparison on FLUX.1-dev at refresh ratios of 50% and 30%. CACO closely preserves the original model outputs in both settings.

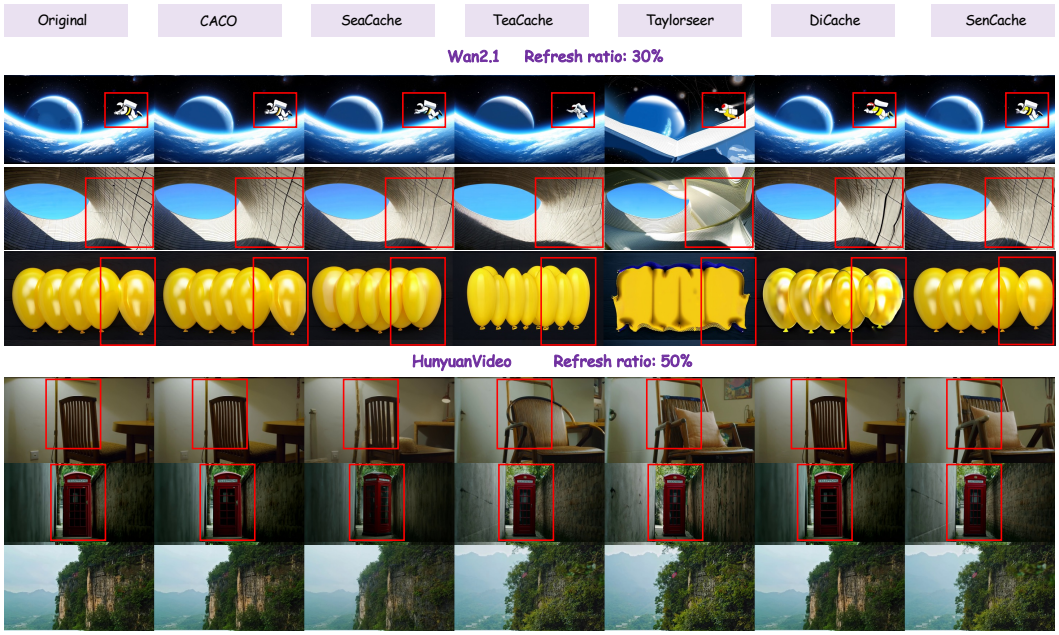


Figure 8: Qualitative comparison on Wan2.1 and HunyuanVideo at refresh ratios of 30% and 50%, respectively. Red boxes highlight local differences. CACO closely preserves the original outputs across both video generation models.