MORCA: OFFLINE-TO-ONLINE REINFORCEMENT LEARNING FOR ADAPTIVE CACHE REUSE IN VIDEO DIFFUSION ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers (DiTs) achieve remarkable performance in video synthesis, but their iterative denoising process suffers from high inference latency. To address this, caching has emerged as an effective acceleration strategy by capitalizing on inter-step redundancy during denoising. Existing dynamic caching methods typically estimate the error that cache reuse would introduce at each denoising step (step error) to guide cache decisions, whereas our concern is how much quality loss cache reuse would cause in the final generated video (terminal error). We show that step error does not directly correspond to terminal error and that latent information helps capture their relationship, thereby informing cache decisions. Moreover, existing threshold-based methods cannot provide precise speedup control, making it difficult to meet practical requirements for user-specified acceleration targets. To address these limitations, we introduce **MORCA**, a cache scheduling framework trained through offline-to-online reinforcement learning to make latent-aware reuse/recompute decisions under user-specified acceleration targets. Extensive experiments on different video generation models across multiple target acceleration ratios demonstrate that MORCA achieves better generation fidelity than state-of-the-art caching methods under comparable computational budgets.

1 INTRODUCTION

Diffusion Transformers (DiTs) (Peebles & Xie, 2023) combined with Flow Matching paradigms (Lipman et al., 2023; Liu et al., 2023) have demonstrated remarkable capabilities in high-fidelity visual generation (Wan et al., 2025; Kong et al., 2025). Despite this progress, the multi-step iterative denoising process incurs substantial computational cost and high inference latency, particularly for video synthesis. To address this bottleneck, various acceleration approaches have emerged, including distillation, quantization, sparse attention, and caching. Among these, distillation requires additional training of the generative model, while quantization and sparse attention typically rely on specialized kernels. Caching offers a lightweight alternative with flexible speed–quality trade-offs.

Cache-based methods exploit inter-step redundancy by skipping the computation of the entire denoising network or selected components (typically the DiT blocks) at some denoising steps and reusing the corresponding results cached at previous steps. Early methods adopt fixed cache schedules (Ma et al., 2024; Li et al., 2024; Zhao et al., 2025b; Liu et al., 2025b) (e.g., recomputing only at fixed intervals), which do not account for runtime denoising dynamics. Dynamic caching methods (Liu et al., 2025a; Kahatapitiya et al., 2025; Chung et al., 2026; Bu et al., 2026) attempt to estimate the error that cache reuse would introduce at each denoising step and use these estimates to make cache decisions through threshold-based rules.

However, this paradigm relies exclusively on the immediate error introduced by cache reuse (**step error**) to make decisions, whereas our concern is the quality loss in the final generated video (**terminal error**). How the former translates into the latter through subsequent denoising steps remains unclear. To investigate this relationship, we conduct a series of experiments. As shown in Fig. 1, step error exhibits similar patterns across these prompts, while terminal error varies substantially. Quantitative results in Table 1 further show that step error alone is insufficient to accurately predict terminal error.

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

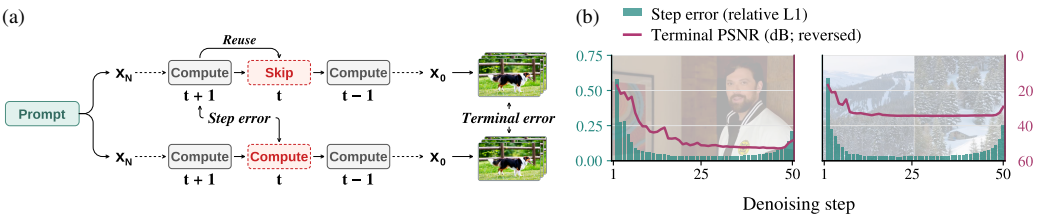

Figure 1: **Step error and terminal error.** (a) Following SeaCache, we cache and reuse the residuals of the DiT blocks. For each step, we evaluate two cases: (i) reusing the residual cached at the previous step and (ii) recomputing the DiT blocks, with all other steps computed normally. We measure step error as the relative L1 difference between the cached and recomputed residuals, and terminal error using the PSNR of the resulting video with cache reuse relative to the no-cache reference. (b) Step error does not directly correspond to terminal error. Results are shown for Wan2.1.

We propose that latents may contain information that helps capture how step error translates into terminal error, thus informing cache decisions. To test this idea, we train a predictor to estimate terminal error from step error. As reported in Table 1, incorporating lightweight features extracted from the input latents at the current and previous steps (see Sec. 3.2) substantially improves prediction accuracy.

Table 1: **Latent features improve terminal-error prediction.** Terminal error is measured using PSNR. Results on Wan2.1.

Input	RMSE (dB) ↓	Spearman ↑
Step error	6.971	0.5288
Step error + step index	4.421	0.7528
Step error + latent	2.849	0.9357
Step error + latent + step index	2.832	0.9361

Motivated by this finding, we train a latent-aware scheduler to make terminal-error-guided cache decisions. We provide the scheduler with SeaCache's proxy signal for step error (see Sec. 3.1) and latent features to help bridge the gap between step error and terminal error. Given the sequential nature of cache scheduling, we formulate it as a Markov decision process (MDP) and optimize the scheduler using reinforcement learning. In practical applications, users often require a specific acceleration ratio. While existing threshold-based caching methods offer only approximate speedup control through threshold adjustment, we use the observed relationship between speedup and the number of reuse steps (Fig. 3) to translate the target acceleration ratio into a reuse budget constraint, then reduce the resulting constrained MDP to an equivalent standard augmented MDP (Sec. 3.2).

Since collecting training data requires costly inference runs, we adopt an offline-to-online reinforcement learning framework based on IQL (Kostrikov et al., 2022) to train the scheduler. As an off-policy algorithm, IQL offers higher data utilization than on-policy methods. Offline training provides an effective and stable initialization from limited data, while online fine-tuning allows the scheduler to explore under its current policy and further improve performance. Our contributions are summarized as follows:

- We propose **MORCA**, a latent-aware cache scheduling framework trained through offline-to-online reinforcement learning to make adaptive cache decisions under user-specified speedups.
- We design a **budget-constrained** and **latent-aware** scheduler that can make terminal-error-guided cache decisions while precisely controlling the acceleration ratio.
- We introduce an **offline-to-online** RL paradigm for scheduler optimization: the offline stage provides an effective initialization, while online fine-tuning further improves its performance.
- Extensive experiments on different video generation models demonstrate that MORCA achieves better speed–quality trade-offs than state-of-the-art caching methods.

2 Related Work

2.1 Diffusion Inference Acceleration

Existing approaches to accelerating diffusion inference mainly include fast sampling, distillation, quantization, sparse attention, and caching. Fast samplers and numerical solvers reduce model eval-

uations but may sacrifice quality under aggressive step reduction (Song et al., 2021; Liu et al., 2022; Lu et al., 2022; 2025; Zhao et al., 2023). Distillation enables few-step or one-step generation but requires additional training (Salimans & Ho, 2022; Song et al., 2023; Luo et al., 2023; Yin et al., 2024). Quantization reduces the precision of weights and activations (Shang et al., 2023; Li et al., 2023; He et al., 2023; Zhao et al., 2025a), while sparse attention restricts attention computation to selected token pairs (Xi et al., 2025). Both effectively lower per-step computation but typically rely on specialized kernels to realize inference speedups.

2.2 Caching-based Diffusion Model Acceleration

Cache-based acceleration methods exploit inter-step redundancy by skipping computation in all or part of the denoising network at selected steps and reusing previously cached results. These methods can be characterized along three complementary dimensions. *Caching Granularity* determines what is cached and reused, including layer or DiT-block outputs (Wimbauer et al., 2024; Chen et al., 2024), attention outputs (Zhao et al., 2025b), CFG branch outputs (Lyu et al., 2025), and finer-grained token features (Zou et al., 2025). *Cache Reuse Strategy* determines how cached features are utilized, evolving from direct reuse (Ma et al., 2024; Zhao et al., 2025b) to feature correction or approximation (Lyu et al., 2025; Bu et al., 2026), and further to explicit feature forecasting (Liu et al., 2025b). *Caching Schedule* determines which steps reuse cached results and which recompute. Early methods typically employ fixed schedules with predefined recomputation steps (Ma et al., 2024; Li et al., 2024; Zhao et al., 2025b; Liu et al., 2025b), which do not adapt to runtime denoising dynamics. Dynamic methods instead estimate the error that cache reuse would introduce at each step and apply threshold-based rules to guide cache decisions. TeaCache uses timestep-modulated input differences as a proxy for output variation, AdaCache further incorporates motion-aware information, DiCache estimates caching errors through shallow-layer probes, and SeaCache derives a spectral-evolution-aware cache signal by suppressing noise-dominated components (Liu et al., 2025a; Kahatapitiya et al., 2025; Bu et al., 2026; Chung et al., 2026). Recent works have also explored learning-based cache scheduling (Zhao et al., 2026b; Xie et al., 2026; Zhao et al., 2026a), which we discuss and compare in detail in Sec. 2.3.

2.3 Reinforcement-Learning-Based Cache Scheduling

Cache scheduling is naturally a sequential decision problem suited to reinforcement learning. RAPID^3 is the first work to formulate cache scheduling as an MDP and train a scheduler through reinforcement learning to make cache-reuse decisions at each denoising step (Zhao et al., 2026b). Concurrent with our work, BAG and OnlineCache explore different approaches to learning cache schedules. BAG constructs a high-quality schedule dataset through expensive offline search and distills these schedules into a lightweight gating network for per-step decisions (Zhao et al., 2026a). OnlineCache uses REINFORCE to train a latent-aware scheduler, but its on-policy RL paradigm limits data reuse and training flexibility. Moreover, its reward-based speedup control cannot guarantee meeting user-specified acceleration targets (Xie et al., 2026).

3 Method

In this section, we present MORCA, an offline-to-online reinforcement learning framework for budget-constrained and latent-aware cache scheduling in video diffusion acceleration, as illustrated in Fig. 2. Building upon SeaCache (Sec. 3.1), MORCA retains its caching granularity and cache reuse strategy while replacing its threshold-based decision rule with a learned scheduler. In addition to SeaCache's signals and related statistics, the scheduler incorporates latent features and reuse budget dynamics to make terminal-error-guided cache decisions under user-specified acceleration targets (Sec. 3.2). We optimize the scheduler through a two-stage offline-to-online IQL paradigm to achieve both high sample efficiency and strong performance (Sec. 3.3).

3.1 Preliminaries

Cache-Based Inference. We first introduce the standard denoising process, followed by a typical cache-based inference pipeline. Given a text condition c, video generation proceeds through N denoising steps. At step t, the input latent x_t passes through a DiT denoiser comprising an input

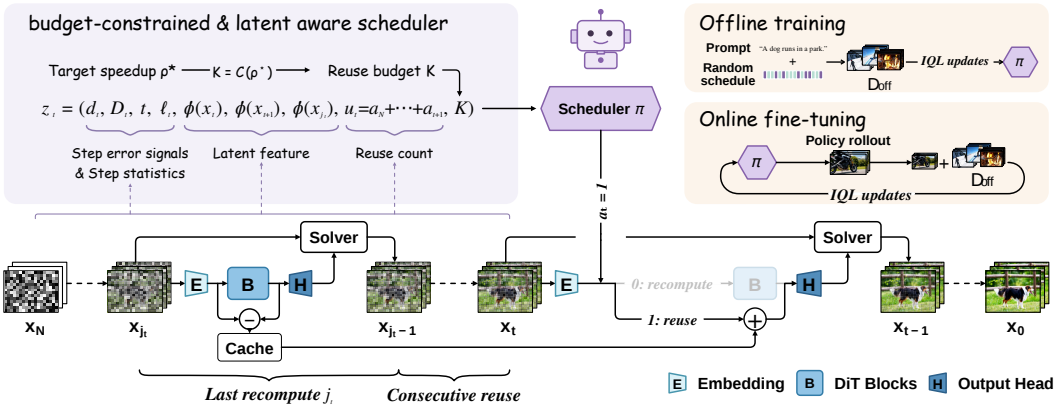

Figure 2: **Overview of MORCA.** Bottom: A learned scheduler makes reuse/recompute decisions at each denoising step. Top left: The scheduler is budget-constrained and latent-aware. The target speedup is mapped to the required total number of reuse steps, which, together with step-error proxies and step statistics, latent features, and the current reuse count extracted from the denoising state, is fed into the scheduler for decision-making (Sec. 3.2). Top right: The scheduler is optimized through offline-to-online reinforcement learning using IQL (Sec. 3.3).

embedding module E_θ, a stack of DiT blocks B_θ, and an output head H_θ:

$$h_t = E_\theta(x_t), \qquad y_t = B_\theta(h_t, t, c), \qquad o_t = H_\theta(y_t, t). \tag{1}$$

The latent is updated as $x_{t-1} = \mathcal{U}_t(x_t, o_t)$, where $\mathcal{U}_t$ denotes the diffusion- or flow-based sampling update rule. With classifier-free guidance (CFG), o_t is replaced by $o_t^{\mathrm{CFG}} = o_t^{\mathrm{u}} + w(o_t - o_t^{\mathrm{u}})$, where o_t^{u} is computed using the empty or negative text condition c_{u}, and w is the guidance scale. After the final denoising step, the latent x_0 is decoded into a video by the VAE decoder.

Cache-based inference decides whether to recompute or reuse at each denoising step. A binary action $a_t \in \{0, 1\}$ determines whether to skip the DiT blocks and directly reuse a previously cached residual ($a_t = 1$) or compute the blocks normally ($a_t = 0$). The block residual at step t is defined as $\Delta_t = y_t - h_t$. Accordingly,

$$h_t = E_\theta(x_t), \qquad \hat{y}_t = \begin{cases} B_\theta(h_t, t, c), & a_t = 0, \\ h_t + \Delta_{j_t}, & a_t = 1, \end{cases} \qquad \hat{o}_t = H_\theta(\hat{y}_t, t), \tag{2}$$

where $j_t = \min\{j > t : a_j = 0\}$ is the most recent recomputation step.

Under CFG, the two branches typically share a_t but maintain separate residual caches. Their outputs are combined using the same guidance rule for the sampling update. The complete cache schedule is $\mathbf{a} = (a_N, \ldots, a_1)$.

SeaCache. SeaCache (Chung et al., 2026) applies a Spectral-Evolution-Aware (SEA) filter to the timestep-modulated input latent and uses the filtered representation, denoted by $\widetilde{x}_t$, to construct a lightweight proxy signal for estimating the relative change between adjacent steps:

$$d_t = \frac{\|\widetilde{x}_t - \widetilde{x}_{t+1}\|_1}{\|\widetilde{x}_{t+1}\|_1 + \epsilon}. \tag{3}$$

It further computes the accumulated distance since the most recent recomputation step as $D_t = \sum_{i=t}^{j_t - 1} d_i$ and determines the action using a threshold δ: $a_t^{\mathrm{SC}} = \mathbb{I}[D_t \le \delta]$.

3.2 Budget-Constrained and Latent-Aware Scheduler

We formulate cache scheduling as a budget-constrained MDP and reduce it to an equivalent standard augmented MDP compatible with standard RL optimization. We then introduce a compact latent-aware state representation for efficient scheduler implementation.

Constrained MDP Formulation. Cache scheduling is inherently a sequential decision problem, as each recompute/reuse decision affects subsequent denoising states and therefore has long-term consequences. We formulate it as a finite-horizon MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, N)$. At each step t, $s_t \in \mathcal{S}$ denotes the denoising state, and $a_t \in \mathcal{A} = \{0, 1\}$ denotes the action, with 0 corresponding to recomputation and 1 to cache reuse. The transition dynamics P describe how the denoising state evolves under the selected action through the corresponding denoising update. We define the reward R in terms of the generated video's quality, measured by its PSNR relative to the no-cache reference $y^\star$ generated with the same prompt and random seed:

$$r_t = \begin{cases} 0, & 1 < t \le N, \\ \min\{\mathrm{PSNR}(y, y^\star),\, r_{\max}\}, & t = 1, \end{cases} \qquad R(\tau) = \sum_{t=1}^{N} r_t. \tag{4}$$

where y denotes the video generated under the cache schedule and $r_{\max}$ is a fixed finite cap on PSNR. The horizon N is the number of denoising steps.

Given a target acceleration ratio $\rho^\star$, we further constrain the computation budget. Since all denoising steps share the same backbone, each reuse operation yields approximately the same computational saving. Empirically, inference latency decreases approximately linearly with the reuse count K and is similar across prompts. Combined with $\rho = T_{\text{full}}/T_{\text{inf}}$, this yields $K \approx \alpha - \beta/\rho$ (Fig. 3). We therefore calibrate the target acceleration ratio to a reuse budget $K = \mathcal{C}(\rho^\star)$ and formulate cache scheduling as a constrained MDP. We seek to learn a scheduler policy π that maximizes the expected return while satisfying the prescribed reuse budget:

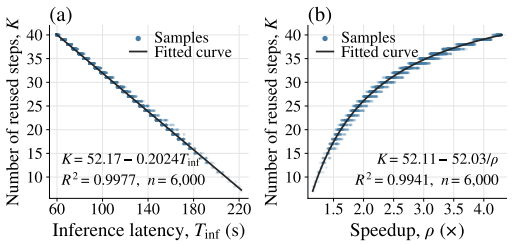

Figure 3: **Calibration of reuse budget K against (a) inference latency and (b) speedup on Wan2.1.** The fitted functions take the forms (a) $K = \alpha - \beta T_{\text{inf}}$ and (b) $K = \alpha - \beta/\rho$.

$$\max_{\pi} \; \mathbb{E}_{\tau\sim\pi}[R(\tau)] \qquad \text{s.t.} \qquad \sum_{t=1}^{N} a_t = K. \tag{5}$$

Reduction to a Standard MDP. However, RL algorithms generally do not directly guarantee constraint satisfaction. To make the constrained scheduling problem compatible with standard RL optimization, we transform it into an equivalent augmented MDP following the reduction approach of McMahan & Zhu (2024). Let $u_t = \sum_{i=t+1}^{N} a_i$ denote the number of reuse actions already taken and $b_t = K - u_t$ the remaining budget. Since the optimal decision depends on both the current denoising state and the remaining budget, we augment the state as

$$\bar{s}_t = (s_t, u_t, K). \tag{6}$$

We further restrict the action set to prevent decisions that make the prescribed reuse budget unattainable. After taking action a_t, the remaining budget $b_t - a_t$ must be achievable within the remaining $t - 1$ denoising steps. We therefore define the feasible action set as

$$\bar{\mathcal{A}}_t(\bar{s}_t) = \{a \in \{0, 1\} \mid 0 \le b_t - a \le t - 1\}. \tag{7}$$

This construction yields a standard augmented MDP $\bar{\mathcal{M}}$ that preserves the optimal feasible value and guarantees the prescribed reuse budget. A formal proof is provided in Appendix D.

Latent-Aware State Representation. In the formulation and reduction above, $\bar{s}_t$ denotes the full denoising state needed to determine transition and reward distributions. In practice, we use a simplified representation of $\bar{s}_t$ as the scheduler input to control computational overhead. Recall from Sec. 1 that our scheduler aims to make terminal-error-guided decisions. Existing methods already provide effective proxy signals for estimating step error, but this local error does not directly correspond to terminal error (Fig. 1). Latent information helps bridge this gap (Table 1).

Our scheduler input therefore consists of three components. (i) Step-error proxies and related statistics: we retain signals and statistics associated with SeaCache's cache decisions, including the

adjacent-step distance d_t, accumulated distance D_t, current denoising step t, and number of consecutive reuse steps $\ell_t = j_t - t - 1$. (ii) Latent information: each latent has shape $C \times T \times H \times W$, where C, T, H, and W denote the channel count, latent frame count, height, and width, respectively. To control computational overhead, we extract compact features instead of using the full latent. We apply 3D pooling to each latent and 2D pooling to its temporal mean and variance. Denoting these pooling operators by $\mathcal{P}_3$ and $\mathcal{P}_2$, and the temporal mean and variance by $\mu_T(x)$ and $\sigma_T^2(x)$, respectively, we define $\phi(x) = \big(\mathcal{P}_3(x),\ \mathcal{P}_2(\mu_T(x)),\ \mathcal{P}_2(\sigma_T^2(x))\big)$. We use ϕ to extract features from latents relevant to the current denoising state: the current latent x_t, the previous-step latent x_{t+1}, and the latent at the most recent recomputation step x_{j_t}. (iii) Budget information: following the reduction above, we include u_t and K to handle the budget constraint. Together, these components form the compact scheduler input:

$$z_t = (d_t, D_t, t, \ell_t, \phi(x_t), \phi(x_{t+1}), \phi(x_{j_t}), u_t, K)\,. \tag{8}$$

3.3 Offline-to-Online Reinforcement Learning Paradigm

Since collecting cache-scheduling trajectories requires expensive video-generation rollouts, we adopt a two-stage offline-to-online reinforcement learning paradigm based on Implicit Q-Learning (IQL) (Kostrikov et al., 2022) to achieve both high sample efficiency and strong performance. As an off-policy algorithm, IQL enables repeated reuse of collected data across optimization updates, unlike on-policy methods such as REINFORCE, which require fresh rollouts under the current policy. In the offline stage, the scheduler learns from a relatively small set of pre-collected random trajectories, providing an effective policy initialization. Starting from this initialization, online fine-tuning enables further exploration and policy improvement.

Offline Training. We randomly sample cache schedules under different prompts and target acceleration ratios, run the video-generation model with these schedules, and convert the resulting trajectories into transition tuples (z_t, a_t, r_t, z_{t-1}) to construct the training dataset $\mathcal{D}_{\text{off}}$. Further details of dataset construction are provided in Appendix B.3.

As illustrated in Algorithm 1(a), IQL maintains a value network V_ψ, two Q-networks Q_{θ_1} and Q_{θ_2}, and their corresponding target networks $Q_{\bar{\theta}_1}$ and $Q_{\bar{\theta}_2}$. The target networks track the Q-networks through exponential moving average (EMA) updates. IQL trains the Q-functions against temporal-difference targets and learns the value function through expectile regression on the target Q-values:

$$\mathcal{L}_Q(\theta_i) = \mathbb{E}_{(z_t,a_t,r_t,z_{t-1})\sim\mathcal{D}}\left[\left(r_t + \gamma V_\psi(z_{t-1}) - Q_{\theta_i}(z_t,a_t)\right)^2\right], \qquad i = 1, 2. \tag{9}$$

$$\mathcal{L}_V(\psi) = \mathbb{E}_{(z_t,a_t)\sim\mathcal{D}}\left[\ell_\tau\left(\bar{Q}(z_t,a_t) - V_\psi(z_t)\right)\right]. \tag{10}$$

where $\ell_\tau(u) = |\tau - \mathbb{I}[u < 0]|u^2$ is the expectile loss, $\bar{Q}(z_t, a_t) = \min_{i\in\{1,2\}} Q_{\bar{\theta}_i}(z_t, a_t)$, and γ is the discount factor. The scheduler policy is then learned through advantage-weighted behavior cloning, assigning larger weights to actions with higher estimated advantages:

$$\mathcal{L}_\pi(\varphi) = -\mathbb{E}_{(z_t,a_t)\sim\mathcal{D}}\left[\exp\left(\beta A(z_t,a_t)\right)\log\pi_\varphi(a_t \mid z_t)\right], \tag{11}$$

where $A(z_t, a_t) = \bar{Q}(z_t, a_t) - V_\psi(z_t)$, and β controls advantage weighting. For further details of IQL, we refer readers to Kostrikov et al. (2022).

By learning preferentially from higher-advantage actions in a relatively small dataset of randomly sampled trajectories, the offline stage provides a good initial policy without requiring prior knowledge of effective cache schedules. In contrast, starting online training from scratch may incur substantial unnecessary data collection and training costs under an initially weak policy.

Online Fine-Tuning. Starting from the offline-trained scheduler, we perform online fine-tuning as shown in Algorithm 1(b). In each round, we collect a batch of online trajectories under randomly sampled prompts and target acceleration ratios, sampling each step's cache decision from the current scheduler policy $\pi_\varphi(\cdot \mid z_t)$. These trajectories are added to the replay buffer alongside the offline and previously collected online data. We first perform a short critic-only warm-up with the policy fixed to adapt the Q- and V-networks to the new data, then jointly train the value functions and policy using the same IQL procedure as in offline training. Sampling trajectories from the current policy enables further exploration, while retaining the offline data in the replay buffer helps stabilize learning during fine-tuning.

Algorithm 1 Offline-to-Online IQL Training for MORCA

(a) Offline training

Randomly initialize $\psi, \theta_1, \theta_2, \varphi$
$\bar{\theta}_i \leftarrow \theta_i, \quad i = 1, 2$
$\mathcal{D} \leftarrow \mathcal{D}_{\text{off}}$
for each gradient step **do**
    CRITICUPDATE
    ACTORUPDATE
end for

Shared updates

procedure CRITICUPDATE
$\psi \leftarrow \psi - \lambda_V \nabla_\psi \mathcal{L}_V(\psi)$
$\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} \mathcal{L}_Q(\theta_i), \quad i = 1, 2$
$\bar{\theta}_i \leftarrow \eta \bar{\theta}_i + (1 - \eta)\theta_i, \quad i = 1, 2$
end procedure
procedure ACTORUPDATE
$\varphi \leftarrow \varphi - \lambda_\pi \nabla_\varphi \mathcal{L}_\pi(\varphi)$
end procedure

(b) Online fine-tuning

$\mathcal{D} \leftarrow \mathcal{D}_{\text{off}}$
Initialize $\psi, \theta_1, \theta_2, \bar{\theta}_1, \bar{\theta}_2, \varphi$ from the offline training results
for $k = 1, \ldots, N_{\text{on}}$ **do**
    Collect new data $\Delta\mathcal{D}$ using π_φ
    $\mathcal{D} \leftarrow \mathcal{D} \cup \Delta\mathcal{D}$
    for each gradient step **do** ▷ warm-up
        CRITICUPDATE
    end for
    for each gradient step **do** ▷ joint
        CRITICUPDATE
        ACTORUPDATE
    end for
end for
return π_φ

N_{on} is the number of online rounds; $\lambda_V, \lambda_Q, \lambda_\pi$ are learning rates; η is the target-network EMA decay.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Model configurations. We evaluate two pretrained text-to-video models from the Wan family (Wan et al., 2025): Wan2.2-T2V-A14B on NVIDIA RTX A6000 GPUs and Wan2.1-T2V-1.3B on NVIDIA GeForce RTX 4090 GPUs. Both use 50 denoising steps; other inference settings are provided in Appendix A.

Training configurations. We collect 2,500 offline trajectories from 100 prompts for Wan2.2 (25 schedules per prompt), with target speedups in $[1.5\times, 3.5\times]$, and 4,000 from 1,000 prompts for Wan2.1 (four schedules per prompt), with target speedups in $[2.0\times, 4.5\times]$. Offline IQL runs for 200 epochs with a learning rate of 10^{-4} for all networks. Online fine-tuning runs for 16 rounds, each adding 100 trajectories and performing five epochs of critic-only warm-up followed by 20 epochs of joint training, with learning rates of 4×10^{-5} for the policy network and 10^{-4} for the Q and V networks. Both stages use the AdamW optimizer and a batch size of 256. Further details are provided in Appendix B.

Baseline configurations. We compare against four representative caching-based acceleration methods: TeaCache (Liu et al., 2025a), MagCache (Ma et al., 2025), DiCache (Bu et al., 2026), and SeaCache (Chung et al., 2026). For Wan2.2, only MagCache has an official implementation; we port the others from Wan2.1 with model-specific adaptations. We use their official Wan2.1 implementations directly. Implementation details are provided in Appendix C.

Evaluation protocol. We sample 200 prompts from VBench (Huang et al., 2024) as a fixed evaluation set and measure visual quality using PSNR, SSIM, and LPIPS against no-cache references generated with the same prompts and random seeds. We report inference latency. We compare methods at approximately $1.8\times$, $2.4\times$, and $3.0\times$ speedup for Wan2.2, and $2.4\times$, $3.0\times$, and $4.0\times$ for Wan2.1. We use lower target speedups for Wan2.2 to preserve generation fidelity, as it is more sensitive to cache reuse.

4.2 QUANTITATIVE COMPARISON

Table 2 compares MORCA with existing caching methods at comparable speedups. Since MORCA builds on SeaCache by replacing its threshold-based scheduling with a learned scheduler, we focus on this comparison to demonstrate the effectiveness of our learned scheduling policy. On Wan2.2, MORCA achieves the best PSNR, SSIM, and LPIPS at all three target speedups, with PSNR gains over SeaCache of 2.22, 1.86, and 1.78 dB at $1.8\times$, $2.4\times$, and $3.0\times$, respectively. On Wan2.1, it achieves the highest PSNR and SSIM at all three targets, improving PSNR over SeaCache by 1.05,

Table 2: **Quantitative comparison with baselines.** The best result is highlighted in bold, while the second-best result is underlined.

Model	Target	Method	Latency (s)↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
Wan2.2	—	Original	990.56	1.00×	—	—	—
	1.8×	TeaCache	562.63	1.76×	17.04	0.5984	0.2882
		MagCache	544.78	1.82×	16.41	0.5662	0.3041
		DiCache	551.29	1.80×	18.51	0.6516	0.2232
		SeaCache	546.99	1.81×	<u>19.91</u>	<u>0.7016</u>	<u>0.1888</u>
		Ours	549.19	1.80×	**22.13**	**0.7507**	**0.1681**
	2.4×	TeaCache	415.21	2.39×	15.31	0.5325	0.3658
		MagCache	417.22	2.37×	15.31	0.5185	0.3632
		DiCache	416.43	2.38×	16.51	0.5759	0.2923
		SeaCache	419.76	2.36×	<u>17.38</u>	<u>0.6074</u>	<u>0.2840</u>
		Ours	421.15	2.35×	**19.24**	**0.6516**	**0.2513**
	3.0×	TeaCache	322.88	3.07×	14.40	0.4913	0.4211
		MagCache	332.74	2.98×	13.41	0.4336	0.4839
		DiCache	340.50	2.91×	<u>15.54</u>	<u>0.5259</u>	<u>0.3623</u>
		SeaCache	320.41	3.09×	14.99	0.5151	0.3941
		Ours	326.93	3.03×	**16.77**	**0.5639**	**0.3435**
Wan2.1	—	Original	255.94	1.00×	—	—	—
	2.4×	TeaCache	106.82	2.40×	20.12	0.7361	0.1596
		MagCache	107.54	2.38×	22.31	0.8075	0.1198
		DiCache	108.27	2.36×	22.28	0.8139	**0.0992**
		SeaCache	109.00	2.35×	<u>22.86</u>	<u>0.8174</u>	0.1131
		Ours	109.68	2.33×	**23.91**	**0.8279**	<u>0.1073</u>
	3.0×	TeaCache	87.71	2.92×	17.39	0.6199	0.2674
		MagCache	86.09	2.97×	<u>20.04</u>	0.6888	0.1910
		DiCache	86.82	2.95×	19.69	<u>0.7170</u>	<u>0.1813</u>
		SeaCache	84.66	3.02×	18.78	0.6710	0.2226
		Ours	84.98	3.01×	**20.95**	**0.7244**	**0.1745**
	4.0×	TeaCache	64.16	3.99×	14.09	0.4625	0.4554
		MagCache	64.09	3.99×	17.03	0.4974	0.3917
		DiCache	64.24	3.98×	14.68	0.4611	0.4198
		SeaCache	64.91	3.94×	<u>17.13</u>	<u>0.5840</u>	<u>0.3025</u>
		Ours	65.49	3.91×	**18.24**	**0.5891**	**0.2898**

2.17, and 1.11 dB at $2.4\times$, $3.0\times$, and $4.0\times$, respectively. LPIPS also improves over this baseline at all three settings and is best among all methods at $3.0\times$ and $4.0\times$. Wan2.2 generally exhibits lower fidelity than Wan2.1 at comparable speedups, suggesting greater sensitivity to cache reuse (Sec. 4.3).

4.3 Qualitative Comparison

Fig. 4 presents visual comparisons with existing caching methods. As shown, MORCA better preserves the original structure while retaining fine details, as highlighted in the boxed regions. In particular, we observe that the spatial structure of videos generated by Wan2.2 is more sensitive to cache reuse, leading to relatively lower fidelity metrics. Compared with other methods, MORCA is notably better at preserving the original structure.

4.4 Ablation And Analysis

Target Speedup Control. In Table 2, our method achieves mean absolute speedup errors of 0.03 and 0.05, and mean absolute relative inference-time errors of 1.15% and 1.55% on Wan2.2 and Wan2.1, respectively. Note that the number of reuse steps is discrete, so attainable acceleration ratios are also discrete, resulting in unavoidable error in speedup control.

Ablation on Latent-Aware State Design. Table 3 compares scheduler performance with and without latent features in the state representation. The results show that incorporating latent features enables better scheduling decisions.

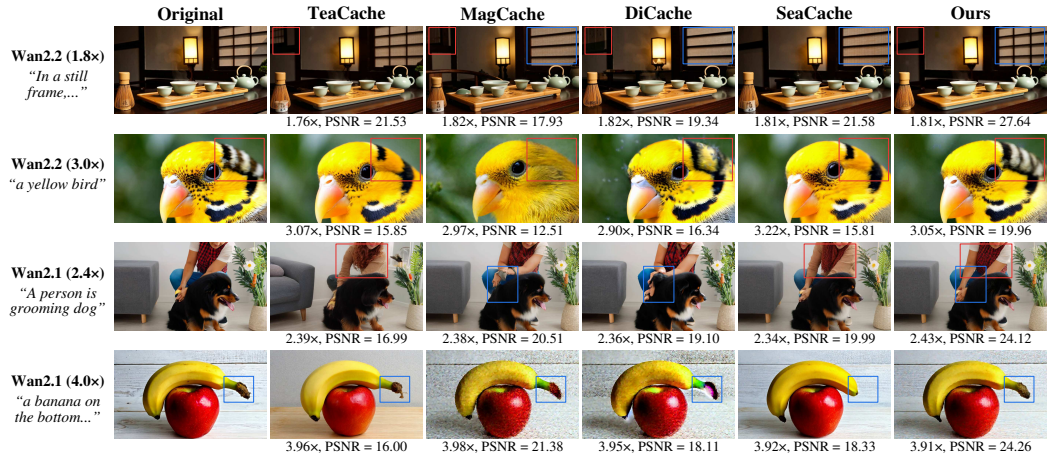

Figure 4: **Qualitative comparison with baselines.** Differences are highlighted in the boxed regions.

Table 3: **Ablation on Latent-Aware State Design.** Offline training results on Wan2.1 at approximately 3.0× speedup.

Variant	$\mathcal{P}_3(x)$	$\mathcal{P}_2(\mu_T(x))$	$\mathcal{P}_2(\sigma_T^2(x))$	Latency (s)↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
A_0	·	·	·	85.46	2.99×	19.93	0.6420	0.2240
A_1	✓	·	·	86.10	2.97×	19.92	0.6515	0.2273
A_2	·	✓	·	86.89	2.95×	19.76	0.6429	0.2314
A_3	·	·	✓	85.39	3.00×	19.27	0.6151	0.2441
A_4	✓	✓	·	85.38	3.00×	20.06	0.6598	0.2197
A_5	✓	·	✓	85.29	3.00×	19.83	0.6484	0.2320
A_6	·	✓	✓	85.12	3.01×	19.71	0.6402	0.2362
Ours	✓	✓	✓	85.27	3.00×	**20.38**	**0.6873**	**0.2036**

Ablation on the Training Paradigm. Table 4 examines our training design through three comparisons. With comparable amounts of training data, (i) IQL outperforms REINFORCE, demonstrating the sample efficiency advantage of off-policy learning; and (ii) offline IQL outperforms online IQL trained from scratch, supporting the effectiveness of offline initialization under a limited data budget. Finally, (iii) online fine-tuning further improves the offline-initialized scheduler.

Table 4: **Ablation on the Training Paradigm.** Results on Wan2.1 at approximately 3.0× speedup.

Variant	Latency (s)↓	Speedup↑	PSNR↑	SSIM↑	LPIPS↓
(i) IQL vs. REINFORCE					
REINFORCE	84.66	3.02×	18.97	0.6251	0.2506
IQL (offline)	85.27	3.00×	**20.38**	**0.6873**	**0.2036**
(ii) Offline-only vs. online-only					
Online-only	84.65	3.02×	18.84	0.5845	0.3212
Offline-only	85.27	3.00×	**20.38**	**0.6873**	**0.2036**
(iii) Offline-only vs. offline-to-online					
Offline-only	85.27	3.00×	20.38	0.6873	0.2036
Offline-to-online	84.98	3.01×	**20.95**	**0.7244**	**0.1745**

5 Conclusion

We presented MORCA, a cache scheduling framework for accelerating video diffusion models. Motivated by the gap between step error and terminal error that existing methods do not adequately address, along with their limited control over acceleration ratios, we formulated cache scheduling as an MDP, designed a budget-constrained and latent-aware scheduler for terminal-error-guided decisions under user-specified acceleration targets, and optimized it through offline-to-online IQL training. Extensive experiments on different video generation models demonstrate that MORCA achieves better speed–quality trade-offs than state-of-the-art caching methods. As a schedule-level approach, MORCA can potentially be combined with different caching granularities and cache reuse strategies for further gains. Exploring these combinations is a promising direction for future work.

AI USE STATEMENT

We used generative AI tools for language polishing, manuscript formatting, and coding assistance during method implementation. These tools were not used to design the proposed method. All AI-assisted text was reviewed by the authors for technical accuracy and consistency with our research findings, and all AI-assisted code was reviewed and tested by the authors for correctness. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we will publicly release the source code of MORCA. Details of the model configurations, training data construction, network architecture, training hyperparameters, and baseline implementations are provided in Appendices A–C. We hope these resources will support future research on caching-based acceleration for video diffusion models.

REFERENCES

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 73778–73803, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/78288ef33b18a351c3cd679dc9a15c8d-Paper-Conference.pdf.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. δ-dit: A training-free acceleration method tailored for diffusion transformers, 2024. URL https://arxiv.org/abs/2406.01125.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. Seacache: Spectral-evolution-aware cache for accelerating diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14283–14294, June 2026.

Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 13237–13249. Curran Associates, Inc., 2023. doi: 10.52202/075280-0580. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/2aab8a76c7e761b66eccaca0927787de-Paper-Conference.pdf.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21807–21818, June 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 (ICCV)*, pp. 15240–15252, October 2025.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Junkun Yuan, Yanxin Long, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Dax Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. Hunyuanvideo: A systematic framework for large video generative models, 2025. URL https://arxiv.org/abs/2412.03603.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Senmao Li, Taihang Hu, Joost van de Weijer, Fahad Shahbaz Khan, Tao Liu, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of the encoder for diffusion model inference. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 85203–85240. Curran Associates, Inc., 2024. doi: 10.52202/079017-2706. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/9ad996b5c45130de2bc00b60d8607904-Paper-Conference.pdf.

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 (ICCV)*, pp. 17535–17545, October 2023.

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

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

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 (ICCV)*, pp. 15853–15863, October 2025b.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=PlKWVd2yBkY.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan LI, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 5775–5787. Curran Associates, Inc., 2022. doi: 10.52202/068431-0418. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/260a14acce2a89dad36adc8eefe7c59e-Paper-Conference.pdf.

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, Aug 2025. ISSN 2731-5398. doi: 10.1007/s11633-025-1562-4. URL https://doi.org/10.1007/s11633-025-1562-4.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference, 2023. URL https://arxiv.org/abs/2310.04378.

Zhengyao Lyu, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K Wong. Fastercache: Training-free video diffusion model acceleration with high quality. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 33132–33156, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/518046d86bbc41a0707727c38301ad8e-Paper-Conference.pdf.

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 (CVPR)*, pp. 15762–15772, June 2024.

Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. Magcache: Fast video generation with magnitude-aware cache. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 34348–34380. Curran Associates, Inc., 2025. doi: 10.52202/085713-1153. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/311207bb626e36a8f1d3eb92aa67af22-Paper-Conference.pdf.

Jeremy McMahan and Xiaojin Zhu. Anytime-constrained reinforcement learning. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li (eds.), *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 4321–4329. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/mcmahan24a.html.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, October 2023.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1972–1981, June 2023.

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

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32211–32252. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/song23a.html.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models, 2025. URL https://arxiv.org/abs/2503.20314.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6211–6220, June 2024.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, Jianfei Chen, Ion Stoica, Kurt Keutzer, and Song Han. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68208–68224. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/xi25c.html.

Zhikang Xie, Xichen Ye, Yifan Wu, Haoshen Yu, Li chenan, Peizhu Gong, Weizhong Zhang, and Cheng Jin. Onlinecache: Learning dynamic caching policies with error correction for efficient diffusion inference, 2026. URL https://arxiv.org/abs/2607.29398.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo 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 (CVPR)*, pp. 6613–6623, June 2024.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Rui Wan, Widyadewi Soedarmadji, Enshu Liu, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, Huazhong Yang, Xuefei Ning, and Yu Wang. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65811–65841, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4a1ee071ce0fe63b83bce507c9dc4d7-Paper-Conference.pdf.

Tong Zhao, Mingkun Lei, Yucheng Han, and Chi Zhang. Bag: Budget-aware gating for diffusion caching, 2026a. URL https://arxiv.org/abs/2608.09231.

Wangbo Zhao, Yizeng Han, Zhiwei Tang, Jiasheng Tang, Pengfei Zhou, Kai Wang, Bohan Zhuang, Zhangyang Wang, Fan Wang, and Yang You. Rapidˆ3: Tri-level reinforced acceleration policies for diffusion transformer. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 149423–149446, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 49842–49869. Curran Associates, Inc., 2023. doi: 10.52202/075280-2170. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/9c2aa1e456ea543997f6927295196381-Paper-Conference.pdf.

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 3296–3319, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/092c2d45005ea2db40fc24c470663416-Paper-Conference.pdf.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 75467–75488, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/bbe024e0517fe12ac3a8d388b19ff9fe-Paper-Conference.pdf.

APPENDIX

A Model Configurations 15

B MORCA Implementation Details 15

B.1 Network Architecture 15

B.2 Training Hyperparameters 15

B.3 Dataset Preparation 15

C Baseline Implementation Details 16

D Reduction of the Hard-Constrained MDP 17

E Additional Experiments 18

E.1 Additional Qualitative Comparisons 18

E.2 Scheduler Overhead 18

E.3 Detailed Comparisons with SeaCache 18

E.3.1 Schedule Visualizations at the Same Reuse Budget 18

E.3.2 Additional Target Acceleration Ratios 20

F Limitations 20

A Model Configurations

Table 5 summarizes the inference configurations for Wan2.2 and Wan2.1. Both models use 50 denoising steps at a resolution of 832×480 and a frame rate of 16 fps. Wan2.2 generates 45-frame videos using DPM++, while Wan2.1 generates 81-frame videos using UniPC. Model-specific settings, including flow shift, CFG scale, and model offloading, are also reported.

Table 5: **Inference configurations for Wan2.2 and Wan2.1.**

Setting	Wan2.2	Wan2.1
Resolution ($W \times H$)	832×480	832×480
Frames	45	81
Frame rate	16 fps	16 fps
Sampler	DPM++	UniPC
Denoising steps	50	50
Flow shift	12	5
CFG scale	$4/3$ (high/low)	5
Noise-stage boundary	0.875	N/A
Model offloading	Enabled	Disabled

B MORCA Implementation Details

B.1 Network Architecture

For both video generation models, we use four separate networks for π_φ, Q_{θ_1}, Q_{θ_2}, and V_ψ. The pooled features of x_t, x_{t+1}, and x_{j_t} form a $48 \times 4 \times 8 \times 8$ spatiotemporal tensor and two $48 \times 8 \times 8$ tensors representing temporal means and variances, respectively (batch dimension omitted). The spatiotemporal tensor passes through a 3D CNN branch, while the temporal mean and variance tensors are concatenated along the channel dimension and passed through a 2D CNN branch. Each branch has three convolutional layers of widths $(32, 32, 64)$, followed by adaptive average pooling and flattening, producing 512- and 256-dimensional vectors for the 3D and 2D branches, respectively. Concatenation and a linear projection followed by SiLU map the combined 768-dimensional vector to a 128-dimensional latent embedding. Appending seven scalar state features—d_t, D_t, t, ℓ_t, u_t, K, and a cache-valid bit—gives a 135-dimensional input to an MLP with two 256-unit hidden layers, each followed by LayerNorm and SiLU. The policy outputs two action logits, each Q-network outputs two action values, and the V-network outputs a scalar. Two additional target Q-networks share the architecture of their corresponding Q-networks and are updated through EMA. Only the policy network is needed at inference time. Further implementation details are provided in the released code.

B.2 Training Hyperparameters

Table 6 summarizes the training and IQL settings, where γ denotes the discount factor, τ the expectile level for value regression, and β the advantage-weighting coefficient. The weight cap $w_{\max}$ limits the policy weights as $\min(\exp(\beta \widetilde{A}), w_{\max})$, where $\widetilde{A}$ is the normalized advantage. The EMA decay η controls target-Q network updates via $\theta_{\text{target}} \leftarrow \eta\theta_{\text{target}} + (1-\eta)\theta_Q$.

B.3 Dataset Preparation

Data Generation Pipeline. A straightforward way to generate random cache schedules is to independently sample a reuse/recompute decision at each denoising step. However, this is likely to produce trajectories with limited learning value, wasting costly generation rollouts. Instead, we adopt a more stable approach by randomizing at the threshold level: we sample a smooth threshold curve that varies across the N denoising steps, replace SeaCache's fixed threshold with this curve, and run inference to collect the resulting denoising trajectory.

To generate a random threshold curve, we control three aspects: its mean level, amplitude, and shape. The mean level is determined by mapping a sampled target acceleration ratio $\rho^\star$ to a threshold center $\mu = g(\rho^\star)$ through calibration. A randomly sampled amplitude factor determines the magnitude of

Table 6: **Training and IQL hyperparameters for the two stages, shared across Wan2.2 and Wan2.1.**

Group	Setting	Offline IQL	Online fine-tuning
Training	Training duration	200 epochs	16 rounds
	Optimizer	AdamW	AdamW
	π LR	10^{-4}	4×10^{-5}
	Q/V LR	10^{-4}	10^{-4}
	Batch size	256	256
	Adam betas	$(0.9, 0.999)$	$(0.9, 0.999)$
	Adam epsilon	10^{-8}	10^{-8}
	Weight decay	0.01	0.01
	Maximum gradient norm	1.0	1.0
IQL	γ	1.0	1.0
	τ	0.7	0.7
	β	1.5	1.0
	$w_{\max}$	30	20
	η	0.999	0.999
	New trajectories	N/A	100 per round
	Critic-only warm-up	N/A	5 epochs per round
	Joint updates	N/A	20 epochs per round

threshold variations. To generate the shape, we randomly select a few reference steps from the N denoising steps, sample random relative threshold values at these steps, and apply cubic smoothstep interpolation across all steps. These three factors jointly determine the threshold curve, followed by adjustments to satisfy the required bounds and smoothness constraints.

After each rollout, we convert the trajectory into N transition tuples. For each step t, we compute the cumulative reuse count $u_t = \sum_{i=t+1}^{N} a_i$ and assign the trajectory's total reuse count $K = \sum_{i=1}^{N} a_i$ as the reuse budget. We combine these counts with the step-error signals, step statistics, and latent features extracted from the recorded denoising process to construct the state vector z_t. Using the reward defined in the main text, each trajectory yields N transition tuples:

$$(z_t, a_t, r_t, z_{t-1}), \qquad t = N, \ldots, 1.$$

Further implementation details are available in our released code.

Dataset scale. We collect 2,500 random cache-scheduling trajectories for Wan2.2 (100 prompts $\times$ 25 schedules), covering speedups from $1.5\times$ to $3.5\times$, and 4,000 for Wan2.1 (1,000 prompts $\times$ 4 schedules), covering speedups from $2.0\times$ to $4.5\times$.

Forced-action handling. We identify two types of forced actions: (1) actions imposed by the exact reuse-budget constraint, when only one action remains feasible; and (2) mandatory recomputation at the first denoising step and the last denoising step, as well as the high-noise/low-noise expert-switching step in Wan2.2. Since these actions are determined by action restrictions rather than policy choices, we exclude the corresponding transitions from the policy loss while retaining them for the standard Q- and V-function updates.

C Baseline Implementation Details

For Wan2.2, we port the official Wan2.1 implementations and add mandatory recomputation at the high-noise/low-noise expert-switching step to initialize the newly activated expert's cache. MagCache uses its official Wan2.2 implementation directly. For TeaCache on Wan2.2, we fit separate fourth-degree polynomials for the high- and low-noise stages using 70 calibration prompts. For Wan2.1, we use the official Wan2.1 implementations released by the respective caching methods. Table 7 summarizes the key configurations for three approximate acceleration regimes.

Method-specific settings. TeaCache and SeaCache use `use_ret_steps=False` for both models. For Wan2.1, SeaCache additionally uses `power_exp=3` and `power_const=1`. DiCache uses `probe_depth=1` for both models, with DCTA enabled and its correction coefficients clipped to $[1, 2]$.

Table 7: **Baseline configurations at three target acceleration ratios for each model.** δ denotes the cache threshold, K the maximum number of consecutive reuse steps for MagCache, and R the retention ratio.

Model	Method	1.8×	2.4×	3.0×
Wan2.2	TeaCache	$\delta=0.29$	$\delta=0.45$	$\delta=0.68$
	MagCache	$\delta=0.072$, $K=2$, $R=0.2$	$\delta=0.199$, $K=4$, $R=0.2$	$\delta=0.198$, $K=5$, $R=0.1$
	DiCache	$\delta=0.075$, $R=0.2$	$\delta=0.172$, $R=0.2$	$\delta=0.396$, $R=0.2$
	SeaCache	$\delta=0.24$	$\delta=0.38$	$\delta=0.55$
Model	**Method**	**2.4×**	**3.0×**	**4.0×**
Wan2.1	TeaCache	$\delta=0.142$	$\delta=0.195$	$\delta=0.36$
	MagCache	$\delta=0.1500$, $K=6$, $R=0.2$	$\delta=0.9246$, $K=8$, $R=0.2$	$\delta=8.0$, $K=40$, $R=0.2$
	DiCache	$\delta=0.155$, $R=0.2$	$\delta=0.365$, $R=0.2$	$\delta=12.0$, $R=0.17$
	SeaCache	$\delta=0.28$	$\delta=0.47$	$\delta=0.7$

D Reduction of the Hard-Constrained MDP

We use the MDP and terminal reward defined in the main text, with s_t denoting the full Markov state and P_t denoting the step-t transition kernel corresponding to P in the main text. Assume standard Borel state spaces and measurable transition and reward kernels. PSNR is capped at the fixed finite maximum $r_{\max}$ defined in the main text, yielding bounded rewards for videos in a fixed bounded pixel range.

The displayed formulas assume that every step permits reuse. Fix an integer budget $K \in \{0, \dots, N\}$. Let $b_t = K - u_t$ denote the remaining reuse budget, as in the main text. Recall the augmented state and action restriction:

$$\bar{s}_t = (s_t, u_t, K), \qquad u_t = \sum_{i=t+1}^{N} a_i, \\ \bar{\mathcal{A}}_t(\bar{s}_t) = \{a \in \{0,1\} : 0 \le b_t - a \le t-1\}. \tag{12}$$

Only integer counts satisfying $0 \le u_t \le \min\{K, N-t\}$ and $0 \le b_t \le t$ are included. The augmented process preserves the initial distribution and transitions P_t of the original state component and all rewards, with $u_N = 0$ and $u_{t-1} = u_t + a_t$. Predetermined recomputation steps are handled by excluding reuse at those steps and adjusting the budget range, bounds on the cumulative reuse count, and action restriction to count only steps at which reuse is permitted.

Proposition D.1. *Let Π_K^{hist} denote all possibly randomized history-dependent policies satisfying $\sum_{t=1}^{N} a_t = K$ almost surely. Let $\bar{\Pi}^{\mathrm{hist}}$ and $\bar{\Pi}^{\mathrm{det}}$ denote, respectively, all admissible possibly randomized history-dependent policies and all admissible time-dependent deterministic Markov policies of the augmented MDP. The reduction guarantees exactly K reuse actions almost surely and preserves the optimal feasible value:*

$$\sup_{\pi \in \Pi_K^{\mathrm{hist}}} \mathbb{E}_{\tau\sim\pi}[R(\tau)] = \max_{\bar{\pi} \in \bar{\Pi}^{\mathrm{det}}} \mathbb{E}_{\bar{\tau}\sim\bar{\pi}}[R(\bar{\tau})], \tag{13}$$

where $R(\bar{\tau})$ denotes the unchanged return along the corresponding original trajectory.

Proof. The augmented process is Markov because s_t is Markov and the count update is deterministic.

Step 1. Feasibility. It is straightforward to verify that the augmented action sets are nonempty at every nonterminal state, every admissible augmented policy satisfies the prescribed reuse budget almost surely, and every feasible original policy respects the augmented action restriction almost surely.

Step 2. Policy correspondence. Since cumulative counts are determined by the original history, any feasible original policy can be represented as an admissible augmented policy by adding these counts, and any admissible augmented policy can be executed in the original process by maintaining them. In either direction, corresponding policies induce identical original state-action trajectory

distributions and equal expected returns, with measurable feasible actions assigned on histories of probability zero where necessary. Consequently,

$$\sup_{\pi \in \Pi_K^{\mathrm{hist}}} \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \sup_{\bar{\pi} \in \bar{\Pi}^{\mathrm{hist}}} \mathbb{E}_{\bar{\tau} \sim \bar{\pi}}[R(\bar{\tau})]. \tag{14}$$

Step 3. Markov optimality. Since the terminal reward is included in r_1, set $\bar{V}_0^\star = 0$ and define the optimal action-value and value functions of the augmented MDP backward along the denoising trajectory, in the order $t = 1, \ldots, N$

$$\begin{aligned} \bar{Q}_t^\star(\bar{s}, a) &= \mathbb{E}\big[r_t + \bar{V}_{t-1}^\star(\bar{s}_{t-1}) \mid \bar{s}_t = \bar{s},\ a_t = a\big], \\ \bar{V}_t^\star(\bar{s}) &= \max_{a \in \bar{\mathcal{A}}_t(\bar{s})} \bar{Q}_t^\star(\bar{s}, a). \end{aligned} \tag{15}$$

The value functions are bounded and measurable, and the finite nonempty action sets admit a measurable maximizing rule f_t, with ties broken by choosing the smaller action. For any $\bar{\pi} \in \bar{\Pi}^{\mathrm{hist}}$, backward induction from the terminal state $t = 0$, using the tower property and the Markov transition, gives almost surely, for $1 \le t \le N$,

$$\begin{aligned} \mathbb{E}^{\bar{\pi}}\left[\sum_{i=1}^{t} r_i \mid \bar{h}_t\right] &\le \sum_{a \in \bar{\mathcal{A}}_t(\bar{s}_t)} \bar{\pi}_t(a \mid \bar{h}_t)\, \bar{Q}_t^\star(\bar{s}_t, a) \\ &\le \bar{V}_t^\star(\bar{s}_t), \end{aligned} \tag{16}$$

where $\bar{h}_t$ is the augmented history through time t. Under $f = (f_N, \ldots, f_1)$, equality holds throughout. Integrating over the initial state distribution shows that f attains the optimal expected return over all admissible augmented history-dependent policies. Together with the policy correspondence, this proves the proposition. □

E Additional Experiments

E.1 Additional Qualitative Comparisons

Figs. 7 and 8 present additional qualitative comparisons between MORCA and the baseline methods on Wan2.2 and Wan2.1, respectively, at three target acceleration ratios.

E.2 Scheduler Overhead

We additionally measure the time spent on scheduler execution and calculate its proportion of the total inference time. As shown in Table 8, this proportion is approximately 1%, indicating acceptable overhead.

Table 8: **Scheduler overhead on Wan2.2.** The scheduler introduces acceptable overhead.

Target	Latency (s)	Scheduler time (s)	Proportion (%)
1.8×	549.19	3.95	0.72
2.4×	421.15	3.94	0.94
3.0×	326.93	3.93	1.20

E.3 Detailed Comparisons with SeaCache

E.3.1 Schedule Visualizations at the Same Reuse Budget

Fig. 5 visualizes the cache schedules of MORCA and SeaCache under the same reuse budgets. MORCA learns schedules that differ from SeaCache's and achieves better generation fidelity in both examples.

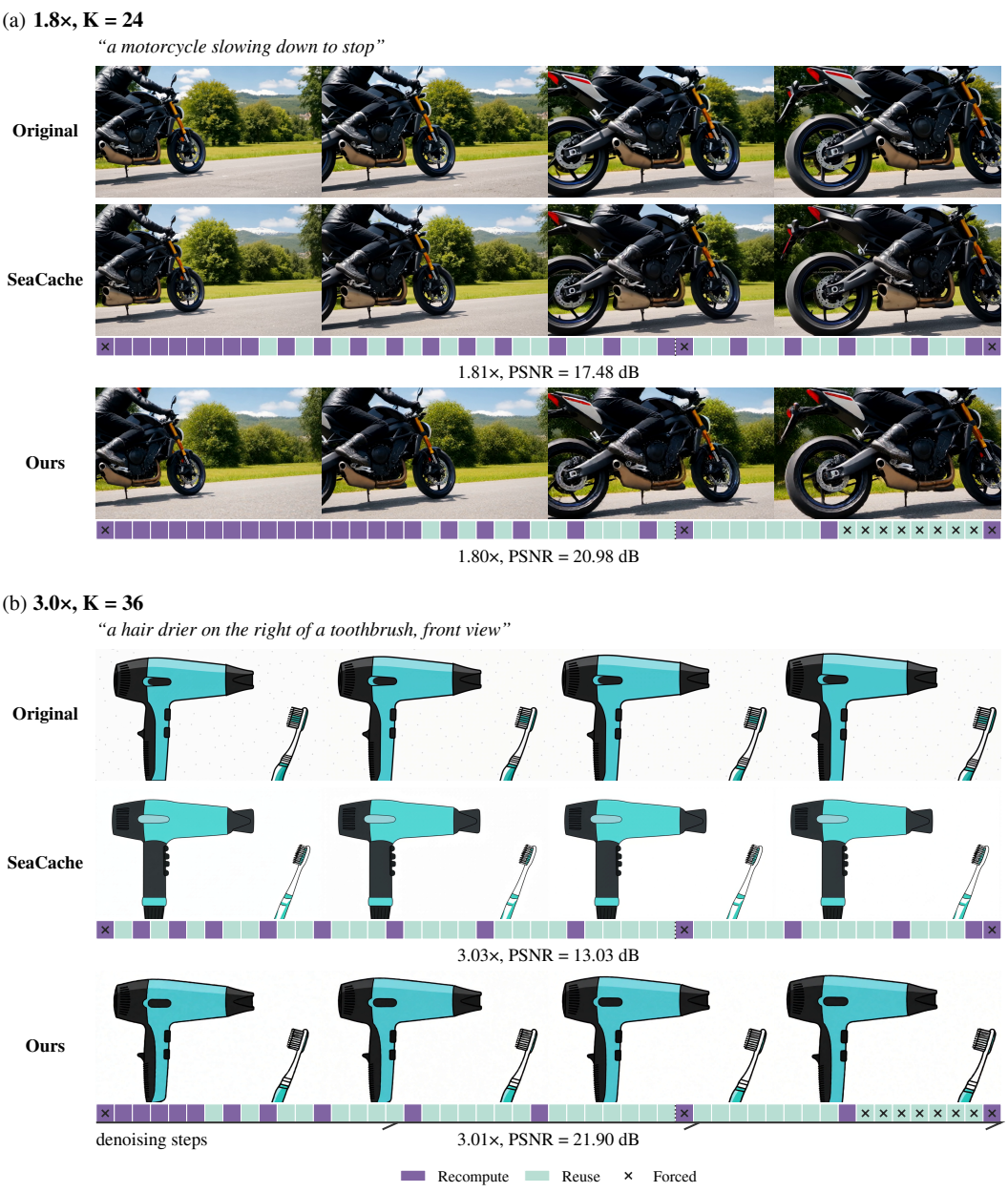

Figure 5: **Visualization of schedules and corresponding generation results under the same reuse budget on Wan2.2.**

E.3.2 Additional Target Acceleration Ratios

We compare MORCA with SeaCache on Wan2.2 at nine target acceleration ratios ranging from $1.5\times$ to $3.5\times$ in increments of $0.25\times$. As shown in Fig. 6, MORCA consistently outperforms SeaCache in PSNR, SSIM, and LPIPS across all settings.

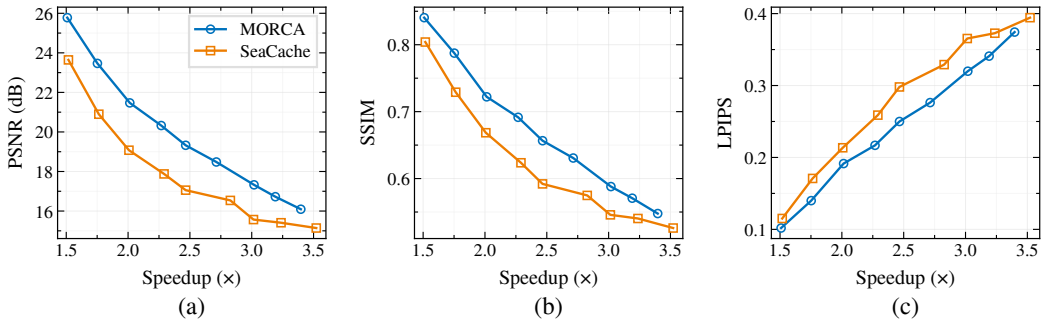

Figure 6: **Speed–quality trade-offs of MORCA and SeaCache on Wan2.2.** MORCA consistently achieves better fidelity at comparable speedups.

F Limitations

Due to differences in denoising dynamics across models, MORCA's learned scheduler cannot be directly transferred between models and requires separate training for each model. This introduces additional costs for collecting cache-scheduling trajectories and optimizing the scheduler. These costs may limit its practical adoption when training resources are constrained.

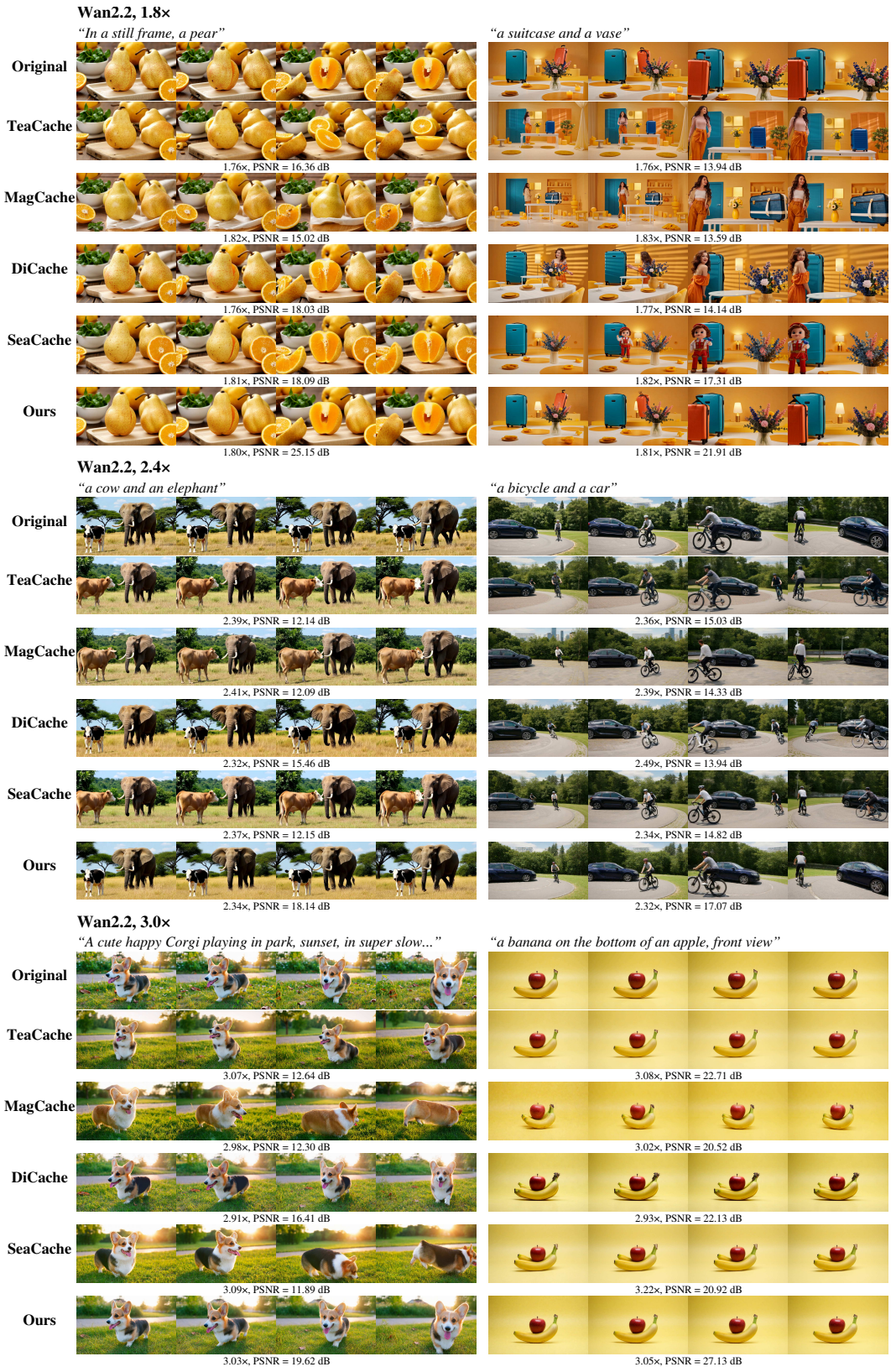

Figure 7: **Additional qualitative comparisons on Wan2.2.** From top to bottom: target acceleration ratios of $1.8\times$, $2.4\times$, and $3.0\times$.

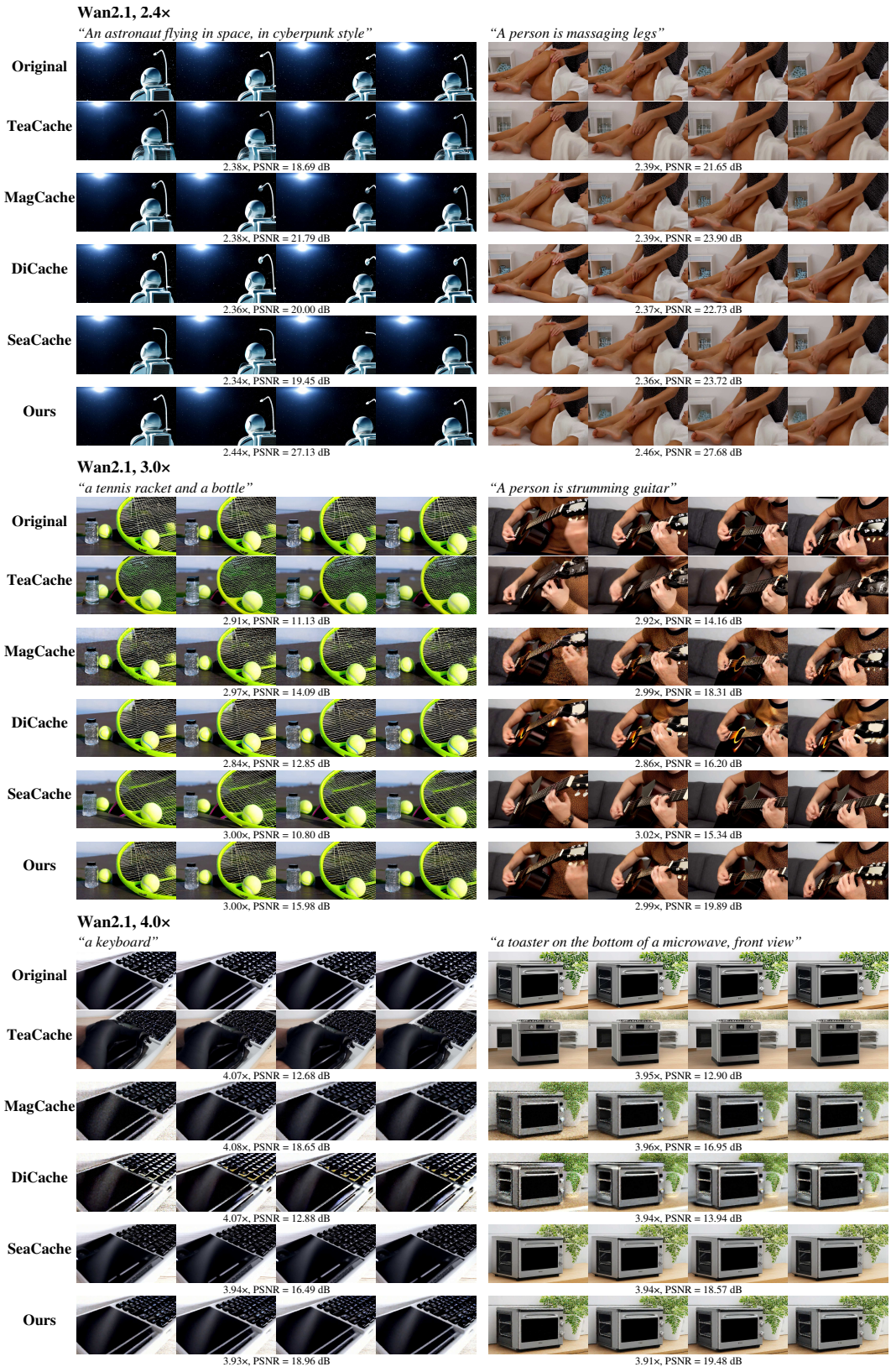

Figure 8: **Additional qualitative comparisons on Wan2.1.** From top to bottom: target acceleration ratios of $2.4\times$, $3.0\times$, and $4.0\times$.