FORCINGBOOSTER: ADAPTIVE PATCH-WISE DENOISING FOR FAST AUTOREGRESSIVE VIDEO DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive video diffusion models have become a leading approach to real-time interactive video generation and video world modeling. However, existing methods typically allocate computation uniformly across space when generating the next video chunk, updating all patches with the same denoising schedule and number of function evaluations. This ignores the spatial heterogeneity of video content: static, simple, or recurring regions are often easier to denoise, while other regions require further refinement or additional context. A uniform budget thus leads to redundant computation on easy patches, whereas reducing the fixed step count can leave difficult patches insufficiently refined. We introduce ForcingBooster, a general framework that dynamically allocates denoising steps to individual patches: easy patches stop early to improve efficiency, while difficult patches receive more refinement steps. This design accelerates generation with negligible quality degradation. Our method is compatible with most existing few-step autoregressive diffusion models, requiring only a lightweight routing module with modest training. Experiments demonstrate that ForcingBooster reduces the total number of denoising steps applied across video patches by about 50% while maintaining visual quality comparable to the four-step baseline.

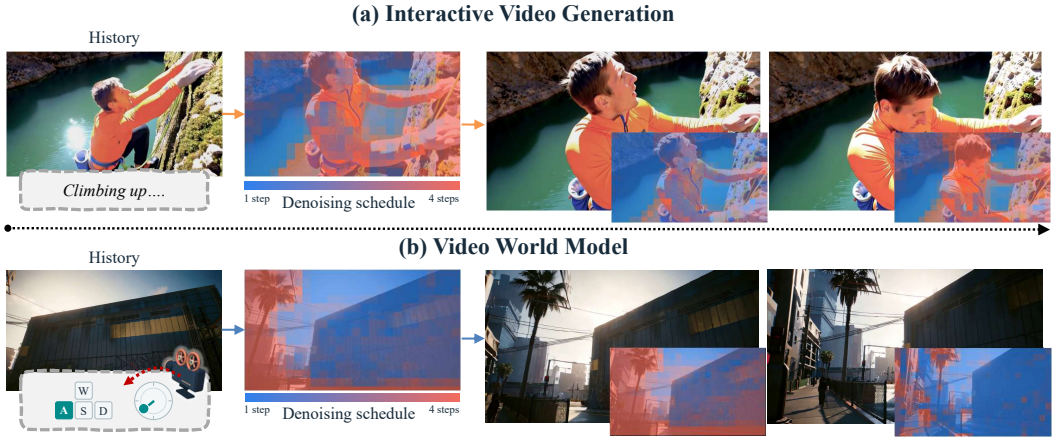

Figure 1: ForcingBooster accelerates interactive video generation and world model by adaptively allocating different denoising steps to different patches. Heatmap overlays visualize router allocations from matching conditional samples: **static backgrounds need fewer updates than moving objects, new regions or detailed textures.**

1 INTRODUCTION

Video generation Wan et al. (2025); HaCohen et al. (2026) is becoming a core approach for digital content creation Liang et al. (2025), scene simulation Sun et al. (2024); Wen et al. (2023), and interactive video world modeling Team et al. (2026); Yuan et al. (2026b). Despite their different goals, these applications all require models to preserve visual quality while responding to input signals

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.

and generating future content with low latency. Recent video diffusion models have substantially improved generation quality, yet their iterative sampling process remains too slow for real-time interaction. Bridging this quality–latency gap is therefore a central challenge for real-time video generation and video world models.

Autoregressive video diffusion converts joint full-video generation into a sequence of next-chunk predictions by introducing causal attention. These methods (Chen et al., 2024; Guo et al., 2026) partition a video into short temporal chunks and generate each chunk conditioned on the preceding ones. Once the first chunk is generated, the model can begin producing output and continue with subsequent chunks. This allows the model to output completed chunks without waiting for the entire video. Causal attention enables the use of KV caching to avoid repeatedly processing previous chunks. Distillation methods Yin et al. (2025); Huang et al. (2025); Zhu et al. (2026); Zhao et al. (2026b) further reduce the generation time of each chunk, compressing models that require dozens of denoising steps into autoregressive generators that use four steps or fewer. Recent methods reduce computation by compressing historical context Yuan et al. (2026a) or choosing the number of denoising steps for entire frames or chunks Le et al. (2026).

Although these methods substantially reduce the denoising iterations required for each frame or chunk, they typically assign the same denoising schedule and computational budget to all latent patches within that chunk. Every new chunk still requires a complete multi-step diffusion process. Some work has explored different denoising schedules across spatial regions in reasoning and image generation Wewer et al. (2025); Schusterbauer et al. (2026). Our analysis indicates that some patches produce predictions close to the full denoising result at earlier steps, while others still require further refinement. For example, as shown in Fig. 1(a), fewer denoising steps are allocated to the background water (blue), while more are allocated to the moving climber and the textured rock face (red). In Fig. 1(b), as the camera pans left, the sky and the large building on the right receive fewer steps, whereas the trees and newly revealed structures on the left receive more. Applying all steps to every patch therefore wastes computation on patches that are already stable, whereas reducing the step count for the entire chunk can degrade regions that still need refinement. Consequently, even when the base model has already been distilled to very few steps, allocating computation according to local content still offers substantial room for improvement.

We propose ForcingBooster, which equips an existing few-step autoregressive generator with a lightweight router that dynamically determines which latent patches require further denoising at each step. The router examines each patch in the context of the generated history and the current conditioning signals, such as camera poses or text controls, to assess whether its prediction has already sufficiently converged. Patches deemed complete stop denoising, while only the remaining uncertain or condition-sensitive regions continue to be refined. In this way, ForcingBooster allocates computation adaptively across space and denoising steps, reducing redundant refinement without modifying the generator itself.

Two key challenges arise in enabling patch-wise early stopping. First, if a patch terminates before completing the full denoising schedule, we need a valid clean latent that can be directly used as its output. To address this, we observe that diffusion and flow-matching generators admit a clean-latent estimate at every intermediate denoising step, either predicted directly or recovered from their native parameterization. This intermediate $\hat{\mathbf{x}}_0$ prediction provides a natural output for early-exited patches, allowing different regions to terminate at different denoising steps without modifying the original sampling schedule.

Second, there is no ground-truth supervision indicating when each patch should stop. We therefore formulate router learning as a trade-off between generation quality and computation. Specifically, we introduce a stopping surrogate loss that penalizes high stop probabilities when the current clean-latent prediction deviates from the full-step denoised latent, together with a computational loss that encourages patches to terminate with fewer denoising steps. These objectives are optimized with on-policy rollouts of the current router, so the estimated continuation cost reflects its evolving routing behavior. The full-step outputs of the frozen generator serve as training references, enabling the router to be trained without additional video data or manually defined stopping labels.

We validate ForcingBooster across video generation and world models with different scales and architectures, ranging from camera-conditioned world models to real-time interactive autoregressive video generators. On average, the router requires only approximately 500 training steps and reduces

the total number of denoising steps applied across video patches by about 50%, while maintaining VBench scores comparable to those of the corresponding four-step baselines.

2 Related Work

2.1 Autoregressive Video Diffusion

Autoregressive video diffusion generates temporal chunks conditioned on earlier ones. Diffusion Forcing trains causal models with independent noise levels across tokens Chen et al. (2024). Later work improves long video generation through history management and noise scheduling Yang et al. (2025); Liu et al. (2025b), or trains on generated histories to reduce error accumulation Po et al. (2025); Guo et al. (2026); Cui et al. (2025). Other work studies multiplayer world modeling Savva et al. (2026) and flexible chunking and generation orders Ma et al. (2026a).

2.2 Diffusion Distillation and Post-Training

Few-step distillation reduces the number of denoiser evaluations. DMD and DMD2 distill diffusion models through distribution matching Yin et al. (2024b;a); Zhao et al. (2026a), and CausVid extends this approach to autoregressive video generation Yin et al. (2025). Self Forcing trains on generated histories Huang et al. (2025), while Causal Forcing uses an autoregressive teacher for initialization Zhu et al. (2026). Further work studies generation in one or two steps Zhao et al. (2026b); Feng et al. (2026), flexible sampling budgets Gu et al. (2026), diversity preservation Chen et al. (2026), and reward supervision Lu et al. (2025). Recent on-policy self-distillation (OPSD) methods use richer teacher context to supervise the model's own rollouts Jiang et al. (2026); Liu et al. (2026a), while DanceOPD Zhou et al. (2026) learns from expert velocity fields on student-generated states. These methods update the generator and typically use a shared step budget across spatial patches within each frame or chunk.

2.3 Adaptive Computation for Diffusion

Adaptive sampling selects denoising budgets for individual samples Zhang et al. (2024), while spatial reasoning models allow different regions to follow different denoising schedules Wewer et al. (2025). Other methods reduce computation within each denoiser evaluation Zhao et al. (2024); You et al. (2025) or process fewer tokens through history compression and multiscale generation Yuan et al. (2026a).

Caching methods reuse intermediate features across denoising steps Ma et al. (2023); Lv et al. (2025). TeaCache estimates output changes from inputs modulated by timestep embeddings to guide reuse Liu et al. (2025a), while AdaCache adapts cache intervals using video features Kahatapitiya et al. (2024). Other methods selectively recompute tokens or regions Zou et al. (2025); Liu et al. (2026b); Xu et al. (2026), extend caching to autoregressive video chunks Ma et al. (2026b), or adjust cache thresholds during denoising Agrawal et al. (2026).

DSA Le et al. (2026) jointly trains a confidence head and the generator to select denoising steps for entire frames or chunks, but spatial patches within each unit still share the same budget. Patch Forcing Schusterbauer et al. (2026) trains image generators with spatially varying noise levels and uses predicted patch difficulty to guide sampling. HSA Chu & Patel (2026) assigns token-level step budgets without training and preserves KV context, while continuing to update skipped latents with cached velocities. In contrast, ForcingBooster learns patch-wise stopping in a frozen few-step autoregressive generator, allowing easy regions to finish early while retaining further refinement for difficult ones. We train only a lightweight router and retain the original denoising schedule. Training balances reconstruction error against denoising cost using full-step references and histories generated under the current routing policy. Rather than advancing skipped latents with cached velocities, we fix each stopped patch's clean latent prediction. Sparse inference then removes it from subsequent Transformer updates once its KV cache is built. This reduces redundant computation without discarding the context needed by active patches and subsequent chunks.

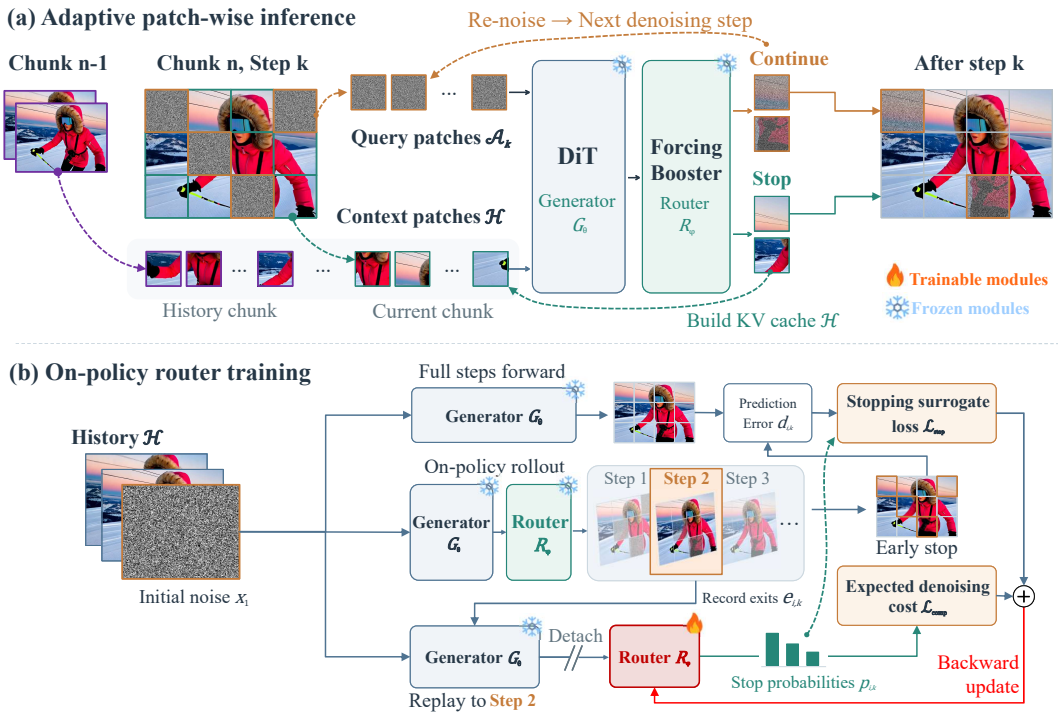

Figure 2: **Adaptive patch-wise inference and on-policy router training.** (a) The router selects patches to continue denoising; stopped predictions provide reusable KV context. (b) A no-gradient rollout records exits for replay. Reconstruction and computational costs train the router through its stopping probabilities.

3 METHOD

3.1 PRELIMINARY AND OVERVIEW

ForcingBooster is designed to improve the efficiency of an existing autoregressive video generator with a router that allocates an adaptive denoising budget for different regions. Specifically, we will first introduce the $\mathbf{x}_0$-prediction mechanism of diffusion or flow matching models, which will be used in our denoising router to early terminate the denoising process for well generated patches.

$\mathbf{x}_0$-prediction of autoregressive video generator. The generator produces videos chunk by chunk in latent space in an autoregressive manner. Each chunk starts from Gaussian noise x_1 and follows a fixed schedule $\{t_k\}_{k=1}^{K}$ with $t_0 = 1,\ t_K = 0$. We express the denoising process at step k as

$$\hat{\mathbf{x}}_{0,t_{k+1}} = G_\theta\left(\mathbf{x}_{t_k}, t_k; \mathcal{H}, \mathbf{y}\right), \qquad \mathbf{x}_{t_{k+1}} = \mathcal{E}_{t_{k+1}}(\hat{\mathbf{x}}_{0,t_{k+1}}) \qquad k = 1, \ldots, K, \tag{1}$$

where G_θ is the denoising network with parameters θ, $\mathbf{x}_{t_k}$ is the noisy input at noise level t_k, $\mathbf{y}$ denotes generation conditions, $\mathcal{H}$ denotes the generation history represented by the KV cache of previously generated frames, $\hat{\mathbf{x}}_{0,t_{k+1}}$ is the predicted clean latent at this step, and $\mathcal{E}_{t_{k+1}}$ maps this clean prediction to the next noisy state $\mathbf{x}_{t_{k+1}}$. This formulation abstracts over the specific parameterization used by diffusion or flow-matching models: even when the network predicts noise or velocity, an equivalent clean-latent estimate $\hat{\mathbf{x}}_{0,t_{k+1}}$ can be obtained at each step. Importantly, this intermediate clean prediction allows a patch to terminate denoising early and directly use its current $\hat{\mathbf{x}}_{0,t_k}$ as the final latent prediction.

Denoising router. A vanilla autoregressive video generator will use the same number of denoising steps K for every patch in the current generation chunk. However, different patches may require substantially different amounts of denoising budget. In particular, regions that can be reliably inferred from the generation history $\mathcal{H}$ may converge much earlier than newly revealed or condition-

dependent regions. To exploit this redundancy, ForcingBooster applies a denoising router that predicts a stop probability $p_{i,k}$ for all patches after every denoising step. Then, for patches with large stop probability, we will stop denoising these patches in the subsequent steps, use their predicted clean latents $\hat{\mathbf{x}}_{0,t_k}$ at this step as the output, and further store the KV caches of these patches to $\mathcal{H}$ for the denoising of other non-stopped patches.

In the following, we describe how our router changes and speeds up the generation process (Section 3.2) and how we train our router with a surrogate loss (Section 3.3).

3.2 Adaptive Patch-wise Denoising Router

After each denoising step, the router predicts a stop probability for each patch to stop the denoising of well-generated patches. Then, we add the KV cache of these stopped patches to the history $\mathcal{H}$ and proceed to the next denoising step.

Stop probability prediction. Let $\mathcal{A}_k$ denote the active patches at the beginning of step k and $\mathcal{A}_0$ includes all patches. After the denoising step k, the router R_ϕ reuses hidden features $\mathbf{u}_k$ from the generator's last transformer layer to predict the stop probability $p_{i,k}$ by

$$p_{i,k} = \left[R_\phi(\mathbf{u}_k)\right]_i, \qquad i \in \mathcal{A}_k. \tag{2}$$

where ϕ denotes the router parameters. A threshold η is used to determine the active patches $\mathcal{A}_{k+1}$

$$\mathcal{A}_{k+1} = \{i \in \mathcal{A}_k \mid p_{i,k} \le \eta\}, \qquad k < K. \tag{3}$$

As shown in Fig. 2(a), only $\mathcal{A}_{k+1}$ will be used in the denoising of the next step. For patches stopped at this step, their $\hat{\mathbf{x}}_{0,t_{k+1}}$ from Eq. (1) are used as output. Meanwhile, the KV values of these stopped patches will be stored into the KV cache $\mathcal{H}$ to help the denoising of the remaining patches.

Computation reduction. The router progressively reduces the number of active patches throughout denoising. Once a patch stops, it is removed from subsequent denoising updates and its key-value representations are cached for reuse. Therefore, later denoising steps only compute new representations for the remaining active patches, while stopped patches continue to provide contextual information through the KV cache without being recomputed. As more patches terminate, the active set shrinks monotonically, reducing the per-step computation and accelerating the generation.

3.3 On-policy Router Training

Learning the stop probability. Training the router is non-trivial because there is no ground-truth annotation indicating at which denoising step each patch should stop. Instead of constructing discrete stopping labels, we directly relate the stop probability to the quality of the current clean-latent prediction. Intuitively, if the predicted clean latent $\hat{\mathbf{x}}_{0,t_k}$ at step k is already close to the target clean latent, further denoising brings little benefit and the router should assign a high stop probability. Conversely, a large prediction error indicates that the patch still requires further refinement and should receive a low stop probability. Based on this observation, we introduce a differentiable surrogate loss and a denoising cost loss to train the router.

Stopping surrogate loss. Let $\mathbf{x}_0^*$ denote the target clean latent, and let

$$d_{i,k} = \left\| \hat{\mathbf{x}}^{\,i}_{0,t_k} - \mathbf{x}_0^{*,i} \right\|_2^2 \tag{4}$$

measure the prediction error of patch i after step k. We penalize stopping in proportion to this error:

$$\mathcal{L}_{\text{stop}} = \frac{1}{|\mathcal{A}_k|} \sum_{i \in \mathcal{A}_k} p_{i,k}\, d_{i,k}. \tag{5}$$

This objective discourages a high stop probability when the current prediction is still inaccurate, while imposing little penalty on stopping patches with converged clean-latent predictions.



Figure 3: **Qualitative comparison.** The rows show the results from four-step baseline to one-step sampling, and ForcingBooster. One-step and Two-step sampling exhibits a decline in quality and visible artifacts. Red boxes indicate visual artifacts.

Expected denoising cost. Optimizing the surrogate loss alone will lead to a trivial solution in which the router assigns a zero stop probability to every patch, allowing all patches to run through the complete denoising schedule. To encourage early stopping, we introduce a denoising cost loss that minimizes the expected number of denoising steps for each patch. For a candidate patch i at step k, stopping immediately incurs a cost of k, whereas continuing denoising incurs a cost determined by the final stopping step $e_{i,k}$ of our routing policy. The denoising cost is then

$$\mathcal{L}_{\text{comp}} = \frac{1}{|\mathcal{A}_k|} \sum_{i \in \mathcal{A}_k} \left[p_{i,k} \cdot k + (1 - p_{i,k}) \cdot e_{i,k} \right]. \tag{6}$$

Here, $p_{i,k} \cdot k$ corresponds to the cost of stopping at the current step, while $(1 - p_{i,k}) \cdot e_{i,k}$ represents the cost of continuing until the patch eventually exits. Since $e_{i,k} > k$, minimizing this objective favors a larger stop probability whenever further denoising is unnecessary, thereby reducing the overall denoising computation. To estimate $e_{i,k}$, we roll out the remaining denoising steps using the current router and stopping threshold η, and record when each active patch eventually exits. This on-policy estimation allows the computational cost to reflect the behavior of the current routing policy as it evolves during training.

Self-generated training targets. The target $\mathbf{x}_0^*$ can be obtained directly from the frozen generator itself, without requiring additional video data or stopping annotations. Specifically, we first run the original generator with its full denoising schedule and use the resulting clean latent as the training target, and record this trajectory. We then use this trajectory perturb this latent to a randomly sampled noise level t_k and feed the noisy latent, which is recorded in the trajectory, back to the frozen generator to obtain the intermediate clean prediction $\hat{\mathbf{x}}_{0,t_k}$ and the corresponding hidden features for router prediction. We refer to this process as "replay." The discrepancy between $\hat{\mathbf{x}}_{0,t_k}$ and the full-step result therefore directly measures how much quality would be lost by stopping at step k. In this way, the pretrained generator provides both the training inputs and supervision for the router, allowing it to be trained without collecting any additional data.

4 Experiments

We evaluate ForcingBooster on several video generation methods and world models, comparing generation quality and inference time with uniform fixed-step sampling. We then analyze spatial computation allocation and router settings.

Table 1: Comparison on Causal Forcing checkpoint. Time includes generation through VAE decoding. Speedup and denoising patch-NFE reduction are relative to each model's four-step baseline.

Method	Quality ↑	Semantic ↑	Total ↑	Time (s) ↓	Speedup ↑	NFE Reduction ↑
1 step	83.06	69.50	80.35	7.08	1.58×	64.29%
2 steps	83.69	71.66	81.29	8.67	1.29×	42.86%
3 steps	84.24	71.96	81.78	9.78	1.14×	21.43%
4 steps (baseline)	84.54	72.21	82.08	11.15	-	-
ForcingBooster	83.88	72.20	81.55	8.56	1.30×	50.19%

4.1 Implementation Details

Our main experiments use Causal Forcing 1.3B (Zhu et al., 2026) to generate 81-frame videos at 480×832 and 16 FPS. Following its training data setup, we train only the router for 500 steps using AdamW, a learning rate of 10^{-4}, and a global batch size of 8. The pretrained components and denoising schedule remain fixed. Following common practice Gu et al. (2026); Huang et al. (2025); Zhu et al. (2026), we adopt VBench Huang et al. (2023) to evaluate generated videos across 16 fine-grained dimensions. We report the aggregated Quality and Semantic scores, together with their weighted combination, the Total score. For this model, inference uses a single A800 GPU with BF16 precision and batch size 1. We report mean end-to-end generation time, including text encoding, the first chunk, router execution, KV cache construction, and VAE decoding.

4.2 Main Results

We compare adaptive patch-wise denoising with uniform sampling using one to four steps. The original four-step sampler (vanilla) without routing serves as the baseline. All methods use four steps for the first chunk and apply their respective strategies to subsequent chunks. Fixed-step baselines execute a prefix of the original schedule and return the clean latent prediction from the last executed step. Generation conditions and noise are matched across methods.

Table 1 shows that ForcingBooster achieves a $1.3\times$ end-to-end speedup reduces the total number of denoising steps applied across video patches by about 50%, while retaining Quality and Total scores close to the four-step baseline and an almost unchanged Semantic score. At a runtime similar to uniform two-step sampling, it also achieves higher scores on all three metrics, showing the benefit of allocating denoising steps across patches rather than reducing them uniformly. The qualitative comparison in Fig. 3 reveals a clearer advantage of ForcingBooster than the quantitative results suggest. The one-step, two-step, and three-step baselines all produce artifacts in regions with complex textures. In contrast, ForcingBooster computes more on these patches and avoids artifacts.

4.3 Further Evaluation

We further evaluate Wan2.1 14B, LingBot-World-V2, and LongLive. Table 2 reports Quality, Semantic, Total, and speedup for each model's original four-step baseline, one-step, two-step baseline and ForcingBooster. We compare methods within each base model to examine whether patch-wise computation allocation offers a useful trade-off across model scales and generation tasks.

Across these models, ForcingBooster reduces runtime while maintaining Quality and Total scores close to the corresponding four-step baselines (Table 2). It also achieves higher Quality and Total scores than uniform one-step sampling in every case. The qualitative comparison in Fig. 4 also indicates that one-step sampling results exhibit artifacts and a decline in quality across various models. Overall, adaptive patch-wise denoising achieves visual quality comparable to the four-step results while reducing inference time, and delivers superior visual quality and overall performance compared to uniform one-step sampling.

4.4 Patch-wise Allocation

We compare different computation allocation strategies to examine whether the improvement in generation quality comes from effectively selecting spatial positions. Random preserves the distribution of exit steps within each latent frame but randomly permutes the spatial positions assigned to

Table 2: Results on additional world model and interactive video generation methods. Speedups include VAE decoding and are relative to each model's four-step baseline. All methods use four steps for the first chunk.

Base model	Quality ↑	Semantic ↑	Total ↑	Speedup ↑
Wan2.1 14B (1 step)	82.10	71.35	79.95	1.96×
Wan2.1 14B (2 step)	84.71	77.93	83.35	1.29×
Wan2.1 14B (4 steps)	84.96	78.04	83.57	-
Wan2.1 14B + ForcingBooster	84.73	77.65	83.32	1.31×
LingBot-World-V2 14B (1 step)	78.79	69.72	76.98	1.98×
LingBot-World-V2 14B (2 step)	80.13	71.65	78.43	1.36×
LingBot-World-V2 14B (4 steps)	80.66	72.96	79.12	-
LingBot-World-V2 14B + ForcingBooster	80.34	71.32	78.54	1.25×
LongLive (1 step)	78.91	73.18	77.76	1.54×
LongLive (2 step)	79.91	72.67	78.46	1.21×
LongLive (4 steps)	80.32	73.76	79.01	-
LongLive + ForcingBooster	80.02	72.82	78.58	1.15×

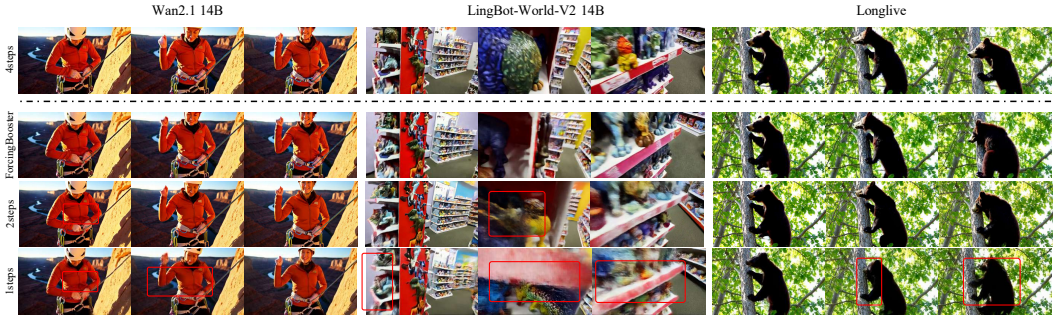

Figure 4: **Qualitative comparison across additional models.** ForcingBooster produces visual results comparable to the four-step baseline, while uniform one-step sampling often exhibits a decline in quality and visible artifacts. Red boxes indicate visual artifacts.

those steps. Chunk-wise router aggregates patch-wise exit decisions into a uniform number of denoising steps for each chunk. We compare these two strategies with ForcingBooster and the original four-step baseline.

At comparable runtimes, ForcingBooster achieves a higher Total score than both alternatives (Table 3). The comparison with Random shows that the locations of early exits matter, not just their step-count distribution. The improvement over Chunk-wise router further supports assigning different step counts within the same chunk to preserve generation quality. Fig. 5 shows the four methods at multiple time points under the same generation conditions. Chunk-wise router produces blurry frames in chunks assigned fewer denoising steps, with weaker details in regions such as the face and hands. Random exhibits severe flickering and detail loss in the generated video, with unstable local structures in the person and objects over time. In contrast, ForcingBooster better preserves visual sharpness and temporal consistency, producing results closest overall to four-step baseline.

4.5 Router Analysis

We study the computational cost weight λ_{nfe} and stopping threshold η. We measure agreement with the four-step output using LPIPS and report full-video runtime normalized by the four-step baseline.

Computational cost. We vary λ_{nfe} in the training objective from Section 3.3. With zero or insufficient weight on $\mathcal{L}_{\text{nfe}}$, the router retains all four steps and adds overhead without reducing runtime (Fig. 6(a)). Weights between 0.015 and 0.030 enable substantial time savings with low LPIPS. Further increases bring smaller runtime gains and larger deviations from the four-step output. These results show that a sufficient cost penalty is needed to learn effective early stopping, while overemphasizing it offers limited further acceleration.



Figure 5: **Qualitative comparison of computation allocation.** Columns show frames at matching time points under the same generation conditions. Red boxes indicate visual artifacts.

Table 3: Comparison of computation allocation strategies. Time measures full-video generation, and speedups are relative to the four-step baseline.

Method	Total ↑	Time (s)↓	Speedup ↑
Random	79.03	8.60	1.29×
Chunk-wise router	80.01	8.38	1.33×
ForcingBooster	81.54	8.53	1.30×
4 steps (baseline)	82.07	11.13	-

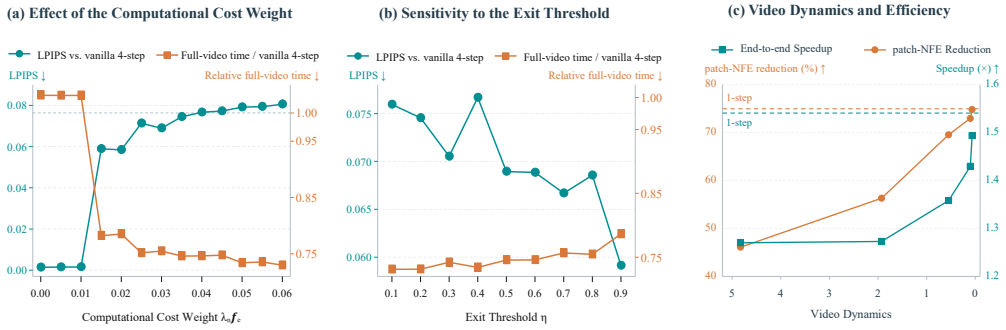

Figure 6: **Router analysis.** **(a)** Effect of the computational cost weight λ_{nfe}. **(b)** Effect of the stopping threshold η, with a separate router trained for each threshold and the same threshold used at inference. Both panels report LPIPS against the four-step output and normalized full-video runtime; lower is better.

Stopping threshold. We vary η from Section 3.2, training a separate router for each value from 0.1 to 0.9 with $\lambda_{\text{nfe}} = 0.030$. Each router uses the same threshold during training and inference. Across these routers, higher thresholds generally increase runtime but do not consistently lower LPIPS (Fig. 6(b)). Thus, a more conservative stopping threshold does not necessarily produce outputs closer to the four-step reference.

Video dynamics and speedup. We quantify video dynamics using RAFT Teed & Deng (2020) to estimate optical flow between adjacent frames, averaging its magnitude over space and time. We divide the videos into five equally sized groups by this measure and report the inference speedup and patch-wise NFE reduction for each group. As shown in Fig. 6(c), ForcingBooster achieves larger speedups on lower-motion videos, approaching the runtime of one-step denoising with up to approximately $1.5\times$ speedup or reduces the total number of denoising steps applied across video patches by about 75% over the four-step baseline.

5 Conclusion

We introduce ForcingBooster, a general framework that dynamically allocates denoising steps to individual patches: simple patches terminate early to improve efficiency, whereas complex patches receive additional refinement steps. This design accelerates generation with negligible quality degradation. ForcingBooster is compatible with most existing few-step autoregressive diffusion models and requires only a lightweight routing module and modest training, without fine-tuning the heavyweight generator or relying on additional video data.

AI USE STATEMENT

We used generative AI tools to assist with language editing, translation, and organization of the manuscript, and to automate parameter sweeps in the ablation study. The authors reviewed the AI-assisted text and checked the experimental settings, execution scripts, and reported results. The authors take full responsibility for the final content of this work.

ETHICS STATEMENT

This work introduces adaptive patch-wise denoising to accelerate autoregressive video diffusion models while preserving generation quality. The router is trained using outputs from a frozen generator and does not require additional real videos as reconstruction targets. We present the method for research purposes and encourage downstream users to use appropriately licensed data, obtain necessary consent, protect personal information, disclose synthetic content, and adopt safeguards against misuse.

REPRODUCIBILITY STATEMENT

The method, training objectives, and evaluation protocol are described in Sections 3 and 4. The appendix provides implementation details for the base models, router architecture, evaluation samples, timing protocol, and ablation settings (Appendix A), together with additional controlled experiments and efficiency measurements (Appendix B). Upon acceptance, we will release all code and training configurations to reproduce our experiments.

REFERENCES

Om Agrawal, Saurabh Agarwal, and Aditya Akella. Acid: Adaptive caching for video generation, 2026. URL https://arxiv.org/abs/2607.12358.

Boyuan Chen, Diego Marti Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion, 2024. URL https://arxiv.org/abs/2407.01392.

Siyi Chen, Shaowei Liu, Yixuan Jia, Zian Wang, Huan Ling, Qing Qu, and Jun Gao. Data-forcing distillation: Restoring diversity and fidelity in few-step video generation, 2026. URL https://arxiv.org/abs/2606.18478.

Ernie Chu and Vishal M. Patel. Not all tokens need 40 steps: Heterogeneous step allocation in diffusion transformers for efficient video generation, 2026. URL https://arxiv.org/abs/2605.06892.

Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Self-forcing++: Towards minute-scale high-quality video generation, 2025. URL https://arxiv.org/abs/2510.02283.

Jiaqi Feng, Justin Cui, Yuanhao Ban, and Cho-Jui Hsieh. One-forcing: Towards stable one-step autoregressive video generation, 2026. URL https://arxiv.org/abs/2605.23458.

Yuchao Gu, Guian Fang, Yuxin Jiang, Weijia Mao, Song Han, Han Cai, and Mike Zheng Shou. Anyflow: Any-step video diffusion model with on-policy flow map distillation, 2026. URL https://arxiv.org/abs/2605.13724.

Yuwei Guo, Ceyuan Yang, Hao He, Yang Zhao, Meng Wei, Zhenheng Yang, Weilin Huang, and Dahua Lin. End-to-end training for autoregressive video diffusion via self-resampling, 2026. URL https://arxiv.org/abs/2512.15702.

Yoav HaCohen, Benny Brazowski, Nisan Chiprut, Yaki Bitterman, Andrew Kvochko, Avishai Berkowitz, Daniel Shalem, Daphna Lifschitz, Dudu Moshe, Eitan Porat, Eitan Richardson, Guy Shiran, Itay Chachy, Jonathan Chetboun, Michael Finkelson, Michael Kupchick, Nir Zabari, Nitzan Guetta, Noa Kotler, Ofir Bibi, Ori Gordon, Poriya Panet, Roi Benita, Shahar Armon, Victor Kulikov, Yaron Inger, Yonatan Shiftan, Zeev Melumian, and Zeev Farbman. Ltx-2: Efficient joint audio-visual foundation model, 2026. URL https://arxiv.org/abs/2601.03233.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion, 2025. URL https://arxiv.org/abs/2506.08009.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Vbench: Comprehensive benchmark suite for video generative models, 2023. URL https://arxiv.org/abs/2311.17982.

Dengyang Jiang, Xin Jin, Dongyang Liu, Zanyi Wang, Mingzhe Zheng, Ruoyi Du, Xiangpeng Yang, Qilong Wu, Zhen Li, Peng Gao, Harry Yang, and Steven Hoi. D-opsd: On-policy self-distillation for continuously tuning step-distilled diffusion models, 2026. URL https://arxiv.org/abs/2605.05204.

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, 2024. URL https://arxiv.org/abs/2411.02397.

Thanh-Tung Le, Yunhan Zhao, Menglei Chai, Zhengyang Shen, Zhe Cao, Danhang Tang, Xiaohui Xie, and Deying Kong. Dsa: Dynamic step allocation for fast autoregressive video generation, 2026. URL https://arxiv.org/abs/2606.04432.

Ruofan Liang, Zan Gojcic, Huan Ling, Jacob Munkberg, Jon Hasselgren, Zhi-Hao Lin, Jun Gao, Alexander Keller, Nandita Vijaykumar, Sanja Fidler, and Zian Wang. Diffusionrenderer: Neural inverse and forward rendering with video diffusion models, 2025. URL https://arxiv.org/abs/2501.18590.

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, 2025a. URL https://arxiv.org/abs/2411.19108.

Hongyu Liu, Chun Wang, Feng Gao, Xuanhua He, Yue Ma, Ziyu Wan, Yong Zhang, Xiaoming Wei, and Qifeng Chen. Opsd-v: On-policy self-distillation for post-training few-step autoregressive video generators, 2026a. URL https://arxiv.org/abs/2607.08766.

Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling forcing: Autoregressive long video diffusion in real time, 2025b. URL https://arxiv.org/abs/2509.25161.

Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers, 2026b. URL https://arxiv.org/abs/2502.10389.

Yunhong Lu, Yanhong Zeng, Haobo Li, Hao Ouyang, Qiuyu Wang, Ka Leong Cheng, Jiapeng Zhu, Hengyuan Cao, Zhipeng Zhang, Xing Zhu, Yujun Shen, and Min Zhang. Reward forcing: Efficient streaming video generation with rewarded distribution matching distillation, 2025. URL https://arxiv.org/abs/2512.04678.

Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. Fastercache: Training-free video diffusion model acceleration with high quality, 2025. URL https://arxiv.org/abs/2410.19355.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free, 2023. URL https://arxiv.org/abs/2312.00858.

Xinyin Ma, Julius Berner, Chao Liu, Arash Vahdat, Weili Nie, and Xinchao Wang. Flex-forcing: Towards a unified autoregressive and bidirectional video diffusion model, 2026a. URL https://arxiv.org/abs/2607.03509.

Yuexiao Ma, Xuzhe Zheng, Jing Xu, Xiwei Xu, Feng Ling, Xiawu Zheng, Huafeng Kuang, Huixia Li, Xing Wang, Xuefeng Xiao, Fei Chao, and Rongrong Ji. Flow caching for autoregressive video generation, 2026b. URL https://arxiv.org/abs/2602.10825.

Ryan Po, Eric Ryan Chan, Changan Chen, and Gordon Wetzstein. Bagger: Backwards aggregation for mitigating drift in autoregressive video diffusion models, 2025. URL https://arxiv.org/abs/2512.12080.

Georgy Savva, Oscar Michel, Daohan Lu, Suppakit Waiwitlikhit, Timothy Meehan, Dhairya Mishra, Srivats Poddar, Jack Lu, and Saining Xie. Solaris: Building a multiplayer video world model in minecraft, 2026. URL https://arxiv.org/abs/2602.22208.

Johannes Schusterbauer, Ming Gui, Yusong Li, Pingchuan Ma, Felix Krause, and Björn Ommer. Denoising, fast and slow: Difficulty-aware adaptive sampling for image generation, 2026. URL https://arxiv.org/abs/2604.19141.

Shuo Sun, Zekai Gu, Tianchen Sun, Jiawei Sun, Chengran Yuan, Yuhang Han, Dongen Li, and Marcelo H. Ang. DriveSceneGen: Generating Diverse and Realistic Driving Scenarios from Scratch. *IEEE Robotics and Automation Letters*, 9(8):7007–7014, 2024. doi: 10.1109/LRA.2024.3416792. URL https://arxiv.org/abs/2309.14685.

Robbyant Team, Zelin Gao, Qiuyu Wang, Yanhong Zeng, Jiapeng Zhu, Ka Leong Cheng, Yixuan Li, Hanlin Wang, Yinghao Xu, Shuailei Ma, Yihang Chen, Jie Liu, Yansong Cheng, Yao Yao, Jiayi Zhu, Yihao Meng, Kecheng Zheng, Qingyan Bai, Jingye Chen, Zehong Shen, Yue Yu, Xing Zhu, Yujun Shen, and Hao Ouyang. Advancing open-source world models, 2026. URL https://arxiv.org/abs/2601.20540.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow, 2020. URL https://arxiv.org/abs/2003.12039.

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

Yuqing Wen, Yucheng Zhao, Yingfei Liu, Fan Jia, Yanhui Wang, Chong Luo, Chi Zhang, Tiancai Wang, Xiaoyan Sun, and Xiangyu Zhang. Panacea: Panoramic and controllable video generation for autonomous driving, 2023. URL https://arxiv.org/abs/2311.16813.

Christopher Wewer, Bart Pogodzinski, Bernt Schiele, and Jan Eric Lenssen. Spatial reasoning with denoising models, 2025. URL https://arxiv.org/abs/2502.21075.

Jing Xu, Yuexiao Ma, Xuzhe Zheng, Xing Wang, Shiwei Liu, Chenqian Yan, Xiawu Zheng, Rongrong Ji, Fei Chao, and Songwei Liu. Motion-aware caching for efficient autoregressive video generation, 2026. URL https://arxiv.org/abs/2605.01725.

Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Yingcong Chen, Yao Lu, Song Han, and Yukang Chen. Longlive: Real-time interactive long video generation, 2025. URL https://arxiv.org/abs/2509.22622.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis, 2024a. URL https://arxiv.org/abs/2405.14867.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation, 2024b. URL https://arxiv.org/abs/2311.18828.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models, 2025. URL https://arxiv.org/abs/2412.07772.

Haoran You, Connelly Barnes, Yuqian Zhou, Yan Kang, Zhenbang Du, Wei Zhou, Lingzhi Zhang, Yotam Nitzan, Xiaoyang Liu, Zhe Lin, Eli Shechtman, Sohrab Amirghodsi, and Yingyan Celine Lin. Layer- and timestep-adaptive differentiable token compression ratios for efficient diffusion transformers, 2025. URL https://arxiv.org/abs/2412.16822.

Shenghai Yuan, Yuanyang Yin, Zongjian Li, Xinwei Huang, Xiao Yang, and Li Yuan. Helios: Real real-time long video generation model, 2026a. URL https://arxiv.org/abs/2603.04379.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination?, 2026b. URL https://arxiv.org/abs/2603.16666.

Hui Zhang, Zuxuan Wu, Zhen Xing, Jie Shao, and Yu-Gang Jiang. Adadiff: Adaptive step selection for fast diffusion models, 2024. URL https://arxiv.org/abs/2311.14768.

Chuyang Zhao, Yifei Song, Hongfa Wang, Jianlong Yuan, Yuan Zhang, Siming Fu, Zhineng Chen, Huilin Deng, Haoyang Huang, and Nan Duan. Perceptual flow matching for few-step generative modeling, 2026a. URL https://arxiv.org/abs/2607.03524.

Min Zhao, Hongzhou Zhu, Kaiwen Zheng, Zihan Zhou, Bokai Yan, Xinyuan Li, Xiao Yang, Chongxuan Li, and Jun Zhu. Causal forcing++: Scalable few-step autoregressive diffusion distillation for real-time interactive video generation, 2026b. URL https://arxiv.org/abs/2605.15141.

Wangbo Zhao, Yizeng Han, Jiasheng Tang, Kai Wang, Yibing Song, Gao Huang, Fan Wang, and Yang You. Dynamic diffusion transformer, 2024. URL https://arxiv.org/abs/2410.03456.

Wei Zhou, Xiongwei Zhu, Zelin Xu, Bo Dong, Lixue Gong, Yongyuan Liang, Meng Chu, Leigang Qu, Lingdong Kong, Wei Liu, and Tat-Seng Chua. Danceopd: On-policy generative field distillation, 2026. URL https://arxiv.org/abs/2606.27377.

Hongzhou Zhu, Min Zhao, Guande He, Hang Su, Chongxuan Li, and Jun Zhu. Causal forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation, 2026. URL https://arxiv.org/abs/2602.02214.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching, 2025. URL https://arxiv.org/abs/2410.05317.

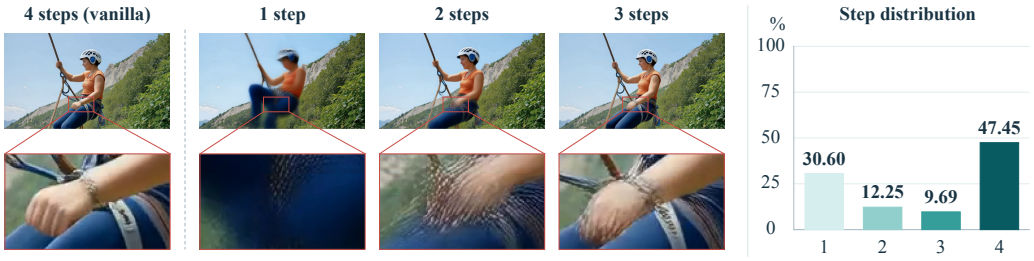

Figure 7: **Denoising steps required by different patches.** Left: fixed-step generation results with enlarged local details. Right: patch-wise first-hit step distribution at 35 dB PSNR relative to the four-step reference in a separate controlled experiment.

A Additional Implementation Details

Base models. Causal Forcing 1.3B is based on Wan2.1-T2V-1.3B. Other experiments use a four-step causal model based on Wan2.1 14B, LingBot-World-V2 14B, and LongLive. The Wan2.1 14B model uses the same video length, resolution, and frame rate as the main experiments. For LingBot-World-V2 14B, we evaluate 77- and 157-frame videos and retain the original preprocessing procedure, which preserves the input aspect ratio.

Router. The router takes patch-wise features from the generator's last Transformer layer before the output head. It is a two-layer MLP with LayerNorm and a SiLU activation, containing approximately 3.15 million parameters. The frozen generator produces both the training histories and supervision outputs, without requiring additional real videos as reconstruction targets.

Evaluation samples. The main VBench evaluation uses the 944 unique prompts in the official metadata, with five base samples generated per prompt. We generate additional candidate videos for temporal flickering evaluation, resulting in 6,220 videos per method.

Timing and ablation settings. For Causal Forcing 1.3B, inference uses neither compilation optimizations nor sequence parallelism. End-to-end timing starts with text encoding and ends when the VAE-decoded pixels are available on the GPU. For the stopping-threshold ablation, all configurations use the same initialization and random seed. Each video in this ablation is timed three times, and its median runtime is used for evaluation.

B Additional Experiment

B.1 Denoising Steps Required by Different Patches

A few-step schedule reduces the number of denoising updates, but still applies the same step count to every patch. Some patches may already be close to the full-step output while others still need further updates. If this difference exists, we may be able to reduce computation selectively rather than shorten the schedule for all patches. We therefore conduct a controlled experiment to examine when each patch's prediction first approaches the full-step output.

We use Causal Forcing as the base model, fix the context to the first ground-truth video chunk, and generate only the immediately following target chunk. Each target chunk contains three latent frames. We hold the generation conditions, initial noise, and noise between denoising steps fixed and record the clean latent prediction at each step of a single four-step sampling run.

We decode each prediction and compute the peak signal-to-noise ratio (PSNR) against the decoded four-step output over the region corresponding to each patch. We record the first step at which this score reaches 35 dB. This measures agreement with the generator's own full-step output, rather than reconstruction of the ground-truth video.

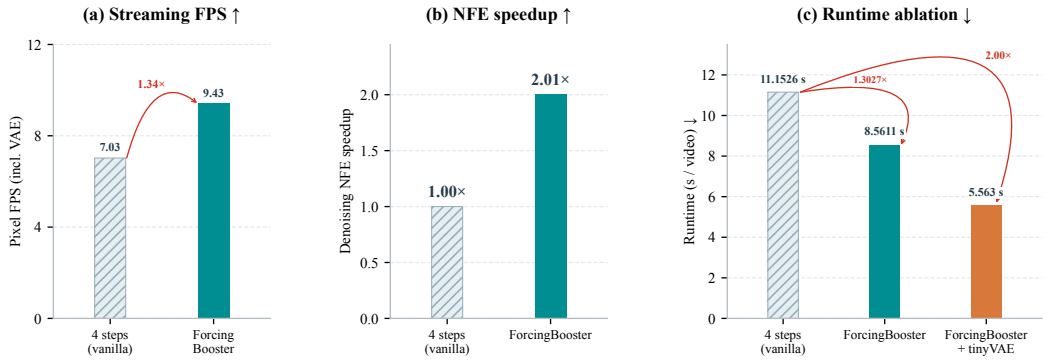

Figure 8: **Computational efficiency and practical runtime performance.** **(a)** Streaming pixel-frame throughput of the four-step baseline and ForcingBooster. **(b)** Denoising NFE speedup relative to the four-step baseline. **(c)** Full-video runtime of the four-step baseline, ForcingBooster, and ForcingBooster with tinyVAE.

As shown in Fig. 7 (right), 30.60% of patches reach the threshold after the first step. A further 12.25% and 9.69% first reach it at the second and third steps, respectively. The remaining 47.45% do not reach it within the first three steps.

Figure 7 (left) shows the final frames from separate fixed-step generation runs with the same seed. Each run uses four steps for the first chunk and the indicated step count for subsequent chunks. The sky and hillside look broadly similar across the results, whereas the enlarged hand and forearm show clear differences. With one step, the hand is poorly defined. Two and three steps recover more of its shape, but visible distortions remain around the hand and wrist compared with the four-step result.

These results show that patches within the same chunk approach the full-step output at different rates. They motivate reducing later updates at positions already close to that output while allowing other positions to continue denoising.

B.2 Computational Efficiency and Practical Runtime Performance

We evaluate the computational efficiency of ForcingBooster from three complementary views: sustained streaming throughput, denoising computation, and end-to-end video runtime. For streaming throughput, we treat text encoding and the first chunk as startup and measure the subsequent six chunks, including KV-cache operations and VAE decoding. As shown in Fig. 8, ForcingBooster increases pixel-frame throughput from 7.03 to 9.43 FPS, corresponding to a $1.34\times$ gain.

The patch-wise denoising computation is reduced by approximately half: the measured NFE speedup is $2.01\times$ relative to the four-step baseline. For complete video generation, the runtime panel reports 11.1526 seconds for vanilla four-step sampling and 8.5611 seconds for ForcingBooster, a $1.3027\times$ practical speedup. The historical tinyVAE measurement reaches 5.563 seconds, or approximately $2.00\times$ relative to the vanilla baseline. The gap between NFE reduction and end-to-end speedup reflects cache construction, decoding, and other fixed overheads.