UNSTEP: TRAINING-FREE ACCELERATION OF CAUSAL VIDEO DIFFUSION WITH FEWER STEPS THAN DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Distilling bidirectional multi-step video diffusion transformers into few-step causal models has become a common approach for streaming video generation. While these few-step students are significantly faster than the teachers they are distilled from, they remain slow for real-time generation. In this work we present UNSTEP, a training-free wrapper that accelerates few-step causal video models at inference by running them with fewer diffusion transformer (DiT) steps than during distillation and limiting the temporal window retained in the attention KV cache. We propose two inference-only mechanisms to recover quality lost by step reduction and attention windowing: renoising the generated latent frames to a near clean level and reusing the existing clean-cache pass to refine them, and applying truncated SVD to the DiT attention value and output projections. We also accelerate inference with a quality-preserving runtime stack for the DiT and VAE decoder, including more efficient attention calls and KV indexing, fused Triton RoPE with cached coefficients, and VAE decoding with optimized memory layout, precision, and convolution kernels. By reducing computation and optimizing the runtime stack, UNSTEP sets a new throughput regime for causal video diffusion, by running substantially faster than current methods, reaching 50 FPS on a single H100 without quality loss, and 77 FPS on GB200, all without retraining.

1 INTRODUCTION

Video diffusion models are now capable of generating high-quality clips, but their computational cost at inference remains a major bottleneck. Wan 2.1 (Wan Team et al., 2025) has emerged as a powerful open video foundation model capable of producing videos with significantly higher fidelity than earlier generative video models (Ho et al., 2022; Blattmann et al., 2023). Wan employs a bidirectional diffusion transformer (Peebles & Xie, 2023) that attends over the full clip, so no frame can be emitted until the entire clip has been generated. It also requires 50 denoising steps to generate the video. As a result, it is not suitable for real-time or streaming use cases.

Causal distillation addresses the limitations of multi-step bidirectional models by reducing the number of steps to only a few (e.g., four steps) and converting the bidirectional attention to causal attention. This allows frames to be emitted as they are generated, analogous to autoregressive generation in language models. CausVid (Yin et al., 2025) is a prominent example of causal distillation: it produces a fast streaming model, but with a notable quality drop relative to the teacher it was distilled from.

Self Forcing (Huang et al., 2025a) introduces a new paradigm for causal diffusion distillation. It enables a 4-step causal student to match the performance of the bidirectional 50-step Wan 2.1 teacher (Wan Team et al., 2025) by distilling the student on its own rollouts. Self Forcing has therefore become a standard distillation technique in causal diffusion, paving the way for later work to improve quality (Zhu et al., 2026a; Li et al., 2026b), extend generation to long clips (Yang et al., 2026b; Liu et al., 2026c), add interactive control (Shin et al., 2026), and make inference faster (Yuan et al., 2026; Zheng et al., 2026a).

Self Forcing streams at 17 FPS on an H100. To improve throughput, several works distill to fewer denoising steps (e.g. 1 or 2 steps) (Feng et al., 2026; Zhao et al., 2026; Zheng et al., 2026a), use

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.

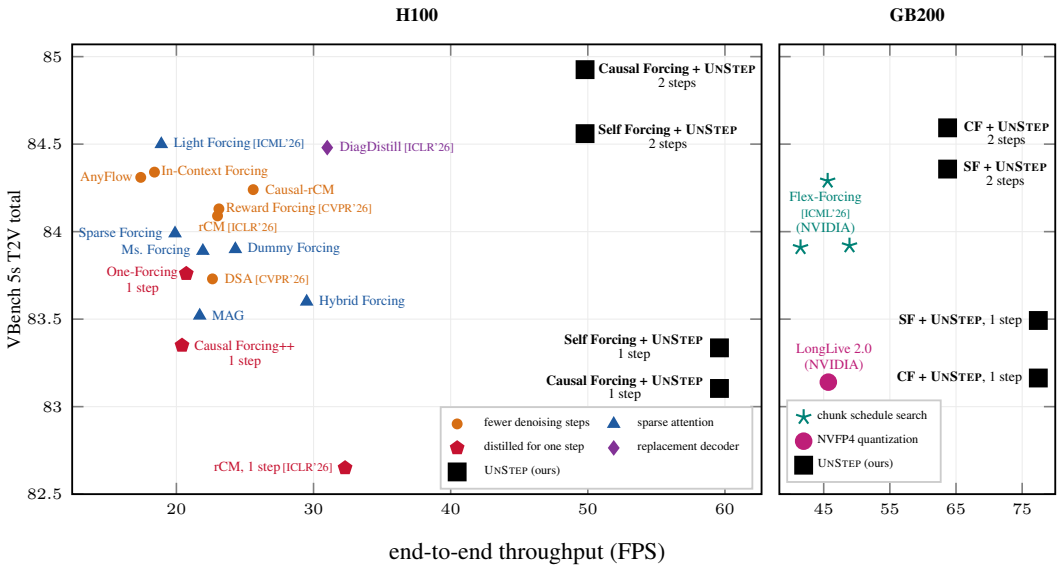


Figure 1: Speed in FPS against VBench quality in the H100 comparison (left) and on GB200 (right). In the GB200 plot, SF and CF denote Self Forcing and Causal Forcing. Appendix C.2 lists the details of every point.

a compact VAE decoder (Liu et al., 2026b; Zhu et al., 2026b), or train with limited KV attention windows (Lv et al., 2026; Xu et al., 2026). The extra speed usually comes at the cost of quality, and as these methods require training, the speed quality tradeoff is determined during distillation.

The flexibility of choosing the speed quality tradeoff freely at inference motivates a training-free approach, where the same model checkpoint can be directly used without retraining the checkpoint. Toward this goal, we present UNSTEP, a training-free wrapper that runs on the original Self Forcing checkpoint and boosts its speed from 17 FPS to 50 FPS while slightly raising its quality (Table 2). The 1-step variant of UNSTEP runs at 60 FPS with comparable quality to other models trained explicitly with 1 step, while being significantly faster (Figure 1). The choice between the 2-step (default) or 1-step variant of UNSTEP is determined only at inference, without any retraining.

UNSTEP achieves these gains by optimizing the runtime stack for both DiT and VAE, and reducing DiT computation. The runtime optimizations target KV indexing, attention calls, rotary position embeddings (RoPE), and VAE decoding (§2.1). We keep KV position indices on CPU to avoid synchronization overhead with GPU, and read token counts for FlashAttention calls from query and key tensor shapes without allocating separate GPU tensors. We also compute RoPE rotations using a fused Triton kernel together with cached sine and cosine coefficients, which reduces repeated computation and kernel launches. For VAE decoding, we use FP16 precision and a channels-last 3D memory layout, with convolution autotuning that selects fast kernels for the decoder's tensor shapes. This runtime stack preserves the denoising schedule, attention context, and overall quality while boosting throughput by about 10 FPS (Table 1).

We then reduce DiT computation by using fewer denoising steps (§2.2) and limiting the KV attention window (§2.3). Since these changes degrade quality, we propose two inference-only recovery mechanisms. We refine the clean-cache pass originally used only to update the KV cache for later generation by first renoising the current latent frames to a near clean level, then reusing the same pass to refine them (§2.4). Next, we replace the attention value and output projections by their truncated SVD (§2.5). Appendix E provides theoretical intuition for why both mechanisms can help under reduced step inference. Together, these components recover the quality lost while preserving the training-free setting and boosting FPS by a further 18 FPS (Table 1). Figure 3 summarizes the complete UNSTEP pipeline.

We take the standard VBench 5s (5-second clips) T2V (text-to-video) suite (Huang et al., 2024) as the main quality benchmark. To our knowledge, UNSTEP is the first method to enable Wan2.1-1.3B to stream close to 50 FPS on an H100 while maintaining competitive VBench performance (Figure



Figure 2: Sample videos generated by Self Forcing and Causal Forcing, with and without UNSTEP, for an otter surfing (top) and a ferret playing ball with a puppy (bottom). More examples are in Appendix G.

1—left), and it does so without retraining. The fastest existing methods Liu et al. (2026b); Li et al. (2026c) stream at around 30 FPS despite relying on training with smaller decoders, shortened denoising schedules, or sparse attention. More strikingly, on a GB200, UNSTEP is over 30 FPS faster than NVIDIA's recent LongLive 2.0 (Chen et al., 2026) with slightly better VBench performance (Figure 1—right). This is notable as LongLive 2.0 is specifically optimized for GB200 with NVFP4 quantization. UNSTEP achieves this unusually large speed jump by targeting bottlenecks in the surrounding runtime stack in addition to the DiT computation reductions, while remaining entirely inference-only.

We design and tune UNSTEP's parameters in a single setup: Self Forcing as the causal model, H100 as the GPU accelerator, and VBench 5s T2V as the quality benchmark. We then fix these parameters and design choices and apply UNSTEP as is, in a zero-shot setting, across several different setups: different checkpoints, including Causal Forcing (Zhu et al., 2026a) and LongLive 1.0 (Yang et al., 2026b) (§3.1), different GPU, GB200 (§3.3), a 1-denoising-step variant (§3.4), image-to-video generation on VBench I2V (Huang et al., 2025b) (§3.5), and 30-second long clips on VBench Long (Huang et al., 2025b) (§3.6). Across all setups, the central observation remains the same: UNSTEP preserves the quality of the original checkpoint, while substantially improving its speed.

To summarize, our key contributions are:

- We introduce UNSTEP, a training-free inference wrapper that accelerates few-step causal video diffusion models built on Wan 2.1 (§2), and enables streaming at 50 FPS on H100 with strong VBench performance, and up to 77 FPS on GB200 (Figure 1).
- We reduce runtime stack overheads with more efficient attention calls and KV indexing, fused Triton RoPE with cached coefficients, and FP16 VAE decoding with a convolution friendly memory layout and convolution autotuning (§2.1).
- We reduce computation by using fewer DiT denoising steps (§2.2) and a limited temporal KV attention window (§2.3), and recover the resulting quality loss with a refined clean-cache pass (§2.4) and truncated SVD of the attention value/output projections (§2.5).
- We show that UNSTEP allows the speed quality tradeoff to be chosen flexibly at inference from the same checkpoint (§3.4) and generalizes across checkpoints (§3.1), GPUs (§3.3), image-to-video generation (§3.5), and long video generation (§3.6).
- Finally, we develop a toy mathematical model that provides intuition for why clean-cache refinement and spectral truncation can help recover quality under reduced step inference (Appendix E).

Appendix D discusses further related work.

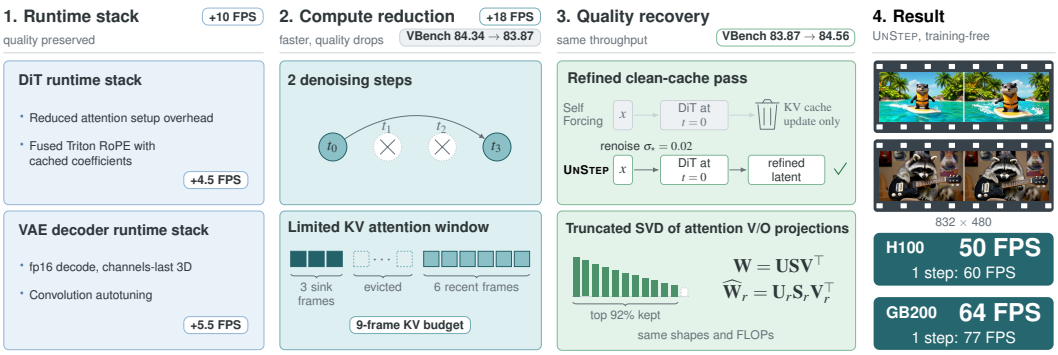


Figure 3: Overview of UNSTEP. First column shows the kernel and runtime optimizations that accelerate the DiT and VAE without degrading quality. Second column shows the reduced DiT computation that further increases speed but reduces quality. Third column shows the quality recovery mechanisms, which add no runtime. All changes are inference-only, with no retraining.

2 UNSTEP

As shown in Figure 3, UNSTEP accelerates few-step causal video diffusion entirely at inference. We first describe the optimized runtime stack for the DiT and VAE decoder, then show how UNSTEP reduces DiT computation through fewer denoising steps and a limited temporal KV attention window. Finally, we describe the two inference-only mechanisms used to recover the quality lost by these speedups: the refined clean-cache pass and V/O truncated SVD. Appendix E gives a toy model intuition for why these two recovery mechanisms can help under reduced step inference. Appendix A gives the exact implementation configuration and reports additional ablations.

Table 1 summarizes the cumulative ablation of UNSTEP. C0 starts from original Self Forcing with FlashAttention-3 (Shah et al., 2024) and applies standard `torch.compile` Ansel et al. (2024) to the DiT and VAE decoder to optimize execution for the fixed tensor shapes. Each subsequent row adds one component of UNSTEP and reports its effect on end-to-end throughput on H100 and VBench 5s T2V quality. The rest of this section follows the rows of Table 1, describing the runtime stack, the generation trajectory changes, and the two quality recovery mechanisms.

2.1 THE RUNTIME STACK

The computational cost of operations such as handling KV cache positions, preparing and applying rotary position embeddings (RoPE), setting up attention sequence lengths, and decoding latents with the VAE are usually hidden in multi-step diffusion models, as repeated DiT forward passes dominate runtime. In few-step causal models, these computational costs become visible in end-to-end throughput. In this subsection we target these operations, and show that making them more efficient can significantly improve throughput.

DiT runtime stack: Before each self-attention call, Self Forcing uses cache position indices to select where to write the current keys and values and which cached range to pass to attention. These are stored on GPU, so Python reads require CPU–GPU synchronization. These indices depend on token positions and counts, not on the computed keys or values. UNSTEP therefore maintains them on CPU, which eliminates the CPU–GPU communication overhead. Self Forcing also uses variable-length FlashAttention, which supports different sequence lengths within a batch but requires extra GPU tensors recording their lengths and boundaries, which are rebuilt on every attention call. For single video inference, UNSTEP uses fixed-length FlashAttention, which reads token counts from the query and key tensor dimensions. This avoids repeated GPU tensor allocations and the operations used to construct them. Together with the runtime changes detailed in Appendix A.5, these changes add 2.3 FPS, reaching 24.5 FPS.

UNSTEP next replaces the RoPE application path with a fused Triton kernel (Appendix A.5). RoPE encodes token positions by rotating query and key coordinate pairs. Original Self Forcing computes these rotations through separate PyTorch operations that create temporary tensors and launch multiple GPU kernels. UNSTEP computes the same rotations in one kernel using `float32` instead of

Table 1: The components of UNSTEP applied cumulatively. Speed is measured on H100 end-to-end for both DiT and VAE, and quality on the VBench 5s T2V suite. Base model is Wan2.1-1.3B distilled to 4 steps with Self Forcing with resolution 832×480. FA3 stands for FlashAttention 3.

		Configuration	FPS	Total	Quality	Semantic	§
Original	C0	Self Forcing + FA3	17.0	84.31	85.28	80.44	2
		+ compilation	22.2	84.28	85.26	80.35	
Runtime stack	C1	C0 + transformer runtime stack	26.7	84.34	85.24	80.72	2.1
	C2	C1 + decoder runtime stack	32.2	84.34	85.26	80.68	2.1
Speedup	C3	C2 + two denoising steps	44.3	83.91	84.68	80.82	2.2
	C4	C3 + limited attention window	**49.8**	83.87	84.67	80.71	2.3
Quality recovery	C5	C4 + clean-cache pass	**49.8**	84.33	84.91	81.99	2.4
	C6	C5 + V/O truncated SVD	**49.8**	**84.56**	85.20	82.02	2.5

`float64`, avoiding intermediate writes and repeated launches. Original Self Forcing caches the base RoPE table, but builds the coefficients for each chunk's token positions on every call. UNSTEP caches these per chunk sine and cosine tensors by chunk shape and starting latent frame index, across layers and denoising steps. The fused RoPE kernel with cached coefficients adds another 2.2 FPS, reaching 26.7 FPS, as shown in C1 in Table 1.

VAE decoder: We next optimize the VAE decoder while keeping the same VAE weights and causal temporal behavior. Original Self Forcing runs the VAE in `bfloat16` and leaves its convolution weights in PyTorch's default memory layout. UNSTEP casts both the decoder weights and latent inputs to `fp16`, and uses a channels-last 3D memory layout for convolution weights and decoder inputs. This places the channel values at each spatial and temporal position next to one another in memory. The precision and layout are chosen together to use faster convolution kernels on H100. Together with output preallocation and the decoding changes detailed in Appendix A.6, these changes add 2.8 FPS, reaching 29.5 FPS.

Next, UNSTEP utilizes convolution autotuning to select an efficient GPU kernel. It divides the computation across GPU threads and access memory differently, so their relative speed depends on input shape, precision, and memory layout. During warmup, it measures the execution time of these alternatives and keeps the fastest tested kernel for subsequent calls with the same configuration. This adds another 2.7 FPS, reaching 32.2 FPS, which corresponds to C2 in Table 1.

2.2 RUNNING BELOW THE DISTILLED STEP COUNT

We next run the model with a shorter inference schedule than the one it was distilled with. Self Forcing is distilled with four DiT steps per chunk. Following common practice in reduced step causal video models (Zhao et al., 2026; Feng et al., 2026), we keep the first chunk at the original four steps and reduce later chunks to two denoising steps. The first chunk remains unchanged to provide a high-quality initial anchor for the later chunks to reuse its causal context. As shown in C3 of Table 1, this gives the largest single speedup, 12 FPS, but also causes the main quality drop, reducing VBench total from 84.34 to 83.91. We also evaluate a faster one-step variant, which uses one denoising step for later chunks instead of two (§3.4).

2.3 LIMITED KV ATTENTION WINDOW

Causal models cache past keys and values so that each new chunk can attend to earlier generated chunks without recomputation. In original Self Forcing, this cache grows with the video length, so the attention cost increases as generation continues. UNSTEP limits this attention cost with a sink and a sliding window cache. Following the same idea as StreamingLLM (Xiao et al., 2024) and its use in video diffusion by LongLive 1.0 (Yang et al., 2026b), we retain the first 3 latent frames as sink tokens and the 6 most recent frames as a sliding window, for a total KV budget of 9 latent frames, evicting older middle tokens from the cache.

This reduces the attention context, increasing throughput by 5.5 FPS as seen in C4 of Table 1. Since the cache size no longer grows with sequence length, the same mechanism also supports

long streaming generation (§3.6). This is the last speed oriented change, the remaining components recover the quality lost from using fewer steps and limited attention.

2.4 Refining the clean-cache pass

We next introduce the first inference-only quality recovery mechanism. In flow matching diffusion, the DiT predicts a velocity field. Given the current estimate $\mathbf{x}_i$, a sampler step first places it at the desired noise level, evaluates the DiT at the corresponding timestep label, and converts the predicted velocity into the next estimate as follows:

$$\tilde{\mathbf{x}}_i = (1 - \sigma_i)\mathbf{x}_i + \sigma_i \boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{1}$$

$$\mathbf{v}_i = f_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_i, t_i), \tag{2}$$

$$\mathbf{x}_{i+1} = \tilde{\mathbf{x}}_i - \sigma_i \mathbf{v}_i. \tag{3}$$

Here $f_{\boldsymbol{\theta}}$ is the DiT, t_i is the scalar conditioning label given to the DiT, and σ_i is the scalar noise level corresponding to that label by the scheduler. The same σ_i controls both the amount of noise injected before the DiT call and the scale of the velocity subtraction after it.

Self Forcing already runs one extra DiT evaluation after each chunk at the clean label $t = 0$ (equation (2)) to update keys and values for later chunks. In the original sampler, this clean-cache pass uses $\sigma = 0$, so equations (1) and (3) are identities: the input is unchanged, the DiT output is discarded, and the pass only updates the KV cache. UnStep keeps the DiT conditioning label at $t = 0$, but uses a small positive noise level $\sigma_\star = 0.02$ in equations (1) and (3). This keeps the cache update close to the clean label behavior used by Self Forcing, while allowing the same already run forward pass to refine the current chunk and produce the emitted latent. In C5 of Table 1, this refinement keeps throughput unchanged and recovers VBench total from 83.87 to 84.33. Appendix E.1 gives a toy model explanation for why a nonzero clean-cache noise level can help after step reduction.

2.5 Truncated SVD of the value and output projections

We finally add a second inference-only quality improving mechanism. Although the clean-cache pass already recovers the original Self Forcing quality, a small spectral truncation of the DiT attention projections further improves quality. We replace the value and output projections by a truncated SVD of the same matrix. For a projection matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ with $m \leq n$, let

$$\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \qquad \mathbf{S} = \mathrm{diag}(\lambda_1, \ldots, \lambda_m), \quad \lambda_1 \geq \cdots \geq \lambda_m \geq 0. \tag{4}$$

We keep the leading r singular directions and use

$$\widehat{\mathbf{W}}_r = \mathbf{U}_r \mathbf{S}_r \mathbf{V}_r^\top. \tag{5}$$

Here $\mathbf{U}_r$, $\mathbf{S}_r$, and $\mathbf{V}_r$ contain only the singular vectors and singular values corresponding to the r largest singular values. We set $r = 0.92m$, i.e., we retain 92% of the singular directions. We form $\widehat{\mathbf{W}}_r$ once when the wrapper is initialized and store it as a dense matrix with the same shape as $\mathbf{W}$. Therefore, the resulting projection has the same parameter count and multiplication cost as before, only the spectrum of the matrix changes. This differs from post-training compression methods that apply SVD truncation to reduce parameters or FLOPs (Lin et al., 2025a; Chiang et al., 2026). Our motivation is instead closer to LASER: low rank truncation can act as a spectral filter that improves a trained transformer by removing weak directions (Sharma et al., 2024).

We apply this spectral edit to the value and output projections, leaving the query and key projections unchanged. For fixed attention inputs and cached keys, this changes the layer's output without changing its attention probabilities. In C6 of Table 1, V/O truncated SVD improves VBench total from 84.33 to 84.56 with no throughput loss. Appendix E.2 gives a toy model explanation for why truncating these projections can remove error from the shortened trajectory.

3 Experiments

Our main evaluation uses Self Forcing (Huang et al., 2025a) on Wan2.1-1.3B (Wan Team et al., 2025): text-to-video generation at resolution 832×480 for five-second clips, batch size one, on a

single H100. Quality is measured with the VBench 5s T2V suite (Huang et al., 2024) using all 946 prompts, which is the standard benchmark for evaluating causal video models. Appendix B provides further details on VBench. Throughout the paper, the reported UNSTEP scores, are averages over five runs, with statistics for the default wrapper in Appendix C.1.

We measure throughput as generated video frames per second (FPS) at batch size one. This includes DiT denoising, KV cache operations, VAE decoding, and GPU postprocessing needed to produce the video tensor. It excludes model loading, compilation, GPU warmup, VBench scoring, and saving videos to disk. Appendix B gives the benchmark, scoring, timing, and reproducibility details.

All UNSTEP parameters are chosen based on the ablations in Table 1 with Self Forcing as the base checkpoint, a single H100 as the target GPU, and VBench 5s T2V as the quality benchmark. These parameters include the denoising schedule, attention window and sink sizes, the near clean noise level used in the clean-cache pass, SVD retention, and runtime stack choices. The remaining experiments apply the same wrapper without retuning to other checkpoints, GB200, one-step generation, image-to-video generation, and long video generation.

The experiments show a consistent pattern: UNSTEP significantly accelerates causal video models built on Wan 2.1 while keeping quality close to, and often above, the unwrapped checkpoint. Figure 2 shows sample outputs with and without the wrapper. Appendix G provides further comparisons.

3.1 OTHER BASE MODELS

We now test UNSTEP on checkpoints beyond Self Forcing. We consider two prominent causal models built on the same Wan2.1 backbone. Causal Forcing (Zhu et al., 2026a) changes the distillation recipe. Instead of initializing the causal student directly from a bidirectional teacher, it first uses an autoregressive teacher for ODE initialization and then applies distribution matching distillation. NVIDIA's LongLive 1.0 (Yang et al., 2026b) targets interactive streaming and long video generation. These checkpoints therefore test UNSTEP on strong causal models that differ from Self Forcing in how they are trained or the setting they target.

Table 2 shows the results. All three wrapped checkpoints reach similar throughput, around 50 FPS, since they all share the same wrapper configuration and backbone Wan 2.1. VBench quality is preserved in all 3 cases, while the total score is slightly higher than the unwrapped checkpoint. This shows that UNSTEP transfers to other causal checkpoints built on Wan 2.1.

Table 2: UNSTEP applied unchanged to three causal checkpoints on H100. Performance is measured on VBench 5s T2V with resolution 832×480. Appendix C.3 and Appendix Table 10 provide more details.

System	FPS	Total	Quality	Semantic
Self Forcing (Huang et al., 2025a)	17.0	84.31	85.28	80.44
+ UNSTEP	**49.8**	84.56	85.20	82.02
Causal Forcing (Zhu et al., 2026a)	17.0	84.82	85.64	81.55
+ UNSTEP	**49.8**	84.93	85.79	81.46
LongLive 1.0 (Yang et al., 2026b)	17.0	83.31	83.64	81.98
+ UNSTEP	**49.8**	83.42	83.82	81.81

3.2 METHODS THAT TARGET SPEED

The left panel of Figure 1 compares UNSTEP to recent causal video methods that explicitly target faster inference and report throughput and VBench scores. We group the methods based on the mechanism they use for the speedup: reducing the denoising schedule or changing the causal sampling procedure (Zhao et al., 2026; Gu et al., 2026; Yang et al., 2026a; Le et al., 2026; Zheng et al., 2026b; Lu et al., 2026; Zheng et al., 2026a), reducing attention or KV cache cost (Lv et al., 2026; Xu et al., 2026; Zhu et al., 2025; Li et al., 2026e;c; Guo et al., 2026), and replacing or distilling the VAE decoder (Liu et al., 2026b). Training explicitly for one-step generation is discussed in §3.4.

We report the results form the original papers. Appendix Table 7 provides more details on the points in Figure 1.

The main observation from Figure 1 is that recent methods that target speed cluster around the 20–30 FPS range on H100, whereas UNSTEP reaches close to 50 FPS at comparable or higher VBench quality. This is an unusually large jump in a regime where many methods already use few denoising steps or restricted attention. The key difference is that UNSTEP also targets the runtime stack overheads that become visible once the DiT already runs for only a few steps: KV indexing and attention calls, RoPE computation, GPU kernel overheads, and VAE decoding as explained in §2.1. By optimizing these operations together with step reduction and limited KV attention, UNSTEP becomes a substantially faster end-to-end video accelerator without any training.

3.3 BLACKWELL GPU

So far, we have used H100 GPUs, following most recent causal video generation work. Two recent NVIDIA papers, LongLive 2.0 (Chen et al., 2026) and Flex-Forcing (Ma et al., 2026a), are instead built around NVIDIA's newer Blackwell generation, GB200. GB200 provides faster tensor core and memory system execution for the matrix multiplications, attention kernels, and convolutions used in video generation. LongLive 2.0 accelerates inference by quantizing weights, activations, and the KV cache to NVFP4, a 4-bit floating-point format designed for Blackwell GPUs. Flex-Forcing searches over handcrafted chunk schedules and sizes and reports the best GB200 configuration for the 5-second VBench benchmark.

We evaluate UNSTEP unchanged on GB200, using both the 1-step and 2-step variants on the Self Forcing and Causal Forcing checkpoints. Figure 1 compares these results to the fast versions of LongLive 2.0 and Flex-Forcing. Without retuning for GB200, the 1-step UNSTEP variant is more than 30 FPS faster than LongLive 2.0 while also achieving higher VBench quality. The 2-step variant is substantially faster than the best Flex-Forcing configuration and also scores higher. Thus, even though UNSTEP is designed and tuned on H100, it remains faster than these GB200 optimized models with slightly better VBench quality. Appendix Table 9 lists the exact GB200 values.

3.4 ONE DENOISING STEP

Several recent video models target the low-latency setting of one denoising step per generated chunk (Feng et al., 2026; Zhao et al., 2026; Zheng et al., 2026b). These models correspond to the three red points in Figure 1. They are trained specifically for the 1-step regime, so the latency quality tradeoff is determined during distillation. UNSTEP is tuned for the default 2-step setting, but the number of denoising steps remains an inference time choice. Appendix Table 8 gives the score and throughput sources for all one-step points.

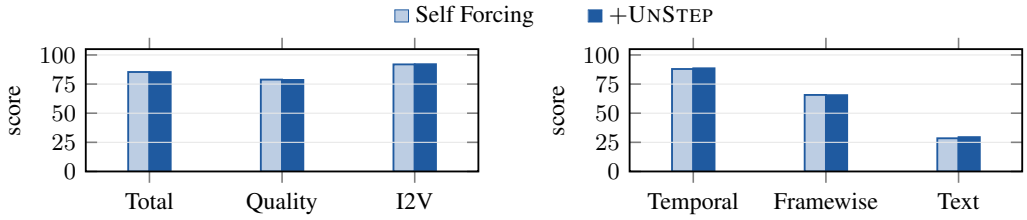
Figure 1 shows that the default 2-step UNSTEP variant is already faster and higher quality than the reported 1-step baselines on H100. We also run a 1-step UNSTEP variant to show that the same checkpoint can become even faster when desired. This variant reaches 59.6 FPS, trading quality relative to the 2-step wrapper for additional throughput.

A key point is flexibility. With UNSTEP, the same checkpoint can be run with 2 steps for higher quality, or with 1 step for higher throughput, whereas a model distilled directly for one step fixes this choice in the weights, and must be redistilled for deploying in another setup.

3.5 IMAGE-TO-VIDEO

We next test UNSTEP on image-to-video (I2V) tasks. In I2V, the model is conditioned on an input image in addition to the text prompt, so acceleration must preserve both generated video quality and fidelity to the conditioning image. We evaluate with VBench-I2V (Huang et al., 2025b), which reports Total, Quality, and I2V aggregates. Appendix B.2 details the I2V protocol.

We run the original Self Forcing checkpoint with and without UNSTEP, without I2V-specific retuning. Original Self Forcing runs at 17.0 FPS, while UNSTEP reaches 49.8 FPS. The quality results are shown in Figure 4a. UNSTEP preserves video quality and conditioning fidelity while boosting throughput. Appendix C.5 reports all nine component dimensions in Figure 6.



(a) VBench-I2V aggregates

(b) VBench-Long aggregates

Figure 4: Aggregate image-to-video results for 5-second clips (left) and long video results for 30-second clips (right) on a single H100. Pale bars are Self Forcing and solid bars are the same checkpoint wrapped with UnStep. Appendix C.5 reports the full dimension scores.

3.6 Long video

All previous experiments use five-second clips, matching the standard VBench T2V setting. We next evaluate 30-second streaming generation. We evaluate on VBench-Long (Huang et al., 2025b), which reports temporal quality, framewise quality, and text alignment aggregates instead of the standard sixteen dimension VBench total. Figure 4b compares these aggregates. Appendix B.3 details the protocol, and Appendix C.5 reports all eight component dimensions in Figure 7.

UnStep is again applied on Self Forcing without retuning. It reaches 51.2 FPS, close to its steady-state streaming speed while performance is maintained. Compared with five-second generation, UnStep is faster because a longer clip contains more later chunks with two denoising steps, so the initial chunk's four-step cost is spread over more generated frames. Self Forcing is slightly slower at 16.6 FPS because a larger fraction of the video is generated with its KV cache at the full five-second capacity. This shows that UnStep transfers to long streaming generation without specific retuning.

4 Limitations

Single setup tuning: All UnStep parameters are tuned in one setting: Self Forcing as the base checkpoint, text-to-video generation, five-second VBench clips, and a single H100. We then keep the same denoising schedule, attention window and sink sizes, clean-cache noise level, SVD retention, and runtime stack choices fixed for all other experiments. This shows that the wrapper can transfer without retuning, but it also means the reported results are not necessarily optimal for every checkpoint, GPU, task, or clip length. Task-specific tuning could improve speed, quality, or both, and we leave that to future work.

Training-free constraint: Throughout the paper we maintain the training-free constraint and avoid applying gradient updates. This constraint makes the wrapper easy to apply to existing checkpoints, but it also limits the extent of acceleration we can reach. More aggressive sparse attention patterns, shorter schedules, quantization, or a smaller decoder could further improve speed, especially if followed by finetuning or distillation to recover quality. We view those training-based extensions as future work complementary to UnStep, but outside the scope of this paper.

5 Conclusion

We presented UnStep, a training-free inference wrapper for few-step causal video diffusion models. UnStep accelerates generation by combining runtime stack optimizations for the DiT and VAE decoder with reduced denoising steps and a limited KV attention window. It recovers the resulting quality loss using a refined clean-cache pass and V/O truncated SVD, two mechanisms that we motivate theoretically using toy models. It enables streaming at 50 FPS on H100 with strong VBench quality, and up to 77 FPS on GB200. The same fixed wrapper transfers across checkpoints and to image-to-video generation and long video streaming, while also supporting a faster one-step configuration. UnStep delivers substantially faster causal video generation while maintaining strong quality, making existing models more practical for real-time applications without retraining.

Reproducibility statement

Our experiments use publicly available Wan-2.1-1.3B distilled checkpoints such as Self Forcing (Huang et al., 2025a) and Causal Forcing (Zhu et al., 2026a), which can be downloaded through their official project repositories. The benchmark data and evaluation code for VBench text-to-video, image-to-video, and long video evaluation are available through the VBench GitHub repository. Appendix A specifies our model and inference configurations, and Appendix B describes the evaluation, timing, and reproducibility protocols. Appendix F provides the assumptions and proofs for our theoretical results. The UNSTEP code, together with all reproduction instructions and generated sample videos, will be available in our GitHub repository and our project page.

AI use statement

We used generative AI tools to aid and polish writing. This included grammar corrections, clearer phrasing, and final polishing, which we reviewed against the intended meaning. We used them to draft sections of the paper as well. We revised the drafts and checked descriptions of the methods and results against our implementation and experimental records. We also used generative AI tools to assist in proving mathematical claims, which we reviewed step by step against the assumptions and provided complete proofs for them. We have reviewed all AI-assisted work and take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Krish Agarwal, Zhuoming Chen, Cheng Luo, Yongqi Chen, Haizhong Zheng, Xun Huang, et al. Monarchrt: Efficient attention for real-time video generation. *arXiv:2602.12271*, 2026. URL https://arxiv.org/abs/2602.12271.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Hmrishav Bandyopadhyay, Nikhil Pinnaparaju, Rahim Entezari, Jim Scott, Yi-Zhe Song, and Varun Jampani. Block cascading: Training free acceleration of block-causal video models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Findings (CVPR)*, 2026.

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Boyuan Chen, Diego Marti Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yukang Chen, Luozhou Wang, Wei Huang, Shuai Yang, Bohan Zhang, Yicheng Xiao, et al. Longlive-2.0: An nvfp4 parallel infrastructure for long video generation. *arXiv:2605.18739*, 2026. URL https://arxiv.org/abs/2605.18739.

Zhiyuan Chen, Keyi Li, Yifan Jia, Le Ye, and Yufei Ma. Accelerating diffusion transformer via increment-calibrated caching with channel-aware singular value decomposition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Hung-Yueh Chiang, Chi-Chih Chang, Yu-Chen Lu, Chien-Yu Lin, Kai-Chiang Wu, Mohamed S. Abdelfattah, and Diana Marculescu. UniQL: Unified Quantization and Low-rank Compression for Adaptive Edge LLMs. In *International Conference on Learning Representations (ICLR)*, 2026.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Haipeng Fang, Yu Li, Fan Tang, Yixing Lu, Juan Cao, and Sheng Tang. ResCa: Residual caching for diffusion transformers acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

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

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations (ICLR)*, 2025.

Xingtong Ge, Yi Zhang, Yushi Huang, Dailan He, Xiahong Wang, Bingqi Ma, Guanglu Song, Yu Liu, and Jun Zhang. Salt: Self-consistent distribution matching with cache-aware training for fast video generation. In *European Conference on Computer Vision (ECCV)*, 2026.

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. *arXiv:2605.13724*, 2026. URL https://arxiv.org/abs/2605.13724.

Hang Guo, Zhaoyang Jia, Jiahao Li, Bin Li, Yuanhao Cai, Jiangshan Wang, Yawei Li, and Yan Lu. Efficient autoregressive video diffusion with dummy head. *arXiv:2601.20499*, 2026. URL https://arxiv.org/abs/2601.20499.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Vbench: Comprehensive benchmark suite for video generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Ziqi Huang, Fan Zhang, Xiaojie Xu, Yinan He, Jiashuo Yu, Ziyue Dong, Qianli Ma, Nattapol Chanpaisit, Chenyang Si, Yuming Jiang, Yaohui Wang, Xinyuan Chen, Ying-Cong Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench++: Comprehensive and versatile benchmark suite for video generative models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2025b.

Yicheng Ji, Zhizhou Zhong, Jun Zhang, Qin Yang, XiTai Jin, Ying Qin, Wenhan Luo, Shuiyang Mao, Wei Liu, and Huan Li. Forcing-kv: Hybrid kv cache compression for efficient autoregressive video diffusion models. *arXiv:2605.09681*, 2026. URL https://arxiv.org/abs/2605.09681.

Animesh Karnewar, Denis Korzhenkov, Ioannis Lelekas, Adil Karjauv, Noor Fathima, Hanwen Xiong, Vancheeswaran Vaidyanathan, Will Zeng, Rafael Esteves, Tushar Singhal, Fatih Porikli, Mohsen Ghafoorian, and Amirhossein Habibian. Neodragon: Mobile video generation using diffusion transformer. In *International Conference on Learning Representations (ICLR)*, 2026.

Akio Kodaira, Tingbo Hou, Ji Hou, Markos Georgopoulos, Felix Juefei-Xu, Masayoshi Tomizuka, et al. Streamdit: Real-time streaming text-to-video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Denis Korzhenkov, Adil Karjauv, Animesh Karnewar, Mohsen Ghafoorian, and Amirhossein Habibian. Pyramidalwan: On making pretrained video model pyramidal for efficient inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Thanh-Tung Le, Yunhan Zhao, Menglei Chai, Zhengyang Shen, Zhe Cao, Danhang Tang, et al. Dsa: Dynamic step allocation for fast autoregressive video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Findings (CVPR)*, 2026.

Haobo Li, Yanhong Zeng, Yunhong Lu, Jiapeng Zhu, Hao Ouyang, Qiuyu Wang, et al. Aad-1: Asymmetric adversarial distillation for one-step autoregressive video generation. In *International Conference on Machine Learning (ICML)*, 2026a.

Jiaxing Li, Kai Zou, Cindy Zhou, Kaichen Huang, Junyao Gao, Zile Wang, et al. Distillalign: Coordinating mode covering and mode seeking in autoregressive video distillation. *arXiv:2607.26811*, 2026b. URL https://arxiv.org/abs/2607.26811.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank component for 4-bit diffusion models. In *International Conference on Learning Representations (ICLR)*, 2025.

Ruibin Li, Tao Yang, Fangzhou Ai, Tianhe Wu, Shilei Wen, Bingyue Peng, and Lei Zhang. Long-horizon streaming video generation via hybrid attention with decoupled distillation. *arXiv:2604.10103*, 2026c. URL https://arxiv.org/abs/2604.10103.

Xingyang Li, Samuel Tesfai, Zhekai Zhang, Haocheng Xi, Shuo Yang, Lvmin Zhang, et al. DeltaQuant: 4-bit video diffusion models with spatiotemporal delta smoothing. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026d.

Zekun Li, Xiaoyan Cong, Hongyu Li, Zhiyang Dou, Chuan Guo, Abhay Mittal, et al. Ms. Forcing: Efficient Streaming Video Generation with Multi-Scale Patchification and Attention. *arXiv:2607.20940*, 2026e. URL https://arxiv.org/abs/2607.20940.

Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. MoDeGPT: Modular Decomposition for Large Language Model Compression. In *International Conference on Learning Representations (ICLR)*, 2025a.

Shanchuan Lin, Anran Wang, and Xiao Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv:2402.13929*, 2024. URL https://arxiv.org/abs/2402.13929.

Shanchuan Lin, Xin Xia, Yuxi Ren, Ceyuan Yang, Xuefeng Xiao, and Lu Jiang. Diffusion adversarial post-training for one-step video generation. In *International Conference on Machine Learning (ICML)*, 2025b.

Shanchuan Lin, Ceyuan Yang, Hao He, Jianwen Jiang, Yuxi Ren, Xin Xia, et al. Autoregressive adversarial post-training for real-time interactive video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025c.

Shangran Lin, Lu Lu, Jian Chen, and Qiang Liu. RAPID: Reusing attention sparsity with inter-step adaptation for efficient video diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Enhuai Liu, Yunke Wang, Changming Sun, and Chang Xu. D2Cache: Second-order delta caching for higher video diffusion acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026a.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Jinxiu Liu, Xuanming Liu, Kangfu Mei, Yandong Wen, Ming-Hsuan Yang, and Weiyang Liu. Streaming autoregressive video generation via diagonal distillation. In *International Conference on Learning Representations (ICLR)*, 2026b.

Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling forcing: Autoregressive long video diffusion in real time. In *International Conference on Learning Representations (ICLR)*, 2026c.

Cheng Lu and Yang Song. Simplifying, stabilizing and scaling continuous-time consistency models. In *International Conference on Learning Representations (ICLR)*, 2025.

Yunhong Lu, Yanhong Zeng, Haobo Li, Hao Ouyang, Qiuyu Wang, Ka Leong Cheng, et al. Reward forcing: Efficient streaming video generation with rewarded distribution matching distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Chengtao Lv, Yumeng Shi, Yushi Huang, Ruihao Gong, Shen Ren, and Wenya Wang. Light forcing: Accelerating autoregressive video diffusion via sparse attention. In *International Conference on Machine Learning (ICML)*, 2026.

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. In *International Conference on Learning Representations (ICLR)*, 2025.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Xinyin Ma, Julius Berner, Chao Liu, Arash Vahdat, Weili Nie, and Xinchao Wang. Flex-forcing: Towards a unified autoregressive and bidirectional video diffusion model. In *International Conference on Machine Learning (ICML)*, 2026a.

Yuexiao Ma, Xuzhe Zheng, Jing Xu, Xiwei Xu, Feng Ling, Xiawu Zheng, et al. Flow caching for autoregressive video generation. In *International Conference on Learning Representations (ICLR)*, 2026b.

Kepan Nan, Wangbo Zhao, Penghao Zhou, Jun Li, Zhenheng Yang, Jian Yang, and Ying Tai. Accelerating autoregressive video diffusion via history-guided cache and residual correction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Weili Nie, Julius Berner, Nanye Ma, Chao Liu, Saining Xie, and Arash Vahdat. Transition matching distillation for fast video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Arani Roy, Shristi Das Biswas, and Kaushik Roy. SlimDiff: Training-Free, Activation-Guided Hands-free Slimming of Diffusion Models. *arXiv:2509.21498*, 2025. URL https://arxiv.org/abs/2509.21498.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations (ICLR)*, 2022.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision (ECCV)*, 2024.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Pratyusha Sharma, Jordan T. Ash, and Dipendra Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. In *International Conference on Learning Representations (ICLR)*, 2024.

Jinliang Shen, Lianghao Su, Zheming Li, Kang He, ZiLiang Lai, Yanbing Jiang, and Chengru Song. Headcast: Casting attention heads for efficient autoregressive video generation. *arXiv:2607.20125*, 2026. URL https://arxiv.org/abs/2607.20125.

Shuwei Shi, Zhen Li, Muyao Niu, Chuanhao Li, Bo Zheng, Kaipeng Zhang, et al. Surprise forcing: What to remember, when to skip in long video generation. In *European Conference on Computer Vision (ECCV)*, 2026.

Joonghyuk Shin, Zhengqi Li, Richard Zhang, Jun-Yan Zhu, Jaesik Park, Eli Shechtman, and Xun Huang. Motionstream: Real-time video generation with interactive motion controls. In *International Conference on Learning Representations (ICLR)*, 2026.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning (ICML)*, 2023.

Wan Team, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv:2503.20314*, 2025. URL https://arxiv.org/abs/2503.20314.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations (ICLR)*, 2025.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, Jianfei Chen, Ion Stoica, Kurt Keutzer, and Song Han. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. In *International Conference on Machine Learning (ICML)*, 2025.

Yifei Xia, Suhan Ling, Fangcheng Fu, Yujie Wang, Huixia Li, Xuefeng Xiao, and Bin Cui. Training-free and adaptive sparse attention for efficient long video generation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

Boxun Xu, Yuming Du, Zichang Liu, Siyu Yang, Ziyang Jiang, Siqi Yan, Rajasi Saha, Albert Pumarola, Wenchen Wang, and Peng Li. Sparse forcing: Native trainable sparse attention for real-time autoregressive diffusion video generation. *arXiv:2604.21221*, 2026. URL https://arxiv.org/abs/2604.21221.

Lingxiao Yang, Liu Liu, Moran Li, Han Feng, Wenjian Cao, Jiangning Zhang, and Ye Shi. In-Context Forcing: Uncovering Context Effects in Autoregressive Video Diffusion. *arXiv:2608.05237*, 2026a. URL https://arxiv.org/abs/2608.05237.

Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, et al. Longlive: Real-time interactive long video generation. In *International Conference on Learning Representations (ICLR)*, 2026b.

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, Jianfei Chen, Song Han, Kurt Keutzer, and Ion Stoica. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yongqi Yang, Huayang Huang, Xu Peng, Xiaobin Hu, Donghao Luo, Jiangning Zhang, et al. Towards One-step Causal Video Generation via Adversarial Self-Distillation. In *International Conference on Learning Representations (ICLR)*, 2026c.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

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. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Yifei Yu, Xiaoshan Wu, Xinting Hu, Tao Hu, Yangtian Sun, Xiaoyang Lyu, et al. Videossm: Autoregressive long video generation with hybrid state-space memory. *arXiv:2512.04519*, 2025. URL https://arxiv.org/abs/2512.04519.

Chao Yuan and Pan Li. Accelerating video generation inference with sequential-parallel 3d positional encoding using a global time index. *arXiv:2603.06664*, 2026. URL https://arxiv.org/abs/2603.06664.

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

Zhihang Yuan, Pu Lu, Hanling Zhang, Xuefei Ning, Linfeng Zhang, Tianchen Zhao, Shengen Yan, Guohao Dai, and Yu Wang. Ditfastattn: Attention compression for diffusion transformer models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yuechen Zhang, Jinbo Xing, Bin Xia, Shaoteng Liu, Bohao Peng, Xin Tao, Pengfei Wan, Eric Lo, and Jiaya Jia. Training-free efficient video generation via dynamic token carving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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. *arXiv:2605.15141*, 2026. URL https://arxiv.org/abs/2605.15141.

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. In *International Conference on Learning Representations (ICLR)*, 2025.

Kaiwen Zheng, Guande He, Min Zhao, Jintao Zhang, Huayu Chen, Jianfei Chen, Chen-Hsuan Lin, Ming-Yu Liu, Jun Zhu, and Qianli Ma. Causal-rcm: A unified teacher-forcing and self-forcing open recipe for autoregressive diffusion distillation in streaming video generation and interactive world models. *arXiv:2606.25473*, 2026a. URL https://arxiv.org/abs/2606.25473.

Kaiwen Zheng, Yuji Wang, Qianli Ma, Huayu Chen, Jintao Zhang, Yogesh Balaji, Jianfei Chen, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Large scale diffusion distillation via score-regularized continuous-time consistency. In *International Conference on Learning Representations (ICLR)*, 2026b.

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. In *International Conference on Machine Learning (ICML)*, 2026a.

Lunjie Zhu, Yushi Huang, Xingtong Ge, Yufei Xue, Zhening Liu, Yumeng Zhang, et al. Flashvaed: Plug-and-play vae decoders for efficient video generation. In *International Conference on Machine Learning (ICML)*, 2026b.

Tianrui Zhu, Shiyi Zhang, Zhirui Sun, Jingqi Tian, and Yansong Tang. Memorize-and-generate: Towards long-term consistency in real-time video generation. *arXiv:2512.18741*, 2025. URL https://arxiv.org/abs/2512.18741.

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

Ya Zou, Jingfeng Yao, Siyuan Yu, Shuai Zhang, Wenyu Liu, and Xinggang Wang. Turbo-vaed: Fast and stable transfer of video-vaes to mobile devices. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

APPENDIX

A IMPLEMENTATION DETAILS AND ABLATIONS

This section describes the model, sampling, and runtime configuration used by UNSTEP, and provides ablations. It specifies the denoising and clean-cache procedures, the KV attention window, the V/O SVD truncation, and the DiT and VAE runtime optimizations. UNSTEP performs no retraining or gradient updates. Table 3 summarizes the model and sampling configuration.

Table 3: Overview of the original Self Forcing and default two-step UNSTEP configurations for one 81-frame, 5-second clip. The forward counts include one clean-cache step for each chunk.

Setting	Self Forcing	UNSTEP	§
Output clip	81 frames, 832×480, 16 FPS	81 frames, 832×480, 16 FPS	A.1
Initial noise shape	$1 \times 21 \times 16 \times 60 \times 104$	$1 \times 21 \times 16 \times 60 \times 104$	
Causal chunks	seven of three latent frames	seven of three latent frames	
First chunk	four denoising steps	four denoising steps	
Each later chunk	four denoising steps	two denoising steps	
Total DiT forwards (5-s clip)	35	23	
Clean-cache DiT label	$t = 0$	$t = 0$	A.2
Clean-cache noise scale	0	$\sigma_\star = 0.02$	
Clean-cache input	final denoised estimate	near clean renoised estimate	
Emitted latent	before clean-cache pass	output of clean-cache pass	
Temporal self-attention	growing history	9-frame limit	A.3
Protected cache prefix	none	first three latent frames	
Attention SVD	none	V/O, layers 8–29, 92% retained	A.4

A.1 MODEL AND GENERATION SETUP

The default setup follows the VBench 5s T2V output format (Huang et al., 2024): each clip contains 81 RGB frames at resolution 832×480 (width $\times$ height), played at 16 FPS. The backbone is the causal Wan2.1-T2V-1.3B Self Forcing checkpoint, which generates in the latent space of its Wan VAE. The VAE compresses the height and width of the RGB space by a factor of eight, giving a latent spatial size of 60×104. It compresses time by a factor of four, except that the first RGB frame is encoded separately, giving $1 + (81 - 1)/4 = 21$ latent frames.

The VAE architecture uses 16 latent channels, so the initial Gaussian noise has shape $1 \times 21 \times 16 \times 60 \times 104$. Self Forcing divides this latent sequence into 7 chunks, each of size $1 \times 3 \times 16 \times 60 \times 104$, ordered in batch size, temporal frames, channels, height, and width. Each DiT forward pass emits one chunk with 3 latent frames.

For each of the seven chunks, Self Forcing runs four denoising steps followed by one clean-cache step, giving $7 \times (4+1) = 35$ DiT forwards for the 5-second clip. UNSTEP keeps four denoising steps for the first chunk, but uses two for each of the six later chunks, and also performs one clean-cache step for all chunks. It therefore uses $(4 + 1) + 6 \times (2 + 1) = 23$ DiT forwards.

A.2 DENOISING SCHEDULES AND CLEAN-CACHE PROCEDURE

Using the notation in § 2.4, t_i is the DiT conditioning label and σ_i is its noise level. Self Forcing begins with four equally spaced raw schedule coordinates $u_i \in \{1000, 750, 500, 250\}$, corresponding to noise levels $\{1, 0.75, 0.5, 0.25\}$. It then uses a shift five scheduler to map these values to $\{1, 0.9375, 0.8333, 0.625\}$, respectively, concentrating the schedule at higher noise levels:

$$\sigma_i = \frac{5u_i}{1000 + 4u_i}, \qquad t_i = 1000\sigma_i. \tag{6}$$

The shift remaps the timesteps leaving the endpoints $\{1000, 0\}$ unchanged. Table 4 gives the resulting schedule and the separate clean-cache settings.

Self Forcing uses schedule indices 0, 1, 2, and 3 for every chunk. UNSTEP uses the same four indices for the first chunk. For each later chunk, it uses index 0 followed by either index 2 or index 3, alternating between them because this performed better than always using index 2 or 3. The one-step variant of UNSTEP keeps the first chunk unchanged and uses only index 0 for later chunks, with no change to the clean-cache step. The input to every denoising step after the first is formed by renoising the previous step's clean prediction with fresh Gaussian noise at the new step's noise level, as in Equation (1).

After each chunk's last denoising step, both methods run the clean-cache DiT call in Equation (2) at $t = 0$. Self Forcing uses $\sigma = 0$, so Equations (1) and (3) leave the latent unchanged, it discards the

Table 4: Self Forcing's four denoising steps after shift five mapping and the clean-cache settings used by Self Forcing and UNSTEP.

Schedule index i	Raw u_i	Noise level σ_i	DiT label t_i
0	1000	1	1000
1	750	$15/16 = 0.9375$	937.5
2	500	$5/6 \approx 0.8333$	833.3
3	250	$5/8 = 0.625$	625
Clean SF	–	0	0
Clean UNSTEP	–	0.02	0

prediction and only updates the KV cache. UNSTEP instead uses $\sigma_\star = 0.02$, and emits the updated latent from Equation (3). Its KV cache is updated from the same near clean input.

Figure 5a shows VBench total as we vary the clean-cache noise level $\sigma_\star$, keeping all other wrapper settings at their defaults.

A.3 KV ATTENTION WINDOW

Original Self Forcing retains the keys and values of every generated latent frame, so its self-attention context grows by three latent frames after each chunk. For an 81-frame, 5-second clip, the final chunk can attend to all 21 latent frames. At 832×480, each 60×104 latent frame becomes $30 \times 52 = 1{,}560$ tokens after the DiT's $1 \times 2 \times 2$ patch embedding, giving Self Forcing a maximum context of $21 \times 1{,}560 = 32{,}760$ visual tokens.

UNSTEP instead caps the cache at nine latent frames, or 14,040 tokens. Once the cache is full, it contains three sink frames (the first three latent frames) and the six most recent frames, including the current three-frame chunk.

We found that further reducing the attention window from nine to eight frames in all layers increases speed but lowers quality. As a tradeoff, the default wrapper uses eight frames in some selected layers and nine in the remaining layers, preserving quality while slightly improving speed. The selected layers attend to two sink frames plus the same six recent frames.

A.4 LAYERWISE V/O TRUNCATED SVD

The Wan2.1-T2V-1.3B DiT contains 30 transformer layers, indexed 0–29. UNSTEP leaves the self-attention query and key projections unchanged in every layer, and leaves all self-attention projections in layers 0–7 unchanged. In layers 8–29, each $1{,}536 \times 1{,}536$ value and output weight $\mathbf{W}$ is replaced by the dense truncated SVD reconstruction in Equation (5) at rank

$$r = \mathrm{round}(0.92 \times 1536) = 1413. \tag{7}$$

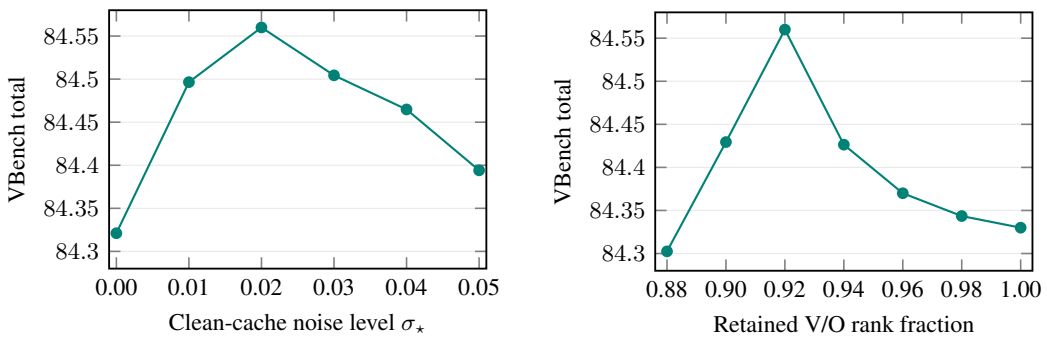
This layer selective truncation is motivated by LASER, which found that truncating early transformer layers can degrade performance, whereas improvements are typically concentrated later (Sharma et al., 2024).

The decomposition and reconstruction are computed once in `float32` during wrapper initialization. Biases are copied unchanged, and the modified transformer is then cast to `bfloat16`. Each reconstructed weight matrix retains its original dense shape, so the projection's parameter count and FLOPs are unchanged.

Figure 5b shows VBench total as we vary V/O rank retention, keeping all other wrapper settings at their defaults.

A.5 DIFFUSION TRANSFORMER RUNTIME

As described in § 2.1, Self Forcing stores the global and local KV cache end positions as GPU scalar tensors. The global position is the token index immediately after the most recently written chunk in the full video. The local position is the index immediately after its keys and values in the cache



(a) V/O rank retention fixed at 0.92.

(b) Clean-cache noise level fixed at $\sigma_\star = 0.02$.

Figure 5: VBench 5s T2V total for clean-cache noise level and V/O rank retention sweeps. Points show measured scores, joined by lines.

buffer. Repeated denoising calls overwrite the same chunk's cache entries without advancing these positions. Python reads these values to select where to write keys and values and which cached range to use. UNSTEP stores these indices on CPU and reuses each read within the attention call, avoiding GPU synchronization for these scalar reads. The keys and values remain on GPU. Self Forcing uses variable-length FlashAttention, which supports different sequence lengths within a batch without padding every sequence to the longest one. Its wrapper constructs extra GPU tensors containing the query and key sequence lengths and their start/end indices on every call, including at batch size one. For a single video, each sequence starts at index zero and its end boundary is the token count, one past the last token's index. UNSTEP uses fixed-length FlashAttention for this setting. The query and key tensors' sequence dimensions already specify their token counts, so this requires no separate GPU tensors for lengths or boundaries. This avoids their repeated allocation and the GPU operations used to construct them. The query and key lengths may differ and can change between calls as the KV cache grows.

UNSTEP also converts the scheduler's fixed timestep and noise level lookup tables to `float64`, copies them to the GPU, and caches them there. Each denoising step uses the cached scheduler tables to obtain the noise level σ_i corresponding to its conditioning label t_i and applies Equation (3) to convert the predicted flow into the clean latent estimate. Original Self Forcing also retains its native scheduler tables on GPU after the first renoising step, but converts them to `float64` again for each clean estimate update. UNSTEP additionally retains the `float64` copies, avoiding repeated dtype conversions.

The fused Triton RoPE kernel described in § 2.1 uses cached `float32` sine and cosine tables. Self Forcing already stores a base table of rotation coefficients for temporal and spatial positions. On every RoPE call, it selects the coefficients for the chunk's frame, row, and column positions and combines them into one coefficient vector per token. UNSTEP builds this once and retains the resulting sine and cosine tables on GPU for each starting latent frame index and grid size. The starting index is the position of the chunk's first latent frame in the full video. For three-frame chunks, these indices are $0, 3, 6, \ldots$, so the chunk starting at 6 uses temporal positions $6, 7, 8$ for RoPE. The grid size specifies the number of latent frames, spatial token rows, and spatial token columns in the chunk. These cached tables provide the coefficients for the current chunk's queries and newly computed keys in all 30 self-attention layers and denoising calls.

Wan2.1-1.3B reshapes each query and newly computed key, after linear projection and RMS normalization, into 12 attention heads of width 128. For one token and one head, we can write this vector as $\mathbf{x} = (x_0, \ldots, x_{127})$. The index $j \in \{0, \ldots, 63\}$ selects the coordinate pair (x_{2j}, x_{2j+1}). In the snippet below, `x_even` and `x_odd` are these two coordinates, while `cos` and `sin` are the cached RoPE coefficients for that token and pair, shared across heads:

```
x_even = tl.load(x_ptr + x_base, mask=mask, other=0.0).to(tl.float32)
x_odd = tl.load(x_ptr + x_base + x_stride_d, mask=mask, other=0.0).to(tl.float32)
cos = tl.load(cos_ptr + freq_base, mask=mask, other=1.0)
sin = tl.load(sin_ptr + freq_base, mask=mask, other=0.0)

y_even = x_even * cos - x_odd * sin
y_odd = x_even * sin + x_odd * cos
```

```
tl.store(out_ptr + out_base, y_even, mask=mask)
tl.store(out_ptr + out_base + out_stride_d, y_odd, mask=mask)
```

The snippet shows the complete arithmetic done by the Triton kernel used by UNSTEP for one coordinate pair. Original Self Forcing converts the input to `float64`, treats coordinate pairs as complex numbers, multiplies by the complex RoPE coefficients, and concatenates and stacks the results before converting back to `bfloat16`. These stages allocate and write intermediate tensors. For each query or key RoPE call, Triton loads the `bfloat16` input, rotates it in `float32`, and stores the `bfloat16` result in one kernel, avoiding those intermediate copies and launches.

The 30-second setting uses the same scheduler cache, Triton kernel, three-latent-frame chunks, and nine-frame KV limit as the 5-second setting. It executes more chunks at later latent frame positions and caches the corresponding sine and cosine tables for those chunks.

A.6 VAE DECODER RUNTIME

The original Self Forcing baseline runs both the DiT and VAE in `bfloat16`. UNSTEP keeps the DiT in `bfloat16`, but casts the VAE weights to `float16` before compilation and the generated latents to `float16` immediately before decoding. Both methods rescale the normalized DiT latents to the scale expected by the VAE decoder using Wan's fixed per channel mean and standard deviation. Each channel uses the same constants across all temporal and spatial positions. The rescaling tensors are constructed once per video on the GPU, in `bfloat16` for Self Forcing and `float16` for UNSTEP, matching their decoder precision. Both methods convert the decoded output to `float32` before rescaling to $[0, 1]$.

Self Forcing leaves the VAE's 3D convolution weights in PyTorch's default layout and passes decode inputs as a permuted $[B, C, T, H, W]$ view of the $[B, T, C, H, W]$ latent tensor, where B, C, T, H, W denote batch size, channels, temporal length, height, and width. This permutation changes the axis order seen by the decoder without rearranging the stored values. UNSTEP stores both the convolution weights and decode inputs in the `channels_last_3d` memory format. For a decode input with logical axes $[B, C, T, H, W]$, this format uses $[B, T, H, W, C]$ storage, with channels as the innermost dimension, enabling faster H100 convolution kernels. UNSTEP also enables cuDNN benchmarking with a limit of 32.

Self Forcing appends each decoded temporal slice by allocating a larger output tensor and copying both the previously decoded output and the new slice into it. UNSTEP instead preallocates the output tensor once and writes each decoded slice into its assigned temporal range, avoiding repeated allocation and copying of earlier slices. The maximum temporal block size is selected to fit decoder memory, and is set to five latent frames in our setup. Both methods cache intermediate results from earlier frames during decoding. For 5-second clips, UNSTEP compiles the full VAE decoding code, including latent rescaling, the input convolution, decoder calls, and output writes. For long clips of 30 seconds or more, compiling all these operations for the full clip can exhaust GPU memory. UNSTEP instead compiles the decoder alone and calls it on successive temporal blocks, writing each block's decoded frames into the same preallocated output tensor.

A.7 RECOVERY MECHANISMS WITH THE FULL DENOISING SCHEDULE

In this section we provide an ablation for the recovery mechanisms with the full denoising schedule. We keep Self Forcing's full four-step denoising schedule for every chunk and evaluate VBench 5s T2V with full attention as in default Self Forcing, or limited attention as in default UNSTEP, using its nine-latent-frame KV window (Appendix A.3). Table 5 compares clean-cache refinement and V/O SVD truncation, applied separately and together, with all other settings fixed within each attention setting.

Under the full schedule, these mechanisms provide little benefit and can reduce quality, with only a slight gain from SVD under limited attention. This contrasts with their combined benefit under the shortened schedule (Table 1).

Table 5: VBench total with the full denoising schedule. SVD retains 92% of V/O rank, and clean-cache refinement uses $\sigma_\star = 0.02$. The baseline uses $\sigma_\star = 0$ and full rank. Bold marks the highest score per row.

Attention	Baseline	SVD truncation	Clean-cache refinement	Both
Full	**84.31**	84.18	83.88	83.83
Limited	84.03	**84.10**	83.59	83.53

B Evaluation and Reproducibility Details

B.1 VBench 5s text-to-video evaluation

We evaluate our 5-second T2V outputs on the standard VBench prompt suite (Huang et al., 2024), which contains 946 text prompt entries assigned across 16 dimensions. Seven quality dimensions score the generated video without using prompt text, while nine semantic dimensions evaluate whether the video satisfies the semantic content of the original VBench prompt. We generate one 81-frame, 832×480 video at 16 FPS for each entry.

Following Self Forcing (Huang et al., 2025a), we use its extended prompts for generation but use the corresponding original VBench prompts for semantic evaluation. Self Forcing produced the extended prompts with Qwen2.5-7B-Instruct using Wan2.1's English prompt extension instruction. For example, the original prompt "a suitcase and a vase" becomes "A well-lit, detailed close-up of a vintage leather suitcase and a delicate glass vase placed side by side on a wooden table…"

Each VBench dimension is evaluated on the prompts assigned to that dimension. The seven quality dimensions are subject consistency, background consistency, temporal flickering, motion smoothness, dynamic degree, aesthetic quality, and imaging quality. Their evaluators use DINO ViT-B/16, CLIP ViT-B/32, consecutive frame pixel differences, AMT-S frame interpolation, RAFT optical flow, the LAION aesthetic predictor with CLIP ViT-L/14 features, and MUSIQ-SPAQ, respectively. The nine semantic dimensions are object class, multiple objects, human action, color, spatial relationship, scene, appearance style, temporal style, and overall consistency. GRiT evaluates object class, multiple objects, color, and spatial relationships. UMT evaluates human actions. Tag2Text evaluates scenes. CLIP evaluates appearance style. ViCLIP evaluates temporal style and overall consistency.

We use the official `vbench_standard` metric implementations and aggregation. VBench first normalizes each dimension. After that quality and semantic scores are calculated as weighted means of their seven and nine dimensions, respectively. Dynamic degree has weight 0.5 and every other dimension has weight 1. The total is $(4\,\mathrm{Quality} + \mathrm{Semantic})/5$, and all reported scores are multiplied by 100.

B.2 Image-to-video evaluation

VBench-I2V (Huang et al., 2025b) contains 1,118 image–prompt entries derived from 355 reference images: 246 subject entries, 109 background entries, and seven camera motion instructions for each background image, giving 763 camera motion entries. We generate one 5-second clip per prompt using the English prompt and the official 16:9 reference image crop, with the same 832×480 resolution and 16 FPS as T2V. The UnStep denoising schedule, KV cache policy, clean-cache setting, SVD, and runtime optimizations are applied without I2V-specific retuning.

VBench-I2V reports three main aggregates: Total, Quality, and I2V. Quality aggregates six video quality dimensions: subject consistency, background consistency, motion smoothness, dynamic degree, aesthetic quality, and imaging quality. I2V aggregates three conditioning fidelity dimensions: video–image subject fidelity, video–image background fidelity, and video–text camera motion adherence. Subject and background fidelity use DINO ViT-B/16 and DreamSim features, respectively. Each score combines the maximum similarity between the reference and a generated frame with the mean and minimum similarities between consecutive video frames, using weights $0.4, 0.3, 0.3$. CoTracker2 checks whether the generated motion matches the prompt's camera instruction: pan left or right, tilt up or down, zoom in or out, or static.

After normalization, the I2V score weights subject and background fidelity by 1 and camera motion by 0.1. The Quality score weights dynamic degree by 0.5 and its other five dimensions by 1. The Total score is the equal average of I2V and Quality. Figure 4a reports the aggregates: Total, Quality, and I2V, and Figure 6 reports their nine component dimensions on a 0–100 scale.

For measuring I2V throughput, we follow the same per clip timing region as in Appendix B.4. The reference image is preprocessed and VAE-encoded before the timer begins. The timed region covers generation conditioned on that encoded latent and VAE decoding of the complete video. As before, the first clip speed is not considered. Scoring on the other hand uses the standard VBench-I2V evaluator explained above.

B.3 Long video evaluation

We use the 40-prompt long video competition track from VBench++ (Huang et al., 2025b). We generate one video per prompt with 123 latent frames in chunks of three, yielding 489 RGB frames, or 30.56 seconds at 16 FPS. UnStep retains the T2V denoising schedule, KV cache limit, sink size, and SVD configuration without retuning. Its long video VAE compilation path is described in Appendix A.6. UnStep's throughput timer uses the same GPU generation region as T2V. The reported long video FPS excludes the first clip, which includes compilation and autotuning, and averages the remaining measurements.

VBench-Long reports three main aggregates instead of a combined total: Temporal Quality, Framewise Quality, and Text Alignment. These aggregates are formed from eight component metrics. The scoring pipeline uses PySceneDetect to detect scene cuts and divides each resulting scene into 2-second clips. A final partial clip is extended with preceding frames, while a scene shorter than two seconds is retained as one clip. Each metric is first averaged across clips from the same video and then across the 40 videos. Temporal Quality combines DINO subject consistency, CLIP background consistency, AMT-S motion smoothness, and RAFT dynamic degree with weights $1, 1, 1, 0.5$. Our configuration uses the within clip consistency scores and gives the optional cross clip term zero weight. Framewise Quality is the equal average of the LAION aesthetic and MUSIQ-SPAQ imaging scores. Text Alignment is the equal average of ViCLIP overall consistency and CLIP ViT-B/32 frame–text similarity. Figure 4b reports the aggregates: Temporal Quality, Framewise Quality, and Text Alignment, and Figure 7 reports their eight component metrics on a 0–100 scale.

B.4 Reproducibility and timing

We divide the 946 benchmark entries into eight fixed groups (shards) of consecutive entries. At the start of each shard, the process resets the global random number generator (RNG), which produces the Gaussian noise used during sampling, to seed zero. For every entry preceding the shard in benchmark order, UnStep consumes its initial noise draw, while the upstream Self Forcing baseline advances the random draws made by its original inference path. This defines a deterministic random sequence under the fixed logical shard protocol. The eight shards can run concurrently on eight GPUs as one per GPU, or in waves on fewer GPUs. Preserving the logical shard count and boundaries preserves prompt assignment and RNG consumption. Each video is generated on one GPU, so the reported FPS remains single GPU throughput.

For UnStep, each FPS measurement uses CUDA synchronization at both boundaries and includes all DiT calls, KV cache updates, VAE decoding, and final GPU postprocessing. Model and wrapper loading, compilation, and text encoding occur before the timer starts.

The first generated clip on each GPU completes compilation and autotuning and serves as warmup. We report the average FPS over all remaining clips, starting from the second clip on each GPU. The timings therefore exclude the first prompt, but the quality scoring includes all 946 prompts.

Our H100 runs use PyTorch `2.15.0a0+git65cd5f0`, CUDA runtime 13.3 (toolkit 13.3.1), cuDNN 9.25.1, Triton 3.8.0, and FlashAttention-3. For UnStep, the DiT and the Wan VAE's `decode` and `cached_decode` functions use `torch.compile` in `max-autotune-no-cudagraphs` mode, and cuDNN benchmarking is enabled with `torch.backends.cudnn.benchmark_limit = 32`. Our GB200 runs follow the same setup, except that we use FlashAttention-2 instead of FlashAttention-3.

C Extended Results

C.1 Run-to-run variation

As described in Appendix B.4, repeated runs use the same generation seed and sharding protocol, with the checkpoint and sampling configuration also fixed. Despite these fixed settings, scores vary between executions. Therefore all VBench quality scores reported in the paper are the mean of five runs. For the UnStep default two-step version on the Self Forcing checkpoint, Table 6 reports the mean and standard deviation of the Total, Quality, and Semantic scores over the full 946-prompt VBench benchmark.

Table 6: Run-to-run variation for Self Forcing + UnStep with two denoising steps on H100.

Metric	Mean	Standard deviation
Total	84.56	0.10
Quality	85.20	0.11
Semantic	82.02	0.12

C.2 Figure 1 comparison details

This section provides more details on the configurations in Figure 1. Table 7 lists the H100 comparison without the one-step configurations. We consider methods that target fast generation and report VBench scores as well as end-to-end GPU throughput, including both the DiT and VAE. All FPS and VBench scores are taken from the cited papers.

Table 7: Details on the entries in the H100 panel of Figure 1 (without one-step models).

Method	Venue	FPS	Total	Quality	Semantic	Dominant speed gain
Causal Forcing + UnStep, 2 steps	This work	49.8	84.93	85.79	81.46	Runtime + steps/KV
Self Forcing + UnStep, 2 steps	This work	49.8	84.56	85.20	82.02	Runtime + steps/KV
DiagDistill (Liu et al., 2026b)	ICLR'26	31.0	84.48	85.26	81.73	Tiny decoder
Hybrid Forcing (Li et al., 2026c)	arXiv	29.5	83.60	84.45	80.19	Sparse/linear attention
Causal-rCM (Zheng et al., 2026a)	arXiv	25.6	84.24	84.96	81.36	Fewer denoising steps
Dummy Forcing (Guo et al., 2026)	arXiv	24.3	83.90	84.63	80.98	Reduced KV attention
Reward Forcing (Lu et al., 2026)	CVPR'26	23.1	84.13	84.84	81.32	Reduced KV attention
rCM, 2 steps (Zheng et al., 2026b)	ICLR'26	23.0	84.09	84.90	80.86	Fewer denoising steps
DSA (Le et al., 2026)	CVPR'26	22.6	83.73	84.75	79.67	Adaptive denoising steps
Ms. Forcing (Li et al., 2026e)	arXiv	21.9	83.89	84.61	81.01	Multiple patch sizes
MAG (Zhu et al., 2025)	arXiv	21.7	83.52	84.11	81.14	KV compression
Sparse Forcing (Xu et al., 2026)	arXiv	19.9	83.99	84.65	81.36	Sparse attention
Light Forcing (Lv et al., 2026)	ICML'26	18.9	84.50	85.40	80.90	Sparse attention
In-Context Forcing (Yang et al., 2026a)	arXiv	18.4	84.34	84.99	81.77	Parallel causal sampling
AnyFlow (Gu et al., 2026)	arXiv	17.4	84.31	85.15	80.94	Fewer denoising steps

Table 8 lists the one-step H100 configurations. We use the VBench scores reported in the original papers for all methods. We measured end-to-end H100 throughput for both One-Forcing variants (Feng

et al., 2026) and Causal Forcing++ (Zhao et al., 2026), because One-Forcing reports no throughput and Causal Forcing++ reports only DiT throughput on A800. The rCM throughput is taken from its paper.

Table 8: One-step models in the H100 panel of Figure 1.

Method	Venue	FPS	Total	Quality	Semantic
Self Forcing + UNSTEP, 1 step	This work	59.6	83.34	83.84	81.34
Causal Forcing + UNSTEP, 1 step	This work	59.6	83.10	83.81	80.27
rCM, 1 step (Zheng et al., 2026b)	ICLR'26	32.3	82.65	83.60	78.82
One-Forcing, 1 step (chunkwise) (Feng et al., 2026)	arXiv	25.2	81.60	83.65	73.41
One-Forcing, 1 step (framewise) (Feng et al., 2026)	arXiv	20.7	83.76	85.22	77.91
Causal Forcing++, 1 step (Zhao et al., 2026)	arXiv	20.4	83.35	84.50	78.75

Table 9 lists the GB200 configurations. Flex-Forcing uses larger chunks, which denoises more latent frames per emitted chunk and therefore reduces the number of DiT forward passes. Attention is bidirectional within each chunk and causal between chunks. Since the chunk sizes affect speed and quality, the paper searches different configurations of the 21 latent frames in a 5-second video and selects the best based on VBench scores and throughput. LongLive 2.0 quantizes weights, activations, and the KV cache to NVFP4. Its GB200 configuration uses an additional GPU for VAE decoding, which decodes one chunk while the DiT denoises the next. Both methods remain slower than UNSTEP.

Table 9: Points in the GB200 panel of Figure 1.

Method	Venue	FPS	Total	Quality	Semantic	Dominant speed gain
Self Forcing + UNSTEP, 2 steps	This work	63.7	84.36	85.06	81.56	Runtime + steps/KV
Causal Forcing + UNSTEP, 2 steps	This work	63.7	84.59	85.50	80.97	Runtime + steps/KV
Self Forcing + UNSTEP, 1 step	This work	77.4	83.49	84.09	81.08	Runtime + steps/KV
Causal Forcing + UNSTEP, 1 step	This work	77.4	83.16	84.15	79.22	Runtime + steps/KV
Flex-Forcing (12–6–3) (Ma et al., 2026a)	ICML'26	45.6	84.29	85.33	80.12	Chunk schedule search
Flex-Forcing (7–7–7) (Ma et al., 2026a)	ICML'26	48.9	83.92	84.96	79.78	Chunk schedule search
Flex-Forcing (3–3–3–3–3–3–3) (Ma et al., 2026a)	ICML'26	41.5	83.91	84.85	80.16	Chunk schedule search
LongLive 2.0 (Chen et al., 2026)	arXiv	45.7	83.14	85.40	74.12	NVFP4 quantization / additional VAE GPU

C.3 REPRODUCED BASELINES

To evaluate UNSTEP on Self Forcing, Causal Forcing and LongLive 1.0 as shown in Table 2, we evaluate the VBench score of each checkpoint before and after applying the wrapper. We evaluate all checkpoints under the same 946-prompt protocol in Appendix B.

We reproduce Self Forcing's published total, with slightly different quality and semantic scores. Our reproduced Causal Forcing total is higher than the original paper's report, while our LongLive 1.0 total is lower. LongLive's official repository does not provide a VBench evaluation command or the prompt preprocessing used for that result, therefore we could not reproduce the reported score. However, our LongLive 1.0 total remains in the 83-point range reported by other recent work (Lv et al., 2026; Zheng et al., 2026a; Lu et al., 2026; Ji et al., 2026; Yu et al., 2025).

C.4 Full text-to-video dimension scores

Table 10 reports all 16 normalized VBench T2V dimensions for Self Forcing and Causal Forcing before and after applying UnStep. Each delta compares runs generated and scored with the same protocol, using the aggregation in Appendix B.1. For Self Forcing, UnStep changes the quality aggregate by -0.08 and the semantic aggregate by $+1.58$. For Causal Forcing, the corresponding changes are $+0.16$ and -0.09. The total scores increase by 0.25 and 0.11, respectively, but individual dimensions move in both directions.

Table 10: Normalized VBench dimension scores from runs over all 946 prompts. SF and CF denote Self Forcing and Causal Forcing. Each Δ is the wrapped score minus its corresponding unwrapped score. Deltas are computed before rounding.

	SF	SF + UnStep	Δ_{SF}	CF	CF + UnStep	Δ_{CF}
Quality dimensions						
Subject consistency	94.78	95.10	+0.31	94.99	94.67	−0.32
Background consistency	95.11	95.16	+0.05	94.73	95.04	+0.31
Temporal flickering	97.42	97.64	+0.22	95.13	96.79	+1.66
Motion smoothness	94.94	95.53	+0.59	92.07	93.19	+1.12
Dynamic degree	73.83	69.79	−4.04	86.11	84.72	−1.39
Aesthetic quality	65.85	66.19	+0.34	66.16	66.20	+0.04
Imaging quality	69.29	69.29	0.00	70.50	69.40	−1.10
Semantic dimensions						
Object class	93.05	96.00	+2.95	95.97	94.09	−1.88
Multiple objects	85.71	87.52	+1.80	89.71	93.79	+4.08
Human action	97.65	96.00	−1.65	96.00	95.25	−0.75
Color	88.19	91.00	+2.81	89.24	90.13	+0.89
Spatial relationship	82.52	84.11	+1.59	79.26	76.88	−2.38
Scene	67.28	70.05	+2.77	70.54	70.07	−0.46
Appearance style	71.08	71.57	+0.49	71.93	71.21	−0.73
Temporal style	65.94	67.10	+1.17	68.36	68.45	+0.09
Overall consistency	72.52	74.82	+2.30	72.95	73.26	+0.32
Quality aggregate	85.28	85.20	−0.08	85.64	85.79	+0.16
Semantic aggregate	80.44	82.02	+1.58	81.55	81.46	−0.09
Total	84.31	84.56	+0.25	84.82	84.93	+0.11

C.5 Full image-to-video and long video dimension scores

Figures 6 and 7 expand the aggregate results in Figure 4 into the component dimensions defined in Appendix B.2 and B.3. Pale bars are Self Forcing and solid bars are the same checkpoint wrapped with UnStep.

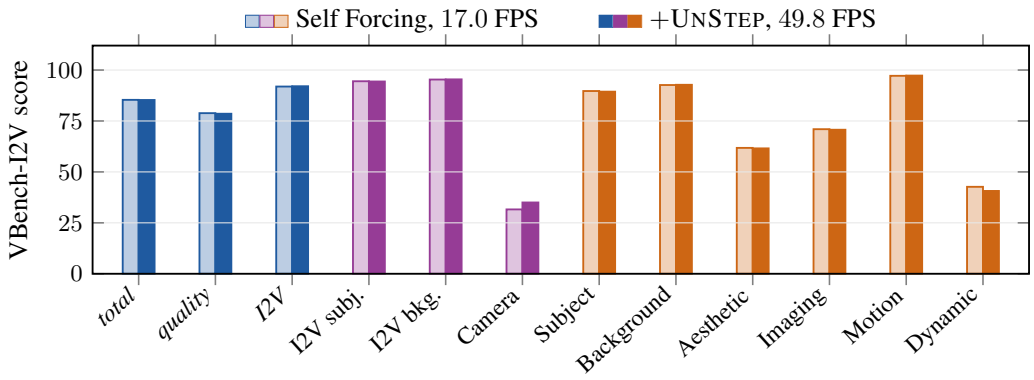


Figure 6: Full VBench-I2V results on 1,118 image–prompt entries with 16:9 crops on one H100. The first three bars are the Total, Quality, and I2V aggregates. The remaining bars are their component dimensions.

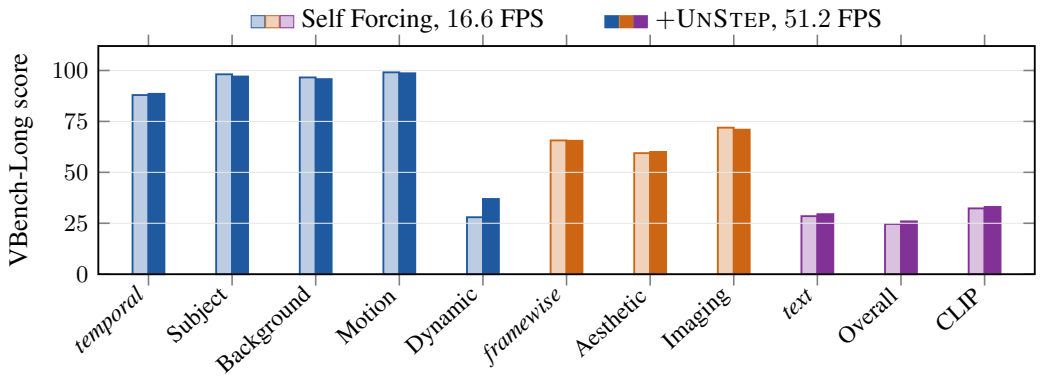


Figure 7: Full VBench-Long results on 40 prompts at 30 seconds per clip on one H100. The first bar in each color group is the aggregate, followed by its component dimensions.

D Related Work

Few-step causal video models. CausVid distilled a bidirectional video diffusion model into an autoregressive generator (Yin et al., 2025). Self Forcing reduced the train–test mismatch by distilling on the student's own rollouts (Huang et al., 2025a), whereas Causal Forcing initialized the causal student using an autoregressive ODE teacher before applying asymmetric distribution matching (Zhu et al., 2026a). These methods train few-step causal models through distillation. UNSTEP instead changes the inference procedure of an already distilled causal checkpoint.

Broader few-step generation. Diffusion Forcing introduced per token noise levels sampled independently for sequence generation (Chen et al., 2024). Closest to UNSTEP are the causal few-step video models distilled from multi-step bidirectional teachers. The broader literature includes progressive distillation, consistency models, and distribution matching distillation (Salimans & Ho, 2022; Song et al., 2023; Yin et al., 2024b;a), as well as adversarial distillation, shortcut models, and continuous time consistency models (Sauer et al., 2024; Lin et al., 2024; Frans et al., 2025; Lu & Song, 2025). Video-specific extensions include adversarial post-training and one-step or transition matching objectives (Lin et al., 2025b; Li et al., 2026a; Lin et al., 2025c; Yang et al., 2026c; Nie et al., 2026). Other work reduces the number of denoising evaluations through few-step consistency or distribution matching objectives (Zheng et al., 2026b; Zhao et al., 2026; Zheng et al., 2026a; Lu et al., 2026), reduced steps or one-step training (Gu et al., 2026; Feng et al., 2026), and noisy causal context with parallel denoising (Yang et al., 2026a). Salt combines self-consistent distribution matching with cache-aware mixed step training to improve low step autoregressive generation (Ge et al., 2026). DSA learns to allocate steps across frames, while Flex-Forcing trains a unified causal and bidirectional model and evaluates different chunk configurations (Le et al., 2026; Ma et al., 2026a). These methods provide broader context for trained low step generation, but require optimization of the model rather than changing only its inference procedure.

Training-free acceleration of causal video. Several causal video approaches change the autoregressive computation without retraining. Block Cascading changes the block schedule (Bandyopadhyay et al., 2026), while Dummy Head and HeadCast restrict history access for certain attention heads (Guo et al., 2026; Shen et al., 2026). FlowCache reuses computations across autoregressive flow evaluations, while history-guided caching combines feature reuse with residual correction (Ma et al., 2026b; Nan et al., 2026). Surprise Forcing jointly decides what history to retain and when to skip computation (Shi et al., 2026). UNSTEP instead uses a fixed shorter schedule and bounded KV window, then recovers their quality loss through the clean-cache pass and V/O spectral edit.

Caching and sparse attention for diffusion models. Feature reuse in conventional diffusion models includes DeepCache, tokenwise feature caching, and Pyramid Attention Broadcast (Ma et al., 2024; Zou et al., 2025; Zhao et al., 2025). Feature caching methods such as TeaCache and FasterCache reuse computations across nearby denoising evaluations in video diffusion (Liu et al., 2025; Lv et al., 2025). Other methods reduce diffusion transformer attention or token computation through compressed attention, spatiotemporal sparsity, or adaptive sparsity (Yuan et al., 2024; Xi et al., 2025; Yang et al., 2025; Xia et al., 2025). Jenga selects which token pairs interact in attention and progres-

sively increases latent resolution (Zhang et al., 2025). Residual, sparsity, and second-order caching methods provide further acceleration for multi-step diffusion trajectories (Fang et al., 2026; Lin et al., 2026; Liu et al., 2026a). The caching approaches are less direct comparisons for a few-step autoregressive model, where there are fewer adjacent denoising evaluations to reuse.

Attention, decoder, and runtime acceleration. Trained causal models reduce attention or memory cost through sparse, hybrid, or multiscale context (Lv et al., 2026; Xu et al., 2026; Li et al., 2026c;e), while Forcing-KV uses training-free head-dependent KV compression (Ji et al., 2026). MonarchRT also targets efficient attention for Self Forcing (Agarwal et al., 2026). DiagDistill and Flash-VAED use small replacement decoders (Liu et al., 2026b; Zhu et al., 2026b). UNSTEP keeps the original checkpoint and VAE weights, but combines inference-only runtime changes across the DiT and decoder in addition to changes to the schedule and attention window.

Efficient video generation and decoding. FlashAttention, FlashAttention-3, and `torch.compile` reduce standard transformer execution overhead (Dao et al., 2022; Shah et al., 2024; Ansel et al., 2024), while video-specific systems optimize positional encoding, attention execution, or low-precision hardware paths (Yuan et al., 2026; Yuan & Li, 2026; Chen et al., 2026). StreamDiT targets real-time streaming text-to-video generation (Kodaira et al., 2026), while PyramidalWan finetunes pretrained video diffusion models to denoise at progressively higher resolutions (Korzhenkov et al., 2026). Turbo-VAED distills compact video VAE decoders for mobile devices (Zou et al., 2026). Neodragon combines distillation and denoiser pruning for mobile video generation (Karnewar et al., 2026), while DeltaQuant quantizes video diffusion models to four bits (Li et al., 2026d). UNSTEP instead reduces denoising steps, limits KV attention, and optimizes execution of the pretrained causal DiT and VAE decoder, without retraining or four-bit quantization.

KV window and spectral edit. The limited KV attention window follows the attention sink idea in StreamingLLM (Xiao et al., 2024) and its adaptation to causal video generation in LongLive (Yang et al., 2026b). UNSTEP combines reduced attention window with reduced step inference and clean-cache recovery. Its V/O truncation is motivated by LASER, which showed selective post-training rank reduction can improve some tasks (Sharma et al., 2024). UNSTEP replaces selected V/O weight matrices by dense truncated SVD reconstructions to recover quality instead of reducing projection cost.

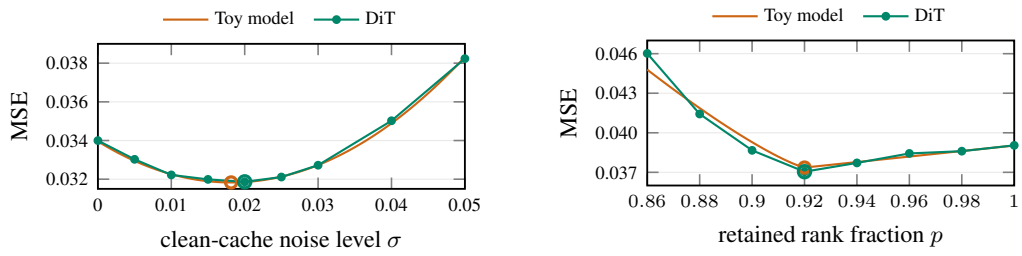
Low rank compression and slimming. SVD-LLM uses matrix decomposition for language model compression (Wang et al., 2025). SVDQuant uses low rank components to absorb outliers during low-bit diffusion model quantization (Li et al., 2025), while SlimDiff uses activation statistics for training-free diffusion model slimming (Roy et al., 2025). Increment-calibrated caching uses channel-aware SVD to cheaply update cached outputs based on how the input changes between denoising steps (Chen et al., 2025). These methods use low rank decomposition or structured slimming to reduce storage or execution cost. UNSTEP instead reconstructs the selected V/O matrices at their original dense shape and uses truncation only as a spectral edit for quality recovery.

E TOY MODELS FOR INFERENCE-ONLY RECOVERY

This appendix uses toy models to provide intuition why the refined clean-cache pass and reduced rank of the projection matrices can help reduce the error introduced by the shortened schedule. We plot the mean squared error (MSE) between full and shortened schedule latent vectors, using the toy models' mean risks and empirical measurements from the DiT. Figure 8 shows both plots as we vary the clean-cache noise level and retained V/O rank fraction. Appendix F states the model, schedule, and noise assumptions used in the proofs.

E.1 CLEAN-CACHE REFINEMENT

To analyze the clean-cache refinement, we compare the latent estimate produced by a shortened schedule with the estimate produced by the full schedule. We fix the text prompt and preceding KV cache. Let $\mathbf{x}_{\text{full}}, \mathbf{x}_{\text{short}} \in \mathbb{R}^d$ be the random latent estimates obtained by recursively applying the flow matching updates in equations (1)–(3) from the same initial noise with the full and shortened schedules, respectively, before the clean-cache pass.



(a) Clean-cache noise level.

(b) Retained V/O rank fraction.

Figure 8: Mean toy risks (orange) and measured DiT latent MSE (green) as a function of clean-cache noise level σ (left) and retained V/O rank fraction p (right). The toy curves evaluate equation (9) conditional on the initial vector and equation (13), respectively.

Let $\mathbf{x}_{\text{clean}}(\sigma)$ denote the clean-cache output obtained from $\mathbf{x}_{\text{short}}$ using equations (1)–(3) with $t_i = 0$, $\sigma_i = \sigma \in [0, 1]$, and Gaussian noise $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ independent of both schedule estimates. Here $\mathbf{I}_d$ is the $d \times d$ identity matrix. Appendix F derives $\mathbf{x}_{\text{full}}$ and $\mathbf{x}_{\text{short}}$ in terms of $\mathbf{A}$ and $\boldsymbol{\epsilon}$, and Appendix F.1 derives $\mathbf{x}_{\text{clean}}(\sigma)$ in terms of $\mathbf{A}$, $\boldsymbol{\epsilon}$, and σ. We model the DiT at every conditioning label, including the clean label, by the same linear map:

$$f_{\boldsymbol{\theta}}(\mathbf{x}, t) = \mathbf{A}\mathbf{x}, \qquad \mathbf{A} = \mathbf{A}^\top, \qquad \mathbf{0} \preceq \mathbf{A} \prec \mathbf{I}_d. \tag{8}$$

We define the residual as the difference between the estimate after the clean-cache pass and the full schedule estimate, and the risk as the expected squared residual:

$$\mathbf{r}(\sigma) = \mathbf{x}_{\text{clean}}(\sigma) - \mathbf{x}_{\text{full}}, \qquad R_{\text{clean}}(\sigma) = \mathbb{E}\left[\|\mathbf{r}(\sigma)\|_2^2\right], \tag{9}$$

where the expectation is over $\boldsymbol{\epsilon}$. Using the linear model in equation (8) and expanding the squared residual gives

$$\begin{aligned} R_{\text{clean}}(\sigma) &= \mathbb{E}\|\mathbf{x}_{\text{short}} - \mathbf{x}_{\text{full}}\|_2^2 - 2g\sigma + \rho(\sigma), \\ g &= \mathbb{E}\left[(\mathbf{x}_{\text{short}} - \mathbf{x}_{\text{full}})^\top (\mathbf{I}_d + \mathbf{A})\mathbf{x}_{\text{short}}\right], \qquad |\rho(\sigma)| \le k\sigma^2, \end{aligned} \tag{10}$$

where k is a finite positive scalar, and $g > 0$ for the shortened schedule (proof in Appendix F.1).

Our objective is to choose $\sigma \in [0, 1]$ that minimizes $R_{\text{clean}}(\sigma)$. For the full schedule, the clean-cache pass at $\sigma = 0$ leaves $\mathbf{x}_{\text{full}}$ unchanged and gives zero risk. Every $\sigma > 0$ gives positive risk, so $\sigma^\star_{\text{full}} = 0$ is the unique minimizer.

For the shortened schedule, let $\sigma^\star_{\text{short}} \in \arg\min_{\sigma \in [0,1]} R_{\text{clean}}(\sigma)$ denote a risk minimizer. The shortened schedule minimizer satisfies

$$\sigma^\star_{\text{short}} > \frac{1}{3} \min\left\{1, \frac{g}{k}\right\} > 0. \tag{11}$$

Appendix F.1 gives the proof. This motivates UNSTEP to reuse the existing clean-cache pass with a tuned nonzero noise level after shortening the denoising schedule.

To visualize equation (9), Figure 8a compares the shortened schedule's mean toy risk for a fixed initial vector with the measured DiT MSE, computed between the full and shortened schedule latent vectors. Both the toy model and the DiT measurements have their lowest error at a nonzero noise level. Appendix F expands the risk into the polynomial in equation (24) and gives the fixed numerical parameters used to plot it.

E.2 V/O TRUNCATED SVD

As in Appendix E.1, we use the latent estimates $\mathbf{x}_{\text{full}}$ and $\mathbf{x}_{\text{short}}$ and the linear DiT $f_{\boldsymbol{\theta}}(\mathbf{x}, t) = \mathbf{A}\mathbf{x}$ under the matrix assumptions in equation (8), with the additional assumption that $\mathbf{A}$ is rank deficient, i.e., $\text{rank}(\mathbf{A}) < d$.

To model V/O truncation, we represent the part of the DiT being truncated by a fixed symmetric matrix $\mathbf{W} \in \mathbb{R}^{d \times d}$ and the part kept unchanged by $b\mathbf{I}_d$, where b is a small positive number. We replace $\mathbf{W}$ by its rank-r truncated SVD $\widehat{\mathbf{W}}_r$:

$$\mathbf{A} = b\mathbf{I}_d + \mathbf{W}, \qquad \mathbf{A}_r = b\mathbf{I}_d + \widehat{\mathbf{W}}_r, \qquad f_r(\mathbf{x}, t) = \mathbf{A}_r\mathbf{x}. \tag{12}$$

Let $\mathbf{x}_{\text{full}}(r)$ and $\mathbf{x}_{\text{short}}(r)$ denote the estimates obtained by the same denoising updates with $\mathbf{A}_r$ instead of $\mathbf{A}$. Appendix F.2 applies the shared derivation in Appendix F to these estimates. At $r = d$, these are the original estimates $\mathbf{x}_{\text{full}}$ and $\mathbf{x}_{\text{short}}$.

We define the risk as the expected squared difference between $\mathbf{x}_{\text{short}}(r)$ and $\mathbf{x}_{\text{full}}$:

$$R_{\text{svd}}(r) = \mathbb{E}\left[\|\mathbf{x}_{\text{short}}(r) - \mathbf{x}_{\text{full}}\|_2^2\right]. \tag{13}$$

The expectation is over $\boldsymbol{\epsilon}$. The clean-cache noise level is fixed at zero for this toy rank comparison. Let $q = \text{rank}(\mathbf{A})$. Equation (13) can be written as

$$R_{\text{svd}}(r) = R_{\text{svd}}(q) + \begin{cases} \displaystyle\sum_{r<\ell\le q} \eta_\ell, & 0 \le r < q, \\ (r-q)\kappa, & q \le r \le d. \end{cases} \tag{14}$$

Here η_ℓ and κ are strictly positive scalars derived from the expected squared errors after recursively applying equations (1)–(3) from the same initial noise, using $\mathbf{A}$ for the full schedule and $\mathbf{A}_r$ for the shortened schedule (Proof in Appendix F.2).

Relative to $R_{\text{svd}}(q)$, choosing $r < q$ adds η_ℓ for each discarded direction $r < \ell \le q$, while choosing $r > q$ adds κ for each additional retained direction. Since these added terms are positive for every $r \ne q$, equation (14) is uniquely minimized at $r^\star_{\text{short}} = q < d$. For the full schedule, replacing $\mathbf{x}_{\text{short}}(r)$ by $\mathbf{x}_{\text{full}}(r)$ in equation (13) gives zero risk at $r = d$ and positive risk for every $r < d$, so $r^\star_{\text{full}} = d$ is the unique minimizer. This motivates UNSTEP to use a tuned reduced rank for selected V/O projections after shortening the denoising schedule.

To visualize equation (13), Figure 8b compares the shortened schedule's mean toy risk, plotted against the retained rank fraction $p = r/d$, with the measured DiT MSE between full and shortened schedule latent vectors. Both the toy model and the DiT measurements have their lowest error below full retention. Appendix F gives the fixed matrices, schedules, and risk calculation used for the toy curve.

F PROOFS FOR THE TOY MODELS

This appendix gives the proofs and plotting details for the clean-cache and V/O truncation models in Appendix E.

Shared model and schedule assumptions. We assume the linear DiT and matrix conditions in equation (8) at every conditioning label, with the text prompt and preceding KV cache fixed. Write the full schedule's noise levels as $(s_0, \dots, s_n)$, and let $F = \{0, \dots, n\}$ be its indices. We assume $s_0 = 1$ and $0 < s_j < 1$ for $j > 0$. Let J denote the indices retained by the shortened schedule. We assume $J \subsetneq F$ and $0 \in J$. These conditions include the full and shortened schedules in Appendix A.2.

Shared noise assumptions. We assume independent $\boldsymbol{\epsilon}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ for $j = 0, \dots, n$. Here $\boldsymbol{\epsilon}_0$ is the initial noise, and $\boldsymbol{\epsilon}_j$ is the renoising vector at level s_j for $j > 0$. We assume that all compared trajectories use the same $\boldsymbol{\epsilon}_j$ at each retained level, including initialization. In both proofs, the expectation over $\boldsymbol{\epsilon}$ includes initialization and the subsequent noises.

Derivation of the shared denoising update from equations (1)–(3). For a linear DiT $f_{\boldsymbol{\theta}}(\mathbf{x}, t) = \mathbf{A}\mathbf{x}$ and an incoming latent estimate $\mathbf{x} \in \mathbb{R}^d$, equations (1)–(3) give

$$\mathbf{x}_{\text{next}} = (\mathbf{I}_d - s_j\mathbf{A})\left[(1 - s_j)\mathbf{x} + s_j\boldsymbol{\epsilon}_j\right]. \tag{15}$$

At $s_0 = 1$, this gives $(\mathbf{I}_d - \mathbf{A})\boldsymbol{\epsilon}_0$, independently of the incoming estimate. The full schedule applies this update at every index in F. The shortened schedule applies it only at indices in J. At each later step, $\mathbf{x}$ is the output of the preceding step in that schedule.

Shared derivation of $\mathbf{x}_{\text{full}}$ and $\mathbf{x}_{\text{short}}$. The matrix assumptions imply that $\mathbf{A}$ has orthonormal eigenvectors $\mathbf{u}_\ell$ with eigenvalues $a_\ell \in [0, 1)$. Write $\epsilon_{j,\ell} = \mathbf{u}_\ell^\top \boldsymbol{\epsilon}_j$. For a scalar eigenvalue $a \in [0, 1)$, one step of equation (15) multiplies the previous coordinate by $q_j(a)$ and adds $v_j(a)\epsilon_{j,\ell}$, where

$$q_j(a) = (1 - s_j)(1 - s_j a), \qquad v_j(a) = s_j(1 - s_j a). \tag{16}$$

The schedule assumptions and these eigenvalue bounds give $q_0(a) = 0$ and $v_0(a) = 1 - a > 0$. For $j > 0$, they give $0 < q_j(a) < 1$ and $v_j(a) > 0$. For a schedule with index set $S \in \{F, J\}$, define the coefficient of the noise at index j by

$$c_{S,j}(a) = \begin{cases} v_j(a) \prod_{\substack{k \in S \\ k > j}} q_k(a), & j \in S, \\ 0, & j \notin S. \end{cases} \tag{17}$$

An empty product equals one. Recursively applying equation (15) gives

$$\begin{aligned} \mathbf{u}_\ell^\top \mathbf{x}_{\text{full}} &= \sum_{j=0}^{n} c_{F,j}(a_\ell)\epsilon_{j,\ell}, \\ \mathbf{u}_\ell^\top \mathbf{x}_{\text{short}} &= \sum_{j=0}^{n} c_{J,j}(a_\ell)\epsilon_{j,\ell}. \end{aligned} \tag{18}$$

For $j \in J$, the full schedule's coefficient contains the additional factors from the omitted steps. Thus

$$c_{F,j}(a) = c_{J,j}(a)P_j(a), \qquad P_j(a) = \prod_{\substack{k \in F \setminus J \\ k > j}} q_k(a). \tag{19}$$

F.1 Clean-cache refinement

Derivation of $\mathbf{x}_{\text{clean}}(\sigma)$ in equation (9). For the schedule estimates in equation (18), write

$$\mathbf{e} = \mathbf{x}_{\text{short}} - \mathbf{x}_{\text{full}}. \tag{20}$$

For the clean-cache step, we assume an additional standard Gaussian vector $\boldsymbol{\epsilon}_i$, with $i = n + 1$, independent of the denoising noises. We fix the DiT conditioning label at $t_i = 0$, treat the noise level as a variable $\sigma \in [0, 1]$, and write the updates in equations (1)–(3) as

$$\tilde{\mathbf{x}}_{\text{short}}(\sigma) = (1 - \sigma)\mathbf{x}_{\text{short}} + \sigma\boldsymbol{\epsilon}_i, \qquad \mathbf{x}_{\text{clean}}(\sigma) = \tilde{\mathbf{x}}_{\text{short}}(\sigma) - \sigma f_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{\text{short}}(\sigma), 0), \tag{21}$$

Proof of the risk expansion in equation (10). Define $\mathbf{M}(\sigma) = (1 - \sigma)(\mathbf{I}_d - \sigma\mathbf{A})$. Substituting the linear predictor into equation (21) gives

$$\mathbf{r}(\sigma) = \mathbf{M}(\sigma)\mathbf{x}_{\text{short}} - \mathbf{x}_{\text{full}} + \sigma(\mathbf{I}_d - \sigma\mathbf{A})\boldsymbol{\epsilon}_i. \tag{22}$$

The independence and standard Gaussian law of $\boldsymbol{\epsilon}_i$ imply that, conditional on the two schedule outputs, the last term has zero mean and expected squared norm $\sigma^2\|\mathbf{I}_d - \sigma\mathbf{A}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm. Hence

$$R_{\text{clean}}(\sigma) = \mathbb{E}\,\|\mathbf{M}(\sigma)\mathbf{x}_{\text{short}} - \mathbf{x}_{\text{full}}\|_2^2 + \sigma^2\|\mathbf{I}_d - \sigma\mathbf{A}\|_F^2. \tag{23}$$

Since $\mathbf{M}(\sigma) = \mathbf{I}_d - \sigma(\mathbf{I}_d + \mathbf{A}) + \sigma^2\mathbf{A}$, expanding equation (23) gives

$$R_{\text{clean}}(\sigma) = \mathbb{E}\|\mathbf{e}\|_2^2 - 2g\sigma + \beta_2\sigma^2 + \beta_3\sigma^3 + \beta_4\sigma^4, \tag{24}$$

where

$$\begin{aligned} \beta_2 &= \mathbb{E}\|(\mathbf{I}_d + \mathbf{A})\mathbf{x}_{\text{short}}\|_2^2 + 2\mathbb{E}[\mathbf{e}^\top\mathbf{A}\mathbf{x}_{\text{short}}] + d, \\ \beta_3 &= -2\mathbb{E}\big[((\mathbf{I}_d + \mathbf{A})\mathbf{x}_{\text{short}})^\top\mathbf{A}\mathbf{x}_{\text{short}}\big] - 2\,\mathrm{tr}(\mathbf{A}), \\ \beta_4 &= \mathbb{E}\|\mathbf{A}\mathbf{x}_{\text{short}}\|_2^2 + \|\mathbf{A}\|_F^2. \end{aligned} \tag{25}$$

Thus equation (10) holds with

$$\rho(\sigma) = \beta_2\sigma^2 + \beta_3\sigma^3 + \beta_4\sigma^4, \qquad k = |\beta_2| + |\beta_3| + \beta_4. \tag{26}$$

All moments are finite because the outputs are finite linear combinations of Gaussian noise. If $\mathbf{A} \neq \mathbf{0}$, then $\beta_4 > 0$. If $\mathbf{A} = \mathbf{0}$, then $\beta_2 = \mathbb{E}\|\mathbf{x}_{\text{short}}\|_2^2 + d > 0$. In either case, $k > 0$, and $|\rho(\sigma)| \le k\sigma^2$ for $\sigma \in [0, 1]$.

Proof that $g > 0$ for equation (11). The shared noise assumptions and orthonormality of the eigenvectors imply that the $\epsilon_{j,\ell}$ have zero mean and unit variance and are independent across levels. Substituting equation (18) into g and taking the expectation eliminates cross terms between distinct noise indices. Since $c_{J,j} = 0$ for $j \notin J$, equation (19) gives

$$\begin{aligned} g &= \mathbb{E}\big[\mathbf{e}^\top(\mathbf{I}_d + \mathbf{A})\mathbf{x}_{\text{short}}\big] \\ &= \sum_{\ell=1}^{d}(1+a_\ell)\sum_{j\in J} c_{J,j}(a_\ell)^2[1 - P_j(a_\ell)] > 0. \end{aligned} \tag{27}$$

Every summand is nonnegative. For every ℓ, the initial noise term is strictly positive because $c_{J,0}(a_\ell) > 0$ and $P_0(a_\ell) < 1$, as the full schedule contains at least one additional level.

Proof of the unique full schedule minimizer. For the full schedule, replace $\mathbf{x}_{\text{short}}$ by $\mathbf{x}_{\text{full}}$ in equation (23). The risk is zero at $\sigma = 0$. For $\sigma \in (0, 1]$, the noise term $\sigma^2\|\mathbf{I}_d - \sigma\mathbf{A}\|_F^2$ is strictly positive because every eigenvalue of $\mathbf{I}_d - \sigma\mathbf{A}$ is positive. The other term is nonnegative, so $\sigma^\star_{\text{full}} = 0$ is the unique minimizer.

Proof of the minimizer bound in equation (11). The risk is a polynomial, so it attains a minimum on $[0, 1]$. For the shortened schedule, equation (27) gives $g > 0$. Set $\delta = \min\{1, g/k\}$. Because $k\delta \le g$, the remainder bound gives

$$R_{\text{clean}}(\delta) \le R_{\text{clean}}(0) - 2g\delta + k\delta^2 \le R_{\text{clean}}(0) - g\delta. \tag{28}$$

For every $0 \le \sigma \le \delta/3$, the lower remainder bound $\rho(\sigma) \ge -k\sigma^2$ gives

$$\begin{aligned} R_{\text{clean}}(\sigma) &\ge R_{\text{clean}}(0) - 2g\sigma - k\sigma^2 \\ &\ge R_{\text{clean}}(0) - \frac{7}{9}g\delta > R_{\text{clean}}(\delta). \end{aligned} \tag{29}$$

No minimizer lies in $[0, \delta/3]$. This proves equation (11) and shows that every shortened schedule minimizer has strictly lower risk than $\sigma = 0$.

Derivation of the toy curve in Figure 8a. For the plotted instance, choose $\mathbf{A} = a\mathbf{I}_d$, a full schedule $(1, s_1, s_2, s_3)$, and a shortened schedule $(1, s_1)$. We evaluate the risk conditional on the initial vector $\boldsymbol{\epsilon}_0 = \mathbf{z}$. Both schedules use equation (15). The full schedule uses independent standard Gaussian vectors $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \boldsymbol{\epsilon}_3$ after initialization. For this plot, we instead assume that both schedules share $\boldsymbol{\epsilon}_3$ at their final denoising step. The shortened schedule's clean-cache pass uses fresh standard Gaussian noise independent of the denoising noises. Appendix F.3 gives the numerical parameters.

For this conditional risk, we compute g from the two schedule outputs. Write $q(s) = (1-s)(1-as)$ and $b(s) = s(1-as)$ for the scalar coefficients of equation (15), and define

$$c_{\text{short}} = (1-a)q(s_1), \qquad c_{\text{full}} = (1-a)q(s_1)q(s_2)q(s_3). \tag{30}$$

Recursively applying equation (15) at each denoising step gives

$$\begin{aligned} \mathbf{x}_{\text{short}} &= c_{\text{short}}\mathbf{z} + b(s_1)\boldsymbol{\epsilon}_3, \\ \mathbf{x}_{\text{full}} &= c_{\text{full}}\mathbf{z} + b(s_1)q(s_2)q(s_3)\boldsymbol{\epsilon}_1 + b(s_2)q(s_3)\boldsymbol{\epsilon}_2 + b(s_3)\boldsymbol{\epsilon}_3. \end{aligned} \tag{31}$$

Independence of $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \boldsymbol{\epsilon}_3$ and their standard Gaussian moments give

$$\frac{g}{d} = (1+a)\left[c_{\text{short}}(c_{\text{short}} - c_{\text{full}})\frac{\|\mathbf{z}\|_2^2}{d} + b(s_1)\big(b(s_1) - b(s_3)\big)\right]. \tag{32}$$

The parameters in Appendix F.3 give $c_{\text{short}} > c_{\text{full}} > 0$ and $b(s_1) > b(s_3) > 0$, so $g > 0$. The same risk expansion and minimizer bound argument apply to this conditional risk.

To obtain the plotted function, set $m(\sigma) = (1-\sigma)(1-a\sigma)$. Substituting the two outputs into equation (23) and taking the expectation gives

$$\begin{aligned} \overline{R}_{\text{clean}}(\sigma) &= \frac{1}{d}\mathbb{E}\big[\|\mathbf{r}(\sigma)\|_2^2 \mid \boldsymbol{\epsilon}_0 = \mathbf{z}\big] \\ &= \frac{\|\mathbf{z}\|_2^2}{d}\big(m(\sigma)c_{\text{short}} - c_{\text{full}}\big)^2 + \big(m(\sigma)b(s_1) - b(s_3)\big)^2 \\ &\quad + \big(b(s_1)q(s_2)q(s_3)\big)^2 + \big(b(s_2)q(s_3)\big)^2 + \sigma^2(1-a\sigma)^2. \end{aligned} \tag{33}$$

F.2 V/O TRUNCATED SVD

Additional matrix assumptions for equations (13)–(14). We additionally assume $0 < \operatorname{rank}(\mathbf{A}) < d$. Define $q = \operatorname{rank}(\mathbf{A})$, and let $a_{\min}^{+}$ be the smallest positive eigenvalue of $\mathbf{A}$. We assume $0 < b < \min\{a_{\min}^{+}/2, b_{\text{schedule}}\}$. The existence of $b_{\text{schedule}} > 0$ is proved below when deriving $\kappa > 0$. We use the reference $\mathbf{x}_{\text{full}} = \mathbf{x}_{\text{full}}(d)$ and set the clean-cache noise level to zero in this calculation.

Derivation of the truncated model in equation (12). Set $\mathbf{W} = \mathbf{A} - b\mathbf{I}_d$ and $\mathbf{A}_r = b\mathbf{I}_d + \widehat{\mathbf{W}}_r$ as in equation (12). Since $\mathbf{A}$ is positive semidefinite and has rank q, it has q positive eigenvalues a_ℓ and $d-q$ zero eigenvalues. The assumed bound $b < a_{\min}^{+}/2$ then gives q eigenvalues $w_\ell = a_\ell - b > b$ of $\mathbf{W}$ and $d-q$ eigenvalues equal to $-b$. Its singular values are $|w_\ell|$, so the positive eigenvalues come first when ordered by nonincreasing magnitude. Using the same ordered orthonormal eigenvectors $\mathbf{u}_\ell$ for every rank, the truncated SVD gives

$$\mathbf{W} = \sum_{\ell=1}^{d} w_\ell \mathbf{u}_\ell \mathbf{u}_\ell^\top, \qquad \widehat{\mathbf{W}}_r = \sum_{\ell=1}^{r} w_\ell \mathbf{u}_\ell \mathbf{u}_\ell^\top. \tag{34}$$

In this eigenbasis, $\mathbf{A}_r$ has eigenvalue $a_\ell = b + w_\ell$ for $\ell \le r$ and b otherwise. Since $0 \le a_\ell < 1$ and $0 < b < 1$, this proves $\mathbf{0} \preceq \mathbf{A}_r \prec \mathbf{I}_d$ for every rank.

Applying the shared derivation to $\mathbf{A}_r$. The truncated matrix has eigenvalue $a_{r,\ell} = a_\ell$ for $\ell \le r$ and $a_{r,\ell} = b$ otherwise. Replacing a_ℓ by $a_{r,\ell}$ in equation (18) gives $\mathbf{x}_{\text{full}}(r)$ and $\mathbf{x}_{\text{short}}(r)$. The reference $\mathbf{x}_{\text{full}}$ retains the original eigenvalues a_ℓ.

Derivation of the risk in equation (13). The shared noise assumptions imply that the noise coordinates have zero mean and unit variance and are independent across indices. Squaring the difference from the fixed reference and taking its expectation gives

$$R_{\text{svd}}(r) = \sum_{\ell=1}^{d} \sum_{j=0}^{n} \left[c_{J,j}(a_{r,\ell}) - c_{F,j}(a_\ell)\right]^2. \tag{35}$$

Proof of the risk expansion in equation (14). In direction $\mathbf{u}_\ell$, retaining the corresponding component of $\mathbf{W}$ leaves the model eigenvalue at a_ℓ, while discarding it replaces that eigenvalue by b. Denote the expected squared errors for these two choices by e_ℓ^{keep} and e_ℓ^{drop}. Equation (35) gives

$$\begin{aligned} e_\ell^{\text{keep}} &= \sum_{j=0}^{n} [c_{J,j}(a_\ell) - c_{F,j}(a_\ell)]^2, \\ e_\ell^{\text{drop}} &= \sum_{j=0}^{n} [c_{J,j}(b) - c_{F,j}(a_\ell)]^2. \end{aligned} \tag{36}$$

The total risk is therefore

$$R_{\text{svd}}(r) = \sum_{\ell=1}^{r} e_\ell^{\text{keep}} + \sum_{\ell=r+1}^{d} e_\ell^{\text{drop}}. \tag{37}$$

For $\ell > q$, the original eigenvalue is zero and the noise law is the same in every direction. Both e_ℓ^{keep} and e_ℓ^{drop} are therefore constant across these directions. Define

$$\begin{aligned} \eta_\ell &= e_\ell^{\text{drop}} - e_\ell^{\text{keep}}, \quad 1 \le \ell \le q, \\ \kappa &= e_\ell^{\text{keep}} - e_\ell^{\text{drop}}, \quad \ell > q. \end{aligned} \tag{38}$$

At rank q, the first q directions are retained and the rest are discarded. For $r < q$, discarding the directions $r < \ell \le q$ adds $\sum_{r<\ell\le q} \eta_\ell$ to this risk. For $r \ge q$, retaining the additional $r-q$ directions adds $(r-q)\kappa$. This proves equation (14).

Proof that $\eta_\ell > 0$ and $\kappa > 0$ in equation (14). For $j \in J$, write $C_j(a) = c_{J,j}(a)$, so equation (19) gives $c_{F,j}(a) = C_j(a)P_j(a)$. For $a \in [0, 1)$, $C_j(a) > 0$, $0 < P_j(a) \leq 1$, and $C_j(a)$ is strictly decreasing because each factor $1 - s_k a$ is positive and strictly decreasing.

For $\ell \leq q$, $a_\ell > b$ gives

$$0 \leq C_j(a_\ell)[1 - P_j(a_\ell)] < C_j(b) - C_j(a_\ell)P_j(a_\ell). \tag{39}$$

Squaring shows that each term with $j \in J$ in e_ℓ^{keep} is smaller than its counterpart in e_ℓ^{drop}. Terms with $j \notin J$ are equal because $c_{J,j} = 0$ and the reference is unchanged. Thus $e_\ell^{\text{keep}} < e_\ell^{\text{drop}}$, proving $\eta_\ell > 0$.

For $\ell > q$, $a_\ell = 0$. Writing κ as a function of b and substituting equation (36) into equation (38) gives

$$\kappa(b) = \sum_{j \in J} \big\{ C_j(0)^2[1 - P_j(0)]^2 - [C_j(b) - C_j(0)P_j(0)]^2 \big\}. \tag{40}$$

We have $\kappa(0) = 0$, and differentiating gives

$$\kappa'(0) = -2 \sum_{j \in J} C_j'(0) C_j(0)[1 - P_j(0)] > 0. \tag{41}$$

Indeed, $C_j'(0) = -C_j(0) \sum_{k \in J,\, k \geq j} s_k < 0$. Every contribution to $\kappa'(0)$ is nonnegative, and the term at $j = 0$ is positive because the shortened schedule omits at least one later step, giving $P_0(0) < 1$. Since $\kappa(b)$ is a polynomial, it is positive throughout some interval $0 < b < b_{\text{schedule}}$. Choosing b in this interval and below half the smallest positive eigenvalue of $\mathbf{A}$ ensures both the required singular value ordering and $\kappa > 0$.

Proof of the unique shortened schedule minimizer. Equation (14) and the strict positivity of η_ℓ and κ give the unique minimizer $r^\star_{\text{short}} = q < d$.

Proof of the unique full schedule minimizer. Using the full schedule coefficients gives

$$\mathbb{E}\|\mathbf{x}_{\text{full}}(r) - \mathbf{x}_{\text{full}}\|_2^2 = \sum_{\ell=r+1}^{d} \sum_{j=0}^{n} [c_{F,j}(b) - c_{F,j}(b + w_\ell)]^2. \tag{42}$$

For each removed direction, the final noise coefficient differs by $v_n(b) - v_n(b + w_\ell) = s_n^2 w_\ell \neq 0$. Hence every $r < d$ gives positive risk, while $r = d$ gives zero risk. Thus the unique full schedule minimizer is d.

Verification of the toy curve in Figure 8b. Use the full schedule $(1, s_1, s_2, s_3)$ and shortened schedule $(1, s_3)$, with the shared indexed standard Gaussian noises defined above. For integers $0 < k < q < d$ and $b < h < 1 - b$, define the fixed matrix $\mathbf{W} = \operatorname{diag}(w_1, \ldots, w_d)$ by

$$w_\ell = \begin{cases} h, & 1 \leq \ell \leq k, \\ b + (h - b)\left(\dfrac{q - \ell + 1/2}{q - k}\right)^{1/3}, & k < \ell \leq q, \\ -b, & q < \ell \leq d. \end{cases} \tag{43}$$

The magnitudes are nonincreasing, and every positive eigenvalue exceeds the magnitude b of the negative eigenvalues. We retain the first r coordinate directions in the SVD of $\mathbf{W}$, including when equal singular values occur. All eigenvalues of $\mathbf{A}_r$ lie in $[0, b + h] \subset [0, 1)$ for every r.

This spectrum gives $\operatorname{rank}(\mathbf{A}) = q$. The positive eigenvalues of $\mathbf{W}$ give $\eta_\ell > 0$ for $\ell \leq q$, and Appendix F.3 verifies $\kappa > 0$ for the plotted parameters.

F.3 Plotting parameters

Clean-cache curve. Figure 8a plots equation (33) using $a = 0.1$, $\|\mathbf{z}\|_2^2/d = 7.57$, full denoising levels $(1, 0.131, 0.015, 0.0033)$, and shortened levels $(1, 0.131)$. The plotted interval is $\sigma \in [0, 0.05]$.

SVD curve. Figure 8b plots $\overline{R}_{\mathrm{svd}}(r) = R_{\mathrm{svd}}(r)/d$ from equation (35), with the weights in equation (43). We use $d = 500$, $b = 0.05$, $h = 0.534$, $k = 430$, and $q = 460$. The full denoising levels are $(1, 15/16, 5/6, 5/8)$ and the shortened levels are $(1, 5/8)$, one of the two-step schedules in Appendix A.2. Substituting these parameters into equation (40) gives $\kappa \approx 0.0209 > 0$. We plot every integer rank from k to d against $p = r/d$.

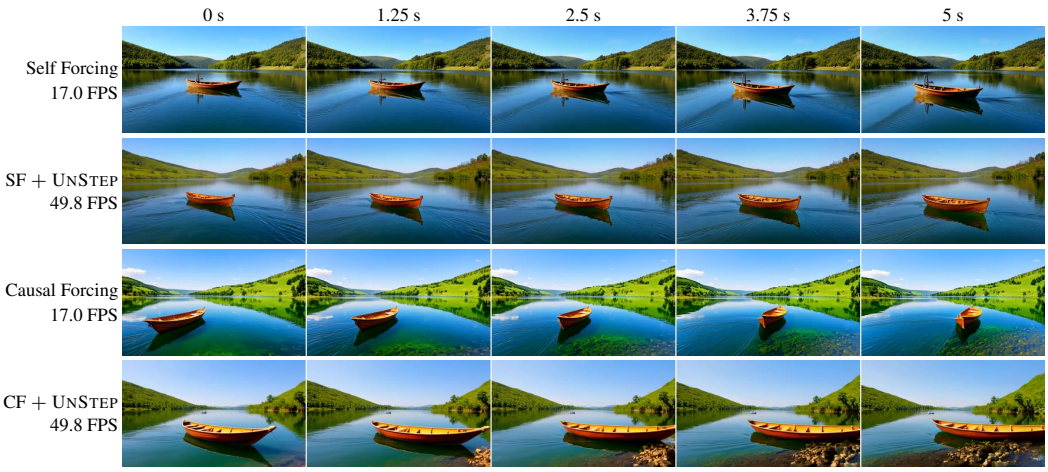
DiT curves. For each sweep value, we average each schedule's latent tensors over five paired runs, giving vectors $\overline{\mathbf{x}}_{\mathrm{full}}$ and $\overline{\mathbf{x}}_{\mathrm{short}}$ with d_{lat} coordinates. The plotted MSE is

$$\frac{\|\overline{\mathbf{x}}_{\mathrm{short}} - \overline{\mathbf{x}}_{\mathrm{full}}\|_2^2}{d_{\mathrm{lat}}}. \tag{44}$$

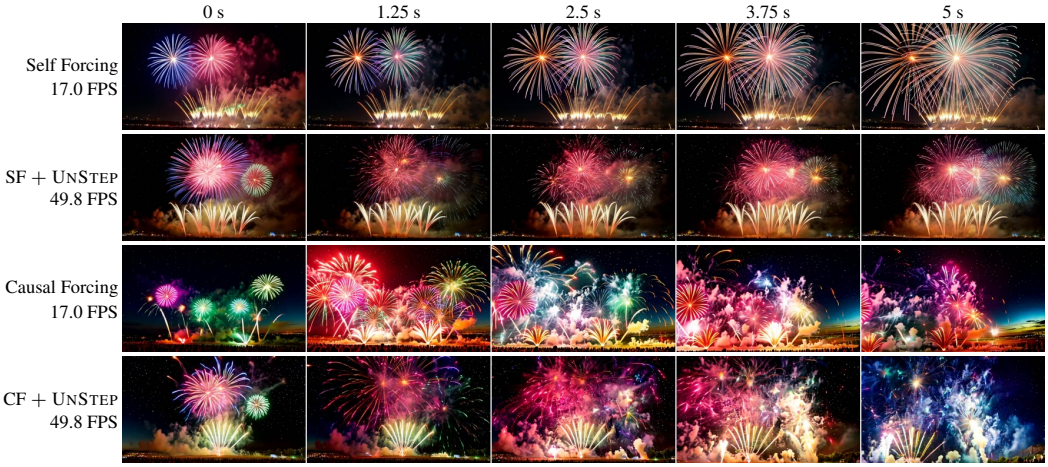
In both sweeps, the full schedule reference uses $\sigma = 0$ and full V/O retention ($p = 1$). For the shortened schedule, we sweep σ with $p = 1$, then sweep p with $\sigma = 0.02$, following the clean-cache then SVD order in Table 1. Both use the denoising noise levels specified in Appendix A.2 and are shown over the same horizontal ranges as their corresponding toy curves.

G VISUALIZATIONS

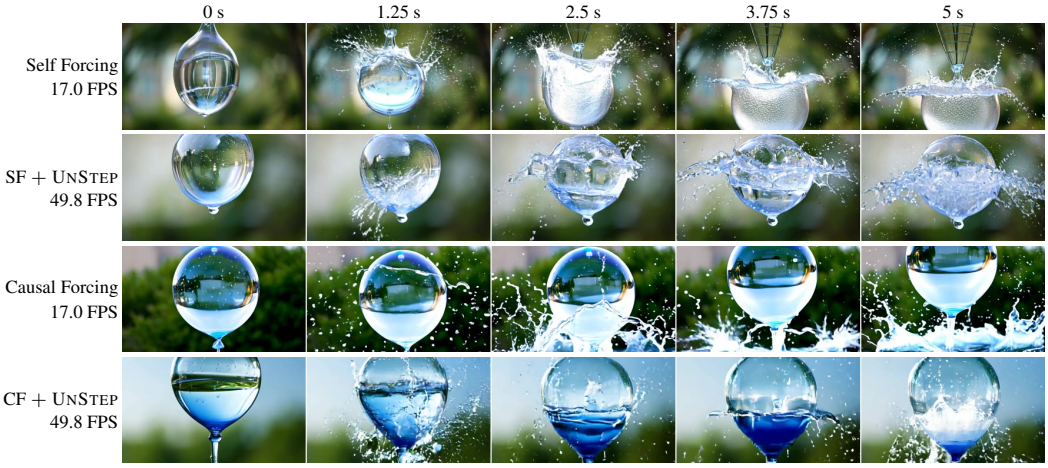
For the text-to-video examples below, we compare Self Forcing and Causal Forcing with the same checkpoints wrapped by the default two-step UNSTEP configuration. Every comparison uses the same VBench entry and seed across all four rows. We show the original models at 17.0 FPS and their UNSTEP variants at 49.8 FPS. Each row contains frames 0, 20, 40, 60, and 80 from an 81-frame clip, corresponding to 0, 1.25, 2.5, 3.75, and 5 seconds. Every clip is generated from the extended prompt. The first example shows its full extended prompt, while the remaining examples show only the original VBench prompts.



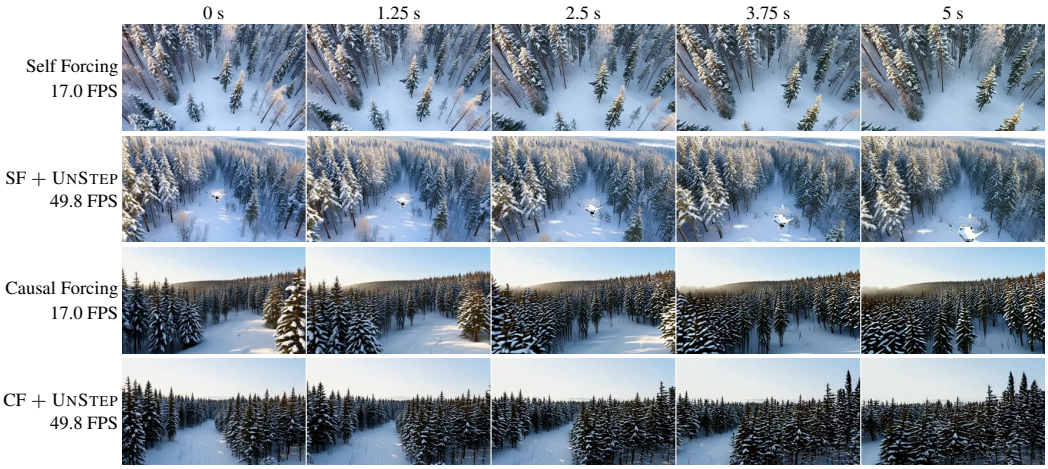
A serene, picturesque scene of a small wooden boat gently sailing across a calm, tranquil lake under a clear blue sky. The boat moves smoothly with the gentle breeze, creating only minimal ripples on the water's surface. The background showcases lush green hills and a few scattered trees reflecting in the crystal-clear water. The camera focuses on the boat from a medium distance, capturing the peaceful atmosphere of the lake. The scene remains static, emphasizing the calmness and tranquility of the environment.



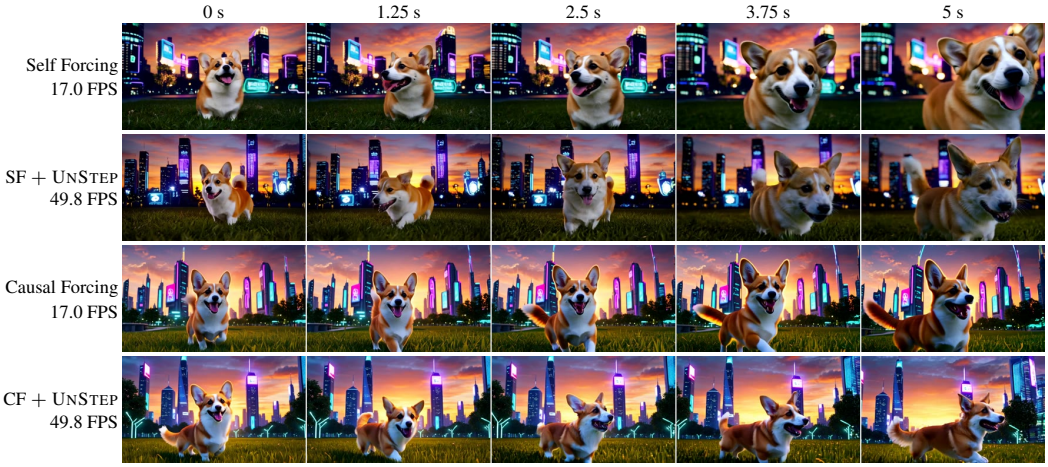
Fireworks.



Balloon full of water exploding in extreme slow motion.



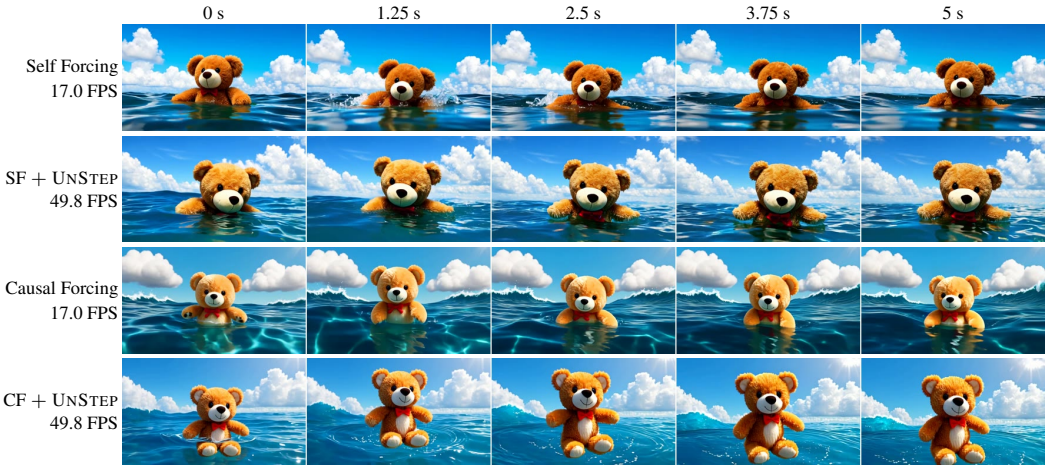
a drone flying over a snowy forest.



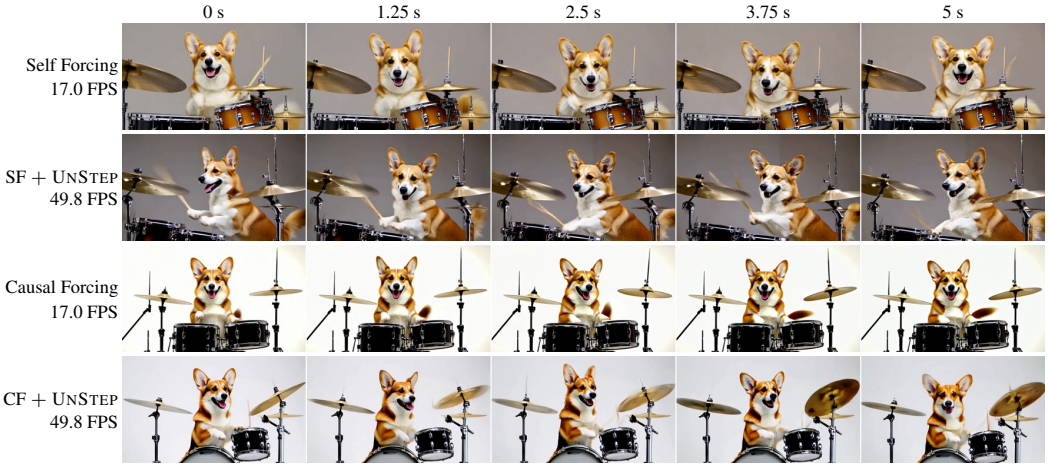
A cute happy Corgi playing in park, sunset, in cyberpunk style



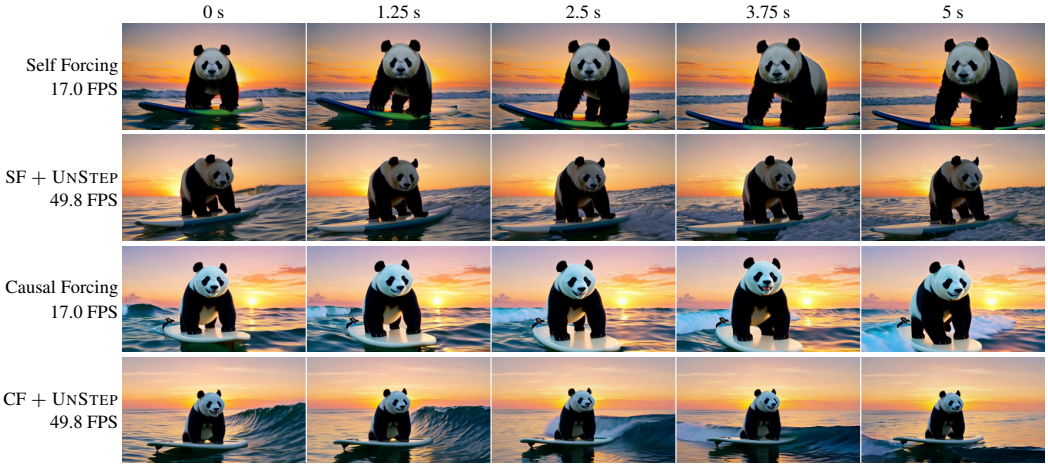
A cat wearing sunglasses and working as a lifeguard at a pool.



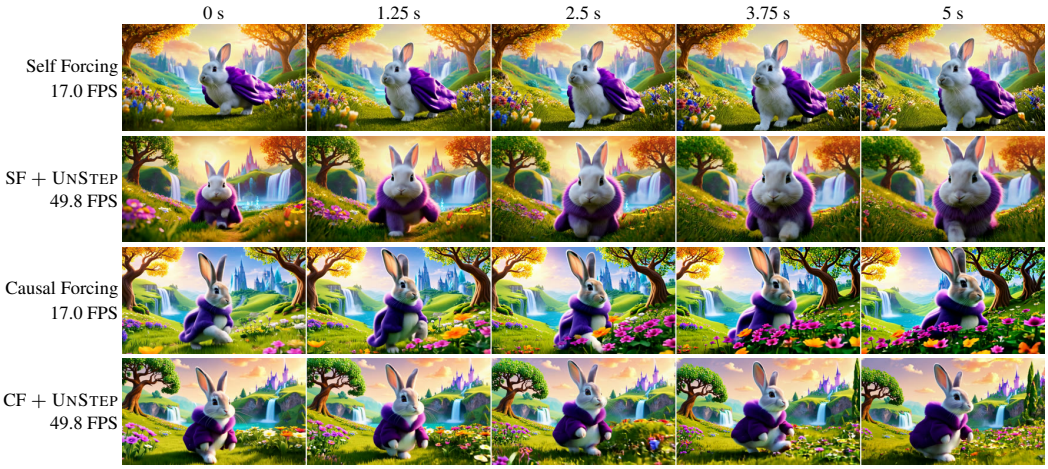
a teddy bear is swimming in the ocean.



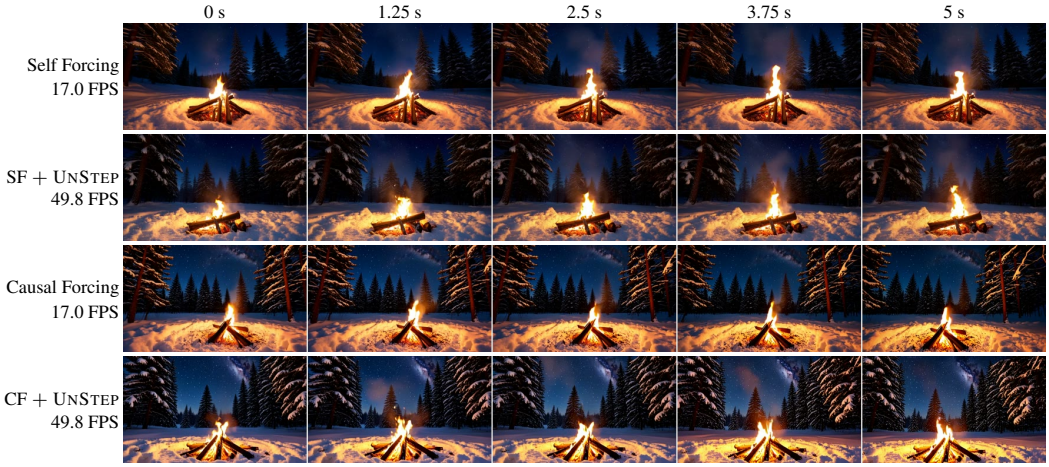
A corgi is playing drum kit.



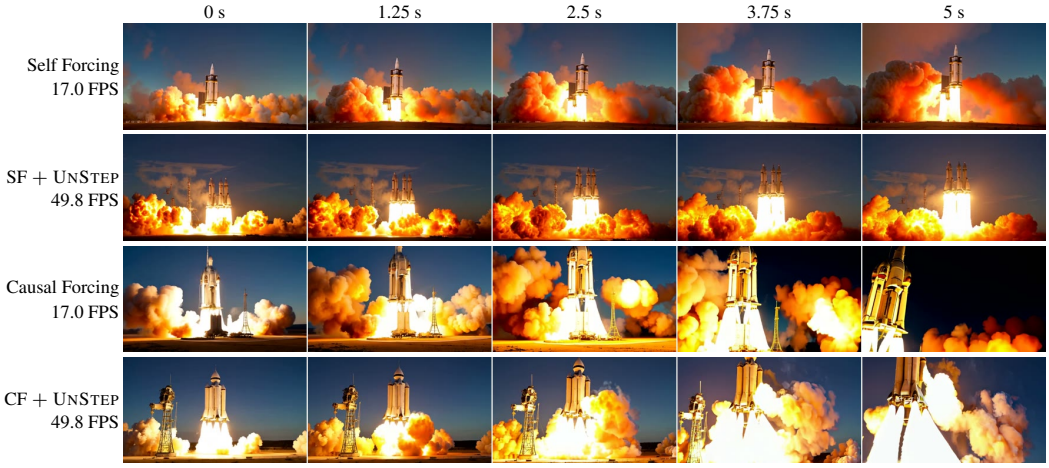
A panda standing on a surfboard in the ocean in sunset.



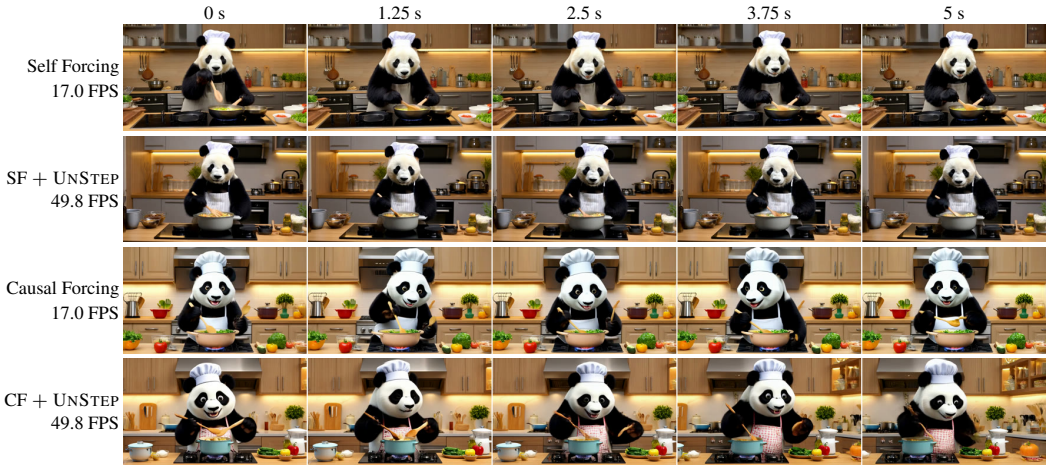
A fat rabbit wearing a purple robe walking through a fantasy landscape.



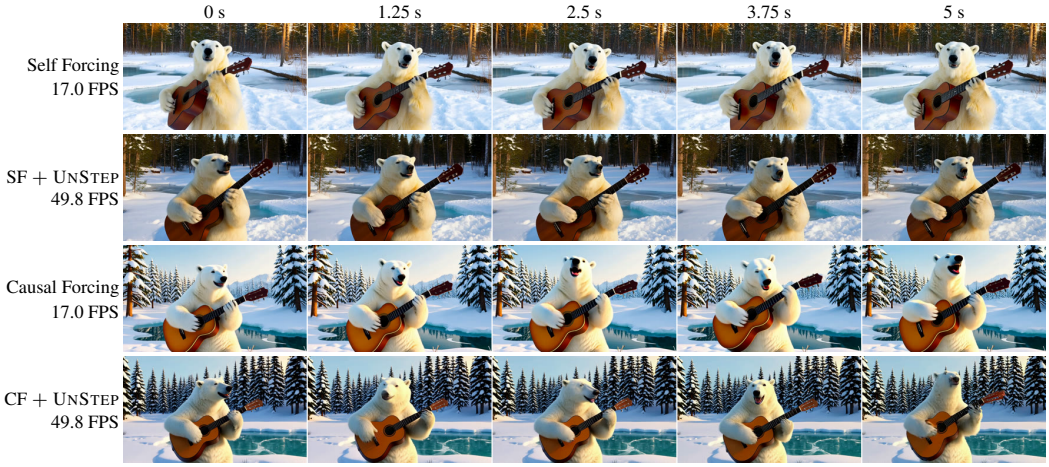
Campfire at night in a snowy forest with starry sky in the background.



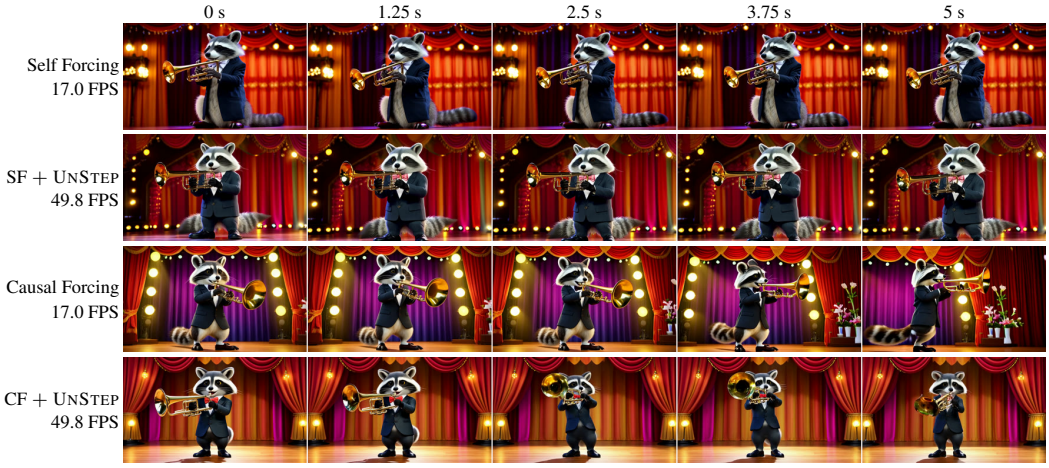
A space shuttle launching into orbit, with flames and smoke billowing out from the engines



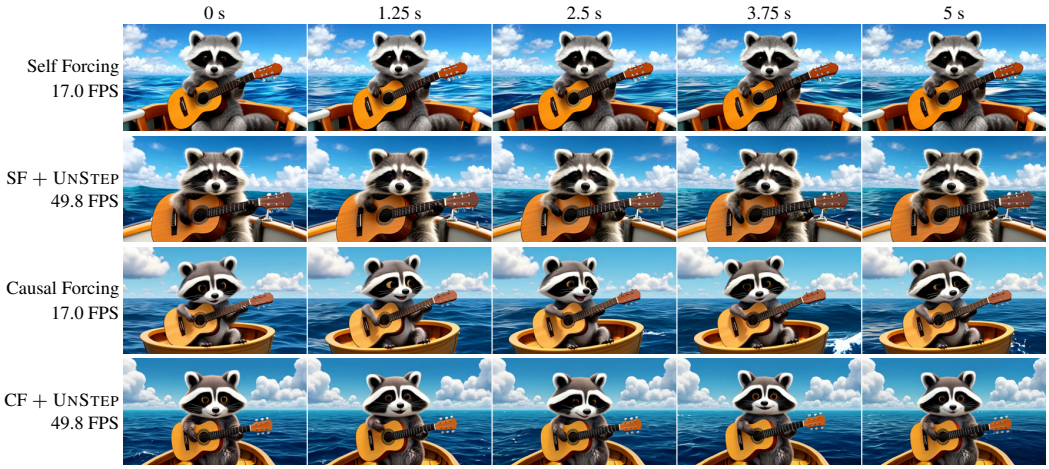
A panda cooking in the kitchen



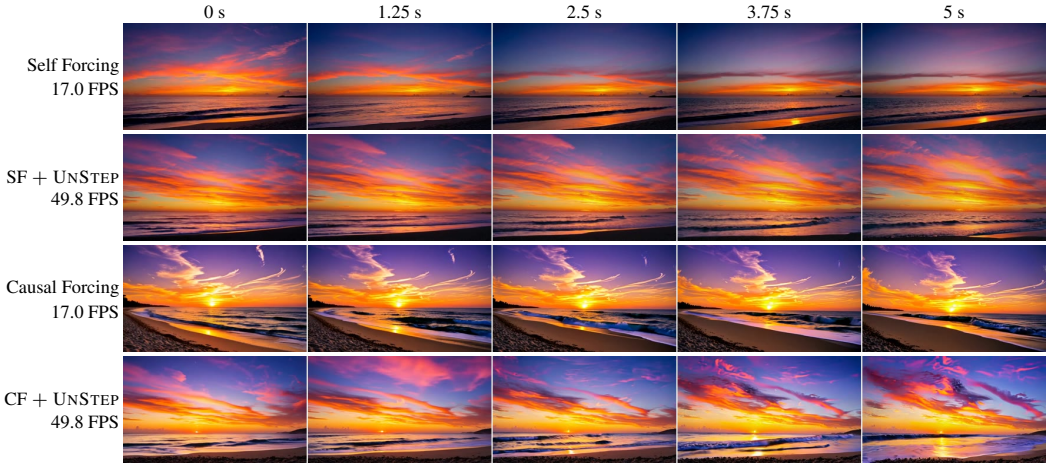
A polar bear is playing guitar



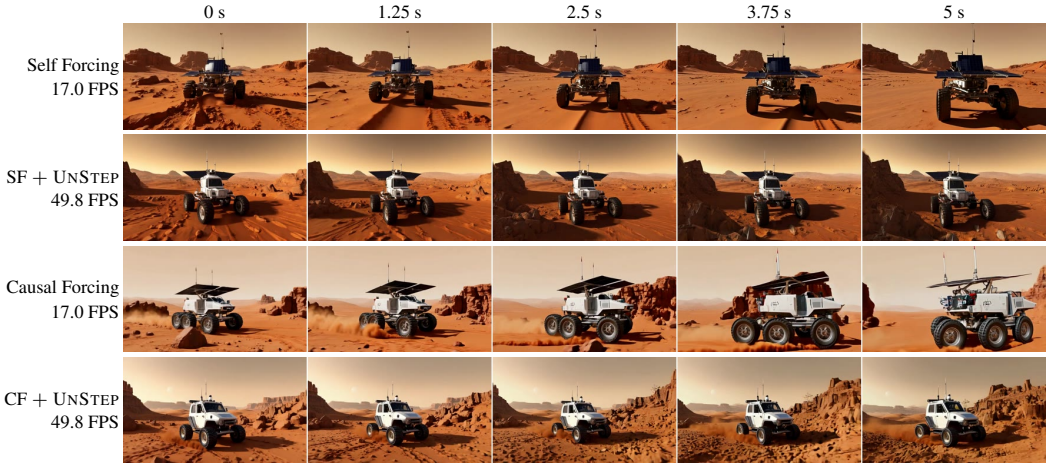
A raccoon dressed in suit playing the trumpet, stage background



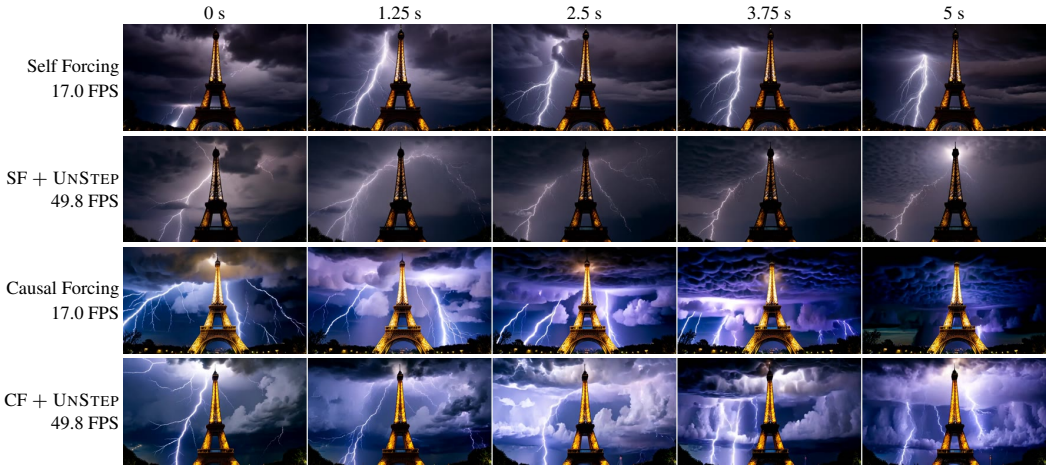
A cute raccoon playing guitar in a boat on the ocean



Sunset time lapse at the beach with moving clouds and colors in the sky.



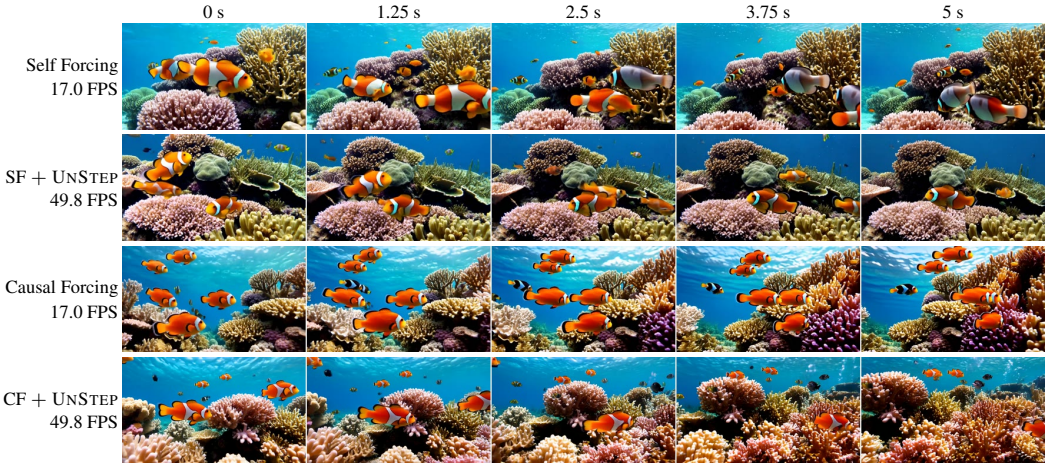
A Mars rover moving on Mars



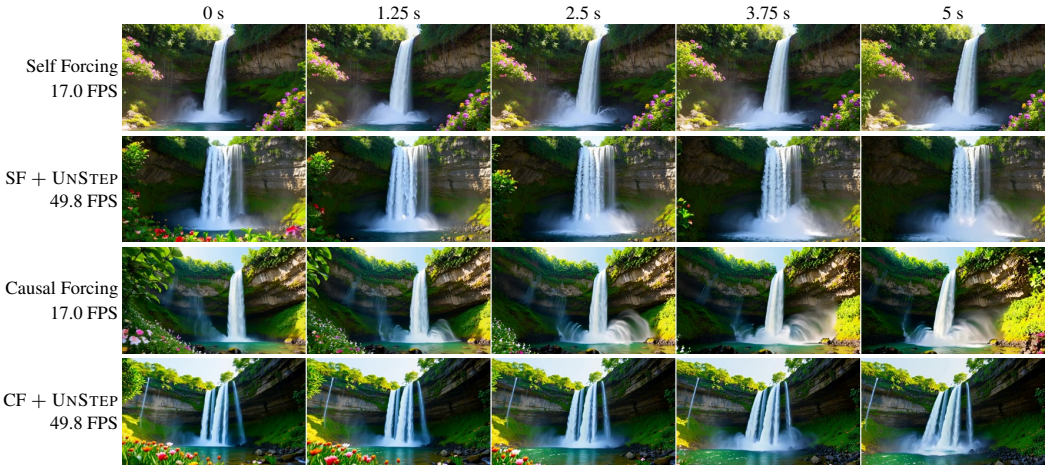
A lightning striking atop of eiffel tower, dark clouds in the sky



A car moving slowly on an empty street, rainy evening



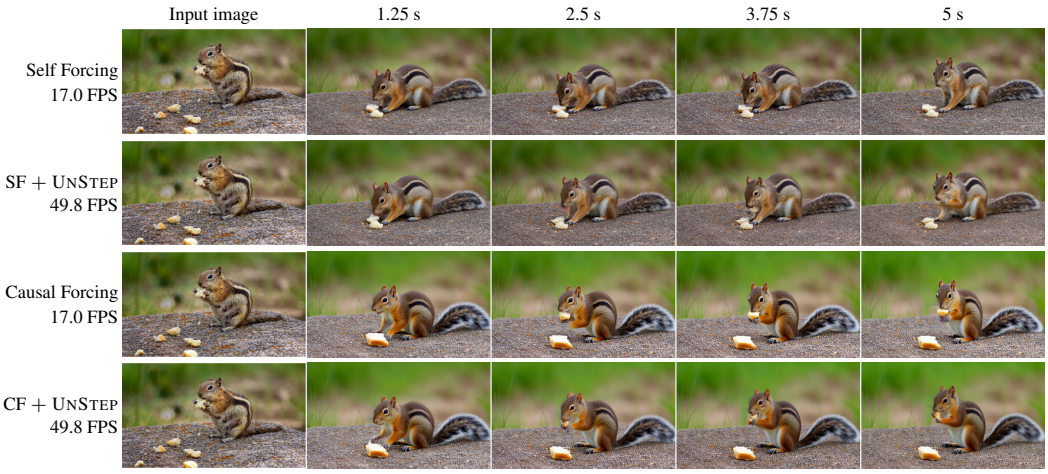
Clown fish swimming through the coral reef



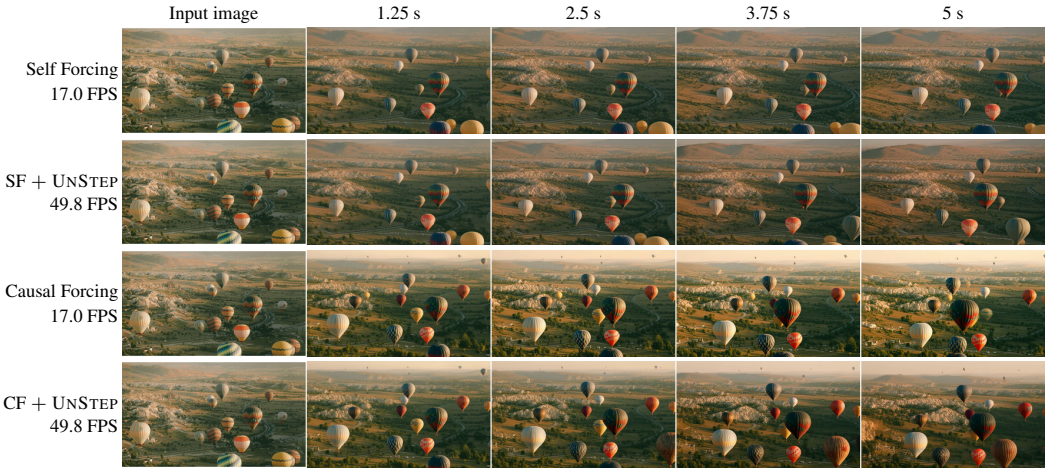
waterfall

IMAGE-TO-VIDEO EXAMPLES

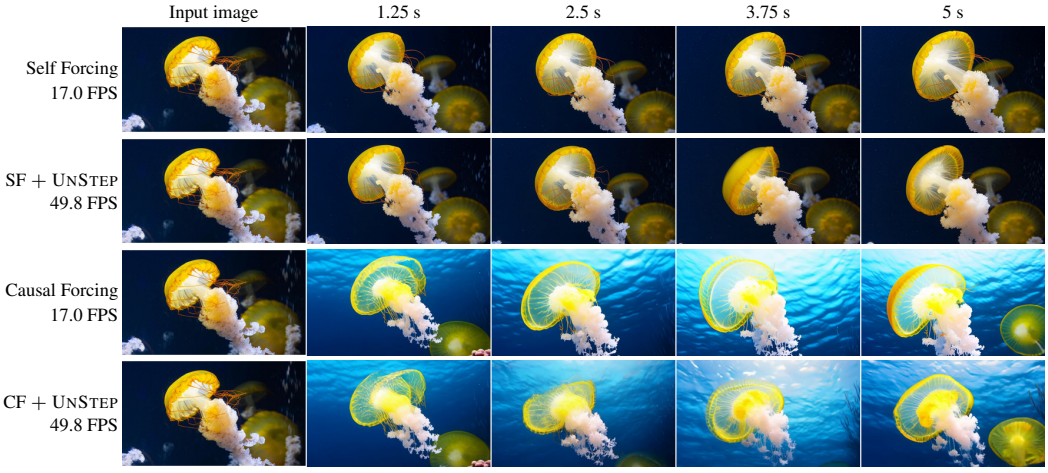
For image-to-video generation, all four variants receive the same reference image, VBench I2V prompt, and initial noise. The first column shows the supplied reference image, and the remaining columns show generated frames.



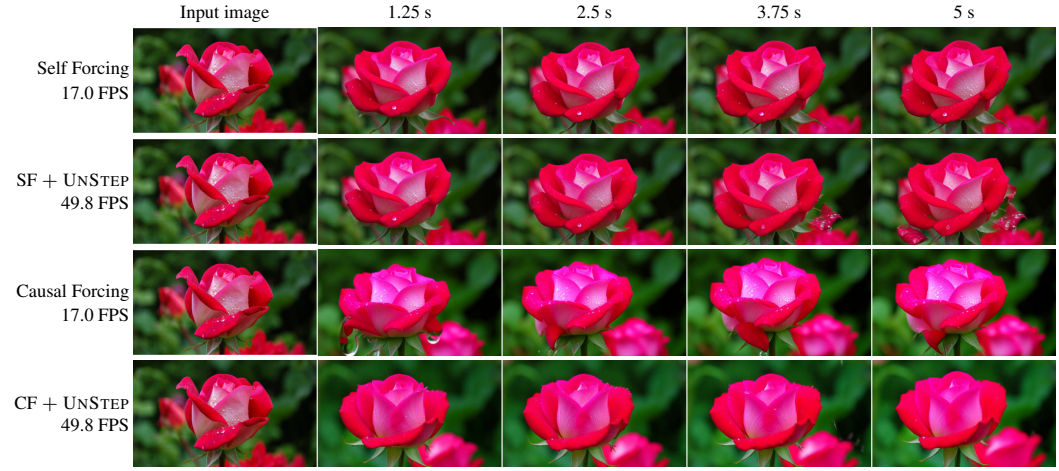
a squirrel sitting on the ground eating a piece of bread



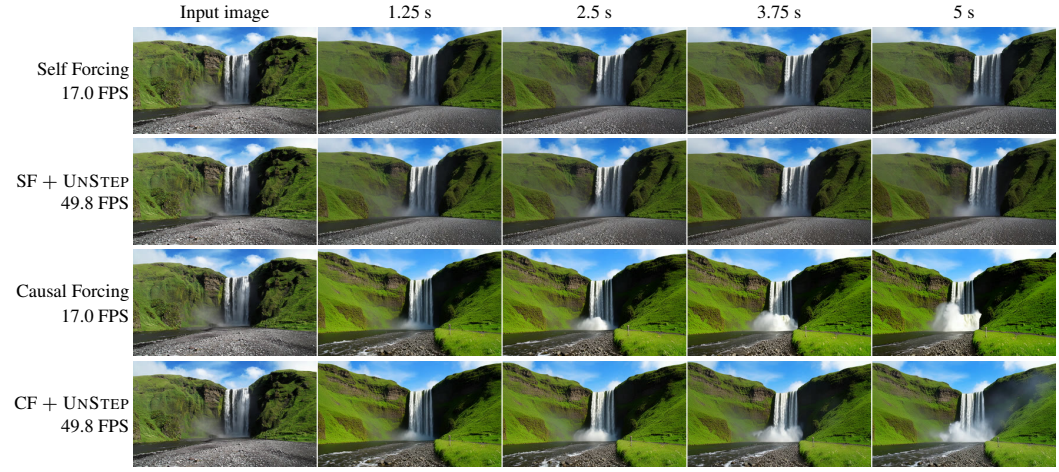
a group of hot air balloons flying over a valley



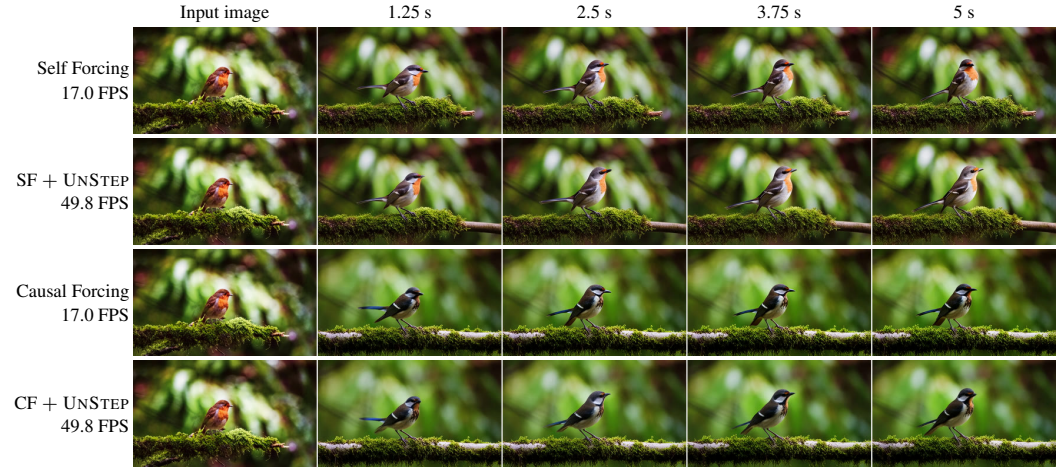
a yellow and white jellyfish is floating in the ocean



a close-up of a pink rose with water droplets on it



a large waterfall in the middle of a lush green hillside



a small bird sits on a moss covered branch