IN-FLIGHT KV CACHE WITH CLEAN ANCHORS FOR FASTER AUTOREGRESSIVE VIDEO DIFFUSION

Anonymous authors
Paper under double-blind review



Figure 1: **FlashForward redesigns the autoregressive diffusion generation pipeline to remove heavy cache-update-only forwards, significantly reducing generation latency**. **(a)** Compared with pipelines that reconstruct clean (Self-Forcing (Huang et al., 2025)) or less-noisy (HiAR (Zou et al., 2026)) KV through context encoding that does not advance an output latent, FlashForward substantially reduces latency while preserving generation quality and coherence, even in long videos. **(b)** FlashForward adopts a planner to autoregressively produce sparse auxiliary anchors for two-sided temporal conditioning. A renderer can then run in standard autoregressive form conditioned on these anchors such that every ordinary forward advances one chunk and publishes stage-matched history for later chunks, allowing different denoising stages to run on different GPUs. **(c)** Quality versus latency for 1.3B models on 480p videos, using each method's fastest setup with up to four GPUs. *See the supplementary material for an interactive demo.*

ABSTRACT

Few-step autoregressive video diffusion generates a long video by splitting the video into temporal chunks and generating chunk-by-chunk, each through a short sequence of denoising stages. To memorize chunks that are already generated, previous methods reconstruct a clean or less-noisy key–value (KV) cache by additional forwards to build the cache without advancing an output latent. However, every denoising forward itself already computes the in-flight KV of the current chunk. We introduce FlashForward, which directly reuses this cache to avoid the

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.

heavy cache-update-only model forwards. After the current chunk completes one denoising stage, its stage-specific cache is already available for the next chunk. Assigning one GPU to each stage therefore lets different chunks occupy different stages concurrently. This early availability has a quality cost: the resulting stage-matched history is noisy, causing appearance and motion drift among chunks. To complement it, FlashForward produces sparse auxiliary clean anchor latents before the corresponding region is generated so the generation trajectories can be stabilized by this two-sided conditioning. The two memories operate at different temporal scales: sparse clean anchor KV supplies coarse, long-range two-sided structural guidance, while dense stage-matched history preserves fine, recent evolution. With up to four GPUs, FlashForward runs 1.16–$1.69\times$ faster than HiAR and 1.42–$2.92\times$ faster than Self-Forcing for 16 FPS videos of 20 seconds or longer across 1.3B and 14B backbone scales at 480p and 720p. On VBench, for the 1.3B model at 480p, it achieves higher scores and remains stable at longer durations, demonstrating that FlashForward generates high-quality and temporally consistent videos across durations of 20s, 35s and 65s at a much faster generation speed.

1 INTRODUCTION

Autoregressive video diffusion (Ho et al., 2022; Voleti et al., 2022; Harvey et al., 2022) generates a video one short chunk at a time. Each chunk is produced by generating its latent through a sequence of denoising stages. To memorize generated content, the model stores features from earlier chunks in a key–value (KV) cache for subsequent generation with additional model passes (Gao et al., 2025; Deng et al., 2025; Sand AI et al., 2025). Recent methods reduce the denoising sequence to only a few stages (Yin et al., 2025; Huang et al., 2025; Zheng et al., 2026; Liu et al., 2026b;a), making any additional model forward used only to prepare memory an increasingly important bottleneck.

Existing methods incur this overhead in different ways. Self-Forcing (Huang et al., 2025) completes a chunk and then passes its clean output through the model again to construct the cache. HiAR (Zou et al., 2026) allows different segments to overlap in execution by conditioning on less-noisy rather than clean-endpoint context, but re-encoding at every denoising stage. These additional passes build memory without directly advancing video generation. However, the computation during denoising already establishes an *in-flight KV* for the current chunk. If we directly reuse this cache, we could avoid the heavy cache-update-only model forwards and further reduce the generation latency.

More broadly, we analyze the design choice of the *cross-chunk state contract*, specifying what representation each chunk publishes, at what noise level, and when later chunks may consume it. The contract determines not only the temporal memory available to the model, but also the dependency graph of generation. In previous clean or less-noisy history contracts, the cache-update-only forwards build state without advancing an output latent. In this paper, we propose FlashForward to instead publish an ordinary same-stage cache for reuse at the same denoising stage, making the required state available as part of generation itself. Among the designs in Tab. 1, it is the only memory that is both emitted by an output-advancing forward and available early to pipeline chunks across stage workers.

This early availability has a cost. Stage-matched history comes from unfinished, noisy chunks. Used alone, it can propagate uncertainty in appearance and motion (Chen et al., 2024; Zou et al., 2026). Returning to dense clean state for every completed chunk would restore the heavy cache-update-only work. FlashForward therefore complements this promptly available history with sparse auxiliary anchors distributed across the timeline and planned in advance for a clean two-sided memory.

This availability rule produces a planner–renderer generation graph. The *planner role* produces the auxiliary anchor latents and their clean anchor KV cache bank; the *renderer role* generates the video while reading a local anchor KV block and the stage-matched renderer history bank, as in Fig. 1(b). The two memories divide temporal responsibility: sparse clean anchor KV constrains coarse structure over a longer interval, whereas dense stage-matched renderer history carries fine appearance and motion changes from recent chunks. Only sparse planner blocks require a clean-endpoint cache-extraction forward, and each local anchor window conditions multiple renderer chunks. Hence the efficiency of dense rendering without cache-update-only forwards is preserved.

Table 1: **How memory determines execution** for S denoising stages with noise levels $\sigma_S > \cdots > \sigma_0 = 0$. FlashForward additionally uses sparse clean anchor KV prepared before generation.

Method	Memory noise level	When it becomes usable	Cache-update-only forwards	Chunk execution
Self-Forcing	σ_0	After the chunk is re-encoded	1 per chunk	Serial
N-C-Causal-rCM	$\sigma_1 \to \sigma_0$	After the last denoising stage	0	Serial
HiAR	σ_{k-1}	After each stage-specific re-encoding	S per chunk	Overlapped
FlashForward	$\sigma_k \to \sigma_{k-1}$	Immediately after each denoising stage	0	Overlapped

FlashForward uses a shared generator backbone for both roles. A learned role embedding and role-specific LoRA adapters (Hu et al., 2022) specialize their temporal dependencies while retaining common visual and language knowledge. We train FlashForward in two phases: Packed supervised fine-tuning establishes the planner–renderer graph from real videos; self-rollout distillation distills the guidance and denoising stages, and adapts its few-step generation process to generated context.

We evaluate and compare the latency of FlashForward with other generation pipelines across 1.3B and 14B backbone scales on 480p and 720p videos. For 16 FPS videos of 20 seconds or longer in up to four-GPU settings, FlashForward is 1.16–$1.69\times$ faster than less-noisy history (HiAR) and 1.42–$2.92\times$ faster than clean history (Self-Forcing). We evaluate generation quality for the 1.3B model at 480p under VBench. FlashForward scores 0.838 on Total, surpassing 0.805 for Self-Forcing and 0.821 for HiAR. Furthermore, as the duration increases to 65 seconds, the generation quality remains stable for FlashForward, demonstrating its effectiveness as a faster generation pipeline.

Contributions. Our contribution is one state-availability design, realized as a single causal chain. **(1)** Each ordinary renderer forward publishes stage-matched renderer history while advancing the current chunk, providing state available early enough to induce an inter-chunk wavefront over stage workers without renderer cache-update-only forwards. **(2)** We instantiate a temporal-scale allocation of cross-chunk state: sparse clean anchor KV prepared ahead of each local region supplies coarse long-range structure, while promptly available but noisy and past-only renderer history supplies fine recent evolution. **(3)** We realize this graph with planner and renderer roles sharing a base generator, train it through real-video supervision and self-rollout distillation, and demonstrate latency benefits across model sizes and resolutions together with quality superiority for the 1.3B generator at 480p.

2 Related Work

Few-step autoregressive video diffusion. Video generators combine diffusion or flow objectives with pixel-space, latent, and Transformer backbones (Ho et al., 2020; 2022; Lipman et al., 2023; Singer et al., 2023; Rombach et al., 2022; Blattmann et al., 2023b;a; Peebles & Xie, 2023). Few-step objectives use implicit sampling, distillation, and consistency training (Song et al., 2021; 2023; Salimans & Ho, 2022; Zhai et al., 2024; Cai et al., 2026). Autoregressive video models factor generation into sequential units and generated histories (Weissenborn et al., 2020; Villegas et al., 2023; Weng et al., 2024; Henschel et al., 2025); FlashForward changes what is passed between those units, and with it their execution order.

Cross-chunk memory and execution. We view an autoregressive diffusion method as defining a *cross-chunk state contract*: which representation a chunk publishes, at what noise level, and when that representation becomes available to later chunks. Together with the within-chunk denoising order, this contract determines both the number of cache-update-only forwards and which chunk updates can overlap (Tab. 1). Self-Forcing (Huang et al., 2025) publishes clean KV only after a completed chunk undergoes one cache-update-only forward, so later chunks wait at a serial boundary. The noisy-context variant of Causal-rCM (Zheng et al., 2026) (N-C-Causal-rCM), following Diagonal Distillation (Liu et al., 2026a), reuses KV from the final denoising forward, avoiding that extra update; because the state is published only when the predecessor reaches its final stage, chunks remain serial. HiAR (Zou et al., 2026) conditions each chunk at intermediate stage on its predecessors' less-noisy context, enabling an anti-diagonal pipeline at the cost of per-stage context encoding per consumed chunk. FlashForward instead publishes the KV produced by every ordinary renderer forward to later chunks at the same stage. Each publication advances the current chunk rather than serving only to build memory, and the resulting same-stage dependencies form a wavefront. Because this

stage-matched renderer history is noisy and past-only, FlashForward complements it with sparse clean anchor KV made available before each local region is rendered. Another research direction (Gu et al., 2025; Yang et al., 2026a; Samuel et al., 2026; Ji et al., 2026; Ma et al., 2026; Luo et al., 2026; Yang et al., 2026b) studies how to determine which cached content to retain, compress, merge, reuse, or switch; these policies are largely orthogonal to the state contract.

Plan-then-infill and future-guided generation. Sparse-to-dense video generation first establishes coarse temporal structure and then fills the intervening frames (Ge et al., 2022; He et al., 2022; Harvey et al., 2022; Yin et al., 2023). Xiang et al. (2025) and Ouyang et al. (2026) separate keyframe planning from segment population; FramePack (Zhang et al., 2025) and SneakPeek (Hong et al., 2025) predict anchors and future keyframes ahead of the frames between them; Bendel et al. (2026) and Zhang et al. (2026) use two-sided anchors for interval generation. These works establish sparse planning and two-sided conditioning as effective sources of temporal coherence. FlashForward repurposes this form of conditioning as a coarse-timescale state that complements fine-timescale stage-matched renderer history. Unlike in conventional infilling, its auxiliary anchor latents are conditioning-only. Their clean anchor KV lets the renderer retain promptly available stage-matched renderer history without reverting to dense clean cache-update-only forwards.

Noise schedules and parallel execution. Diffusion Forcing (Chen et al., 2024), FIFO-Diffusion (Kim et al., 2024), Rolling Forcing (Liu et al., 2026b), and Diagonal Distillation (Liu et al., 2026a) assign different noise levels or update times across temporal positions, while Ms. Forcing (Li et al., 2026) adapts computation to the noise level. DistriFusion (Li et al., 2024) and PipeFusion (Fang et al., 2025) parallelize spatial computation within a denoising trajectory. HiAR (Zou et al., 2026) pipelines successive chunks across denoising-stage workers, placing them on an anti-diagonal schedule that it sustains by re-encoding each consumed predecessor at every stage. Its context-only work is therefore paid per stage per consumed chunk, rather than once per chunk as in Self-Forcing. A noise schedule specifies when each chunk is updated, whereas a cross-chunk state rule specifies which earlier representation conditions that update. In FlashForward, the same anti-diagonal follows directly from the same-stage state dependency, and no output chunk is re-encoded.

3 Stage-Matched History with Clean Two-Sided Anchors

FlashForward uses one generator in two roles. The planner first produces a sparse clean plan, and the renderer then generates the video while reading that plan and the recent history left by earlier renderer chunks. The key operation is one ordinary renderer forward: it both advances the current chunk and leaves reusable KV for later renderer chunks during generation.

A concrete generation example. Fig. 2 follows an output of 81 latent frames. The planner generates nine auxiliary clean anchor latents at positions $\{0, 10, \ldots, 80\}$, three anchors at a time. It runs one cache-extraction forward on each completed block to obtain clean anchor KV. The renderer then generates all 81 output positions in 27 three-latent chunks guided by a two-sided clean plan and in-flight memory. For example, the chunk at positions 6–8 reads nearby clean anchor KV at $\{0, 10, 20\}$, which bracket this interior chunk, together with the KV left by recent chunks at its current denoising stage. As each chunk traverses four stages, the next chunk can follow one stage behind, producing the wavefront in panel (c). The complete data flow is therefore: generate each auxiliary anchor block and publish its clean KV, then render the dense video while every renderer forward advances the output and publishes stage-matched renderer history.

Notation. Given a condition c, the target contains L latent positions indexed by $\mathcal{I} = \{0, \ldots, L-1\}$. The *auxiliary anchor* indices form a sparse subset of this timeline with anchor stride Δ as

$$\mathcal{P} = \{m\Delta \mid m \in \mathbb{N}_0,\ m\Delta < L\} \cup \{L-1\}, \qquad M = |\mathcal{P}|. \tag{1}$$

The planner processes $\mathcal{P}$ in $N_{\mathrm{P}} = \lceil M/B_{\mathrm{P}} \rceil$ ordered blocks $\{\mathcal{Q}_j\}$ of at most B_{P} auxiliary anchor latents. The renderer processes $\mathcal{I}$ in $N_{\mathrm{R}} = \lceil L/B_{\mathrm{R}} \rceil$ ordered chunks $\{\mathcal{C}_i\}$ of at most B_{R} latents, and $\Gamma(i) = \{2u\Delta, (2u{+}1)\Delta, (2u{+}2)\Delta\} \subseteq \mathcal{P}, u = \lfloor B_{\mathrm{R}} i/(2\Delta) \rfloor$ identifies the clean anchor KV entries visible to renderer chunk i. Interior regions use anchor windows with indices before and after each region; at the beginning or end of a finite video, $\Gamma(i)$ uses the available boundary-side anchors. Let θ denote the shared model with role-specific embeddings and LoRA adapter sets, while

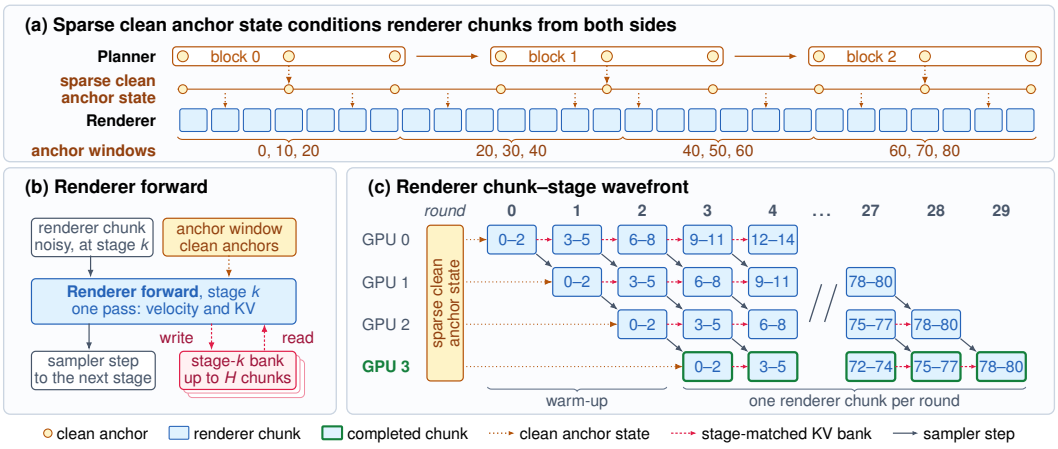

Figure 2: **(a)** The planner produces sparse auxiliary anchor latents; local three-anchor windows guide the renderer. **(b)** One renderer forward reads clean anchor KV and stage-matched renderer history, predicts its latent update, and publishes its own KV. **(c)** These dependencies form a wavefront that completes one chunk per pipeline round after warm-up. See the concrete generation example in Sec. 3. *See the supplementary material for an interactive demo.*

F_θ^{P} and F_θ^{R} denote its planner and renderer routes. At inference, both use S denoising stages indexed by k, $\sigma_S > \cdots > \sigma_0 = 0$, and $\tau_k = \tau(\sigma_k)$ denotes the corresponding timestep. Our empirical configuration uses $B_{\mathrm{P}} = B_{\mathrm{R}} = 3$, $\Delta = 10$, $S = 4$, a context budget of $W_{\mathrm{ctx}} = 21$ latents, a renderer history length of $H = 5$ chunks, and a planner history length of $H_{\mathrm{P}} = 6$ blocks. Sec. A.1 explains how equal planner block and renderer chunk sizes together with $\Delta = 10$ balance anchor production and consumption during streaming generation.

3.1 Stage-Matched Renderer History Enables Pipelined Rendering

A renderer forward takes the current chunk's noisy latents at a given denoising stage as *inputs*. It reads KV from earlier chunks at the same stage together with clean KV from nearby anchors as *memory*. The forward produces the prediction used to advance denoising as *outputs* and stores its own KV for later chunks at that stage as *updated memory*. See Fig. 2(b) for an illustration.

Precisely, $\mathcal{A}_{\Gamma(i)}$ contains clean anchor KV entries selected by $\Gamma(i)$. The stage-matched renderer history bank $\mathcal{B}_k^{(i)}$ contains KV produced when up to H preceding renderer chunks passed through denoising stage k. Under the rectified-flow parameterization (Liu et al., 2023), $x_\sigma = (1-\sigma)x_0 + \sigma\varepsilon$ and the velocity target is $v = \varepsilon - x_0$. At stage k, the renderer takes the current noisy chunk $x_{i,k}^{\mathrm{R}}$, conditions on $\mathcal{A}_{\Gamma(i)}$ and $\mathcal{B}_k^{(i)}$, and predicts $\widehat{v}_{i,k}^{\mathrm{R}}$ to advance the latent from σ_k toward σ_{k-1}. The same forward publishes its KV to the stage-k bank for later chunks, as shown in Fig. 2(b):

$$\left(\widehat{v}_{i,k}^{\mathrm{R}}, K_{i,k}^{\mathrm{R}}, V_{i,k}^{\mathrm{R}}\right) = F_\theta^{\mathrm{R}}\left(x_{i,k}^{\mathrm{R}}, c, \tau_k; \mathcal{A}_{\Gamma(i)}, \mathcal{B}_k^{(i)}\right), \quad \mathcal{B}_k^{(i+1)} = \mathrm{Tail}_H\left(\mathcal{B}_k^{(i)} \cup (i, K_{i,k}^{\mathrm{R}}, V_{i,k}^{\mathrm{R}})\right). \tag{2}$$

Both KV banks are ordered sequences: $\cup$ adds their newest entry, and Tail_H retains the H most recent renderer chunks. No renderer cache-update-only forward. In practice, the renderer can process the clip autoregressively, updating the KV cache on the fly as it handles one chunk at a time, or process the full clip at once using a chunk-wise causal mask.

Why the two states are complementary. The two states are deliberately assigned different temporal scales. Dense stage-matched renderer history is available immediately and records fine changes in recent appearance and motion, but it is noisy; used alone, it can propagate uncertainty across chunk boundaries. The sparse clean anchor provides a stable, coarse structural reference over a longer interval for both the past and the future. The clean-endpoint cache extraction is confined to sparse planner blocks, which advance by a large stride: one planner transition advances 30 latents, compared with 3 for one renderer transition, nearly a tenfold reduction in autoregressive depth. This reduces the

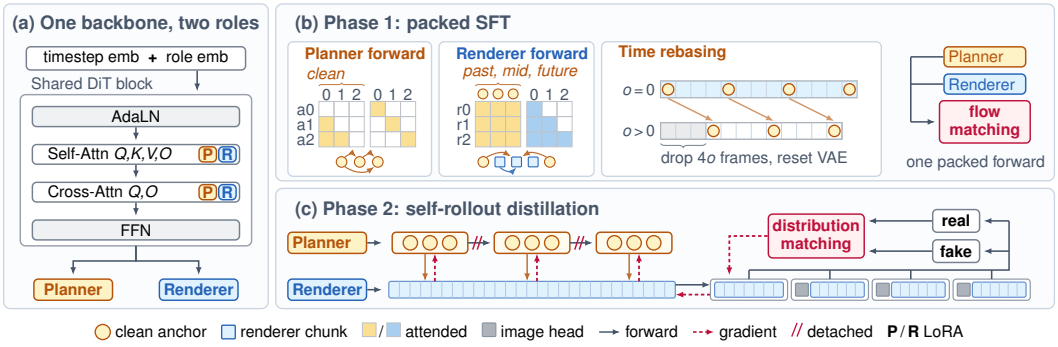

Figure 3: **(a)** Shared backbone with role-specific embeddings and LoRA adapters. **(b)** Packed supervised fine-tuning (SFT) trains both roles in one forward. **(c)** Self-rollout distillation adapts the model to long generated contexts. *See the supplementary material for an interactive demo.*

number of per-chunk cache-update-only forwards and leads to inter-chunk pipelining for the renderer. This coarse plan also improves temporal coherence over long horizons.

Planning the clean anchor KV bank. The planner generates auxiliary anchor latents autoregressively across blocks $\{\mathcal{Q}_j\}$. After each planner block is generated, one planner cache-extraction forward publishes its clean anchor KV entries. Renderer chunk i reads the local window $\mathcal{A}_{\Gamma(i)}$ with earlier and later anchor indices while adjacent regions share anchors. In Fig. 2(a), the windows are $\{0, 10, 20\}$, $\{20, 30, 40\}$, $\{40, 50, 60\}$, and $\{60, 70, 80\}$.

Wavefront rendering. A renderer node (i, k) depends on the noisier chunk at node $(i, k{+}1)$ and the earlier chunk at node $(i{-}1, k)$. With one worker assigned to each of the S stages, nodes on the same anti-diagonal can therefore run concurrently for different chunks, as shown in Fig. 2(c). A *pipeline round* is one such forward slot on every active stage worker. After warm-up, the wavefront completes one renderer chunk per round while every forward both denoises and publishes memory. This pipeline avoids the repeated stage-specific re-encoding in HiAR. Alg. 1 gives the batched four-device rollout; device allocation and timing details are deferred to Sec. A.

Streaming generation. FlashForward naturally supports streaming generation by dedicating one GPU to the planner and the remaining GPUs to the renderer. This allows rendering to begin as soon as the planner produces the first anchor block, further reducing latency. We discuss this in Sec. A.2.3.

3.2 Training a Shared Planner and Renderer

The two roles share visual and language knowledge but require different temporal dependencies. We use role-specific embeddings and LoRA adapters to specialize these dependencies, as in Fig. 3(a). We learn this in two phases: supervised fine-tuning first establishes both roles from real videos, and self-rollout distillation then adapts them to generated context and guidance-free few-step inference.

Phase 1: Supervised fine-tuning. The bidirectional generator is not yet adapted to either the planner's large-stride block-causal dependence or the renderer's clean anchor KV and stage-matched renderer history dependencies. Phase 1 trains both roles together in one packed forward from real videos. The planner blocks attend causally to clean KV from earlier blocks as in teacher forcing (Williams & Zipser, 1989). The renderer targets attend to clean anchor KV admitted by $\Gamma(i)$, and to up to H preceding renderer chunks at the same noise level. A training clip supplies planner and renderer targets directly. As planner latents are sparse, we time-rebase each clip by cropping it from temporal offsets for augmentation, as in Fig. 3(b). Alg. 2 and Sec. C.1 give the details.

Phase 2: Self-rollout distillation. Phase 1 conditions on ground-truth latents, whereas inference uses planner-generated anchor KV and renderer-generated history throughout a few-step trajectory. Phase 2 therefore trains on a planner–renderer self-rollout and applies the distribution-matching distillation objective (Yin et al., 2024b;a), as shown in Fig. 3. As in Self-Forcing (Huang et al.,

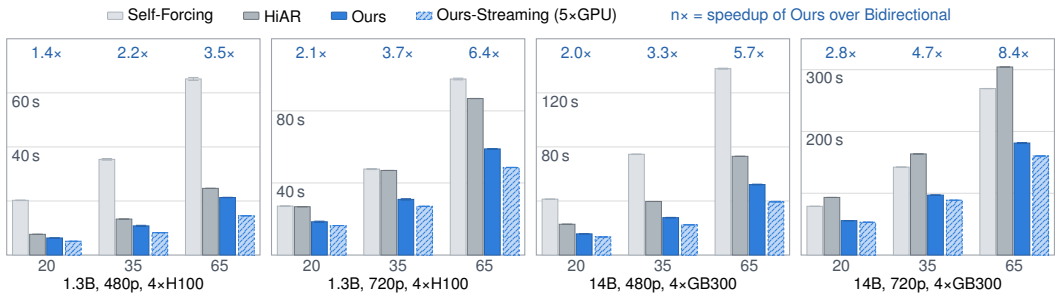

Figure 4: Average denoising latency on four-GPUs and our five-GPU streaming version with standard deviations. Tab. 4 reports exact numbers. *See the supplementary material for an interactive demo.*

Table 2: VBench-1.0 on 20 s videos (top; 5 s for bidirectional models) and VBench-Long (bottom).

VBench-1.0 Metric	LTX-Video	Wan2.1-T2V-1.3B	NOVA	Pyramid Flow	SkyReels-V2-1.3B	MAGI-1-4.5B	CausVid	Self-Forcing	Causal Forcing	HiAR	**FlashForward**
Total ↑	0.766	0.802	0.773	0.775	0.788	0.757	0.764	0.805	0.810	0.821	**0.838**
Quality ↑	0.789	0.813	0.777	0.804	0.808	0.785	0.771	0.829	0.837	0.846	**0.859**
Semantic ↑	0.685	**0.766**	0.757	0.670	0.707	0.647	0.740	0.708	0.701	0.723	0.753

VBench-Long	Self-Forcing			Causal Forcing			HiAR			**FlashForward**		
Duration	Total ↑	Quality ↑	Semantic ↑	Total ↑	Quality ↑	Semantic ↑	Total ↑	Quality ↑	Semantic ↑	Total ↑	Quality ↑	Semantic ↑
20 s	0.8258	0.8460	0.7452	0.8178	0.8515	0.6833	0.8265	0.8440	**0.7565**	**0.8404**	**0.8635**	0.7481
35 s	0.8087	0.8330	0.7116	0.7906	0.8343	0.6159	0.8230	0.8452	0.7339	**0.8401**	**0.8636**	**0.7460**
65 s	0.7806	0.8186	0.6288	0.7539	0.8145	0.5113	0.8219	0.8419	0.7418	**0.8395**	**0.8617**	**0.7505**

2025), conditioning on self-generated context mitigates exposure bias. The student first performs a causal planner rollout over auxiliary anchor blocks and then renders the full clip at once. Both roles follow the same four-stage guidance-free student schedule. Although the score networks use shorter windows, the loss covers the full clip. We split the clip into score-window-sized tiles and decode and re-encode the first frame of each later segment to obtain its image head. During training, the renderer part is implemented as a standard block-causal forward, rather than a chunk-by-chunk cache update. This computation graph lets gradients flow from later chunks to earlier chunks, and from the renderer to the planner. Alg. 3 and Sec. C.2 introduce these in detail.

4 Experiments

We build FlashForward on Wan2.1-T2V-1.3B (Wan et al., 2025) and train it on 256K Shutterstock videos (Shutterstock, 2026) with generated captions. Each clip has 81 latents, corresponding to 321 RGB frames at 16 FPS. Training takes 1,800 steps for the SFT phase and 100 student steps for the distillation phase with a batch size of 128. See Sec. C for details. We evaluate generation quality on the 1.3B model at 480p, following previous works. We use VBench-1.0 (Huang et al., 2024) on 20-second videos following HiAR (Zou et al., 2026), and VBench-Long (Huang et al., 2026) on 20-, 35-, and 65-second videos. We mainly compare with clean-history methods Self-Forcing (Huang et al., 2025) and Causal Forcing (Zhu et al., 2026), and less-noisy-history method HiAR (Zou et al., 2026). We also report results from LTX-Video (HaCohen et al., 2025), Wan2.1 (Wan et al., 2025), NOVA (Deng et al., 2025), Pyramid Flow (Jin et al., 2025), SkyReels-V2 (Chen et al., 2025), MAGI-1 (Sand AI et al., 2025), and CausVid (Yin et al., 2025) for reference.

4.1 Fast and High-Quality Generation

Faster generation. Fig. 4 and Tab. 4 show that, from 20 s onward and using each method's fastest setup with up to four GPUs, FlashForward is the fastest autoregressive schedule in every model–resolution setting, 1.16–1.69× faster than HiAR and 1.42–2.92× faster than Self-Forcing. The gain comes from two properties of its state contract: stage-matched renderer history avoids the cache-update-only forwards required by Self-Forcing and HiAR, while its availability leads to an inter-chunk stage pipeline. Sec. A provides the detailed analysis.

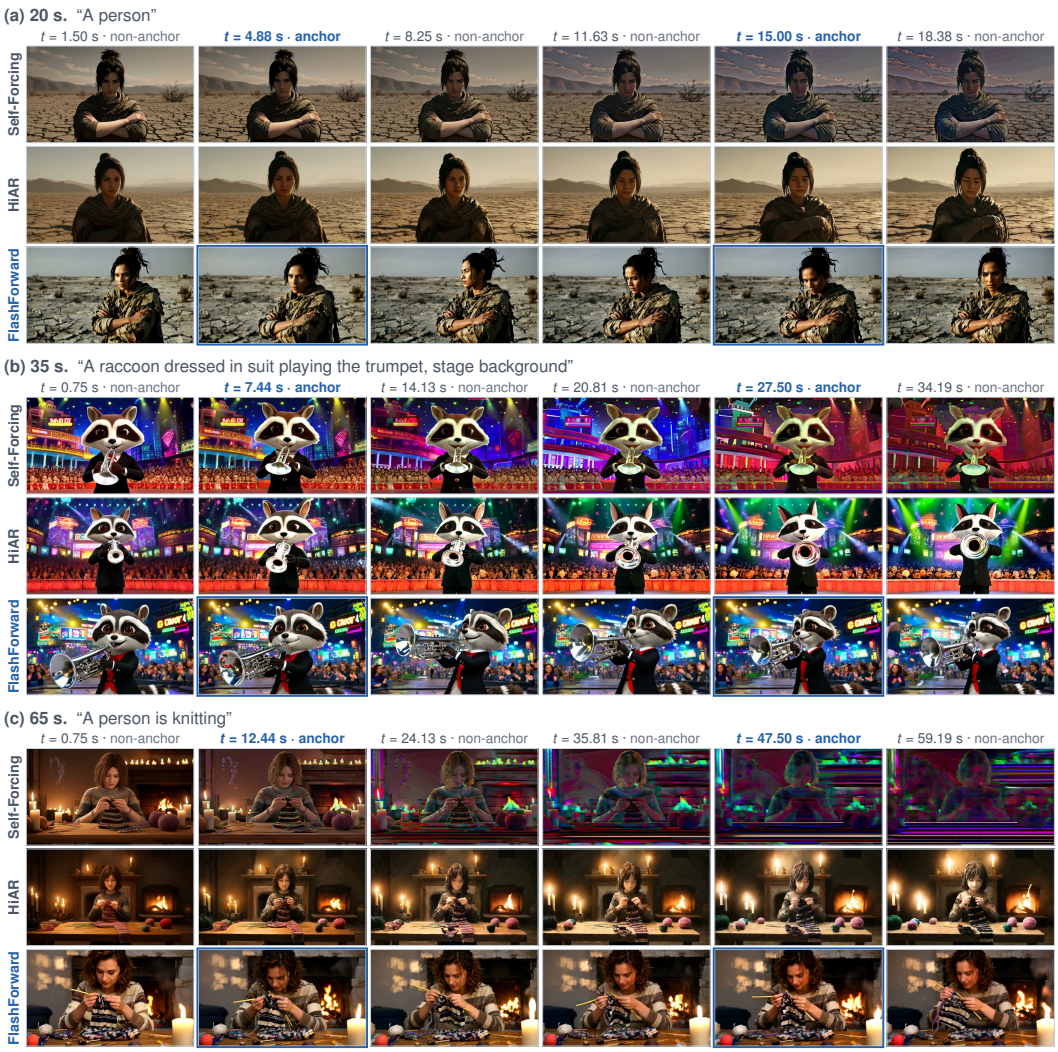

Figure 5: Qualitative comparison at 20, 35, and 65 s. Blue boxes indicate anchor frames. The baselines have no anchors. *See full prompt and video comparisons in the supplementary material.*

Quality. **VBench-1.0** results are in Tab. 2 (top). Competing methods' results are taken from HiAR (Zou et al., 2026). FlashForward achieves the highest Total and Quality scores. Its Semantic score is the highest among the distilled autoregressive methods. **VBench-Long** evaluates long-video quality. We run all four methods under the same protocol (Tab. 2, bottom). FlashForward leads in Total and Quality at every duration, maintaining long-duration quality, while its semantic alignment remains competitive. The behavior of FlashForward under different durations is stable and consistent.

Qualitative comparison. Fig. 5 shows the same trend qualitatively. Self-Forcing accumulates color drift. HiAR avoids this collapse but drifts in other ways: the raccoon's face and ears change in (b) and the knitter's face becomes doll-like in (c), while in (a) the person's face changes and the arms stay blurred under the cloak. FlashForward maintains temporal coherence while preserving motion. Its frames at anchor positions show no visible difference from those between anchors.

4.2 Further Analysis

Why regenerate the anchor positions? Keeping the planner's anchors as output would be cheaper, but we empirically find that this produces periodic seams at the anchor positions (*see the supplementary material for video examples*). We compare and analyze two variants that both use the planner's anchor latents as part of the video output and differ in the source of the anchor KV: one directly

Table 3: Top: training ablations. Bottom: conditioning-state ablations.

	SFT phase (50-step sampling)			Distillation phase (4-step sampling)		
Ablation	Total ↑	Quality ↑	Semantic ↑	Total ↑	Quality ↑	Semantic ↑
Dual-role SFT	0.8172	0.8371	0.7376	0.8184	0.8332	0.7593
+ time-rebased supervision	0.8150	0.8345	0.7371	0.8198	0.8338	**0.7638**
+ role-specific embedding and LoRA	**0.8205**	**0.8377**	**0.7516**	**0.8380**	**0.8593**	0.7528

Conditioning module	Total ↑	Quality ↑	Semantic ↑
Stage-matched history	Failed	Failed	0.732
Clean history and future	0.826	0.849	0.735
Stage-matched history + clean anchor KV (ours)	**0.838**	**0.859**	**0.753**

reuses the planner's anchor KV, while the other feeds the anchor latents to the renderer role to obtain renderer-side KV. Both suffer from a similar seam issue. We analyze this in details in Sec. B.

Ablation study. Tab. 3 (top) reports the training ablations. Time-rebased supervision provides a better initialization for distillation. Role specialization lets the model learn role-specific behavior more freely, and more importantly, reduces seams between anchor and non-anchor positions.

Conditioning modules. We compare the following cross-chunk conditioning designs: (1) Stage-matched history; (2) Clean history and future; (3) Stage-matched history with clean history and future, our final design. Variants are trained under the same setup and reported in Tab. 3 (bottom). With only stage-matched history, the generated videos flicker so heavily that VBench fails to score their temporal flickering dimension. With only clean history and future, the model becomes a variant of Self-Forcing with a different temporal generation order. It mitigates the drift issue of Self-Forcing but still generates at a high latency; hence, we only utilize this as a sparse conditioning module rather than a dense generation model. By combining these modules, FlashForward demonstrates better generation quality with faster generation speed.

Limitations and future work. FlashForward targets long-sequence, few-step generation on multiple GPUs. Planner and pipeline warm-up reduce its gains for short videos, while one-GPU inference cannot exploit the wavefront parallelism. Our training data consists only of real-world videos, which biases the model toward photorealistic content and may limit its performance on animation and other stylized domains. We use a fixed anchor stride, conditioning window, and vanilla training recipe except for the time-rebased augmentation; relaxing the current clean anchor-state assumption and adapting anchor selection remain future work. The cross-chunk state contract may also complement KV selection and compression, RoPE (Su et al., 2024) modifications, and anti-drift techniques. Parallel generation of anchor-delimited intervals (Xiang et al., 2025) is also promising.

5 Conclusion

FlashForward treats cross-chunk memory as a state-availability problem: a cross-chunk state contract specifies what KV is published and when a later chunk may consume it. Publishing the KV produced by each ordinary renderer forward makes stage-matched renderer history available early enough to induce an inter-chunk wavefront over stage workers, so every renderer pass advances the output. Because this promptly available history is noisy, fine-scale, and past-only, sparse auxiliary anchor latents are converted into clean anchor KV ahead of the local regions they guide, providing a complementary coarse structural reference. The renderer uses this clean anchor KV only as conditioning and regenerates every final position. This temporal-scale allocation enables inter-chunk stage pipelining. Planner and renderer roles with a shared backbone realize the dependency graph through packed real-video supervision and full-clip self-rollout distillation. Across 1.3B and 14B backbone scales at 480p and 720p, FlashForward is the fastest evaluated autoregressive schedule on four GPUs for outputs of 321 RGB frames or more. On four H100 GPUs with 1.3B model at 480p, it generates videos faster than other methods with better quality and temporal coherence across several durations.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks: (1) generating synthetic data, in that the text captions of our 256K training videos were generated by the vision–language model Qwen3.5-VL; the videos themselves are real footage and were not generated. (2) editing text drafted by the authors to improve grammar and readability; (3) editing software code, namely optimizing the authors' own training and evaluation code; (4) and assisting in the creation of the interactive visualizations of the supplementary webpage.

We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, implement methods, clean and reformat datasets, or interpret results. Assisting with translation and supporting qualitative and thematic data analysis are not applicable.

We have reviewed all AI-assisted work. All AI-edited text was read and revised by the authors, and all AI-edited code was reviewed and validated by the authors. All reported numbers were produced by code run by the authors; no generative AI tool produced or altered any reported number, table entry, or plotted value. The animations on the supplementary webpage are simplified, schematic illustrations of the method, and all latency values shown there are taken from Tab. 4. The evaluation prompts are the publicly released VBench prompts and were not written or modified by us. We take responsibility for the final content of this work, including text, claims or artifacts.

REPRODUCIBILITY STATEMENT

The components needed to reproduce this work are specified in the paper and appendix. Alg. 2 gives the packed Phase-1 supervised forward; Sec. C.1 states the time-rebasing construction and the optimizer, schedule, and EMA settings. Alg. 3 gives the Phase-2 self-rollout distillation, including the student exit, the score-network tiling, and the gradient routing, and Sec. C.2 states the four-stage student schedule of Eq. (6), the guidance scales, and the optimizer settings for both the student and the critic. Alg. 1 states the multi-device generation procedure in full, Sec. A.1 explains anchor spacing, and Sec. A.2 states the measurement protocol behind every latency and memory number. The base generator, teacher, critic, and VBench evaluation suite are publicly available.

ETHICS STATEMENT

Lower latency helps creative production and simulation but also reduces the cost of deceptive media, impersonation, and misinformation. The method inherits the backbone's biases, copyright risks, and unsafe behavior. Deployments should retain safeguards, require consent and provenance, disclose synthetic content, and monitor abuse.

REFERENCES

Matthew Bendel, Stephen W. Bailey, Mithilesh Vaidya, Sumukh Badam, and Xingzhe He. Goodbye drift: Anchored tree sampling for long-horizon video-to-video generation. *arXiv preprint arXiv:2605.20476*, 2026.

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, Varun Jampani, and Robin Rombach. Stable Video Diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023a.

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 *CVPR*, 2023b.

Shengqu Cai, Weili Nie, Chao Liu, Julius Berner, Lvmin Zhang, Nanye Ma, Hansheng Chen, Maneesh Agrawala, Leonidas Guibas, Gordon Wetzstein, and Arash Vahdat. Mode seeking meets mean seeking for fast long video generation. In *ICML*, 2026.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion Forcing: Next-token prediction meets full-sequence diffusion. In *NeurIPS*, 2024.

Guibin Chen, Dixuan Lin, Jiangping Yang, Chunze Lin, Junchen Zhu, Mingyuan Fan, Hao Zhang, Sheng Chen, Zheng Chen, Chengcheng Ma, Weiming Xiong, Wei Wang, Nuo Pang, Kang Kang, Zhiheng Xu, Yuzhe Jin, Yupeng Liang, Yubing Song, Peng Zhao, Boyuan Xu, Di Qiu, Debang Li, Zhengcong Fei, Yang Li, and Yahui Zhou. SkyReels-V2: Infinite-length film generative model. *arXiv preprint arXiv:2504.13074*, 2025.

Haoge Deng, Ting Pan, Haiwen Diao, Zhengxiong Luo, Yufeng Cui, Huchuan Lu, Shiguang Shan, Yonggang Qi, and Xinlong Wang. Autoregressive video generation without vector quantization. In *ICLR*, 2025.

Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. PipeFusion: Patch-level pipeline parallelism for diffusion transformers inference. In *NeurIPS*, 2025.

Kaifeng Gao, Jiaxin Shi, Hanwang Zhang, Chunping Wang, Jun Xiao, and Long Chen. Ca2-VDM: Efficient autoregressive video diffusion model with causal generation and cache sharing. In *ICML*, 2025.

Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic VQGAN and time-sensitive transformer. In *ECCV*, 2022.

Yuchao Gu, Weijia Mao, and Mike Zheng Shou. Long-context autoregressive video modeling with next-frame prediction. *arXiv preprint arXiv:2503.19325*, 2025.

Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, Poriya Panet, Sapir Weissbuch, Victor Kulikov, Yaki Bitterman, Zeev Melumian, and Ofir Bibi. LTX-Video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2025.

William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Weilbach, and Frank Wood. Flexible diffusion modeling of long videos. In *NeurIPS*, 2022.

Yingqing He, Tianyu Yang, Yong Zhang, Ying Shan, and Qifeng Chen. Latent video diffusion models for high-fidelity long video generation. *arXiv preprint arXiv:2211.13221*, 2022.

Roberto Henschel, Levon Khachatryan, Hayk Poghosyan, Daniil Hayrapetyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. StreamingT2V: Consistent, dynamic, and extendable long video generation from text. In *CVPR*, 2025.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. In *NeurIPS*, 2022.

Cheeun Hong, German Barquero, Fadime Sener, Markos Georgopoulos, Edgar Schönfeld, Stefan Popov, Yuming Du, Oscar Mañas, and Albert Pumarola. SneakPeek: Future-guided instructional streaming video generation. *arXiv preprint arXiv:2512.13019*, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self Forcing: Bridging the train-test gap in autoregressive video diffusion. In *NeurIPS*, 2025.

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 *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. *TPAMI*, 2026.

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 preprint arXiv:2605.09681*, 2026.

Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, Hao Jiang, Nan Zhuang, Quzhe Huang, Yang Song, Yadong Mu, and Zhouchen Lin. Pyramidal flow matching for efficient video generative modeling. In *ICLR*, 2025.

Jihwan Kim, Junoh Kang, Jinyoung Choi, and Bohyung Han. FIFO-Diffusion: Generating infinite videos from text without training. In *NeurIPS*, 2024.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. DistriFusion: Distributed parallel inference for high-resolution diffusion models. In *CVPR*, 2024.

Zekun Li, Xiaoyan Cong, Hongyu Li, Zhiyang Dou, Chuan Guo, Abhay Mittal, Sizhe An, and Srinath Sridhar. Ms. Forcing: Efficient streaming video generation with multi-scale patchification and attention. *arXiv preprint arXiv:2607.20940*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

Jinxiu Liu, Xuanming Liu, Kangfu Mei, Yandong Wen, Ming-Hsuan Yang, and Weiyang Liu. Streaming autoregressive video generation via diagonal distillation. In *ICLR*, 2026a.

Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling Forcing: Autoregressive long video diffusion in real time. In *ICLR*, 2026b.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Jiayi Luo, Qiyan Liu, Tengyang Wang, JunHao Liu, Jiayu Chen, Cong Wang, Hanxin Zhu, Chen Gao, Xiaobin Hu, Qingyun Sun, and Zhibo Chen. Future Forcing: Future-aware training-free KV cache policy for autoregressive video generation. *arXiv preprint arXiv:2605.30083*, 2026.

Yuexiao Ma, Xuzhe Zheng, Jing Xu, Xiwei Xu, Feng Ling, Xiawu Zheng, Huafeng Kuang, Huixia Li, Xing Wang, Xuefeng Xiao, Fei Chao, and Rongrong Ji. Flow caching for autoregressive video generation. In *ICLR*, 2026.

Junyi Ouyang, Wenbin Teng, Gonglin Chen, Yajie Zhao, and Haiwei Chen. DCARL: A divide-and-conquer framework for autoregressive long-trajectory video generation. In *ECCV*, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022.

Dvir Samuel, Issar Tzachor, Matan Levy, Michael Green, Gal Chechik, and Rami Ben-Ari. FAST-AR: Fast autoregressive video diffusion and world models with temporal cache compression and sparse attention. In *ICML*, 2026.

Sand AI, Hansi Teng, Hongyu Jia, Lei Sun, Lingzhi Li, Maolin Li, Mingqiu Tang, Shuai Han, Tianning Zhang, W. Q. Zhang, Weifeng Luo, Xiaoyang Kang, Yuchen Sun, Yue Cao, Yunpeng Huang, Yutong Lin, Yuxin Fang, Zewei Tao, Zheng Zhang, Zhongshu Wang, Zixun Liu, Dai Shi, Guoli Su, Hanwen Sun, Hong Pan, Jie Wang, Jiexin Sheng, Min Cui, Min Hu, Ming Yan, Shucheng Yin, Siran Zhang, Tingting Liu, Xianping Yin, Xiaoyu Yang, Xin Song, Xuan Hu, Yankai Zhang, and Yuqiao Li. MAGI-1: Autoregressive video generation at scale. *arXiv preprint arXiv:2505.13211*, 2025.

Shutterstock. Shutterstock. `https://www.shutterstock.com/`, 2026. Accessed: 2026-01-01.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-A-Video: Text-to-video generation without text-video data. In *ICLR*, 2023.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *ICML*, 2023.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual descriptions. In *ICLR*, 2023.

Vikram Voleti, Alexia Jolicoeur-Martineau, and Chris Pal. MCVD: Masked conditional video diffusion for prediction, generation, and interpolation. In *NeurIPS*, 2022.

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

Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *ICLR*, 2020.

Wenming Weng, Ruoyu Feng, Yanhui Wang, Qi Dai, Chunyu Wang, Dacheng Yin, Zhiyuan Zhao, Kai Qiu, Jianmin Bao, Yuhui Yuan, Chong Luo, Yueyi Zhang, and Zhiwei Xiong. ART-V: Auto-regressive text-to-video generation with diffusion models. In *CVPR Workshops*, 2024.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989.

Xunzhi Xiang, Yabo Chen, Guiyu Zhang, Zhongyu Wang, Zhe Gao, Quanming Xiang, Gonghu Shang, Junqi Liu, Haibin Huang, Yang Gao, Chi Zhang, Qi Fan, and Xuelong Li. Macro-from-micro planning for high-quality and parallelized autoregressive long video generation. *arXiv preprint arXiv:2508.03334*, 2025.

Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Ying-Cong Chen, Yao Lu, Song Han, and Yukang Chen. LongLive: Real-time interactive long video generation. In *ICLR*, 2026a.

Yang Yang, Tianyi Zhang, Wei Huang, Jinwei Chen, Boxi Wu, Xiaofei He, Deng Cai, Bo Li, and Peng-Tao Jiang. Anchor Forcing: Anchor memory and tri-region RoPE for interactive streaming video diffusion. In *ECCV*, 2026b.

Shengming Yin, Chenfei Wu, Huan Yang, Jianfeng Wang, Xiaodong Wang, Minheng Ni, Zhengyuan Yang, Linjie Li, Shuguang Liu, Fan Yang, Jianlong Fu, Ming Gong, Lijuan Wang, Zicheng Liu, Houqiang Li, and Nan Duan. NUWA-XL: Diffusion over diffusion for eXtremely long video generation. In *ACL*, 2023.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Frédo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. In *NeurIPS*, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *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 *CVPR*, 2025.

Yuanhao Zhai, Kevin Lin, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Chung-Ching Lin, David Doermann, Junsong Yuan, and Lijuan Wang. Motion consistency model: Accelerating video diffusion with disentangled motion-appearance distillation. In *NeurIPS*, 2024.

Lin Zhang, Sicheng Mo, Zefan Cai, Jinhong Lin, Zihao Lin, Jiuxiang Gu, Krishna Kumar Singh, Yuheng Li, and Yin Li. UniTemp: Unlocking video generation in any temporal order via bidirectional distillation. In *ECCV*, 2026.

Lvmin Zhang, Shengqu Cai, Muyang Li, Gordon Wetzstein, and Maneesh Agrawala. Frame context packing and drift prevention in next-frame-prediction video diffusion models. In *NeurIPS*, 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 preprint arXiv:2606.25473*, 2026.

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 *ICML*, 2026.

Kai Zou, Dian Zheng, Hongbo Liu, Tiankai Hang, Bin Liu, and Nenghai Yu. HiAR: Efficient autoregressive long video generation via hierarchical denoising. In *ECCV*, 2026.

Appendix overview. The appendix is organized into three sections. Sec. A presents the batched multi-device wavefront algorithm; Sec. A.1 explains wavefront execution, anchor spacing, and planner–renderer rate matching, while Sec. A.2 documents the measurement protocol and planner allocations, analyzes latency and peak memory, analyzes the theoretical acceleration ratio with wall-clock break-even conditions, and evaluates streaming generation with a dedicated planner. Sec. B explains why the renderer regenerates anchor positions, using visual and quantitative evidence to separate the effects of the anchor-KV source and the final anchor source. Sec. C gives the full training and implementation details: Sec. C.1 covers packed time-rebased SFT and its optimization, whereas Sec. C.2 describes self-rollout guidance, score tiling, gradient routing, and the optimization.

We provide an interactive demo in the supplementary material that compares generation quality and latency with other methods, illustrates our generation process and training phases, and shows examples of the chunk-continuity and anchor–non-anchor seam issues that our framework resolves.

A Generation Efficiency Analysis

Throughout this appendix, a *rank* is one GPU worker, and a renderer *round* is one synchronized pipeline interval in which every active rank executes at most one chunk-sized forward at its assigned stage. The stage index $k \in \{S, \dots, 1\}$ denotes the update from noise level σ_k to σ_{k-1}, while $\tau_k = \tau(\sigma_k)$ denotes the corresponding model time input. We use *context parallelism* (CP) for a different form of parallelism: sharding one logical model forward across multiple ranks.

A.1 Wavefront Execution and Anchor Spacing

Algorithm 1 Batched multi-device wavefront generation.

Input: Condition c; length L, positions $\mathcal{I} = \{0, \dots, L-1\}$; local anchor-window map Γ; roles $F_\theta^{\mathrm{P}}, F_\theta^{\mathrm{R}}$.
Constants: S=4 stages with $\sigma_S > \cdots > \sigma_0$=0, Δ=10, B_{P}=B_{R}=3, and a context window of $W_{\mathrm{ctx}} = 21$ latent-frame positions for both roles; H_{P}=$\lfloor (W_{\mathrm{ctx}} - B_{\mathrm{P}})/B_{\mathrm{P}} \rfloor$=6 is the planner-history limit, and H=$\lfloor (W_{\mathrm{ctx}} - B_{\mathrm{P}} - B_{\mathrm{R}})/B_{\mathrm{R}} \rfloor$=5 preceding renderer chunks fit in each stage-matched renderer history.
Layout: $M \leftarrow \lfloor (L-1)/\Delta \rfloor + 1$ anchors $\mathcal{P}$, $N_{\mathrm{P}} \leftarrow \lceil M/B_{\mathrm{P}} \rceil$ blocks $\mathcal{Q}_j$, and $N_{\mathrm{R}} \leftarrow \lceil L/B_{\mathrm{R}} \rceil$ chunks $\mathcal{C}_i$; the last block or chunk contains the remainder when necessary.
Device: Ranks $r \in \{0, \dots, S-1\}$; renderer rank r owns stage $k = S - r$. $\mathcal{B}_k^{(0)} \leftarrow \emptyset, \mathcal{A} \leftarrow \emptyset, \forall r$.
Phase A: Planning. ▷ Complete the planner rollout before rendering, ending with $\mathcal{A}$ in host memory.
for $j = 0, \dots, N_{\mathrm{P}} - 1$ **do**
  Sample $x_{j,S}^{\mathrm{P}} \sim \mathcal{N}(0, 1)$.
  for $k = S, \dots, 1$ **do**
    $\widehat{v}_{j,k}^{\mathrm{P}} \leftarrow F_\theta^{\mathrm{P}}(x_{j,k}^{\mathrm{P}}, c, \tau_k; \mathrm{Tail}_{H_{\mathrm{P}}}(\mathcal{A}))$; step $x_{j,k-1}^{\mathrm{P}} \leftarrow x_{j,k}^{\mathrm{P}} - (\sigma_k - \sigma_{k-1})\widehat{v}_{j,k}^{\mathrm{P}}$.
  end for
  $(K_j^{\mathrm{P}}, V_j^{\mathrm{P}}) \leftarrow F_\theta^{\mathrm{P}}(x_{j,0}^{\mathrm{P}}, c, \tau_0; \mathrm{Tail}_{H_{\mathrm{P}}}(\mathcal{A}))$; $\mathcal{A} \leftarrow \mathrm{Append}(\mathcal{A}, (K_j^{\mathrm{P}}, V_j^{\mathrm{P}})), \forall j$. ▷ including the last
end for
Phase B: Renderer wavefront, $N_{\mathrm{R}} + S - 1$ rounds.
for $n = 0, \dots, N_{\mathrm{R}} + S - 2$ **do**
  for all r **in parallel**, with $i \leftarrow n - r$ and $k \leftarrow S - r$, active iff $0 \le i < N_{\mathrm{R}}$ **do**
    Obtain $x_{i,k}^{\mathrm{R}}$: rank 0 samples noise from $\mathcal{N}(0, 1)$; rank r>0 takes what rank $r-1$ sent in round $n-1$.
    Read anchor KV $\mathcal{A}_{\Gamma(i)}$ and stage-matched history $\mathcal{B}_k^{(i)}$.
    Evaluate Eq. (2): $(\widehat{v}_{i,k}^{\mathrm{R}}, K_{i,k}^{\mathrm{R}}, V_{i,k}^{\mathrm{R}}) \leftarrow F_\theta^{\mathrm{R}}(x_{i,k}^{\mathrm{R}}, c, \tau_k; \mathcal{A}_{\Gamma(i)}, \mathcal{B}_k^{(i)})$.
    Step $x_{i,k-1}^{\mathrm{R}} \leftarrow x_{i,k}^{\mathrm{R}} - (\sigma_k - \sigma_{k-1})\widehat{v}_{i,k}^{\mathrm{R}}$.
    Write in place: $\mathcal{B}_k^{(i+1)} \leftarrow \mathrm{Tail}_H(\mathrm{Append}(\mathcal{B}_k^{(i)}, (i, K_{i,k}^{\mathrm{R}}, V_{i,k}^{\mathrm{R}})))$. ▷ keep the most recent H chunks
    If $r < S-1$, *send* $x_{i,k-1}^{\mathrm{R}}$ to rank $r+1$; otherwise *keep* the finished $x_{i,0}^{\mathrm{R}}$ on $\mathcal{C}_i$.
  end for
end for
Output: The L latents of $\mathcal{I}$, all produced by the renderer, on the last rank.

Wavefront generation. Alg. 1 describes the batched S-device setup used in our main latency matrix. During generation, the devices first run the planner. We could use only one rank to run the planner or use all ranks to run the planner with context parallelism. We discuss the design choice later in details. Either way, the planner will produce the clean anchor KV bank $\mathcal{A}$. Each device then runs one renderer stage and stores its own cache bank. After warm-up, the renderer chunk–stage wavefront in Eq. (2) generates one chunk per round.

Anchor spacing and streaming rate. We keep the planner block size equal to the renderer chunk size, $B_{\mathrm{P}} = B_{\mathrm{R}} = 3$, so that their forwards have comparable cost. The stride $\Delta = 10$ is chosen by matching anchor production and consumption in the streaming allocation with one planner device and four renderer devices. One anchor block requires four denoising forwards and one clean-endpoint KV forward, or five planner rounds. Each local three-anchor window then conditions six to seven renderer chunks, which the steady-state wavefront retires in six to seven rounds. The planner can therefore publish the next anchors before the renderer exhausts its current window. A smaller stride can make planning the bottleneck; a larger stride provides no latency gain once planning stays ahead, and leaves longer regions less constrained temporally.

A.2 Latency and Peak Memory Analysis

Measurement protocol. Throughout the paper, reported latency and FPS refer to *denoising-path* latency. All results in Tab. 4, Tab. 5, and Fig. 4 use identical timing boundaries. Timing spans the first planner or baseline denoising forward through the availability of completed video latents on the reporting rank. Text encoding and VAE decoding are therefore excluded. Planner denoising and clean-endpoint forwards, renderer forwards, cache operations, pipeline synchronization and communication, and the role-weight transfer described below are included whenever the corresponding schedule performs them. Each cell reports the average over 20 generated videos after excluding the cold-start steps. We reset the memory counters and synchronize all ranks before and after each timed window. For each video, we report the slowest-rank elapsed time and the highest peak allocation across all ranks. Among cells with per-video memory measurements, the standard deviation of peak memory has a median of 0.0000 GiB and a maximum of 0.4028 GiB, so we omit it from the memory table. The three autoregressive methods, Self-Forcing, HiAR, and FlashForward, use three-latent chunks and a context budget of $W_{\mathrm{ctx}} = 21$ latent-frame positions, while the bidirectional baseline does not use caching. For FlashForward, these 21 positions comprise three clean anchor-KV positions in $\mathcal{A}_{\Gamma(i)}$, up to 15 renderer-history positions from $\mathcal{B}_k^{(i)}$, and up to $B_{\mathrm{R}} = 3$ positions in the current chunk. The planner uses the same budget: up to 18 completed anchor latents and up to $B_{\mathrm{P}} = 3$ latents in its current block. We evaluate the 1.3B and 14B models on H100 and GB300 GPUs, respectively. The 14B rows are systems-only timing measurements: they instantiate the same tensor shapes, masks, cache paths, and execution schedules with the 14B backbone.

Serving the two roles inside the timed window. Routing latents through adapters would add computation to every forward pass. Therefore, for latency evaluation, we merge each role's adapter into the backbone, producing two full-weight models. A FlashForward forward is then a standard backbone forward and has the same cost as a baseline forward on the same model. This keeps the cost of a block-sized forward comparable across all three schedules. To limit peak device memory, only the active model is kept on the device. At the anchor-to-renderer boundary, the planner weights are off-loaded and the renderer weights are loaded from pinned host memory. This transfer takes 0.10–0.13 s for the 1.3B model and 0.13–0.17 s for the 14B model. The transfer time is included in the four-device FlashForward latencies reported in Tab. 4 and Fig. 4. Streaming keeps both models resident on their assigned devices.

Planner allocations. We support two four-GPU planner allocations. The default allocation (shown as FlashForward in the tables) runs the planner on a single device. FlashForward-CP instead shards each logical planner forward across four devices using context parallelism, which introduces collective-communication overhead. In our experiments, the vanilla version is generally faster for 480p videos, whereas the context-parallel version is faster for 720p videos. We report both variants so that users can choose the best deployment strategy for their target resolution. Sec. A.2.3 evaluates a five-GPU variant with a planner concurrent with the four-device renderer wavefront.

A.2.1 Analysis

Generation latency. Among the four-GPU schedules in Tab. 4, the faster of the two FlashForward planner allocations is the fastest in every model–resolution setting from 20 seconds onward, and it retains this lead at both 35 and 65 seconds. At 65 seconds, FlashForward reduces latency by 3.54–$8.44\times$ relative to bidirectional generation, 1.49–$3.06\times$ relative to Self-Forcing, and 1.16–$1.68\times$ relative to HiAR across these settings. The preferred planner allocation follows the workload: vanilla

Table 4: Per-video denoising-path latency in seconds (lower is better). Each cell is the mean over videos with its standard deviation beneath it. Each setting spans four output lengths: 5, 20, 35, and 65 seconds, which are 21, 81, 141, and 261 latents at 16 FPS. The two FlashForward planner allocations are both measured in every four-GPU cell. All multi-GPU columns use four GPUs unless marked otherwise; FlashForward-Streaming uses one planner GPU and four renderer GPUs. Boldface identifies the fastest schedule at an equal device count and therefore excludes the five-GPU row.

	Wan2.1-T2V-1.3B (H100)											
	480p, 1×GPU				480p, multi-GPU				720p, multi-GPU			
Schedule	5	20	35	65	5	20	35	65	5	20	35	65
Bidirectional	**3.31** ±0.00	32.02 ±0.01	91.22 ±0.04	295.83 ±0.12	**1.13** ±0.01	8.95 ±0.01	24.01 ±0.02	75.28 ±0.05	**3.74** ±0.00	39.59 ±0.06	115.79 ±0.04	379.17 ±0.64
Self-Forcing	3.69 ±0.01	**17.44** ±0.11	**31.48** ±0.14	**57.89** ±0.20	5.10 ±0.05	20.25 ±0.10	35.40 ±0.31	65.14 ±0.55	6.04 ±0.05	27.17 ±0.11	47.71 ±0.27	97.68 ±0.48
HiAR	5.57 ±0.04	27.86 ±0.03	50.41 ±0.20	94.74 ±0.15	2.06 ±0.05	7.67 ±0.00	13.31 ±0.01	24.67 ±0.03	6.49 ±0.01	26.82 ±0.02	46.94 ±0.03	86.85 ±0.06
FlashForward	4.56 ±0.07	17.77 ±0.14	32.80 ±0.21	59.64 ±0.46	2.21 ±0.08	**6.32** ±0.02	**10.79** ±0.22	**21.29** ±0.08	5.71 ±0.05	19.33 ±0.27	35.14 ±0.13	66.53 ±0.12
FlashForward-CP	-	-	-	-	2.34 ±0.02	7.24 ±0.06	12.30 ±0.28	22.07 ±0.20	5.50 ±0.03	**18.47** ±0.35	**30.90** ±0.38	**58.89** ±0.18
FlashForward-Streaming (5×GPU)	-	-	-	-	1.85 ±0.01	5.11 ±0.01	8.27 ±0.01	14.53 ±0.02	5.47 ±0.00	16.37 ±0.02	27.05 ±0.05	48.61 ±0.07

	Wan2.1-T2V-14B (GB300)							
	480p, 4×GPU				720p, 4×GPU			
Schedule	5	20	35	65	5	20	35	65
Bidirectional	**3.50** ±0.01	31.83 ±0.01	90.23 ±0.01	295.45 ±0.01	**12.79** ±0.01	154.47 ±0.01	454.53 ±0.01	1530.42 ±0.03
Self-Forcing	9.75 ±0.09	41.35 ±0.15	74.56 ±0.17	137.89 ±0.37	15.84 ±0.03	78.88 ±0.07	142.29 ±0.11	269.41 ±0.20
HiAR	5.64 ±0.00	22.79 ±0.02	39.63 ±0.02	73.03 ±0.03	21.72 ±0.00	93.25 ±0.03	163.66 ±0.05	304.35 ±0.08
FlashForward	4.79 ±0.04	**15.67** ±0.11	**27.67** ±0.22	53.43 ±0.41	17.19 ±0.09	61.39 ±0.34	109.89 ±0.03	214.83 ±0.54
FlashForward-CP	5.23 ±0.02	16.66 ±0.12	28.10 ±0.33	**52.18** ±0.27	16.28 ±0.07	**55.66** ±0.20	**97.02** ±0.50	**181.25** ±0.80
FlashForward-Streaming (5×GPU)	4.72 ±0.05	13.40 ±0.02	22.35 ±0.05	39.43 ±0.32	17.44 ±0.10	53.17 ±0.14	88.79 ±0.01	160.34 ±0.27

Table 5: Peak allocated memory per rank in GiB.

	Wan2.1-T2V-1.3B (H100)											
	480p, 1×GPU				480p, 4×GPU				720p, 4×GPU			
Schedule	5	20	35	65	5	20	35	65	5	20	35	65
Bidirectional	22.1	31.7	41.2	60.3	15.7	18.4	21.1	26.5	17.0	23.2	29.4	41.8
Self-Forcing	24.9	24.9	25.0	25.1	20.6	20.6	20.7	20.7	28.1	28.2	28.3	28.5
HiAR	24.9	24.9	25.0	25.1	24.9	24.9	25.0	25.1	32.9	33.0	33.1	33.3
FlashForward	24.9	25.0	25.2	25.4	24.9	24.9	25.0	25.1	32.9	33.0	33.1	33.3

	Wan2.1-T2V-14B (GB300)							
	480p, 4×GPU				720p, 4×GPU			
Schedule	5	20	35	65	5	20	35	65
Bidirectional	28.8	34.0	39.1	49.4	31.2	43.1	54.9	78.7
Self-Forcing	60.1	60.2	60.2	60.3	93.2	93.3	93.4	93.6
HiAR	79.0	79.0	79.1	79.1	112.8	112.9	113.0	113.2
FlashForward	79.0	79.0	79.1	79.2	112.8	112.9	113.0	113.3

is generally faster at 480p, whereas context parallelism consistently wins at 720p. Before amortization at 5 seconds, the bidirectional baseline remains fastest in all five settings.

On a single GPU, where the renderer cannot exploit the inter-chunk stage pipeline, FlashForward is nevertheless 1.54–$1.59\times$ faster than HiAR for 20–65-second outputs because it removes most of HiAR's cache-update-only forwards. Compared with Self-Forcing, FlashForward also reduces cache-update-only forwards, but it materializes and stores renderer-history KV at all four denoising stages, rather than once per chunk; these writes occur inside output-advancing forwards but still carry bookkeeping cost. These two effects largely offset each other, leaving FlashForward close to Self-Forcing in latency and only 1.9–4.2% slower.

Peak memory. Tab. 5 reports the memory allocation. Bidirectional and Self-Forcing require less per-rank memory thanks to context parallelism. Because the autoregressive methods use a fixed sliding window, their peak memory is nearly independent of video length. The reported FlashForward allocation remains within 0.3 GiB of HiAR's in every cell.

A.2.2 Discussion: When Is the Pipeline Faster?

Beyond the empirical latency results above, we provide a complementary theoretical comparison of the three autoregressive schedules. Our goal is to characterize the conditions under which FlashForward is expected to be faster. We begin with their forward counts.

Forward-count model. We compare the three autoregressive schedules in *logical block-sized forward equivalents*. Let $N_{\mathrm{P}} = \lceil M/B_{\mathrm{P}} \rceil$ be the number of planner blocks, $N_{\mathrm{R}} = \lceil L/B_{\mathrm{R}} \rceil$ the number of renderer chunks, and $N_{\mathrm{C}} = \lceil L/B_{\mathrm{C}} \rceil$ the number of chunks for either autoregressive baseline. All schedules use $B_{\mathrm{C}} = B_{\mathrm{R}} = 3$. For S denoising stages, their full-video logical work is

$$\begin{aligned} W_{\mathrm{FlashForward}} &= N_{\mathrm{P}}(S+1) + N_{\mathrm{R}}S, \\ W_{\text{Self-Forcing}} &= N_{\mathrm{C}}S + (N_{\mathrm{C}}-1), \\ W_{\mathrm{HiAR}} &= N_{\mathrm{C}}S + (N_{\mathrm{C}}-1)S. \end{aligned} \tag{3}$$

The extra 1 per planner block accounts for clean-KV construction, while the terminal cache-update-only forward is omitted for the baselines because it has no consumer. At 81 latents, $N_{\mathrm{P}} = 3$, $N_{\mathrm{R}} = N_{\mathrm{C}} = 27$, and $S = 4$, giving 123, 134, and 212 forwards, respectively. On four devices, FlashForward and HiAR assign different denoising stages to different ranks, whereas Self-Forcing remains a serial chain whose individual logical forwards use context parallelism; FlashForward's planner also remains serial across anchor blocks. We therefore use two workload-specific CP speedups. Let γ_{plan} be the latency of one complete one-rank planner rollout divided by that of its four-rank CP rollout, and let γ_{SF} be the latency of a one-rank Self-Forcing logical forward divided by that of its four-rank CP counterpart under the same setting.

When does the forward reduction become a wall-clock speedup? Including the $S-1$ wavefront fill/drain slots, the loads and two break-even conditions on S-GPUs are

$$\begin{aligned} R_{\mathrm{FF}} &= N_{\mathrm{R}} + \frac{N_{\mathrm{P}}(S+1)}{\gamma_{\mathrm{plan}}} + (S-1), \\ R_{\mathrm{SF}} &= \frac{N_{\mathrm{C}}S + (N_{\mathrm{C}}-1)}{\gamma_{\mathrm{SF}}}, \\ R_{\mathrm{HiAR}} &= 2N_{\mathrm{C}} - 1 + (S-1), \\ R_{\mathrm{FF}} < R_{\mathrm{HiAR}} &\iff \gamma_{\mathrm{plan}} > \frac{N_{\mathrm{P}}(S+1)}{2N_{\mathrm{C}} - 1 - N_{\mathrm{R}}}, \\ R_{\mathrm{FF}} < R_{\mathrm{SF}} &\iff \gamma_{\mathrm{SF}} < \frac{N_{\mathrm{C}}S + (N_{\mathrm{C}}-1)}{R_{\mathrm{FF}}}. \end{aligned} \tag{4}$$

Against HiAR, the required γ_{plan} falls from 0.83 at 21 latents to 0.58, 0.54, and 0.52 at 81, 141, and 261 latents. As for $\gamma_{\mathrm{plan}} < 1$ we could simply use one rank to run the planner and degenerates $\gamma_{\mathrm{plan}} = 1$, we can always has less loads for model forwards compared with HiAR. Against Self-Forcing, even the conservative choice $\gamma_{\mathrm{plan}} = 1$ allows γ_{SF} to rise from 2.27 to 2.98, 3.12, and 3.21 over the same lengths before the ordering reverses. Thus we can beat Self-Forcing in a large interval.

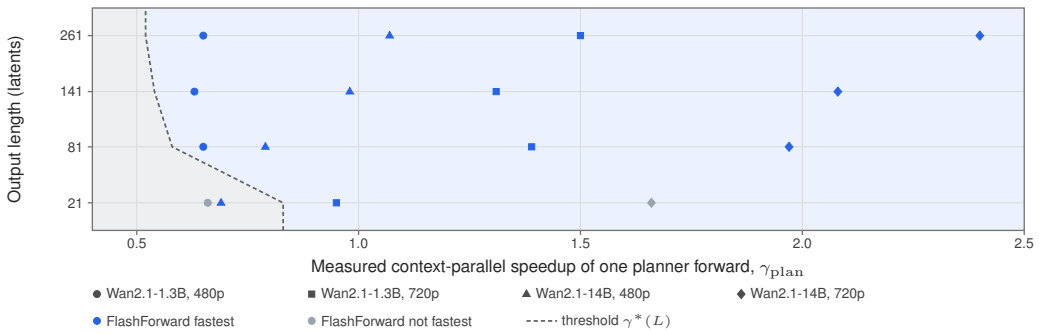

Figure 6: Forward-count break-even boundary between FlashForward and HiAR when the planner uses context parallelism, evaluated in the sixteen four-GPU cells of Tab. 4. Marker shape denotes the system setting; marker color reports whether FlashForward is the fastest measured schedule among FlashForward, HiAR, and Self-Forcing, using the same two colors as the predicted regions containing the markers. The two gray markers are the cells in which FlashForward does not win. A marker whose color disagrees with its region is a cell in which the pairwise boundary alone does not recover the three-way winner. Both outliers occur at a length of 21 latents, suggesting that FlashForward is not well amortized for short sequences in these settings.

Fig. 6 compares this prediction with the measurements in Tab. 4. For each model, resolution, and output length, a point to the right of the dashed boundary predicts that FlashForward is faster than HiAR, while a point to the left predicts the reverse; marker color reports whether FlashForward is the fastest measured schedule of the three, so a marker whose color disagrees with the background region containing it is a cell in which the pairwise boundary alone does not recover the three-way winner. The pairwise FlashForward–HiAR prediction is correct in fifteen of the sixteen cells. When all three schedules are compared, FlashForward is fastest in fourteen cells: the only exceptions are the two 21-latent settings in which HiAR wins at 1.3B 480p and Self-Forcing wins at 14B 720p. From 81 latents onward, FlashForward is fastest in every model–resolution setting.

As videos become longer and each forward becomes more compute-intensive through a larger model or higher resolution, the measured context-parallel speedup tends to improve, while the break-even requirements above become less restrictive and fixed pipeline costs are better amortized. ***The conclusion that FlashForward is faster than both baselines is therefore most robust for long, large-model, and high-resolution generation***. For small forwards with $\gamma_{\text{plan}} < 1$, the same analysis gives a direct deployment rule: use the planner on one rank rather than use context parallelism.

From forward counts to wall-clock speedup. The forward-count ratio is an upper-bound estimate rather than an exact latency prediction: in practice, context parallelism provides sublinear speedup because of collective overhead, and the wavefront pays fixed fill, drain, and synchronization costs. The inter-stage latent tensor is small and its transfer overlaps the next forward, so communication volume is not the dominant term. At 81 latents, the raw counts suggest a $212/123 = 1.72\times$ advantage over HiAR, while the measured speedup is 1.21–$1.68\times$ across the four settings. The gap is largest for short, low-compute workloads, but these costs are progressively amortized as the workload grows; at 14B and 720p, the measured $1.68\times$ speedup approaches the theoretical $1.72\times$, showing that the forward-count advantage translates into practical gains.

A.2.3 Streaming Generation with a Dedicated Planner

The four-device configuration in our main latency matrix reuses the same devices for both roles: all four first complete the sparse-anchor rollout and are then assigned to the four stages of the renderer wavefront to make a fair comparison with other autoregressive methods. This serial planner prologue increases the time to the first completed latent chunk (our TTFF boundary), but it is an implementation choice rather than a dependency imposed by FlashForward: a renderer chunk needs only its local anchor window, not anchors from later windows. For deployment, we therefore dedicate one GPU to the planner and keep four GPUs on the four-stage renderer wavefront. As soon as the

planner publishes the first anchor block, rendering can begin; the planner then produces later blocks concurrently with rendering. This reduces the anchor contribution to TTFF from a complete planner rollout to one block and, more importantly, supports streaming generation.

Measured five-GPU latency. We measure this one-planner–four-renderer schedule on both models, as reported in the FlashForward-Streaming ($5\times$GPU) rows of Tab. 4. Here we compare against the four-GPU FlashForward row, which uses the same planner algorithm without CP; the comparison adds one device and measures the deployment benefit of overlapping that planner with rendering. At 21 latents (5 seconds), there is only one anchor block and hence no later planner work to overlap. The differences at this length therefore reflect model residency and, for the five-GPU 14B setup, cross-node communication rather than pipeline overlap. As sequence length increases, the overlap benefit dominates. From 20 to 65 seconds, the dedicated H100 planner makes 1.3B generation 1.18–$1.47\times$ faster at both 480p and 720p. The 14B configuration spans two GB300 nodes, so each anchor block crosses the inter-node link and communication time grows with sequence length, but most anchor generation remains hidden behind rendering: relative to the four-GPU single-planner row, 14B generation is 1.15–$1.36\times$ faster over the same lengths. In both deployments, the overlap leaves renderer peak memory essentially unchanged from the standard four-GPU version. Together, these results show that a dedicated producer converts the planner prologue into a one-block streaming warm-up and yields increasing denoising-path latency gains as the pipeline is amortized.

Only the first anchor block is exposed. To test whether the planner lies on the steady-state critical path, we separately record the producer's full anchor time and `anchor_wait`, the time for which renderer ranks wait for unavailable anchors. At 480p, the full anchor phase grows from 0.510 to 5.617 s as the number of planner blocks grows from one to nine, while `anchor_wait` remains between 0.506 and 0.509 s. At 720p, the full phase grows from 1.067 to 17.956 s, while the wait remains between 1.051 and 1.066 s. Thus the exposed fraction of planning falls from 99–100% for a single block to 9% at 480p and 6% at 720p for nine blocks. The wait is always approximately the cost of the first block: after startup, the planner stays ahead and all later anchor generation is hidden beneath the renderer wavefront, as predicted by the rate-matching analysis in Sec. A.1.

B Why the Renderer Regenerates the Anchor Positions

Cheaper variants that do not work. A cheaper design would keep the planner anchors in the final video and render only the positions between them. We test two versions of this splice design. The first reuses the KV produced by the planner, while the second passes the clean auxiliary anchor latents through the renderer to build renderer-side KV. Both versions mix two output sources: planner latents at the anchor positions and renderer latents everywhere else. Our method instead uses the planner anchors only as conditioning and lets the renderer generate every output position. This comparison separates the effect of the anchor KV source from the effect of the final output source.

Experiments. To evaluate these choices, we run the experiments summarized in Tab. 6. All experiments use the same training setup as our final model. Panel (a) compares the splice variants with all-position rendering. Panel (b) measures how the mean luminance levels of planner anchor frames and neighboring renderer frames change across training checkpoints for these splice variants. Panel (c) replaces generated planner anchors with latents encoded from real videos to test whether planner generation error alone explains the seam.

What the seam looks like. Fig. 7 visualizes this comparison on four 20-second videos, using the same frame indices for all three variants. In (a) and (b), the anchor frames of both splice variants are darker than their neighbors; under renderer-side KV, the rocks in (c) turn lighter and warmer; and in (d), the bowl turns a brighter red. The blurred differences make the offset visible. For both splice variants, the differences highlight the static background as well as the moving subject, indicating a color change across the whole frame rather than motion. For FlashForward, they stay dark except where large regions move, such as the candle smoke in (a), the bear's body in (b), and the steam above the bowl in (d). The KV source changes how strongly the offset appears in each clip, but neither splice variant removes it, consistent with Panel (a). We *include videos of the two variants in the supplementary material*; they show more directly what the seam looks like in motion.

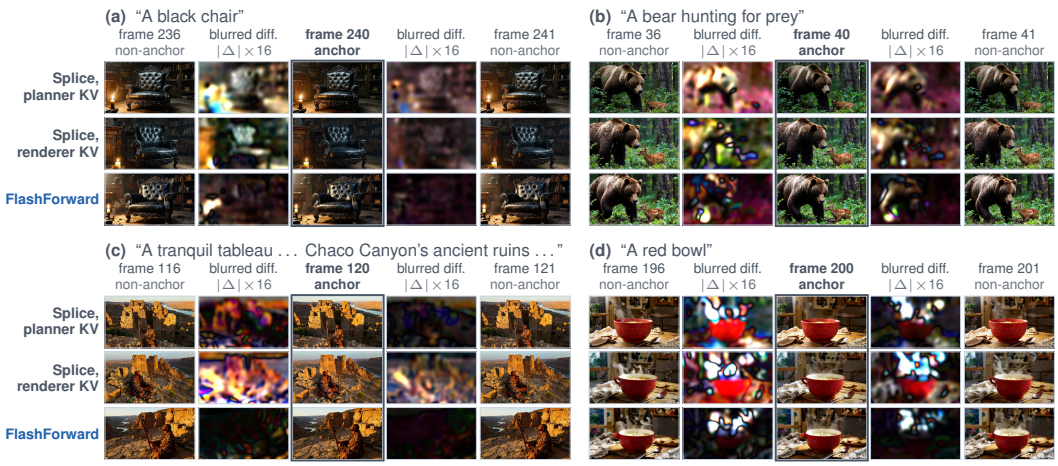

Figure 7: **Color seams at the anchor positions of 20-second videos.** The two splice rows keep the planner anchor latents in the final video and build the anchor KV with the planner or with the renderer, as in Panel (a) of Tab. 6; FlashForward uses the planner anchors only as conditioning and lets the renderer generate every output position. In each block, the boxed column is the last frame decoded from an anchor latent, and the outer columns are the nearest non-anchor frames before and after it. Between each non-anchor frame and the anchor frame, we show their per-channel absolute RGB difference, computed after a Gaussian blur and amplified $16\times$. The blur suppresses edges and fine texture, leaving mostly low-frequency color changes; large moving regions can remain visible.

Table 6: **All-position renderer output leads to smaller seam measurements.** *Panel (a)* compares anchor-KV and final-anchor sources. All seam columns are normalized adjacent-frame luminance-jump statistics, and lower is better: Anchor seam is measured at anchor/non-anchor transitions, Chunk seam at renderer-chunk boundaries away from anchors, and Interior at transitions of neither type. "Periodic" sums the excess over Interior across the transitions in one anchor period. *Panel (b)* reports mean luminance levels for planner-generated anchor frames and their neighboring renderer-generated frames. *Panel (c)* compares real and generated anchors over paired clips; chromatic share is the fraction of the RGB offset orthogonal to the grayscale direction $(1, 1, 1)$, and spatial excess is $\sqrt{s^2_{\text{anchor}} - s^2_{\text{control}}}$, where s is the spatial standard deviation of the per-cell RGB change. The paired t-statistic compares each clip's anchor-boundary offset with its mid-chunk control.

(a) Compared modes: KV source and final anchor source

Denoising steps	Final anchor	Anchor KV	Anchor seam	Chunk seam	Interior	Periodic
50	Planner	Planner	3.93	2.71	1.32	7.66
50	Planner	Renderer	3.34	2.37	1.29	5.99
50	Renderer (ours)	Planner	**1.83**	2.27	1.32	**3.54**
4	Planner	Renderer	6.20	1.68	1.40	9.55
4	Renderer (ours)	Planner	**2.43**	2.50	1.26	**5.61**

(b) Seam formation when planner anchors are used as final outputs

Checkpoint	Anchor level	Renderer level	Anchor − renderer
50-step supervised	87.804	87.898	−0.094
4-step distilled, backbone frozen	98.192	99.364	−1.172
4-step distilled, backbone updated	130.043	136.287	**−6.244**
Paired change: updated − frozen	+31.85	+36.92	−5.07

(c) Compared modes: real anchors and generated planner anchors in spliced output

Anchor source	Anchor offset	Mid-chunk control	Chromatic share	Spatial excess	Paired t-statistic
Real-video anchors	9.47	0.52	49%	6.69	4.34
Generated planner anchors	3.00	0.37	47%	4.62	11.51

At 50 steps, the final output source has the larger measured effect. Panel (a) gives the direct comparison. At 50 steps, replacing planner KV with renderer-side KV only modestly reduces the anchor seam and periodic total, and the seam remains visible. These one-factor comparisons show that changing the final anchor source has the larger measured effect at this phase. In the deployed four-step comparison, all-position rendering again cuts the anchor seam by more than half relative to the splice configuration and clearly lowers the periodic total.

Planner–renderer output levels diverge after distillation. A splice baseline works only if planner anchors and renderer frames have matching output statistics. Panel (b) shows that they nearly match after supervised fine-tuning, with a negligible mean luminance gap. Four-step distillation widens this gap by more than an order of magnitude even with a frozen backbone, and updating the shared backbone widens it several times further. Relative to the frozen model, updating the backbone raises both levels, but the renderer level rises more than the anchor level, so the anchor path lags behind the renderer's shift. This unequal shift becomes visible when planner anchors are inserted into output.

The mismatch is not simply poor planner output. Panel (c) shows that the seam remains when generated planner anchors are replaced with anchors encoded from real videos. The real-anchor probe removes planner generation error, yet all clips still have more error at the anchor boundary than at the mid-chunk control. The mismatch also has more than one component: 47–49% of the frame-level offset is chromatic, and the spatially varying excess remains large for both real and generated anchors. These results are consistent with an output-interface mismatch rather than an artifact explained solely by a bad planner sample or a single brightness offset.

Why all-position rendering avoids cross-role splicing. Our method does not require planner and renderer outputs to match at a splice boundary. The planner anchors provide clean structural guidance, but they are not copied into the final video. The renderer generates every output latent, including the anchor positions, so the final sequence never switches between planner and renderer outputs. This removes the cross-role output switch as one possible source of a periodic seam while preserving the anchors as conditioning, and thus leads to a substantial reduction of anchor seams.

C Training and Implementation Details

Given a clean latent x_0, Gaussian noise ε, and a noise level σ, Rectified Flow (Liu et al., 2023) uses

$$x_\sigma = (1-\sigma)x_0 + \sigma\varepsilon, \qquad v = \varepsilon - x_0. \tag{5}$$

We use a flow shift of 5.0 at every denoising stage. We reserve $k \in \{S, \dots, 1\}$ for a discrete denoising-stage index. For any noise level σ, let $\tau(\sigma)$ denote the scalar model time input produced by the Wan scheduler with flow shift 5.0; at an inference stage, $\tau_k = \tau(\sigma_k)$. During distillation and inference, the student generator follows the four-stage guidance-free trajectory. Only the frozen real-data score uses classifier-free guidance (Ho & Salimans, 2021), with a scale of 3.0, to construct the distillation target. *We provide an interactive demo in the supplementary material to illustrate inputs and training phases.*

C.1 Phase 1: Time-Rebased SFT and Optimization

Packed input sequence and attention mask. Alg. 2 packs both roles into one forward. For a clip of $L = 81$ latents, the planner uses the $M = 9$ anchors $\mathcal{P} = \{0, 10, \dots, 80\}$ in three blocks of three latents, while the renderer treats all 81 positions as targets in 27 chunks of three and reads up to 5 preceding chunks. Each anchor appears as a clean planner-context copy in $\mathcal{P}^0$, a noised planner input in $\mathcal{P}^\sigma$, and a noised renderer input in $\mathcal{I}^\sigma$; every non-anchor appears only as a renderer input in $\mathcal{I}^\sigma$. All noised copies use the clip's single sampled σ, and copies of the same latent share ε. The attention mask isolates the planner, renderer, and time-rebased sequences: planner inputs attend within their current block and to clean earlier planner blocks, whereas each renderer chunk attends within the chunk, to up to five preceding renderer chunks at the same noise level, and to the clean anchors selected by $\Gamma(i)$. The renderer's read of these clean anchors is the only cross-role connection.

Training on 20-second real videos. Real-video supervision is essential for learning the planner's autoregressive dependence across anchor blocks. Under a 5-second training setup, each clip contains

Algorithm 2 Packed SFT forward.

Input: Captioned real videos; frozen VAE $\mathcal{E}$ and text encoder; trainable parameterization θ comprising the shared backbone, rank-256 role adapters $\psi_{\mathrm{P}}, \psi_{\mathrm{R}}$, and two zero-initialized role-embedding vectors.
Constants: $L{=}81$, $\mathcal{P}{=}\{0, 10, \ldots, 80\}$, $H{=}5$, $\Gamma(i)$, flow shift 5.0.
Mask: Precompute the attention mask $\mathbf{M}$ with the following visible keys:
$\mathrm{Vis}(\mathcal{P}^{\rho}[\mathcal{Q}_j]) = \mathcal{P}^{\rho}[\mathcal{Q}_j] \cup \mathcal{P}^{0}[\bigcup_{q<j} \mathcal{Q}_q]$, $\rho \in \{0, \sigma\}$; ▷ same for rebased anchors
$\mathrm{Vis}(\mathcal{I}^{\sigma}[\mathcal{C}_i]) = \mathcal{I}^{\sigma}[\bigcup_{q=\max(0,i-H)}^{i} \mathcal{C}_q] \cup \mathcal{P}^{0}[\Gamma(i)]$.

1: **for** each training step **do**
2: *Encode* $x_0 \leftarrow \mathcal{E}(\text{clip})$ on $\mathcal{I}$; $c \leftarrow$ caption, with 10% text dropout.
3: *Rebase:* Sample distinct $o_1, \ldots, o_4 \in \{1, \ldots, 9\}$; for each n, reset $\mathcal{E}$, $\tilde{x}_0^{(n)} \leftarrow \mathcal{E}(\text{clip}[4o_n, 4o_n{+}281))$, keep its 8 anchors at indices $\{0, 10, \ldots, 70\}$.
4: *Pack* z from planner copies $\mathcal{P}^0, \mathcal{P}^\sigma$, the renderer copy $\mathcal{I}^\sigma$, and rebased planner copies $\{(\widetilde{\mathcal{P}}_n^0, \widetilde{\mathcal{P}}_n^\sigma)\}_{n=1}^4$. ▷ a clean copy only where a later block reads it
5: *Noise:* Sample one σ per clip and set its model time input $\tau_\sigma \leftarrow \tau(\sigma)$; $x_\sigma \leftarrow (1{-}\sigma)x_0 + \sigma\varepsilon$; clean copies stay at $\sigma_0{=}0$. ▷ copies of one latent share ε
6: *Route* planner copies via ψ_{P} and renderer copies via ψ_{R}; add the corresponding role embedding to AdaLN. ▷ one θ gives F_θ^{P} and F_θ^{R}
7: *Forward* $(\widehat{v}^{\mathrm{P}}, \widehat{v}^{\mathrm{R}}, \{\widehat{\widetilde{v}}_n^{\mathrm{P}}\}) \leftarrow F_\theta(z, c, \tau_\sigma; \mathbf{M})$ on noised tokens; clean copies provide KV only.
8: *Score* $v \leftarrow \varepsilon - x_0$ (Eq. (5)); $\ell \leftarrow \|\widehat{v} - v\|^2$; average $\bar{\ell}_{\mathrm{P}}$, $\bar{\ell}_{\mathrm{R}}$, $\bar{\ell}_{\mathrm{P}}^{\mathrm{reb}}$ over the corresponding inputs; $\mathcal{L} \leftarrow \bar{\ell}_{\mathrm{P}} + \bar{\ell}_{\mathrm{R}} + \bar{\ell}_{\mathrm{P}}^{\mathrm{reb}}$.
9: *Update:* Backpropagate once; take an AdamW step; update the EMA at decay 0.999.
10: **end for**
Output: The trained model and its EMA copy.

only three anchor latents, thus provides no supervision for autoregressive generation. Distillation-based methods learn from trajectories produced by a teacher model; unless their training pipelines are substantially redesigned, their supervision horizon is typically limited to the teacher's 5-second generation window (Yin et al., 2025; Huang et al., 2025). Because our SFT phase learns directly from real videos rather than teacher trajectories, it can use clips of arbitrary duration. Balancing stronger long-horizon supervision against training efficiency, we use 20-second clips, which contain nine anchors and form exactly three anchor blocks, ensuring that the planner's autoregressive generation is explicitly trained on real data.

Time-rebased planner supervision. At each step, we sample four distinct latent-time offsets $o_1, \ldots, o_4$ from $\{1, \ldots, 9\}$. Since the causal VAE downsamples time by four, offset o_n defines the 281-frame crop $[4o_n, 4o_n + 281)$. The resulting 71 latents provide eight planner targets at local positions $\{0, 10, \ldots, 70\}$. These targets form blocks of sizes $3 + 3 + 2$. Only the first two blocks have clean context copies because the final block has no later consumer, so each rebased crop adds eight noised inputs and six clean copies and remains isolated from the other packed sequences.

Objective and optimization. The shared backbone is initialized from Wan2.1-T2V-1.3B (Wan et al., 2025); its planner and renderer rank-256 LoRA (Hu et al., 2022) adapters act on the self-attention query, key, value, and output projections and on the cross-attention query and output projections. One zero-initialized vector per role is selected from a role-embedding table and added through adaptive layer normalization (AdaLN). The VAE encoder remains frozen. We apply per-token squared error to the rectified-flow velocity $v = \varepsilon - x_0$ and sum the averages over the 9 original planner latents, 81 renderer latents, and 32 time-rebased planner latents, respectively. Supervised training uses AdamW (Loshchilov & Hutter, 2019) for 1,800 steps of global batch size 128, $\beta_1 = 0$, $\beta_2 = 0.999$, weight decay 0.01, and gradient clipping at 1.0. The learning rate warms up to 10^{-5} over the first 100 steps and then remains constant, and the exponential-moving-average (EMA) decay is 0.999.

C.2 Phase 2: Self-Rollout Guidance and Step Distillation

Self-rollout and gradient routing. Phase 2 starts from the Phase-1 student and retains the same planner–renderer layout. Its four guidance-free denoising stages select the following base scheduler indices, ordered from highest to lowest noise:

$$\mathcal{T}_{\text{student}} = (q_4, q_3, q_2, q_1) = (999, 937, 833, 624). \tag{6}$$

Algorithm 3 Self-rollout distribution-matching distillation.

Input: Trainable SFT student; frozen real-data score F^{real} (Wan2.1-T2V-14B); critic F_ϕ^{fake} (Wan2.1-T2V-1.3B, with a trainable rank-256 LoRA); frozen VAE $\mathcal{E}$, $\mathcal{D}$.
Constants: L=81, M=9 anchors, N_{P}=3, B_{P}=3, N_{R}=27, B_{R}=3, S=4, H=5, W_{score}=21.

function ROLLOUT(c)
  $\mathcal{A} \leftarrow \emptyset$.
  Exit: Sample $k^\star \sim \mathcal{U}\{1,\dots,S\}$; stages above $k^\star$ run detached. ▷ the rollout stops at $k^\star$
  for $\mathcal{Q}_j = \{0,10,20\}, \{30,40,50\}, \{60,70,80\}$ **do**
    $x^{\text{P}}_{j,S} \leftarrow$ noise on $\mathcal{Q}_j$.
    for $k = S,\dots,k^\star$ **do**
      $\widehat{v}^{\text{P}}_{j,k} \leftarrow F^{\text{P}}_\theta(x^{\text{P}}_{j,k}, c, \tau_k; \text{sg}[\mathcal{A}])$.
      If $k > k^\star$, step to $x^{\text{P}}_{j,k-1}$.
    end for
    $\hat{x}^{\text{P}}_j \leftarrow x^{\text{P}}_{j,k^\star} - \sigma_{k^\star}\widehat{v}^{\text{P}}_{j,k^\star}$; run a cache-extraction forward at σ_0=0; $\mathcal{A} \leftarrow \text{Append}(\mathcal{A}, (K^{\text{P}}_j, V^{\text{P}}_j))$.
  end for
  Pack the L samples from $\mathcal{N}(0,1)$, under the block-causal mask of Alg. 2, with $\hat{x}^{\text{P}}$ entering through $\mathcal{A}$.
  for $k = S,\dots,k^\star$ **do**
    In one packed forward at τ_k, evaluate $\widehat{v}^{\text{R}}_{i,k} \leftarrow F^{\text{R}}_\theta(x^{\text{R}}_{i,k}, c, \tau_k; \mathcal{A}_{\Gamma(i)}, \mathcal{B}^{(i)}_k)$ for every i; the mask realizes each same-stage $\mathcal{B}^{(i)}_k$ from the preceding H chunks in this forward.
    If $k > k^\star$, set $x^{\text{R}}_{i,k-1} \leftarrow x^{\text{R}}_{i,k} - (\sigma_k - \sigma_{k-1})\widehat{v}^{\text{R}}_{i,k}$ for every i.
  end for
  $x_{0,i} \leftarrow x^{\text{R}}_{i,k^\star} - \sigma_{k^\star}\widehat{v}^{\text{R}}_{i,k^\star}$ for every i; **return** $(x_0, k^\star)$.
end function
function TILEDSCORE(F, x_0, $x_{\widetilde{\sigma}}$, ε^{h}, c, $\widetilde{\sigma}$, $\widetilde{\tau}$, g)
  Tile $\mathcal{I}$ into $(L-1)/(W_{\text{score}}-1)$=4 tiles T_0=[0, 21), T_1=[21, 41), T_2=[41, 61), T_3=[61, 81).
  $img_{T_0}, img_{T_1}, img_{T_2} \leftarrow \mathcal{D}(x_0 \text{ before } T_3))$;$h_{\ell,0} \leftarrow \mathcal{E}(img_{T_{\ell-1}}), \forall \ell \in \{1,2,3\}$.
  $\widehat{v}[T_0] \leftarrow F(x_{\widetilde{\sigma}}[T_0], c, \widetilde{\tau})$ at guidance g.
  for $\ell = 1,2,3$ **do**
    $h_{\ell,\widetilde{\sigma}} \leftarrow \text{sg}[(1-\widetilde{\sigma})h_{\ell,0} + \widetilde{\sigma}\varepsilon^{\text{h}}]$.
    $u_\ell \leftarrow h_{\ell,\widetilde{\sigma}} \| x_{\widetilde{\sigma}}[T_\ell]$; evaluate $F(u_\ell, c, \widetilde{\tau})$ at guidance g and store the prediction in $\widehat{v}[T_\ell]$.
  end for
  return $\widehat{v}$ on $\mathcal{I}$; discard the image-head predictions.
end function
for each training step with caption c **do**
  if generator phase **then**
    $(x_0, k^\star) \leftarrow$ ROLLOUT(c).
    Sample $\widetilde{\sigma}$, set $\widetilde{\tau} \leftarrow \tau(\widetilde{\sigma})$, and sample $\varepsilon, \varepsilon^{\text{h}} \sim \mathcal{N}(0, I)$; $x_{\widetilde{\sigma}} \leftarrow (1-\widetilde{\sigma})x_0 + \widetilde{\sigma}\varepsilon$.
    $\widehat{v}^{\text{real}} \leftarrow$ TILEDSCORE($F^{\text{real}}, x_0, x_{\widetilde{\sigma}}, \varepsilon^{\text{h}}, c, \widetilde{\sigma}, \widetilde{\tau}, 3.0$).
    $\widehat{v}^{\text{fake}} \leftarrow$ TILEDSCORE($F^{\text{fake}}_\phi, x_0, x_{\widetilde{\sigma}}, \varepsilon^{\text{h}}, c, \widetilde{\sigma}, \widetilde{\tau}, 1.0$).
    $\widehat{x}^{\text{real}}_0 \leftarrow x_{\widetilde{\sigma}} - \widetilde{\sigma}\widehat{v}^{\text{real}}$; $\widehat{x}^{\text{fake}}_0 \leftarrow x_{\widetilde{\sigma}} - \widetilde{\sigma}\widehat{v}^{\text{fake}}$.
    $y \leftarrow \text{sg}[\, x_0 - \big(\widehat{x}^{\text{fake}}_0 - \widehat{x}^{\text{real}}_0\big) / \mathbb{E}\big|x_0 - \widehat{x}^{\text{real}}_0\big|\,]$ ▷ distribution-matching target
    $\mathcal{L}_{\text{gen}} \leftarrow \frac{1}{L}\sum_{p\in\mathcal{I}} \frac{1}{2}\|x_{0,p} - y_p\|^2$; backpropagate; take a student AdamW step; update the student EMA.
  else
    $(x_0, k^\star) \leftarrow$ ROLLOUT(c); sample $\widetilde{\sigma}$, set $\widetilde{\tau} \leftarrow \tau(\widetilde{\sigma})$, and sample $\varepsilon, \varepsilon^{\text{h}} \sim \mathcal{N}(0, I)$; form $x_{\widetilde{\sigma}}$ as above.
    $\widehat{v}^{\text{fake}} \leftarrow$ TILEDSCORE($F^{\text{fake}}_\phi, x_0, x_{\widetilde{\sigma}}, \varepsilon^{\text{h}}, c, \widetilde{\sigma}, \widetilde{\tau}, 1.0$).
    $\mathcal{L}_\phi \leftarrow \frac{1}{L}\sum_{p\in\mathcal{I}} \|\widehat{v}^{\text{fake}}_p - (\varepsilon_p - x_{0,p})\|^2$.
    Backpropagate $\mathcal{L}_\phi$ through F^{fake}_ϕ only; take a critic AdamW step.
  end if
end for
Output: The distilled four-step guidance-free student and its EMA model.

The terminal endpoint is $\sigma_0 = 0$. At each iteration, all ranks sample the same student exit $k^\star$ uniformly from the four stages and retain gradients only at that stage, following the stochastic gradient truncation of Self-Forcing (Huang et al., 2025); the higher-noise stages run under stop-gradient. The rollout first generates the three planner blocks and then all $L = 81$ renderer positions, stopping at $k^\star$ and using the corresponding $\widehat{x}_0$ as its output. A clean-endpoint cache-extraction forward after each planner block constructs $\mathcal{A}$. We detach this bank when it is read by later planner blocks but keep the renderer's read of $\mathcal{A}_{\Gamma(i)}$ connected at the student exit. Since the renderer produces

every output position, this renderer-to-anchor connection is the only path by which the generator loss trains the planner. Alg. 3 gives the complete rollout and loss computation.

At a fixed renderer stage, training evaluates all chunks in one packed forward. The block-causal renderer mask exposes to chunk i only its own latents, the anchor window $\mathcal{A}_{\Gamma(i)}$, and at most the H preceding chunks from that same stage. Inference materializes the equivalent same-stage bank sequentially through Eq. (2), whereas the packed training forward retains the same dependencies and permits gradients through their KV edges.

Windowed teacher and critic scores. Each score network accepts at most $W_{\text{score}} = 21$ latent-frame positions. Alg. 3 therefore divides the query into four target tiles while preserving the full-length student rollout and gradient path. Following LongLive (Yang et al., 2026a), we prepend to each later tile a detached image head obtained by re-encoding the last RGB frame of the decoded clean prefix. This head predictions are discarded after getting the scores.

Optimization. The frozen real-score is Wan2.1-T2V-14B, and the critic is Wan2.1-T2V-1.3B with a trainable rank-256 LoRA; the full student remains trainable. We run 600 AdamW steps of global batch size 128 at a $1{:}5$ generator/critic ratio (100 generator updates). Generator and critic-LoRA learning rates are 10^{-5} and 10^{-6}, respectively; EMA decay is 0.999. The real-data score and critic use guidance scales of 3.0 and 1.0, respectively.