FASTFRAMES: VIDEO GENERATION AT INTERACTIVE LATENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive video diffusion models generate frames based on new observations and inputs, making them a promising medium for interactive visual experiences. However, existing models are at least an order of magnitude too slow for fluid interaction. This paper introduces FASTFRAMES, a family of fast autoregressive video diffusion models that achieves interactive latency without compromising quality. We profile the inference pipeline of state-of-the-art models and find that operations often treated as fixed implementation details, such as VAE decoding and KV cache updates, in fact account for over half of end-to-end inference latency. We challenge the assumption that these components are indispensable and replace them with lightweight alternatives. Our 1.3B variant runs at 36 ms per frame (28 *non-amortized* FPS) on a single B200 GPU while matching the generation quality of a state-of-the-art model that is nearly 8× slower. Our 14B variant runs at 45 ms per frame (22 *non-amortized* FPS) on an 8× B200 GPU node and yields significantly better visual quality and realism. We show that FASTFRAMES enables real-time interaction in applications such as camera-conditioned and physics-conditioned video generation.

1 INTRODUCTION

Video diffusion models (Ho et al., 2022b;a; Singer et al., 2023; Blattmann et al., 2023) synthesize realistic videos (Wang et al., 2025; Seedance Team et al., 2026; Black Forest Labs, 2026; MiniMax, 2026), compose diverse scenes and visual concepts (Deng et al., 2025b; Chou et al., 2025; Pan et al., 2026), and respond to camera and action controls (Wang et al., 2024; He et al., 2025). In particular, autoregressive (AR) models (Jin et al., 2025; Deng et al., 2025a; Chen et al., 2025; Teng et al., 2025) generate videos incrementally, allowing them to incorporate new controls or observations. They are a promising medium for interactive visual applications, such as world simulation (Chou et al., 2026; Seo et al., 2026), games (Yu et al., 2025; Robbyant et al., 2026), and predictive models for robotics (Ye et al., 2026; NVIDIA, 2026).

Recent works have significantly accelerated AR video diffusion models, primarily through distillation and few-step sampling (Yin et al., 2024a; 2025). Some now report real-time generation speeds based on *amortized* frames-per-second (FPS) (Huang et al., 2026; Zhu et al., 2026). However, amortized FPS alone does not establish whether a model can respond to user inputs in real time. For instance, a model that generates a 30-frame clip as a single batch in one second may report 30 FPS, but the user must still wait for one second for the first visible response. Seamless real-time interactivity demands low input-to-response latency, as established in interactive systems research (Miller, 1968). Human-computer interaction studies have reported delay-detection thresholds of approximately 50 to 60 ms (Annett et al., 2014; Deber et al., 2015; Forch et al., 2017).

Thus, we target *end-to-end per-frame latency* below 50 ms, defined as the delay between the arrival of a new input signal (e.g., a steering command) and the presentation of the first RGB frame that depicts its effects. In the baseline methods we study (Yin et al., 2025; Huang et al., 2026; Zhao et al., 2026a), the inference pipeline consists of KV cache updates, diffusion sampling, and pixel rendering. Measuring latency requires accounting for the complete critical path through these operations. We also report the corresponding *non-amortized* FPS, defined as the reciprocal of this latency in seconds; 50 ms corresponds to 20 non-amortized FPS. As shown in Fig. 1, the fastest baselines remain nearly an order of magnitude slower than this target.

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.

To close this gap, we introduce FASTFRAMES, a family of autoregressive video diffusion models that, to the best of our knowledge, is the first to take end-to-end per-frame latency into the interactive regime (≤ 50 ms). We analyze the limiting factors on existing models by profiling the critical inference path of a reference 1.3B-parameter model, a common model size that balances speed and generation quality (Fig. 2). We find that dominant costs lie not in diffusion denoising steps, but rather operations often treated as fixed implementation details, such as VAE decoding and KV cache updates. For instance, a single VAE decoding step takes as long as 3.5 DiT passes.

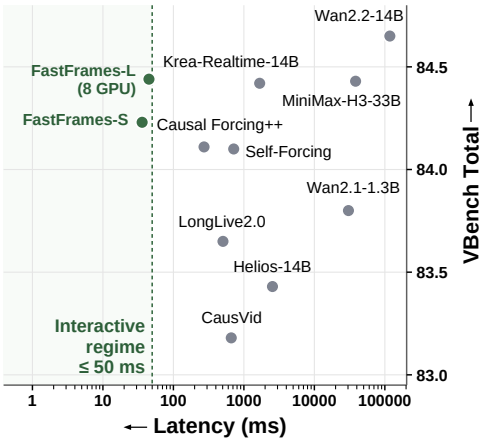


Figure 1: **Quality-speed Pareto frontier.** VBench scores vs. per-frame latency. Upper left is better. Only FASTFRAMES meets an acceptable threshold for seamless interactivity (≤ 50 ms latency or ≥ 20 *non-amortized* FPS) while matching or even exceeding comparable baselines in visual quality.

We challenge the assumption that these components are indispensable and replace them with lightweight alternatives. First, we replace the VAE decoder with a ConvNeXt-based (Liu et al., 2022) head that operates only in low-dimensional latent space, followed by a single linear layer that unpatchifies and upsamples the latents to pixel dimensions, reducing decoding latency by 147× while matching quality. Furthermore, because the head is lightweight, training the model end-to-end in pixel-space (e.g., perceptual (Zhang et al., 2018) and GAN (Goodfellow et al., 2014) losses) becomes practical, bypassing the reconstruction bottleneck of the VAE. Second, the baseline inference pipeline includes an additional DiT evaluation on each fully denoised latent solely to refine the KV cache (Sec. 3.3); we replace the DiT evaluation with a lightweight network, reducing cache-update latency 10x without measurable degradation in quality. Finally, we perform multiple system-level, hardware-aware optimizations that specifically target the end-to-end per-frame critical path. Our techniques target components shared by many AR video diffusion models and are therefore complementary to advances in backbone architecture, such as hybrid transformers (Chen et al., 2026a), and improved distillation methods (Liu et al., 2025).

In the standard text-to-video (T2V) setting, where the model simply treats the current generated frame as input for generating the next, our quantitative and qualitative evaluations show that FASTFRAMES establishes a new Pareto frontier between generation quality and speed (Fig. 1). The 1.3B variant achieves an end-to-end per-frame latency of 36 ms (28 non-amortized FPS) on a single B200 GPU at 832×480 resolution (480p), matching the visual quality of a state-of-the-art baseline (Zhao et al., 2026a) that is 7.6× slower. Scaling FASTFRAMES to 14B improves visual fidelity while achieving 45 ms (22 non-amortized FPS) on an 8× B200 GPU node.

The T2V AR pipeline can be extended to accept controls between generation steps, such as camera poses (Jiang et al., 2026), 2D tracks (Shin et al., 2026), and 3D dynamics (Li et al., 2025). We show that FASTFRAMES enables fluid interactive experiences in two applications: camera-conditioned and physics-conditioned video generation. We will release all models and code.

2 RELATED WORK

Accelerating diffusion sampling. Diffusion models require iterative sampling (Ho et al., 2020). Earlier work reduces the number of sampling steps using training-free accelerated samplers (Song et al., 2021; Lu et al., 2022; 2025) or by training models to take larger steps along their sampling trajectories (Frans et al., 2025; Geng et al., 2026; Shaul et al., 2026). The state of the art relies on distribution matching (Yin et al., 2024b;a; Cui et al., 2026), where a few-step student generator approximates the distribution of a teacher network. Another line of work focuses on the inference cost of DiTs, including hybrid attention (Li et al., 2026; Chen et al., 2026a), sliding-window attention for long context (Xiao et al., 2024; Zhang et al., 2025), and layer pruning or model-size distillation (Fang et al., 2025; Wu et al., 2025). A smaller line of work focuses specifically on com-

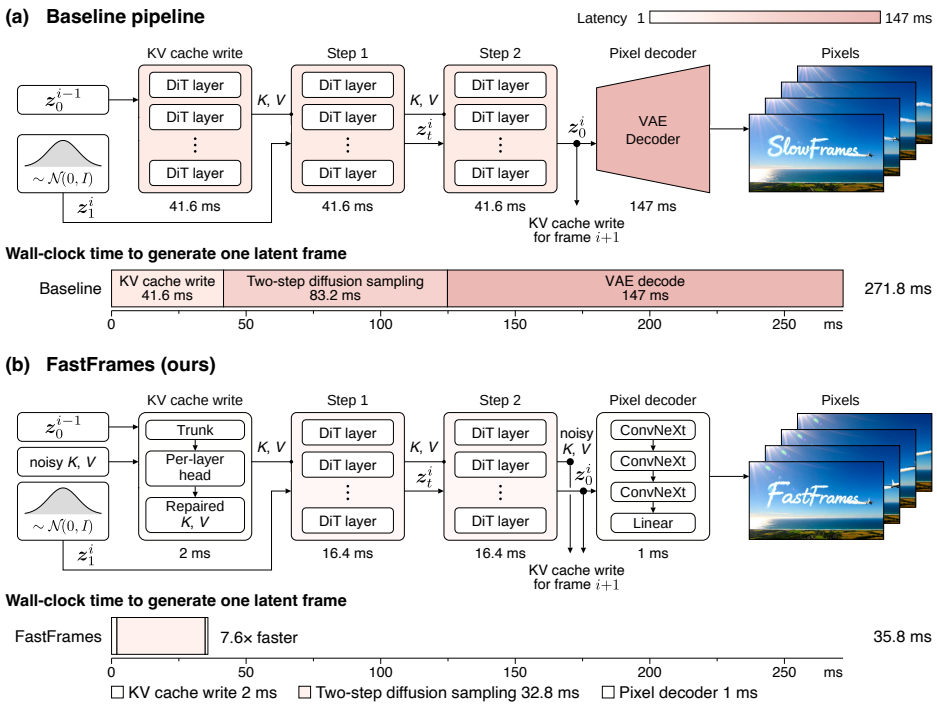


Figure 2: **Inference pipeline and latency profiling for one autoregressive generation.** State-of-the-art few-step autoregressive video diffusion models generate each new frame by first updating the KV cache using the previous frame's denoised latent (z_0^{i-1}) through a DiT evaluation, then performing multiple denoising steps followed by VAE decoding. The color gradient of the model components denotes speed: from white (fastest operations) to red (slowest). **(a)** The fastest baseline, Causal-Forcing++ (Zhao et al., 2026a), takes 271 ms to generate a frame at 480p but only 30% of wall time is spent on diffusion sampling. **(b)** FastFrames reduces latency nearly 8× by replacing the non-denoising "accessories" with lightweight alternatives and performing systems-level optimizations to speed up each DiT evaluation.

pressing or sparsifying the KV cache of autoregressive video diffusion models to reduce attention overhead (Ji et al., 2026; Liu et al., 2026a; Guo et al., 2026). However, these methods are generally model-specific, so instead we target other components on the critical path. Our work builds on state-of-the-art few-step models, but we observe that the iterative denoising process is no longer the latency bottleneck.

Reducing VAE decoding time. State-of-the-art video diffusion models perform diffusion in latent space (Blattmann et al., 2023), generally encoded via a variational autoencoder (VAE) (Kingma & Welling, 2014). For instance, the Wan (Wang et al., 2025) VAE can perform up to a 48× reduction in dimensionality. This makes generation cheaper but pixel decoding becomes expensive. VAE decoders repeatedly upsample features in space and time, and apply dense 3D convolutions at each resolution. For example, a tensor containing four 480×832 frames requires convolution at approximately 1.6 million spatiotemporal positions. A line of work aims to speed up VAEs (Bohan, 2023; Buzovkin & Shilov, 2024) by reducing the number of parameters and removing expensive operations like attention and wide 3D convolutions. These methods gain up to 4× speedup but incur degradation in visual quality (Bohan, 2025; Huang et al., 2026). Instead, we treat the pixel decoder as a lightweight readout specific to a frozen DiT, rather than a general-purpose reconstruction problem. Our pixel head maintains visual quality while being two orders of magnitude faster.

Systems-level optimizations. Systems-level techniques accelerate diffusion inference through more efficient scheduling and hardware utilization. Kodaira et al. (2025) and Bandyopadhyay et al. (2026) use batching and pipeline parallelism to improve streaming throughput but do not improve per-frame latency. StreamDiffusionV2 (Feng et al., 2025) emphasizes non-amortized time-to-first-frame but to the best of our understanding, this metric focuses on the start of a stream, in contrast to our steady-state measurements. Optimized attention kernels (Dao, 2023), compilation and operator fusion (Ansel et al., 2024), and CUDA graph execution (Ghosh et al., 2025) reduce memory traffic

and execution overhead. Weight and activation quantization (Zhao et al., 2025; Li et al., 2024b; Su et al., 2026; Zhao et al., 2026b) reduce memory requirements and can accelerate computation with suitable hardware support. Distributed inference spreads computation across multiple GPUs through patch, sequence, and pipeline parallelism (Li et al., 2024a; Fang et al., 2026; 2024). Our work incorporates many of these techniques.

3 METHOD

3.1 PRELIMINARIES: FEW-STEP AUTOREGRESSIVE VIDEO DIFFUSION MODELS

Autoregressive (AR) video diffusion models generate a video one latent chunk at a time, using diffusion to synthesize each chunk conditioned on previously generated ones. We represent a video sequence as $\boldsymbol{z}^{1:N} = (\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^N)$, where each chunk $\boldsymbol{z}^i \in \mathbb{R}^{C\times T\times H\times W}$ contains $T \geq 1$ latent frames with C channels and spatial resolution $H \times W$. Each latent frame corresponds to K RGB frames, where K is the temporal compression factor of the VAE. Generating larger chunks (e.g., one chunk = three latent frames, as in Self-Forcing (Huang et al., 2026)) is closer to bidirectional models and empirically easier to optimize, while generating a smaller chunk (e.g., one chunk = one latent frame, as in Causal Forcing++ (CF++) (Zhao et al., 2026a) and FASTFRAMES) is faster. The joint distribution factorizes autoregressively as $p(\boldsymbol{z}^{1:N}) = \prod_{i=1}^{N} p(\boldsymbol{z}^i \mid \boldsymbol{z}^{<i})$, where $\boldsymbol{z}^{<i} = (\boldsymbol{z}^1, \ldots, \boldsymbol{z}^{i-1})$ denotes the preceding chunks. Each conditional distribution is modeled by a diffusion transformer (DiT), which generates the next chunk by iteratively denoising an initial noise sample and attends to the preceding chunks via a KV cache. State-of-the-art models first convert a pretrained bidirectional teacher model into an AR model, and then distill it into a few-step AR student using distribution-matching distillation (Yin et al., 2025; Huang et al., 2026; Zhu et al., 2026). We refer readers to these works for training details.

To examine the inference cost, we profile CF++, the fastest few-step AR video diffusion model at the time of writing. It is built on the Wan 2.1 1.3B-parameter backbone. Fig. 2 shows its inference latency on a single B200 GPU, decomposed into three stages: KV-cache updates, diffusion sampling, and pixel rendering. We use its two-diffusion-step variant, which provides the best speed-quality tradeoff; reducing sampling steps to one significantly degrades visual quality while merely reducing latency by 15%. Each DiT evaluation takes 41.6 ms using BF16 precision, cuDNN attention backend (NVIDIA Corporation, 2026), and `torch.compile` (Ansel et al., 2024). The KV cache update is an additional DiT evaluation. Finally, the VAE decoder takes 147 ms to decode each generated latent into four RGB frames, resulting in a total end-to-end latency of 271.8 ms.

This profile suggests that our latency budget of 50 ms leaves room for only a single diffusion sampling step. Instead, we replace the VAE with lightweight pixel rendering (Sec. 3.2) and accelerate KV cache updates (Sec. 3.3). Then, by reducing the latency of each DiT evaluation through systems-level optimizations (Sec. 3.4), we fit the full inference pipeline within the target latency. Our final 1.3B model runs in 36 ms on a single B200 GPU, making it 7.6× faster than CF++. Our 14B model runs 10.6× faster than the fastest open-source AR model of the same parameter count and achieves a further 3.5× speedup when distributed across eight B200 GPUs.

3.2 REMOVING THE VAE DECODER

We first target the VAE decoder since it accounts for more than half of the end-to-end inference latency and alone exceeds our real-time budget 3×. The VAE decoder only has 73M parameters (compared to the DiT's 1.3B) but repeatedly upsamples its features in space and time and applies dense 3D convolutions at each resolution. For example, a tensor containing four 480×832 frames requires convolution at approximately 1.6 million spatiotemporal positions.

In light of recent advances in pixel diffusion models, we question whether such a decoder is necessary when the DiT already produces rich and spatially contextualized features, For instance, JiT (Li & He, 2026) produces sharp images by mapping between image patches and transformer features using only linear projections. Thus, we first attempt replacing the VAE with a single linear layer and train it with an MSE loss against the VAE decoded output, keeping the DiT frozen. We found that the outputs were blurry yet recognizable (Fig. 3).

We then increase the size of the pixel decoder, until arriving at our final architecture consisting of three causal 3D ConvNeXt-style blocks (Liu et al., 2022). Each block uses a causal 3×3×3 depthwise

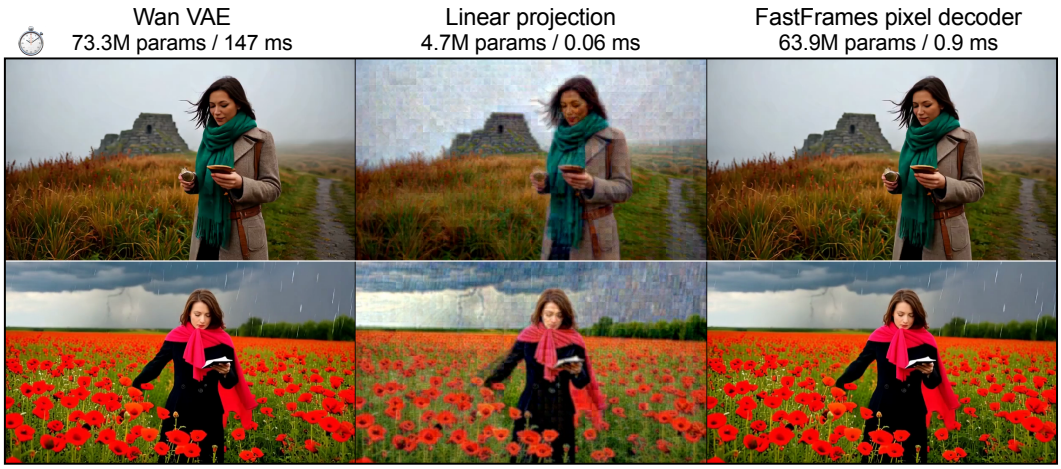


Figure 3: **Comparing pixel decoders.** Our pixel decoder (right) renders thin structures and details like hair and raindrops while being two orders of magnitude faster than Wan's VAE (left). A single linear projection (middle) cannot render sharp outputs, but is sufficient for recognizing the scene and suggests that pixel rendering does not require an expensive decoder.

convolution followed by a channel-mixing MLP. All blocks operate only on the transformer's low-resolution 21×30×52 token grid. A final linear layer converts each token into a 4×16×16 RGB patch, yielding an 84-frame, 480×832 video. We discard the first three frames following the conventions of the Wan VAE, which does not temporally upsample the first frame. The total number of parameters is comparable (63.9M for ours vs. 73.3M), yet two orders of magnitude faster (1 ms vs. 147 ms).

We train the pixel decoder using spatial and temporal reconstruction objectives. Let $\hat{y}_{\text{pix}}$ denote the RGB video predicted by the ConvNeXt decoder, and let y_{VAE} denote the output obtained by decoding the same final latent x_0 with the frozen VAE. We optimize

$$\mathcal{L}_{\text{rec}}(\hat{y}_{\text{pix}}, y_{\text{VAE}}) = \lambda_1 \mathcal{L}_1 + \lambda_{\text{MSE}} \mathcal{L}_{\text{MSE}} + \lambda_{\text{grad}} \mathcal{L}_{\text{grad}} + \lambda_{\text{temp}} \mathcal{L}_{\text{temp}} + \lambda_{\text{motion}} \mathcal{L}_{\text{motion}}. \quad (1)$$

Here, $\lambda_1 = 0.5$, $\lambda_{\text{MSE}} = 5.0$, $\lambda_{\text{grad}} = \lambda_{\text{temp}} = 0.05$, and $\lambda_{\text{motion}} = 0.5$. All five terms are computed between $\hat{y}_{\text{pix}}$ and y_{VAE}. The $\mathcal{L}_1$ and $\mathcal{L}_{\text{MSE}}$ terms enforce pixel-level fidelity, while $\mathcal{L}_{\text{grad}}$ matches spatial image gradients and $\mathcal{L}_{\text{temp}}$ matches frame-to-frame changes. The motion-weighted term $\mathcal{L}_{\text{motion}}$ emphasizes reconstruction errors in regions exhibiting motion in the VAE target. Complete definitions of the loss terms are provided in the supplement.

In addition to reducing latency, our pixel decoder is less memory-intensive to optimize. For a batch-size-one, 81-frame video at 832×480 resolution, a VAE decoder update with BF16 precision and activation checkpointing requires approximately 68 GB of peak GPU memory, compared with approximately 5 GB for our pixel decoder. This makes it practical to optimize the decoder directly in pixel space with adversarial losses to improve sharpness and realism.

We curated an internal dataset of 10K real videos and used a frozen DINOv3 (Siméoni et al., 2025) ViT-S/16 backbone as our discriminator. From layers 4, 8, and 12, we each train two lightweight token-level discriminator heads: a spatial classifier and a temporal classifier that receives both the current patch feature and its change from the preceding frame. Let $\mathcal{H}$ denote the six discriminator heads, let y denote a real video crop, and let $\hat{y}$ denote a generated crop. We use the following relativistic softplus discriminator objective (Jolicoeur-Martineau, 2018):

$$\mathcal{L}_D = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \Big\langle \text{softplus}(D_h(\hat{y}) - D_h(y)) + \lambda_{\text{center}} \left[D_h(y) + D_h(\hat{y}) \right]^2 \Big\rangle + \lambda_{\text{slope}}(R_1 + R_2). \quad (2)$$

where $\langle \cdot \rangle$ denotes averaging over clips, frames, and patch tokens, $\lambda_{\text{center}} = 0.01$, and $\lambda_{\text{slope}} = 0.25$. The terms R_1 and R_2 are finite-difference slope penalties on real and generated inputs, respectively (Lin et al., 2026). We estimate them on 25% of the clips by adding Gaussian RGB perturbations with standard deviation $\epsilon = 0.01$.

The generator remains the frozen DiT and the trainable ConvNeXt pixel decoder. We continue using the reconstruction objective (Eq. 1) with the reverse relativistic comparison:

$$\mathcal{L}_{\text{adv}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \langle \text{softplus}\left(-\left[D_h(\hat{y}) - \text{sg}(D_h(y))\right]\right) \rangle \,, \tag{3}$$

where sg denotes stop-gradient. We cap the adversarial RGB-gradient magnitude relative to the reconstruction gradient for stable optimization. We also tried finetuning the DiT with low learning rate, but observed training instability.

As seen in Fig. 3, our pixel decoder faithfully reconstructs thin structures like hair and raindrops. However, unlike the Wan VAE decoder, which can be shared across models of the same family (e.g., 1.3B and 14B variants), our pixel decoder must be finetuned for each individual model. Our main takeaway is that we can turn pixel decoding into a lightweight readout rather than a general-purpose reconstruction problem.

3.3 Faster KV Cache Updates

Autoregressive video diffusion models attend to earlier frames through a KV cache (Yin et al., 2025; Huang et al., 2026). Each DiT layer maintains its own, holding entries in an attention sink plus sliding window fashion. During each generation, the final denoising pass leaves the current chunk's cache entries computed from a noisy input. The baselines therefore run an additional DiT pass on the predicted clean latent $\hat{z}_i$ at $t = 0$ and replace those entries with its keys and values (the KV cache write in Fig. 2). They discard the pass's output prediction.

In a two-step diffusion setup, KV refinement accounts for one-third of DiT evaluations. We first investigate whether refinement is necessary. Naively removing the refinement leads to substantial degradation and training the model to use noisy cached features still produces noticeable artifacts (see supplement for visualizations). However, we hypothesize that a lightweight network can replace the expensive DiT pass. We design a KV refinement network with a shared feature extractor and per-layer KV-repair heads. The feature extractor takes $\hat{z}_i$ and applies the DiT's frozen patch embedding followed by four lightweight transformer blocks trained from scratch. Each repair head takes these output features and its layer's noisy keys and values to predict residual corrections.

During training, we freeze the DiT and use the keys and values produced by the original DiT refinement pass to supervise residual corrections to the noisy cache. We train our network with two losses: a direct KV reconstruction loss combining cosine distance and relative ℓ_2 error, and an attention-output loss that compares the outputs obtained when subsequent chunks attend to the predicted versus target cache. The network performs comparably to the original refinement pass in terms of quantitative and qualitative comparisons, while taking approximately 2 ms for the 1.3B model and 6 ms for the 14B model on a B200 GPU, around 20× and 12× faster respectively than the baselines' refinement. We provide details in the supplement.

3.4 Systems-Level Optimizations

After reducing non-denoising overhead, DiT evaluations remain the main bottleneck to real-time generation. "Free" optimizations, including faster attention backends and `torch.compile()`, are already accounted for in the baseline profiling (Fig. 2). We observe that the repeated structure of DiT evaluations offers opportunities to reuse computation. We maintain persistent KV-cache buffers and write new entries directly into their designated slots; precompute and store RoPE tables, text and timestep conditions; and fuse QKV projections into one larger matrix multiplication.

Second, we find that per-frame AR generation can provide too few query tokens to fully utilize the GPU. We therefore split the KV sequence into segments processed in parallel, then combine their attention outputs using a softmax merge. Concretely, for a full KV cache of length 32,760, we found that splitting it into five segments of 6,552 each led to a 13% latency reduction. Third, we reduce the overhead of launching and coordinating GPU work via CUDA graphs. Together, these optimizations bring the DiT evaluation latency from 41.6 ms to 16.4 ms for the 1.3B model on one B200 GPU at the full 32,760-token KV prefix and from 89.2 to 75.6 ms for the 14B model, corresponding to approximately 2.5× and 1.2× speedups.

Next, we distribute computation across 8× B200 GPUs. We experimented with tensor parallelism, attention-head parallelism, KV sharding, joint query–KV partitioning, and context parallelism (CP).

Table 1: Quantitative comparisons: end-to-end latency of one generation unit and VBench scores. All results are run with the same seeds and on B200 GPUs. All methods were evaluated at approximately 480p based on the officially supported dimensions. Green indicates latency that meets the interactive target (≤ 50 ms); yellow indicates sub-second latency ($50 <$ latency $< 1{,}000$ ms).

			VBench ↑		
Model	Size (B)	Latency (ms) ↓	Total	Quality	Semantic
Bidirectional Video Models					
Wan2.1 (Wang et al., 2025)	1.3	30,444	83.80	84.56	80.73
Wan2.2 (Wang et al., 2025)	14.3	117,734	84.65	85.41	81.60
MiniMax-H3 (MiniMax, 2026)	33.1	38,454	84.43	85.17	81.47
HunyuanVideo1.5	8.3	1,555,508	83.18	84.14	79.35
LTX2.5	19.0	11,264	83.62	84.78	78.96
Autoregressive Video Models					
MAGI1.1 Distilled (Teng et al., 2025)	23.9	19,736	82.04	83.40	76.60
Helios Distilled (Yuan et al., 2026)	14.3	2,544	83.43	84.53	79.03
Krea-Realtime (Millon, 2025)	14.3	1,675	84.42	85.08	81.81
LongLive2.0 (Chen et al., 2026b)	5.0	505	83.65	84.54	80.09
CausVid (Yin et al., 2025)	1.3	662	83.18	84.15	79.31
Self-Forcing* (Huang et al., 2026)	1.3	717	84.10	84.84	81.13
Causal Forcing++ (Zhao et al., 2026a)	1.3	272	84.11	85.09	80.20
FastFrames-S	1.3	36	84.23	85.30	79.92
FastFrames-S (8 GPU)	1.3	14	84.23	85.30	79.92
FastFrames-L	14.3	158	84.44	85.48	80.29
FastFrames-L (8 GPU)	14.3	45	84.44	85.48	80.29

*Self-Forcing reports latency averaged over a sequence, including cheaper early frames as the KV cache fills. We measure steady-state latency with a full cache. See the supplement for details.

Among them, only CP reduced computation sufficiently to justify communication overhead. CP partitions query tokens across GPUs and distributes both attention and feed-forward computation and requires only one exchange of computed keys and values per transformer layer. We implement these exchanges using CUDA-graph-compatible NVLink symmetric-memory gathers. This configuration reduces DiT latency from 16.4 to 5.8 ms for the 1.3B model and from 75.6 to 20.3 ms for the 14B model, corresponding to 2.8× and 3.7× speedups over their optimized single-GPU implementations.

4 Experiments

Models and baselines. We build FastFrames on the Wan 2.1 backbone at two model sizes: FastFrames-S with 1.3B parameters and FastFrames-L with 14B. We initialize FastFrames-S from the two-step Causal Forcing++ (CF++) checkpoint and finetune it using our method. We freeze the DiT and train our pixel decoder and KV refinement network against their respective targets. FastFrames-L required starting from the base bidirectional model. Following the training recipe in CF++, we converted Wan 2.1 14B into an autoregressive model, distilled it into few-step, and trained it with DMD. Both FastFrames-S and FastFrames-L perform two-step diffusion sampling and generate one latent frame per autoregressive step.

Quantitative comparisons. Tab. 1 provides a quantitative comparison against bidirectional video generators and autoregressive models. The bidirectional models provide a broader reference for generation quality and latency, while the autoregressive models provide direct comparisons. We report model size, latency, and VBench (Huang et al., 2024) scores. All models are evaluated using the same VBench prompts and seeds. We report end-to-end latency, following the definition in the introduction. We account for diffusion sampling, KV cache updates where applicable, and pixel rendering. For AR models, we measure latency at steady state after warm-up and with a full KV cache. For bidirectional models, we let the model warm up and then generate the configured native clip lengths. All models are evaluated on a B200 GPU unless otherwise specified, at approximately 480p using the model's officially supported resolutions.

We find that FastFrames delivers slightly higher quality than the corresponding baselines with the same Wan 2.1 backbone (Krea-Realtime vs. FastFrames-L, CF++ vs. FastFrames-S). Between

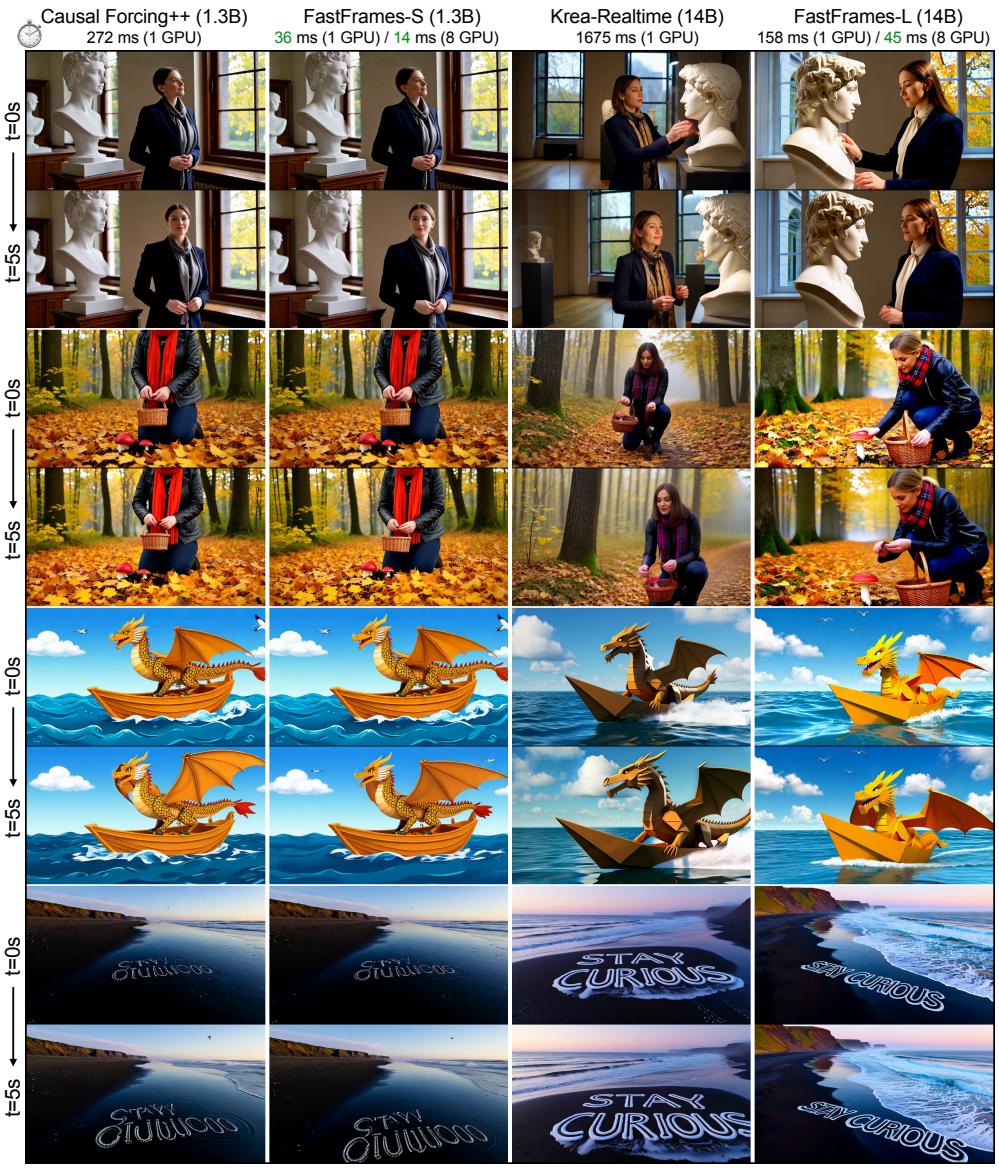
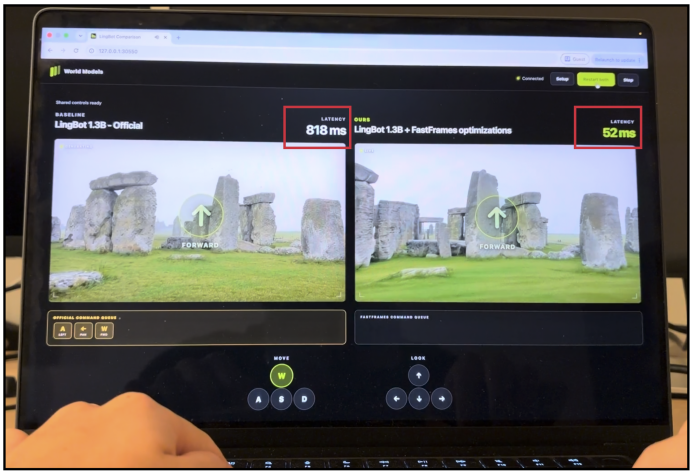


Figure 4: **Qualitative comparisons.** We compare FastFrames-S with Causal Forcing++ at 1.3B parameters and FastFrames-L with Krea-Realtime at 14B. Each example shows frames at 0 and 5 seconds. FastFrames-S achieves comparable visual sharpness to Causal Forcing++, such as the details in the sculpture (first example) and leaves (second example), while being nearly 8× faster. FastFrames-L achieves comparable visual realism to Krea-Realtime, including composition of "paper origami dragon" (third), and legible text of "stay curious" (fourth) while being 10× faster. See our website (https://anonymous4563d.github.io/) for video playback.

FastFrames-S and CF++, which use the same DiT weights, we attribute the higher visual quality to end-to-end adversarial training in pixel space, which boosts realism and sharpness.

None of the baselines meet the 50 ms latency target required for interactive applications. The fastest baseline, CF++, is 5.4× slower than the threshold. In contrast, FastFrames-S attains a latency of 36 ms on a single B200 GPU and FastFrames-L attains 45 ms on an 8-GPU node.

Qualitative comparisons. Fig. 4 shows a side-by-side comparison between FastFrames-S, FastFrames-L, and their respective baselines. FastFrames-S maintains visual sharpness relative to CF++, capturing fine details like hair and raindrops while being nearly 8× faster. FastFrames-L achieves realism, prompt adherence, composition, and text rendering capabilities comparable to Krea while being over 10× faster.



Configuration	Latency (ms)
LingBot 1.3B	
Official inference - 1 GPU	1085
Official + Ulysses - 2 GPU	842
Official + Ulysses - 4 GPU	865
W/ FastFrames - 1 GPU	224
W/ FastFrames - 8 GPU	52
LingBot 14B	
Official inference - 1 GPU	3862
Official + Ulysses - 8 GPU	1373
W/ FastFrames - 1 GPU	1263
W/ FastFrames - 8 GPU	233
RealWonder 1.3B	
Official inference - 1 GPU	1141
W/ FastFrames - 1 GPU	153
W/ FastFrames - 8 GPU	51

Figure 5: **Interactive generation and latency results.** *Left:* example of interaction with LingBot's official implementation vs. with FastFrames optimizations. Given a single keypress, FastFrames is 16× faster to respond. *Right:* generation latency of officially released implementations and our optimized implementations. We measure steady-state latency on NVIDIA B200 GPUs at 832×480 resolution with a full KV cache and include distributed optimizations such as Ulysses where officially supported. Times include diffusion sampling (four steps for LingBot and three for RealWonder), action conditioning, KV-cache updates, and pixel decoding. Our optimizations provide speedups of 3× to 22×. See the anonymous website for the video demonstration.

Fig. 4 also illustrates the improved capabilities of the 14B models relative to their 1.3B counterparts. The larger models produce more realistic lighting and shadows in the first two examples, better capture the folded-paper appearance of the origami dragon in the third, and render more legible text in the fourth. We encourage readers to view the supplementary videos on the anonymous website.

Interactive applications. We showcase FastFrames in interactive settings by accelerating camera-conditioned and physics-conditioned video generation: LingBot (Gao et al., 2026), a family of camera-conditioned video models with 1.3B and 14B variants, and RealWonder (Liu et al., 2026b), a 1.3B physics-conditioned video model. Our techniques transfer directly to these off-the-shelf models and yield large speedups, even over official code that already includes optimizations such as Ulysses sequence parallelism (Jacobs et al., 2024). Fig. 5 summarizes the latency profiles.

On LingBot 1.3B, we reduce latency from 842 ms (the fastest official configuration, 2 GPUs) to 52 ms on 8 GPUs, a 16× speedup. On RealWonder, latency drops from 1141 ms (official, 1 GPU) to 51 ms (22×), so both models approximately reach our target interactive latency. On LingBot 14B, latency drops from 1373 ms to 233 ms on 8 GPUs (5.9×). These models remain slower than our T2V models because they require more diffusion sampling steps, predict multiple latent frames per autoregressive step, and need extra DiT computation to process additional conditioning.

We show a side-by-side comparison between the interactive demos on the anonymous website; the original models exhibit noticeable lag while our optimized versions of LingBot 1.3B and RealWonder respond to user inputs fluidly.

5 Conclusion

We presented FastFrames, a family of autoregressive video diffusion models that achieves interactive latency by rethinking components of the inference pipeline often treated as fixed implementation details. The generation quality of our 1.3B and 14B variants are both comparable to or better than their respective baselines.

While our evaluation focuses on text-to-video generation at 480p on high-end GPUs, important directions for future work include adding action conditioning without increasing latency, scaling to higher resolutions, and supporting lighter hardware form factors. Improving long-horizon consistency and physical plausibility will further broaden the utility of video world models.

AI USE STATEMENT

Paper writing. The initial layout and drafting of the paper, all technical and conceptual details including equations, literature survey, and the final polishing and editing were all completed by the (human) authors. LLMs were used for brainstorming paper structure, as advanced thesauruses for wording and phrasing suggestions, for verifying citations (and then painstakingly double-checked for hallucinations by humans), and for formatting tables and prototyping figures.

Research. LLMs and coding agents were used to design and provide feedback on research methodology and experiments, and to implement methods and code. The authors have verified the correctness of the code and results and take responsibility for all presented content in this paper.

REFERENCES

Michelle Annett, Albert Ng, Paul Dietz, Walter F. Bischof, and Anoop Gupta. How low should we go? Understanding the perception of latency while inking. In *Graphics Interface*, 2014.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *ASPLOS*, 2024.

Hmrishav Bandyopadhyay, Nikhil Pinnaparaju, Rahim Entezari, Jim Scott, Yi-Zhe Song, and Varun Jampani. Block Cascading: Training free acceleration of block-causal video models. In *CVPR*, 2026.

Black Forest Labs. FLUX 3 – real world models: Towards multimodal flow models as the backbone of visual intelligence, July 2026. URL https://bfl.ai/blog/flux-3. Model release.

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv:2311.15127*, 2023.

Ollin Boer Bohan. TAESD: Tiny autoencoder for stable diffusion. https://github.com/madebyollin/taesd, 2023. GitHub repository.

Ollin Boer Bohan. TAEHV: Tiny autoencoder for hunyuan video. https://github.com/madebyollin/taehv, 2025.

Alexey Buzovkin and Evgeny Shilov. Toward lightweight and fast decoders for diffusion models in image and video generation. *arXiv:2503.04871*, 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:2504.13074*, 2025.

Junsong Chen, Yuyang Zhao, Jincheng Yu, Ruihang Chu, Junyu Chen, Shuai Yang, Xianbang Wang, Yicheng Pan, Daquan Zhou, Huan Ling, Haozhe Liu, Hongwei Yi, Hao Zhang, Muyang Li, Yukang Chen, Han Cai, Sanja Fidler, Ping Luo, Song Han, and Enze Xie. SANA-Video: Efficient video generation with block linear diffusion transformer. In *ICLR*, 2026a.

Yukang Chen, Luozhou Wang, Wei Huang, Shuai Yang, Bohan Zhang, Yicheng Xiao, Ruihang Chu, Weian Mao, Qixin Hu, Shaoteng Liu, et al. Longlive-2.0: An NVFP4 parallel infrastructure for long video generation. *arXiv:2605.18739*, 2026b.

Gene Chou, Kai Zhang, Sai Bi, Hao Tan, Zexiang Xu, Fujun Luan, Bharath Hariharan, and Noah Snavely. Generating 3D-consistent videos from unposed internet photos. In *CVPR*, 2025.

Gene Chou, Charles Herrmann, Kyle Genova, Boyang Deng, Songyou Peng, Bharath Hariharan, Jason Y. Zhang, Noah Snavely, and Philipp Henzler. CityRAG: Stepping into a city via spatially-grounded video generation. *arXiv:2604.19741*, 2026.

Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Self-Forcing++: Towards minute-scale high-quality video generation. In *ICLR*, 2026.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv:2307.08691*, 2023.

Jonathan Deber, Ricardo Jota, Clifton Forlines, and Daniel Wigdor. How much faster is fast enough? User perception of latency & latency improvements in direct and indirect touch. In *CHI*, 2015.

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*, 2025a.

Yufan Deng, Yuanyang Yin, Xun Guo, Yizhi Wang, Jacob Zhiyuan Fang, Shenghai Yuan, Yiding Yang, Angtian Wang, Bo Liu, Haibin Huang, and Chongyang Ma. MAGREF: Masked guidance for any-reference video generation with subject disentanglement. *arXiv:2505.23742*, 2025b.

Gongfan Fang, Kunjun Li, Xinyin Ma, and Xinchao Wang. TinyFusion: Diffusion transformers learned shallow. In *CVPR*, 2025.

Jiarui Fang, Jinzhe Pan, Xibo Sun, Aoyu Li, and Jiannan Wang. xDiT: An inference engine for diffusion transformers (DiTs) with massive parallelism. *arXiv:2411.01738*, 2024.

Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. PipeFusion: Patch-level pipeline parallelism for diffusion transformers inference. In *NeurIPS*, 2026.

Tianrui Feng, Zhi Li, Shuo Yang, Haocheng Xi, Muyang Li, Xiuyu Li, Lvmin Zhang, Keting Yang, Kelly Peng, Song Han, Maneesh Agrawala, Kurt Keutzer, Akio Kodaira, and Chenfeng Xu. StreamDiffusionV2: A streaming system for dynamic and interactive video generation. In *MLSys*, 2025.

Valentin Forch, Thomas Franke, Nadine Rauh, and Josef F. Krems. Are 100 ms fast enough? Characterizing latency perception thresholds in mouse-based interaction. In *HCI*, 2017.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *ICLR*, 2025.

Zelin Gao, Qiuyu Wang, Jiapeng Zhu, Jingye Chen, Zichen Liu, Qingyan Bai, Jiahao Wang, Yufeng Yuan, Hanlin Wang, Yichong Lu, Ka Leong Cheng, Haojie Zhang, Jian Gao, Tianrui Feng, Yuzheng Liu, Yao Yao, Yinghao Xu, Xing Zhu, Yujun Shen, and Hao Ouyang. Infinite worlds with versatile interactions. *arXiv:2607.07534*, 2026.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *NeurIPS*, 2026.

Abhishek Ghosh, Ajay Nayak, Ashish Panwar, and Arkaprava Basu. PyGraph: Robust compiler support for CUDA graphs in PyTorch. *arXiv:2503.19779*, 2025.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

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

Hao He, Yinghao Xu, Yuwei Guo, Gordon Wetzstein, Bo Dai, Hongsheng Li, and Ceyuan Yang. CameraCtrl: Enabling camera control for video diffusion models. In *ICLR*, 2025.

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

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. Imagen Video: High definition video generation with diffusion models. *arXiv:2210.02303*, 2022a.

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

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self Forcing: Bridging the train-test gap in autoregressive video diffusion. *NeurIPS*, 2026.

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.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. System optimizations for enabling training of extreme long sequence transformer models. In *PODC*, 2024.

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

Fan Jiang, Zhaoxu Sun, Mengchao Wang, Ziyu Zhu, Chiyu Wang, Yunpeng Zhang, Wenlin Liu, Yun Wang, Xue Zheng, Rui Sun, Junfeng Ni, Hongyu Pan, Zhongxu Sun, Fei Yu, Zengye Ge, Mengmeng Du, Nianfei Fan, Mingchao Sun, Yu Liu, Yongchang, Yanqing Zhu, Jiahang Wang, Ning Ying, Yuze Xuan, Di Yang, Zhicheng Liu, Zhe Gao, Tingbing Xu, Jiacheng Sui, Wenjin Yang, Junnan Lai, Shufeng Liu, Yuan Liu, Zheng Zhou, Yingliang Peng, Dawei Cao, Kaifeng Sheng, Yuxiang Cai, Fei Lu, Mu Xu, and Ning Guo. ABot-World-0: Infinite interactive world rollout on a single desktop GPU, 2026. URL https://arxiv.org/abs/2607.19191.

Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, 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.

Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv:1807.00734*, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Akio Kodaira, Chenfeng Xu, Toshiki Hazama, Takanori Yoshimoto, Kohei Ohno, Shogo Mitsuhori, Soichi Sugano, Hanying Cho, Zhijian Liu, Masayoshi Tomizuka, and Kurt Keutzer. StreamDiffusion: A pipeline-level solution for real-time interactive generation. In *ICCV*, 2025.

Kunyang Li, Mubarak Shah, and Yuzhang Shang. Attend locally, remember linearly: Linear attention as cross-frame memory for autoregressive video diffusion. *arXiv:2605.16579*, 2026.

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

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv:2411.05007*, 2024b.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *CVPR*, 2026.

Zizhang Li, Hong-Xing Yu, Wei Liu, Yin Yang, Charles Herrmann, Gordon Wetzstein, and Jiajun Wu. WonderPlay: Dynamic 3D scene generation from a single image and actions. In *ICCV*, 2025.

Shanchuan Lin, Ceyuan Yang, Zhijie Lin, Hao Chen, and Haoqi Fan. Adversarial flow models. In *ICML*, 2026.

Dongyang Liu, Peng Gao, David Liu, Ruoyi Du, Zhen Li, Qilong Wu, Xin Jin, Sihan Cao, Shifeng Zhang, Hongsheng Li, et al. Decoupled DMD: CFG augmentation as the spear, distribution matching as the shield. *arXiv:2511.22677*, 2025.

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

Wei Liu, Ziyu Chen, Zizhang Li, Yue Wang, Hong-Xing Yu, and Jiajun Wu. Realwonder: Real-time physical action-conditioned video generation. *arXiv:2603.05449*, 2026b.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *CVPR*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. *NeurIPS*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver++: Fast solver for guided sampling of diffusion probabilistic models. *Mach. Intell. Res.*, 22(4), 2025.

Robert B. Miller. Response time in man-computer conversational transactions. In *AFIPS Fall Joint Comp. Conf.*, 1968.

Erwann Millon. Krea realtime 14B: Real-time video generation, 2025. URL https://github.com/krea-ai/realtime-video.

MiniMax. MiniMax H3: An open model breaking the boundaries between tasks and modalities, July 2026. URL https://www.minimax.io/blog/minimax-h3. Model release.

NVIDIA. Cosmos 3: Omnimodal world models for physical ai. *arXiv:2606.02800*, 2026.

NVIDIA Corporation. NVIDIA cuDNN: Scaled dot product attention. https://docs.nvidia.com/deeplearning/cudnn/latest/operations/Attention.html, 2026. cuDNN documentation, accessed September 2, 2026.

Kaihang Pan, Qi Tian, Jianwei Zhang, Weijie Kong, Jiangfeng Xiong, Yanxin Long, Shixue Zhang, Haiyi Qiu, Tan Wang, Zheqi Lv, Yue Wu, Liefeng Bo, Siliang Tang, and Zhao Zhong. OmniWeaving: Towards unified video generation with free-form composition and reasoning. *arXiv:2603.24458*, 2026.

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

Seedance Team et al. Seedance 2.0: Advancing video generation for world complexity. *arXiv:2604.14148*, 2026.

Junyoung Seo, Hyunwook Choi, Minkyung Kwon, Jinhyeok Choi, Siyoon Jin, Gayoung Lee, Junho Kim, JoungBin Lee, Geonmo Gu, Dongyoon Han, Sangdoo Yun, Seungryong Kim, and Jin-Hwa Kim. Seoul world model: Grounding world simulation models in a real-world metropolis. In *ECCV*, 2026.

Neta Shaul, Chao Liu, Arash Vahdat, and Julius Berner. Parallel decoding distillation for fast image and video generation. *arXiv:2607.26004*, 2026.

Joonghyuk Shin, Zhengqi Li, Richard Zhang, Jun-Yan Zhu, Jaesik Park, Eli Shechtman, and Xun Huang. MotionStream: Real-time video generation with interactive motion controls. In *ICLR*, 2026.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3. *arXiv:2508.10104*, 2025.

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.

Rundong Su, Jintao Zhang, Zhihang Yuan, Haojie Duanmu, Jianfei Chen, and Jun Zhu. 6Bit-Diffusion: Inference-time mixed-precision quantization for video diffusion models. *arXiv:2603.18742*, 2026.

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:2505.13211*, 2025.

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:2503.20314*, 2025.

Zhouxia Wang, Ziyang Yuan, Xintao Wang, Yaowei Li, Tianshui Chen, Menghan Xia, Ping Luo, and Ying Shan. MotionCtrl: A unified and flexible motion controller for video generation. In *SIGGRAPH*, 2024.

Yiming Wu, Zhenghao Chen, Huan Wang, and Dong Xu. Individual content and motion dynamics preserved pruning for video diffusion models. In *MM*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies. *arXiv:2602.15922*, 2026.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. In *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, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *CVPR*, 2025.

Jiwen Yu, Yiran Qin, Xintao Wang, Pengfei Wan, Di Zhang, and Xihui Liu. GameFactory: Creating new games with generative interactive videos. *arXiv:2501.08325*, 2025.

Shenghai Yuan, Yuanyang Yin, Zongjian Li, Xinwei Huang, Xiao Yang, and Li Yuan. Helios: Real real-time long video generation model. *arXiv preprint arXiv:2603.04379*, 2026.

Peiyuan Zhang, Yongqi Chen, Runlong Su, Hangliang Ding, Ion Stoica, Zhengzhong Liu, and Hao Zhang. Fast video generation with sliding tile attention. In *ICML*, 2025.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

Min Zhao, Hongzhou Zhu, Kaiwen Zheng, Zihan Zhou, Bokai Yan, Xinyuan Li, Xiao Yang, Chongxuan Li, and Jun Zhu. Causal Forcing++: Scalable few-step autoregressive diffusion distillation for real-time interactive video generation. *arXiv:2605.15141*, 2026a.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Enshu Liu, Rui Wan, Widyadewi Soedarmadji, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, Huazhong Yang, Xuefei Ning, and Yu Wang. ViDiT-Q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *ICLR*, 2025.

Yuyang Zhao, Yicheng Pan, Qiyuan He, Jincheng Yu, Junsong Chen, Tian Ye, Haozhe Liu, Enze Xie, and Song Han. SANA-Streaming: Real-time streaming video editing with hybrid diffusion transformer. *arXiv:2605.30409*, 2026b.

Hongzhou Zhu, Min Zhao, Guande He, Hang Su, Chongxuan Li, and Jun Zhu. Causal Forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation. *arXiv:2602.02214*, 2026.

APPENDIX

We organize our Appendix with the following sections: A) Limitations and Future Work; B) Additional Latency Profiling; C) Architecture and Training Details of Pixel Decoder; D) Architecture and Training Details of KV Refinement Network; E) Adaptation Details of Interactive Applications.

We also encourage readers to view our videos on our anonymous website (https://anonymous4563d.github.io/).

A LIMITATIONS AND FUTURE WORK

Resolution. Most baselines and base video models use 480p resolution, which we finetune from. Higher resolution like 720p or even 4k is desired for a variety of applications like gaming and content creation. In the next section, we find that increasing resolution to 720p increases latency significantly; further optimization is required to bring high-res video generation into the interactive regime.

Accessible Hardware. We use B200 GPUs for our latency measurements in the main paper. While we assume compute will continue to become cheaper and more accessible, being able to run interactive video models on-device is an exciting possibility.

Action Conditions Designed for Lower Latency. We applied FASTFRAMES to off-the-shelf autoregressive models that were slower than their T2V counterparts. Training base models with lower latency in mind, such as concatenating camera poses in the channel dimension rather than the sequence dimension, without compromising quality, is essential for reducing latency.

B ADDITIONAL LATENCY PROFILING

In Tab. 2, we include detailed profiling of each operation on B200 and H100 GPUs, and at 480p and 720p resolution. In short, FASTFRAMES-S runs at 36 ms at 480p and 127 ms at 720p on a B200 GPU, and FASTFRAMES-L runs at 45 ms at 480p and 116 ms at 720p on 8 B200 GPUs. Savings

Configuration	Resolution	B200 Latency (ms)	H100 Latency (ms)
FASTFRAMES-S (1.3B) on 1 GPU			
DiT evaluation	832×480	16	25
DiT evaluation	1280×720	60	101
KV refinement network	832×480	2	3
KV refinement network	1280×720	5	7
VAE	832×480	147	175
VAE	1280×720	327	408
Our pixel decoder	832×480	1	1
Our pixel decoder	1280×720	2	3
FASTFRAMES-L (14B) on 8 GPUs			
DiT evaluation	832×480	20	30
DiT evaluation	1280×720	54	67
KV refinement network	832×480	3	5
KV refinement network	1280×720	7	12
Our pixel decoder	832×480	1	1
Our pixel decoder	1280×720	1	1

Table 2: **Additional latency profiling at 720p resolution and on H100 GPUs**. In short, FASTFRAMES-S runs at 36 ms at 480p and 127 ms at 720p on a B200 GPU. FASTFRAMES-L runs at 45 ms at 480p and 116 ms at 720p on 8 B200 GPUs. Savings from our lightweight network replacements are more noticeable at 720p. H100s are approximately 25% to 50% slower depending on the configuration. Distributing the VAE across 8 GPUs is non-trivial; we leave it for future work.

from our lightweight networks are more noticeable at 720p. H100s are approximately 25% to 50% slower depending on the configuration.

Some baselines like Self-Forcing (Huang et al., 2026) report latency in their paper. However, we did not find detailed descriptions or breakdowns of those numbers and can only infer their profiling from source code. We found that Self-Forcing measured latency by generating a video sequence of N latent chunks (where each chunk ≥ 1 latent frame(s)), then dividing the total time by the number of chunks. However, earlier chunks are faster to generate due to an initially empty KV cache that gradually fills, and therefore does not represent steady-state latency at long contexts. Occasionally, baselines provide ambiguous latency numbers without explanation; Causal Forcing++ (Zhao et al., 2026a) reports 270 ms for its 1, 2, and 4 step models without explanation.

All of our profiling numbers include a breakdown of the specific operation, and is measured at steady-state with a full KV cache where applicable based on officially released configurations (e.g., KV=32760 for Self-Forcing and its follow-up works).

C ARCHITECTURE AND TRAINING DETAILS OF PIXEL DECODER

Our pixel decoder is a causal ConvNeXt model operating directly on the pre-head features of the frozen two-step DiTs. For a 21-frame latent sequence, the DiT produces 32,760 tokens on a $(21 \times 30 \times 52)$ spatiotemporal grid with channel width 1536. The head applies per-frame adaptive shift/scale modulation, a (1×1) channel projection, and three ConvNeXt-style causal 3-D residual blocks—two low-resolution blocks followed by one output block—using depthwise $(3 \times 3 \times 3)$ convolutions, channel-wise layer normalization, and width-4 pointwise GELU expansions. All convolutional processing remains at the DiT token resolution; the selected configuration has no learned temporal or spatial upsampler, latent adapter, displacement branch, auxiliary feature fusion, or post-RGB refiner. A final per-token linear projection maps the 1536-dimensional features to $(4 \times 16 \times 16 \times 3)$ RGB values, which are unpatchified into 84 frames at (480×832) and aligned to the VAE convention by dropping the first three frames, yielding 81 output frames. The decoder contains 63.87M parameters and is causal in time, so each output block depends only on the current and preceding latent blocks.

We train the pixel decoder using a paired spatial-temporal reconstruction objective. Let $\hat{y}_{\text{pix}}$ denote the RGB video predicted by the ConvNeXt decoder, and let y_{VAE} denote the output obtained by decoding the same final latent x_0 with the frozen VAE. Both videos are represented in the range

$[-1, 1]$. Our reconstruction objective is

$$\mathcal{L}_{\text{rec}} = 0.5\,\mathcal{L}_1 + 5.0\,\mathcal{L}_{\text{MSE}} + 0.05\,\mathcal{L}_{\text{grad}} + 0.05\,\mathcal{L}_{\text{temp}} + 0.50\,\mathcal{L}_{\text{motion}}. \tag{4}$$

The pixelwise terms are $\mathcal{L}_1 = \text{mean}(|\hat{y}_{\text{pix}} - y_{\text{VAE}}|)$ and $\mathcal{L}_{\text{MSE}} = \text{mean}((\hat{y}_{\text{pix}} - y_{\text{VAE}})^2)$. To preserve spatial structure, we match horizontal and vertical finite differences finite differences:

$$\mathcal{L}_{\text{grad}} = \text{mean}(|\Delta_h \hat{y}_{\text{pix}} - \Delta_h y_{\text{VAE}}|) + \text{mean}(|\Delta_w \hat{y}_{\text{pix}} - \Delta_w y_{\text{VAE}}|)\,. \tag{5}$$

To preserve temporal dynamics, we match consecutive-frame differences:

$$\mathcal{L}_{\text{temp}} = \text{mean}(|\Delta_t \hat{y}_{\text{pix}} - \Delta_t y_{\text{VAE}}|)\,. \tag{6}$$

Finally, we place greater reconstruction weight on regions that move in the VAE target:

$$\mathcal{L}_{\text{motion}} = \text{mean}_{t>1,p}[(1 + \gamma\ \text{mean}_c\,|\Delta_t y_{\text{VAE},t,c,p}|)\,\text{mean}_c\,|\hat{y}_{\text{pix},t,c,p} - y_{\text{VAE},t,c,p}|]\,, \quad \gamma = 10. \tag{7}$$

Here, Δ_h, Δ_w, and Δ_t denote vertical, horizontal, and temporal finite-difference operators, respectively; c indexes RGB channels; and p indexes spatial locations. The multiplier γ controls how strongly moving target regions are emphasized within $\mathcal{L}_{\text{motion}}$, while the coefficient 0.50 controls the overall contribution of this term to $\mathcal{L}_{\text{rec}}$.

Using the `vidprom_filtered_extended.txt` file provided in Self-Forcing's repo, we rollout the frozen DiT and VAE 150k times to get the input and output targets, running 4 optimization steps per rollout. We train on 32 B200 GPUs for about a week; we find that validation loss and metrics (PSNR and SSIM) continue to improve steadily though slowly.

We take the last checkpoint and train with the adversarial setup as described in the main paper (Sec. 3.2). We train on 8 B200 GPUs and run 5000 rollouts per GPU and 4 optimization steps per rollout. Training completes in a few hours.

We will release all checkpoints and training code.

D Architecture and Training Details of KV Refinement Network

As described in the main paper, we first investigate whether KV refinement is necessary, given that the last diffusion sampling step writes a noisy KV cache for free. In Fig. 6, we find that removing KV refinement, even after training the model to learn to use the noisy KV cache, leads to drift and artifacts. These experiments motivate our design.

Our KV refinement network is a cache-repair module that replaces the expensive DiT evaluation of the baselines. Each DiT layer maintains its own KV cache, partitioned into one slice for each generated latent frame. During every diffusion evaluation for frame i, the KV tensors for that frame overwrite its current cache slice, while the slices for earlier frames remain available as causal context. At the final denoising step, the DiT still receives a nonzero-noise latent, although it predicts the clean latent $\boldsymbol{z}_0$; consequently, the KV written to the current slice correspond to the noisy input. The baseline therefore performs an additional, non-denoising DiT forward pass on $\boldsymbol{z}_0$ with zero-noise time conditioning ($t = 0$), overwriting the same slice with clean-context KV.

Instead, our network consumes the noisy K/V tensors from the final denoising pass together with the predicted $\boldsymbol{z}_0$, predicts corrected K/V tensors, and overwrites the same cache slice. Subsequent latent frames therefore attend to the repaired context without requiring the additional full DiT forward pass. A shared stem maps the 1,560 patch tokens per frame from 1,536 to 768 channels, then applies four lightweight 8-head spatial self-attention blocks with 4× GELU MLPs and timestep-conditioned adaptive layer normalization. Each of the DiT layers has a separate repair head that projects its noisy K/V, concatenates them with the shared per-token $\boldsymbol{z}_0$ features, and predicts residual corrections. Keys are corrected in post-RoPE space and renormalized per attention head, while values receive a learned residual and per-channel scale. All residual outputs are zero-initialized so the adapter initially behaves like the no-prime baseline.

During inference, frame 0 retains a real clean-prime anchor, while subsequent frames use the repaired K/V to overwrite their cache slots before the next frame is processed. The adapter is trained with the DiT and VAE frozen using teacher-forced targets from the real prime pass: the primary



Figure 6: **KV refinement ablation.** AR baselines refine the KV cache using an additional DiT evaluation, which is costly but effective. Removing that refinement naively (third column) substantially degrades the scene's consistency; the objects in the scenes change over time. Even training without the refinement (fourth column) does not fix the problem. Either the subject changes over time (the face changes in the second example), or noticeable artifacts appear (clothing in the first example, fingers in the second example). Our KV refinement network (second column) replaces the DiT evaluation and performs equally while being an order of magnitude faster.

objective matches future attention outputs, supplemented by direct K/V reconstruction with approximately three times more weight on value than key errors, since we found that value corruption dominates the degradation.

E Adaptation Details of Interactive Applications

We use the official model and weights released by the authors. We follow the procedure described in the paper to train both the pixel decoder and the KV refinement network. Neither component requires large-scale training data. We simply rollout the models to create target pixels and clean key and values on-the-fly.

LingBot requires four diffusion sampling steps per autoregressive generation, and generates four latent frames at once. RealWonder requires three sampling steps and generates three latent frames at once. Both are substantially more expensive than our setup in FastFrames, which performs two sampling steps, and generates one latent frame at a time. Thus, even the 1.3B models require 8 GPUs to reach interactive latency. RealWonder is slower than LingBot at the same parameter count due to its more expensive processing of its action conditions. Distilling these models to fewer steps and fewer latent frames per generation while maintaining output quality and adherence to conditions is important future work.