VideoMLA: Low-Rank Latent KV Cache for Minute-Scale Autoregressive Video Diffusion

Hidir Yesiltepe[1] **Jiazhen Hu**[1] **Tuna Han Salih Meral**[1] **Adil Kaan Akan**[2]
Kaan Oktay[2] **Hoda Eldardiry**[1] **Pinar Yanardag**[1]

[1]Virginia Tech [2]fal

Abstract

Long-rollout causal video diffusion has converged on a fixed-size sliding-window KV cache, with recent progress innovating *within* this layout by changing which tokens occupy the window or how their positions are encoded. The per-head KV layout itself, a dominant contributor to streaming memory and latency, has been mostly left unchanged. In this paper, we present the first study of Multi-Head Latent Attention (MLA) in video diffusion. VideoMLA replaces per-head keys and values with a shared low-rank content latent and a shared decoupled 3D-RoPE positional key, reducing per-token KV memory by 92.7% at every cached layer. We further investigate why MLA succeeds in video diffusion even though the spectral assumption often used to motivate it in language models does not hold: pretrained video attention is not low-rank, with 99%-energy effective rank far above any practical latent dimension. VideoMLA retains quality at compression ratios where direct spectral approximation would predict large reconstruction error. We show that the MLA bottleneck, rather than the pretrained spectrum, determines the effective rank: both spectral and random initialization occupy nearly the full rank budget from initialization, and Stage-1 training preserves this budget while adapting within it. On VBench, VideoMLA matches short-horizon streaming video diffusion baselines, achieves the best overall score at long horizons among evaluated methods, and improves throughput by $1.23\times$ on a single B200.

1 Introduction

Causal video diffusion models [9, 4, 17, 38, 32, 31, 19, 34, 29, 3, 30, 13, 37, 14] have become the dominant approach to streaming, long-horizon video generation. Distilled from bidirectional teachers, they generate frames [38, 5, 12, 8] or chunks autoregressively while attending to a rolling KV cache of past frames, producing minute-long videos at interactive rates on a single GPU. As rollouts grow longer, the per-head KV cache increasingly defines the operating point. At Wan-1.3B scale [27], each cached token stores $2 \times 12 \times 128 = 3{,}072$ KV scalars per layer, for keys and values across 12 heads of 128 channels. Caching the full 21-latent-frame context of a 5s clip (1,560 tokens per latent frame, 30 layers) therefore takes 3.02B scalars, about 6.0GB in bf16. This is why streaming systems use fixed-size sliding windows, but a window only bounds the number of cached tokens, not the per-token, per-layer cost of the layout. Reducing that cost is a direct lever for longer windows, larger batches, and faster inference.

Most recent work innovates inside a fixed-size window. CausVid [34] introduced causal distillation with a sliding KV cache, and Self-Forcing [9] closed the train–test gap by training on self-generated rollouts. Subsequent work added attention sinks, token selection, and compressed memories [17, 32, 19, 35, 13, 36], multi-minute training strategies [4, 29, 7], improved distillation [38], and positional reparameterization [31]. These methods change which tokens are kept or how they are positioned, but every kept token still stores the dense per-head layout. A complementary line changes the attention

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

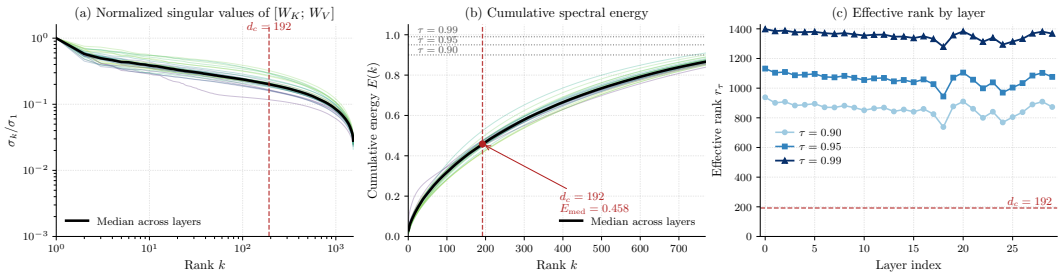


Figure 1: **Pretrained video diffusion attention is not low-rank.** Singular value analysis of $[W_K; W_V] \in \mathbb{R}^{3072\times1536}$ across the 30 transformer blocks of Wan2.1-T2V-1.3B. At $d_c = 192$, the median layer captures only $E_{\text{med}} = 0.458$ of the spectral energy, and the 99%-energy effective rank exceeds 1300 in every layer.

computation: SANA-Video [3] uses block-causal linear attention with a constant-size state, and SCD [2] caches only a 25-layer causal encoder, a 16.7% reduction under the same Wan geometry. VideoMLA is orthogonal to both. It keeps softmax attention and all 30 cached layers but shrinks each token's state from 3,072 to 224 scalars, an $11.4\times$ smaller cache than SCD for the same window.

We reduce the per-token state with Multi-Head Latent Attention (MLA) [15]. DeepSeek-V2 introduced MLA as a pretraining architecture; later methods that convert pretrained MHA models into MLA [11, 21] rely on pretrained W_K, W_V being approximately low-rank. Video diffusion breaks this assumption: in Wan-1.3B the 99%-energy rank of $[W_K; W_V]$ exceeds 1300 in every layer (Fig. 1), and directly truncating it to our latent size collapses quality. Whether a latent KV cache could work for pretrained video models was therefore an open question. We show that it does, and explain why: the MLA bottleneck, not the pretrained spectrum, sets the effective rank, and the compressibility that training exploits is data-conditioned. Rather than a new video-specific attention mechanism, VideoMLA builds on the MLA factorization of DeepSeek-V2 and contributes a rigorously validated transfer to video diffusion together with an explanation of why it works. Our contributions are:

- **MLA for causal video diffusion, validated under full control.** A fully matched dense-MHA control (same backbone, data, training, kernels, window, prompts, seeds, and hardware) attributes both quality and efficiency gains to the cache layout: VideoMLA leads on all six VBench dimensions at 30s and 60s ($+0.078$ and $+0.107$ Overall), is $1.23\times$ faster, and serves $8\times$ more streams.
- **Which ingredients matter.** At matched cache size and training cost, VideoMLA outperforms GQA/MQA, KIVI-style quantization, coupled per-head RoPE, and separate key/value latents. Low-rank sharing, head-shared positions, and NoPE/RoPE decoupling each contribute, and 75% of each head's channels can score content without positional encoding.
- **Why MLA works in video.** The bottleneck, not the pretrained spectrum, sets the effective rank: SVD and random initialization both saturate the budget, converge to the same data-conditioned subspace, and reach the same quality. The design question becomes which latent budget preserves quality.

2 Related Work

Causal Video Generation. Causal video diffusion converts a bidirectional teacher into a streaming student that generates frames or chunks autoregressively with a rolling KV cache. CausVid [34] initiated this line with Distribution Matching Distillation (DMD) [33] based causal distillation, and Self-Forcing [9] reduced train–test mismatch by training on self-generated rollouts. Subsequent work improves long-horizon stability through joint denoising and attention sinks [17], teacher-guided correction [4], causal ODE initialization [38], reward-weighted distillation and EMA sinks [19], deep sink and cache pruning [32], KV recaching for prompt switches [29], and block-relative temporal RoPE [31]. These methods change what is stored in the window or how it is positioned, but retain the dense per-head KV layout.

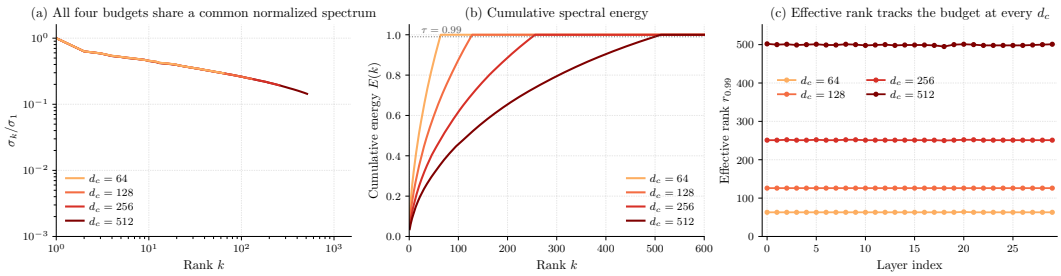


Figure 2: **The composed operator occupies its full rank-d_c budget at every d_c and every layer.** Singular value analysis of the composed operator $M_{\text{learned}} = [W_{\uparrow}^{K} W_{\downarrow}^{KV}; W_{\uparrow}^{V} W_{\downarrow}^{KV}]$ for SVD-initialized VideoMLA students at $d_c \in \{64, 128, 256, 512\}$. **(a)** Median normalized spectra share a common envelope, truncated at d_c. **(b)** Cumulative spectral energy. **(c)** Layer-wise 99%-energy effective rank: $r_{0.99} \approx 0.98\, d_c$ at every budget, uniformly across depth. The composed operator's rank is determined by the architectural bottleneck, not by the spectral structure of the dense source.

Efficient Causal Video Generation. A complementary line restructures attention to reduce memory or compute. SANA-Video [3] replaces softmax with block-causal linear attention and uses a constant-size cumulative state. SCD [2] separates temporal reasoning from frame-wise rendering, caching only the causal encoder. VideoSSM [35] augments sliding-window KV with an SSM-compressed global memory. These approaches reduce temporal or layer-wise memory, but do not compress the per-token, per-head KV state at every cached layer.

Multi-Head Latent Attention and KV Compression. DeepSeek-V2 [15] introduced Multi-Head Latent Attention (MLA) as a pretraining architecture, replacing per-head KV with a shared low-rank latent and a decoupled positional key; DeepSeek-V3 [16] scaled this design. MTLA [6] further compresses along time, while MHA2MLA [11] and TransMLA [21] convert pretrained MHA [26] or GQA [1] LLMs into MLA, relying on approximately low-rank pretrained key/value weights. Other KV reductions share key/value heads (MQA [25], GQA [1]) or quantize the cache (KIVI [18]); we compare against both in Sec. 4.5. All of these works target language models. We study MLA conversion in video diffusion, where the low-rank premise does not hold.

3 Method

We write $x_t \in \mathbb{R}^d$ for the attention input at current chunk t, where a chunk denotes a group of latent frames. Let d be the model dimension, n_h the number of heads, and d_h the per-head dimension, so that $d = n_h d_h$. VideoMLA follows the MLA factorization of DeepSeek-V2 [15]. It introduces a shared KV latent dimension d_c for cached content and splits each head into a NoPE content-scoring subspace and a RoPE positional subspace, $d_h = d_h^{\text{nope}} + d_h^{\text{rope}}$. The NoPE part is reconstructed from the shared latent and is not rotary-position encoded; the RoPE part uses a head-shared decoupled 3D-RoPE key.

3.1 Compressed KV Cache Construction

Each video latent token $x_t \in \mathbb{R}^d$ is compressed into a low-rank latent that summarizes its key and value content:

$$c_t^{KV} \;=\; W_{\downarrow}^{KV} x_t \;\in\; \mathbb{R}^{d_c}, \tag{1}$$

where $W_{\downarrow}^{KV} \in \mathbb{R}^{d_c \times d}$ is the joint KV down-projection (Fig. 3, KV Down). Since $d_c \ll n_h d_h$, c_t^{KV} replaces the dense per-head content keys and values. Positional information is stored separately in the decoupled key k_t^R (Section 3.2), so each cached token stores the pair (c_t^{KV}, k_t^R).

The per-head keys and values needed by attention are obtained from c_t^{KV} through two up-projections,

$$k_{t,h}^{\text{nope}} \;=\; W_{\uparrow,h}^{K}\, c_t^{KV}, \qquad v_{t,h} \;=\; W_{\uparrow,h}^{V}\, c_t^{KV}, \tag{2}$$

where h indexes heads and $W_{\uparrow}^{K}, W_{\uparrow}^{V}$ are the key and value up-projections (Fig. 3). One cached latent serves all heads, and the reconstructed key $k_{t,h}^{\text{nope}}$ carries no rotary information. The per-token

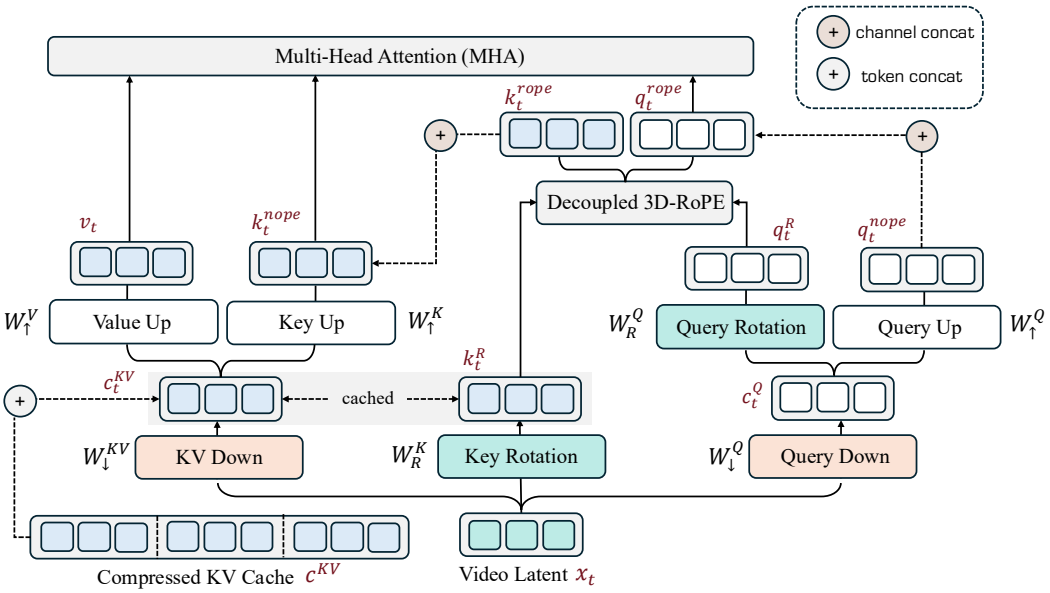


Figure 3: **Overview of VideoMLA.** VideoMLA replaces the dense per-head KV cache of causal Wan2.1-1.3B with a low-rank latent obtained by jointly compressing keys and values through shared down/up projections, with positional information carried by a single decoupled key. **Orange** blocks denote down projections, **green** blocks denote the positional projections whose outputs are rotated by 3D-RoPE, and white blocks denote up projections; latent frames are colored **blue** for the key/value stream and white for the query stream. Each block is annotated with the corresponding weight matrix from Section 3; named latents are shown in **red**.

cached state shrinks from the $2n_h d_h$ scalars of dense MHA to $d_c + d_h^{\text{rope}}$, which is 224 scalars per token per layer in our default setting, a 92.7% reduction.

The query path uses an analogous down/up structure:

$$c_t^Q = W_\downarrow^Q x_t \in \mathbb{R}^{d_q}, \qquad q_{t,h}^{\text{nope}} = W_{\uparrow,h}^Q c_t^Q, \tag{3}$$

where d_q is the query latent dimension. Queries are recomputed at every step, so c_t^Q is never cached. Head-sharing occurs only in the decoupled positional key; NoPE queries, NoPE keys, and values remain head-specific. The latent size d_c is the main capacity knob (Section 4.5, Fig. 6); memory and compute costs relative to full, local, and linear causal attention are given in Appendix Table 7.

3.2 Decoupled 3D-RoPE

The latent c_t^{KV} is kept position-free, so that it can be shared across heads and reused under sliding-window re-indexing. Position is carried by a separate RoPE subspace whose d_h^{rope} channels are partitioned, as in Wan, across temporal, height, and width axes using high-frequency rotary bands. Each token has a single head-shared positional key

$$k_t^R = W_R^K x_t \in \mathbb{R}^{d_h^{\text{rope}}}, \qquad k_t^{\text{rope}} = \text{RoPE}_{3D}(k_t^R), \tag{4}$$

rather than n_h per-head keys. The cache stores the unrotated (c_t^{KV}, k_t^R) and rotation is applied when the active window is assembled, which keeps cached states independent of absolute rollout time. The positional query for head h is

$$q_{t,h}^R = W_{R,h}^Q c_t^Q, \qquad q_{t,h}^{\text{rope}} = \text{RoPE}_{3D}(q_{t,h}^R). \tag{5}$$

3.3 Attention and Inference-Time Absorption

For a query token at position i and a cached token at position j, head h combines the content and positional contributions into a single score

$$\text{score}_{i,j}^{(h)} = \frac{q_{i,h}^{\text{nope}} \cdot k_{j,h}^{\text{nope}} + q_{i,h}^{\text{rope}} \cdot k_{j}^{\text{rope}}}{\sqrt{d_h^{\text{nope}} + d_h^{\text{rope}}}}, \tag{6}$$

followed by a softmax over the active window, a weighted sum of $v_{j,h}$, and the output projection W^O. The score has the shape of a dense attention score, so during training VideoMLA reconstructs per-head keys and values on demand and reuses the block-causal masks, sink tokens, and FlexAttention kernels of the dense baseline unchanged. At inference we use the standard MLA absorption: $W_{\uparrow,h}^{Q\top} W_{\uparrow,h}^{K}$ becomes a fixed $d_q \times d_c$ score matrix and $W_h^O W_{\uparrow,h}^{V}$ is folded into the output projection, so dense per-head keys and values are never materialized (Appendix F).

3.4 Conversion from a Pretrained Dense Model

VideoMLA converts a pretrained dense causal model. We initialize $W_{\downarrow}^{KV}$, $W_{\uparrow}^{K}$, and $W_{\uparrow}^{V}$ from a joint truncated SVD of the stacked content key and value maps, with per-head RMS scales absorbed, and keep the highest-frequency rotary bands for the decoupled RoPE channels. Training then follows the three-stage Causal Forcing pipeline [38]. Stage 1 (Teacher Forcing) opens with a warm-up in which all non-MLA parameters are frozen and only the MLA projections and the two latent RMSNorms are trained (3.22M parameters per layer, 96.6M in total). This isolates any functional recovery to reorganization within the rank budget (Section 5). The full generator is trained for the rest of Stage 1 and in Stages 2 (Consistency Distillation) and 3 (DMD [33]). Neither score network is compressed: the DMD critic is a dense Wan2.1-T2V-1.3B and the real score is the frozen dense Wan2.1-T2V-14B teacher. Random initialization reaches the same quality as SVD (Section 5).

4 Experiments

4.1 Setup

Implementation. We implement VideoMLA on Wan2.1-T2V-1.3B [27] (30 blocks, hidden size 1536, 12 heads of dimension 128) and replace only the self-attention layers. Unless stated otherwise, $d_c = 192$, $d_q = 768$, $d_h^{\text{nope}} = 96$, and $d_h^{\text{rope}} = 32$, with the RoPE channels allocated to temporal, height, and width axes as $(6, 5, 5)$ complex pairs from the highest-frequency bands. Each token caches $d_c + d_h^{\text{rope}} = 224$ scalars per layer, $13.7\times$ fewer than the dense 3,072. Training follows the three-stage pipeline of Section 3.4: Teacher Forcing for 9.5k steps (lr $5{\times}10^{-6}$), Consistency Distillation to a 4-step sampler for 2.5k steps (lr $2{\times}10^{-6}$), and DMD for 750 steps (lr $2{\times}10^{-6}$). Every stage uses a global batch of 128 on $8\times$ B200 GPUs in bf16; micro-batch $\times$ accumulation $\times$ GPUs is $1{\times}16{\times}8$, $2{\times}8{\times}8$, and $8{\times}2{\times}8$ for the three stages. Stages 1–2 use 47,680 videos, 29,471 from OpenVid-1M [22] and 18,209 synthesized with Wan2.1-T2V-14B; Stage 3 uses VidProM prompts [28]. Full details are in Appendix E.

Cache geometry. All long-horizon generation uses a six-latent-frame window, one sink plus five recent frames (9,360 cached tokens), with FIFO eviction outside the sink and window-local RoPE re-indexing [31]. Memory and throughput measurements (Section 1, Fig. 6, Tables 2b and 3) use 5s generations whose full 21-latent-frame context (32,760 tokens) is cached without eviction, the worst case for memory. Per-token ratios are identical in both regimes.

Evaluation protocol. All metrics use the official VBench [10] implementation (v0.1.5). Following [9, 17, 32, 31], 30s and 60s results use 200 MovieGen Video Bench prompts [23] scored on the six VBench-Long dimensions; Semantic, Quality, and Total (30s) and the 5s comparison use the full 946-prompt suite. Seeds are shared across methods (seed = prompt index) with one $832{\times}480$ generation per prompt, and all baselines are regenerated from official checkpoints under identical conditions with their published samplers and cache geometries (Appendix B).

Baselines. We compare with eight streaming methods that keep a dense softmax KV cache (Table 1) and with LongSANA [3], which uses linear attention. Because these systems also differ in training recipe, we attribute gains to the cache layout only through a matched dense-MHA control (Section 4.3).

Table 1: **Long-horizon performance and user preference.** VBench-Long dimensions on 30s and 60s generations (200 prompts, one seed each) and user-study ratings on 60s generations (1–5 Likert; 50 raters, 200 ratings per model; Appendix B). **AQ**: Aesthetic Quality, **BC**: Background Consistency, **DD**: Dynamic Degree, **IQ**: Imaging Quality, **MS**: Motion Smoothness, **SC**: Subject Consistency. User study: **PA**: Prompt Adherence, **TC**: Temporal Consistency, **DC**: Dynamic Consistency (plausible, sustained motion). **Bold**: best; underline: second best.

Model	Results on 30s ↑							Results on 60s ↑							User Study ↑			
	AQ	BC	DD	IQ	MS	SC	Overall	AQ	BC	DD	IQ	MS	SC	Overall	PA	TC	DC	Overall
Self-Forcing [9]	0.541	0.948	0.624	0.577	0.952	0.932	0.762	0.565	0.958	0.393	0.650	0.987	0.974	0.755	2.79	2.79	2.70	2.76
CausVid [34]	0.597	0.921	0.473	0.663	0.935	0.913	0.750	0.497	0.929	0.723	0.574	0.948	0.933	0.767	–	–	–	–
Causal Forcing [38]	0.526	0.945	0.738	0.628	0.968	0.947	0.792	0.503	0.936	0.847	0.608	0.935	0.920	0.792	2.59	2.63	2.81	2.68
Rolling Forcing [17]	0.620	0.953	0.742	0.688	0.982	0.960	0.824	0.580	0.958	0.380	0.670	0.988	0.977	0.759	2.55	2.68	2.60	2.61
Deep Forcing [32]	0.621	0.953	0.713	0.660	0.979	0.961	0.815	0.597	0.957	0.402	0.690	0.987	0.979	0.769	2.60	2.76	2.68	2.68
Reward Forcing [19]	0.644	0.956	0.954	0.683	0.981	0.957	**0.863**	0.585	0.952	0.676	0.673	0.985	0.974	0.808	2.91	2.99	2.83	2.91
LongLive [29]	**0.654**	0.959	0.649	0.678	0.983	0.967	0.816	0.606	0.961	0.433	0.664	0.991	0.982	0.773	2.56	2.70	2.58	2.61
Infinity-RoPE [31]	0.640	0.958	0.847	0.669	0.982	0.966	0.844	**0.607**	0.959	0.647	0.638	0.988	0.979	0.803	2.46	2.44	2.41	2.43
LongSANA [3]	0.573	**0.976**	0.149	0.683	0.974	**0.988**	0.723	0.529	**0.976**	0.103	0.702	0.991	**0.986**	0.714	2.48	2.63	2.56	2.56
VideoMLA (Ours)	0.601	0.942	**0.981**	**0.697**	**0.986**	0.952	0.859	0.569	0.963	**0.958**	**0.715**	**0.993**	0.954	**0.859**	**3.04**	**3.24**	**3.22**	**3.17**

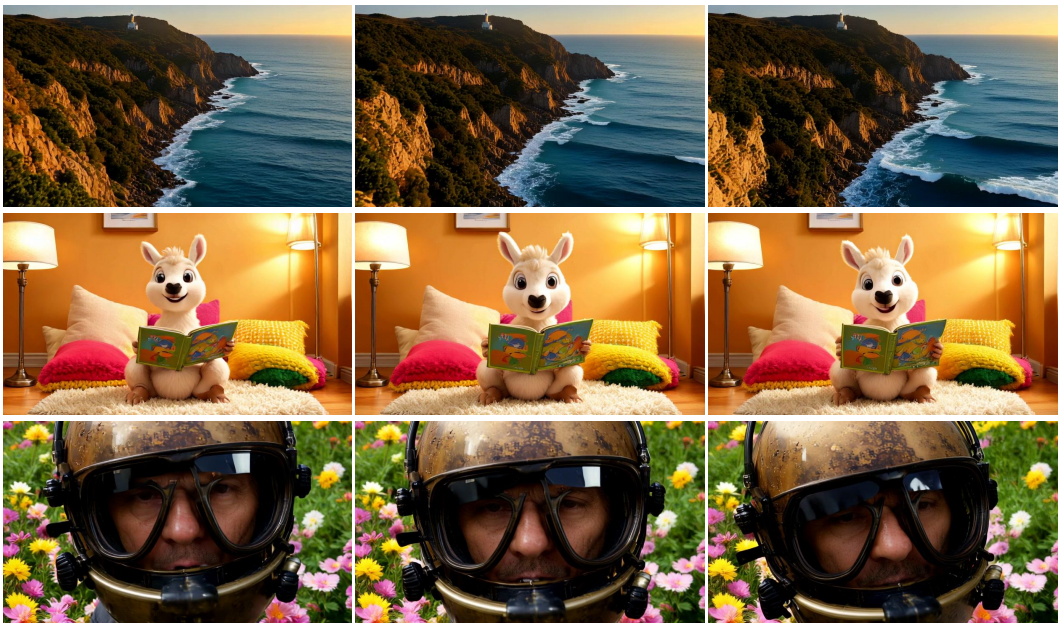


Figure 4: **Qualitative results.** Samples generated by VideoMLA. Frames are shown at uniformly spaced timestamps from each 30s rollout, illustrating that the compressed latent KV cache preserves scene structure, subject identity, and visual fidelity over time.

4.2 Comparison with Streaming Baselines

Table 1 reports long-horizon results. VideoMLA achieves the best Dynamic Degree at both horizons (0.981 at 30s, 0.958 at 60s), together with the best Imaging Quality and Motion Smoothness. It reaches the best 60s Overall (0.859, vs. 0.808 for Reward Forcing), and its Overall score does not drop from 30s to 60s; at 30s it is second to Reward Forcing (0.859 vs. 0.863). Human raters prefer VideoMLA on all three user-study questions; Appendix C shows, using joint VBench metrics, the matched control, and these motion ratings, that its higher Dynamic Degree reflects sustained motion rather than instability. Figures 4 and 5 show that VideoMLA preserves subject identity, scene structure, and visual fidelity over long rollouts while sustaining motion. These comparisons confound the cache layout with each system's training recipe, which motivates the controlled comparison below.



Figure 5: **Qualitative comparison.** Long-rollout samples from VideoMLA and causal video diffusion baselines under the same prompt and seed. Each row shows uniformly spaced frames from one method.

Table 2: **Matched dense-MHA control.** Both models share backbone, data, stages, steps, batch size, kernels, window, prompts, seeds, and hardware; only the cached-token layout differs. (a) Long-horizon VBench (as in Table 1). (b) Efficiency on one B200 at the 21-latent-frame geometry; cache per token per layer (scalars) and per sample (bf16).

(a) Quality

Model	30s ↑							60s ↑						
	AQ	BC	DD	IQ	MS	SC	Overall	AQ	BC	DD	IQ	MS	SC	Overall
Dense MHA (matched)	0.520	0.941	0.702	0.619	0.964	0.940	0.781	0.470	0.905	0.786	0.566	0.902	0.883	0.752
VideoMLA	**0.601**	**0.942**	**0.981**	**0.697**	**0.986**	**0.952**	**0.859**	**0.569**	**0.963**	**0.958**	**0.715**	**0.993**	**0.954**	**0.859**

(b) Efficiency

Model	Cache/token ↓	Cache/sample ↓	Slope (GB/sample) ↓	Max batch ↑	FPS ↑	Latency (s) ↓	Peak mem. (GB) ↓
Dense MHA (matched)	3,072	6.04 GB	6.26	28	19.54	4.14	14.9
VideoMLA	**224**	**0.44 GB**	**0.82**	**224** (8.0×)	**23.96**	**3.38**	**9.4**

4.3 Matched Dense-MHA Control

We trained a dense-MHA counterpart under our exact pipeline (Table 2). The only difference is what each cached token stores: per-head (K, V) versus (c_t^{KV}, k_t^R).

Quality. VideoMLA leads on all six dimensions at both horizons, and the margin widens with horizon ($+0.078$ Overall at 30s, $+0.107$ at 60s). From 30s to 60s the control drifts: Dynamic Degree rises ($0.702 \rightarrow 0.786$) while IQ, MS, BC, and SC all fall, whereas VideoMLA keeps its Overall score and improves IQ, MS, BC, and SC. VideoMLA also leads the control after every training stage (Appendix Table 9). We hypothesize that the gap comes mainly from positional coupling: the control applies RoPE to every scoring channel and inherits rollout-time dependence under re-indexing, whereas 75% of VideoMLA's scoring channels are position-free (Section 4.5).

Efficiency. With the layout as the only difference, VideoMLA is $1.23\times$ faster (23.96 vs. 19.54 fps) and lowers peak memory from 14.9 to 9.4 GB. Because weights and activations dominate memory at batch size 1, the $13.7\times$ cache compression pays out mainly in concurrency: the per-sample slope drops $7.6\times$ and one B200 holds $8.0\times$ more streams (224 vs. 28).

4.4 Efficiency

Among chunk-wise models, VideoMLA has the highest throughput, the lowest latency, and the best CLIP-T (Table 3); Self-Forcing keeps a slightly higher HPSv3, and LongSANA's higher CLIP-F

Table 3: **Efficiency and 5s text-to-video quality on VBench.** Throughput (FPS) and latency (s) are measured with batch size 1 on one B200. CLIP-T, CLIP-F [24], and HPSv3 [20] use the full VBench prompt suite. **Bold**: best; underline: second best. Extended coverage of all baselines is in Appendix Table 14.

Model	#Params	Resolution	Throughput ↑	Latency ↓	CLIP-T ↑	CLIP-F ↑	HPSv3 ↑
Frame-wise autoregressive models							
NOVA [5]	0.6B	768×480	2.26	14.63	0.2764	0.9673	2.95
Pyramid Flow [12]	2B	640×384	1.39	87.32	0.2888	0.9795	8.02
Chunk-wise autoregressive models							
Self-Forcing [9]	1.3B	832×480	19.48	4.19	0.3036	0.9689	**9.86**
LongSANA [3]	2B	832×480	19.35	4.48	0.2978	**0.9887**	7.54
VideoMLA (Ours)	1.3B	832×480	**23.96**	**3.38**	**0.3278**	0.9686	9.74

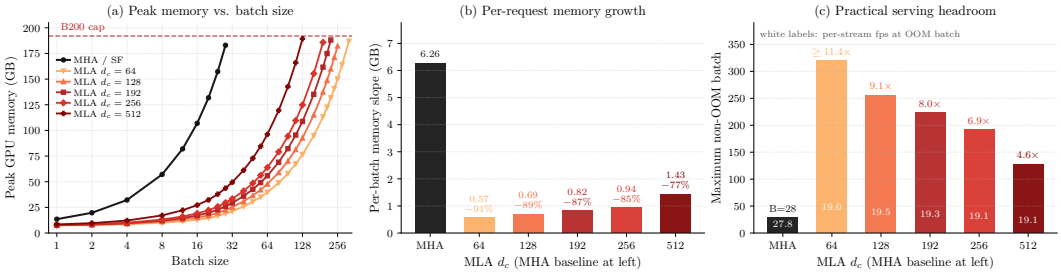


Figure 6: **Serving headroom under a fixed B200 memory budget** (21-latent-frame geometry). (a) Peak memory vs. batch size. (b) Per-sample memory slope. (c) Largest batch that fits; white labels give per-stream throughput (fps) at that batch. Dense MHA runs out of memory at $B = 28$; the default $d_c = 192$ fits $8.0\times$ more streams.

reflects near-static generations (Fig. 7). Across all nine baselines plus Anchor Forcing [30] and MemRoPE [13] (Appendix Table 14), every softmax baseline sits on the same memory line (6.26–6.58 GB per added sample, out of memory at 26–28 streams): token policies decide which tokens are kept, not how much each stores. VideoMLA reaches 0.82 GB per sample and 224 streams while keeping softmax attention, and across $d_c \in \{64, \ldots, 512\}$ the slope drops by 77–91% (Fig. 6). Compression also buys window capacity: with the bytes dense MHA spends on six frames, VideoMLA caches 82, raising Overall from 0.859 to 0.864 at 60s and from 0.836 to 0.858 at 120s (Appendix Table 10).

4.5 What the Operating Point Requires

Table 4 compares alternatives that shrink the same cache under the identical pipeline. **Head sharing alone is not enough.** MQA and GQA-2 fall below dense MHA at 60s (0.718 and 0.741 vs. 0.752), matching the ordering MLA > MHA > GQA > MQA reported for DeepSeek-V2 [15]. **Quantization is complementary.** INT8 keeps dense quality at $6.9\times$ our cache and INT4 degrades at $3.4\times$, while our latent quantizes to INT8 with negligible loss (0.855) for a further $2\times$. **Positional design matters.** A shared latent with per-head coupled RoPE reaches 0.834 at $2.6\times$ our cache, and decoupled RoPE on dense content (0.783) gives no compression; per-head positional keys alone would also exceed the budget ($12 \times 32 = 384$ scalars). The NoPE/RoPE split favors 96/32 (Appendix Table 8b), so 75% of each head's channels score on content alone. **A joint latent beats separate ones.** At identical bytes, separate key and value latents reach 0.821, since keys and values share content structure. An uncompressed query path matches our quality (0.856) but needs $1.31\times$ the Stage-3 training memory. **Latent size.** Quality is flat for d_c from 128 to 512 and drops at $d_c = 64$ (Appendix Table 8a).

5 Why MLA Works in Video Diffusion: Rank Budget vs. Spectral Structure

Pretrained video attention is not low-rank. MHA-to-MLA conversion in language models [11, 21] assumes approximately low-rank key/value maps. In Wan2.1-T2V-1.3B, the median layer retains

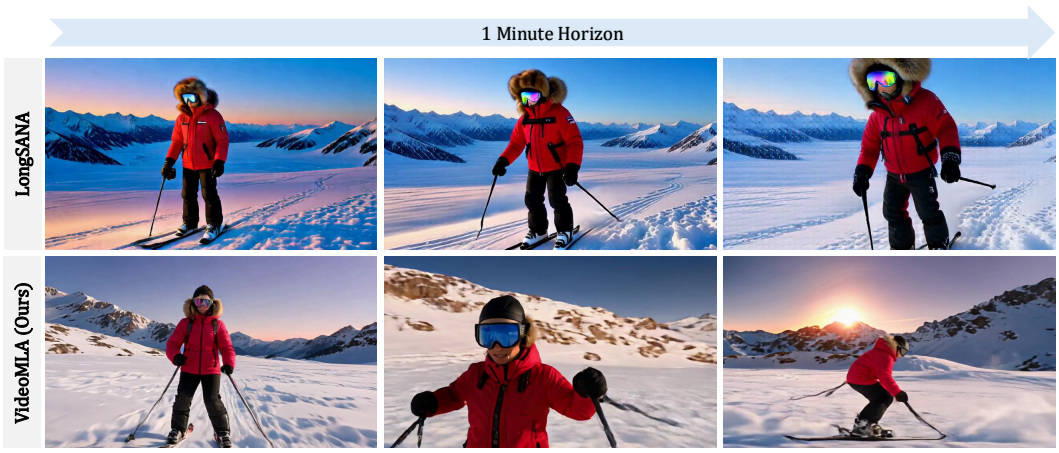


Figure 7: **Long-horizon generation quality.** Frames sampled across a one-minute rollout of the same prompt. **(Bottom)** VideoMLA sustains visual fidelity with diverse, evolving motion, while **(Top)** LongSANA produces near-static content that degrades over time. VideoMLA yields higher visual fidelity and more diverse motion while achieving higher generation throughput and lower latency than LongSANA, and reduces KV cache size by 92.7% relative to the Self-Forcing baseline.

Table 4: **Isolating the ingredients of the operating point.** All variants use the identical three-stage pipeline; KV quantization is post-training (KIVI-style [18]). **Cache**: bytes per token per layer (224 bf16 scalars = 448 B). Semantic/Quality/Total: 30s, full VBench suite; 60s: Overall as in Table 1.

	Configuration	Cache (B) ↓	Semantic ↑	Quality ↑	Total ↑	60s Overall ↑
Reference	Dense MHA (matched control)	6,144	79.2	82.4	80.8	0.752
Head sharing	GQA-2 (2 KV heads)	1,024	79.3	81.5	80.4	0.741
	MQA (1 KV head)	512	78.4	80.9	79.7	0.718
Quantization	Dense + INT8 KV cache	3,072	79.1	82.1	80.6	0.748
	Dense + INT4 KV cache	1,536	77.8	80.2	79.0	0.726
Positional key	Dense content + shared decoupled RoPE	5,440	79.9	82.3	81.1	0.783
	Shared latent + per-head coupled RoPE	1,152	82.2	83.8	83.0	0.834
Latent	Separate K/V latents (d_c^K=d_c^V=96)	448	81.1	82.9	82.0	0.821
	Uncompressed query path	448	82.9	84.7	83.8	0.856
	VideoMLA (joint latent + decoupled RoPE)	448	**83.0**	**84.8**	**83.9**	**0.859**
	VideoMLA + INT8 latent	**224**	82.8	84.6	83.7	0.855

only 45.8% of the spectral energy at $d_c = 192$, and the 99%-energy rank exceeds 1300 in every layer (Fig. 1). Truncating W to rank 192 without training collapses Total from 80.8 (dense) to 46.9 (Table 5a), and to 35.9 at $d_c = 64$, yet the trained model at the same rank exceeds the dense control (83.9). MLA therefore succeeds here for a different reason than in language models.

Compressibility is data-conditioned. Attention sees the weights only through activations, so for a rank-d_c approximation $\hat{W}$ the relevant error is $\mathbb{E}_x\|(W - \hat{W})x\|_2^2 = \|(W - \hat{W})\Sigma_x^{1/2}\|_F^2$ with $\Sigma_x = \mathbb{E}[xx^\top]$; Fig. 1 is the special case $\Sigma_x = I$. Video latent tokens are strongly anisotropic: with Σ_x from 512 clips, 91.4% of the whitened operator's energy lies within rank 192 at the median layer (vs. 45.8%). Median relative errors also contract along the computation (SVD-initialized model, frozen warm-up): 0.74 on the weights ($= \sqrt{1 - 0.458}$), 0.29 on KV activations ($= \sqrt{1 - 0.914}$), 0.17 on logits, and 0.04 on the predicted velocity. The data-optimal static truncation recovers only part of the gap (61.3); training closes the rest because its objective weights directions by their effect on the output rather than by input variance.

The bottleneck, not the spectrum, sets the rank. The composed operator $M = [W_\uparrow^K W_\downarrow^{KV}; W_\uparrow^V W_\downarrow^{KV}]$ has rank at most d_c, and the learned operator uses nearly all of it ($r_{0.99} \approx 0.98\, d_c$ for every d_c and layer; Fig. 2). SVD and random initialization both saturate the budget from the first checkpoint (98.4% and 97.4%), keep a stable spectrum (Appendix Fig. 9), and converge to the same quality (83.9 vs. 83.6) and subspace: overlap with the data-whitened top-192

Table 5: **Static truncation vs. a trained bottleneck, and latent allocations** (30s, full VBench suite). (a) Rank-192 variants differing only in how the subspace is chosen; truncating $W\Sigma_x^{1/2}$ is the data-optimal static choice. (b) Allocations at matched cache bytes.

(a) Static truncation vs. training

Configuration	Rank	Train	Sem.	Qual.	Total
Dense MHA control	full	✓	79.2	82.4	80.8
Trunc. SVD of W	192	×	41.8	52.0	46.9
Trunc. SVD of $W\Sigma_x^{1/2}$	192	×	56.1	66.5	61.3
Ours, SVD init	192	✓	**83.0**	**84.8**	**83.9**
Ours, random init	192	✓	82.8	84.5	83.6

(b) Video-specific allocations

Allocation	Sem.	Qual.	Total	60s
Uniform d_c (default)	**83.0**	**84.8**	**83.9**	**0.859**
Motion-adaptive d_c	82.5	84.2	83.4	0.848
Time/space, asymmetric	82.4	84.1	83.3	0.846
Recency, sink-heavy	82.1	83.9	83.0	0.842
Time/space, symmetric	81.9	83.6	82.8	0.839
Recency, recent-heavy	81.7	83.5	82.6	0.833

subspace rises from 0.64 to 0.83 (SVD) and from chance level 0.12 to 0.79 (random), with mutual overlap 0.81. Since the Stage-1 warm-up freezes all non-MLA parameters, this is reorganization within the rank budget rather than backbone drift. The same holds at $d_c = 64$ (overlap 0.77; 78.3 and 79.8 Total; Appendix Table 11), so small-budget degradation is a capacity limit. The pretrained spectrum sets where training starts; the budget and the data set where it ends.

A uniform latent is the right allocation. Budgets tailored to time, space, recency, or motion would help only if redundancy were anisotropic along those axes. It is not: $r_{0.90}$ of dense KV activations lies between 168 and 189 for every partition we measured (Appendix Table 13), and the uniform latent outperforms all five tailored allocations at matched bytes (Table 5b). The strongly anisotropic axis is content versus position, which is exactly where VideoMLA places its asymmetry.

6 Limitations and Broader Impact

Scope. VideoMLA builds on the MLA factorization of DeepSeek-V2; our focus is transferring it to pretrained causal video diffusion, validating it under controlled conditions, and explaining why it works. Video-tailored latent designs beyond the allocations we tested remain open.

Evaluation and efficiency. Results use one backbone, resolution, training clip length, and distillation pipeline, with one generation per prompt and no confidence intervals. VBench is an imperfect proxy for long-horizon fidelity [4], some 60s rollouts still show artifacts, and our Dynamic Degree evidence relies on VBench, the matched control, and a user study rather than VLM judgment. Prompt switching, scene cuts, larger backbones, and horizons beyond 120s are untested. Savings are in cache state: at batch size 1, weights and activations dominate peak memory, so compression pays out mainly in concurrency; timings use FlexAttention on B200 and may differ elsewhere, and the latent cannot shrink arbitrarily ($d_c = 64$ loses detail).

Broader impact. Cheaper long-form generation broadens access to creative tools but also lowers the cost of deceptive synthetic media; releases should carry the base model's usage restrictions and provenance tools such as watermarking. Training data (OpenVid-1M and Wan2.1-T2V-14B clips) are used under their licenses; user-study details are in Appendix B.

7 Conclusion

We presented VideoMLA, the first study of Multi-Head Latent Attention in autoregressive video diffusion. Replacing the dense per-head KV cache with a shared low-rank content latent and a head-shared decoupled 3D-RoPE key reduces per-token KV memory by 92.7% at every cached layer. Under a fully matched comparison with dense MHA, VideoMLA improves long-horizon quality while running faster and serving $8\times$ more concurrent streams. We also showed why MLA works in video even though pretrained video attention is not low-rank: the bottleneck, not the pretrained spectrum, sets the effective rank, and training finds a data-conditioned subspace from either initialization. We hope these findings establish the per-token KV state as a first-class design axis for streaming video generation, complementary to methods that decide which tokens to keep.

8 Acknowledgment

This research is supported by the National Science Foundation under Grant No. 2543524.

References

[1] Ainslie, J., Lee-Thorp, J., De Jong, M., Zemlyanskiy, Y., Lebrón, F., Sanghai, S.: GQA: Training generalized multi-query transformer models from multi-head checkpoints. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. pp. 4895–4901 (2023)

[2] Bai, X., He, G., Li, Z., Shechtman, E., Huang, X., Wu, Z.: Causality in video diffusers is separable from denoising. arXiv preprint arXiv:2602.10095 (2026)

[3] Chen, J., Zhao, Y., Yu, J., Chu, R., Chen, J., Yang, S., Wang, X., Pan, Y., Zhou, D., Ling, H., et al.: SANA-Video: Efficient video generation with block linear diffusion transformer. arXiv preprint arXiv:2509.24695 (2025)

[4] Cui, J., Wu, J., Li, M., Yang, T., Li, X., Wang, R., Bai, A., Ban, Y., Hsieh, C.J.: Self-Forcing++: Towards minute-scale high-quality video generation. arXiv preprint arXiv:2510.02283 (2025)

[5] Deng, H., Pan, T., Diao, H., Luo, Z., Cui, Y., Lu, H., Shan, S., Qi, Y., Wang, X.: Autoregressive video generation without vector quantization. arXiv preprint arXiv:2412.14169 (2024)

[6] Deng, K., Woodland, P.C.: Multi-head temporal latent attention. arXiv preprint arXiv:2505.13544 (2025)

[7] Gao, J., Chen, Z., Liu, X., Feng, J., Si, C., Fu, Y., Qiao, Y., Liu, Z.: LongVie: Multimodal-guided controllable ultra-long video generation. arXiv preprint arXiv:2508.03694 (2025)

[8] Henschel, R., Khachatryan, L., Poghosyan, H., Hayrapetyan, D., Tadevosyan, V., Wang, Z., Navasardyan, S., Shi, H.: StreamingT2V: Consistent, dynamic, and extendable long video generation from text. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 2568–2577 (2025)

[9] Huang, X., Li, Z., He, G., Zhou, M., Shechtman, E.: Self Forcing: Bridging the train-test gap in autoregressive video diffusion. arXiv preprint arXiv:2506.08009 (2025)

[10] Huang, Z., He, Y., Yu, J., Zhang, F., Si, C., Jiang, Y., Zhang, Y., Wu, T., Jin, Q., Chanpaisit, N., Wang, Y., Chen, X., Wang, L., Lin, D., Qiao, Y., Liu, Z.: VBench: Comprehensive benchmark suite for video generative models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2024)

[11] Ji, T., Guo, B., Wu, Y., Guo, Q., Shen, L., Chen, Z., Qiu, X., Zhang, Q., Gui, T.: Towards economical inference: Enabling DeepSeek's multi-head latent attention in any transformer-based LLMs. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 33313–33328 (2025)

[12] Jin, Y., Sun, Z., Li, N., Xu, K., Jiang, H., Zhuang, N., Huang, Q., Song, Y., Mu, Y., Lin, Z.: Pyramidal flow matching for efficient video generative modeling. arXiv preprint arXiv:2410.05954 (2024)

[13] Kim, Y., Hu, Q., Kuo, C.C.J., Beerel, P.A.: MemRoPE: Training-free infinite video generation via evolving memory tokens. arXiv preprint arXiv:2603.12513 (2026)

[14] Li, H., Liu, S., Lin, Z., Chandraker, M.: Rolling Sink: Bridging limited-horizon training and open-ended testing in autoregressive video diffusion. arXiv preprint arXiv:2602.07775 (2026)

[15] Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Deng, C., Ruan, C., Dai, D., Guo, D., et al.: DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. arXiv preprint arXiv:2405.04434 (2024)

[16] Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al.: DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437 (2024)

[17] Liu, K., Hu, W., Xu, J., Shan, Y., Lu, S.: Rolling Forcing: Autoregressive long video diffusion in real time. arXiv preprint arXiv:2509.25161 (2025)

[18] Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., Hu, X.: KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235 (2024)

[19] Lu, Y., Zeng, Y., Li, H., Ouyang, H., Wang, Q., Cheng, K.L., Zhu, J., Cao, H., Zhang, Z., Zhu, X., et al.: Reward Forcing: Efficient streaming video generation with rewarded distribution matching distillation. arXiv preprint arXiv:2512.04678 (2025)

[20] Ma, Y., Shui, Y., Wu, X., Sun, K., Li, H.: HPSv3: Towards wide-spectrum human preference score. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2025)

[21] Meng, F., Tang, P., Tang, X., Yao, Z., Sun, X., Zhang, M.: TransMLA: Multi-head latent attention is all you need. arXiv preprint arXiv:2502.07864 (2025)

[22] Nan, K., Xie, R., Zhou, P., Fan, T., Yang, Z., Chen, Z., Li, X., Yang, J., Tai, Y.: OpenVid-1M: A large-scale high-quality dataset for text-to-video generation. arXiv preprint arXiv:2407.02371 (2024)

[23] Polyak, A., Zohar, A., Brown, A., Tjandra, A., Sinha, A., Lee, A., Vyas, A., Shi, B., Ma, C.Y., Chuang, C.Y., et al.: Movie Gen: A cast of media foundation models. arXiv preprint arXiv:2410.13720 (2024)

[24] Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: Proceedings of the 38th International Conference on Machine Learning. pp. 8748–8763 (2021)

[25] Shazeer, N.: Fast transformer decoding: One write-head is all you need. arXiv preprint arXiv:1911.02150 (2019)

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

[27] Wan, T., Wang, A., Ai, B., Wen, B., Mao, C., Xie, C.W., Chen, D., Yu, F., Zhao, H., Yang, J., et al.: Wan: Open and advanced large-scale video generative models. arXiv preprint arXiv:2503.20314 (2025)

[28] Wang, W., Yang, Y.: VidProM: A million-scale real prompt-gallery dataset for text-to-video diffusion models. In: Advances in Neural Information Processing Systems (Datasets and Benchmarks Track) (2024)

[29] Yang, S., Huang, W., Chu, R., Xiao, Y., Zhao, Y., Wang, X., Li, M., Xie, E., Chen, Y., Lu, Y., et al.: LongLive: Real-time interactive long video generation. arXiv preprint arXiv:2509.22622 (2025)

[30] Yang, Y., Zhang, T., Huang, W., Chen, J., Wu, B., He, X., Cai, D., Li, B., Jiang, P.T.: Anchor Forcing: Anchor memory and tri-region RoPE for interactive streaming video diffusion. arXiv preprint arXiv:2603.13405 (2026)

[31] Yesiltepe, H., Meral, T.H.S., Akan, A.K., Oktay, K., Yanardag, P.: Infinity-RoPE: Action-controllable infinite video generation emerges from autoregressive self-rollout. arXiv preprint arXiv:2511.20649 (2025)

[32] Yi, J., Jang, W., Cho, P.H., Nam, J., Yoon, H., Kim, S.: Deep Forcing: Training-free long video generation with deep sink and participative compression. arXiv preprint arXiv:2512.05081 (2025)

[33] Yin, T., Gharbi, M., Park, T., Zhang, R., Shechtman, E., Durand, F., Freeman, B.: Improved distribution matching distillation for fast image synthesis. Advances in neural information processing systems **37**, 47455–47487 (2024)

[34] Yin, T., Zhang, Q., Zhang, R., Freeman, W.T., Durand, F., Shechtman, E., Huang, X.: From slow bidirectional to fast autoregressive video diffusion models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 22963–22974 (2025)

[35] Yu, Y., Wu, X., Hu, X., Hu, T., Sun, Y., Lyu, X., Wang, B., Ma, L., Ma, Y., Wang, Z., et al.: VideoSSM: Autoregressive long video generation with hybrid state-space memory. arXiv preprint arXiv:2512.04519 (2025)

[36] Zhang, L., Agrawala, M.: Packing input frame context in next-frame prediction models for video generation. arXiv preprint arXiv:2504.12626 (2025)

[37] Zhao, Z., Lu, Y., Liu, Z., Song, J., Deng, J., Patras, I.: Relax Forcing: Relaxed KV-memory for consistent long video generation. arXiv preprint arXiv:2603.21366 (2026)

[38] Zhu, H., Zhao, M., He, G., Su, H., Li, C., Zhu, J.: Causal Forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation. arXiv preprint arXiv:2602.02214 (2026)

Table of Contents

A Videos and Website **13**

B Evaluation Protocol and User Study **13**

C Is the Higher Dynamic Degree Genuine Motion? **14**

D Background **15**

D.1 Wan2.1-T2V-1.3B Backbone 15

E Implementation Details **16**

E.1 Backbone and Tokenization 16

E.2 VideoMLA Block 16

E.3 NoPE/RoPE Split and 3D RoPE 17

E.4 Chunk-Causal Sliding-Window Attention 17

E.5 Long-Horizon RoPE Re-indexing 18

E.6 Training Pipeline 18

F Inference-Time Reparameterization **19**

G Attention Cost Comparison **20**

H Additional Ablations **20**

H.1 Latent Dimension and NoPE/RoPE Split 20

H.2 Per-Stage Curriculum 21

H.3 Cache Eviction and Window Geometry 21

I Additional Rank-Budget Analysis **22**

J Extended Efficiency Comparison **23**

A Videos and Website

Video results covering qualitative examples, ablation studies, comparisons, and failure cases are provided in the supplementary material (`website.html`).

B Evaluation Protocol and User Study

VBench protocol. All automatic metrics use the official VBench implementation (v0.1.5) without modification. The 30s and 60s results (Tables 1, 2, the 60s column of Table 4, and Tables 9–10) use 200 MovieGen Video Bench prompts extended with the Self-Forcing prompt pipeline and are scored with the six VBench-Long custom-prompt dimensions, following Self-Forcing, Rolling Forcing, Deep Forcing, and Infinity-RoPE. Semantic, Quality, and Total scores (Tables 4, 5, and 8) are computed on 30s generations with the full 946-prompt suite, since these aggregates require all sixteen dimensions. The 5s comparison (Table 3) also uses the full suite. Seeds are deterministic (seed $=$ prompt index), so every method receives identical noise per prompt, and we generate one 832×480 video per prompt. All baselines are regenerated from their official public checkpoints on the same

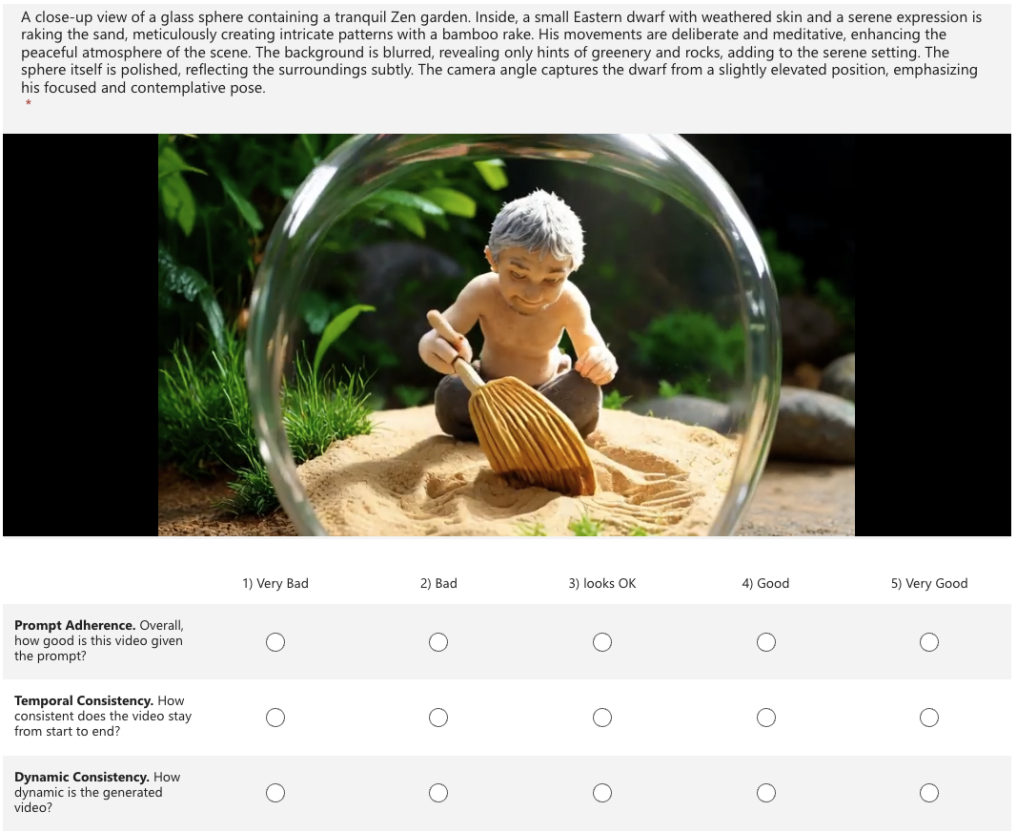


Figure 8: **User study interface** for one-minute video generation.

hardware, prompts, seeds, resolution, and horizon protocol, each with its own published sampler settings and cache geometry (e.g., Reward Forcing uses 3 sink and 6 recent frames, Deep Forcing 14 and 7). Our six-frame window (one sink plus five recent frames) is the most compact among the evaluated methods.

User study. We evaluate perceptual quality of one-minute generations from nine models (all methods in Table 1 except CausVid) on 20 prompts, for 180 one-minute videos in total. We recruited 50 participants on Prolific; each participant was paid $1. Each participant rates the videos of all nine models for four prompts, which gives 10 raters per prompt and 200 ratings per model and question. Videos are anonymized, and their order is randomized per participant. For each video, participants answer three questions on a five-point Likert scale, from 1 (very bad) to 5 (very good): *Prompt Adherence*, how well the video follows the prompt; *Temporal Consistency*, whether the video remains coherent from start to end; and *Dynamic Consistency*, whether the video contains plausible and sustained motion. The last two questions target the distinction between genuine motion and instability discussed in Section C. The interface is shown in Fig. 8. We report mean ratings and do not report inter-rater agreement or confidence intervals.

C Is the Higher Dynamic Degree Genuine Motion?

A high Dynamic Degree can reflect richer content or uncontrolled motion from rollout instability, and VBench metrics can be poorly aligned with long-horizon degradation [4]. We use three complementary checks.

(i) Joint metric fingerprints. Motion collapse and drift leave distinct signatures in Table 1 of the main paper. Under collapse, DD falls to 0.10–0.43 at 60s (Self-Forcing, Rolling Forcing, Deep Forcing, LongLive, LongSANA) while MS, SC, and BC stay near ceiling, because static content is trivially smooth and consistent. Under drift, DD rises while fidelity and consistency fall together:

from 30s to 60s, CausVid's DD grows from 0.473 to 0.723 and Causal Forcing's from 0.738 to 0.847, and these two methods have the lowest 60s IQ, MS, BC, and SC in the table. VideoMLA shows neither pattern: it has the highest DD together with the best MS and IQ, and its IQ and MS improve from 30s to 60s, the opposite of an instability trajectory.

(ii) A controlled failure case. On the same prompts and seeds, the matched dense control reproduces the drift signature (Section 4.3), while VideoMLA does not.

(iii) Human judgment. The user study asks raters directly whether motion is plausible and sustained (Dynamic Consistency) and whether the video stays coherent from start to end (Temporal Consistency), on anonymized, order-randomized one-minute videos. VideoMLA receives the highest rating on both (3.22 and 3.24, vs. at most 2.83 and 2.99 for any baseline). Motion produced by instability would be penalized on both questions. These checks do not fully settle the question: the study uses coarse Likert ratings, it does not include the dense control, and some 60s rollouts still show visible artifacts (Section 6).

D Background

D.1 Wan2.1-T2V-1.3B Backbone

Our experiments use Wan2.1-T2V-1.3B as the base video diffusion backbone. Wan2.1-T2V-1.3B is a latent video diffusion transformer operating over spatiotemporal latent tokens rather than RGB pixels. The video is encoded by a 3D VAE that compresses the temporal dimension by $4\times$ and each spatial dimension by $8\times$, so an input video $V \in \mathbb{R}^{F\times H\times W\times 3}$ is mapped to a latent tensor with temporal length $1 + \lceil (F-1)/4 \rceil$ and spatial resolution $H/8 \times W/8$. The denoising model follows the rectified-flow formulation, where a clean latent x_0 and Gaussian noise ϵ are linearly interpolated as

$$x_t = (1-t)x_0 + t\epsilon, \qquad t \in [0,1],$$

and the reverse process is parameterized by a neural velocity field and solved with Euler integration at inference time.

Wan2.1-T2V-1.3B uses a diffusion transformer with multi-head self-attention over video latent tokens. In our implementation, the backbone contains 30 transformer blocks, hidden dimension $d = 1536$, $n_h = 12$ attention heads, and per-head dimension $d_h = 128$. In the dense baseline, each cached token stores both keys and values for all heads, giving a per-token, per-layer KV cache size of

$$2n_h d_h = 2 \times 12 \times 128 = 3072$$

scalars. Caching the full 21-latent-frame context of a 5s clip, with 1,560 tokens per latent frame and 30 cached transformer layers, corresponds to 3.02B cached scalars, or approximately 6.0GB in bf16. This is the geometry used for all memory measurements; long-horizon generation instead uses a six-latent-frame window (Appendix E.4). This dense per-head KV layout is the main memory target of VideoMLA.

The backbone uses 3D rotary position embeddings (3D-RoPE) to encode temporal and spatial token coordinates before self-attention. For latent features $x \in \mathbb{R}^{B\times S\times C}$ with $S = FHW$, the channel dimension is partitioned across temporal, height, and width axes, and RoPE is applied separately to the corresponding coordinate subspaces before concatenation. In Wan, each RoPE dimension has a fixed maximum sequence length of 1024; although RoPE remains mathematically defined beyond this range, generation outside the positional regime observed during training can degrade attention quality.

For autoregressive long-video generation, Wan2.1-T2V-1.3B is commonly used after causal distillation or self-rollout training. The model generates latent frame chunks sequentially and conditions each chunk on a rolling KV cache of previous chunks. This cache enables efficient streaming generation because previous key and value states are reused rather than recomputed. However, in the dense Wan attention layout, the cache still stores full per-head keys and values for every retained token and every cached layer. VideoMLA keeps the Wan backbone and causal rollout setting intact, but replaces this dense per-token KV state with a shared latent content cache and a decoupled head-shared 3D-RoPE key.

Table 6: **Training and model hyperparameters.** Unless otherwise stated, all experiments use the default VideoMLA setting. The matched dense-MHA control uses identical values except for the attention layout.

Hyperparameter	Value
Backbone	Wan2.1-T2V-1.3B
Transformer blocks / hidden dimension d	30 / 1536
Heads n_h / per-head dimension d_h	12 / 128
KV latent dimension d_c / query latent dimension d_q	192 / 768
NoPE / RoPE channels per head	96 / 32
3D-RoPE complex pairs (t, h, w)	$(6, 5, 5)$
Per-token cache size	224 scalars (92.7% reduction, 13.7×)
Training precision / hardware	bf16 / 8× B200, FSDP
Optimizer	AdamW, $\beta_1 = 0$, $\beta_2 = 0.999$
Global batch size (every stage)	128
Micro-batch × accum. × GPUs (Stages 1/2/3)	1×16×8 / 2×8×8 / 8×2×8
Optimization steps (Stages 1/2/3)	9,500 / 2,500 / 750
Generator LR (Stages 1/2/3)	5×10^{-6} / 2×10^{-6} / 2×10^{-6}
Critic LR (Stages 2–3)	4×10^{-7}
Stage-1 warm-up trainable parameters	MLA projections + latent RMSNorms (96.6M)
DMD critic / real score	dense Wan2.1-T2V-1.3B / frozen Wan2.1-T2V-14B
Training data (Stages 1–2 / Stage 3)	47,680 videos / VidProM prompts
Inference steps	4 ($[1000, 750, 500, 250]$)
Inference cache window	1 sink + 5 recent latent frames

E Implementation Details

E.1 Backbone and Tokenization

VideoMLA is implemented on top of Wan2.1-T2V-1.3B. The backbone contains $L = 30$ transformer blocks, hidden dimension $d = 1536$, $n_h = 12$ attention heads, and per-head dimension $d_h = 128$, so that $d = n_h d_h$. The feed-forward hidden dimension is 8960. We keep the Wan text-conditioning branch and all non-attention modules unchanged, and replace only the self-attention layers with the VideoMLA block described in Section 3.

We train on 5-second clips at 480×832 resolution and 16 fps. The Wan VAE encodes each clip into a latent tensor with 16 channels, 21 latent frames, and spatial size 60×104. A 3D patch embedding with patch size $(1, 2, 2)$ maps each latent frame into $30 \times 52 = 1560$ visual tokens. Thus, each 5-second clip contains $21 \times 1560 = 32760$ self-attention tokens. Autoregressive generation is performed in chunks of 3 latent frames.

E.2 VideoMLA Block

Each self-attention layer follows the notation of Section 3. Given an attention input $x_t \in \mathbb{R}^d$, VideoMLA first forms a shared content cache latent

$$c_t^{KV} = W_{\downarrow}^{KV} x_t \in \mathbb{R}^{d_c},$$

and a query latent

$$c_t^{Q} = W_{\downarrow}^{Q} x_t \in \mathbb{R}^{d_q}.$$

In the default model, we use

$$d_c = 192, \qquad d_q = 768, \qquad d_h^{\text{nope}} = 96, \qquad d_h^{\text{rope}} = 32,$$

with $d_h^{\text{nope}} + d_h^{\text{rope}} = d_h = 128$. The down-projection shapes are

$$W_{\downarrow}^{KV} \in \mathbb{R}^{192\times1536}, \qquad W_{\downarrow}^{Q} \in \mathbb{R}^{768\times1536}.$$

Both c_t^{KV} and c_t^{Q} are normalized by RMSNorm before the corresponding up-projections.

The content key and value for head h are reconstructed from the shared cache latent:

$$k_{t,h}^{\text{nope}} = W_{\uparrow,h}^{K} c_t^{KV}, \qquad v_{t,h} = W_{\uparrow,h}^{V} c_t^{KV}.$$

Aggregating all heads, the projection shapes are

$$W_{\uparrow}^{K} \in \mathbb{R}^{(n_h d_h^{\text{nope}}) \times d_c} = \mathbb{R}^{1152 \times 192},$$

$$W_{\uparrow}^{V} \in \mathbb{R}^{(n_h d_h) \times d_c} = \mathbb{R}^{1536 \times 192}.$$

The NoPE query is reconstructed analogously:

$$q_{t,h}^{\text{nope}} = W_{\uparrow,h}^{Q} c_t^{Q}, \qquad W_{\uparrow}^{Q} \in \mathbb{R}^{(n_h d_h^{\text{nope}}) \times d_q} = \mathbb{R}^{1152 \times 768}.$$

The RoPE branch is decoupled from the content cache. For each token, VideoMLA computes a single head-shared positional key

$$k_t^R = W_R^K x_t \in \mathbb{R}^{d_h^{\text{rope}}}, \qquad W_R^K \in \mathbb{R}^{32 \times 1536},$$

and per-head positional queries

$$q_{t,h}^R = W_{R,h}^{Q} c_t^{Q}, \qquad W_R^{Q} \in \mathbb{R}^{(n_h d_h^{\text{rope}}) \times d_q} = \mathbb{R}^{384 \times 768}.$$

The output projection is the original attention output projection

$$W_O \in \mathbb{R}^{d \times (n_h d_h)} = \mathbb{R}^{1536 \times 1536}.$$

Therefore, each cached token stores only

$$(c_t^{KV}, k_t^R) \in \mathbb{R}^{d_c + d_h^{\text{rope}}},$$

rather than dense per-head keys and values. With the default setting, this is $192 + 32 = 224$ scalars per token per layer, compared with $2 n_h d_h = 2 \cdot 12 \cdot 128 = 3072$ scalars for dense MHA, corresponding to a $13.7\times$ cache reduction.

E.3 NoPE/RoPE Split and 3D RoPE

Each head is split as

$$d_h = d_h^{\text{nope}} + d_h^{\text{rope}},$$

where the NoPE subspace is used for content matching and the RoPE subspace is used for position-aware matching. The per-head query and key are

$$q_{t,h} = [q_{t,h}^{\text{nope}}; q_{t,h}^{\text{rope}}], \qquad k_{t,h} = [k_{t,h}^{\text{nope}}; k_t^{\text{rope}}],$$

where

$$q_{t,h}^{\text{rope}} = \text{RoPE}_{3D}(q_{t,h}^R), \qquad k_t^{\text{rope}} = \text{RoPE}_{3D}(k_t^R).$$

The positional key is shared across heads, while the NoPE keys, NoPE queries, and values remain head-specific after up-projection.

For the default split $d_h^{\text{rope}} = 32$, the RoPE subspace contains $d_h^{\text{rope}}/2 = 16$ complex frequency pairs. Following the Wan 3D-RoPE factorization, these pairs are allocated across temporal, height, and width axes as

$$(6, 5, 5),$$

using the highest-frequency bands from the corresponding axis groups.

E.4 Chunk-Causal Sliding-Window Attention

VideoMLA preserves the chunk-causal attention pattern used by the autoregressive Wan backbone. Tokens within the same 3-latent-frame chunk can attend to one another, while tokens in a later chunk cannot be attended to by earlier chunks. For long-horizon generation, attention is restricted to a fixed six-latent-frame window: one sink latent frame plus the five most recent latent frames, following the 1 sink + 5 recent geometry of Infinity-RoPE. Since each latent frame contains 1560 tokens, the sink

occupies 1560 cached token slots and the recent frames occupy 5×1560, for 9,360 cached tokens in total.

During training, the same chunk-causal and sliding-window structure is enforced with block-sparse attention masks. During inference, the cache stores

$$c^{KV} \in \mathbb{R}^{B \times T_{\text{cache}} \times d_c}, \qquad k^R \in \mathbb{R}^{B \times T_{\text{cache}} \times d_h^{\text{rope}}},$$

where T_{cache} is the number of cached tokens in the sink-plus-window context. When the cache is full, tokens outside the sink are evicted in FIFO order. The attention computation reads the active cache, reconstructs the content keys and values through $W_\uparrow^K$ and $W_\uparrow^V$ (or, at inference, uses the absorbed form of Appendix F), applies 3D-RoPE to the active positional keys, and evaluates the per-head attention scores of Eq. 6.

E.5 Long-Horizon RoPE Re-indexing

For rollouts beyond the 21 latent frames seen during 5-second training, cached positional keys are stored before RoPE is applied. When an attention window is assembled, the active cached keys are assigned local temporal coordinates inside the current sink-plus-window context and are then rotated by RoPE_{3D} following [31]. The current query chunk is rotated in the same local coordinate system. Thus, both queries and cached keys use a bounded, window-relative positional frame even after cache eviction.

This re-indexing keeps the RoPE phase within the short-horizon regime observed by the backbone during training, while allowing the generated video to extend beyond the original 21-latent-frame clip length. All reported long-video rollouts use one sink latent frame plus five recent latent frames (a six-frame window) and the 4-step student sampler.

E.6 Training Pipeline

We train the same VideoMLA architecture in three stages on 8 NVIDIA B200 GPUs, using FSDP full sharding, bf16 mixed precision, AdamW with $\beta_1 = 0$ and $\beta_2 = 0.999$, and the rectified-flow denoising objective. Every stage uses the same global batch of 128 samples per optimizer step; the split into per-GPU micro-batch and gradient accumulation follows each stage's activation footprint. Stages 1–2 use 47,680 videos (29,471 from OpenVid-1M and 18,209 synthesized with Wan2.1-T2V-14B); Stage 3 uses VidProM text prompts only. The matched dense-MHA control of Section 4.3 uses exactly the same stages, step counts, data, batch composition, and hyperparameters.

Stage 1: Teacher Forcing. We initialize the MLA projections from an SVD-style decomposition of the pretrained Wan dense attention matrices at the target configuration

$$(d_c, d_q, d_h^{\text{nope}}, d_h^{\text{rope}}) = (192, 768, 96, 32).$$

The model is then trained as a chunk-causal flow-matching student with clean previous-block context from the teacher-encoded latents. Stage 1 opens with a warm-up in which all non-MLA parameters are frozen and only the MLA projections and the two latent RMSNorms are trained (3,220,416 parameters per layer from the shapes in Appendix E.2, 96.6M in total); the full generator is trained afterwards. We use learning rate 5×10^{-6}, a per-GPU micro-batch of 1 with 16 gradient-accumulation steps on 8 GPUs (global batch 128), gradient checkpointing, 1000 training timesteps, and timestep shift 5.0, for 9,500 optimization steps.

Stage 2: Consistency Distillation. Starting from the Stage-1 checkpoint, we distill the model to a 4-step sampling schedule

$$[1000, 750, 500, 250].$$

We use timestep shift 5.0 and classifier-free guidance scale 3.0. The generator learning rate is 2×10^{-6} and the critic learning rate is 4×10^{-7}. We use a per-GPU micro-batch of 2 with 8 gradient-accumulation steps on 8 GPUs (global batch 128) and gradient checkpointing, for 2,500 optimization steps.

Stage 3: Distribution Matching Distillation. Finally, we initialize from the Stage-2 checkpoint at iteration 2500 and fine-tune with distribution matching distillation on the same 4-step schedule. The real score is provided by the frozen dense Wan2.1-T2V-14B teacher, and the fake score (critic) is a dense Wan2.1-T2V-1.3B learned online; neither score network uses the latent cache. We use five critic updates per generator update, EMA weight 0.99 starting from step 1, generator learning rate 2×10^{-6}, critic learning rate 4×10^{-7}, guidance scale 3.0, and timestep shift 5.0. The global batch size is 128, obtained with a per-GPU batch size of 8 on 8 GPUs and gradient accumulation of 2, and we train for 750 optimization steps.

The Stage-3 checkpoint is used for all reported VideoMLA results, including the long-horizon evaluations.

F Inference-Time Reparameterization

The training-time formulation in Section 3 is written to make the connection to standard multi-head attention explicit: each cached token stores (c_j^{KV}, k_j^R), and the per-head NoPE keys and values are reconstructed through Eq. 2 before applying the usual attention computation. This is convenient during training because it allows VideoMLA to reuse the same block-causal masks, sink-token logic, and attention kernels as the dense baseline. At inference, however, explicitly reconstructing $k_{j,h}^{\mathrm{nope}}$ and $v_{j,h}$ would partially undo the benefit of latent caching by materializing dense per-head tensors after every cache read. We therefore use an equivalent reparameterization that keeps the cache and the attention computation in latent form.

For the score computation, the only contribution of the reconstructed NoPE key is through its inner product with the reconstructed NoPE query. Substituting Eqs. 2 and 3 into the content term of Eq. 6 gives

$$\begin{aligned} q_{i,h}^{\mathrm{nope}} \cdot k_{j,h}^{\mathrm{nope}} &= \left(W_{\uparrow,h}^{Q} c_i^{Q}\right)^{\top} \left(W_{\uparrow,h}^{K} c_j^{KV}\right) \\ &= \left(c_i^{Q}\right)^{\top} \left(W_{\uparrow,h}^{Q}\right)^{\top} W_{\uparrow,h}^{K} c_j^{KV} \\ &= \left(c_i^{Q}\right)^{\top} A_h c_j^{KV}, \end{aligned} \tag{7}$$

where

$$A_h = \left(W_{\uparrow,h}^{Q}\right)^{\top} W_{\uparrow,h}^{K} \in \mathbb{R}^{d_q \times d_c}. \tag{8}$$

The matrix A_h depends only on learned parameters and is independent of the current sequence, cache contents, diffusion timestep, and rollout position. It can therefore be precomputed once when the model is loaded. During inference, the NoPE content score for head h is computed directly from the query latent c_i^Q and the cached content latent c_j^{KV}, without forming either $q_{i,h}^{\mathrm{nope}}$ or $k_{j,h}^{\mathrm{nope}}$ as explicit per-head vectors.

The value path admits an analogous absorption. Let W_h^O denote the slice of the output projection applied to the output of head h. Using Eq. 2,

$$\begin{aligned} W_h^O v_{j,h} &= W_h^O W_{\uparrow,h}^V c_j^{KV} \\ &= B_h c_j^{KV}, \end{aligned} \tag{9}$$

where

$$B_h = W_h^O W_{\uparrow,h}^V. \tag{10}$$

Thus, the value up-projection can also be folded into the output mixer. In practice, after the attention weights for head h are computed, the weighted sum can be accumulated over the cached latents c_j^{KV} and then projected by B_h, rather than first reconstructing all dense values $v_{j,h}$ and then applying the output projection.

The RoPE branch is kept separate from this absorption. The cache stores the unrotated, head-shared positional key k_j^R from Eq. 4. When the active attention window is assembled, k_j^R is rotated by $\mathrm{RoPE}_{\mathrm{3D}}(\cdot)$ using the current window indexing, and the positional score term in Eq. 6 is computed as

$$q_{i,h}^{\mathrm{rope}} \cdot k_j^{\mathrm{rope}}. \tag{11}$$

This separation is important because RoPE is position-dependent and cannot be folded into a fixed parameter matrix in the same way as the NoPE content projections. Storing k_j^R unrotated also preserves the ability to re-index cached tokens within a sliding window, as described in Section 3.2.

After reparameterization, the inference-time cache is never expanded into dense per-head keys and values. Each cached token contributes only a content latent $c_j^{KV} \in \mathbb{R}^{d_c}$ and a head-shared positional key $k_j^R \in \mathbb{R}^{d_h^{\text{rope}}}$. Therefore, the per-token cached state remains

$$d_c + d_h^{\text{rope}}, \tag{12}$$

instead of the dense baseline cost

$$2n_h d_h. \tag{13}$$

For an attention window of size W, cache memory traffic is reduced from

$$\mathcal{O}(W\, 2n_h d_h) \tag{14}$$

to

$$\mathcal{O}(W\, (d_c + d_h^{\text{rope}}))\,. \tag{15}$$

With the default configuration $d_c = 192$ and $d_h^{\text{rope}} = 32$, this corresponds to 224 cached scalars per token per layer, compared with $2n_h d_h = 3072$ scalars for dense MHA. The reparameterization therefore preserves the mathematical attention computation of the training-time formulation while ensuring that the inference-time implementation realizes the intended latent-cache memory and bandwidth savings.

G Attention Cost Comparison

Table 7: **Memory and compute costs across four attention variants.** For a sequence of length N with hidden dimension D, n heads, per-head dimension $d_h = D/n$, local window W, latent KV dimension d_c ($d_c \ll D$), and shared decoupled-RoPE dimension d_h^{rope}. Memory is per layer; MLA compute is for the absorbed inference form.

Metric	Causal Full	Causal Local	Causal Linear	MLA Local
Memory	$2ND$	$2WD$	$D\,d_h$	$W(d_c + d_h^{\text{rope}})$
Comp. (N-th token)	ND	WD	$D\,d_h$	$nW(d_c + d_h^{\text{rope}})$
Comp. (N tokens)	$\frac{1}{2}N^2D$	NWD	$N\,D\,d_h$	$nNW(d_c + d_h^{\text{rope}})$

H Additional Ablations

H.1 Latent Dimension and NoPE/RoPE Split

Table 8 studies two architectural choices: the latent KV dimension d_c and the NoPE/RoPE channel split. The latent dimension controls the main quality–efficiency trade-off. At $d_c = 64$, VideoMLA gives the largest cache compression and memory headroom, but the budget is too restrictive: both semantic and quality scores drop, consistent with the loss of fine-grained visual details under overly aggressive compression. Increasing to $d_c = 128$ largely recovers quality while retaining a large compression ratio. Further increasing to $d_c = 256$ or 512 gives only marginal gains, but substantially reduces the memory advantage. The useful operating regime is therefore the smallest budget that preserves task-relevant video features, not the largest possible latent dimension.

The NoPE/RoPE split also has a clear effect. With only 16 RoPE channels, positional capacity is too limited, leading to weak temporal and spatial anchoring. Conversely, the RoPE-heavy $32/96$ split leaves too little capacity for cached content and hurts semantic fidelity. The balanced $64/64$ setting improves over these extremes but remains below the content-heavy default. The best result comes from the $96/32$ split: streaming video benefits from allocating most channels to the position-free content path while retaining a smaller dedicated RoPE subspace for positional structure.

Table 8: **Latent dimension and NoPE/RoPE split.** Left: sweep over latent KV dimension d_c (with $d_h^{\text{rope}} = 32$). Right: decoupled RoPE dimension with $d_h^{\text{nope}} + d_h^{\text{rope}} = 128$ at $d_c = 192$; the $96/32$ row is the default model. Semantic/Quality/Total: 30s generations, full VBench suite. **Mem.**: KV cache compression ratio relative to the dense per-token KV cache. **FPS**: throughput at batch size 1 on one H100 80 GB.

(a) Latent dimension d_c

d_c	Semantic↑	Quality↑	Total↑	Mem.↓	FPS↑
64	77.42	79.18	78.30	32.00×	26.93
128	82.16	84.31	83.24	19.20×	27.00
256	82.74	84.58	83.66	10.67×	26.93
512	82.41	84.39	83.40	5.65×	26.79

(b) NoPE/RoPE split

d_h^{nope}	d_h^{rope}	Semantic↑	Quality↑	Total↑
112	16	74.62	78.31	76.47
64	64	79.54	82.12	80.83
32	96	75.88	80.74	78.31
96	32	83.02	84.76	83.89

H.2 Per-Stage Curriculum

Table 9 evaluates every stage checkpoint of VideoMLA and of the matched dense-MHA control under the protocol of Table 1. The latent student leads the dense control at every stage, starting from Teacher Forcing. Each stage narrows the drop from 30s to 60s; only the full VideoMLA pipeline closes it (0.859 at both horizons), whereas the dense control still loses 0.029 after DMD. Sensitivity of MLA students to distillation has been reported for spectral-initialization conversions in language models, a premise VideoMLA does not rely on (Section 5); the bottleneck adds no distillation fragility here.

Table 9: **Per-stage curriculum** (VBench-Long Overall, protocol of Table 1). Both models share the identical pipeline.

	Dense MHA (matched)		VideoMLA	
Stage	30s	60s	30s	60s
Teacher Forcing only	0.638	0.549	0.702	0.641
+ Consistency Distillation	0.729	0.671	0.798	0.756
+ DMD (full pipeline)	0.781	0.752	**0.859**	**0.859**

H.3 Cache Eviction and Window Geometry

Table 10 varies the eviction policy and the window geometry for the same VideoMLA checkpoint at 60s and 120s. With a fixed six-frame budget, the three eviction families (FIFO with a sink, Deep-Forcing-style sink plus pruning, and learned retention weighting) are equivalent within noise. Retrieval is largely content-addressed (75% of scoring channels are position-free), so the choice of interior slots matters little; only the sink is indispensable, and removing it costs 0.038 at 60s and 0.062 at 120s. What moves long-horizon quality is window size. Because the per-token state is 13.7× smaller, VideoMLA can cache 82 frames with the bytes dense MHA spends on six, and longer windows improve quality monotonically.

Table 10: **Eviction policy and window geometry** (VBench-Long Overall). Same checkpoint throughout. The last row uses the cache bytes that dense MHA spends on a six-frame window.

Policy	**60s**	**120s**
FIFO, 1 sink + 5 recent (default)	0.859	0.836
Deep-Forcing-style sink + pruning	0.856	0.838
Learned retention weighting	0.857	0.839
2 sinks + 4 recent	0.851	0.831
No sink, 6 recent	0.821	0.774
FIFO, 1 sink + 13 recent	0.861	0.849
FIFO, 1 sink + 81 recent (dense-equal bytes)	**0.864**	**0.858**

I Additional Rank-Budget Analysis

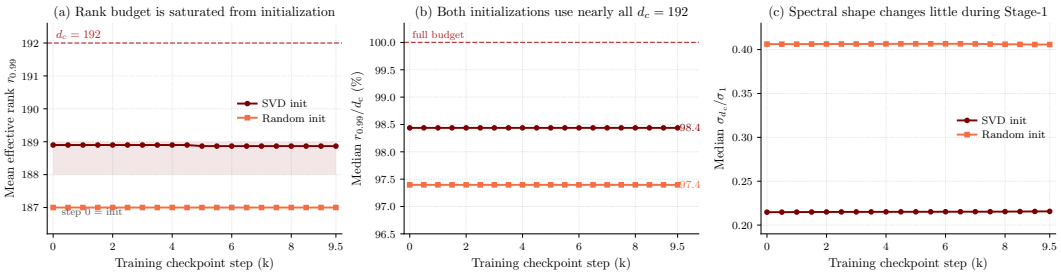


Figure 9: **Rank-budget saturation during Stage-1 training.** At $d_c = 192$, both SVD and random initialization occupy nearly the full latent rank budget from initialization, with stable effective rank and spectral tail throughout training.

Initialization at an aggressive budget. Fig. 9 compares SVD and random initialization at $d_c = 192$. Table 11 repeats the comparison at $d_c = 64$, where quality already degrades. Both initializations saturate the budget from the first checkpoint, converge to the same data-conditioned subspace, and reach comparable quality, far above untrained truncation. Untrained truncation is even less predictive at $d_c = 64$. The degradation at small budgets is therefore a capacity limit, not an initialization or spectrum effect.

Table 11: **SVD vs. random initialization at $d_c = 192$ and $d_c = 64$.** $r_{0.99}/d_c$: budget utilization of the composed operator. Total: 30s, full VBench suite. Cross-init overlap: subspace overlap between the trained SVD and random branches (chance level $d_c/1536$).

d_c	Initialization	$r_{0.99}/d_c$ (%)	Total	Cross-init overlap
192	SVD	98.4	83.89	0.81
192	Random	97.4	83.62	
64	SVD	97.8	78.30	0.77
64	Random	96.9	79.8	
64	Truncated SVD, untrained	–	35.9	–

Subspace reorganization. Table 12 reports the overlap of the learned top-d_c subspace with the weight-spectral subspace, with the data-whitened subspace, and with the other initialization branch, at initialization and after the frozen Stage-1 warm-up. Both branches move toward the data-whitened subspace and end in nearly the same place. Because only MLA parameters are trainable during the warm-up, this is rotation at constant rank.

Table 12: **Subspace overlap at initialization $\rightarrow$ after the frozen Stage-1 warm-up** ($d_c = 192$; chance level $192/1536 = 0.12$).

Initialization	Weight-spectral	Data-whitened	Other branch
SVD	$1.00 \rightarrow 0.78$	$0.64 \rightarrow 0.83$	$0.12 \rightarrow 0.81$
Random	$0.12 \rightarrow 0.31$	$0.12 \rightarrow 0.79$	–

Data-conditioned rank by partition. To test whether a video-specific allocation could exploit anisotropic redundancy, we measure the data-conditioned effective rank $r_{0.90}$ of dense KV activations on different token partitions (Table 13). Every partition lies in a narrow band (168–189), and $d_c = 192$ sits just above all of them. The only asymmetry runs against a recency-tiered design, and it is not a rank effect: every later chunk reads the sink, so errors in the sink compound longest. The matched-byte variants built on these partitions are reported in Table 5b. Time/space splits fix per-axis budgets that the joint latent already allocates per layer and token. Both recency tiers lose, compressing the sink most. Motion-adaptive budgets never win, even on motion-heavy prompts, because starved static chunks fall below their own rank and the error compounds.

Table 13: **Data-conditioned effective rank of dense KV activations by partition** (median layer, 512 clips).

Partition	$r_{0.90}$	**Energy at rank 192**
Pooled (all tokens)	181	91.4%
Spatial variation	178	91.8%
Temporal variation	184	90.9%
Sink slot	180	91.2%
Middle slots	176	92.0%
Recent slots	182	91.1%
Static chunks (bottom motion quartile)	168	93.1%
Motion-heavy chunks (top motion quartile)	189	90.2%
Weight-spectral (Fig. 1)	–	45.8%

J Extended Efficiency Comparison

Table 14 extends the efficiency comparison of Table 3 to every named system with an official release: all nine baselines of Table 1 plus Anchor Forcing and MemRoPE. Relax Forcing and VideoSSM have no public code or checkpoints. All measurements use the same B200, resolution, prompts, and seeds as Table 3. Throughput follows attended-token count and sampler: shorter windows are faster, and joint re-denoising and cache bookkeeping are slower. VideoMLA is fastest despite attending to its full window, because absorption removes dense reconstruction traffic. Ten of the eleven baselines share one memory line up to small constants: they decide which tokens are kept, but every kept token stores the same dense layout. Token policies are orthogonal to VideoMLA and can run unchanged on the latent cache.

Table 14: **Efficiency of all baselines** on one B200 (batch size 1 unless noted; memory at the 21-latent-frame geometry). *Frame-level systems emit single frames and therefore have lower first-output latency; VideoMLA has the lowest latency among chunk-wise systems.

Model	**Cache (scalars/token/layer)**	**Throughput (fps)** ↑	**Latency (s)** ↓	**Peak mem. (GB)** ↓	**Slope (GB/sample)** ↓	**Max batch** ↑
CausVid	3,072	10.85	7.42	14.9	6.27	27
Self-Forcing	3,072	19.48	4.19	14.9	6.26	28
Causal Forcing	3,072	18.21	4.15	14.9	6.26	28
Rolling Forcing	3,072	16.02	1.18*	15.0	6.29	27
Deep Forcing	3,072	16.88	4.36	15.1	6.26	28
Reward Forcing	3,072	22.63	3.51	14.9	6.26	28
LongLive	3,072	21.34	1.05*	14.9	6.26	28
Infinity-RoPE	3,072	19.42	3.92	14.9	6.26	28
Anchor Forcing [30]	3,072 + anchors	17.66	4.24	15.6	6.58	26
MemRoPE [13]	3,072 + memory tokens	19.87	3.86	15.2	6.26	28
LongSANA	linear state	19.35	4.48	12.3	1.28	138
VideoMLA	**224**	**23.96**	**3.38**	**9.4**	**0.82**	**224**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The introduction states the contribution as a validated transfer of DeepSeek-V2 MLA to causal video diffusion together with an analysis of why it works, rather than a new video-specific attention mechanism. Each claim is supported by the matched dense-MHA control (Table 2), the isolation ablations (Table 4), the long-horizon comparison (Table 1), and the analysis in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the scope of the contribution, the evaluation limits (one backbone, resolution, and pipeline; one generation per prompt; no confidence intervals; VBench as an imperfect proxy; remaining long-horizon artifacts; no VLM-based judgment), the efficiency caveats (cache-state vs. full-system memory, kernel and hardware dependence), and the lower bound on the latent budget.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: We propose an architectural modification to base model rather than fully theoretical framework.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 3.4, Section 4.1, and Appendices B–E (Table 6) specify the architecture and projection shapes, initialization, Stage-1 warm-up, per-stage step counts, learning rates, batch composition, training data, cache geometry, and the evaluation protocol (VBench version, prompt sets, seeds, and baseline regeneration).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The code and data are not publicly available at submission time. We plan to release the **code publicly after publication**, along with instructions for reproducing the main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main settings are given in Section 4.1; all hyperparameters, the optimizer, per-stage batch composition and step counts, and the evaluation protocol are listed in Appendices B and E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: All methods are evaluated with deterministic seeds shared across methods (seed = prompt index) on 200 long-horizon prompts or the full 946-prompt VBench suite, with one generation per prompt, and user-study scores are means over 200 ratings per model. We do not report error bars, confidence intervals, or significance tests; this is stated as a limitation in Section 6.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All training uses one node of $8\times$ NVIDIA B200 GPUs with the per-stage step counts and batch composition given in Section 4.1 and Table 6; all inference and efficiency measurements use a single B200, except the latent-size ablation FPS, which is measured on one H100 80 GB.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research was conducted in accordance with the NeurIPS Code of Ethics. The only human-subject component is an anonymized perceptual user study with crowd-sourced raters (Appendix B); no private or sensitive personal data are collected. Potential misuse and broader impacts are discussed in Section 6.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 6 discusses positive impacts (lower deployment cost, broader access to creative and educational tools) and negative impacts (cheaper production of deceptive synthetic media), together with mitigations such as usage restrictions and provenance tools.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We are not releasing dataset / models at the time of submission.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the creators of all models, datasets, and benchmarks we use (Wan2.1, OpenVid-1M, VidProM, VBench, MovieGen Video Bench, and all baselines) and use them under their respective licenses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: We are not releasing any new asset.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: Appendix B gives the full protocol (recruitment, number of participants, prompts and videos, assignment, anonymization and randomization, the three rating questions, and compensation), and Fig. 8 shows the interface with the instructions given to participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [Yes]

    Justification: The paper includes a user study with human participants. The study was conducted following institutional review procedures, and IRB approval or an equivalent ethics review/exemption was obtained as required by our institution. Participants were informed of the study purpose, potential risks, and data handling procedures before participation, and informed consent was obtained.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are only used for writing and editing. Not in core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.