HoltSeer: Exponential Smoothing for Robust Cache-then-Forecast in Highly Accelerated Diffusion Sampling

Anonymous authors
Paper under double-blind review

Abstract

Feature caching offers training-free acceleration for diffusion models by skipping selected network evaluations. Cache-then-reuse methods substitute cached features, whereas cache-then-forecast methods predict their temporal evolution. At high acceleration ratios, computed features become sparse, making forecasts over the intervening timesteps increasingly sensitive to local modeling errors. The challenge is to represent local feature evolution accurately with a lightweight predictor under this limited computation budget. We propose **HoltSeer**, a training-free cache-then-forecast method that models local diffusion feature evolution through a smoothed level–trend state inspired by Holt's exponential smoothing. The level component summarizes recent cached features, the trend component tracks local directional changes, and a trend-scaling coefficient controls extrapolation. We implement this predictor through a bounded sequence of recursive forecasts with periodic full network evaluations; predicted features enter the same state update as computed features. Experiments on FLUX.1-dev, Qwen-Image, and Wan2.1 show improved reference-based fidelity over the evaluated caching baselines at large cache intervals while retaining competitive latency. Our code will be released upon acceptance.

1 Introduction

Diffusion models Ho et al. (2020), particularly those based on Diffusion Transformers (DiTs) Peebles and Xie (2023), have become a standard approach for high-fidelity visual generation. However, their high inference cost limits practical deployment at scale. To improve efficiency, various acceleration techniques have been proposed, including distillation Li et al. (2023c), quantization Li et al. (2023b); Shang et al. (2023), and feature caching. Among these, feature caching is appealing due to its training-free formulation, ease of integration, and compatibility with other acceleration methods, and has therefore attracted increasing interest in both practice and research. Generally, feature caching-based methods can be categorized into two paradigms: "cache-then-reuse" and "cache-then-forecast". They approach from different perspectives and exploit the temporal redundancy of diffusion features in distinct ways.

Cache-then-reuse. This paradigm accelerates diffusion sampling by directly reusing features computed at previous timesteps, thereby skipping redundant network computations. Despite its simplicity and efficiency, cache-then-reuse methods (Ma et al., 2024; Li et al., 2023a; Selvaraju et al., 2024; Chen et al., 2024; Zou et al., 2024; Liu et al., 2025b) share a fundamental limitation: they reuse cached features without explicitly accounting for their temporal evolution. As the acceleration ratio increases, the interval between feature computation and reuse becomes larger, making cached features increasingly stale and mismatched with the target timestep. Consequently, direct feature reuse becomes unreliable under high acceleration and often leads to noticeable degradation in generation quality.

Cache-then-forecast. To overcome the limitation of direct reuse, the cache-then-forecast paradigm considers the temporal evolution of diffusion features. Instead of copying stale cached features, it assumes that feature representations evolve smoothly across denoising timesteps and predicts skipped features from historical features and their temporal variations. By approximating the target features

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: Visual demonstration of HoltSeer on Qwen-Image, with 7.00× acceleration in FLOPs.

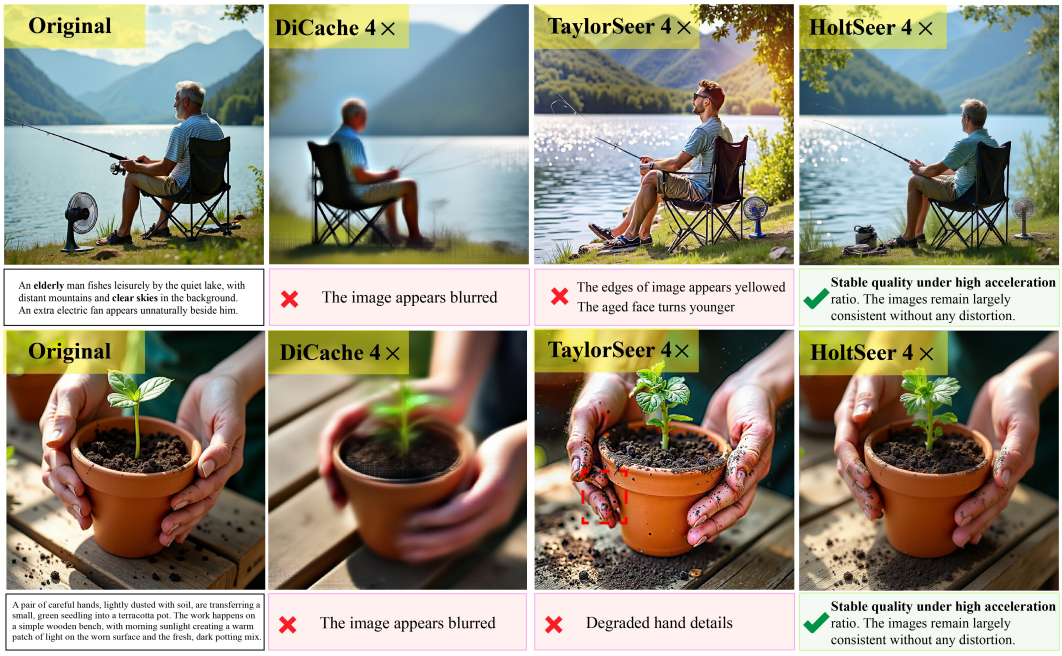

Figure 2: Visual comparison on FLUX.1-dev, highlighting the rapid quality degradation of cache-then-forecast methods at high acceleration ratios.

rather than simply reusing outdated ones, cache-then-forecast methods usually provide more faithful feature estimates and better preserve generation quality.

However, representative cache-then-forecast methods (Liu et al., 2025d; Feng et al., 2025; Bu et al., 2025; Liu et al., 2025c) estimate skipped features using computed anchors; see Appendix B for a detailed discussion. As the acceleration ratio increases, fewer network evaluations are available and the temporal gaps between computed anchors widen. Forecasting across these longer intervals becomes increasingly sensitive to local trajectory assumptions and sparse-anchor estimates, which can lead to severe visual degradation, as shown in Figure 2. Our goal is to develop a lightweight representation of local feature evolution that preserves generation fidelity when computed features are sparse.

We approach this problem through level–trend time-series modeling. Rather than relying on repeated estimates of higher-order slopes or curvatures, we summarize recent cached features using a smoothed level and a local trend. Holt's exponential smoothing (Holt, 2004) provides a classical model for this representation: the level captures the local baseline, and the trend captures directional changes between successive level estimates. We apply this representation over a local feature window.

Based on this perspective, we propose HoltSeer, a training-free method for diffusion feature forecasting. HoltSeer smooths recent cached features into a level–trend state and scales the trend contribution to control local extrapolation, as shown in Figure 3. To execute this predictor across skipped timesteps, we use a bounded rolling protocol: each predicted feature is inserted into the local input window, and a full network evaluation periodically supplies a new computed feature.

Overall, our main contributions are as follows:

- **Level–Trend Time-Series Feature Forecasting.** We propose HoltSeer, a training-free method that represents local diffusion feature evolution through Holt-inspired level and trend smoothing with scaled trend extrapolation. A bounded prediction-and-refresh protocol applies this predictor between full network evaluations.
- **Evaluation across Models and Modalities.** We evaluate HoltSeer on FLUX.1-dev, Qwen-Image, and Wan2.1 across multiple cache intervals. At the largest evaluated intervals, HoltSeer improves reference-based fidelity metrics over the evaluated caching baselines while retaining competitive latency. Within-method protocol comparisons and parameter sensitivity tests characterize its behavior in the evaluated settings.

2 Related Works

As diffusion models continue to scale, their inference cost increases substantially, posing growing efficiency challenges and motivating the development of acceleration methods. Among these, feature caching exploits the temporal similarity of features across adjacent denoising steps. By caching intermediate features and reusing or predicting them at subsequent timesteps, such methods effectively reduce redundant computation and improve inference efficiency.

2.1 Cache-then-reuse

Naïve Cache. Early static caching (Naïve Cache) reused features at fixed timesteps, causing exponential error accumulation due to ignored temporal dynamics. DeepCache Ma et al. (2024), Faster Diffusion Li et al. (2023a), and FORA Selvaraju et al. (2024) exploit inter-step feature similarity by caching deep U-Net features, encoder features, or Attention/MLP activations, respectively, while Δ-DiT Chen et al. (2024) reconstructs block outputs by reusing cached input–output residuals.

Dynamic Cache. Modern caching approaches overcome the limitations of the Naïve Cache by measuring feature differences between adjacent timesteps and dynamically deciding whether to reuse cached computations, thereby accounting for the temporal evolution of diffusion features and achieving improved performance. ToCa Zou et al. (2024) introduces token-level caching, determining cache reuse for each token at every layer based on comprehensive metrics. TeaCache proposes a timestep-embedding–aware strategy that estimates relative L1 distances, refines them via polynomial correction, and uses the accumulated difference to decide on cache reuse. DBCache follows a similar dynamic approach based on relative L1 distance and further incorporates acceleration techniques from Δ-DiT.

2.2 Cache-then-forecast

Unlike "cache-then-reuse", "cache-then-forecast" views diffusion as a smooth, continuous process with predictable dynamics and focuses on accurate feature approximation to reduce errors. Proposed by Liu et al. (2025d), this paradigm inspired several methods: TaylorSeer predicts future features via Taylor expansion on continuous trajectories. HiCache Feng et al. (2025) replaces the polynomial basis with Hermite polynomials for complex trajectories. Based on Adams–Bashforth multi-step integration, AB-Cache linearly weights cached features to predict features at future time steps. DiCache Bu et al. (2025) dynamically selects per-sample caching with trajectory alignment. FastCache Liu

et al. (2025a) skips computation using statistical similarity with learnable approximations and token pruning. And FreqCa Liu et al. (2025c) separates low and high frequency components to combine reuse and prediction. TC-Padé Cui et al. (2026) addresses the issues of feature similarity degradation and trajectory drift in low-step sampling on Padé rational function approximation.

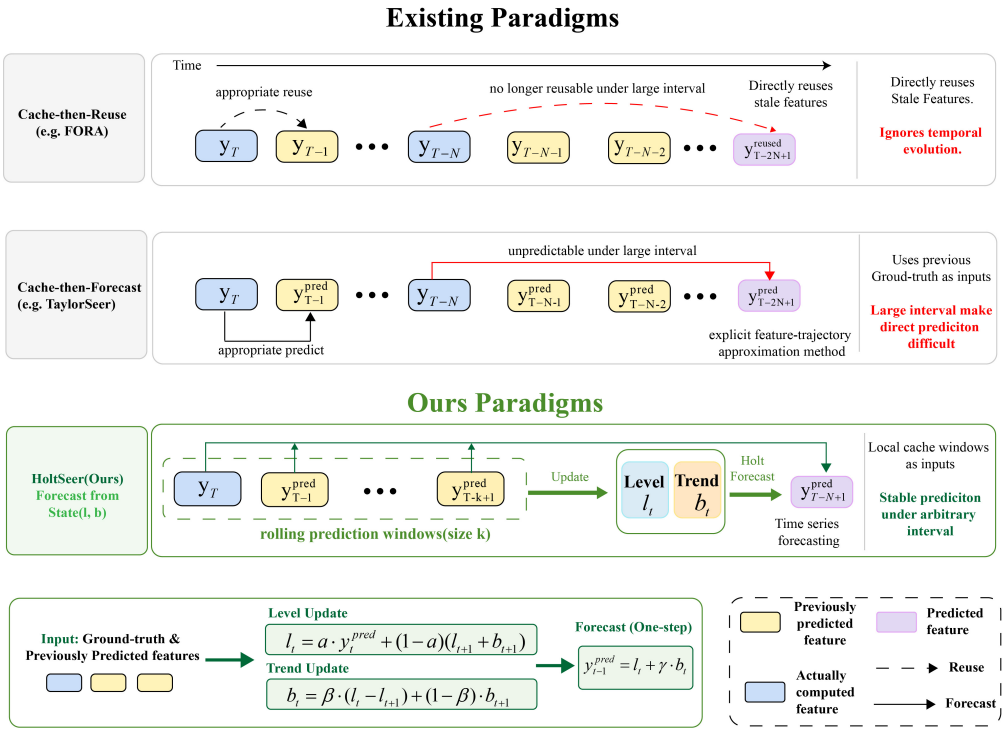

Figure 3: Overview of HoltSeer and the two feature-caching paradigms. Cache-then-reuse substitutes previously computed features, while cache-then-forecast estimates skipped features from temporal information. HoltSeer follows the latter paradigm using a smoothed level–trend representation and scaled trend extrapolation. Its execution protocol feeds adjacent computed and predicted features into a local window and periodically refreshes the inputs through a full network evaluation.

3 METHODOLOGY

3.1 PRELIMINARY

In this section, we first review the diffusion process and then state the local smoothness assumption and adjacent-step approximation that motivate HoltSeer.

Diffusion Model. A diffusion model consists of a forward process and a reverse process. The forward process incrementally adds Gaussian noise to a clean image $\boldsymbol{x}_0$, eventually yielding a fully noisy image $\boldsymbol{x}_T$. This process can be expressed as Lu et al. (2022):

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = N(\boldsymbol{x}_t; \sqrt{\alpha_t}\boldsymbol{x}_0, \sqrt{1-\alpha_t}I), \tag{1}$$

where α_t denotes the noise schedule parameter. As the timestep progresses from 0 to T, $\boldsymbol{x}_0$ gradually transforms into a Gaussian distribution. The reverse process performs step-by-step denoising to generate a clean image. Given two timesteps s and t, where $t < s$, according to the computational formula from Lu et al. (2022), this process can be expressed as:

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_\theta\left(\boldsymbol{x}_{t_\lambda(\lambda)}, t_\lambda(\lambda)\right) \mathrm{d}\lambda, \tag{2}$$

where $\lambda_t = log(\alpha_t/\sigma_t)$, representing one half of the log-SNR. $t_\lambda(\lambda)$ is the inverse function of λ and $t_\lambda(\lambda_t) = t$. $\hat{\boldsymbol{\epsilon}}_\theta(\cdot)$ is the output of the prediction network, denoting noise or vector field. According to Lu et al. (2022), the integral term in Equation (2) can be approximated using Taylor expansion, with the first-order case as an example, the formula is as follows:

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \sigma_t\left(e^{\lambda_t-\lambda_s}-1\right)\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}_s, s\right). \tag{3}$$

Assumption 3.1. Within each bounded cache interval between successive full network evaluations, the residual-feature sequence is locally smooth with respect to the discretized diffusion time. Consequently, adjacent residual features admit a short-horizon approximation with a finite local residual. This assumption does not require the complete diffusion trajectory to be globally linear or free of abrupt changes.

Proposition 3.2. *Yu et al. (2025) Consider adjacent log-SNR points with equal local spacing $h = \lambda_t - \lambda_s = \lambda_s - \lambda_o$. Under Assumption 3.1, their outputs admit the following local scaled approximation:*

$$\hat{\boldsymbol{\epsilon}}_\theta\left(\boldsymbol{x}_{t_\lambda(\lambda_s)}, t_\lambda(\lambda_s)\right) = k_t \cdot \hat{\boldsymbol{\epsilon}}_\theta\left(\boldsymbol{x}_{t_\lambda(\lambda_o)}, t_\lambda(\lambda_o)\right) + O(h^2), \tag{4}$$

where $k_t = \alpha_t h/\left(3e^{-h}\alpha_t h - 2\sigma_{\lambda_t}e^{\lambda_o}(e^h-1)\right)$. The $O(h^2)$ residual makes the local nature of this relation explicit; the proposition does not assert a globally linear feature trajectory.

3.2 HoltSeer

HoltSeer models local residual-feature evolution as a time series using Holt's two-parameter exponential smoothing model (Holt, 2004). The predictor represents each local sequence by separate level and trend states and scales the trend when forecasting the next feature. Assumption 3.1 and Proposition 3.2 motivate this local approximation. We first define the predictor and then specify its execution between full network evaluations.

Level–Trend Predictor. Given a local input window of K cached residual features ($K \geq 2$), which may contain computed and previously predicted values, $\{\boldsymbol{y}_T, \cdots, \boldsymbol{y}^{\text{pred}}_{T-K+1}\}$, we compute the level, trend, and next-feature prediction as follows:

$$\boldsymbol{l}_t = \alpha\boldsymbol{y}^{\text{pred}}_t + (1-\alpha)(\boldsymbol{l}_{t+1} + \boldsymbol{b}_{t+1}), \tag{5}$$

$$\boldsymbol{b}_t = \beta(\boldsymbol{l}_t - \boldsymbol{l}_{t+1}) + (1-\beta)\boldsymbol{b}_{t+1}, \tag{6}$$

$$\boldsymbol{y}^{\text{pred}}_{t-1} = \boldsymbol{l}_t + \gamma\cdot\boldsymbol{b}_t. \tag{7}$$

Here $\alpha, \beta, \gamma \in (0,1)$ and $t = T, \ldots, T-K+2$. The level update in Equation (5) exponentially smooths the cached features, with larger α assigning more weight to the newest feature. The trend update in Equation (6) smooths changes between consecutive level estimates. The coefficient γ scales the trend contribution in Equation (17), controlling how aggressively the local trend is extrapolated. Together, these updates define the level–trend forecasting formulation.

Execution Protocol: Prediction Feedback and Refresh. We apply the predictor through a rolling window of size K containing the most recent residual features, including both computed and predicted values. At each skipped timestep, HoltSeer forecasts the next residual feature and inserts it into the window for the subsequent state update. For example, the window $\{\boldsymbol{y}_T, \boldsymbol{y}^{\text{pred}}_{T-1}, \boldsymbol{y}^{\text{pred}}_{T-2}\}$ produces $\boldsymbol{y}^{\text{pred}}_{T-3}$; the next window is $\{\boldsymbol{y}^{\text{pred}}_{T-1}, \boldsymbol{y}^{\text{pred}}_{T-2}, \boldsymbol{y}^{\text{pred}}_{T-3}\}$. Within a cache interval of length N, at most $N-1$ consecutive forecasts precede a full network evaluation that supplies a new computed feature.

The protocol gives the predictor a bounded operating context. Within each cache interval, the state receives a sequence of local estimates and is then corrected by a newly computed feature before the next interval begins. Thus, a predicted feature contributes to the next local update through the smoothed state, while the periodic network evaluation prevents the forecasting chain from extending over the entire denoising trajectory. This distinction is important for interpreting the method: the equations define the temporal forecasting mechanism, and the refresh schedule limits how long that mechanism operates without a new network observation.

Baseline and Diagnostic Protocols. In the main experiments, TaylorSeer denotes its original non-rolling procedure, which forecasts from computed anchors. Appendix A.1 separately examines a diagnostic rolling-adapted TaylorSeer variant that feeds predictions back as trajectory anchors.

A simple example for HoltSeer Predictor. The computation follows a recurrence process that requires initializing the starting point, with time from larger to smaller. Let $K = 2$, and we have two cached features $\left\{\boldsymbol{y}_T, \boldsymbol{y}_{T-1}^{\text{pred}}\right\}$. The aim is to predict the cached feature at $t = T - 2$, as shown in Figure 3. Next, with the initial value of $\boldsymbol{l}_T = \boldsymbol{y}_T$ and $\boldsymbol{b}_T = \boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T$, we have:

$$\boldsymbol{l}_{T-1} = \alpha \boldsymbol{y}_{T-1}^{\text{pred}} + (1-\alpha)(\boldsymbol{y}_T + \boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T) \quad = \boldsymbol{y}_{T-1}^{\text{pred}}, \tag{8}$$

$$\boldsymbol{b}_{T-1} = \beta(\boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T) + (1-\beta)(\boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T) = \boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T, \tag{9}$$

$$\boldsymbol{y}_{T-2}^{\text{pred}} = \boldsymbol{y}_{T-1}^{\text{pred}} + \gamma \cdot (\boldsymbol{y}_{T-1}^{\text{pred}} - \boldsymbol{y}_T). \tag{10}$$

Subsequently, $\{\boldsymbol{y}_{T-1}^{\text{pred}}, \boldsymbol{y}_{T-2}^{\text{pred}}\}$ is used as the input for the next bounded prediction. Under this $K = 2$ initialization, α and β cancel algebraically, yielding a local affine extrapolation controlled by γ.

Error Analysis. Under rolling prediction, the prediction error of HoltSeer at timestep $t - m$ can be written as

$$\boldsymbol{e}_{t-m} = c^m \boldsymbol{e}_t + \sum_{i=1}^{m} c^{m-i} \boldsymbol{\rho}_{t-i}, \quad m \geq 1, \tag{11}$$

where $c = \alpha(1+\gamma\beta)$, and $\boldsymbol{\rho}_{t-i}$ depends on computed features, predicted features, and the level–trend state. Consider a segment of $M \leq N - 1$ consecutive forecasts between full network evaluations. If $0 < c < 1$ and, for a fixed vector norm, $\|\boldsymbol{\rho}_{t-i}\| \leq R_N$ for all $1 \leq i \leq M$, then

$$\|\boldsymbol{e}_{t-m}\| \leq c^m \|\boldsymbol{e}_t\| + R_N \frac{1-c^m}{1-c}, \qquad 1 \leq m \leq M. \tag{12}$$

Under this assumption, the accumulated residual contribution is at most $R_N(1-c^m)/(1-c)$, while the explicit inherited-error term has weight c^m. The residual bound R_N is an assumption on the state-dependent residuals within this segment and may vary with the cache interval and configuration. Appendix A.2 derives the decomposition and proves this conditional bound.

4 Experiments

4.1 Experiment Settings

In the experiment settings, we referred to acceleration methods such as TaylorSeer, HiCache, and FreqCa. We follow standard evaluation protocols for all models; detailed experimental settings are provided in Appendix E.

4.2 Quantitative and Qualitative Study

To fully evaluate the performance of the proposed method, we consider multiple acceleration settings ($N = 5, 6, 8, 10$) and conduct a comprehensive comparison with existing cache-based approaches.

Table 1, 2, and 3 depict the quantitative results of FLUX.1-dev and Qwen-Image on DrawBench, Wan2.1 on VBench2. A downward arrow indicates lower metric values are better, while an upward arrow indicates higher metric values are better. Best results are highlighted in bold, and second-best are underlined. Note that "–" indicates the code of these methods are not released and lack data.

The quantitative results evaluate HoltSeer as a complete time-series feature forecasting method. Across FLUX.1-dev, Qwen-Image, and Wan2.1, the results at large cache intervals show improved reference-based fidelity, measured by SSIM, PSNR, and LPIPS, while retaining competitive latency.

The fidelity advantage is particularly apparent at large evaluated cache intervals, where computed features are sparse. In these settings, the original TaylorSeer baseline exhibits greater degradation in SSIM and LPIPS than HoltSeer. This comparison supports the effectiveness of HoltSeer under sparse computation.

For example, on Qwen-Image at $N = 10$, HoltSeer and TaylorSeer both require 925.77 T FLOPs and have a reported latency of 4.63 s (Table 2). At this matched computational cost, HoltSeer improves

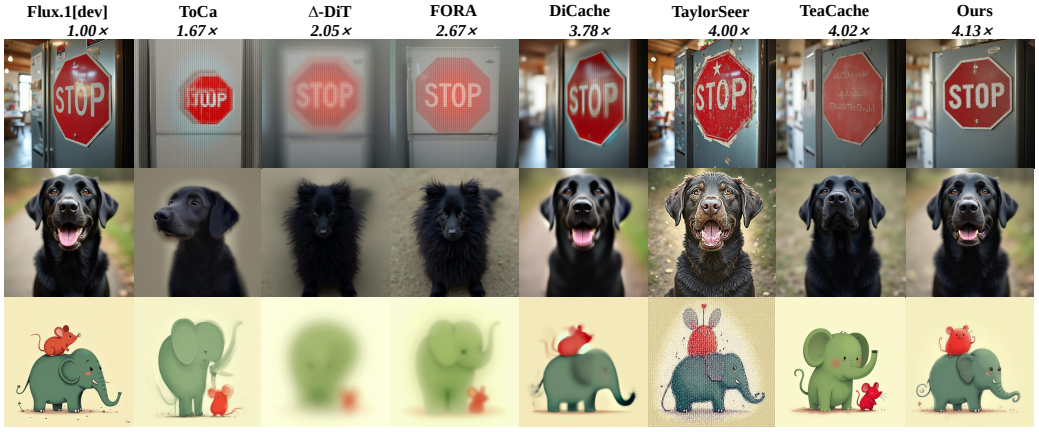

Figure 4: Qualitative comparison in text-to-image generation for FLUX.1-dev. Each column corresponds to a distinct method with an acceleration ratio.

SSIM from 0.4635 to 0.7242 and PSNR by 1.52 dB, while reducing LPIPS from 0.6128 to 0.4191, a relative reduction of 31.6%. These gains demonstrate improved reference-based fidelity at the same reported compute and latency.

HoltSeer also remains competitive on CLIP Score and Image Reward. At smaller cache intervals, the reference-based metrics show trade-offs: on Qwen-Image at $N = 5$, TaylorSeer achieves higher SSIM and lower LPIPS, whereas HoltSeer achieves higher PSNR. HoltSeer's fidelity advantage is most apparent under aggressive timestep skipping.

Table 1: **Quantitative comparison in text-to-image generation** for FLUX.1-dev on DrawBench. †: "cache-then-forecast".

Method	Acceleration				Visual Quality				
	Latency(s)↓	Speed↑	FLOPs(T)↓	Speed↑	CLIP↑	Image Reward↑	SSIM↑	PSNR↑	LPIPS↓
Flux.1[dev], 50 steps	30.46	1.00×	3734.56	1.00×	31.2627	0.9940	1.0000	∞	0.0000
40% steps	12.21	2.49×	1503.02	2.48×	31.4590	0.9967	0.7006	29.1031	0.3485
ToCa ($N = 10$)	18.26	1.67×	**714.66**	**5.23**×	**31.9004**	0.6986	0.4558	28.0943	0.7160
Δ-DiT ($N = 3$)	14.86	2.05×	1349.82	2.77×	31.0951	0.5434	0.4549	28.0532	0.6725
FORA ($N = 3$)	11.40	2.67×	1012.59	3.69×	31.4370	0.7103	0.4412	28.0688	0.6729
DiCache† ($l = 0.7$)	10.08	3.02×	1127.77	3.31×	31.3857	0.6443	0.7104	**30.4077**	0.3981
TaylorSeer† ($N = 5, O = 2$)	10.12	3.01×	908.81	4.11×	31.2967	1.0016	0.7029	29.1885	0.3291
SeaCache ($\tau = 0.45$)	12.11	2.52×	1201.18	3.11×	31.5388	1.0120	**0.7394**	28.8769	0.3616
Ours†($N = 5, K = 3$)	**9.86**	**3.09**×	958.15	3.90×	31.5444	**1.0142**	0.7204	29.4514	**0.3220**
TeaCache ($l = 0.75$)	8.78	3.47×	982.94	3.80×	31.3304	1.0097	0.6907	29.1491	0.3653
TC-Padé† (Medium)	8.74	3.49×	**760.08**	**4.91**×	31.2708	0.9402	0.6908	29.3693	0.3909
DBCache (Medium)	8.74	3.49×	812.02	4.60×	31.7855	**1.0341**	0.6870	29.1724	0.3568
AB-Cache† ($N = 6, O = 2$)	**8.72**	**3.49**×	812.02	4.60×	**31.7967**	0.9748	0.6457	28.9627	0.4307
TaylorSeer† ($N = 6, O = 2$)	9.29	3.28×	**760.08**	**4.91**×	31.4531	1.0131	0.6521	28.8128	0.3882
Ours † ($N = 6, K = 3$)	**8.72**	**3.49**×	812.02	4.60×	31.4734	1.0189	**0.7081**	**29.3786**	**0.3362**
TeaCache ($l = 0.8$)	8.15	3.74×	908.57	4.11×	31.2885	1.0284	0.6797	29.0206	0.3828
TC-Padé† (Fast)	7.92	3.84×	685.46	5.45×	31.1870	0.8943	0.6707	29.2031	0.4182
DBCache (Fast)	**7.71**	**3.95**×	665.89	5.61×	31.6514	0.9646	0.6816	**29.2549**	0.3852
HiCache† ($N = 7, O = 2$)	8.79	3.47×	670.44	5.57 ×	**32.4500**	0.9979	0.6572	28.9370	0.3982
TaylorSeer † ($N = 8, O = 2$)	8.35	3.65×	**611.36**	**6.11**×	31.4602	0.9754	0.5868	28.4292	0.4788
SeaCache ($\tau = 0.56$)	8.98	3.39×	774.80	4.82×	31.2797	**1.0440**	0.6690	29.1685	0.4117
Ours † ($N = 8, K = 3$)	7.80	3.91×	665.89	5.61×	31.5622	0.9921	**0.6822**	29.2267	**0.3696**
TeaCache ($l = 0.95$)	7.57	4.02×	834.20	4.48×	31.3291	**0.9999**	0.6541	28.8061	0.4255
TC-Padé† (Ultra)	7.45	4.08×	611.09	6.11×	31.2176	0.8547	0.6518	29.1051	0.4513
HiCache † ($N = 9, O = 2$)	8.27	3.68×	596.07	6.27×	31.9600	0.9113	0.6443	28.6470	0.4374
FreqCa † ($N = 10$)	–	–	597.47	6.25×	**32.5300**	0.9700	0.6200	28.7700	0.4300
TaylorSeer † ($N = 10, O = 2$)	7.61	4.00×	**537.00**	**6.95**×	31.1420	0.8508	0.5100	28.2043	0.5811
Ours †($N = 10, K = 3$)	**7.37**	**4.13**×	592.83	6.30×	31.4838	0.9739	**0.6731**	**29.1191**	**0.3922**

Table 2: **Quantitative comparison in text-to-image generation** for Qwen-Image on DrawBench. †: "cache-then-forecast".

Method	Acceleration				Visual Quality				
	Latency(s)↓	Speed↑	FLOPs(T)↓	Speed↑	CLIP↑	Image Reward↑	SSIM↑	PSNR↑	LPIPS↓
Qwen-Image, 50steps	26.06	1.00×	6482.07	1.00×	31.2359	0.8616	1.0000	∞	0.0000
40% steps	10.89	2.39×	2605.40	2.49×	30.8574	0.6640	**0.8429**	30.2915	0.2156
DBCache (Fast)	7.44	3.50×	1652.11	3.92×	30.2694	0.0813	0.7036	28.1459	0.5335
TaylorSeer† ($N = 5, O = 2$)	7.13	3.66×	1571.85	4.12×	30.8254	0.7032	0.8361	30.2798	**0.2026**
Ours† ($N = 5, K = 3$)	7.13	3.66×	1571.85	4.12×	**31.1605**	**0.7037**	0.8346	**31.0658**	0.2356
TaylorSeer ($N = 6, O = 2$)	6.11	4.27×	1313.42	4.94×	30.5860	**0.6484**	0.7583	29.5023	**0.2652**
Ours ($N = 6, K = 3$)	6.11	4.27×	1313.42	4.94×	**30.9941**	0.5795	**0.8144**	**30.5316**	0.2753
TaylorSeer ($N = 8, O = 2$)	5.12	5.09×	1054.99	6.14×	30.1475	**0.4538**	0.7001	28.7401	0.3633
Ours ($N = 8, K = 3$)	5.12	5.09×	1054.99	6.14×	**30.8247**	0.4435	**0.7579**	**29.9845**	**0.3600**
TaylorSeer ($N = 10, O = 2$)	4.63	5.63×	925.77	7.00×	27.9766	-0.3527	0.4635	28.0842	0.6128
Ours ($N = 10, K = 3$)	4.63	5.63×	925.77	7.00×	**30.8555**	**0.3550**	**0.7242**	**29.6076**	**0.4191**

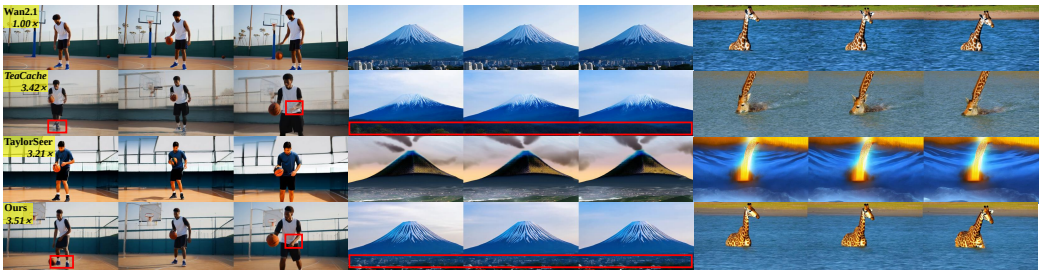

Figure 5: Qualitative comparison in text-to-video generation for Wan2.1. Each row corresponds to a method with its acceleration ratio.

Figures 4 and 5 show representative qualitative comparisons across different methods, with additional batch examples provided in Figures 7, 8 and 9 of Appendix F. Consistent with the quantitative results, HoltSeer better preserves visual fidelity under high acceleration ratios, producing outputs that remain closer to the original full-step generation in both structure and fine details. In contrast, existing methods exhibit more noticeable degradation. TaylorSeer often introduces color distortion and noise artifacts, DiCache tends to preserve the overall composition but produces blurred details, and TeaCache may cause semantic deviations from the original generation. These visual results further support the robustness of HoltSeer under aggressive timestep skipping.

Table 3: **Quantitative comparison in text-to-video generation** for Wan2.1 on VBench2. †: "cache-then-forecast".

Method	Acceleration				Visual Quality		
	Latency(s)↓	Speed↑	FLOPs(T)↓	Speed↑	SSIM↑	PSNR↑	LPIPS↓
Wan2.1, 50steps	142.32	1.00×	8494.51	1.00×	1.0000	∞	0.0000
TeaCache ($l = 0.35$)	41.59	3.42×	1928.68	4.40×	0.5941	28.3448	0.4672
TC-Padé† (Medium)	**40.54**	**3.51**×	1928.68	4.40×	0.5968	28.6603	0.4895
TaylorSeer † ($N = 6, O = 1$)	43.58	3.27×	1928.68	4.40×	0.4847	28.2236	0.5981
TaylorSeer ($N = 6, O = 2$)	44.32	3.21×	1928.68	4.40×	0.5032	28.2797	0.5710
Ours † ($N = 6, K = 2$)	**40.54**	**3.51**×	1928.68	4.40×	0.6134	**28.7041**	0.4092
Ours ($N = 6, K = 3$)	**40.54**	**3.51**×	1928.68	4.40×	**0.6173**	28.6939	**0.4054**
TeaCache ($l = 0.6$)	33.51	4.25×	1436.24	5.91×	0.5682	28.1539	0.5395
TC-Padé† (Fast)	**33.37**	**4.26**×	1436.24	5.91×	0.5664	28.4528	0.5627
TaylorSeer ($N = 10, O = 1$)	35.23	4.04×	1436.24	5.91×	0.4256	28.1220	0.6878
TaylorSeer ($N = 10, O = 2$)	36.11	3.94×	1436.24	5.91×	0.4531	28.1735	0.6546
Ours ($N = 10, K = 2$)	**33.37**	**4.26**×	1436.24	5.91×	0.5855	28.4712	0.4623
Ours ($N = 10, K = 3$)	**33.37**	**4.26**×	1436.24	5.91×	**0.5923**	**28.4887**	**0.4513**

4.3 Ablation Studies

We examine how the execution protocol and parameter choices affect the proposed time-series predictor. On FLUX.1-dev at $N = 10, K = 3$, HoltSeer improves SSIM, PSNR, and LPIPS relative to its tested non-rolling variant, which forecasts from computed features only (Table 5).

Level and trend smoothing. Table 4 evaluates five (α, β) pairs on FLUX.1-dev with $N = 6, K = 3, \gamma = 0.3$ fixed. These parameters control how strongly recent features update the level and how strongly recent level changes update the trend. Across the tested pairs, SSIM ranges from 0.7037 to 0.7081, PSNR from 29.3104 to 29.3786, and LPIPS from 0.3362 to 0.3437, indicating limited variation in reference-based fidelity within this setting. The pair $(0.8, 0.3)$ gives the best values on these three metrics among the tested combinations. Our main cross-model comparisons use the same $\alpha = 0.8, \beta = 0.3, \gamma = 0.3$, as detailed in Appendix E.

Table 4: Sensitivity to level and trend smoothing parameters on FLUX.1-dev, with $N = 6, K = 3, \gamma = 0.3$ fixed.

Method (Ours)	CLIP↑	Image Reward↑	SSIM↑	PSNR↑	LPIPS↓
$\alpha = 0.3, \beta = 0.3$	31.4649	1.0083	0.7055	29.3454	0.3405
$\alpha = 0.3, \beta = 0.8$	31.4816	1.0097	0.7037	29.3104	0.3437
$\alpha = 0.5, \beta = 0.5$	31.5056	1.0008	0.7039	29.3138	0.3434
$\alpha = 0.8, \beta = 0.3$	31.4734	1.0189	0.7081	29.3786	0.3362
$\alpha = 0.8, \beta = 0.8$	31.4508	1.0044	0.7069	29.3662	0.3383

Trend scaling and window size. The coefficient γ controls the strength of extrapolation from the estimated level. At $N = 6, K = 2$, increasing γ from 0.3 to 0.5 reduces SSIM from 0.7123 to 0.6493 and increases LPIPS from 0.3282 to 0.4378, showing that stronger trend extrapolation can reduce fidelity. Window size produces a different trade-off: $K = 2$ achieves higher SSIM and lower LPIPS than $K = 3, 4$, while $K = 4$ gives the highest PSNR in this comparison. Appendix C reports the full K, γ scans in Table 6.

The main quantitative comparisons report the latency–fidelity trade-off across cache intervals N.

4.4 Deployment Feasibility

We evaluate HoltSeer's deployment practicality in terms of compatibility with SmoothQuant and memory overhead. On FLUX.1-dev, HoltSeer reduces latency from 30.46 s to 7.37 s, and applying SmoothQuant on top reduces it further to 4.01 s (Table 8). The reported FID, CLIP, and aesthetic scores change only modestly across these configurations, indicating that the two acceleration mechanisms can be combined without replacing the quality evaluation used for HoltSeer.

Maintaining the forecasting state adds 0.14 GB on FLUX.1-dev (36.32 to 36.46 GB) and 0.17 GB on Qwen-Image (65.24 to 65.41 GB) relative to the baseline (Table 7). Thus, the temporal state introduces limited memory overhead while remaining compatible with standard deployment optimizations.

5 Conclusion

We propose HoltSeer, a training-free cache-then-forecast method that models local diffusion feature evolution through a smoothed level–trend state. Inspired by Holt's exponential smoothing, the predictor summarizes recent cached features, tracks local directional changes, and scales trend extrapolation. A bounded rolling protocol and periodic full network evaluations implement this forecasting formulation across skipped timesteps.

Experiments on image and video generation models show that HoltSeer improves reference-based fidelity at large evaluated cache intervals while maintaining competitive latency. These results support level–trend forecasting as a practical design for high-ratio diffusion acceleration.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation and the preparation of mathematical proofs. We did not use generative AI tools to implement the proposed method or to propose or refine the research hypotheses; these tasks were completed by the authors. Additionally, we used generative AI tools for language polishing and the creation of figures. The authors reviewed all AI-assisted content, checked the translations and language edits for accuracy, verified the logical steps and validity of the mathematical proofs, and inspected the figures for consistency with the methods and results presented in the paper. We take full responsibility for the final content of this work, including text, claims, proofs, and figures produced with the assistance of generative AI.

REFERENCES

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

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

Benlei Cui, Shaoxuan He, Bukun Huang, Zhizeng Ye, Yunyun Sun, Longtao Huang, Hui Xue, Yang Yang, Jingqun Tang, Zhou Zhao, et al. Tc-pad\'e: Trajectory-consistent pad\'e approximation for diffusion acceleration. arXiv preprint arXiv:2603.02943, 2026.

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

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, pages 7514–7528, 2021.

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

Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. International journal of forecasting, 20(1):5–10, 2004.

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

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

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

Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. Advances in Neural Information Processing Systems, 36:20662–20678, 2023c.

Dong Liu, Yanxuan Yu, Jiayi Zhang, Yifan Li, Ben Lengerich, and Ying Nian Wu. Fastcache: Fast caching for diffusion transformer through learnable linear approximation. arXiv preprint arXiv:2505.20353, 2025a.

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

Jiacheng Liu, Peiliang Cai, Qinming Zhou, Yuqi Lin, Deyang Kong, Benhao Huang, Yupei Pan, Haowen Xu, Chang Zou, Junshu Tang, et al. Freqca: Accelerating diffusion models via frequency-aware caching. arXiv preprint arXiv:2510.08669, 2025c.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. arXiv preprint arXiv:2503.06923, 2025d.

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

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, pages 15762–15772, 2024.

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

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

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. arXiv preprint arXiv:2407.01425, 2024.

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, pages 1972–1981, 2023.

Wan Team. Wan: Open and advanced large-scale video generative models. 2025.

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

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025. URL https://arxiv.org/abs/2508.02324.

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.

Zichao Yu, Zhen Zou, Guojiang Shao, Chenwei Zhang, Shengze Xu, Jie Huang, Feng Zhao, Xiaodong Cun, and Wenyi Zhang. Ab-cache: Training-free acceleration of diffusion models via adams-bashforth cached feature reuse. In Proceedings of the 33rd ACM International Conference on Multimedia, pages 10408–10417, 2025.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 586–595, 2018.

Dian Zheng, Ziqi Huang, Hongbo Liu, Kai Zou, Yinan He, Fan Zhang, Yuanhan Zhang, Jingwen He, Wei-Shi Zheng, Yu Qiao, and Ziwei Liu. VBench-2.0: Advancing video generation benchmark suite for intrinsic faithfulness. arXiv preprint arXiv:2503.21755, 2025.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. arXiv preprint arXiv:2410.05317, 2024.

A Error Propagation in the Forecasting Protocol

A.1 A Diagnostic Rolling-Adapted Trajectory Predictor

We examine error propagation when predicted features are reused as inputs. We first analyze an illustrative trajectory recurrence, then derive the error decomposition of the level–trend predictor.

To illustrate the feedback mechanism considered in the rolling-adapted TaylorSeer diagnostic Liu et al. (2025d), we use the following two-point linear extrapolation recurrence. This simplified recurrence is not the original TaylorSeer algorithm or a full second-order Taylor expansion. With diffusion timesteps indexed from large to small, we have:

$$\boldsymbol{y}_{t-1}^{\text{pred}} = \boldsymbol{y}_t^{\text{pred}} + \left(\boldsymbol{y}_t^{\text{pred}} - \boldsymbol{y}_{t+1}^{\text{pred}}\right) = 2\boldsymbol{y}_t^{\text{pred}} - \boldsymbol{y}_{t+1}^{\text{pred}}. \tag{13}$$

The prediction error is:

$$\begin{aligned}
\boldsymbol{e}_{t-1} &= \boldsymbol{y}_{t-1}^{\text{pred}} - \boldsymbol{y}_{t-1} \\
&= \left(2\boldsymbol{y}_t^{\text{pred}} - \boldsymbol{y}_{t+1}^{\text{pred}}\right) - \boldsymbol{y}_{t-1} \\
&= 2(\boldsymbol{y}_t^{\text{pred}} - \boldsymbol{y}_t) - (\boldsymbol{y}_{t+1}^{\text{pred}} - \boldsymbol{y}_{t+1}) + \left(2\boldsymbol{y}_t - \boldsymbol{y}_{t+1} - \boldsymbol{y}_{t-1}\right) \\
&= 2\boldsymbol{e}_t - \boldsymbol{e}_{t+1} + \boldsymbol{\delta}_{t-1}.
\end{aligned} \tag{14}$$

Unrolling this recurrence gives the prediction error at timestep $t - m$:

$$\boldsymbol{e}_{t-m} = \boldsymbol{e}_t + m(\boldsymbol{e}_t - \boldsymbol{e}_{t+1}) + \sum_{i=1}^{m}(m - i + 1)\boldsymbol{\delta}_{t-i}. \tag{15}$$

Equation (15) separates the inherited error $\boldsymbol{e}_t$, a term with coefficient m multiplying the adjacent error difference, and a weighted sum of local residuals. The coefficient on the adjacent error difference grows linearly with the prediction horizon, but the total error need not grow linearly or monotonically because the residual terms and cancellations also matter. This statement concerns the specified diagnostic recurrence only.

Figure 6 shows degraded outputs from the diagnostic rolling-adapted TaylorSeer variant. The following analysis examines how predicted features enter HoltSeer's level–trend update and contribute to its prediction error.

A.2 Error Decomposition of the Level–Trend Predictor

Proposition A.1. *For fixed parameters satisfying $0 < c = \alpha(1 + \gamma\beta) < 1$, the coefficient on the explicit inherited-error term in the following decomposition decays geometrically. The prediction error at $t - m$ can be written as:*

$$\boldsymbol{e}_{t-m} = c^m \boldsymbol{e}_t + \sum_{i=1}^{m} c^{m-i}\boldsymbol{\rho}_{t-i}, \quad m \geq 1 \tag{16}$$

where $\boldsymbol{\rho}_{t-i}$ depends on computed features, predicted features, and the level–trend state, and can therefore contain effects of earlier prediction errors.

Proof. According to the HoltSeer Predictor, we obtain:

$$\begin{aligned}
\boldsymbol{y}_{t-1}^{\text{pred}} &= \boldsymbol{l}_t + \gamma \cdot \boldsymbol{b}_t, && (17)\\
&= \boldsymbol{l}_t + \gamma\left[\beta(\boldsymbol{l}_t - \boldsymbol{l}_{t+1}) + (1-\beta)\boldsymbol{b}_{t+1}\right], && (18)\\
&= (1 + \gamma\beta)\boldsymbol{l}_t - \gamma\beta\boldsymbol{l}_{t+1} + \gamma(1-\beta)\boldsymbol{b}_{t+1}, && (19)\\
& && (20)
\end{aligned}$$

Then, substitute $\boldsymbol{l}_t = \alpha \boldsymbol{y}_t^{\text{pred}} + (1-\alpha)(\boldsymbol{l}_{t+1} + \boldsymbol{b}_{t+1})$ into the above equation, we get:

$$\boldsymbol{y}_{t-1}^{\text{pred}} = c\boldsymbol{y}_t^{\text{pred}} + a\boldsymbol{l}_{t+1} + d\boldsymbol{b}_{t+1}, \tag{21}$$

where coefficients c, a, d satisfy:

$$c = \alpha(1+\gamma\beta), \tag{22}$$
$$a = 1-\alpha-\alpha\gamma\beta, \tag{23}$$
$$d = 1-\alpha+\gamma-\alpha\gamma\beta. \tag{24}$$

Next, the prediction error of HoltSeer at $t-1$ is:

$$\begin{aligned} \boldsymbol{e}_{t-1} &= \boldsymbol{y}_{t-1}^{\text{pred}} - \boldsymbol{y}_{t-1} && (25)\\ &= c\boldsymbol{y}_t^{\text{pred}} + a\boldsymbol{l}_{t+1} + d\boldsymbol{b}_{t+1} - \boldsymbol{y}_{t-1} && (26)\\ &= c(\boldsymbol{y}_t + \boldsymbol{e}_t) + a\boldsymbol{l}_{t+1} + d\boldsymbol{b}_{t+1} - \boldsymbol{y}_{t-1} && (27)\\ &= c\boldsymbol{e}_t + \boldsymbol{\rho}_{t-1} && (28) \end{aligned}$$

where $\boldsymbol{\rho}_{t-1} = c\boldsymbol{y}_t + a\boldsymbol{l}_{t+1} + d\boldsymbol{b}_{t+1} - \boldsymbol{y}_{t-1}$.

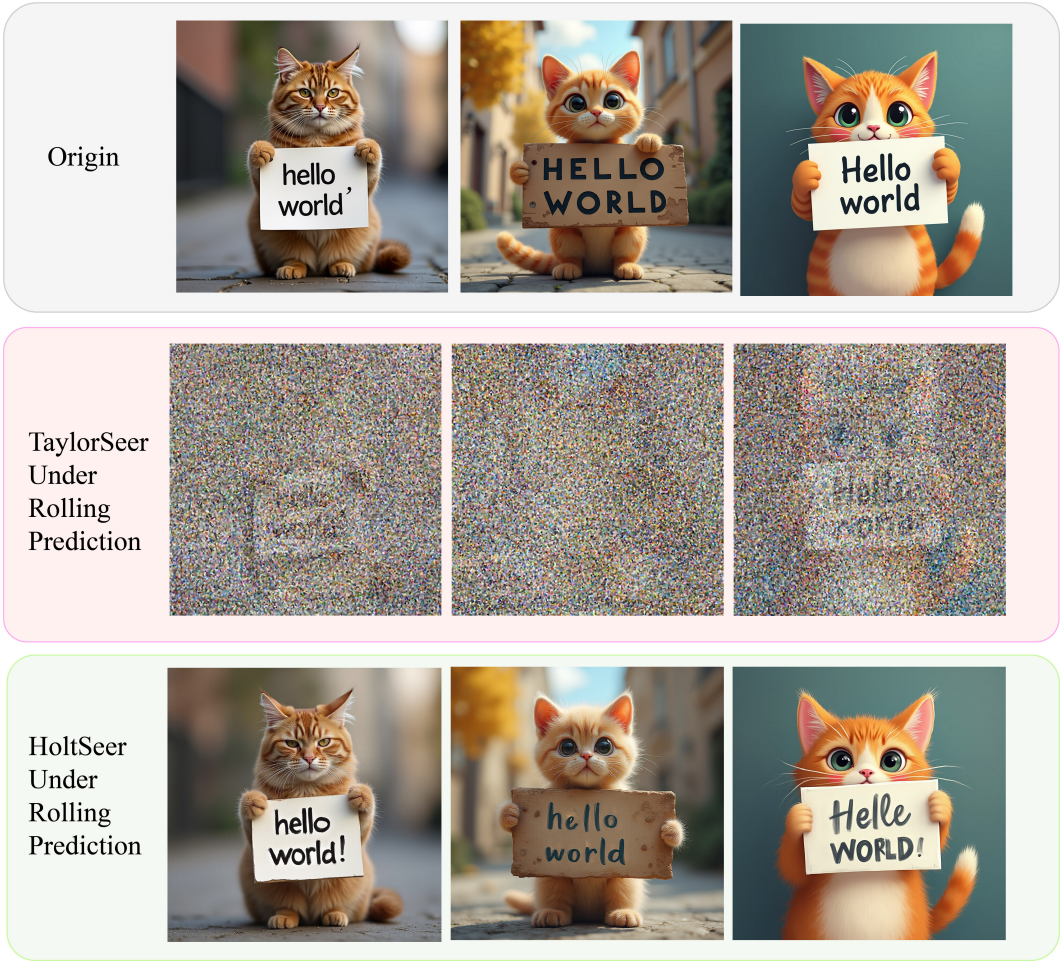

Figure 6: Qualitative results for the diagnostic rolling-adapted TaylorSeer variant, which reuses predicted features as trajectory anchors. The degraded outputs illustrate this adapted protocol, not the original TaylorSeer baseline used in the main comparisons.

Finally, by mathematical induction, the prediction error of HoltSeer at $t - m$ is:

$$\boldsymbol{e}_{t-m} = c^m \boldsymbol{e}_t + \sum_{i=1}^{m} c^{m-i} \boldsymbol{\rho}_{t-i}, \quad m \geq 1 \tag{29}$$

For the default parameters $\alpha = 0.8, \beta = 0.3, \gamma = 0.3$, the coefficient is $c = 0.872 < 1$.

□

Corollary A.2 (Conditional error bound within a cache interval)**.** *Consider a segment of $M \leq N - 1$ consecutive forecasts whose errors satisfy the decomposition in Equation* (11)*, with fixed $0 < c < 1$. For any fixed vector norm, suppose there is a finite $R_N \geq 0$ such that $\|\boldsymbol{\rho}_{t-i}\| \leq R_N$ for all $1 \leq i \leq M$. Then, for $1 \leq m \leq M$,*

$$\|\boldsymbol{e}_{t-m}\| \leq c^m \|\boldsymbol{e}_t\| + R_N \frac{1 - c^m}{1 - c}. \tag{30}$$

Proof. Applying the triangle inequality and the assumed residual bound to the error decomposition yields

$$\begin{aligned} \|\boldsymbol{e}_{t-m}\| &\leq c^m \|\boldsymbol{e}_t\| + \sum_{i=1}^{m} c^{m-i} \|\boldsymbol{\rho}_{t-i}\| \\ &\leq c^m \|\boldsymbol{e}_t\| + R_N \sum_{i=1}^{m} c^{m-i} \\ &= c^m \|\boldsymbol{e}_t\| + R_N \frac{1 - c^m}{1 - c}. \end{aligned}$$

The last equality follows from the finite geometric series. □

The corollary quantifies residual accumulation over a bounded forecast segment: its upper-bound contribution is at most $R_N/(1 - c)$. Here R_N bounds the full state-dependent residual, including effects of previous predictions; it is not implied by $c < 1$. The result is conditional on this residual bound and the stated recurrence, with no assumption that the same R_N applies across different cache intervals or arbitrarily long prediction horizons.

B Temporal Modeling in Representative Cache-then-Forecast Methods

Representative cache-then-forecast methods model temporal feature evolution. TaylorSeer Liu et al. (2025d) uses Taylor-series extrapolation from computed features and finite-difference derivatives. HiCache Feng et al. (2025) uses Hermite polynomial interpolation, and FreqCa Liu et al. (2025c) combines frequency decomposition with Hermite-style prediction of high-frequency components. DiCache Bu et al. (2025) uses adaptive linear prediction for denoising trajectory alignment. These approaches motivate examining how local dynamics are represented when computed anchors become sparse. HoltSeer adopts a Holt-inspired level–trend representation with smoothing and scaled extrapolation.

C Prediction-Protocol Ablation and Parameter Sensitivity

These FLUX.1-dev experiments characterize two aspects of the proposed time-series predictor: its predicted-feature feedback protocol and its sensitivity to parameter choices.

C.1 Prediction-Protocol Comparison within HoltSeer

Table 5 compares HoltSeer with the tested non-rolling variant that uses computed features only. At $N = 10, K = 3$, predicted-feature feedback increases SSIM from 0.6075 to 0.6731 and PSNR from 28.5040 to 29.1191, while reducing LPIPS from 0.4493 to 0.3922.

Table 5: Prediction-protocol comparison within HoltSeer on FLUX.1-dev. The reference configuration uses $N = 10, K = 3$; the non-rolling variant forecasts from computed features only.

Method	CLIP↑	Image Reward↑	SSIM↑	PSNR↑	LPIPS↓
HoltSeer ($N = 10, K = 3$)	31.4838	0.9739	0.6731	29.1191	0.3922
HoltSeer w/o predicted-feature feedback	31.3240	0.9202	0.6075	28.5040	0.4493

C.2 Parameter Sensitivity

The level and trend smoothing comparison is reported in Table 4 of Section 4.3. Here we provide the complete window-size and trend-scaling results.

Window size and trend scaling. Table 6 reports window-size and trend-scaling comparisons at $N = 6$. In the window comparison, $K = 2$ gives higher SSIM and lower LPIPS than $K = 3, 4$, whereas the other metrics favor different window sizes. Under the $K = 2$ initialization in Section 3.2, α and β cancel algebraically. In the trend-scaling comparison, $\gamma = 0.5$ yields lower SSIM and PSNR and higher LPIPS than $\gamma = 0.3$.

Table 6: Sensitivity to window size K and trend scaling γ on FLUX.1-dev at $N = 6$. The reference row uses $K = 2, \gamma = 0.3$. The K rows change window size at $\gamma = 0.3$; the γ rows change trend scaling at $K = 2$. Under the stated $K = 2$ initialization, α, β cancel algebraically.

Method (Ours)	CLIP↑	Image Reward↑	SSIM↑	PSNR↑	LPIPS↓
Reference: $K = 2, \gamma = 0.3$	31.4395	1.0096	0.7123	29.3943	0.3282
$K = 3$	31.4734	1.0189	0.7081	29.3786	0.3362
$K = 4$	31.5078	0.9882	0.7049	29.4476	0.3424
$\gamma = 0.1$	31.4624	0.9892	0.7048	29.3701	0.3449
$\gamma = 0.2$	31.4511	1.0054	0.7104	29.4111	0.3339
$\gamma = 0.4$	31.6165	1.0169	0.7076	29.2934	0.3301
$\gamma = 0.5$	31.7589	0.9967	0.6493	28.7427	0.4378

D Additional Deployment Feasibility

Table 7: GPU memory (GB) usage comparison across different methods.

Method	FLUX.1-dev	Qwen-Image
Baseline	36.32	65.24
TaylorSeer	42.80	76.53
HoltSeer	**36.46**	**65.41**

Table 8: Performance comparison among FLUX.1-dev, HoltSeer, and HoltSeer with quantization.

Method	Latency↓	FID↓	CLIP↑	Aes↑
FLUX.1-dev	30.46	23.38	31.26	6.25
+HoltSeer	7.37	24.02	31.48	6.12
+Quant	4.01	24.23	30.93	6.03

E Additional Experiment Settings

Model Configurations. We conduct experiments on three representative visual generation models and two benchmark datasets. For text-to-image generation, we employ the FLUX.1-dev Labs (2024) and Qwen-Image Wu et al. (2025) on DrawBench Saharia et al. (2022); for text-to-video generation, we employ Wan2.1-1.3BTeam (2025) on VBench2 Zheng et al. (2025). For all experiments, HoltSeer parameters maintain $\alpha = 0.8, \beta = 0.3, \gamma = 0.3$. Standard 50-steps inference is employed as the baseline while retaining all default sampling parameters to ensure strict experimental consistency. Inference and evaluation are performed on NVIDIA L40s GPUs for FLUX.1-dev Labs (2024), on NVIDIA RTX PRO 6000 GPUs for Qwen-Image Wu et al. (2025) and Wan2.1 Team (2025).

Evaluation Metrics. We evaluate the performance of different methods from two key perspectives: acceleration and visual quality. Latency in seconds and FLOPs are used to quantify acceleration. For visual quality, the assessment methods vary across different tasks. For text-to-image generation, we use the 200 prompts provided by DrawBench Saharia et al. (2022). And for text-to-video generation, we use the 1013 prompts from VBench2 Zheng et al. (2025). To accurately evaluate the quality of generated samples, we employ CLIP Score Hessel et al. (2021) and Image Reward Xu et al. (2023), which are based on text-image alignment and human preferences. For further assessment of the realism of generated samples, we also utilize SSIM Wang et al. (2004), PSNR, and LPIPS Zhang et al. (2018). These metrics reference the original image to measure similarity at the pixel level and perceptual consistency.

F More Qualitative Comparison

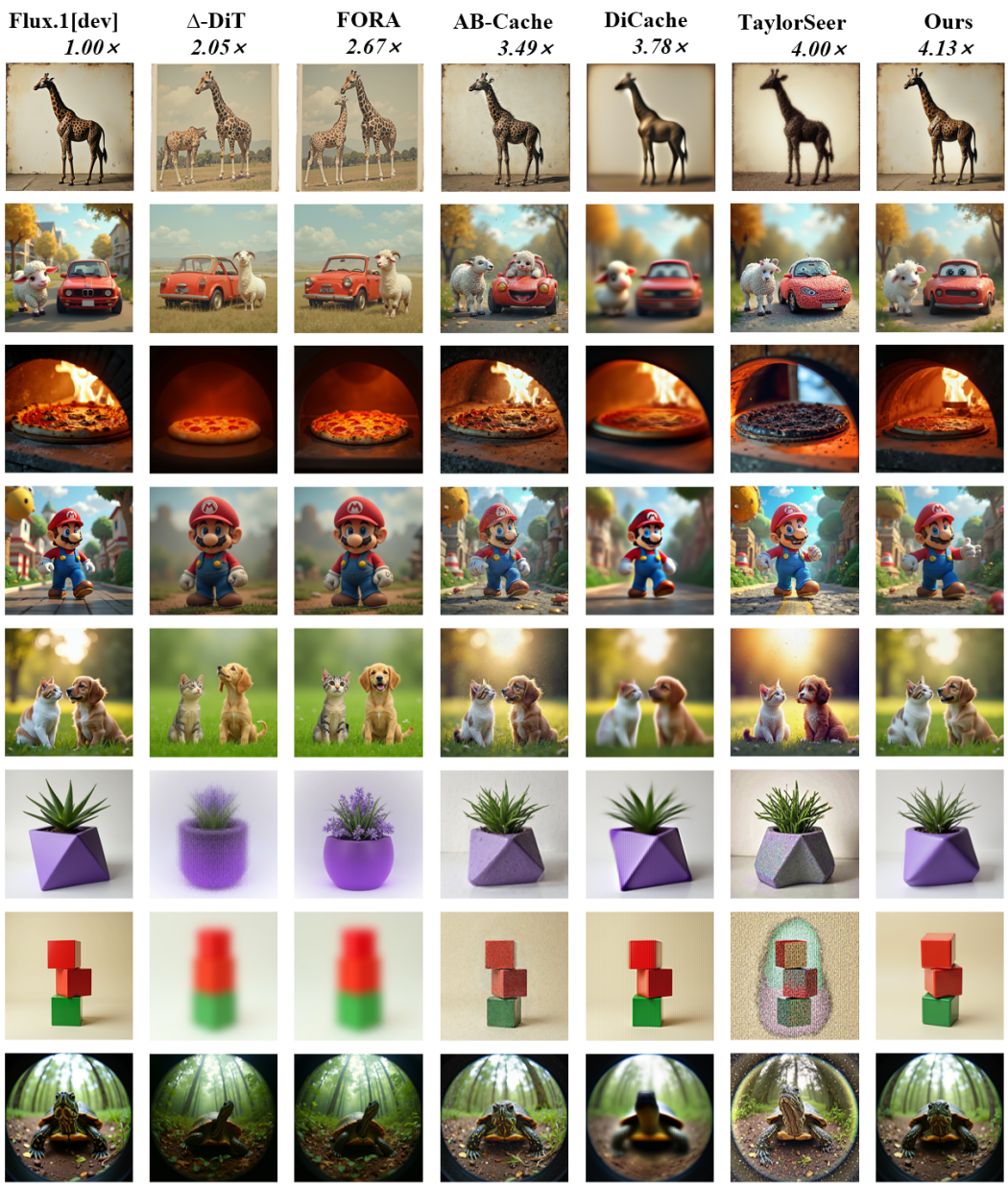

Figure 7: Qualitative comparison in text-to-image generation for FLUX.1-dev.



Figure 8: Qualitative comparison in text-to-image generation for Qwen-Image.

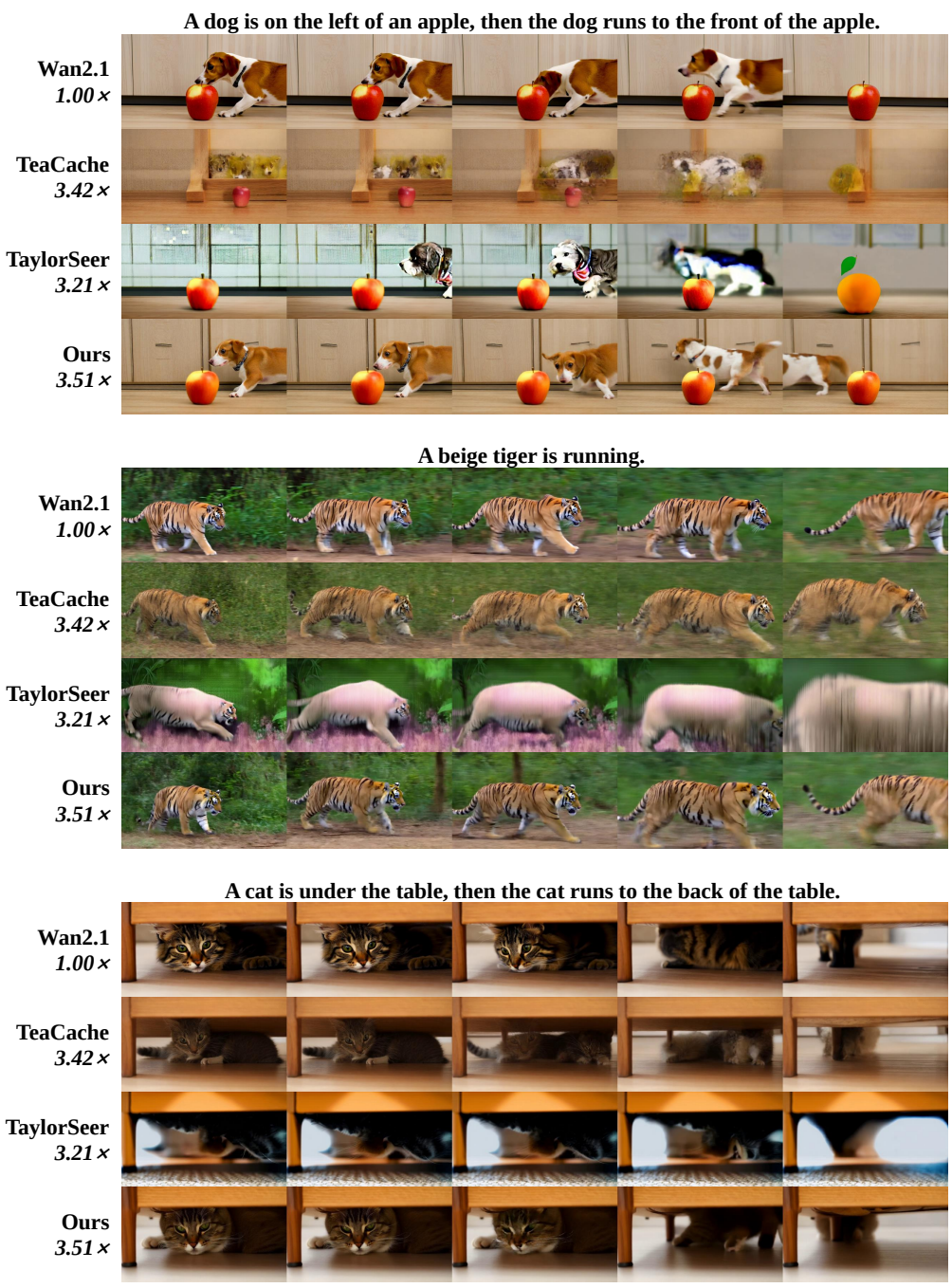

Figure 9: Qualitative comparison in text-to-video generation for Wan2.1 (1/2).

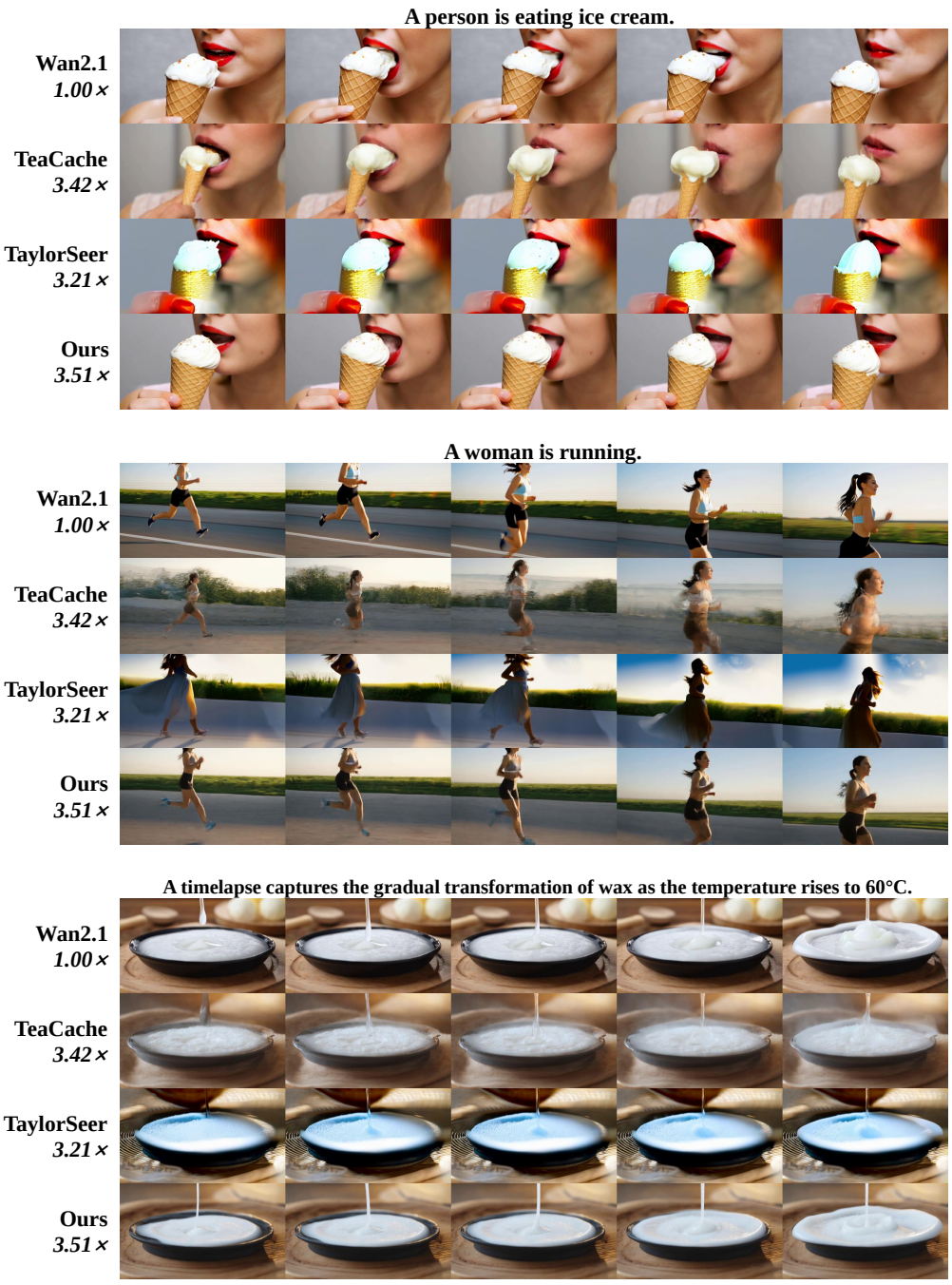

Figure 10: Qualitative comparison in text-to-video generation for Wan2.1 (2/2).

G LIMITATIONS

Although HoltSeer exhibits strong robustness in high acceleration ratio diffusion sampling and achieves favorable outcomes on reference image consistency metrics like SSIM, PSNR, and LPIPS, we still note a significant limitation: CLIP Score and Image Reward occasionally display opposite trends relative to indicators such as SSIM, PSNR, and LPIPS. Specifically, under certain acceleration settings, the generated results exhibit more noticeable blur, noise, or detail degradation visually compared to the original full-step sampling images, but the CLIP Score or Image Reward is higher. This indicates that no-reference metrics based on text-image alignment or preference models do not always accurately reflect the true visual quality of accelerated images relative to the original ones, especially because they may be biased when evaluating structural consistency, local details, and distortion degree. Additionally, HoltSeer's level–trend model is a local approximation, and its recursive execution can propagate prediction errors. The current ablations do not isolate the effect of smoothing from other design choices, and final performance is influenced by the hyperparameters α, β, γ. Therefore, designing more adaptive hyperparameter selection strategies is a crucial direction for further enhancing the stability and generalization capabilities of HoltSeer in the future.