RECACHE: LEARNING BUDGET-AWARE CACHING SCHEDULES FOR DIFFUSION MODELS VIA REINFORCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern diffusion models generate high-quality images and videos, but their iterative denoising process makes inference expensive. Feature caching accelerates sampling by reusing or predicting intermediate activations across diffusion steps, exploiting computational redundancy along the reverse trajectory. In this work, we align caching schedule selection with the goals of inference: preserving fidelity to uncached outputs and improving generation quality under a fixed budget of full model evaluations. We propose **ReCache**, a policy-gradient method that learns a budget-conditioned scheduling policy to select which diffusion steps to fully recompute. Training combines an output-fidelity objective with a generation-quality reward, requiring no labeled data or backpropagation through the full denoising trajectory. ReCache supports both feature reuse and feature forecasting; for each model and caching mechanism, a single trained policy adapts across computational budgets at inference time without retraining. On FLUX.1-dev, ReCache reduces LPIPS by 31% compared to DiCache at a $5.04\times$ FLOPs reduction. On Wan2.1, at a $3.50\times$ FLOPs reduction, it reduces LPIPS by 45% and improves VBench by 6.77 points over uniform HiCache.



Figure 1: ReCache preserves text and visual fidelity under high acceleration on FLUX.1-dev Black Forest Labs (2024). Under a $5.4\times$ acceleration, baselines with uniform (TaylorSeer Liu et al. (2025b)) and optimized (DPCache Cui et al. (2026)) schedules fail to maintain image quality and corrupt text spelling. ReCache learns where to recompute by evaluating each schedule on the final generated image, and thus more closely matches full inference at the same computational cost.

1 INTRODUCTION

Diffusion models have become one of the dominant paradigms for high-fidelity image and video generation (Ho et al., 2020; Rombach et al., 2022; Blattmann et al., 2023). Recent transformer-based architectures, such as DiT (Peebles & Xie, 2023), have further improved scalability and generation quality, enabling increasingly powerful text-to-image and text-to-video systems. However, these advances come at a substantial inference cost: generating a single sample requires repeatedly evaluating a large denoising network over many sampling steps. As model size, resolution, and video length continue to grow, reducing the cost of diffusion inference has become an important practical challenge.

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.

A prominent line of work accelerates diffusion sampling through *feature caching*. The key observation is that consecutive denoising steps often perform highly redundant computations: intermediate activations change smoothly along the reverse trajectory and can therefore be reused or approximated across nearby timesteps (Ma et al., 2024b; Selvaraju et al., 2024). Existing caching methods typically differ along two axes: *what* to cache and *when* to cache. Along the first axis, methods range from directly reusing selected model components (Ma et al., 2024b; Selvaraju et al., 2024; Chen et al., 2024b) to forecasting future features from several cached ones (Liu et al., 2025b; Feng et al., 2025). Along the second axis, most methods place full computations at uniform intervals or trigger them online using per-step error heuristics (Liu et al., 2025a; Bu et al., 2025). More recently, several works have started to treat the *when* question as an optimization problem in its own right: LeMiCa (Gao et al., 2025) and DPCache (Cui et al., 2026) select a schedule offline by solving a path problem over pre-computed error costs.

In this work, we focus on the scheduling problem, arguing that caching schedules should be optimized for the final output rather than chosen by predefined rules or local error criteria. Adaptive heuristics (Liu et al., 2025a; Ma et al., 2026; Bu et al., 2025) and some global schedulers (Cui et al., 2026) mostly aim to keep the accelerated sampling process close to the original full-inference trajectory at intermediate denoising steps. However, this objective can be suboptimal: the steps that best preserve intermediate *trajectory consistency* may not coincide with those most important for the final output. Methods that do measure the impact on the output (Gao et al., 2025) estimate the cost of each cached segment separately and then combine these estimates along the path, which ignores how errors introduced at different reuse steps interact and compound along the trajectory. We therefore argue that caching schedules should be optimized directly for *output consistency*, i.e., for preserving the perceptual and semantic quality of the generated sample relative to full inference.

To achieve this, we frame caching-schedule selection as a Reinforcement Learning (RL) problem. We treat cached inference as a black-box, non-differentiable function of the schedule and optimize it end-to-end, without decomposing it into per-step or per-segment surrogate costs. We propose ReCache, a budget-aware method that learns a scheduling policy for diffusion feature caching. Once a policy is trained for a specific model and caching setup, ReCache takes a target budget and predicts a schedule, determining exactly where full computation is necessary and where cached features are sufficient. Unlike a prior RL-based method (Zhang et al., 2025b), which controls compute only indirectly through a reward trade-off coefficient, ReCache takes the budget as an explicit input, and a single policy serves all budgets. The policy is optimized via a combination of two objectives: minimizing the distance to the full-inference samples and preserving the perceptual and semantic quality of the generated images.

We evaluate ReCache on FLUX.1-dev (Black Forest Labs, 2024), HunyuanVideo (Kong et al., 2024), and Wan2.1 (Wan et al., 2025), and combine it with recent caching mechanisms such as TaylorSeer (Liu et al., 2025b) and HiCache (Feng et al., 2025). A single trained policy (for a given caching mechanism) adapts to multiple FLOPs budgets at inference time, avoiding the need to design a separate schedule for each acceleration regime. Across all evaluated caching mechanisms, budgets, and backbones, ReCache improves both LPIPS and HPS over the uniform schedule. It also outperforms the DP-optimized schedule of DPCache on all metrics at the tightest budget, where caching is most challenging, and performs on par with it at looser budgets.

In summary, our main contributions are:

- We frame caching schedule selection as a budget-aware RL problem that directly optimizes the output consistency of the entire schedule under a fixed compute budget, instead of relying on predefined rules or surrogate error criteria.
- We propose **ReCache**, a lightweight scheduling method compatible with diverse diffusion models and caching mechanisms.
- We show that ReCache improves over existing scheduling strategies, especially under tight budgets, across multiple budgets, caching mechanisms, and state-of-the-art image and video models, including FLUX, HunyuanVideo, and Wan2.1.

2 PRELIMINARIES

Diffusion and Flow Matching models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020b; Lipman et al., 2022; Liu et al., 2022; Albergo & Vanden-Eijnden, 2022) generate samples by numerically integrating a learned ODE that transports a Gaussian latent $\mathbf{z} \sim \mathcal{N}(0, I)$ to a data sample. A diffusion model G produces a sample in N discrete inference steps, determined by a timestep schedule and an ODE solver. Each step evaluates a learned velocity $\boldsymbol{v}(\mathbf{x}_t, t)$ parameterized as a neural network, and this evaluation dominates the per-step cost. In modern image and video models, this network is a diffusion transformer (Peebles & Xie, 2023; Esser et al., 2024; Black Forest Labs, 2024; Wan et al., 2025; Kong et al., 2024), whose forward pass through attention and feedforward blocks is what caching methods aim to amortize. Since our method is agnostic to the specific timestep schedule, we index inference steps by their position $\{1, 2, \ldots, N\}$ throughout the paper.

Feature caching. Recent works observe that the intermediate activations of $\boldsymbol{v}_\theta$ change slowly between adjacent inference steps and propose to amortize the per-step cost by reusing them. The inference steps are split into two groups: *cache steps*, where the network is evaluated in full and selected activations are stored, and *reuse steps*, where the missing activations are reconstructed from the cache. A caching schedule $\mathbf{s}$ is a subset $\mathbf{s} \subseteq \{1, \ldots, N\}$ of cache steps, while its size $|\mathbf{s}| = k$ is a *caching budget* – it controls the number of full model evaluations in the generation process.

Apart from a problem of choosing a caching schedule, *caching mechanisms* $\mathbf{M}$ set a rule of how the reuse steps reconstruct activations that a full forward trajectory would have produced. *Direct reuse* mechanisms copy cached activations from the most recent cache step, either as block outputs (FORA (Selvaraju et al., 2024)) or as residuals added to the current step's input (Δ-DiT (Chen et al., 2024b)). *Feature forecasting* mechanisms instead extrapolate the current activations from values cached at several previous cache steps: TaylorSeer (Liu et al., 2025b) fits a Taylor polynomial to the trajectory of cached features, and HiCache (Feng et al., 2025) replaces the Taylor basis with Hermite, better suited to diffusion feature trajectories. Our experiments evaluate ReCache on top of both direct and forecasting methods, but ReCache is agnostic to the choice of $\mathbf{M}$.

Caching schedules. Many existing methods place cache steps at uniform intervals (Chen et al., 2024b; Selvaraju et al., 2024; Liu et al., 2025b; Feng et al., 2025), which ignores that denoising steps contribute unequally to the final sample. Others trigger them when an estimated activation error exceeds a threshold (Liu et al., 2025a; Bu et al., 2025; Ma et al., 2026), so the number of full computations is controlled only through threshold hyperparameters and varies across samples. Optimization-based methods select the schedule offline by solving a path problem over precomputed error costs, such as the accumulated deviation from the full-inference trajectory (Cui et al., 2026) or per-segment output errors (Gao et al., 2025). ReCache instead learns a budget-conditioned policy by directly optimizing an objective that combines final-output fidelity and generation quality for complete caching schedules. An extended discussion of prior approaches is provided in Appendix A.

k-subset selection Selecting a k-subset $\mathbf{s} \subseteq \{1, \ldots, N\}$ based on a criterion $\mathcal{L}$ is a combinatorial optimization problem with $\binom{N}{k}$ candidates, making the exhaustive search computationally prohibitive. A gradient-based alternative is to instead learn a distribution $p_\theta(\mathbf{s})$ over k-subsets by minimizing the expected loss $\mathbb{E}_{p_\theta(\mathbf{s})}\mathcal{L}(\mathbf{s})$ over θ. The REINFORCE (Williams, 1992) algorithm offers an unbiased estimate $\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{s})}\mathcal{L}(\mathbf{s}) \approx \mathcal{L}(\mathbf{s})\nabla_\theta \log p_\theta(\mathbf{s})$ of the gradient, but often requires baselines to reduce variance. Leave-one-out (LOO) baseline (Kool et al., 2019a; Richter et al., 2020) uses n samples $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)} \sim p_\theta$ and their mean loss value $\overline{\mathcal{L}} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{s}^{(i)})$ to estimate the gradient as

$$\nabla_\theta \, \mathbb{E}_{p_\theta(\mathbf{s})} \, \mathcal{L}(\mathbf{s}) \approx \frac{1}{n-1} \sum_{i=1}^{n} \Big(\mathcal{L}(\mathbf{s}^{(i)}) - \overline{\mathcal{L}} \Big) \, \nabla_\theta \log p_\theta(\mathbf{s}^{(i)}). \tag{1}$$

To enforce the fixed-cardinality constraint, a natural choice is to sample without replacement using the Plackett–Luce distribution (Plackett, 1975). Each element i is assigned an importance logit θ_i, and k distinct elements $(\mathbf{s}_1, \ldots, \mathbf{s}_k)$ are selected sequentially, with probabilities proportional to $\exp(\theta_i)$ among the remaining elements, resulting in a closed-form log-probability

$$\log p_\theta(\mathbf{s}) = \sum_{i=1}^{k} \Big[\theta_{\mathbf{s}_i} - \log \sum_{j \notin \{\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}\}} \exp(\theta_j) \Big]. \tag{2}$$

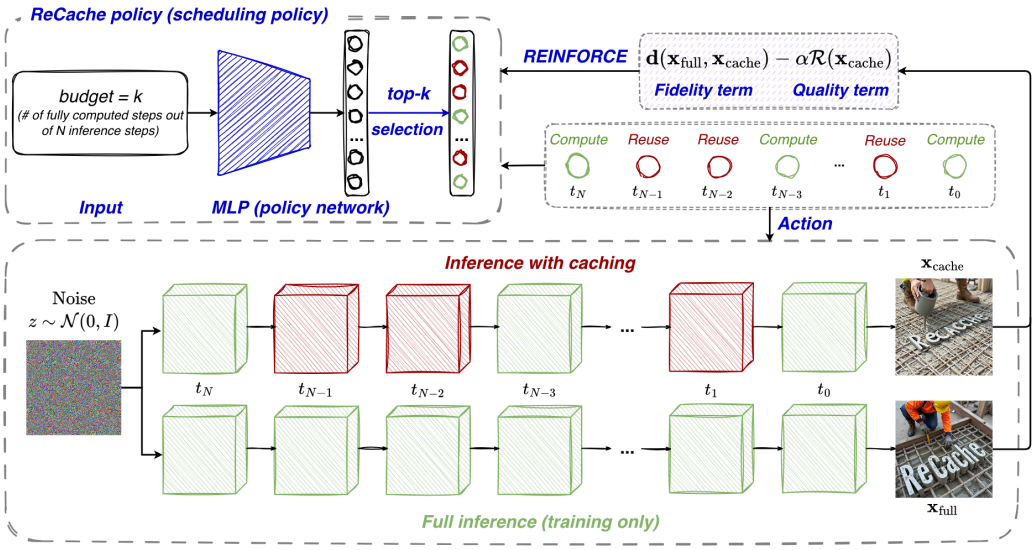


Figure 2: Overview of the ReCache method. Given a target computational budget k, a lightweight MLP policy network predicts importance logits to perform a top-k selection of inference steps for full computation. During the inference with caching, the selected steps are fully computed (green blocks), while the intermediate steps reuse or forecast features (red blocks). The policy is trained using the REINFORCE algorithm without backpropagating through the diffusion process. The reward formulation balances a Fidelity term (minimizing the distance to the $\mathbf{x}_{\text{full}}$ output from the full inference) and a Quality term (maximizing the perceptual quality of the generated $\mathbf{x}_{\text{cache}}$).

In practice, sampling from the Plackett-Luce distribution is done via the Gumbel-Top-k trick (Kool et al., 2019b; Gadetsky et al., 2020; Struminsky et al., 2021), which produces sample in one shot by perturbing each logit with independent Gumbel noise and returning the top-k indices:

$$\begin{aligned} \mathbf{g}_i &= \theta_i - \log(-\log \mathbf{u}_i), \quad \mathbf{u}_i \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0,1), \\ \mathbf{s} &= (\mathbf{s}_1, \ldots, \mathbf{s}_k) = \operatorname{argtopk}(\mathbf{g}_1, \ldots, \mathbf{g}_N). \end{aligned} \tag{3}$$

3 Method

We introduce **ReCache**, a framework that trains a schedule policy: given a target budget k, the policy selects k inference steps to recompute (Figure 2). We cast the training as a stochastic policy-optimization problem and parameterize the policy as a budget-conditioned distribution over k-subsets.

3.1 Policy Optimization

To evaluate a schedule's quality, we combine two complementary signals. The first signal asks that the cached generation stays close to the full-inference one. This target provides strong supervision, since the full-inference output is a direct reference for what the model would have produced without caching. Under aggressive caching, however, exact matching of the full-inference generation is difficult to achieve, and the policy may need to settle for an approximation. The second signal is a reward model (Xu et al., 2023; Wu et al., 2023; Kirstain et al., 2023) that scores generation quality directly, letting the policy fine-tune perceptual details that the distance term alone cannot recover.

Concretely, let $\mathbf{z} \sim \mathcal{N}(0, I)$ and $\mathbf{x}_{\text{full}} = G(\mathbf{z})$, and denote by $\mathbf{x}_{\text{cache}} = G(\mathbf{z} \mid \mathbf{s}, \mathbf{M})$ the output produced under schedule $\mathbf{s}$ and caching mechanism $\mathbf{M}$. Given a distance $\mathbf{d}$ and a reward model $\mathcal{R}$, the optimized loss is equal to

$$\mathcal{L}_{\text{image}}(\mathbf{s}) = \mathbf{d}\left(\mathbf{x}_{\text{cache}}, \mathbf{x}_{\text{full}}\right) - \alpha_{\text{iq}}\, \mathcal{R}\left(\mathbf{x}_{\text{cache}}\right). \tag{4}$$

Direct minimization of $\mathcal{L}_{\text{image}}$ over schedules is infeasible due to the search space containing $\binom{N}{k}$ subsets. As discussed in Section 2, we instead train a distribution p_θ over k-subsets to minimize the

Algorithm 1 ReCache schedule policy **training**

Input: diffusion model G with N backbone steps, caching mechanism $\mathbf{M}$, number of leave-one-out samples n, distance function $\mathbf{d}$, reward function $\mathcal{R}$, loss coefficient α_{iq}, regularizer coefficient λ_{entropy};

repeat

  for all elements in batch **do**

    $\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{x}_{\text{full}} = G(\mathbf{z})$ ▷ pre-sampled noise and precomputed full-inference image

    $k \sim q(k)$ ▷ sample caching budget

    $\theta = \text{MLP}_\phi(k)$ ▷ logits parameterized by MLP

    for $i = 1$ to n **do**

      Sample $\mathbf{s}^{(i)} = (\mathbf{s}_1^{(i)}, \ldots, \mathbf{s}_k^{(i)}) \sim p_\theta$ ▷ via Eq. 3

      $\mathbf{x}_{\text{cache}}^{(i)} = G(\mathbf{z} \mid \mathbf{s}^{(i)}, \mathbf{M})$ ▷ generate image with cached inference

      $\mathcal{L}_{\text{image}}(\mathbf{s}^{(i)}) = \mathbf{d}\big(\mathbf{x}_{\text{cache}}^{(i)}, \mathbf{x}_{\text{full}}\big) - \alpha_{\text{iq}}\, \mathcal{R}(\mathbf{x}_{\text{cache}}^{(i)})$

      Compute $\log p_\theta(\mathbf{s}^{(i)})$ ▷ via Eq. 2

      $\mathcal{L}_{\text{reg}}(\mathbf{s}^{(i)}) = \mathcal{L}_{\text{image}}(\mathbf{s}^{(i)}) + \lambda_{\text{entropy}} \log p_\theta(\mathbf{s}^{(i)})$

    end for

    $\overline{\mathcal{L}_{\text{reg}}} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\text{reg}}(\mathbf{s}^{(i)})$

    $\mathcal{L}(\mathbf{s}^{(i)}) = \big(\mathcal{L}_{\text{reg}}(\mathbf{s}^{(i)}) - \overline{\mathcal{L}_{\text{reg}}}\big)$.stopgrad()

    $\nabla_\theta \mathcal{L}(\mathbf{s}) = \frac{1}{n-1} \sum_{i=1}^{n} \mathcal{L}(\mathbf{s}^{(i)})\, \nabla_\theta \log p_\theta(\mathbf{s}^{(i)})$

  end for

  Accumulate gradients across batch elements and update ϕ via backprop through $\theta = \text{MLP}_\phi(k)$.

until converged

expected loss

$$\min_\theta \mathbb{E}_{p_\theta(\mathbf{s})}\, \mathcal{L}_{\text{image}}(\mathbf{s}). \tag{5}$$

Without further care, the policy can prematurely converge to a suboptimal schedule. We counter this with an entropy bonus $-\lambda_{\text{entropy}} H(\theta)$ that keeps the policy exploring during training, yielding the regularized loss

$$\mathcal{L}_{\text{reg}}(\mathbf{s}) = \mathcal{L}_{\text{image}}(\mathbf{s}) + \lambda_{\text{entropy}} \log p_\theta(\mathbf{s}). \tag{6}$$

3.2 Training Algorithm

We parameterize the budget-aware k-subset selection policy with the Plackett-Luce distribution (Equation 2). A lightweight MLP with parameters ϕ maps the target budget k to step-selection logits, $\theta = \text{MLP}_\phi(k)$.

Algorithm 1 summarizes ReCache training. Before training begins, we pre-sample a small dataset of pairs $(\mathbf{z}, \mathbf{x}_{\text{full}})$, where each pair consists of an initial noise sample and the corresponding full-inference output of G. At each training step, we draw a target budget $k \sim q(k)$, obtain logits $\theta = \text{MLP}_\phi(k)$, sample n schedules from the resulting Plackett-Luce distribution via the Gumbel-Top-k trick (Equation 3), and update ϕ with the REINFORCE-LOO estimator (Equation 1) of the regularized loss $\mathcal{L}_{\text{reg}}$. Training across budgets allows a single policy to serve a range of inference budgets without retraining.

At inference, we drop the Gumbel noise and use the deterministic schedule $\mathbf{s} = \text{argtopk}(\theta_1, \ldots, \theta_N)$. This schedule is computed once per budget and reused across generations, requiring no per-sample policy evaluation. We analyze the learned schedules in Section 4.3.

4 Experiments

4.1 Experimental Setup

We evaluate ReCache on three widely used image and video generation backbones: FLUX.1-dev (Black Forest Labs, 2024) for text-to-image generation, and HunyuanVideo (Kong et al., 2024) and Wan2.1 (Wan et al., 2025) for text-to-video generation. For FLUX.1-dev, we use $N = 50$

inference steps and generate images at 1024×1024 resolution. HunyuanVideo uses $N = 50$ steps and generates videos at $480 \times 640 \times 65$ resolution, while Wan2.1 uses $N = 25$ steps and $480 \times 832 \times 81$.

Baselines. ReCache is a scheduling policy and is agnostic to the underlying caching mechanism. We therefore evaluate it across several computational budgets — 7, 9, and 13 full-computation steps — and combine it with state-of-the-art caching mechanisms. For feature-forecasting methods, including TaylorSeer (Liu et al., 2025b) and HiCache (Feng et al., 2025), we compare ReCache with a uniform schedule. DiCache (Bu et al., 2025) introduces Dynamic Cache Trajectory Alignment as a caching mechanism and Online Probe Profiling as a scheduling strategy. DPCache (Cui et al., 2026) focuses on schedule selection and uses a modified TaylorSeer mechanism called Taylor-DP. For fair comparison with DiCache and DPCache, we keep the corresponding caching mechanism fixed and replace only the schedule with ReCache. We also include a naive step-reduction baseline, where the model is run with fewer denoising steps and no caching. All results are grouped by caching mechanism and budget.

Evaluation metrics. For text-to-image generation, we generate images for 1000 MS-COCO (Lin et al., 2014) prompts and report LPIPS (Zhang et al., 2018) and HPSv2.1 (Wu et al., 2023). LPIPS is computed against the corresponding full-inference output. We also report ImageReward (Xu et al., 2023) on 200 DrawBench (Saharia et al., 2022) prompts (IR DB). Adding DrawBench datasets improves diversity of the test set by covering substantially different categories, including counting, spatial relationships, conflicting concepts, etc.

For text-to-video evaluation, we generate five videos with different random seeds for each of the 944 VBench (Huang et al., 2024) prompts, yielding 4720 videos per configuration. We report VBench scores and frame-level LPIPS and HPSv2.1 on this same set of videos. Frame-level metrics are averaged over frames, prompts, and seeds, with LPIPS computed against the corresponding full-inference frames. For all experiments, we report TFLOPs and acceleration rate to measure inference efficiency.

Training datasets and details. Each training sample contains a prompt, an initial noise sample, and a precomputed full-inference reference. Each training set contains 64 samples, with prompts drawn from MS-COCO for text-to-image experiments and obtained through stratified sampling over VBench dimensions for text-to-video experiments. We use patch-wise LPIPS (PLPIPS) as the distance $\mathbf{d}$ and HPSv2.1 as the generation-quality reward $\mathcal{R}$. For video models, both terms are evaluated only on the first frame: PLPIPS compares the generated first frame with the first frame of the corresponding full-inference reference video, while HPSv2.1 evaluates the generated first frame. We set $\alpha_{\text{iq}} = 7$ and train for 50 epochs; λ_{entropy} is initialized to 0.005 for FLUX.1-dev and 0.01 for video models and annealed to zero. To make ReCache budget-aware, we randomly sample the number of full-computation steps for each training sample, with most values concentrated in the range $k \in [6, 15]$.

More details about model configurations, baseline reproduction, training datasets, and hyperparameters are provided in Appendix B.

4.2 Main experimental results

Our experiments show that ReCache improves the performance of the evaluated caching mechanisms in both image and video generation. The results show that it generally improves both LPIPS fidelity and generative quality considering HPS and VBench scores, which were specifically constructed to be aligned with human perception. The improvements are especially pronounced under tight compute budgets.

Quantitative comparison. Tables 1, 2, and 3 report results for the evaluated caching mechanisms and budgets k on FLUX.1-dev (image), Wan2.1, and HunyuanVideo (video). Compared with uniform and adaptive schedules in the same (mechanism, budget) setup, ReCache generally improves LPIPS fidelity while achieving competitive or higher perceptual quality scores (HPSv2.1 and ImageReward for images; HPSv2.1 for video). VBench scores improve in most video-generation settings and remain close to the baselines otherwise, with the largest decrease of 0.4 points occurring in a

Table 1: ReCache caching schedule policy for FLUX.1-dev. Comparison across caching mechanisms and inference budgets k. In most settings, ReCache achieves better full-inference fidelity and higher image quality under the same computational budget, with particularly strong gains in low-compute regimes. O denotes the forecasting polynomial order.

	Acceleration		Metrics		
Method	**TFLOPs↓**	**Speedup rate↑**	**LPIPS↓**	**HPS↑**	**IR DB↑**
50 steps	2991	×1.00	0.000	0.306	1.008
7 steps					
7 steps of full inference	431	×6.93	0.556	0.277	0.675
DiCache	477	×6.27	0.546	0.276	0.758
ReCache DiCache	477	×6.27	**0.413**	**0.296**	**0.990**
uniform TaylorSeer ($O = 2$)	433	×6.91	0.557	0.281	0.767
ReCache TaylorSeer ($O = 2$)	433	×6.91	**0.550**	**0.290**	**0.976**
uniform HiCache ($O = 2$)	433	×6.91	0.557	0.278	0.689
ReCache HiCache ($O = 2$)	433	×6.91	**0.498**	**0.299**	**1.015**
DPCache	432	×6.92	0.546	0.287	0.931
ReCache Taylor-DP ($O = 2$)	432	×6.92	**0.490**	**0.299**	**1.012**
9 steps					
9 steps of full inference	550	×5.43	0.530	0.288	0.855
DiCache	594	×5.04	0.456	0.296	0.974
ReCache DiCache	594	×5.04	**0.316**	**0.302**	**1.014**
uniform TaylorSeer ($O = 2$)	552	×5.42	0.520	0.293	0.915
ReCache TaylorSeer ($O = 2$)	552	×5.42	**0.430**	**0.304**	**1.008**
uniform HiCache ($O = 2$)	552	×5.42	0.521	0.290	0.886
ReCache HiCache ($O = 2$)	552	×5.42	**0.331**	**0.300**	**0.994**
DPCache	551	×5.43	0.433	0.299	0.996
ReCache Taylor-DP ($O = 2$)	551	×5.43	**0.410**	**0.305**	**1.004**
13 steps					
13 steps of full inference	789	×3.79	0.484	0.295	0.931
DiCache	828	×3.61	0.367	**0.303**	1.001
ReCache DiCache	828	×3.61	**0.197**	**0.303**	**1.026**
uniform TaylorSeer ($O = 2$)	790	×3.79	0.469	0.300	1.001
ReCache TaylorSeer ($O = 2$)	790	×3.79	**0.195**	**0.306**	**1.041**
uniform HiCache ($O = 2$)	790	×3.79	0.471	0.299	0.967
ReCache HiCache ($O = 2$)	790	×3.79	**0.212**	**0.303**	**1.045**
DPCache	790	×3.79	0.276	**0.307**	**1.046**
ReCache Taylor-DP ($O = 2$)	790	×3.79	**0.244**	0.306	1.034

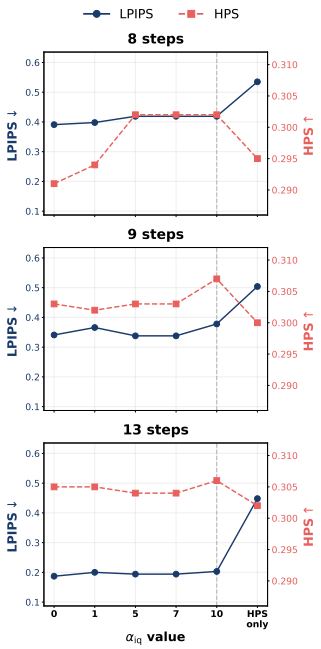


Figure 3: Impact of the quality reward coefficient α_{iq}. At the tightest budget ($k = 8$), a larger α_{iq} trades fidelity for perceptual quality, increasing both HPS and LPIPS. At $k = 9$ and $k = 13$, values up to $\alpha_{\text{iq}} = 7$ show no consistent trend, but $\alpha_{\text{iq}} = 10$ starts trading LPIPS for HPS. Training with the HPS reward alone degrades both metrics.

single configuration. This suggests that the benefits extend beyond frame-wise fidelity to overall video quality. Compared with the DP-optimized schedule of DPCache under the same Taylor-DP mechanism, ReCache is better on all metrics and all three backbones at the tightest budget $k = 7$ — the most aggressive regime, corresponding to $\times 3.5 - 7.1$ acceleration. At larger budgets, the two approaches perform more comparably. The benefits of learned scheduling are particularly apparent under tight compute budgets, where ReCache narrows the gap to full inference even when default schedules degrade substantially. Overall, these results support learned scheduling as a complementary improvement to the underlying caching mechanism across multiple image and video models.

Qualitative comparison. Figure 4 shows that baseline schedules can fail under aggressive caching, producing corrupted text, distorted object layouts, and missing fine-grained details. ReCache preserves these visual structures more reliably at the same compute budget, indicating that the learned schedule better identifies the denoising steps where full model evaluation is most critical.

The same pattern extends to video generation, as shown in Figure 5. Under a tight $k = 7$ budget, ReCache improves both TaylorSeer and DPCache outputs, preserving sharper spatial details and reducing temporal artifacts across frames.

4.3 Ablation Study

We next analyze the design choices of the proposed ReCache method. All of the ablation experiments are conducted on FLUX.1-dev using the TaylorSeer ($O = 1$) caching mechanism.

Table 2: Text-to-video results on Wan2.1 across multiple caching budgets. O denotes the forecasting polynomial order.

Method	Acceleration		Metrics		
	TFLOPs↓	Speedup↑	LPIPS↓	HPS↑	VBENCH↑
25 steps	4138	×1.00	0.000	0.231	78.775
7 steps					
7 steps of full inference	1182	×3.50	0.506	0.185	71.407
uniform TaylorSeer ($O=1$)	1184	×3.50	0.504	0.182	71.158
ReCache TaylorSeer ($O=1$)	1184	×3.50	**0.302**	**0.226**	**77.287**
uniform HiCache ($O=1$)	1183	×3.50	0.512	0.177	70.450
ReCache HiCache ($O=1$)	1183	×3.50	**0.283**	**0.222**	**77.217**
uniform Taylor-DP ($O=1$)	1183	×3.50	0.507	0.181	70.567
DPCache ($O=1$)	1183	×3.50	0.435	0.199	70.898
ReCache Taylor-DP ($O=1$)	1183	×3.50	**0.303**	**0.225**	**76.462**
9 steps					
9 steps of full inference	1511	×2.74	0.482	0.198	74.018
uniform TaylorSeer ($O=1$)	1512	×2.74	0.481	0.196	73.745
ReCache TaylorSeer ($O=1$)	1512	×2.74	**0.204**	**0.230**	**78.079**
uniform HiCache ($O=1$)	1512	×2.74	0.486	0.192	73.255
ReCache HiCache ($O=1$)	1512	×2.74	**0.232**	**0.225**	**77.738**
uniform Taylor-DP ($O=1$)	1511	×2.74	0.484	0.195	73.478
DPCache ($O=1$)	1511	×2.74	**0.226**	**0.231**	**78.329**
ReCache Taylor-DP ($O=1$)	1511	×2.74	0.230	0.230	78.203
13 steps					
13 steps of full inference	2168	×1.91	0.434	0.217	76.201
uniform TaylorSeer ($O=1$)	2168	×1.91	0.433	0.213	76.086
ReCache TaylorSeer ($O=1$)	2168	×1.91	**0.126**	**0.232**	**78.713**
uniform HiCache ($O=1$)	2168	×1.91	0.436	0.211	75.928
ReCache HiCache ($O=1$)	2168	×1.91	**0.139**	**0.230**	**78.598**
uniform Taylor-DP ($O=1$)	2168	×1.91	0.436	0.213	75.951
DPCache ($O=1$)	2168	×1.91	**0.136**	**0.232**	**78.757**
ReCache Taylor-DP ($O=1$)	2168	×1.91	0.143	0.231	78.601

Table 3: Text-to-video results on HunyuanVideo across multiple caching budgets. O denotes the forecasting polynomial order.

Method	Acceleration		Metrics		
	TFLOPs↓	Speedup↑	LPIPS↓	HPS↑	VBENCH↑
50 steps	14050	×1.00	0.000	0.241	79.932
7 steps					
7 steps of full inference	1972	×7.13	0.524	0.208	75.802
uniform TaylorSeer ($O=1$)	10696	×1.31	0.517	0.216	77.380
ReCache TaylorSeer ($O=1$)	10696	×1.31	**0.407**	**0.228**	**77.606**
uniform HiCache ($O=1$)	10696	×1.31	0.523	0.211	76.629
ReCache HiCache ($O=1$)	10696	×1.31	**0.371**	**0.224**	**77.032**
uniform Taylor-DP ($O=1$)	1977	×7.11	0.524	0.212	76.921
DPCache ($O=1$)	1977	×7.11	0.412	0.220	76.194
ReCache Taylor-DP ($O=1$)	1977	×7.11	**0.376**	**0.227**	**77.274**
9 steps					
9 steps of full inference	2533	×5.55	0.494	0.218	77.277
uniform TaylorSeer ($O=1$)	10852	×1.29	0.488	0.224	78.670
ReCache TaylorSeer ($O=1$)	10852	×1.29	**0.342**	**0.236**	**78.808**
uniform HiCache ($O=1$)	10852	×1.29	0.494	0.221	78.098
ReCache HiCache ($O=1$)	10852	×1.29	**0.320**	**0.231**	**78.636**
uniform Taylor-DP ($O=1$)	2539	×5.53	0.495	0.223	78.445
DPCache ($O=1$)	2539	×5.53	**0.308**	0.233	78.416
ReCache Taylor-DP ($O=1$)	2539	×5.53	0.320	**0.234**	**78.615**
13 steps					
13 steps of full inference	3657	×3.84	0.451	0.229	78.597
uniform TaylorSeer ($O=1$)	11164	×1.26	0.441	0.232	**79.370**
ReCache TaylorSeer ($O=1$)	11164	×1.26	**0.197**	**0.237**	79.312
uniform HiCache ($O=1$)	11164	×1.26	0.448	0.230	**79.092**
ReCache HiCache ($O=1$)	11164	×1.26	**0.203**	**0.235**	78.942
uniform Taylor-DP ($O=1$)	3662	×3.84	0.447	0.233	79.352
DPCache ($O=1$)	3662	×3.84	0.224	**0.240**	**79.692**
ReCache Taylor-DP ($O=1$)	3662	×3.84	**0.190**	0.238	79.321

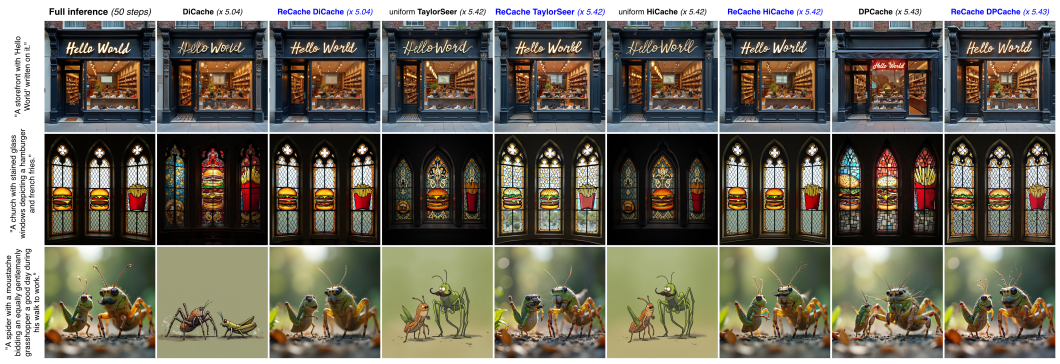


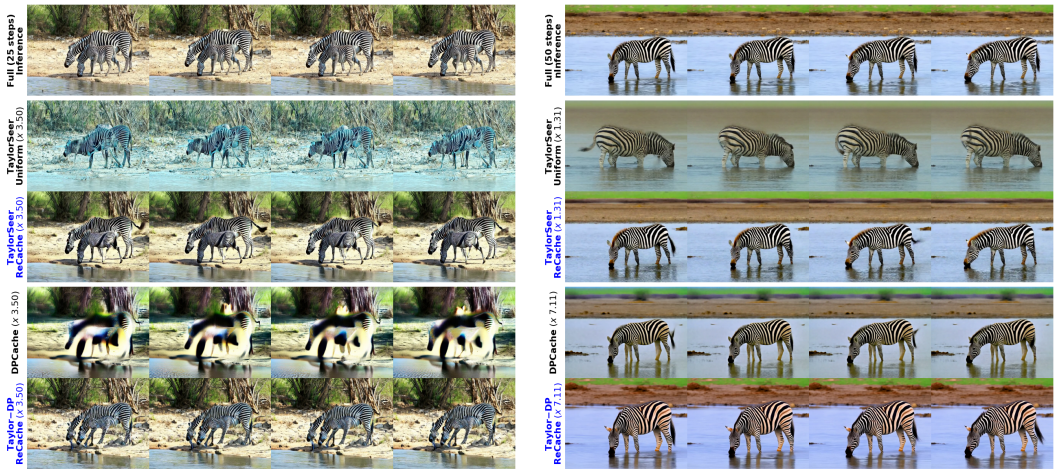
Figure 4: Qualitative comparison on FLUX.1-dev at $k = 9$. The baseline schedules (uniform, DiCache, DPCache) corrupt text spelling (top row), lose image fidelity (middle row), and fine-grained details (bottom row), while ReCache preserves these structures at the same computational cost.

Budget-aware logit predictor and nested schedules. ReCache trains a single budget-conditioned MLP, $\theta = \text{MLP}_\phi(k)$, rather than a separate logit predictor for each k. Table 4 shows that this shared model improves LPIPS at both evaluated budgets and stays within 0.001 of the per-budget predictors on HPS. We attribute this to joint training, which exposes the MLP to the full range of compute regimes and provides statistical context that a single-budget predictor lacks.

Table 4: Budget-aware ReCache versus per-budget logit predictors on FLUX.1-dev with TaylorSeer.

Method	9 steps		13 steps	
	LPIPS↓	HPS↑	LPIPS↓	HPS↑
single step	0.360	**0.303**	0.220	**0.305**
budget-aware	**0.338**	**0.303**	**0.194**	0.304

Inspecting the schedules learned by the budget-aware policy reveals a structural property: they are *nested* across all $k \in \{6, \ldots, 15\}$: the $(k{+}1)$-schedule extends the k-schedule by one step rather than reordering its existing selections. Uniform schedules lack this property entirely: their step indices all shift with k. The MLP does not impose this structure: its logits differ across budgets, yet their top-k selections agree, which suggests a largely budget-invariant ranking of step importance. Schedule visualizations across budgets are provided in Appendix C.3.



(a) Wan2.1 **(b) HunyuanVideo**

Figure 5: Qualitative comparison on Wan2.1 and HunyuanVideo at $k = 7$ for the prompt "a zebra bending down to drink water from a river". Under tight computational budgets, baseline scheduling strategies produce visible artifacts such as color degradation and structural blurring, while ReCache, using the same caching mechanism and budget, reduces these artifacts and more closely matches full inference.

Image-quality reward coefficient. The ReCache training objective combines a fidelity term, PLPIPS, with a quality reward, HPS. Figure 3 reports the effect of varying the coefficient α_{iq}, which controls the relative strength of the latter. At the tightest budget, $k = 8$, it trades fidelity for perceptual quality: raising α_{iq} from 0 to 7 increases HPS from 0.291 to 0.302 and LPIPS from 0.391 to 0.419. At $k = 9$ and $k = 13$, values up to $\alpha_{\text{iq}} = 7$ show no consistent trend, and only $\alpha_{\text{iq}} = 10$ starts trading LPIPS for HPS. Optimizing HPS alone degrades both metrics relative to $\alpha_{\text{iq}} = 7$, so the fidelity term is needed to anchor the schedule to the full-inference output. We use $\alpha_{\text{iq}} = 7$.

The appendix provides additional experiments and analyses. Appendix C.1 extends the main comparisons on with additional baselines, fidelity and quality metrics, and wall-clock latency measurements. Appendix C.2 examines training-set size, the number of sampled schedules, variability across four optimization seeds, and convergence. Appendix C.3 analyzes schedule nestedness, cross-model and cross-sampler schedule transfer without retraining, and the effect of reducing the number of backbone inference steps. Appendix C.4 compares Gumbel-Top-k with Bernoulli schedule selection and ablates the entropy regularization and image-quality reward coefficients. Appendix D reports training costs in GPU hours and GPU memory usage across backbones and caching mechanisms.

5 Conclusion

In this work, we introduce **ReCache**, a framework that learns caching schedules for a given inference budget. By framing schedule selection as a budget-aware reinforcement learning problem, ReCache evaluates each candidate schedule by the final sample it produces rather than by predefined rules or surrogate error criteria, and learns which denoising steps matter most for image and video quality. Across FLUX.1-dev, HunyuanVideo, and Wan2.1, ReCache improves over uniform and online-adaptive schedules in nearly all settings and achieves competitive or better results than path-optimized schedules, with the largest gains under tight budgets.

Ethics Statement

Faster sampling means cheaper generation, which is where most of a deployed diffusion model's lifetime compute actually goes. The flip side of inference efficiency is dual-use: any speedup we give to legitimate users we also give to people generating non-consensual or misleading content. Our method changes the sampling procedure and nothing about the model's safety properties, so it inherits

whatever filtering, watermarking, or access controls the underlying model ships with — no better, no worse. We'd encourage anyone deploying it to pair it with provenance tooling appropriate to their setting.

AI USE STATEMENT

We used generative AI tools to aid or polish the writing of this paper, including refining phrasing, improving readability, and reorganizing passages for clarity. All AI-assisted text was reviewed and edited by the authors to ensure technical accuracy and consistency with the proposed methodology and experimental findings. The authors take full responsibility for the final manuscript, including all scientific claims, results, and conclusions.

REPRODUCIBILITY STATEMENT

We describe the proposed method and training algorithm in Section 3. Appendix B provides model configurations, caching implementations, datasets, evaluation protocols, and training hyperparameters. We report training robustness experiments in Appendix C.2 and computational requirements in Appendix D. To further support reproducibility, we include our anonymized code as supplementary material.

REFERENCES

Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Black Forest Labs. Flux. `https://blackforestlabs.ai/announcing-black-forest-labs`, 2024. Accessed: 2024.

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

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4599–4603, 2023.

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

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36: 4396–4429, 2023.

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

Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *European Conference on Computer Vision*, pp. 74–91. Springer, 2024a.

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

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

Bowen Cui, Yuanbin Wang, Huajiang Xu, Biaolong Chen, Aixi Zhang, Hao Jiang, Zhengzheng Jin, Xu Liu, and Pipei Huang. Denoising as path planning: Training-free acceleration of diffusion models with dpcache. *arXiv preprint arXiv:2602.22654*, 2026.

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

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

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.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models, 2023. URL `https://arxiv.org/abs/2305.10924`.

Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, et al. Hicache: Training-free acceleration of diffusion models via hermite polynomial-based feature caching. *arXiv preprint arXiv:2508.16984*, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Artyom Gadetsky, Kirill Struminsky, Christopher Robinson, Novi Quadrianto, and Dmitry Vetrov. Low-variance black-box gradient estimates for the plackett-luce distribution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10126–10135, 2020.

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.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10696–10706, 2022.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 7514–7528, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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

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

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1383–1392, 2024.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *Advances in neural information processing systems*, 36:36652–36663, 2023.

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.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019a.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International conference on machine learning*, pp. 3499–3508. PMLR, 2019b.

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

Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *arXiv preprint arXiv:2312.09608*, 3:1, 2023a.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17535–17545, 2023b.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 7353–7363, 2025a.

Haozhe Liu, Wentian Zhang, Jinheng Xie, Francesco Faccio, Mengmeng Xu, Tao Xiang, Mike Zheng Shou, Juan-Manuel Perez-Rua, and Jürgen Schmidhuber. Faster diffusion via temporal attention decomposition. *arXiv preprint arXiv:2404.02747*, 2024.

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

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

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

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 22 (4):730–751, 2025.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

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

Xin Ma, Yaohui Wang, Xinyuan Chen, Gengyun Jia, Ziwei Liu, Yuan-Fang Li, Cunjian Chen, and Yu Qiao. Latte: Latent diffusion transformer for video generation. *arXiv preprint arXiv:2401.03048*, 2024a.

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, 2024b.

Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. Magcache: Fast video generation with magnitude-aware cache. *Advances in Neural Information Processing Systems*, 38: 34348–34380, 2026.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14297–14306, 2023.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 1975.

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

Lorenz Richter, Ayman Boustati, Nikolas Nüsken, Francisco Ruiz, and Omer Deniz Akyildiz. Vargrad: a low-variance gradient estimator for variational inference. *Advances in Neural Information Processing Systems*, 33:13481–13492, 2020.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Omid Saghatchian, Atiyeh Gh Moghadam, and Ahmad Nickabadi. Cached adaptive token merging: Dynamic token reduction and redundant computation elimination in diffusion model. *arXiv preprint arXiv:2501.00946*, 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.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Axel Sauer, Frederic Boesel, Tim Dockhorn, Andreas Blattmann, Patrick Esser, and Robin Rombach. Fast high-resolution image synthesis with latent adversarial diffusion distillation. In *SIGGRAPH Asia 2024 Conference Papers*, pp. 1–11, 2024a.

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

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35:25278–25294, 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.

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. *arXiv preprint arXiv:2010.02502*, 2020a.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.

Nikita Starodubcev, Ilya Drobyshevskiy, Denis Kuznedelev, Artem Babenko, and Dmitry Baranchuk. Scale-wise distillation of diffusion models. *arXiv preprint arXiv:2503.16397*, 2025.

Kirill Struminsky, Artyom Gadetsky, Denis Rakitin, Danil Karpushkin, and Dmitry P Vetrov. Leveraging recursive gumbel-max trick for approximate inference in combinatorial spaces. *Advances in Neural Information Processing Systems*, 34:10999–11011, 2021.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Proceedings of machine learning research*, 235:48630, 2024.

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, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. volume 13, pp. 600–612, 2004. doi: 10.1109/TIP.2003.819861.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36:15903–15935, 2023.

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

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6613–6623, 2024b.

Zhihang Yuan, Hanling Zhang, Pu Lu, Xuefei Ning, Linfeng Zhang, Tianchen Zhao, Shengen Yan, Guohao Dai, and Yu Wang. Ditfastattn: Attention compression for diffusion transformer models. *Advances in Neural Information Processing Systems*, 37:1196–1219, 2024.

Evelyn Zhang, Jiayi Tang, Xuefei Ning, and Linfeng Zhang. Training-free and hardware-friendly acceleration for diffusion models via similarity-based token pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 9878–9886, 2025a.

Hui Zhang, Tingwei Gao, Jie Shao, and Zuxuan Wu. Blockdance: Reuse structurally similar spatio-temporal features to accelerate diffusion transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 12891–12900, 2025b.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Enshu Liu, Rui Wan, Widyadewi Soedarmadji, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, et al. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. *arXiv preprint arXiv:2406.02540*, 2024.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36:49842–49869, 2023.

Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *Advances in Neural Information Processing Systems*, 36: 55502–55542, 2023.

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.

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

A Related Work

Acceleration of diffusion models Diffusion models often require a lot of computationally demanding denoising model evaluations. There are several ways to alleviate the heavy inference cost and speed up the generative process: one can either make each step faster or reduce the total number of function evaluations. The former approach includes pruning (Fang et al., 2023) and quantization techniques (Frantar et al., 2022; Chee et al., 2023; Xiao et al., 2023; Ashkboos et al., 2024; Tseng et al., 2024; Dettmers et al., 2023), which are popular in large language modeling and can be adapted for diffusion generation (Li et al., 2024; Zhao et al., 2024; Li et al., 2023b; Chen et al., 2025; Gu et al., 2022). Moreover, computationally heavy attention layers can be sped up using efficient attention (Dao et al., 2022; Yuan et al., 2024) and token merging (Bolya & Hoffman, 2023; Saghatchian et al., 2025) or pruning (Kim et al., 2024; Zhang et al., 2025a) techniques. Other papers focus on reducing the number of inference steps to accelerate the generative process by creating an efficient solver (Song et al., 2020a; Karras et al., 2022; Lu et al., 2022; 2025; Zhao et al., 2023; Zheng et al., 2023) or proposing knowledge distillation (Salimans & Ho, 2022; Meng et al., 2023), consistency-based distillation (Song et al., 2023; Luo et al., 2023) and different implementations of distribution matching (Sauer et al., 2024b;a; Yin et al., 2024b;a; Starodubcev et al., 2025). While these architectural and solver-based methods successfully accelerate generation, they are largely orthogonal to caching techniques (such as our proposed method), which focus on avoiding redundant computations inside denoising steps.

Diffusion caching Recently, it has been observed that the consequent denoising steps share a lot of similar features, and can therefore be cached and reused in the following steps. Early papers investigate U-Net (Ronneberger et al., 2015) diffusion model architectures (Rombach et al., 2022; Podell et al., 2023; Saharia et al., 2022) and reuse the high-level (Ma et al., 2024b), encoder (Li et al., 2023a), or attention (Liu et al., 2024) features at certain adjacent timesteps. With the increasing popularity of diffusion transformer-based (Vaswani et al., 2017) architectures (DiTs) (Peebles & Xie, 2023) in both image (Chen et al., 2023; 2024a; Black Forest Labs, 2024) and video (Ma et al., 2024a; Yang et al., 2024; Wan et al., 2025; Zheng et al., 2024; Kong et al., 2024) domains, research has focused on proposing novel *"What-to-Cache"* strategies (Chen et al., 2024b; Selvaraju et al., 2024; Lv et al., 2024). (Selvaraju et al., 2024) stores and reuses intermediate outputs from the most computationally demanding layers, while (Chen et al., 2024b) caches the deviations between the outputs of the whole transformer blocks. Other papers focused on the *"How-to-Use-the-Cache"* question: (Bu et al., 2025) proposes the dynamic cache trajectory alignment, while (Liu et al., 2025b) and (Feng et al., 2025) incorporate the cache-and-forecast strategy based on the approximation of the higher-order derivatives to better predict the features in future timesteps. Most of these approaches place cache steps at fixed intervals; ReCache instead learns where to place them.

Adaptive caching. Instead of fixed intervals, adaptive methods decide online, for each sample, whether the current timestep can reuse cached features or must be recomputed (Liu et al., 2025a; Ma et al., 2026; Bu et al., 2025; Kahatapitiya et al., 2025; Zhou et al., 2025). They differ mainly in how they estimate the caching error: AdaCache (Kahatapitiya et al., 2025) examines residual connections, EasyCache (Zhou et al., 2025) uses layer inputs, DiCache (Bu et al., 2025) profiles shallow-layer features, and TeaCache (Liu et al., 2025a) rescales differences of timestep-embedding-modulated inputs with a polynomial fitted on calibration prompts. MagCache (Ma et al., 2026) observes that the magnitude ratio of successive residual outputs decreases monotonically and consistently across models and prompts, and uses this ratio to model the accumulated caching error, which requires only a single calibration sample. BlockDance-Ada (Zhang et al., 2025b) instead trains a lightweight decision network with reinforcement learning to decide at each step whether to cache or reuse structure-focused transformer blocks. Most adaptive methods require inference-time threshold tuning and control compute only indirectly, through thresholds or reward weights.

Optimization-based scheduling. Several recent works select the whole caching schedule offline by solving a path-optimization problem. DPCache (Cui et al., 2026) runs full denoising on a small set of calibration prompts to build a cost tensor that measures the error of skipping the timesteps between two key steps. Dynamic programming then selects the sequence of key steps, i.e., a path through the denoising timesteps, that minimizes the accumulated deviation from the full-inference trajectory. LeMiCa (Gao et al., 2025) represents candidate cache segments as edges of a directed

acyclic graph, weighted by their impact on the final output. Among the paths that satisfy the budget, it selects the one with the smallest largest edge error; this lexicographic minimax criterion bounds the worst-case degradation rather than the accumulated error. Both methods are training-free, take the budget as input, and precompute the schedule, so they add no inference-time overhead. However, both rely on costs estimated for each segment separately and then combined along the path, so interactions between errors introduced in different segments are captured only partially. ReCache instead evaluates each complete schedule by the final sample it produces and optimizes it with a reward that combines fidelity to full inference and perceptual quality.

ReCache positioning. ReCache is an offline-trained, budget-conditioned scheduling policy that can be combined with any caching mechanism, including those of the methods above. Unlike uniform schedules, it allocates full computations according to their effect on the final sample. Unlike threshold-based adaptive methods, it takes the budget as an explicit input and requires no inference-time hyperparameter tuning. Unlike previous RL-based method (Zhang et al., 2025b), it satisfies the budget exactly rather than through a reward trade-off coefficient. Unlike optimization-based schedulers (Cui et al., 2026; Gao et al., 2025), it evaluates each complete schedule by the final sample it produces rather than by precomputed per-segment costs. By utilizing learned schedule selection, ReCache maximizes generation quality under explicit computational constraints without needing manual real-time adjustments.

B Experimental details

B.1 Modified MS-COCO dataset

As a base dataset of prompts for our text-to-image experiments we created a modified MS-COCO version, where for each of the images we utilized only one caption, which has the biggest length. This ensured we use long and informative prompts with a lot of details to train and evaluate our model on.

B.2 Baselines implementation

FORA (Selvaraju et al., 2024) is a caching mechanism that caches attention and feedforward layers (FF) in each of the transformer blocks and changes the output to the cached values straightforwardly in the reuse timesteps.

Δ-DiT (Chen et al., 2024b) caches the difference between the cache timestep input and the output of the i-th transformer block, on the reuse timestep adds this deviation to the current input and evaluates the rest of the blocks as is. TeaCache Liu et al. (2025a) and MagCache Ma et al. (2026) decide online whether to reuse the cached residual of the transformer blocks, while the underlying caching mechanism is Δ-DiT. We tune their thresholds so that the average number of full computations matches the budget k. For LeMiCa, we implemented the method and build the error-weighted graph with 64 calibration prompts. By ReCache LeMiCa we denote that we use the LeMiCa's caching mechanism and select the schedule with ReCache.

DiCache (Bu et al., 2025) consists of an Online Probe Profiling Scheme that estimates the caching error in real-time and creates the cache schedule for each sample, and a Dynamic Cache Trajectory Alignment which is a caching mechanism that adaptively approximates the output based on the probe feature trajectory. By a ReCache DiCache we denote that we utilize ReCache policy on the Dynamic Cache Trajectory Alignment.

TaylorSeer Liu et al. (2025b) and HiCache Feng et al. (2025) both cache the outputs of the attention and feedforward layers during the cache steps and then predict the current values of a reuse step using the Taylor or Hermite polynomial respectively.

DPCache Cui et al. (2026) formulates the choice of a caching schedule as a dynamic programming task, it utilizes the TaylorSeer caching mechanism of predicting the values but caches the whole-block outputs instead of separate layers and is called Taylor-DP in its implementation code.

In our HunyuanVideo experiments, we adapt the TaylorSeer baseline to follow the Hugging Face implementation by caching only the attention modules and recomputing the feed-forward layers. This modification is necessary because the original full-cache variant leads to out-of-memory errors in

our setting. We use the same feasible TaylorSeer implementation for both the baseline and ReCache TaylorSeer, ensuring a fair comparison. Moreover, the second-order TaylorSeer variant ($O = 2$) remains out-of-memory even with attention-only caching; therefore, all video experiments use the first-order variant ($O = 1$). This configuration still provides a substantial speed-up, as shown in Table 3.

B.3 ReCache training overview

Policy parameterization. We parameterize the schedule logits with a lightweight MLP, $\theta = \text{MLP}_\phi(k)$, that takes the target budget k as input and outputs N logits, one per backbone step. The MLP has approximately 100K parameters, making its overhead negligible relative to the diffusion backbone G. Per-model architecture details (input embedding and hidden dimensions) are reported in the subsections below.

Loss instantiation. We instantiate the distance term as patch-wise LPIPS (Zhang et al., 2018) and the reward as HPSv2.1 (Wu et al., 2023). The two terms are weighted by α_{iq}. Entropy regularization uses coefficient λ_{entropy}. Each REINFORCE-LOO update draws n schedules per training example; budgets are sampled from a fixed distribution $q(k)$.

Pre-sampled training set. Before training, we precompute a fixed dataset of triples — prompt, initial latent $\mathbf{x}_1^{(i)}$, and the corresponding full-inference output $\mathbf{x}_{\text{full}}^{(i)}$ of the backbone G. These references remain fixed throughout optimization, avoiding repeated full-inference generation inside the training loop. For each sampled caching schedule s, we generate an output $G(\mathbf{x}_1^{(i)} \mid \mathbf{s}, \mathbf{M})$ using caching mechanism M and evaluate the training objective using the stored reference $\mathbf{x}_{\text{full}}^{(i)}$. For video models, PLPIPS compares only the first frames of the generated and reference videos, and HPSv2.1 is also evaluated only on the generated first frame.

B.4 Implementation details

Text-to-image with FLUX.1-dev (Black Forest Labs, 2024)

- Diffusion model:
  - Number of inference steps = 50;
  - Flow Matching Euler Discrete Scheduler;
  - Image resolution 1024×1024;
  - Guidance scale: 3.5;
- Logit predictor model:
  - Input embedding dimension = 128;
  - Hidden dimension = [256, 256, 128];
- Dataset:
  - Training dataset: 64 prompts from modified MS-COCO;
  - Test dataset: 1000 prompts from modified MS-COCO;
  - DrawBench ImageReward dataset: 200 prompts from DrawBench benchmark;
- Training configurations:
  - Batch size = 1, gradient accumulation = 8;
  - Number of samples for REINFORCE leave-one-out = 6;
  - HPS metric in loss with $\alpha_{\text{iq}} = 7$;
  - Linear decreasing each epoch entropy regularizer $\lambda_{\text{entropy}} = 0.005$;
  - Number of full inference steps in training is sampled from a weighted distribution with base weight = 0.5, where $(6, 15)$ steps are sampled with weight 1.0;
  - 50 epochs of training;
  - Adam optimizer, learning rate = 0.0005, betas = $(0.9, 0.999)$, weight decay = 0.0;

Text-to-video with HunyuanVideo (Kong et al., 2024)

- Diffusion model:
  - Number of inference steps = 50;
  - Flow Matching Euler Discrete Scheduler;
  - Video resolution $480 \times 640 \times 65$;
  - Guidance scale: 6.0;
- Dataset:
  - Training dataset: 64 prompts from VBench prompts dataset obtained with stratified sampling from 11 VBench dimensions;
  - Test dataset: 944 VBench prompts with five random seeds per prompt (4720 videos), used for both VBench evaluation and frame-level LPIPS and HPSv2.1;
- Logit predictor model:
  - Input embedding dimension = 128;
  - Hidden dimension = [256, 256, 128];
- Training configurations:
  - Batch size = 1, gradient accumulation = 8;
  - Number of samples for REINFORCE leave-one-out = 6;
  - PLPIPS and HPSv2.1 are computed only on the first frame during policy training;
  - HPSv2.1 metric in loss with $\alpha_{\text{iq}} = 7$;
  - Linear decreasing each epoch entropy regularizer $\lambda_{\text{entropy}} = 0.01$;
  - Number of full inference steps in training is sampled from a weighted distribution with base weight = 0.5, where $(6, 15)$ steps are sampled with weight 1.0;
  - 50 epochs of training;
  - Adam optimizer, learning rate = 0.0005, betas = $(0.9, 0.999)$, weight decay = 0.0;

Text-to-video with Wan2.1 (Wan et al., 2025)

- Diffusion model:
  - Number of inference steps = 25;
  - Flow Matching UniPC Discrete Scheduler (shift=3);
  - Video resolution $480 \times 832 \times 81$;
  - Guidance scale: 5.0;
- Logit predictor model:
  - Input embedding dimension = 128;
  - Hidden dimension = [256, 256, 128];
- Dataset:
  - Training dataset: 64 prompts from VBench prompts dataset obtained with stratified sampling from 11 VBench dimensions;
  - Test dataset: 944 VBench prompts with five random seeds per prompt (4720 videos), used for both VBench evaluation and frame-level LPIPS and HPSv2.1;
- Training configurations:
  - Batch size = 1, gradient accumulation = 8;
  - Number of samples for REINFORCE leave-one-out = 6;
  - PLPIPS and HPSv2.1 are computed only on the first frame during policy training;
  - HPSv2.1 metric in loss with $\alpha_{\text{iq}} = 7$;
  - Linear decrease for each epoch regularizer $\lambda_{\text{entropy}} = 0.01$;
  - Number of full inference steps in training is sampled from a weighted distribution with base weight = 0.5, where $(6, 15)$ steps are sampled with weight 1.0;
  - 50 epochs of training;
  - Adam optimizer, learning rate = 0.0005, betas = $(0.9, 0.999)$, weight decay = 0.0;

C EXTENDED EXPERIMENTAL RESULTS

In this section, we present additional experimental results, including extended metric evaluations, further analysis of ReCache properties, and additional ablation studies.

C.1 EXTENDED TEXT-TO-IMAGE RESULTS

Table 5 (Part 1) and Table 5 (Part 2) present extended results on FLUX.1-dev, comparing ReCache with uniform, online-adaptive, and path-optimized schedules.

ImageReward DrawBench (IR DB) metric stand for an ImageReward (Xu et al., 2023) score calculated over 200 DrawBench (Saharia et al., 2022) prompts. The rest of the metrics – LPIPS (Zhang et al., 2018), patch-wise LPIPS (PLPIPS), L1, PSNR, SSIM (Wang et al., 2004), HPSv2.1 (Wu et al., 2023), CLIP (Hessel et al., 2021), Aesthetic Quality (AQ) (Schuhmann et al., 2022), Image Quality (IQ) and ImageReward (IR) – are computed on 1000 MS-COCO prompts.

Across all mechanisms and budgets, ReCache improves LPIPS, HPS, and ImageReward over the uniform schedule. Against the online-adaptive TeaCache and MagCache, ReCache improves fidelity at all budgets and perceptual metrics at $k \leq 9$, while at $k = 13$ both baselines score slightly higher on HPS and ImageReward. Against the path-optimized LeMiCa, ReCache improves perceptual metrics at comparable LPIPS at $k = 7$, improves all main metrics at $k = 9$, and performs comparably at $k = 13$.

Wall-clock speedups are close to the FLOPs reduction for DiCache, LeMiCa, and Taylor-DP, but lower for FORA, TaylorSeer, and HiCache, which cache features of individual layers; since the schedule is precomputed, ReCache has the same latency as any other schedule with the same mechanism and budget.

C.2 TRAINING-PROTOCOL ROBUSTNESS

We examine the sensitivity of ReCache to different training settings. All experiments use FLUX.1-dev with Taylor-DP ($O = 2$).

Training-set size. In Table 6, we examine the effect of training-set size by training on 64, 128, and 640 prompts for 50, 25, and 5 epochs, respectively, keeping the number of optimizer updates fixed at 400. Under this matched optimization budget, the 64-prompt default setup provides a strong overall performance profile at $k = 7$ and $k = 13$ and remains competitive at $k = 9$. At $k = 7$, both larger training sets yield worse LPIPS and ImageReward scores. At $k = 9$, using 640 prompts improves LPIPS from 0.410 to 0.382, but reduces ImageReward from 1.133 to 1.118. At $k = 13$, using 128 prompts improves LPIPS from 0.244 to 0.217, while PSNR drops from 23.92 to 22.43. Thus, increasing the training-set size by up to $10\times$ yields no consistent gains across budgets and metrics, supporting the 64-prompt default under a fixed optimization budget.

Number of sampled schedules. In Table 7, we examine the effect of the number of schedules sampled per training example in REINFORCE with a leave-one-out (LOO) baseline, varying $n \in 2, 4, 6, 8$. We adjust the number of optimizer updates to keep the total number of FLUX generations approximately fixed at 19,200, nearly matching training cost since generation dominates the runtime. The default $n = 6$ achieves the best results on all reported metrics at $k = 7$ and remains competitive at larger budgets. At $k = 9$, increasing n to 8 improves LPIPS from 0.410 to 0.402, but reduces ImageReward from 1.133 to 1.118. At $k = 13$, $n = 8$ improves LPIPS from 0.244 to 0.196, while $n = 6$ achieves higher PSNR (23.92 versus 23.23). Thus, performance does not improve monotonically with n, and no single value dominates across budgets and metrics. These results support $n = 6$ as a practical default under a fixed generation budget.

Training seeds. In Table 8, we examine sensitivity to the optimization seed by independently training four ReCache policies with seeds 42, 43, 44, and 45. We report the mean and standard deviation across runs, with fixed baselines included for reference. ReCache achieves lower mean LPIPS than both uniform scheduling and DPCache at all three budgets. At $k = 7$, ReCache achieves better mean scores than both baselines on all reported metrics. At $k = 9$, mean LPIPS decreases

Table 5: Part 1 of the extended experimental results on FLUX.1-dev text-to-image model. ReCache is evaluated across different caching mechanisms and is compared to uniform, online-adaptive, and path-optimized baseline schedules. Results are grouped by computational budget and caching mechanism. Part 1 of 2: budgets $k = 7$ and $k = 8$.

	Acceleration				Metrics										
Method	**Latency(s)↓**	**Speed↑**	**FLOPs(T)↓**	**Speed↑**	**LPIPS↓**	**PLPIPS↓**	**L1↓**	**PSNR↑**	**SSIM↑**	**HPS↑**	**CLIP↑**	**AQ↑**	**IQ↑**	**IR↑**	**IR DB↑**
50 steps	10.92	1	2990.80	1	0.0000	0.0000	0.0000	∞	1.0000	0.3063	26.98	6.49	71.21	1.1195	1.0076
7 steps															
7 steps of full inference	1.62	6.74	431.42	6.93	0.556	0.544	0.160	13.58	0.594	0.277	27.09	6.43	65.71	0.884	0.675
uniform FORA ($N = 7$)	2.63	4.15	432.54	6.91	0.567	0.555	0.162	13.54	0.588	0.270	27.17	6.28	66.26	0.808	0.628
ReCache FORA	2.63	4.15	432.54	6.91	**0.492**	**0.474**	**0.080**	**18.32**	**0.665**	**0.292**	**27.88**	**6.38**	**70.99**	**1.025**	**0.981**
uniform Δ-DiT ($N = 7$)	4.87	2.24	1285.13	2.33	0.589	0.577	0.180	12.71	0.593	0.226	25.80	5.55	45.97	0.161	0.185
TeaCache	4.87	2.24	1285.13	2.33	0.542	0.530	0.142	14.49	0.613	0.270	26.90	6.28	65.52	0.779	0.763
MagCache	4.87	2.24	1285.13	2.33	0.505	0.491	0.132	14.81	0.649	0.277	26.92	6.44	64.31	0.910	0.823
ReCache Δ-DiT	4.87	2.24	1285.13	2.33	**0.452**	**0.436**	**0.085**	**17.74**	**0.699**	**0.284**	**26.96**	**6.48**	**67.41**	**0.964**	**0.896**
uniform DiCache ($N = 7$)	1.82	6.00	477.02	6.27	0.553	0.541	0.160	13.58	0.597	0.277	27.05	6.34	65.92	0.889	0.658
DiCache	1.82	6.00	477.02	6.27	0.546	0.535	0.143	14.33	0.610	0.276	27.04	6.37	67.26	0.871	0.758
ReCache DiCache	1.82	6.00	477.02	6.27	**0.413**	**0.396**	**0.069**	**19.15**	**0.724**	**0.296**	**27.29**	**6.41**	**69.90**	**1.059**	**0.990**
uniform LeMiCa ($N = 7$)	1.71	6.39	432.12	6.92	0.562	0.551	0.161	13.55	0.597	0.268	26.95	6.30	63.82	0.801	0.583
LeMiCa	1.71	6.39	432.12	6.92	**0.456**	0.441	**0.076**	**18.66**	**0.687**	0.273	**27.23**	6.31	65.53	0.848	0.844
ReCache LeMiCa	1.71	6.39	432.12	6.92	**0.456**	**0.440**	0.084	17.90	0.684	**0.293**	27.11	**6.47**	**69.78**	**1.018**	**0.930**
uniform TaylorSeer ($O = 1, N = 7$)	2.93	3.73	432.58	6.91	0.555	0.544	0.160	13.56	0.581	0.282	27.30	**6.38**	68.39	0.934	0.780
ReCache TaylorSeer ($O = 1$)	2.93	3.73	432.58	6.91	**0.499**	**0.483**	**0.126**	**15.07**	**0.629**	**0.296**	**27.54**	6.34	**70.01**	**1.057**	**0.972**
uniform TaylorSeer ($O = 2, N = 7$)	3.09	3.53	432.61	6.91	0.557	0.546	0.161	13.54	**0.579**	0.281	**27.28**	**6.38**	68.48	0.933	0.767
ReCache TaylorSeer ($O = 2$)	3.09	3.53	432.61	6.91	**0.550**	**0.539**	**0.146**	**14.19**	0.560	**0.290**	27.25	6.31	**69.67**	**1.022**	**0.976**
uniform HiCache ($O = 1, N = 7$)	3.25	3.36	432.54	6.91	0.557	0.545	0.160	13.58	0.589	0.278	27.28	6.34	67.57	0.888	0.688
ReCache HiCache ($O = 1$)	3.25	3.36	432.54	6.91	**0.441**	**0.423**	**0.080**	**18.27**	**0.697**	**0.298**	**27.64**	**6.41**	**70.68**	**1.052**	**0.973**
uniform HiCache ($O = 2, N = 7$)	3.47	3.15	432.54	6.91	0.557	0.545	0.161	13.57	0.589	0.278	27.29	6.33	67.60	0.890	0.689
ReCache HiCache ($O = 2$)	3.47	3.15	432.54	6.91	**0.498**	**0.481**	**0.106**	**16.23**	**0.639**	**0.299**	**27.75**	**6.42**	**71.02**	**1.065**	**1.015**
uniform Taylor-DP ($O = 2, N = 7$)	1.66	6.58	432.12	6.92	0.554	0.543	0.160	13.56	0.591	0.280	27.08	6.36	67.08	0.925	0.727
DPCache ($O = 2, K = 7$)	1.66	6.58	432.12	6.92	0.546	0.533	0.132	14.96	0.594	0.287	**27.88**	6.22	69.09	1.014	0.931
ReCache Taylor-DP ($O = 2$)	1.66	6.58	432.12	6.92	**0.490**	**0.477**	**0.125**	**15.25**	**0.630**	**0.299**	27.26	**6.47**	**70.31**	**1.092**	**1.012**
8 steps															
8 steps of full inference	1.84	5.93	490.94	6.09	0.541	0.529	0.155	13.77	0.602	0.283	27.11	6.44	67.21	0.940	0.754
uniform FORA ($N = 6$)	2.82	3.87	492.03	6.08	0.550	0.538	0.158	13.68	0.596	0.278	27.28	6.31	67.96	0.904	0.711
ReCache FORA	2.82	3.87	492.03	6.08	**0.439**	**0.420**	**0.076**	**18.64**	**0.699**	**0.297**	**27.47**	**6.43**	**70.83**	**1.041**	**0.991**
uniform Δ-DiT ($N = 6$)	5.00	2.18	1324.80	2.26	0.575	0.563	0.175	12.89	0.599	0.241	26.11	5.82	50.58	0.374	0.333
TeaCache	5.00	2.18	1324.80	2.26	0.519	0.507	0.135	14.78	0.627	0.279	27.00	6.34	67.47	0.893	0.829
MagCache	5.00	2.18	1324.80	2.26	0.494	0.480	0.132	14.83	0.655	0.278	26.83	6.45	65.22	0.925	0.842
ReCache Δ-DiT	5.00	2.18	1324.80	2.26	**0.400**	**0.384**	**0.067**	**19.42**	**0.743**	**0.289**	**27.01**	**6.54**	**67.79**	**1.003**	**0.934**
uniform DiCache ($N = 6$)	2.03	5.38	535.48	5.59	0.537	0.526	0.157	13.70	0.603	0.284	27.11	6.40	67.63	0.953	0.734
DiCache	2.03	5.38	535.48	5.59	0.494	0.482	0.131	14.86	0.638	0.290	26.98	**6.48**	69.60	1.012	0.897
ReCache DiCache	2.03	5.38	535.48	5.59	**0.392**	**0.376**	**0.067**	**19.28**	**0.732**	**0.302**	**27.24**	6.45	**70.70**	**1.093**	**1.005**
uniform LeMiCa ($N = 6$)	1.84	5.93	491.63	6.08	0.545	0.534	0.157	13.685	0.604	0.277	27.10	6.35	65.95	0.882	0.680
LeMiCa	1.84	5.93	491.63	6.08	0.545	0.534	0.157	13.690	0.604	0.277	**27.10**	6.35	65.95	0.882	0.680
ReCache LeMiCa	1.84	5.93	491.63	6.08	**0.402**	**0.385**	**0.068**	**19.32**	**0.730**	**0.297**	27.08	**6.49**	**70.06**	**1.053**	**0.976**
uniform TaylorSeer ($O = 1, N = 6$)	3.11	3.51	492.07	6.08	0.539	0.528	0.156	13.71	0.591	0.288	27.24	**6.41**	69.47	1.002	0.825
ReCache TaylorSeer ($O = 1$)	3.11	3.51	492.07	6.08	**0.419**	**0.403**	**0.087**	**17.55**	**0.693**	**0.302**	**27.47**	6.36	**70.88**	**1.113**	**0.986**
uniform TaylorSeer ($O = 2, N = 6$)	3.28	3.33	492.10	6.08	0.542	0.531	0.157	13.69	0.588	0.288	**27.26**	6.43	69.45	1.002	0.831
ReCache TaylorSeer ($O = 2$)	3.28	3.33	492.10	6.08	**0.515**	**0.502**	**0.139**	**14.50**	**0.601**	**0.297**	27.15	**6.44**	**69.89**	**1.058**	**0.975**
uniform HiCache ($O = 1, N = 6$)	3.44	3.17	492.03	6.08	0.540	0.529	0.157	13.71	0.597	0.285	27.33	**6.37**	68.86	0.966	0.776
ReCache HiCache ($O = 1$)	3.44	3.17	492.03	6.08	**0.352**	**0.339**	**0.075**	**18.65**	**0.738**	**0.297**	**27.36**	**6.37**	**70.52**	**1.049**	**0.984**
uniform HiCache ($O = 2, N = 6$)	3.70	2.95	492.03	6.08	0.540	0.529	0.157	13.71	0.597	0.285	27.31	6.36	68.85	0.966	0.773
ReCache HiCache ($O = 2$)	3.70	2.95	492.03	6.08	**0.432**	**0.414**	**0.076**	**18.61**	**0.705**	**0.301**	**27.81**	**6.39**	**70.89**	**1.081**	**0.989**
uniform Taylor-DP ($O = 2, N = 6$)	1.87	5.84	491.63	6.08	0.539	0.528	0.157	13.70	0.598	0.286	27.11	6.44	68.39	0.988	0.797
DPCache ($O = 2, K = 8$)	1.87	5.84	491.63	6.08	0.440	0.425	0.098	16.90	0.678	0.300	**27.36**	6.41	70.35	1.102	1.030
ReCache Taylor-DP ($O = 2$)	1.87	5.84	491.63	6.08	**0.418**	**0.405**	**0.091**	**17.42**	**0.688**	**0.305**	27.19	**6.50**	**70.59**	**1.126**	**1.045**

from 0.433 for DPCache to 0.401 ± 0.016, while HPS increases from 0.299 to 0.304 ± 0.002. At $k = 13$, ReCache improves PSNR from 21.03 to 23.00 ± 0.67 and SSIM from 0.804 to 0.839 ± 0.010 relative to DPCache, while remaining competitive on the other metrics. These results suggest that the observed fidelity gains are not specific to a single optimization seed.

Convergence analysis. In Figure 6, we track the validation objective $\mathrm{LPIPS} - 7 \cdot \mathrm{HPS}$ on a held-out MS-COCO subset for the same four training runs. Most of the improvement occurs within the first 160–240 optimizer updates, and all runs reach values close to their final scores by approximately 240 updates. Further training up to 400 updates yields little additional improvement, although individual runs exhibit intermediate fluctuations. These results support the default training budget of 400 optimizer updates in the evaluated setting.

C.3 Further analysis

The following experiments analyze additional properties of ReCache, including the nested structure of learned schedules, transferability across backbones, and the interaction between caching budgets and the number of backbone inference steps.

Nestedness of learned schedules. Figure 7 plots the specific cache steps assigned by our budget-aware policy across different target budgets. We illustrate this for FLUX.1-dev using both TaylorSeer ($O = 2$) and Taylor-DP ($O = 2$) mechanisms at $k \in \{7, 9, 13\}$. As the figure shows, the selected

Table 5: (Continued) Extended experimental results on FLUX.1-dev. Part 2 of 2: budgets $k = 9$ and $k = 13$.

Method	Acceleration				Metrics										
	Latency(s)↓	Speed↑	FLOPs(T)↓	Speed↑	LPIPS↓	PLPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	CLIP↑	AQ↑	IQ↑	IR↑	IR DB↑
50 steps	10.92	1	2990.80	1	0.0000	0.0000	0.0000	∞	1.0000	0.3063	26.98	6.49	71.21	1.1195	1.0076
9 steps															
9 steps of full inference	2.06	5.30	550.47	5.43	0.530	0.518	0.150	13.98	0.609	0.288	27.05	6.47	68.11	0.983	0.855
uniform FORA ($N = 5$)	3.01	3.63	551.53	5.42	0.530	0.519	0.153	13.89	0.605	0.285	**27.28**	6.32	69.06	0.968	0.826
ReCache FORA	3.01	3.63	551.53	5.42	**0.370**	**0.355**	**0.073**	**18.91**	**0.729**	**0.297**	27.10	**6.40**	**70.45**	**1.037**	**0.987**
uniform Δ-DiT ($N = 5$)	5.15	2.12	1364.46	2.19	0.555	0.543	0.169	13.15	0.607	0.256	26.45	6.03	56.42	0.573	0.519
TeaCache	5.15	2.12	1364.46	2.19	0.489	0.477	0.126	15.25	0.646	0.288	**27.02**	6.39	**68.98**	0.984	0.876
MagCache	5.15	2.12	1364.46	2.19	0.436	0.422	0.103	16.55	0.698	0.289	26.89	6.53	68.14	1.010	0.919
ReCache Δ-DiT	5.15	2.12	1364.46	2.19	**0.385**	**0.369**	**0.067**	**19.44**	**0.748**	**0.292**	26.99	**6.54**	68.52	**1.019**	**0.940**
uniform DiCache ($N = 5$)	2.25	4.85	593.94	5.04	0.519	0.508	0.151	13.93	0.613	0.289	27.04	6.41	68.84	0.998	0.845
DiCache	2.25	4.85	593.94	5.04	0.456	0.445	0.117	15.56	0.663	0.296	26.92	**6.46**	70.55	1.050	0.974
ReCache DiCache	2.25	4.85	593.94	5.04	**0.316**	**0.304**	**0.064**	**19.61**	**0.770**	**0.302**	**27.08**	6.45	**70.88**	**1.102**	**1.014**
uniform LeMiCa ($N = 5$)	2.06	5.30	551.13	5.43	0.526	0.515	0.152	13.90	0.613	0.283	27.12	6.37	67.71	0.957	0.793
LeMiCa	2.06	5.30	551.13	5.43	0.450	0.437	0.106	16.25	0.673	0.295	**27.29**	6.41	**70.11**	1.048	0.970
ReCache LeMiCa	2.06	5.30	551.13	5.43	**0.348**	**0.334**	**0.066**	**19.48**	**0.754**	**0.297**	27.03	**6.47**	70.07	**1.054**	**0.979**
uniform TaylorSeer ($O = 1$, $N = 5$)	3.30	3.31	551.57	5.42	0.518	0.507	0.150	13.94	0.605	0.293	27.12	**6.44**	70.34	1.044	0.921
ReCache TaylorSeer ($O = 1$)	3.30	3.31	551.57	5.42	**0.338**	**0.327**	**0.082**	**17.99**	**0.735**	**0.303**	**27.18**	6.40	**71.07**	**1.117**	**0.997**
uniform TaylorSeer ($O = 2$, $N = 5$)	3.49	3.13	551.60	5.42	0.520	0.509	0.151	13.92	0.602	0.293	**27.10**	6.44	70.46	1.046	0.915
ReCache TaylorSeer ($O = 2$)	3.49	3.13	551.60	5.42	**0.430**	**0.417**	**0.099**	**16.90**	**0.667**	**0.304**	27.06	**6.45**	**71.09**	**1.106**	**1.008**
uniform HiCache ($O = 1$, $N = 5$)	3.64	3.00	551.53	5.42	0.520	0.509	0.151	13.94	0.608	0.290	27.22	**6.38**	69.68	1.019	0.887
ReCache HiCache ($O = 1$)	3.64	3.00	551.53	5.42	**0.322**	**0.308**	**0.062**	**19.83**	**0.765**	**0.296**	**27.32**	6.37	**70.44**	**1.055**	**0.987**
uniform HiCache ($O = 2$, $N = 5$)	3.89	2.81	551.53	5.42	0.521	0.510	0.151	13.93	0.607	0.290	27.22	**6.38**	69.69	1.018	0.886
ReCache HiCache ($O = 2$)	3.89	2.81	551.53	5.42	**0.331**	**0.318**	**0.072**	**19.04**	**0.750**	**0.300**	**27.35**	6.34	**70.69**	**1.080**	**0.994**
uniform Taylor-DP ($O = 2$, $N = 5$)	2.09	5.22	551.13	5.43	0.519	0.508	0.151	13.92	0.608	0.291	27.00	6.43	69.52	1.023	0.867
DPCache ($O = 2$, $K = 9$)	2.09	5.22	551.13	5.43	0.433	0.417	**0.091**	17.36	**0.693**	0.299	**27.58**	6.37	70.21	1.112	0.996
ReCache Taylor-DP ($O = 2$)	2.09	5.22	551.13	5.43	**0.410**	**0.398**	**0.091**	**17.52**	0.692	**0.305**	27.20	**6.49**	**70.85**	**1.133**	**1.004**
13 steps															
13 steps of full inference	2.91	3.75	788.55	3.79	0.484	0.472	0.135	14.67	0.640	0.295	27.08	6.48	69.35	1.027	0.931
uniform FORA ($N = 3$)	3.77	2.90	789.51	3.79	0.479	0.468	0.135	14.68	0.639	0.296	**27.19**	6.38	70.12	1.055	0.940
ReCache FORA	3.77	2.90	789.51	3.79	**0.256**	**0.242**	**0.045**	**22.32**	**0.817**	**0.302**	27.13	**6.40**	**70.83**	**1.088**	**0.994**
uniform Δ-DiT ($N = 3$)	5.72	1.91	1523.13	1.96	0.505	0.493	0.149	14.00	0.634	0.283	26.91	6.39	64.83	0.925	0.790
TeaCache	5.72	1.91	1523.13	1.96	0.409	0.398	0.100	16.70	0.697	**0.299**	**27.07**	6.43	**70.45**	1.065	**0.996**
MagCache	5.72	1.91	1523.13	1.96	0.294	0.283	0.058	20.39	0.796	0.298	26.84	**6.55**	69.79	**1.072**	0.995
ReCache Δ-DiT	5.72	1.91	1523.13	1.96	**0.273**	**0.260**	**0.046**	**22.09**	**0.818**	0.295	26.82	6.53	69.46	1.046	0.975
uniform DiCache ($N = 3$)	3.08	3.55	827.78	3.61	0.471	0.461	0.134	14.72	0.645	0.298	27.01	6.47	70.09	1.065	0.965
DiCache	3.08	3.55	827.78	3.61	0.367	0.357	0.090	17.37	0.726	**0.303**	26.98	**6.48**	**71.02**	**1.102**	1.001
ReCache DiCache	3.08	3.55	827.78	3.61	**0.197**	**0.185**	**0.031**	**24.50**	**0.866**	**0.303**	**27.08**	6.44	70.92	1.097	**1.026**
uniform LeMiCa ($N = 3$)	2.91	3.75	789.15	3.79	0.477	0.466	0.135	14.67	0.643	0.295	27.10	6.44	69.54	1.041	0.931
LeMiCa	2.91	3.75	789.15	3.79	0.250	0.236	**0.040**	22.81	0.829	**0.301**	**27.08**	**6.47**	**70.60**	**1.082**	**1.005**
ReCache LeMiCa	2.91	3.75	789.15	3.79	**0.246**	**0.233**	**0.040**	**22.99**	**0.833**	0.300	27.05	**6.47**	70.26	1.078	0.991
uniform TaylorSeer ($O = 1$, $N = 3$)	4.06	2.69	789.54	3.79	0.468	0.457	0.131	14.82	0.643	0.300	27.03	**6.47**	70.62	1.093	0.991
ReCache TaylorSeer ($O = 1$)	4.06	2.69	789.54	3.79	**0.194**	**0.184**	**0.038**	**23.43**	**0.854**	**0.304**	**27.13**	6.38	**71.24**	**1.115**	**1.029**
uniform TaylorSeer ($O = 2$, $N = 3$)	4.21	2.59	789.58	3.79	0.469	0.459	0.132	14.79	0.642	0.300	26.98	**6.48**	70.63	1.092	1.001
ReCache TaylorSeer ($O = 2$)	4.21	2.59	789.58	3.79	**0.195**	**0.186**	**0.040**	**22.95**	**0.853**	**0.306**	**27.10**	6.47	**71.34**	**1.126**	**1.041**
uniform HiCache ($O = 1$, $N = 3$)	4.53	2.41	789.51	3.79	0.470	0.460	0.133	14.77	0.643	0.299	27.13	**6.43**	70.44	1.073	0.964
ReCache HiCache ($O = 1$)	4.53	2.41	789.51	3.79	**0.206**	**0.196**	**0.040**	**22.83**	**0.847**	**0.303**	**27.19**	6.39	**71.03**	**1.097**	**1.027**
uniform HiCache ($O = 2$, $N = 3$)	4.64	2.35	789.51	3.79	0.471	0.460	0.133	14.76	0.642	0.299	27.12	**6.44**	70.43	1.077	0.967
ReCache HiCache ($O = 2$)	4.64	2.35	789.51	3.79	**0.212**	**0.201**	**0.041**	**22.63**	**0.843**	**0.303**	**27.22**	6.39	**71.03**	**1.096**	**1.045**
uniform Taylor-DP ($O = 2$, $N = 3$)	2.92	3.74	789.15	3.79	0.470	0.459	0.133	14.75	0.643	0.300	27.02	6.49	70.41	1.079	0.987
DPCache ($O = 2$, $K = 13$)	2.92	3.74	789.15	3.79	0.276	0.264	0.053	21.03	0.804	**0.307**	27.00	**6.50**	71.13	1.117	**1.046**
ReCache Taylor-DP ($O = 2$, $N = 3$)	2.92	3.74	789.15	3.79	**0.244**	**0.230**	**0.036**	**23.92**	**0.845**	0.306	**27.08**	6.48	**71.16**	**1.123**	1.034

Table 6: Training-set size on FLUX.1-dev with Taylor-DP ($O = 2$). All configurations use 400 optimizer updates; the number of epochs is adjusted as the number of training prompts changes.

Configuration	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
7 steps							
64 × 50 (default)	**0.490**	0.125	15.25	**0.630**	**0.299**	**1.092**	**1.012**
128 × 25	0.502	0.124	**15.29**	**0.630**	0.297	1.068	0.992
640 × 5	0.504	**0.122**	15.17	0.619	**0.299**	1.054	0.950
9 steps							
64 × 50 (default)	0.410	0.091	17.52	0.692	0.305	**1.133**	1.004
128 × 25	0.410	0.088	17.61	0.695	**0.307**	1.130	**1.049**
640 × 5	**0.382**	**0.080**	**18.18**	**0.717**	0.306	1.118	1.025
13 steps							
64 × 50 (default)	0.244	**0.036**	**23.92**	**0.845**	**0.306**	**1.123**	1.034
128 × 25	**0.217**	0.043	22.43	0.842	**0.306**	1.117	1.034
640 × 5	0.277	0.061	20.03	0.788	0.304	1.111	**1.038**

Table 7: Effect of the number of sampled schedules n on ReCache on FLUX.1-dev with Taylor-DP ($O = 2$). All configurations use approximately 19,200 training generations.

Configuration	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
7 steps							
$n = 2$	0.508	0.130	14.92	0.605	0.297	1.062	0.948
$n = 4$	0.520	0.140	14.48	0.598	0.294	1.032	0.945
$n = 6$ (default)	**0.490**	**0.125**	**15.25**	**0.630**	**0.299**	**1.092**	**1.012**
$n = 8$	0.517	0.136	14.72	0.608	0.294	1.053	0.982
9 steps							
$n = 2$	0.412	0.096	17.16	0.684	**0.305**	1.108	**1.014**
$n = 4$	0.407	0.092	17.38	0.692	0.304	1.090	1.011
$n = 6$ (default)	0.410	0.091	17.52	0.692	**0.305**	**1.133**	1.004
$n = 8$	**0.402**	**0.084**	**17.94**	**0.705**	**0.305**	1.118	0.996
13 steps							
$n = 2$	0.234	0.047	21.69	0.826	**0.306**	1.117	1.030
$n = 4$	0.236	0.045	22.19	0.831	0.305	1.113	1.032
$n = 6$ (default)	0.244	**0.036**	**23.92**	0.845	**0.306**	**1.123**	1.034
$n = 8$	**0.196**	0.038	23.23	**0.857**	**0.306**	**1.123**	**1.041**

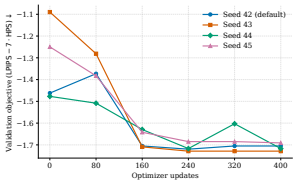
steps are strictly additive: the anchor steps identified at the tightest budget ($k = 7$) remain completely fixed when the budget is expanded to $k = 9$ and $k = 13$. The expanded budgets simply activate new steps in the intermediate gaps without shifting the previously selected ones.

Cross-model schedule transfer. A natural question is whether learned caching schedules are model-specific or reflect general properties of the denoising trajectory. To evaluate this transferability,

Table 8: Sensitivity to optimization seeds on FLUX.1-dev with Taylor-DP ($O = 2$). ReCache results are reported as mean $\pm$ standard deviation across four independently trained policies (seeds 42–45). Fixed baselines are included for reference.

Method	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
7 steps							
uniform	0.554	0.160	13.56	0.591	0.280	0.925	0.727
DPCache	0.546	0.132	14.96	0.594	0.287	1.014	0.931
ReCache	**0.509 ± 0.018**	**0.123 ± 0.003**	**15.27 ± 0.213**	**0.617 ± 0.009**	**0.296 ± 0.004**	**1.079 ± 0.016**	**0.986 ± 0.020**
9 steps							
uniform	0.519	0.151	13.92	0.608	0.291	1.023	0.867
DPCache	0.433	0.091	17.36	0.693	0.299	1.112	0.996
ReCache	**0.401 ± 0.016**	0.092 ± 0.006	17.34 ± 0.512	0.696 ± 0.012	**0.304 ± 0.002**	**1.122 ± 0.008**	1.011 ± 0.021
13 steps							
uniform	0.470	0.133	14.75	0.643	0.300	1.079	0.987
DPCache	0.276	0.053	21.03	0.804	0.307	1.117	1.046
ReCache	**0.235 ± 0.020**	**0.040 ± 0.004**	**23.000 ± 0.670**	**0.839 ± 0.010**	0.306 ± 0.001	1.119 ± 0.003	1.032 ± 0.007

Figure 6: Validation objective LPIPS $-$ 7 HPS during ReCache training on FLUX.1-dev with Taylor-DP ($O = 2$) across four optimization seeds (42–45). Lower values are better.



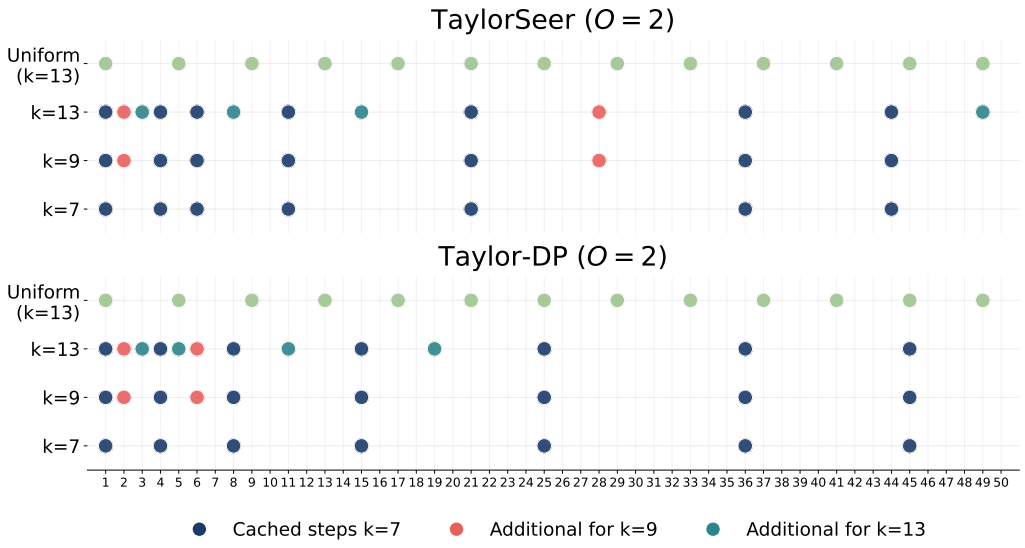


Figure 7: Comparison of cache steps for FLUX.1-dev with the caching mechanisms TaylorSeer ($O = 2$) and Taylor-DP ($O = 2$)

we take a schedule learned by ReCache on Wan2.1 and deploy it directly on HunyuanVideo without retraining, keeping the Taylor-DP ($O = 1$) caching mechanism fixed.

Table 9 shows that the transferred schedule remains competitive on HunyuanVideo and substantially improves over standard uniform scheduling. This suggests that effective caching patterns are not purely model-specific. Instead, ReCache appears to identify structural properties of the denoising trajectory that are shared across DiT-based video generation models.

Sampler shift. Beyond transfer across backbones, we evaluate whether learned schedules can also be reused with a different sampler. In Table 10, we replace Euler with DPM-Solver++ on FLUX.1-dev with Taylor-DP ($O = 2$) caching, reusing the schedules without retraining. ReCache outperforms both uniform scheduling and DPCache on all reported metrics at $k = 7$ and $k = 8$. At $k = 13$, ReCache improves all reported fidelity metrics and ImageReward on DrawBench, while HPS and ImageReward on MS-COCO are slightly lower than those of both baselines.

Reducing the number of backbone inference steps. So far, the main experiments fix the underlying denoising trajectory length N and reduce the number of full model evaluations k through caching. However, ReCache can also be combined with a shorter backbone trajectory. For example, using the TaylorSeer ($O = 1$) mechanism, one can select $k = 13$ full evaluations from an $N = 25$-step trajectory rather than from a 50-step trajectory. Table 11 shows that ReCache still remains effective

Table 9: Cross-model schedule transfer from Wan2.1 to HunyuanVideo with Taylor-DP ($O = 1$) and a 25-step backbone trajectory. The ReCache schedule learned on Wan2.1 is reused without retraining.

Method	Metrics		
	LPIPS↓	HPS↑	VBENCH↑
25 steps	0.000	0.239	79.908
7 steps			
7 steps of full inference	0.524	0.208	75.802
uniform	0.478	0.212	77.048
ReCache	**0.314**	**0.225**	**77.399**
9 steps			
9 steps of full inference	0.494	0.218	77.277
uniform	0.442	0.223	78.477
ReCache	**0.232**	**0.233**	**78.899**
13 steps			
13 steps of full inference	0.451	0.229	78.597
uniform	0.381	0.232	79.507
ReCache	**0.135**	**0.237**	**79.659**

Table 10: Cross-sampler schedule transfer from Euler to DPM-Solver++ on FLUX.1-dev with Taylor-DP ($O = 2$) caching. The schedules obtained with Euler are reused without retraining.

Method	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
7 steps							
uniform	0.567	0.132	14.851	0.548	0.280	0.956	0.953
DPCache	0.572	0.144	14.281	0.563	0.283	0.982	0.910
ReCache	**0.504**	**0.126**	**15.153**	**0.619**	**0.300**	**1.082**	**1.005**
8 steps							
uniform	0.547	0.157	13.663	0.580	0.287	1.004	0.855
DPCache	0.474	0.111	16.042	0.651	0.298	1.095	0.987
ReCache	**0.454**	**0.102**	**16.554**	**0.657**	**0.305**	**1.128**	**1.025**
13 steps							
uniform	0.319	0.063	19.759	0.770	0.307	**1.127**	1.016
DPCache	0.306	0.061	19.961	0.780	**0.308**	**1.127**	1.037
ReCache	**0.293**	**0.047**	**21.936**	**0.809**	0.306	1.122	**1.057**

Table 11: Impact of reducing the number of backbone steps. Results show that ReCache can be applied on both $N = 50$ and $N = 25$ to find more optimal schedules.

Method	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
50 steps	0.000	0.000	∞	1.000	0.306	1.120	1.008
25 steps	0.000	0.000	∞	1.000	0.304	1.092	1.016
8 steps							
uniform (25 steps)	0.611	0.146	14.58	0.543	0.224	0.046	0.015
uniform (50 steps)	0.539	0.156	13.71	0.591	0.288	1.002	0.825
ReCache (25 steps)	0.326	0.065	19.84	0.767	0.295	1.048	0.983
ReCache (50 steps)	0.419	0.087	17.55	0.693	0.302	1.113	0.986
9 steps							
uniform (25 steps)	0.564	0.142	14.72	0.590	0.246	0.496	0.328
uniform (50 steps)	0.518	0.150	13.94	0.605	0.293	1.044	0.921
ReCache (25 steps)	0.231	0.042	22.51	0.833	0.301	1.082	0.994
ReCache (50 steps)	0.338	0.082	17.99	0.735	0.303	1.117	0.997
13 steps							
uniform (25 steps)	0.512	0.130	15.18	0.628	0.281	0.922	0.730
uniform (50 steps)	0.468	0.131	14.82	0.643	0.300	1.093	0.991
ReCache (25 steps)	0.106	0.020	27.86	0.924	0.303	1.087	1.011
ReCache (50 steps)	0.194	0.038	23.43	0.854	0.304	1.115	1.029

Table 12: Comparison of Gumbel-Top-k and Bernoulli selection on FLUX.1-dev for TaylorSeer ($O = 1$). Cells where $|\mathbf{s}| \neq k$ are highlighted in red.

Method	Metrics		
	$\lvert s\rvert$	HPS↑	LPIPS↓
7 steps			
Bernoulli $\lambda_{\text{bern}} = 1$	15	0.306	0.150
Bernoulli $\lambda_{\text{bern}} = 10$	8	0.294	0.427
Gumbel-Top-k	7	**0.296**	**0.499**
9 steps			
Bernoulli $\lambda_{\text{bern}} = 1$	15	0.306	0.150
Bernoulli $\lambda_{\text{bern}} = 10$	9	**0.304**	0.347
Gumbel-Top-k	9	0.303	**0.338**
13 steps			
Bernoulli $\lambda_{\text{bern}} = 1$	15	0.306	0.150
Bernoulli $\lambda_{\text{bern}} = 10$	13	0.303	0.242
Gumbel-Top-k	13	**0.304**	**0.194**

even with fewer backbone steps. This setting further reduces latency because both cache and reuse evaluations are performed over a shorter denoising trajectory.

C.4 Additional ablations

This section provides an extended ablation study on the design choices of ReCache. All experiments are conducted on the text-to-image generative model FLUX.1-dev with the TaylorSeer ($O = 1$) caching strategy.

Gumbel-Top-k vs. Bernoulli schedule selection. ReCache uses Gumbel-Top-k to draw a sample of exactly k inference steps, which strictly respects the budget without any auxiliary penalty. An alternative is to use a Bernoulli distribution: each step i is included independently with probability $p_i = \sigma(\theta_i)$ (similar to (Zhang et al., 2025b)), and at inference we keep i if $p_i \geq 0.5$. Because this does not guarantee a fixed budget, we add a penalty $\left(\left||\mathbf{s}| - k\right|\right)/N$ to the training objective $\mathcal{L}_{\text{reg}}$ with coefficient λ_{bern}.

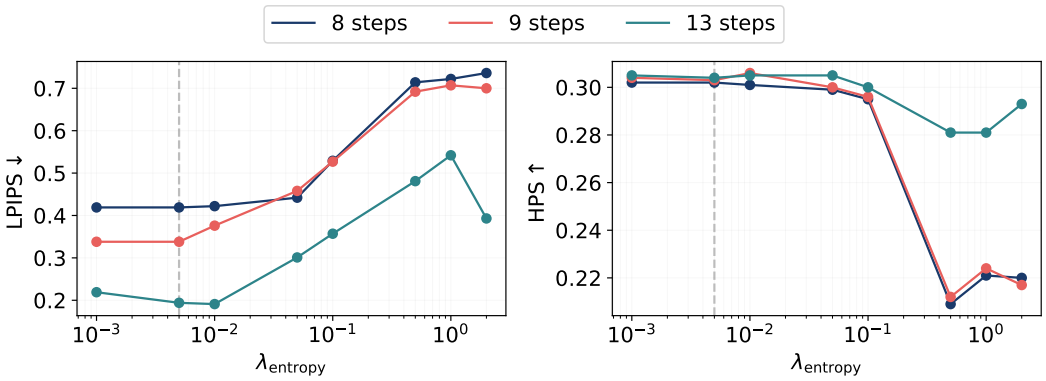


Figure 8: Effect of the entropy regularization coefficient λ_{entropy} on ReCache. Increasing entropy encourages the discovery of more optimal schedules and is particularly beneficial under looser budget constraints; however, excessively high values can slow convergence.

Table 12 shows that the Bernoulli alternative either violates the target budget or requires tuning of λ_{bern} to approximately meet it (e.g., $\lambda_{\text{bern}} = 1$ still violates it). At matched budgets ($\lambda_{\text{bern}} = 10$), Gumbel-Top-$k$ improves LPIPS at $k = 9$ and both reported metrics at $k = 13$. ReCache achieves both strict budget compliance and high-quality results without introducing any budget-related hyperparameter.

Entropy regularization. The entropy regularizer encourages the policy to explore a broader set of caching schedules during training. Figure 8 ablates the coefficient λ_{entropy}. It is especially helpful for looser budget constraints like $k = 13$, where the space of possible schedules is broader and benefits from exploration. The benefit has limits: too-large coefficients slow convergence. In our main FLUX experiments we set $\lambda_{\text{entropy}} = 0.005$ and anneal it to zero, which lets the policy explore freely early in training and converge to a high-quality schedule afterwards.

Image reward loss coefficient. We also provide extended results of the α_{iq} values, which controls the relative importance of the HPS term in the training loss. An extensive discussion can be found in Section 4.3.

D Computational resources

In Table 14, we report the training cost of ReCache across different diffusion backbones and caching mechanisms on NVIDIA H100 GPUs. We report total GPU hours and memory usage per GPU in GB. The training configurations are described in Section B.4.

Table 13: Loss coefficient α_{iq} ablation. The experiments are conducted on FLUX.1-dev with TaylorSeer ($O = 1$) caching mechanisms. We use $\alpha_{\text{iq}} = 7$ as the default in our experiments.

Method	Metrics						
	LPIPS↓	L1↓	PSNR↑	SSIM↑	HPS↑	IR↑	IR DB↑
8 steps							
uniform	0.539	0.156	13.71	0.591	0.288	1.002	0.825
ReCache ($\alpha_{\text{iq}} = 0$)	0.391	0.086	17.28	0.700	0.291	1.065	0.973
ReCache ($\alpha_{\text{iq}} = 1$)	0.398	0.099	16.73	0.691	0.294	1.075	0.974
ReCache ($\alpha_{\text{iq}} = 5$)	0.419	0.087	17.55	0.693	0.302	1.113	0.986
ReCache ($\alpha_{\text{iq}} = 7$)	0.419	0.087	17.55	0.693	0.302	1.113	0.986
ReCache ($\alpha_{\text{iq}} = 10$)	0.419	0.087	17.55	0.693	0.302	1.113	0.986
ReCache (only HPS)	0.535	0.127	14.99	0.603	0.295	1.062	0.968
9 steps							
uniform	0.518	0.150	13.94	0.605	0.293	1.044	0.921
ReCache ($\alpha_{\text{iq}} = 0$)	0.341	0.077	18.15	0.739	0.303	1.117	1.016
ReCache ($\alpha_{\text{iq}} = 1$)	0.366	0.093	17.23	0.715	0.302	1.099	1.000
ReCache ($\alpha_{\text{iq}} = 5$)	0.338	0.082	17.99	0.735	0.303	1.117	0.997
ReCache ($\alpha_{\text{iq}} = 7$)	0.338	0.082	17.99	0.735	0.303	1.117	0.997
ReCache ($\alpha_{\text{iq}} = 10$)	0.378	0.082	17.99	0.718	0.307	1.131	0.992
ReCache (only HPS)	0.504	0.124	15.19	0.619	0.300	1.072	0.982
13 steps							
uniform	0.468	0.131	14.82	0.643	0.300	1.093	0.991
ReCache ($\alpha_{\text{iq}} = 0$)	0.187	0.035	23.94	0.864	0.305	1.115	1.031
ReCache ($\alpha_{\text{iq}} = 1$)	0.200	0.040	22.67	0.847	0.305	1.113	1.028
ReCache ($\alpha_{\text{iq}} = 5$)	0.194	0.038	23.43	0.854	0.304	1.115	1.029
ReCache ($\alpha_{\text{iq}} = 7$)	0.194	0.038	23.43	0.854	0.304	1.115	1.029
ReCache ($\alpha_{\text{iq}} = 10$)	0.203	0.042	22.71	0.847	0.306	1.127	1.054
ReCache (HPS only)	0.448	0.101	16.67	0.675	0.302	1.115	0.999

Table 14: Training cost of ReCache across diffusion backbones and caching mechanisms on NVIDIA H100 GPUs. We report total GPU hours and memory usage per GPU (GB).

Caching Method	GPU hours	GPU memory (GB)
FLUX.1-dev		
FORA	20	49.50
Δ-DiT	31	39.50
DiCache	16	39.60
TaylorSeer ($O = 1$)	22	60.81
TaylorSeer ($O = 2$)	23	72.06
HiCache ($O = 1$)	24	60.81
HiCache ($O = 2$)	25	72.06
Taylor-DP ($O = 2$)	16	39.50
Wan2.1		
TaylorSeer ($O = 1$)	242	62.57
HiCache ($O = 1$)	243	62.57
Taylor-DP ($O = 1$)	230	29.17
HunyuanVideo		
TaylorSeer ($O = 1$)	496	75.78
HiCache ($O = 1$)	498	75.78
Taylor-DP ($O = 1$)	424	60.84