RoDCache: Routing-Drift-Aware Caching for Accelerating Sparse MoE Diffusion Models

Anonymous authors
Paper under double-blind review

Abstract

Diffusion models incur substantial inference cost due to repeated execution of the denoising network across multiple timesteps. Caching methods reduce this cost by reusing results from previous denoising steps based on estimates of cache validity. However, in sparse MoE diffusion models, existing cache-validity signals overlook how routing drift in expert assignments and routing weights alters the active computation path and affects reuse reliability. To address this limitation, we present RoDCache, a training-free caching method that accounts for routing drift in cache-validity assessment. It comprises two components: Routing Drift Quantification (RDQ) and Routing-Drift-Constrained Scheduling (RDCS). RDQ quantifies model-level routing drift by aggregating changes in expert assignments and routing weights across tokens and MoE layers. Based on this signal, RDCS selects the longest reuse interval whose predicted routing drift does not exceed a tunable threshold. Across three sparse MoE diffusion models, RoDCache achieves $3.01\times$–$5.13\times$ speedups and consistently outperforms existing caching methods in PSNR, SSIM, and LPIPS, with PSNR gains of up to 2.76 dB. Further analysis shows that RoDCache adapts computation allocation to prompt difficulty and improves quality consistency across prompts. Code will be released at `https://anonymous.4open.science/r/RoDCache`.

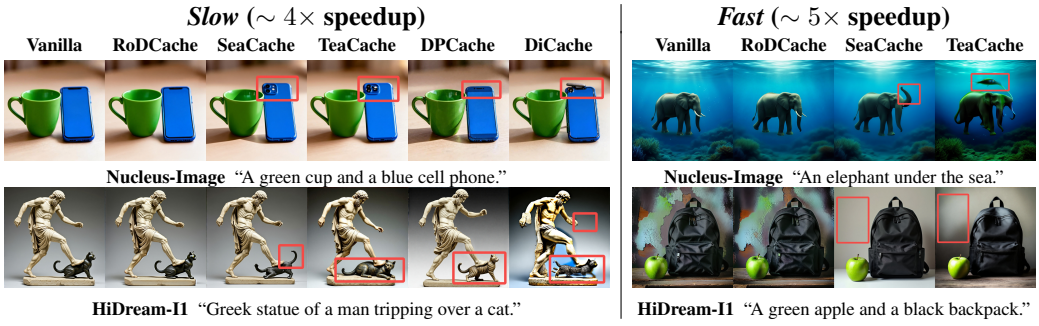


Figure 1: Qualitative comparison of RoDCache and caching baselines on two sparse MoE image diffusion models: Nucleus-Image and HiDream-I1. The *slow* and *fast* settings target approximately $4\times$ and $5\times$ speedups. Red boxes highlight visible baseline deviations from Vanilla outputs.

1 Introduction

Diffusion models are now a leading framework for high-quality image and video generation (Esser et al., 2024; Yang et al., 2025). Within this framework, Diffusion Transformers (DiTs) replace conventional U-Net architectures (Ronneberger et al., 2015) with scalable transformer backbones (Peebles & Xie, 2023; Chen et al., 2024). As diffusion models continue to scale, sparse mixture-of-experts (MoE) architectures have been adopted to increase model capacity without a proportional rise in computation (Cai et al., 2025; Akiti et al., 2026; Ma et al., 2026). Nevertheless, diffusion inference remains expensive because it typically requires repeatedly executing the denoising network across dozens of denoising steps.

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.

To reduce this repeated computation, caching has emerged as an important approach to accelerating diffusion inference (Liu et al., 2025b). Caching methods interleave full-computation steps with reuse steps. These methods reuse stored results (e.g., attention/MLP outputs (Zou et al., 2025) or transformer residuals (Liu et al., 2025a)) from previous full-computation steps during reuse steps to avoid redundant computation. The reliability of reuse depends on ***cache validity***, which indicates whether the stored results remain accurate and can be reused at the current denoising step. Existing methods typically assess cache validity using cross-step changes in intermediate representations as proxies for the accuracy of cached results (Liu et al., 2025a;d).

However, we find that existing cache-validity signals are insufficient for sparse MoE diffusion models, leading to visible deviations from full-computation outputs, as shown in Fig. 1. The underlying issue is that sparse MoE models do not follow a fixed computation path. Unlike dense models, sparse MoE models dynamically determine their active computation paths by routing each token's hidden representation to a subset of experts (Fedus et al., 2022; Zhou et al., 2022). Because expert selection is discrete, even small changes in hidden representations can result in substantially different sets of active experts across denoising steps (Tran Huu et al., 2026). In this paper, we refer to these changes in expert assignments and routing weights as ***routing drift***. Cache-validity assessments that overlook the routing drift might fail to detect changes in the active computation path and consequently misestimate cache validity.

Accounting for routing drift in cache-validity assessment presents two challenges. First, obtaining a reliable measure of routing drift is difficult. Since each token in every MoE layer has its own routing state, quantifying routing drift requires aggregating numerous routing-state changes across all tokens and layers. Second, using routing drift to accurately assess cache validity is challenging. Because routing drift accumulates unevenly over the denoising process, the duration for which cached results remain valid varies across denoising stages.

To address these challenges, we propose RoDCache, a training-free adaptive caching method that uses routing drift to guide cache reuse in sparse MoE diffusion models. Specifically, RoDCache comprises two principal components: **Routing Drift Quantification (RDQ)** and **Routing-Drift-Constrained Scheduling (RDCS)**. RDQ constructs a model-level measure of routing drift by aggregating changes across all tokens and MoE layers. To enable comparison, each token's expert assignments and routing weights are jointly encoded in a normalized representation. RDCS determines how long cached results are expected to remain valid subject to a tunable routing-drift threshold. It uses recently observed routing drift to select the longest reuse interval whose predicted drift does not exceed this threshold. By combining RDQ and RDCS, RoDCache incorporates routing drift into cache-validity assessment and adaptively guides cache reuse. Across three MoE diffusion models, RoDCache achieves $3.01\times$–$5.13\times$ speedups and consistently outperforms caching baselines in PSNR, SSIM, and LPIPS, with PSNR gains of up to 2.76 dB.

Our main contributions are as follows:

- **Routing-Drift-Aware Caching.** We propose RoDCache, a training-free adaptive caching method tailored to sparse MoE diffusion models. It explicitly accounts for routing changes when assessing cache validity and adapts reuse intervals to the uneven accumulation of routing drift over the denoising process.
- **Routing Drift Quantification and Constrained Scheduling.** We develop RDQ and RDCS to transform numerous routing-state changes into a model-level routing-drift signal that guides adaptive cache-reuse scheduling under a single tunable threshold. This design requires no additional model computation at reuse steps.
- **Outstanding Performance.** We evaluate RoDCache on three image and video diffusion models covering both token-choice and expert-choice routing. The results demonstrate favorable quality–efficiency trade-offs and show that RoDCache adapts computation allocation to prompt difficulty.

2 Related Work

Diffusion model acceleration. Diffusion models and their transformer-based variants have advanced image and video generation (Xie et al., 2025; Ma et al., 2025a). However, iterative denoising incurs substantial inference cost. Existing acceleration methods follow several directions. Efficient

samplers reduce the number of denoising steps (Sabour et al., 2024; Zheng et al., 2023), while distillation (Kim et al., 2024) and pruning (Zhu et al., 2024) reduce per-step computation. Caching reduces diffusion inference cost by reusing results from previous denoising steps while preserving cache validity (Zou et al., 2025; 2026). Our work focuses on the caching direction.

Cache-based acceleration for diffusion models. Caching methods reduce diffusion inference cost by reusing results computed at earlier denoising steps. TeaCache (Liu et al., 2025a), MagCache (Ma et al., 2025b), SeaCache (Chung et al., 2026), SenCache (Haghighi & Alahi, 2026), DiCache (Bu et al., 2026), and DPCache (Cui et al., 2026) use cross-step changes in intermediate representations to assess cache validity and determine when to refresh cached results. AdaCache (Kahatapitiya et al., 2025) adjusts cache reuse through a heuristic codebook with multiple cache-rate and threshold hyperparameters. TaylorSeer (Liu et al., 2025c) and MeanCache (Gao et al., 2026) predict subsequent results from previously cached results. These approaches do not consider how changes in expert routing affect cache validity in sparse MoE models. Our RoDCache addresses this limitation by quantifying routing drift and scheduling reuse subject to a routing-drift threshold.

Caching for sparse MoE diffusion models. Sparse MoE models increase model capacity by introducing multiple experts while activating only a small subset of them (Dai et al., 2024; Komatsuzaki et al., 2023). At each denoising step, the active-expert set is discretely selected based on the hidden representations. Despite similar hidden representations at adjacent steps, the selected expert sets can differ substantially. MoECa (Li et al., 2026) aligns cache reuse with expert decomposition by executing the router at every accelerated step to guide branch-level reuse. In contrast, our RoDCache adaptively determines reuse intervals based on routing drift quantified at full-computation steps. It requires neither branch matching nor router execution at reuse steps.

3 PRELIMINARIES

In this section, we introduce sparse MoE routing and formulate routing-drift-aware caching. We summarize the notation used throughout the paper in Table 1.

3.1 SPARSE MOE ROUTING

We index the T denoising timesteps as $t \in \{T-1, \dots, 0\}$. For an MoE layer ℓ with N_ℓ tokens and M_ℓ dynamic experts, let $g^{(t)}_{\ell,i,e} \geq 0$ denote the affinity score between token i and expert e produced by the router after softmax or sigmoid. Sparse MoE diffusion models use either **token-choice routing**, where each token selects its top-K_ℓ experts, or **expert-choice routing**, where each expert selects up to C_ℓ tokens. We use $g^{(t)}_{\ell,i,:}$ and $g^{(t)}_{\ell,:,e}$ to denote the affinity scores across experts and tokens, respectively. Based on these affinity scores, the active-expert set for token i is

$$\mathcal{A}^{(t)}_{\ell,i} = \begin{cases} \mathrm{TopK}\left(g^{(t)}_{\ell,i,:}, K_\ell\right), & \text{token-choice routing,} \\ \left\{ e \mid i \in \mathrm{TopC}\left(g^{(t)}_{\ell,:,e}, C_\ell\right) \right\}, & \text{expert-choice routing.} \end{cases} \tag{1}$$

Token-choice routing activates exactly K_ℓ experts for each token, whereas expert-choice routing may activate different numbers of experts across tokens.

After expert selection, affinity scores may undergo model-specific normalization and scaling to produce routing weights $w^{(t)}_{\ell,i,e}$. Using these routing weights, the output of active experts is

$$\mathbf{y}^{(t)}_{\ell,i} = \sum_{e \in \mathcal{A}^{(t)}_{\ell,i}} w^{(t)}_{\ell,i,e}\, \mathcal{E}_{\ell,e}\left(\mathbf{z}^{(t)}_{\ell,i}\right). \tag{2}$$

$\mathbf{z}^{(t)}_{\ell,i}$ denotes the input representation of token i at layer ℓ, and $\mathcal{E}_{\ell,e}$ denotes the transformation performed by dynamic expert e. This formulation highlights two aspects of MoE routing that may change across denoising steps: expert assignments and routing weights.

3.2 THE DEFINITION OF ROUTING-DRIFT-AWARE CACHING

Caching methods reuse different stored results, such as attention/MLP outputs (Zou et al., 2025) or transformer residuals (Liu et al., 2025a; Ma et al., 2025b). Regardless of the stored results, caching

Table 1: Summary of key notation.

Symbol	Description
t, T, ℓ, i, e	t and T denote the denoising timestep and total number of timesteps, respectively. ℓ, i, and e index MoE layers, tokens, and experts, respectively.
$N_\ell, M_\ell, K_\ell, C_\ell$	N_ℓ and M_ℓ denote the numbers of tokens and experts at layer ℓ, respectively. K_ℓ denotes experts selected per token, while C_ℓ denotes tokens selected per expert.
$g^{(t)}_{\ell,i,e}, w^{(t)}_{\ell,i,e}, \mathcal{A}^{(t)}_{\ell,i}$	$g^{(t)}_{\ell,i,e}$ denotes the affinity score, $w^{(t)}_{\ell,i,e}$ denotes the routing weight, and $\mathcal{A}^{(t)}_{\ell,i}$ denotes the active-expert set.
$\mathbf{X}^{(t)}, \mathbf{Y}^{(t)}, \mathbf{R}^{(t)}$	$\mathbf{X}^{(t)}$, $\mathbf{Y}^{(t)}$, and $\mathbf{R}^{(t)}$ denote the transformer input, output, and residual, respectively.
$\mathcal{S}_{\text{full}}, f_j, a(t)$	$\mathcal{S}_{\text{full}}$ denotes the full-computation timesteps, f_j denotes its j-th element, and $a(t)$ denotes the most recent full-computation timestep before t.
$\widetilde{\mathbf{p}}^{(t)}_{\ell,i}, \mathbf{p}^{(t)}_{\ell,i}$	$\widetilde{\mathbf{p}}^{(t)}_{\ell,i}$ and $\mathbf{p}^{(t)}_{\ell,i}$ denote the unnormalized and normalized routing vectors, respectively.
$D_{\ell,i}, D_\ell, D_{\text{route}}$	$D_{\ell,i}$ denotes routing drift at the token level, D_ℓ denotes routing drift at the layer level, and D_{route} denotes routing drift at the model level.
$\widehat{D}_{\text{route}}, \widehat{v}_j, h_j, \tau$	$\widehat{D}_{\text{route}}$ denotes predicted routing drift, $\widehat{v}_j$ denotes the estimated drift rate, h_j denotes the reuse-interval length, and τ denotes the routing-drift threshold.

follows the same basic procedure. Full-computation steps refresh the cache, whereas reuse steps use the cached results to approximate the current computation. The central problem is to determine whether the stored results remain valid for reuse or should be refreshed through full computation.

For concreteness, we use transformer-residual caching to illustrate how stored results are refreshed and reused. Let $\mathbf{X}^{(t)}$, $\mathbf{c}$, and $\mathcal{F}_t$ denote the network input, conditioning information, and transformer mapping at timestep t, respectively. The transformer output $\mathbf{Y}^{(t)}$ and residual $\mathbf{R}^{(t)}$ are defined as

$$\mathbf{Y}^{(t)} = \mathcal{F}_t(\mathbf{X}^{(t)}, \mathbf{c}), \qquad \mathbf{R}^{(t)} = \mathbf{Y}^{(t)} - \mathbf{X}^{(t)}. \tag{3}$$

Let $\mathcal{S}_{\text{full}} = \{f_0, \ldots, f_J\} \subseteq \{T-1, \ldots, 0\}$ denote the full-computation timesteps, ordered such that $f_0 > \cdots > f_J$. At these timesteps, the denoising network is executed and the residual cache is refreshed. At a reuse step $t \notin \mathcal{S}_{\text{full}}$, the residual cached at the most recent full-computation timestep $a(t)$ is used to approximate the transformer output:

$$a(t) = \min\{f \in \mathcal{S}_{\text{full}} \mid f > t\}, \qquad \widehat{\mathbf{Y}}^{(t)} = \mathbf{X}^{(t)} + \mathbf{R}^{(a(t))}. \tag{4}$$

In sparse MoE diffusion models, changes in expert routing alter the active computation paths and influence cache validity. To account for these changes, routing-drift-aware caching assesses cache validity using predicted routing drift and determines $\mathcal{S}_{\text{full}}$ by constraining this drift over each reuse interval. Given a tunable threshold $\tau > 0$, this constraint is expressed as

$$\widehat{D}_{\text{route}}(a(t), t) \le \tau, \qquad \forall t \notin \mathcal{S}_{\text{full}}, \tag{5}$$

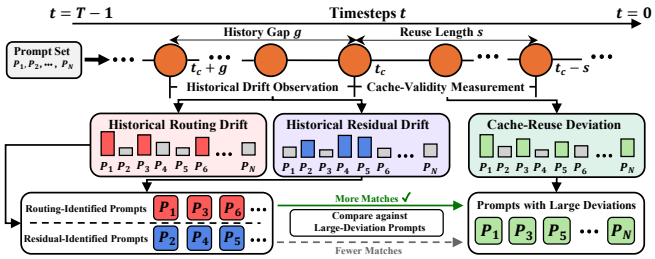
where $\widehat{D}_{\text{route}}$ denotes the predicted routing drift. The effectiveness of routing-drift-aware caching depends on whether routing drift reliably indicates cache validity.

4 Cache-Validity Indicators for Sparse MoE Diffusion Models

Many adaptive caching methods assess cache validity by estimating cross-step changes in the transformer residual, with small changes permitting cache reuse and large changes triggering cache refresh. Since directly measuring residual changes requires full computation, existing methods instead use cross-step changes in intermediate representations as efficient proxies, such as timestep-modulated inputs (Liu et al., 2025a), shallow-layer probe features (Bu et al., 2026), and spectrally filtered input features (Chung et al., 2026). This strategy implicitly assumes that residual drift reliably indicates whether cached results remain valid.

However, in sparse MoE diffusion models, residual drift does not directly reflect shifts in active computation paths. Motivated by this limitation, we empirically study how reliably routing drift and residual drift indicate cache validity in sparse MoE diffusion models. Fig. 2(a) illustrates the experiment design. For a set of prompts, we run a sparse MoE diffusion model with full computation at every denoising timestep and record the exact routing states and transformer residuals. Using these records, we assess the reliability of the two drift indicators through historical drift observation and cache-validity measurement. At each candidate cache timestep t_c, we compute routing and residual

(a) Experiment design for assessing cache-validity indicators.



(b) Indicator comparison results

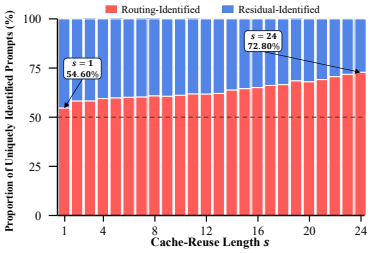


Figure 2: **(a) Experiment design for assessing cache-validity indicators.** Across a prompt set, routing and residual drifts measured over a history gap g are used to identify prompts with large residual deviations at a subsequent cache-reuse length s. A higher proportion of these prompts is uniquely identified by routing drift than by residual drift, indicating that routing drift is a more reliable cache-validity indicator. **(b) Indicator comparison results.** Proportions of prompts with large residual deviations uniquely identified by each drift across cache-reuse lengths $s = 1, \ldots, 24$.

drifts over a history gap g between $t_c + g$ and t_c. Each historical drift is used to identify prompts expected to exhibit a large subsequent residual deviation between t_c and $t_c - s$, which measures cache-reuse error after a reuse length s. We compare the proportions of large-deviation prompts uniquely identified by each indicator, with a higher proportion indicating a more reliable cache-validity signal. Detailed definitions and the experimental protocol are provided in Appendix A.

As shown in Fig. 2(b), routing drift uniquely identifies a larger proportion of prompts with large residual deviations than residual drift for $s \in \{1, \ldots, 24\}$, increasing from 54.60% at $s = 1$ to 72.80% at $s = 24$. These results demonstrate that routing drift more reliably indicates cache validity in sparse MoE diffusion models, even when residual drift is measured directly. Accordingly, we use it to assess cache validity and guide adaptive cache reuse for sparse MoE diffusion models.

5 Method: RoDCache

5.1 RoDCache Overview

Figure 3 presents an overview of RoDCache, a training-free caching method that uses routing drift as a cache-validity signal for adaptive reuse scheduling. It comprises **Routing Drift Quantification (RDQ)** and **Routing-Drift-Constrained Scheduling (RDCS)**. RDQ quantifies model-level routing drift from cross-step changes in normalized routing vectors across tokens and MoE layers. RDCS selects the longest reuse interval that keeps predicted routing drift within a tunable threshold. During this interval, RoDCache reuses the most recent residual without executing the denoising network.

5.2 Routing Drift Quantification

RDQ enables routing-aware cache-validity assessment in sparse MoE models through a model-level measure of routing drift. This measure aggregates cross-step changes in token-wise expert assignments and routing weights.

Concretely, RDQ represents the routing state of token i at layer ℓ and timestep t using a vector $\widetilde{\mathbf{p}}_{\ell,i}^{(t)}$. This representation applies to both token-choice and expert-choice routing. Each of its M_ℓ entries corresponds to an expert. The entry for expert e is its affinity score when active and zero otherwise:

$$\widetilde{p}_{\ell,i,e}^{(t)} = \begin{cases} g_{\ell,i,e}^{(t)}, & e \in \mathcal{A}_{\ell,i}^{(t)}, \\ 0, & e \notin \mathcal{A}_{\ell,i}^{(t)}. \end{cases} \tag{6}$$

Let $r_{\ell,i}^{(t)} = \sum_{e=1}^{M_\ell} \widetilde{p}_{\ell,i,e}^{(t)}$ denote the total affinity score of the active experts. We normalize $\widetilde{\mathbf{p}}_{\ell,i}^{(t)}$ to obtain the token-wise routing vector:

$$\mathbf{p}_{\ell,i}^{(t)} = \begin{cases} \widetilde{\mathbf{p}}_{\ell,i}^{(t)} / r_{\ell,i}^{(t)}, & r_{\ell,i}^{(t)} > 0, \\ \mathbf{0}, & r_{\ell,i}^{(t)} = 0. \end{cases} \tag{7}$$

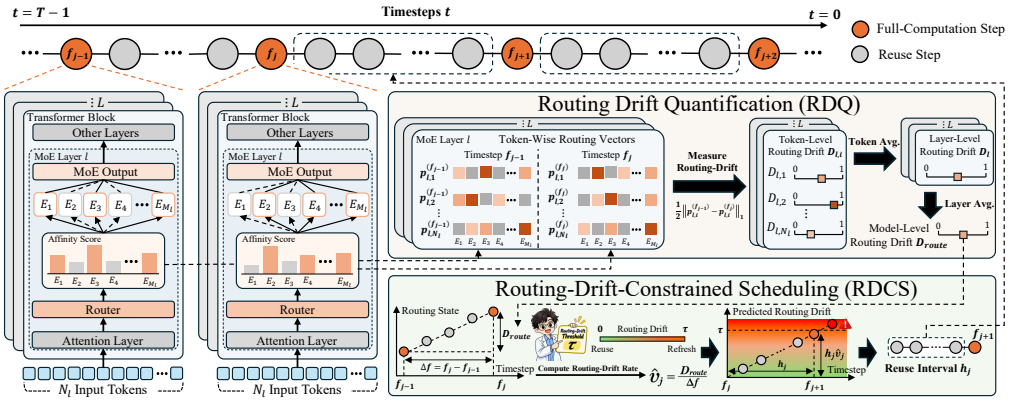


Figure 3: Overview of RoDCache. RDQ quantifies model-level routing drift by aggregating cross-step changes in routing vectors across tokens and MoE layers. Using this drift signal, RDCS then schedules the next reuse interval under a tunable routing-drift threshold.

The zero-vector branch accounts for tokens not selected by any dynamic expert under expert-choice routing. Otherwise, the nonzero entries identify the active experts, while their values are the corresponding normalized affinity scores.

For token i at layer ℓ, RDQ measures its routing drift between successive full-computation steps f_{j-1} and f_j using the normalized L_1 distance between their routing vectors:

$$D_{\ell,i}(f_{j-1}, f_j) = \frac{1}{2} \left\| \mathbf{p}_{\ell,i}^{(f_{j-1})} - \mathbf{p}_{\ell,i}^{(f_j)} \right\|_1 . \tag{8}$$

The token-level routing drift lies in $[0, 1]$, providing a common scale across tokens and MoE layers. A value of 1 indicates that the active-expert sets at the two steps do not overlap. We aggregate these values first over the N_ℓ tokens within each layer and then over the L MoE layers:

$$D_\ell(f_{j-1}, f_j) = \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} D_{\ell,i}(f_{j-1}, f_j), \tag{9}$$

$$D_{\text{route}}(f_{j-1}, f_j) = \frac{1}{L} \sum_{\ell=1}^{L} D_\ell(f_{j-1}, f_j). \tag{10}$$

D_ℓ and D_{route} denote routing drift at the layer and model levels, respectively. This two-level aggregation combines routing drift across tokens and MoE layers into a model-level measure used to assess cache validity.

5.3 Routing-Drift-Constrained Scheduling

RDCS adaptively schedules cache reuse according to routing drift. Based on the drift between the two most recent full-computation steps, it estimates how long cached results will remain valid.

Specifically, $D_{\text{route}}(f_{j-1}, f_j)$ measures the routing drift observed between the two most recent full-computation steps f_{j-1} and f_j. We denote the length of this interval by $a_j = f_{j-1} - f_j$. Since a_j may vary, we divide the observed drift by the interval length to obtain the routing-drift rate:

$$\widehat{v}_j = \frac{D_{\text{route}}(f_{j-1}, f_j)}{a_j}. \tag{11}$$

Using this rate, RDCS predicts how much routing drift will accumulate over the next reuse interval. If the next full-computation step is scheduled at $f_j - h$, the predicted routing drift is $\widehat{D}_{\text{route}}(f_j, f_j - h) = h\widehat{v}_j$. The following theorem shows that the error between the predicted and actual routing drift is bounded.

Theorem 1. *Suppose that the local average routing-drift rate is continuously differentiable over $[f_j - h, f_j + a_j]$, and let M_j denote the maximum absolute value of its derivative on this interval.*

The error between the predicted and actual routing drift satisfies:

$$\left| D_{\text{route}}(f_j, f_j - h) - \widehat{D}_{\text{route}}(f_j, f_j - h) \right| \leq M_j h(a_j + h). \tag{12}$$

The proof is provided in Appendix B. Large predicted routing drift indicates that the cached residual is no longer valid. Therefore, we introduce a tunable routing-drift threshold $\tau > 0$ to constrain the predicted drift over each reuse interval. The length h of a candidate reuse interval must satisfy

$$h\widehat{v}_j \leq \tau. \tag{13}$$

To reduce the frequency of full computation while ensuring that the predicted routing drift does not exceed the threshold, we choose the largest integer h satisfying this constraint. The resulting reuse interval h_j and next full-computation timestep f_{j+1} are defined as

$$h_j = \max\left(1, \left\lfloor \frac{\tau}{\widehat{v}_j} \right\rfloor\right), \tag{14}$$

$$f_{j+1} = \max(f_j - h_j, 0). \tag{15}$$

The first $\max$ ensures that the next full computation is at least one step away, while the second prevents its timestep from falling below zero. An interval of length h_j contains $h_j - 1$ reuse steps. Thus, $h_j = 1$ means that the next denoising step performs full computation. For a fixed threshold τ, slower routing drift permits longer reuse, whereas faster drift triggers full computation sooner.

RDCS initializes the routing-drift estimate by performing full computation at the first two denoising steps, $f_0 = T - 1$ and $f_1 = T - 2$, to obtain $\widehat{v}_1$. At each subsequent full-computation step f_j, RDCS updates $\widehat{v}_j$ and uses it to schedule f_{j+1}. This iterative scheduling continues until denoising is complete, with the routing-drift threshold τ as its sole tunable hyperparameter. A smaller τ triggers more frequent full computation, whereas a larger τ permits longer reuse intervals.

5.4 Time and Space Complexity Analysis

RoDCache stores only the latest residual and its routing state. For token-choice routing, RDQ retains K_ℓ active-expert indices and normalized affinity scores per token. Computing routing drift between successive full-computation steps requires $\mathcal{O}(\sum_{\ell=1}^{L} N_\ell K_\ell^2)$ time complexity and $\mathcal{O}(\sum_{\ell=1}^{L} N_\ell K_\ell)$ space complexity. For expert-choice routing, RDQ retains one M_ℓ-dimensional normalized routing vector per token. Constructing these vectors and computing routing drift require $\mathcal{O}(\sum_{\ell=1}^{L}(M_\ell C_\ell + N_\ell M_\ell))$ time complexity and $\mathcal{O}(\sum_{\ell=1}^{L} N_\ell M_\ell)$ space complexity. These RDQ costs apply only to full-computation steps, with none incurred at reuse steps. RDCS schedules the next reuse interval using scalar operations, requiring $\mathcal{O}(1)$ time complexity and space complexity. Overall, RoDCache introduces only small time and memory overhead relative to denoising-network execution.

6 Experiments

6.1 Experimental Setup

Experimental settings. We conduct experiments on three sparse MoE diffusion models: Nucleus-Image (Akiti et al., 2026) and HiDream-I1 (Cai et al., 2025) for image generation, and LingBot-Video (Ma et al., 2026) for video generation. Nucleus-Image uses expert-choice routing, whereas HiDream-I1 and LingBot-Video use token-choice routing. We evaluate the image models on all 200 DrawBench (Saharia et al., 2022) prompts and the video model on all 946 VBench (Huang et al., 2024) prompts. All methods use identical prompts, initial noise, official samplers, and numerical precision, with their parameters fixed across prompts. Complete inference configurations and hardware details are provided in Appendix C.1.

Baselines. We compare against two non-caching baselines, Vanilla and Vanilla (50% steps), and four cache-based methods: TeaCache (Liu et al., 2025a), SeaCache (Chung et al., 2026), DiCache (Bu et al., 2026), and DPCache (Cui et al., 2026). Vanilla serves as the full-computation reference, whereas Vanilla (50% steps) halves the denoising steps without cache reuse. Each method is evaluated under a moderate *slow* setting and a more aggressive *fast* setting. For RoDCache, the *slow*/*fast* settings use $\tau = 0.55/0.85$, $0.25/0.40$, and $0.65/0.80$ on Nucleus-Image, HiDream-I1, and LingBot-Video, respectively. Baseline configurations are provided in Appendix C.2.

Table 2: Quantitative comparison with baselines on sparse MoE image and video diffusion models. The CLIP/VBench Score column reports CLIP Score for the image models and VBench Score for the video model. The best and second-best results among caching methods under each setting appear in **bold** and underlined, respectively. Red parenthesized values in the RoDCache rows show the advantage of RoDCache over the corresponding second-best result.

Method	Efficiency			Visual Quality				
	FLOPs(T)↓	Latency(s)↓	Speedup↑	CLIP/VBench Score↑	Image Reward↑	PSNR↑	SSIM↑	LPIPS↓
Nucleus-Image								
Vanilla	1431.43	76.82	1.00	35.1062	1.0745	∞	1.0000	0.0000
Vanilla (50% steps)	715.72	42.06	1.83	33.5242	0.7983	13.769	0.6666	0.4209
TeaCache-*slow*	377.94	20.56	3.74	34.8502	1.0296	18.253	0.7751	0.2383
DPCache-*slow*	372.22	20.27	3.79	34.6666	**1.0398**	19.785	0.8180	0.1926
SeaCache-*slow*	372.22	20.41	3.76	35.0030	1.0349	21.613	0.8324	0.1673
DiCache-*slow*	404.34	20.20	3.80	34.8951	1.0332	20.052	0.8207	0.1771
RoDCache-*slow*	**365.16**	**19.87**	**3.87**	**35.0381**	1.0372	**23.819** (+2.206)	**0.8374** (+0.0050)	**0.1564** (-0.0109)
TeaCache-*fast*	286.32	15.74	4.88	32.6519	0.6061	13.384	0.6364	0.4658
DPCache-*fast*	286.32	15.68	4.90	34.2905	0.9478	17.215	0.7291	0.3124
SeaCache-*fast*	286.32	15.89	4.83	34.6954	0.9968	18.639	0.7609	0.2533
DiCache-*fast*	327.29	16.13	4.76	34.6372	0.9648	18.493	0.7607	0.2262
RoDCache-*fast*	**278.41**	**15.39**	**4.99**	**34.7001**	**1.0002**	**20.650** (+2.011)	**0.7623** (+0.0014)	**0.2222** (-0.0040)
HiDream-I1								
Vanilla	7776.89	144.41	1.00	35.3750	1.2938	∞	1.0000	0.0000
Vanilla (50% steps)	3888.44	72.27	2.00	35.1134	1.2778	14.240	0.6150	0.4031
TeaCache-*slow*	2177.53	40.97	3.52	**35.2504**	1.2584	18.319	0.7220	0.2569
DPCache-*slow*	2021.99	38.13	3.79	34.9548	1.2400	15.609	0.6363	0.3715
SeaCache-*slow*	1996.22	38.00	3.80	35.2116	1.2665	19.256	0.7363	0.2368
DiCache-*slow*	1990.69	37.57	3.84	34.6802	1.1409	15.263	0.6386	0.3884
RoDCache-*slow*	**1948.24**	**37.48**	**3.85**	35.2121	**1.2667**	**21.134** (+1.878)	**0.7563** (+0.0200)	**0.2147** (-0.0221)
TeaCache-*fast*	1555.38	29.70	4.86	34.9766	1.2138	15.901	0.6272	0.3750
DPCache-*fast*	1555.38	29.58	4.88	34.9992	1.1885	13.991	0.5625	0.4756
SeaCache-*fast*	1534.38	29.51	4.89	**35.2131**	1.2110	14.249	0.5897	0.4352
DiCache-*fast*	1533.88	29.21	4.94	33.3755	0.8675	13.737	0.5899	0.4901
RoDCache-*fast*	**1448.06**	**28.14**	**5.13**	35.0559	**1.2150**	**16.900** (+0.999)	**0.6396** (+0.0124)	**0.3598** (-0.0152)
LingBot-Video								
Vanilla	30854.61	188.13	1.00	82.09	–	∞	1.0000	0.0000
Vanilla (50% steps)	15427.31	97.28	1.93	80.80	–	16.124	0.6427	0.3513
TeaCache-*slow*	9701.42	63.86	2.94	79.24	–	14.780	0.6137	0.4278
DPCache-*slow*	10027.75	67.06	2.81	81.01	–	22.966	0.8004	0.1449
SeaCache-*slow*	10031.06	65.83	2.86	**81.16**	–	24.838	0.8304	0.1254
DiCache-*slow*	**9490.31**	63.90	2.94	80.84	–	23.382	0.8058	0.1512
RoDCache-*slow*	9627.70	**62.44**	**3.01**	81.09	–	**27.442** (+2.604)	**0.8553** (+0.0249)	**0.1206** (-0.0048)
TeaCache-*fast*	8455.84	54.57	3.45	77.55	–	14.299	0.6042	0.4654
DPCache-*fast*	**7713.65**	52.51	3.58	80.70	–	20.009	0.7247	0.2392
SeaCache-*fast*	8424.77	58.50	3.22	80.74	–	22.665	0.7829	0.1783
DiCache-*fast*	8188.50	55.47	3.39	80.26	–	22.575	0.7884	0.1816
RoDCache-*fast*	7880.51	**52.46**	**3.59**	**80.87**	–	**25.427** (+2.762)	**0.8220** (+0.0336)	**0.1529** (-0.0254)

Evaluation metrics. For image generation, we report CLIP Score (Hessel et al., 2021) and ImageReward (Xu et al., 2023), while VBench Score (Huang et al., 2024) is reported for video generation. For both tasks, PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) measure fidelity to the Vanilla outputs. We report FLOPs, end-to-end latency, and speedup relative to Vanilla to measure inference efficiency. To analyze computation allocation, we rank prompts by the PSNR between Vanilla (50% steps) and Vanilla, defining the bottom and top 10% as the fixed Hard-10% and Easy-10% subsets, respectively. H/E denotes the ratio of average FLOPs between the Hard-10% and Easy-10% subsets, while Hard–Easy Gap denotes their absolute differences in PSNR, SSIM, and LPIPS. Detailed metric definitions and evaluation procedures are provided in Appendix C.3.

6.2 Quality and Efficiency for Image and Video Generation

Table 2 summarizes the results for image and video generation. RoDCache achieves speedups of up to $3.87\times$ under the *slow* setting and $5.13\times$ under the *fast* setting. Relative to the best caching baseline for each metric under the same setting, RoDCache achieves gains of up to 2.76 dB in PSNR and 0.0336 in SSIM, together with an LPIPS reduction of up to 0.0254. It also maintains competitive CLIP Score, ImageReward, and VBench Score, demonstrating a favorable balance between visual quality and inference efficiency. More qualitative comparisons are provided in Appendix F.

Table 3: Computation allocation and visual quality across prompt difficulty levels on HiDream-I1.

Method	FLOPs(T)			Hard-10%			Easy-10%			Hard–Easy Gap		
	Hard	Easy	H/E↑	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↓	SSIM↓	LPIPS↓
TeaCache-*slow*	2177.53	2177.53	1.000	15.584	0.6655	0.3242	20.844	**0.7674**	0.2059	5.260	0.1019	0.1183
DPCache-*slow*	2021.99	2021.99	1.000	13.121	0.5772	0.4464	18.530	0.7109	0.2744	5.409	0.1337	0.1720
SeaCache-*slow*	2009.78	1970.89	1.020	16.343	0.6850	0.2869	21.818	0.7662	**0.1993**	5.475	0.0812	0.0876
DiCache-*slow*	1998.36	1975.50	1.012	12.863	0.5778	0.4523	17.170	0.6988	0.3155	4.307	0.1210	0.1368
RoDCache-*slow*	2002.23	1821.69	**1.099**	**19.675** (+3.332)	**0.7585** (+0.0735)	**0.1969** (-0.0900)	**22.825**	0.7504	0.2222	**3.150** (-1.157)	**0.0081** (-0.0731)	**0.0253** (-0.0623)
TeaCache-*fast*	1555.38	1555.38	1.000	14.351	0.6151	0.3818	18.872	0.7069	0.2786	4.521	0.0918	0.1032
DPCache-*fast*	1555.38	1555.38	1.000	12.061	0.5170	0.5213	16.691	0.6436	0.3906	4.630	0.1266	0.1307
SeaCache-*fast*	1539.82	1508.72	1.021	10.490	0.4829	0.5790	18.358	**0.7103**	**0.2720**	7.868	0.2274	0.3070
DiCache-*fast*	1536.93	1536.93	1.000	11.796	0.5290	0.5576	15.449	0.6566	0.3979	3.653	0.1276	0.1597
RoDCache-*fast*	1555.38	1384.29	**1.124**	**15.980** (+1.629)	**0.6384** (+0.0233)	**0.3383** (-0.0435)	**19.164**	0.6642	0.3197	**3.184** (-0.469)	**0.0258** (-0.0660)	**0.0186** (-0.0846)

Table 4: Cache-validity signal comparison and routing-drift threshold analysis on HiDream-I1.

Components	Setting	Efficiency		Visual Quality				
		Latency(s)↓	Speedup↑	CLIP Score↑	Image Reward↑	PSNR↑	SSIM↑	LPIPS↓
(a) Cache-validity signal	Input Drift	40.11	3.60	34.8420	1.0400	12.789	0.5397	0.5365
	Residual Drift	38.46	3.75	**35.2485**	1.2610	20.626	0.7440	0.2354
	Joint Drift	38.27	3.77	35.2104	1.2599	20.585	0.7391	0.2337
	Routing Drift	**37.48**	**3.85**	35.2121	**1.2667**	**21.134** (+0.508)	**0.7563** (+0.0123)	**0.2147** (-0.0190)
(b) Routing-drift threshold	$\tau = 0.15$	48.35	2.99	**35.2644**	**1.2816**	**24.083**	**0.8286**	**0.1416**
	$\tau = 0.20$	42.02	3.44	35.2004	1.2673	22.296	0.7846	0.1850
	$\boldsymbol{\tau = 0.25}$	37.48	3.85	35.2121	1.2667	21.134	0.7563	0.2147
	$\tau = 0.30$	34.28	4.21	35.2321	1.2405	20.042	0.7254	0.2464
	$\tau = 0.35$	31.19	4.63	35.1055	1.2179	18.582	0.6849	0.2974
	$\boldsymbol{\tau = 0.40}$	**28.14**	**5.13**	35.0559	1.2150	16.900	0.6396	0.3598

6.3 Computation Allocation Across Hard and Easy Prompts

Hard prompts require more frequent full computation to maintain output quality, while easy prompts tolerate more cache reuse. We assess computation allocation by comparing each method's efficiency and visual quality on the fixed Hard-10% and Easy-10% subsets.

Table 3 reports the HiDream-I1 results, with Nucleus-Image and LingBot-Video results provided in Appendix D. RoDCache achieves an H/E ratio of up to 1.124, versus at most 1.021 for the baselines. While remaining competitive on Easy-10%, RoDCache outperforms the best baselines on Hard-10%, improving PSNR and SSIM by up to 3.332 dB and 0.0735, respectively, and reducing LPIPS by up to 0.0900 under the same setting. Consequently, RoDCache reduces the corresponding Hard–Easy Gaps by up to 1.157 dB, 0.0731, and 0.0846, respectively, demonstrating more consistent quality across prompt difficulty levels. Further analysis is provided in Appendix E.

6.4 Effectiveness of Routing Drift

Cache-validity signal. We compare Routing Drift with three alternatives measured between successive full-computation steps: Input Drift and Residual Drift measure changes in latent inputs and transformer residuals, respectively, while Joint Drift combines residual and routing drift. Table 4(a) shows that Routing Drift achieves the lowest latency, outperforming the second-best results by 0.508 dB in PSNR and 0.0123 in SSIM, with 0.0190 lower LPIPS. Residual Drift's slightly higher CLIP Score indicates stronger prompt alignment but greater deviation from Vanilla outputs.

Routing-drift threshold. We vary the routing-drift threshold τ from 0.15 to 0.40. As shown in Table 4(b), increasing τ across this range increases speedup from 2.99× to 5.13×. Meanwhile, PSNR and SSIM decrease from 24.083 dB and 0.8286 to 16.900 dB and 0.6396, respectively, while LPIPS increases from 0.1416 to 0.3598. Thus, τ directly controls the trade-off between inference efficiency and visual quality.

7 Conclusion

We introduce RoDCache, a training-free caching method for sparse MoE diffusion models. Routing Drift Quantification (RDQ) aggregates changes in expert assignments and routing weights into a model-level routing-drift signal, which Routing-Drift-Constrained Scheduling (RDCS) uses to schedule cache reuse subject to a tunable threshold. Experiments show that RoDCache achieves a favorable balance between visual quality and inference efficiency across the evaluated models, as well as more consistent quality across hard and easy prompts.

AI USE STATEMENT

Artificial intelligence (AI) technologies were used only at the final stage of manuscript preparation to improve grammatical accuracy, clarity, and readability. Their use was limited to language refinement and did not extend to the development of research ideas, arguments, analysis, or conclusions. All substantive content and intellectual contributions presented in this paper are solely those of the authors.

ETHICS STATEMENT

We propose a training-free inference method. Its evaluation relies solely on publicly released pre-trained models and established benchmarks. We neither create nor distribute new datasets or sensitive material. No human or animal subjects are involved in this study. We do not collect or process personally identifiable information. All models and benchmarks are used in accordance with their licenses and terms of use. We also follow established data-governance and research-ethics practices to minimize potential privacy and ethical risks.

REPRODUCIBILITY STATEMENT

We facilitate reproducibility by providing the complete inference procedure and mathematical formulation in Sections 3 and 5. Experimental settings, baseline configurations, and evaluation procedures are documented in Appendices C.1, C.2, and C.3, respectively. Our implementation code will be made available through the repository referenced in the abstract.

REFERENCES

Chandan Akiti, Ajay Modukuri, Murali Nandan Nagarapu, Gunavardhan Akiti, and Haozhe Liu. Nucleus-Image: Sparse MoE for image generation. *arXiv preprint arXiv:2604.12163*, 2026.

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

Qi Cai, Yehao Li, Yingwei Pan, Ting Yao, and Tao Mei. HiDream-I1: An open-source high-efficient image generative foundation model. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 13636–13639. ACM, 2025. doi: 10.1145/3746027.3756870.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations*, 2024.

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*, pp. 14283–14294, 2026.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43632–43642, June 2026.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 1280–1297, 2024.

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 *Proceedings of the 41st International Conference on Machine*

Learning, volume 235 of *Proceedings of Machine Learning Research*, pp. 12606–12633. PMLR, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Huanlin Gao, Ping Chen, Fuyuan Shi, Ruijia Wu, Li YanTao, Qiang Hui, Yuren You, Ting Lu, Chao Tan, Shaoan Zhao, Zhaoxiang Liu, et al. MeanCache: From instantaneous to average velocity for accelerating flow matching inference. In *International Conference on Learning Representations*, 2026.

Yasaman Haghighi and Alexandre Alahi. SenCache: Accelerating diffusion model inference via sensitivity-aware caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14295–14304, June 2026.

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.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. VBench: Comprehensive benchmark suite for video generative models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21807–21818. IEEE, 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 *2025 IEEE/CVF International Conference on Computer Vision*, pp. 15240–15252. IEEE, 2025.

Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. BK-SDM: A lightweight, fast, and cheap version of Stable Diffusion. In *European Conference on Computer Vision*, pp. 381–399. Springer, 2024.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pp. 1097–1105, 2012.

Maoliang Li, Haojing Chen, Jiayu Chen, Zihao Zheng, Xinhao Sun, Hailong Zou, and Xiang Chen. MoECa: Aligning feature reuse with expert decomposition in diffusion transformers. *arXiv preprint arXiv:2606.15615*, 2026.

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

Jiacheng Liu, Xinyu Wang, Yuqi Lin, Zhikai Wang, Peiru Wang, Peiliang Cai, Qinming Zhou, Zhengan Yan, Zexuan Yan, Zhengyi Shi, et al. A survey on cache methods in diffusion models: Toward efficient multi-modal generation. *arXiv preprint arXiv:2510.19755*, 2025b.

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

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

Shuailei Ma, Jiaqi Liao, Xinyang Wang, Jingjing Wang, Chaoran Feng, Zijing Hu, Chong Bao, Zichen Xi, Yuqi Gan, Weisen Wang, et al. Scaling mixture-of-experts video pretraining for embodied intelligence. *arXiv preprint arXiv:2607.07675*, 2026.

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. *Transactions on Machine Learning Research*, 2025a.

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: 38592–38624, 2025b. doi: 10.52202/085713-1153.

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

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, 2015.

Amirmojtaba Sabour, Sanja Fidler, and Karsten Kreis. Align your steps: Optimizing sampling schedules in diffusion models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42947–42975. PMLR, 21–27 Jul 2024.

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.

Tho Tran Huu, Huu-Tuan Nguyen, Thien-Hai Nguyen, Nhat-Tri Ho, Viet-Hoang Tran, Tho Quan, and Tan Minh Nguyen. Geometric and stochastic analysis of discontinuities in sparse mixture-of-experts. *arXiv preprint arXiv:2606.19036*, 2026.

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

Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, et al. SANA: Efficient high-resolution image synthesis with linear diffusion transformers. In *The Thirteenth International Conference on Learning Representations*, 2025.

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. In *International Conference on Learning Representations*, 2025.

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

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12): 3848–3860, 2023. doi: 10.14778/3611540.3611569.

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.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Haowei Zhu, Dehua Tang, Ji Liu, Mingjie Lu, Jintu Zheng, Jinzhang Peng, Dong Li, Yu Wang, Fan Jiang, Lu Tian, et al. DiP-GO: A diffusion pruner via few-step gradient optimization. *Advances in Neural Information Processing Systems*, 37:92581–92604, 2024.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In *International Conference on Learning Representations*, 2025.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking token-wise feature caching: Accelerating diffusion transformers with dual feature caching. *IEEE Transactions on Image Processing*, 35:6211–6220, 2026. doi: 10.1109/TIP.2026.3698363.

RoDCache: Routing-Drift-Aware Caching for Accelerating Sparse MoE Diffusion Models

Appendix

A Protocol for Evaluating Cache-Validity Indicators

Experimental setup. We evaluate the two cache-validity indicators on all 200 DrawBench prompts using HiDream-I1. Images are generated at a resolution of 1024×1024 pixels using the official 50-step sampler with a CFG scale of 5 and a random seed of 42. The evaluated settings cover history gaps $g \in \{1, \dots, 24\}$ and cache-reuse lengths $s \in \{1, \dots, 24\}$. For each pair (g, s), we include all candidate cache timesteps t_c satisfying $t_c + g < 50$ and $t_c - s \geq 0$.

Drift definitions. For each prompt, let $\mathbf{R}^{(t)}$ denote the exact transformer residual at timestep t. Following Section 5.2, the historical routing drift is computed as

$$D_{\text{route}}(t_c + g, t_c) = \frac{1}{L} \sum_{\ell=1}^{L} \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} \frac{1}{2} \left\| \mathbf{p}_{\ell,i}^{(t_c+g)} - \mathbf{p}_{\ell,i}^{(t_c)} \right\|_1 . \tag{16}$$

The corresponding historical residual drift is

$$D_{\text{res}}(t_c + g, t_c) = \frac{\left\| \mathbf{R}^{(t_c+g)} - \mathbf{R}^{(t_c)} \right\|_1}{\left\| \mathbf{R}^{(t_c+g)} \right\|_1 + \left\| \mathbf{R}^{(t_c)} \right\|_1}. \tag{17}$$

To measure cache validity after reusing the residual for s denoising steps, we define the subsequent residual deviation as

$$E(t_c, s) = \frac{\left\| \mathbf{R}^{(t_c-s)} - \mathbf{R}^{(t_c)} \right\|_1}{\left\| \mathbf{R}^{(t_c-s)} \right\|_1 + \left\| \mathbf{R}^{(t_c)} \right\|_1}. \tag{18}$$

A larger $E(t_c, s)$ indicates greater error from reusing the residual cached at t_c and therefore lower cache validity.

Evaluation protocol. For each fixed (g, s, t_c), we independently rank the 200 prompts by historical routing drift, historical residual drift, and subsequent residual deviation. The top-ranked 10% of prompts form $\mathcal{Q}_{\text{route}}$, $\mathcal{Q}_{\text{res}}$, and $\mathcal{Q}_{\text{dev}}$, respectively. The large-deviation prompts uniquely identified by routing drift and residual drift are

$$\mathcal{H}_{\text{route}} = (\mathcal{Q}_{\text{dev}} \cap \mathcal{Q}_{\text{route}}) \setminus \mathcal{Q}_{\text{res}}, \tag{19}$$

$$\mathcal{H}_{\text{res}} = (\mathcal{Q}_{\text{dev}} \cap \mathcal{Q}_{\text{res}}) \setminus \mathcal{Q}_{\text{route}}. \tag{20}$$

For each (g, s), we aggregate $|\mathcal{H}_{\text{route}}|$ and $|\mathcal{H}_{\text{res}}|$ over all eligible cache timesteps t_c and normalize their totals to sum to one. For each s, we average the proportions uniformly across all g, ensuring that every history gap contributes equally regardless of its number of eligible cache timesteps.

B Proof of the Routing-Drift Prediction Error Bound

Theorem. Suppose that the local average routing-drift rate is continuously differentiable over $[f_j - h, f_j + a_j]$, and let M_j denote the maximum absolute value of its derivative on this interval. The error between the predicted and actual routing drift satisfies

$$\left| D_{\text{route}}(f_j, f_j - h) - \widehat{D}_{\text{route}}(f_j, f_j - h) \right| \leq M_j h (a_j + h). \tag{21}$$

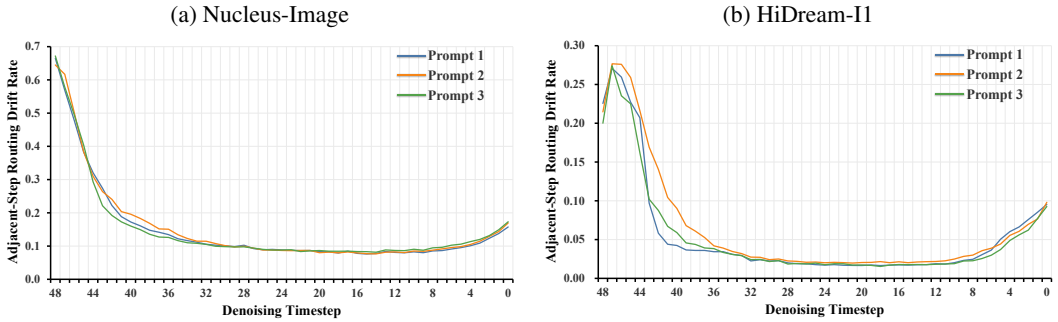


Figure 4: Adjacent-step routing-drift rates on (a) Nucleus-Image and (b) HiDream-I1. Each curve traces $D_{\text{route}}(t+1,t)$ across denoising timesteps for one randomly selected DrawBench prompt.

Proof. For any denoising timestep $t \neq f_j$, let

$$r_j(t) = \frac{D_{\text{route}}(f_j, t)}{|f_j - t|} \tag{22}$$

denote the average routing-drift rate between f_j and t. We assume that the discrete samples of $r_j(t)$ admit a continuously differentiable interpolation over $[f_j - h, f_j + a_j]$, including a continuous extension at $t = f_j$. Let M_j denote the maximum absolute rate of change of $r_j(t)$ over this interval:

$$M_j = \max_{t \in [f_j - h, f_j + a_j]} |r_j'(t)|. \tag{23}$$

Let v_j denote the actual average routing-drift rate over the next reuse interval. Because $a_j = f_{j-1} - f_j$ and routing drift is symmetric in its two endpoints, the mean value theorem gives

$$|v_j - \widehat{v}_j| = |r_j(f_j - h) - r_j(f_j + a_j)| \leq M_j(a_j + h). \tag{24}$$

The actual and predicted routing drifts over the next reuse interval are hv_j and $h\widehat{v}_j$, respectively. Their absolute error therefore satisfies

$$\left| D_{\text{route}}(f_j, f_j - h) - \widehat{D}_{\text{route}}(f_j, f_j - h) \right| = h|v_j - \widehat{v}_j| \leq M_j h(a_j + h), \tag{25}$$

which proves Theorem 1. □

Empirical analysis. To empirically examine how routing-drift rates vary over the denoising process, we randomly select three DrawBench prompts for Nucleus-Image with expert-choice routing and HiDream-I1 with token-choice routing. For each prompt, we compute the routing drift $D_{\text{route}}(t+1,t)$ between every pair of consecutive timesteps throughout the denoising process. Since consecutive timesteps are separated by a unit interval, this quantity also represents the average routing-drift rate over the corresponding interval.

As shown in Fig. 4, adjacent-step routing-drift rates vary smoothly on both models over most of the denoising process, supporting the assumption that the discrete samples of $r_j(t)$ admit a continuously differentiable interpolation. This smooth variation further suggests that M_j is typically small, yielding a small upper bound on the routing-drift prediction error. The few noticeable departures from this smooth pattern occur mainly at the beginning of denoising, where the routing-drift rate is relatively large. RDCS performs full computation at the first two denoising steps, so the initial high-drift regime is directly reflected in $\widehat{v}_1$. A large rate estimate yields a short reuse interval, often with $h_j = 1$. Therefore, RDCS tends to perform consecutive full-computation steps at the beginning of denoising, limiting prediction error when routing-drift rates are high and change rapidly.

C Experimental and Reproducibility Details

C.1 Experimental Settings

All experiments use a random seed of 42 and a batch size of 1. For each model, all methods share the same prompts, initial noise, official sampler, and numerical precision. For image generation, we

Table 5: Method configurations for each model and efficiency setting.

Model	Method	Efficiency Setting	
		slow	***fast***
Nucleus-Image	TeaCache	$\delta = 0.8$	$\delta = 1.1$
	DPCache	$K = 13,\ O = 2,\ M = 3$	$K = 10,\ O = 2,\ M = 3$
	SeaCache	$\delta = 0.4,\ \beta = 2$	$\delta = 0.6,\ \beta = 2$
	DiCache	$\delta = 0.2,\ m = 1,\ \rho = 0.1$	$\delta = 0.3,\ m = 1,\ \rho = 0.1$
	RoDCache	$\tau = 0.55$	$\tau = 0.85$
HiDream-I1	TeaCache	$\delta = 0.8$	$\delta = 1.3$
	DPCache	$K = 13,\ O = 2,\ M = 3$	$K = 10,\ O = 2,\ M = 3$
	SeaCache	$\delta = 0.5,\ \beta = 2$	$\delta = 0.6,\ \beta = 2$
	DiCache	$\delta = 0.3,\ m = 1,\ \rho = 0.1$	$\delta = 0.5,\ m = 1,\ \rho = 0.1$
	RoDCache	$\tau = 0.25$	$\tau = 0.40$
LingBot-Video	TeaCache	$\delta = 0.5$	$\delta = 0.58$
	DPCache	$K = 13,\ O = 2,\ M = 4$	$K = 10,\ O = 2,\ M = 4$
	SeaCache	$\delta = 0.35,\ \beta = 3$	$\delta = 0.5,\ \beta = 3$
	DiCache	$\delta = 0.2,\ m = 1,\ \rho = 0.15$	$\delta = 0.3,\ m = 1,\ \rho = 0.15$
	RoDCache	$\tau = 0.65$	$\tau = 0.80$

use Nucleus-Image and HiDream-I1 to generate images at a resolution of 1024×1024 for all 200 DrawBench prompts. Both models use 50 denoising steps, with CFG set to 8 for Nucleus-Image and 5 for HiDream-I1. For video generation, we use LingBot-Video to generate two-second, 49-frame videos at a resolution of 640×480 pixels and a frame rate of 24 FPS for all 946 VBench prompts. LingBot-Video uses 40 denoising steps, with CFG and shift both set to 3. All image experiments are conducted on a single NVIDIA A800-SXM4-80GB GPU, while video experiments use two such GPUs with FSDP (Zhao et al., 2023).

C.2 Baseline Configurations

Table 5 summarizes the method-specific configurations for each model and efficiency setting. Each configuration remains fixed across all prompts. **Vanilla** performs full computation at every denoising step, whereas **Vanilla (50% steps)** uses half of the configured denoising steps without cache reuse. **TeaCache** uses the rescaling coefficients provided by the official implementation for HiDream-I1 and fits separate fourth-order polynomials for Nucleus-Image and LingBot-Video using 20 prompts per model, with δ denoting the caching threshold. **DPCache** constructs the Path-Aware Cost Tensor from 20 randomly selected prompts per model, with K, O, and M denoting the number of selected key steps, prediction order, and mandatory initial full-computation steps, respectively. **SeaCache** uses δ as the cache threshold and β as the frequency exponent of the power-law spectrum. **DiCache** uses δ as the reuse threshold, m as the probe depth, and ρ as the ratio of initial full-computation steps. **RoDCache** uses a single tunable routing-drift threshold τ to constrain the predicted drift over each reuse interval.

C.3 Evaluation Metrics and Procedures

Efficiency measurement. We measure end-to-end latency from the start of inference to completion, including text encoding, denoising, and VAE decoding but excluding model loading and output writing. CUDA is synchronized before and after each measurement. The first inference in each run serves as a warm-up and is excluded from the reported average. For LingBot-Video, latency is measured using FSDP inference across two GPUs, with FLOPs summed across both GPUs. Both metrics include all method-specific overhead.

Visual quality evaluation. For image generation, we compute CLIP Score (Hessel et al., 2021), ImageReward (Xu et al., 2023), PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) over all 200 prompts. CLIP Score is computed from the cosine similarity between prompt and image embeddings extracted by CLIP-ViT-g-14-laion2B-s12B-b42K, whereas ImageReward is the preference score assigned to each prompt-image pair by the official ImageReward model. PSNR, SSIM, and LPIPS use the Vanilla output generated with the same prompt and random seed as the reference. PSNR is computed from the mean squared pixel error in RGB space. SSIM compares

Table 6: Computation allocation and visual quality across prompt difficulty levels on Nucleus-Image and LingBot-Video. The best result appears in **bold**, whereas the second-ranked result is underlined. Red parenthesized values in the RoDCache rows show the advantage of RoDCache over the corresponding second-best result.

Method	FLOPs(T)			Hard-10%			Easy-10%			Hard–Easy Gap		
	Hard	Easy	H/E↑	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↓	SSIM↓	LPIPS↓
Nucleus-Image												
TeaCache-*slow*	379.31	375.06	1.011	14.318	0.6246	0.3808	22.167	0.8864	0.1485	7.849	0.2618	0.2323
DPCache-*slow*	372.16	372.20	1.000	16.689	0.7101	0.2843	22.952	0.8936	0.1378	6.263	0.1835	0.1465
SeaCache-*slow*	372.16	372.20	1.000	17.805	0.7153	0.2686	24.416	0.8879	0.1293	6.611	0.1726	0.1393
DiCache-*slow*	404.28	407.09	0.993	16.194	0.6938	0.2928	24.512	**0.9157**	**0.0994**	8.318	0.2219	0.1934
RoDCache-*slow*	389.29	337.80	**1.152**	**21.467** (+3.662)	**0.7848** (+0.0695)	**0.1914** (-0.0772)	**26.287**	0.9013	0.1148	**4.820** (-1.443)	**0.1165** (-0.0561)	**0.0766** (-0.0627)
TeaCache-*fast*	286.27	286.31	1.000	9.691	0.4376	0.6735	17.983	0.8127	0.2659	8.292	0.3751	0.4076
DPCache-*fast*	286.27	286.31	1.000	14.553	0.6046	0.4055	20.392	0.8365	0.2368	5.839	0.2319	0.1687
SeaCache-*fast*	286.27	286.31	1.000	15.187	0.6264	0.3766	22.504	0.8721	0.1659	7.317	0.2457	0.2107
DiCache-*fast*	327.64	329.07	0.996	15.181	0.6514	0.3427	22.471	**0.8862**	**0.1390**	7.290	0.2348	0.2037
RoDCache-*fast*	300.56	257.65	**1.166**	**18.225** (+3.038)	**0.6760** (+0.0246)	**0.2766** (-0.0661)	**23.542**	0.8594	0.1644	**5.317** (-0.522)	**0.1834** (-0.0485)	**0.1122** (-0.0565)
LingBot-Video												
TeaCache-*slow*	9729.74	9690.73	1.004	10.956	0.3781	0.5439	24.187	0.8365	0.2638	13.231	0.4584	0.2801
DPCache-*slow*	10052.63	9972.29	1.008	16.405	0.5708	0.2485	31.465	0.9517	0.0580	15.060	0.3808	0.1905
SeaCache-*slow*	10052.63	9972.29	1.008	18.197	0.6306	0.2160	33.477	0.9572	**0.0523**	15.280	0.3266	0.1637
DiCache-*slow*	9483.24	9565.27	0.991	17.571	0.6045	0.2413	31.299	0.9473	0.0643	13.728	0.3428	0.1770
RoDCache-*slow*	9702.57	9472.10	**1.024**	**22.031** (+3.834)	**0.7104** (+0.0798)	**0.1996** (-0.0164)	**34.660**	**0.9584**	0.0554	**12.629** (-0.602)	**0.2480** (-0.0786)	**0.1442** (-0.0195)
TeaCache-*fast*	8497.96	8389.87	1.013	10.688	0.3663	0.5907	24.459	0.8322	0.2865	13.771	0.4659	0.3042
DPCache-*fast*	7732.79	7670.99	1.008	15.004	0.5049	0.3395	26.593	0.9091	0.1304	11.589	0.4042	0.2091
SeaCache-*fast*	8408.42	8357.53	1.006	16.722	0.5576	0.2879	30.463	0.9403	0.0779	13.741	0.3827	0.2100
DiCache-*fast*	8216.09	8150.43	1.008	17.152	0.5818	0.2842	30.086	0.9394	0.0813	12.934	0.3577	0.2029
RoDCache-*fast*	8058.57	7598.63	**1.061**	**21.350** (+4.198)	**0.6437** (+0.0619)	**0.2531** (-0.0311)	**32.549**	**0.9470**	**0.0721**	**11.199** (-0.390)	**0.3033** (-0.0544)	**0.1809** (-0.0220)

local luminance, contrast, and structure on grayscale images. LPIPS measures perceptual distance using features extracted by AlexNet (Krizhevsky et al., 2012). For video generation, we compute VBench Score (Huang et al., 2024), PSNR, SSIM, and LPIPS over all 946 prompts. VBench Score aggregates 16 evaluation dimensions using the official normalization. Video PSNR, SSIM, and LPIPS are computed between generated and Vanilla frames at matching temporal positions, then averaged over all frames.

D Additional Results Across Prompt Difficulty

Table 6 extends the computation-allocation analysis to Nucleus-Image and LingBot-Video. The results on both models confirm the trend observed for HiDream-I1 in Section 6.3. RoDCache allocates more computation to Hard-10% prompts at both acceleration settings, whereas the baselines yield H/E ratios close to 1. On Easy-10%, RoDCache remains competitive, while on Hard-10% it achieves the best visual quality, resulting in the smallest Hard–Easy Gap across all three metrics.

E Analysis of Routing Drift Across Prompt Difficulty Levels

To further analyze how RoDCache allocates computation across prompt difficulty levels, we compare routing drift and full-computation schedules on the fixed Hard-10% and Easy-10% subsets. As shown in Fig. 5, Hard-10% prompts exhibit larger routing drift during early denoising, whereas the difference becomes small in the middle and later stages. The larger early-stage drift leads RDCS to schedule shorter reuse intervals for hard prompts, resulting in more frequent full-computation steps. Figure 6 confirms this behavior, showing higher early-stage full-computation ratios for Hard-10% prompts. This adaptive allocation helps explain RoDCache's more consistent quality across prompt difficulty levels.

F Additional Qualitative Results

We provide additional qualitative comparisons under both the *slow* and *fast* settings. Figures 7 and 8 show the results on Nucleus-Image. Figures 9 and 10 show the results on HiDream-I1. Figures 11, 12, 13, 14, 15, and 16 show the results on LingBot-Video.

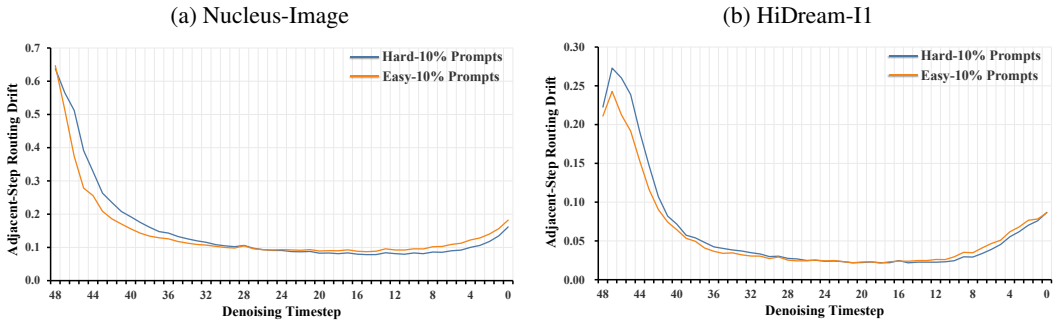


Figure 5: Adjacent-step routing drift across prompt difficulty levels on (a) Nucleus-Image and (b) HiDream-I1. Each curve shows the average $D_{\text{route}}(t+1, t)$ for the Hard-10% or Easy-10% subset at each denoising timestep.

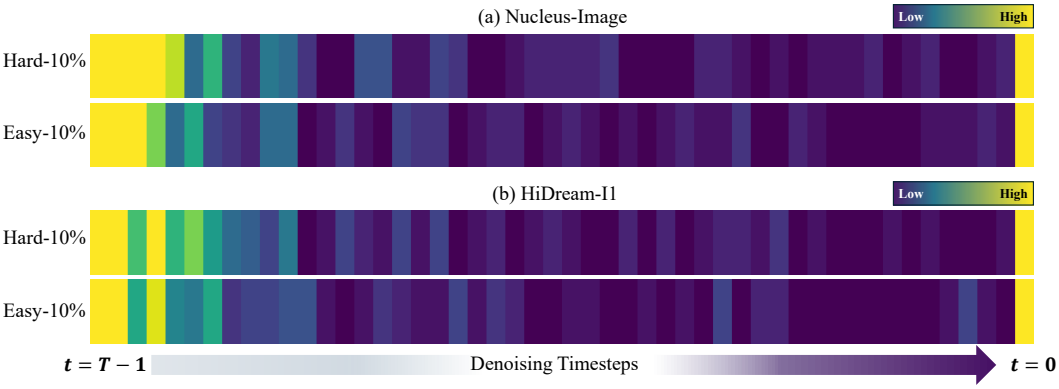


Figure 6: Full-computation patterns of RoDCache across denoising timesteps on (a) Nucleus-Image and (b) HiDream-I1 under the *fast* setting. Hard-10% prompts exhibit higher full-computation ratios during early denoising than Easy-10% prompts.

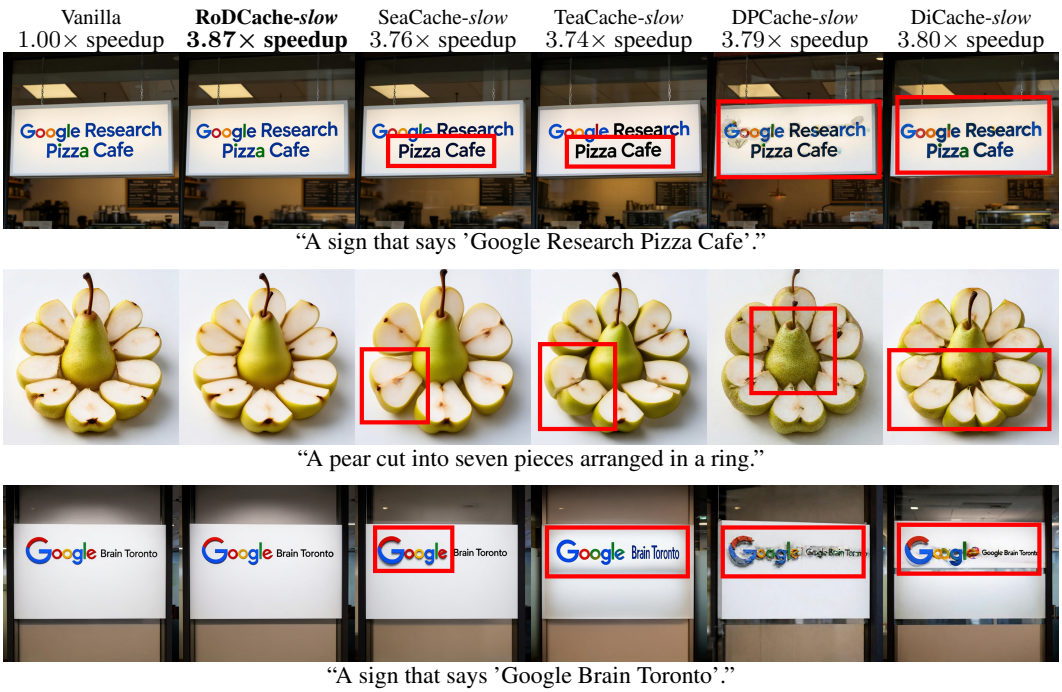


Figure 7: Additional qualitative comparisons on Nucleus-Image under the *slow* setting. Red boxes highlight visible baseline deviations from Vanilla outputs.



Figure 8: Additional qualitative comparisons on Nucleus-Image under the *fast* setting. Red boxes highlight visible baseline deviations from Vanilla outputs.



Figure 9: Additional qualitative comparisons on HiDream-I1 under the *slow* setting. Red boxes highlight visible baseline deviations from Vanilla outputs.

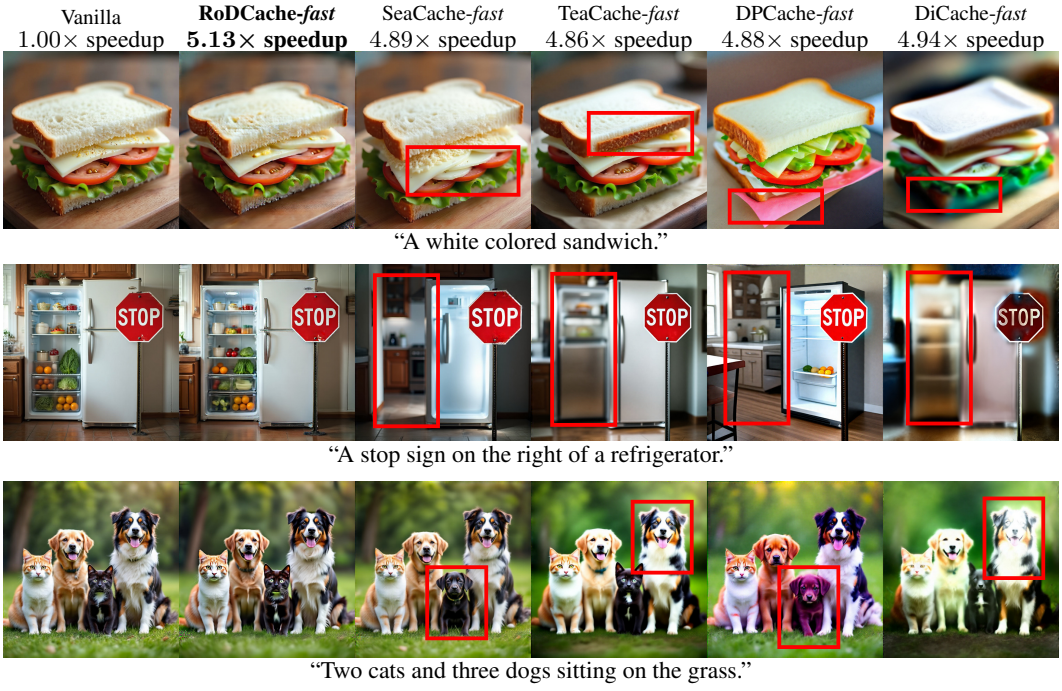


Figure 10: Additional qualitative comparisons on HiDream-I1 under the *fast* setting. Red boxes highlight visible baseline deviations from Vanilla outputs.

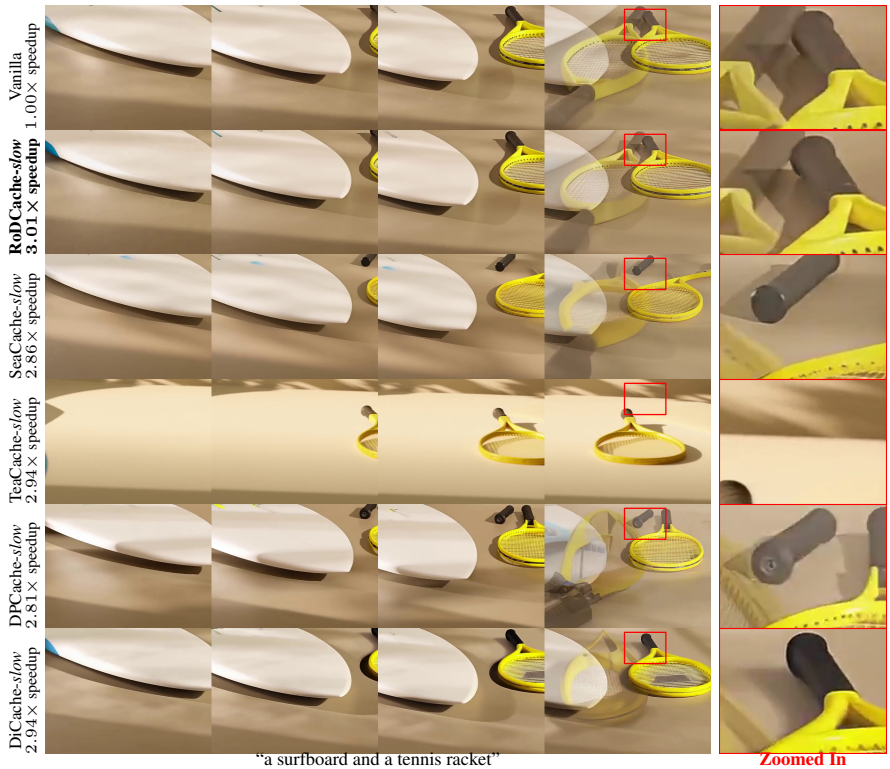


Figure 11: Additional qualitative comparisons on LingBot-Video under the *slow* setting (1/3). Zoom in for a clearer comparison of visual details across methods.



Figure 12: Additional qualitative comparisons on LingBot-Video under the *slow* setting (2/3). Zoom in for a clearer comparison of visual details across methods.

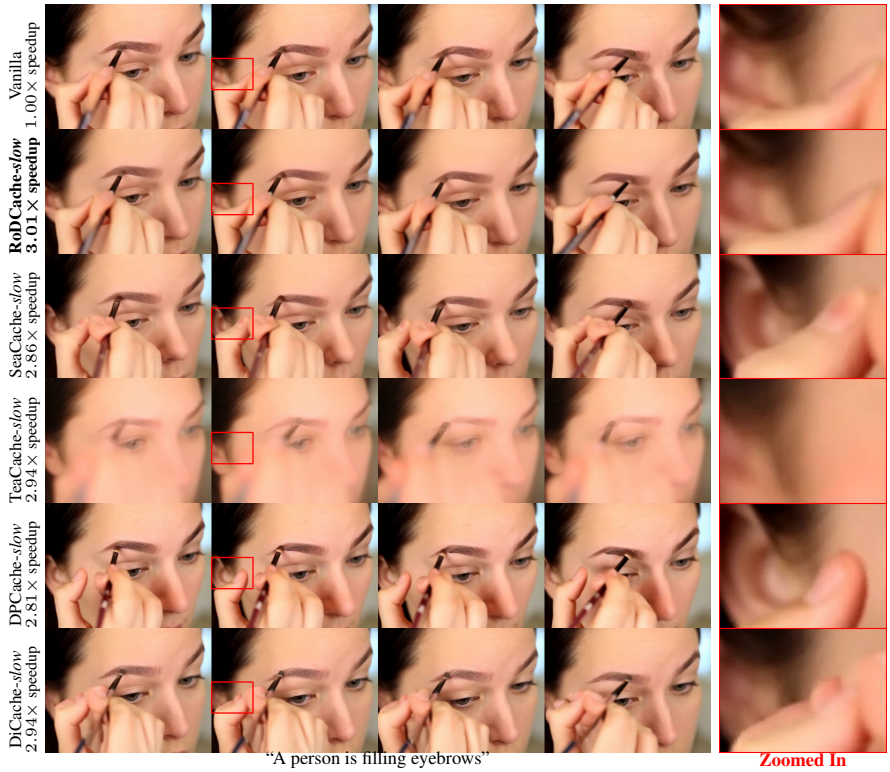


Figure 13: Additional qualitative comparisons on LingBot-Video under the *slow* setting (3/3). Zoom in for a clearer comparison of visual details across methods.

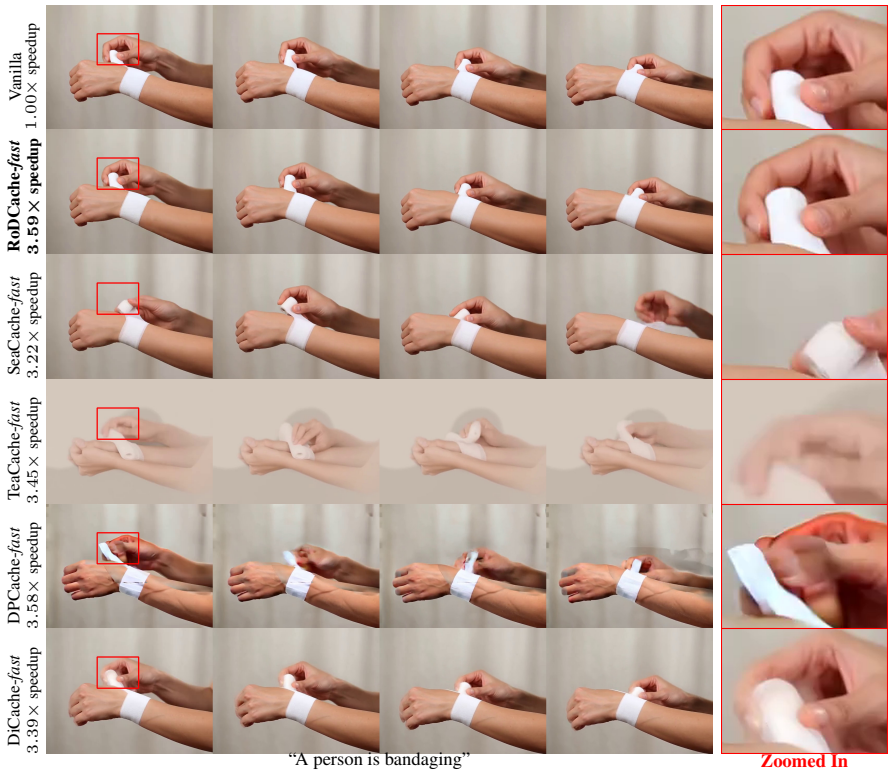


Figure 14: Additional qualitative comparisons on LingBot-Video under the *fast* setting (1/3). Zoom in for a clearer comparison of visual details across methods.



Figure 15: Additional qualitative comparisons on LingBot-Video under the *fast* setting (2/3). Zoom in for a clearer comparison of visual details across methods.



Figure 16: Additional qualitative comparisons on LingBot-Video under the *fast* setting (3/3). Zoom in for a clearer comparison of visual details across methods.