



# No Cache Left Idle: Accelerating diffusion model via Extreme-slimming Caching

## Supplementary Material

### 6. Universal U-Curve Across Prompts

**Setup.** We analyze adjacent timesteps during denoising using prompts from the HPS benchmark across four styles, namely Animation, Concept-art, Painting, and Photo, randomly sampling **100 prompts per style**. For each pair of adjacent steps  $t-1$  and  $t$ , we compute two inline metrics on hidden states: the relative  $L_1$  change  $\|\Delta_t - \Delta_{t-1}\|_1 / \|\Delta_{t-1}\|_1$ , and the cosine similarity between  $\Delta_t$  and  $\Delta_{t-1}$ . These quantify the magnitude of reuse error and the directional consistency of features.

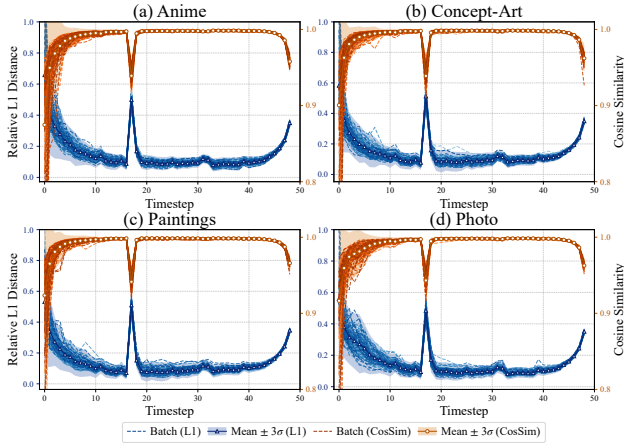


Figure 12. Universal U-Curve Across Prompts.

**Finding and implication.** Across styles and prompts, the curves exhibit a clear U-shape: early and late steps show larger relative change and lower cosine similarity, while mid-trajectory steps show smaller change and cosine similarity close to 1.0. The pattern is governed primarily by the model’s denoising dynamics, with weak prompt influence, suggesting a schedule that allocates more computation to the head and tail and less to the middle. We also examine representative dynamic timestep methods such as TeaCache and EasyCache and observe that strategies tend to converge to nearly the same schedule across prompts, differing mainly at the high-variation ends, yet they require nontrivial preprocessing.

**Our choice and result.** Guided by the prompt-robust U-curve, we adopt a *static* step-level schedule calibrated once on a small set, which is simple, training-free, and

aligned with the backbone’s intrinsic dynamics. In the experimental tables of Section 4, under matched speed our schedule attains higher perceptual quality than dynamic timestep policies, and under matched quality it achieves higher speed, confirming that our method is both practical and effective.

### 7. Position-Sensitive Reuse Error

Building on Supplementary Section 6, the effect of caching is governed by the model’s intrinsic denoising dynamics rather than prompt-specific effects. The *early* and *late* steps undergo large, model-driven changes and are sensitive to perturbations, while the *middle* steps evolve more smoothly and are comparatively stable. We verify this by injecting caching in three phases (early, middle, late) and inspecting the final images.

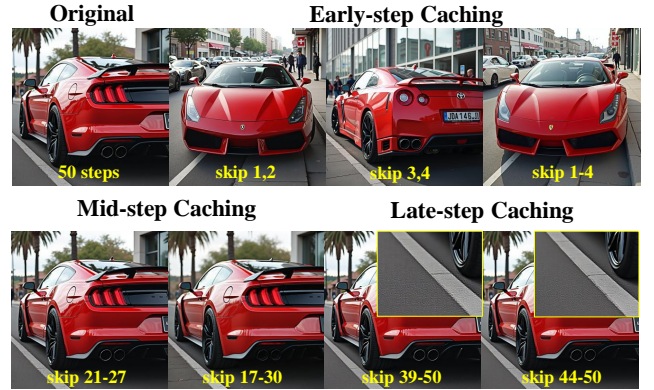


Figure 13. Position-Sensitive Ablations.

As shown in Fig. 13, caching in the early phase often distorts global structure and reduces quality because this stage establishes semantic layout and coarse geometry and small errors are amplified. Caching in the late phase preserves the layout but hurts fidelity: details blur and residual noise appears since this stage refines high-frequency content. By contrast, middle-phase caching is consistently robust: even with stronger reuse, both structure and perceptual quality remain close to a full run.

These observations yield a simple rule that follows the model’s behavior: use less caching at the fragile head and tail, and more caching in the stable middle. X-Slim encodes this with its push-then-polish controller. Thresholds limit caching near the boundaries by triggering refresh or

full steps when needed, while the middle region gains larger reuse to capture most of the speedup without sacrificing quality.

## 8. Detailed Experimental Settings

**Model Configurations.** We follow TaylorSeer’s setup and evaluate three diffusion backbones: FLUX.1-dev for text-to-image, HunyuanVideo for text-to-video, and DiT-XL/2 for class-conditional generation. For FLUX.1-dev, we use 200 DrawBench prompts at  $1024 \times 1024$  and sample 5 images per prompt (1,000 images total). For HunyuanVideo, we adopt the VBench suite with 946 prompts and generate five videos per prompt, totaling 4,730 clips. We use 540p with 81 frames for batch quality and latency evaluation, and 480p for visualization. For DiT-XL/2 on ImageNet, we uniformly cover the 1,000 classes and produce 50,000 images at  $256 \times 256$ .

**Evaluation and Metrics.** We assess efficiency by inference latency and quality by task-appropriate metrics. For text-to-image, we report ImageReward and CLIP for perceptual quality and text-image alignment, and also measure LPIPS, PSNR, and SSIM against full-compute references. For text-to-video, we follow VBench metrics. For class-conditional generation, we report FID-50k as the primary metric, complemented by sFID and Inception Score to capture fidelity and diversity.

**Hardware and Computational Resources.** All images and videos used for quality evaluation are generated in data parallel on 32 Ascend 910B devices. Latency is measured on a single A100 GPU to ensure a fair comparison.

## 9. Refresh settings and ratio selection.

**Setup.** We evaluate the impact of refresh ratios on FLUX.1-dev under two runtime modes named X-Slim-slow and X-Slim-fast. The dual-threshold ratio is set to  $r = \delta_{\text{warn}}/\delta_{\text{crit}} = 0.9$ . When  $r = 1$  the policy reduces to the step only variant X-Slim(S). Block refresh is controlled by  $\delta_{\text{blk}}$  which determines the effective refresh ratio. We ablate refresh ratios at 1.0, 0.8, 0.5, 0.2, and 0. Results are reported in Tables 5 and 6.

**Key findings.** Starting from our step-only variant X-Slim(S) at about  $3.01 \times$  acceleration, adding a 50% block refresh and an 80% token refresh raises the speed to about  $3.37 \times$  with negligible quality loss. Lowering the refresh ratios further causes noticeable degradation because correction becomes too weak and errors accumulate, which behaves close to no refresh. When targeting higher acceleration around  $4.69 \times$  a larger refresh ratio is needed to maintain quality because fewer timesteps leave less slack per

step. Using 80% for both block and token refresh is the most stable choice in this regime.

Table 5. Ablation A4 on X-Slim-slow refresh ratio effects.

Method	Latency(s)	Speed $\uparrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
L1.0+Tok1.0	9.076	3.01	21.7168	0.8162	0.1905
L0.8+Tok0.8	8.618	3.17	21.4582	0.8139	0.1950
L0.8+Tok0.5	8.179	3.34	21.3494	0.8102	0.2073
<b>L0.5+Tok0.8</b>	<b>8.107</b>	<b>3.37</b>	<b>21.2998</b>	<b>0.8097</b>	<b>0.2105</b>
L0.5+Tok0.5	7.610	3.59	20.3400	0.7849	0.2693
L0.5+Tok0.2	7.384	3.70	20.1050	0.7577	0.2801
L0.2+Tok0.5	7.324	3.73	19.9076	0.7568	0.2819
L0.2+Tok0.2	6.969	3.92	19.8064	0.7456	0.3101
L0.0+Tok0.0	6.413	4.26	19.7784	0.7255	0.3442

Table 6. Ablation A4 on X-Slim-fast refresh ratio effects.

Method	Latency(s)	Speed $\uparrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
L1.0+Tok1.0	5.825	4.69	19.5601	0.7520	0.2884
<b>L0.8+Tok0.8</b>	<b>5.497</b>	<b>4.97</b>	<b>19.2407</b>	<b>0.7321</b>	<b>0.2919</b>
L0.8+Tok0.5	5.244	5.21	18.7047	0.7137	0.3385
L0.5+Tok0.8	5.214	5.24	18.5174	0.7068	0.3402
L0.5+Tok0.5	4.923	5.55	18.3918	0.7042	0.3623
L0.0+Tok0.0	3.977	6.87	18.0763	0.6468	0.4732

**Parameter setting and tuning efficiency.** Directly mixing three levels requires per step decisions on level and refresh ratio which creates a large search space and heavy manual tuning. The push then polish paradigm narrows this to a few stable knobs that are introduced only on steps that require refresh. We tune a single threshold ratio  $r$  by setting it to the average relative change on the mid plateau of the reuse error curve. We set the block and token threshold to match a target refresh rate measured on a small calibration set. In practice the settings fall within a narrow band, which simplifies tuning and improves robustness across backbones.

## 10. Visualization on FLUX.1-dev

We present visual comparisons on FLUX.1-dev. In Figs. 14 and 15, X-Slim reaches  $4.97 \times$  acceleration while closely matching the full model, achieving the best balance of speed and fidelity.

## 11. Visualization on HunyuanVideo

As shown in Figs. 16 and 17, we provide visual comparisons on HunyuanVideo. At the highest acceleration, X-Slim produces videos that remain clear and closely match the full model, offering the best balance of speed and fidelity among the compared methods.



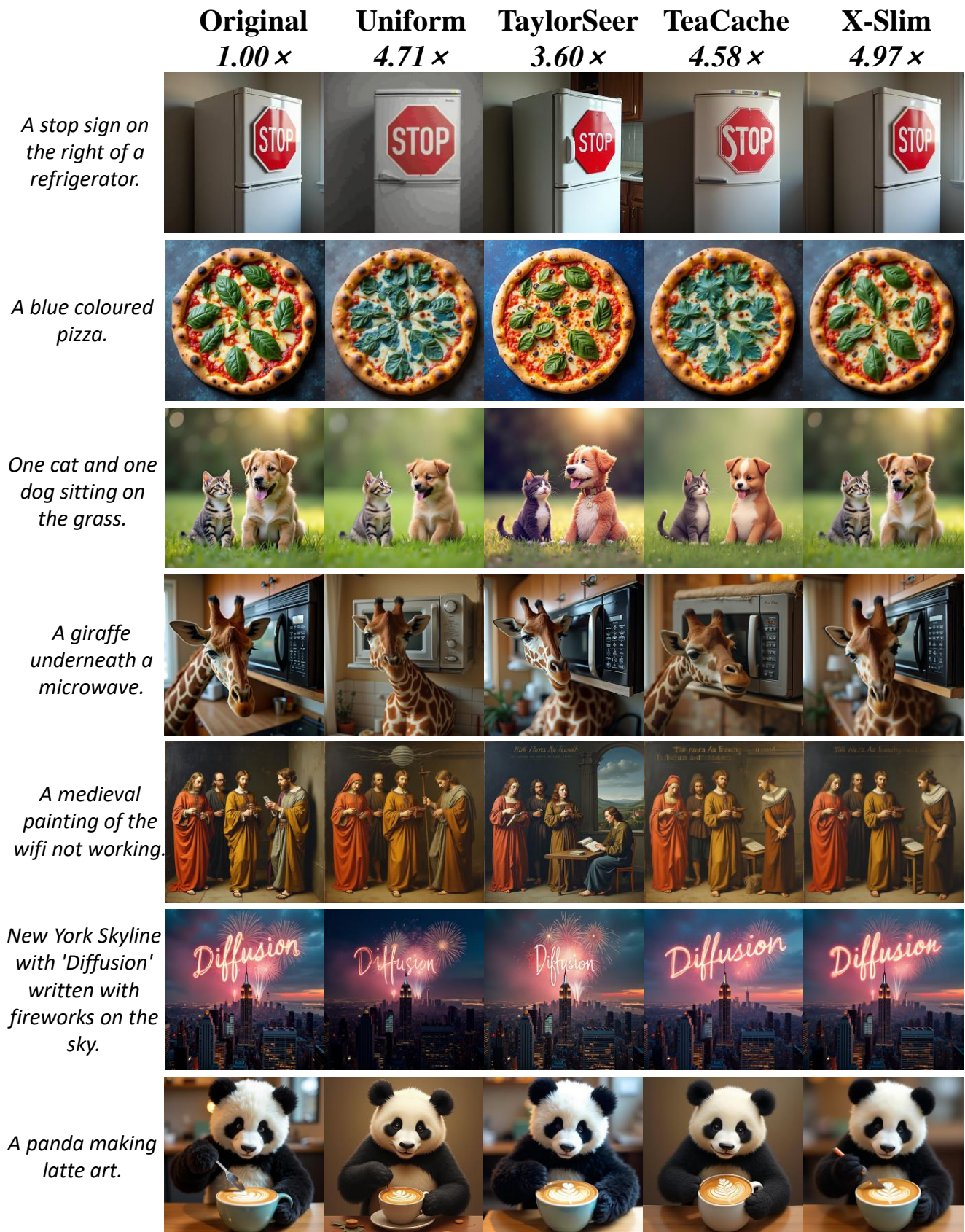


Figure 14. Qualitative comparison on FLUX.1-dev (1/2).



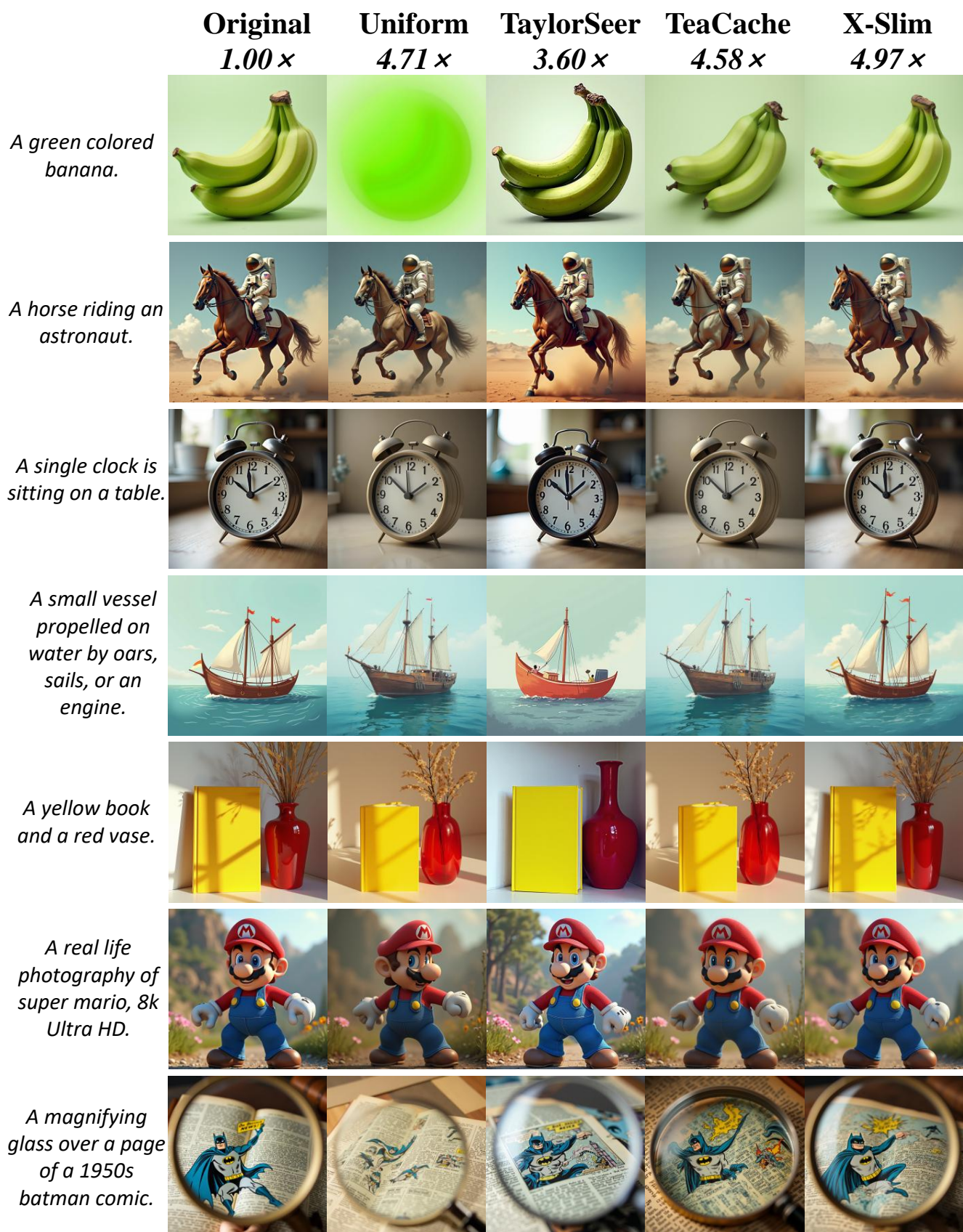


Figure 15. Qualitative comparison on FLUX.1-dev (2/2).

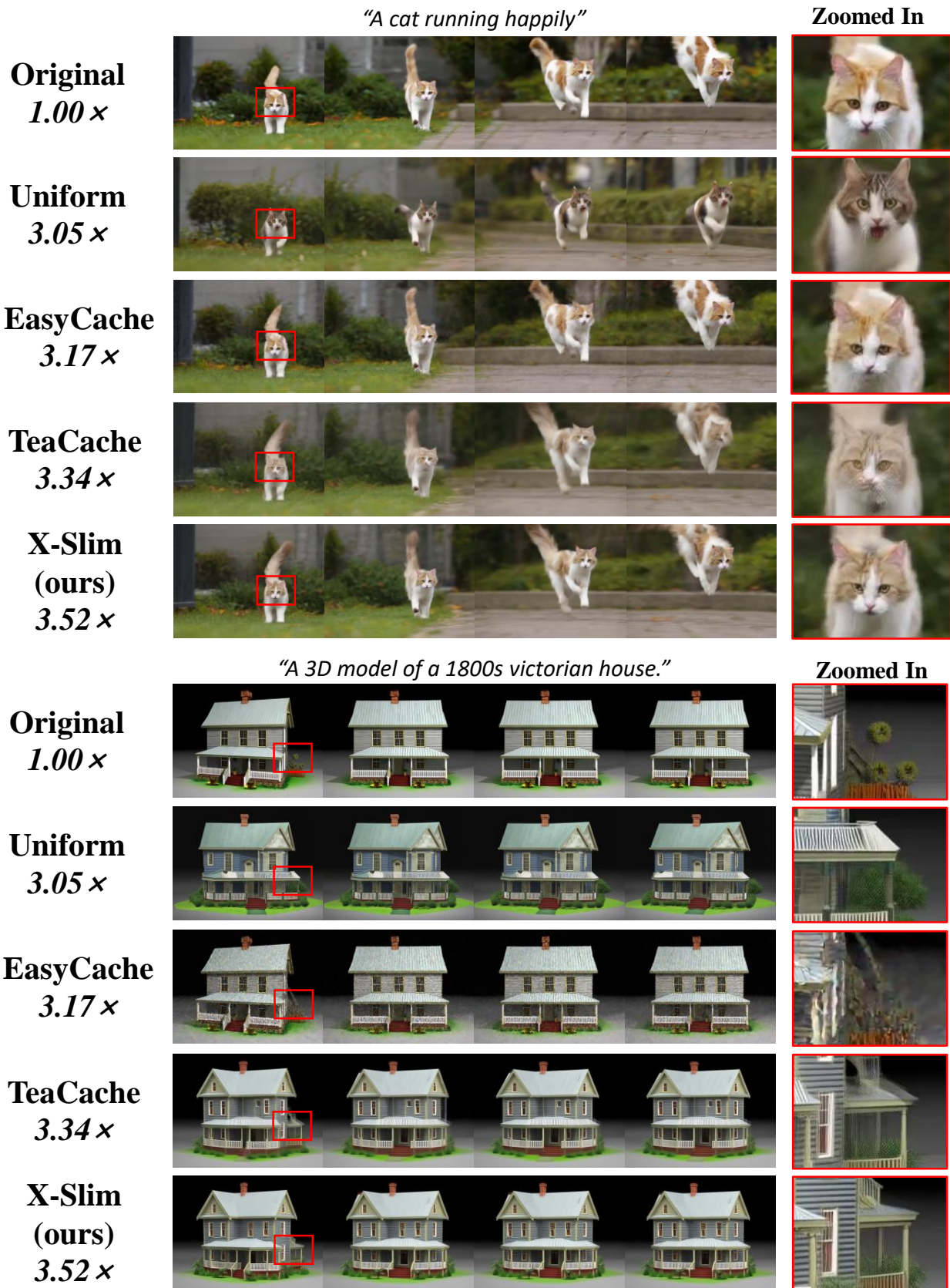


Figure 16. Qualitative comparison on HunyuanVide (1/2).



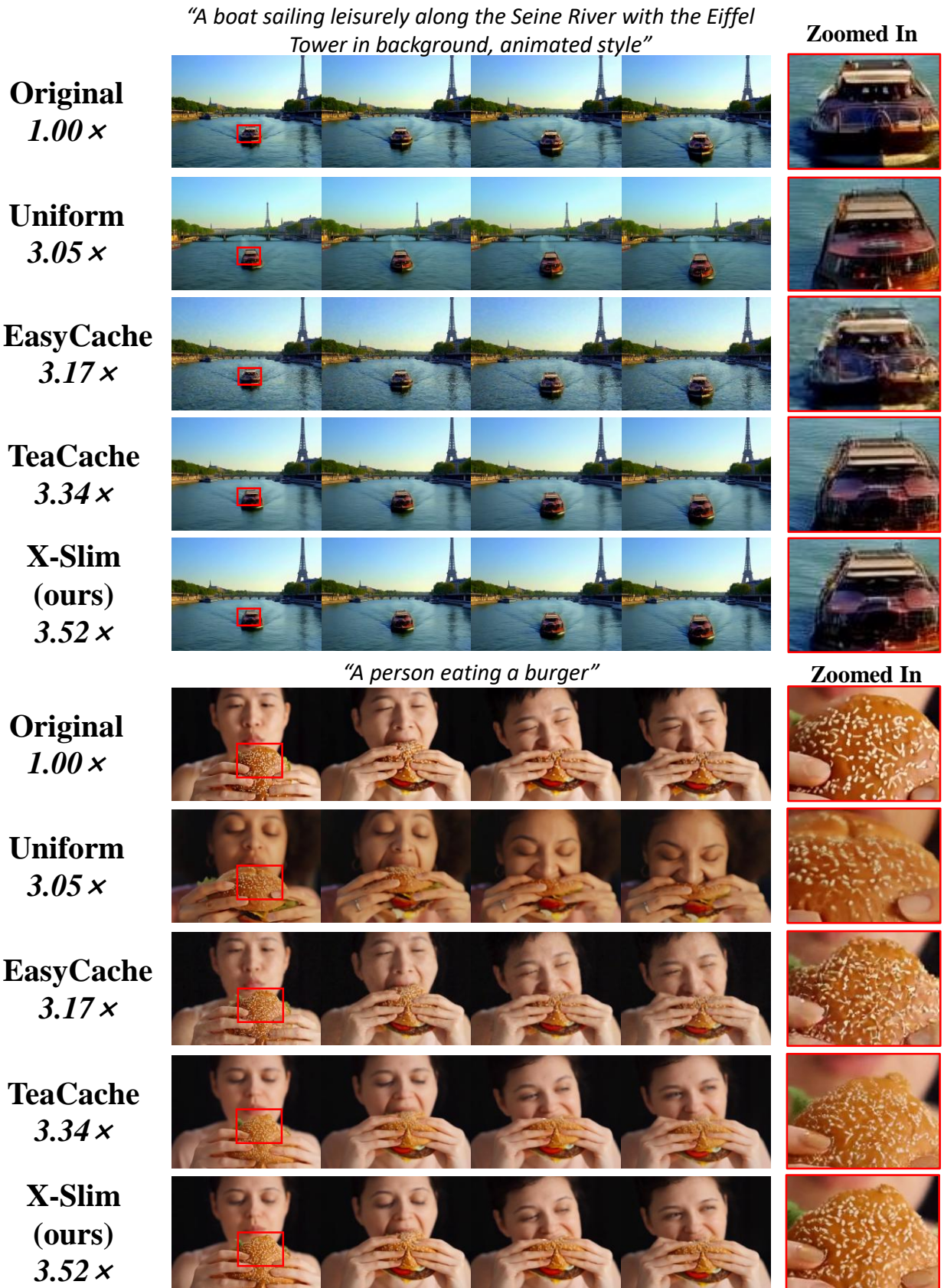


Figure 17. Qualitative comparison on HunyuanVideo (2/2).