OmniMem: Scalable and Adaptive Memory Retrieval for Long Video Generation

Lin Zhao[1,2]* **Yushu Wu**[1]* **Yifan Gong**[2] **Yanzhi Wang**[1] **Pu Zhao**[1]†
[1]Northeastern University [2]Adobe Research
Project Page: `https://wuyushuwys.github.io/OmniMem`



Figure 1: **OmniMem preserves object identity while maintaining rich motion in long video generation.** SWA shows object drift, and Sink-SWA produces repetitive motion.

Abstract

Autoregressive (AR) video generation extends videos by producing latent chunks sequentially, but scaling to long videos requires repeated access to a growing historical KV cache. Existing methods reduce this cost by truncating the KV cache or compressing it into implicit memory, but both lose explicit access to query-relevant historical details. We propose OmniMem, an explicit full-range memory retrieval framework that performs sparse KV retrieval over the historical cache. To make this practical for chunk-based AR video generation, OmniMem addresses two issues: (i) local bias in sparse KV selection and (ii) Union Explosion in memory access. Adaptive Window Exclusion removes local-window blocks from the selection candidates when sufficient long-range history is available, preserving the sparse budget for informative long-range retrieval. Query-Shared KV Selection reduces cross-query diversity, while Per-Head Scattered KV Access avoids expanding head-specific selections into a large selected KV buffer. This allows each attention head to retrieve non-contiguous KV blocks according to its own selection pattern. Experiments on long-video generation show that OmniMem improves Dynamic Degree by 52.3% and preserves strong consistency over strong baselines, while maintaining comparable memory usage.

*Equal contribution.
†Corresponding author.

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

1 Introduction

Autoregressive (AR) video generation with Diffusion Transformer (DiT) backbones [1, 2, 3, 4, 5, 6, 7, 8] has catalyzed significant progress in long video generation. By generating video sequentially, AR models can go beyond the fixed temporal horizon of standard bidirectional models [9, 10, 11, 12, 13, 14, 15]. This property is important for applications that require long, coherent visual sequences, such as interactive gaming, robotics, and world models [16, 17, 12].

Nevertheless, scaling AR video generation to long sequences introduces a fundamental computational challenge. As video length increases, the KV cache grows linearly with the number of generated chunks, increasing both memory consumption and the attention computation required at each step. A simple solution is to discard distant KV states once the cache becomes too large. However, this weakens long-range dependencies and causes the generated subject to gradually lose its identity, resulting in noticeable drift in color, shape, and trajectory across frames as illustrated in Fig. 1. Existing methods mainly address this drift issue in two ways. The first retains a fixed set of anchor tokens or memory banks (*e.g.*, sink tokens) to stabilize the generation [18, 19, 20]. While these help preserve identity, we observe that they often lead to repetitive motion patterns in long rollouts, indicating a trade-off between consistency and motion dynamics (Fig. 1). The other introduces auxiliary modules (such as SSMs [21] and Conv3D encoders [22]) to compress the history into a compact state, but this may lose fine-grained details needed for long-horizon consistency.

These limitations call for a memory mechanism that can perform **full-range** sparse retrieval over the history, explicitly selecting and accessing query-relevant context on demand [23]. Nevertheless, it is non-trivial to apply the sparse memory retrieval for chunk-based AR video generation models, complicated by two specific challenges: (i) **local bias** and (ii) **union explosion**.

Local bias is introduced by the attention scores to select a limited number of full-resolution KV blocks from the pooled historical cache. Since attention naturally favors nearby visual context, the attention scores for selection are dominated by nearby KV blocks, and therefore the selected blocks tend to concentrate heavily around the current chunk, which is unable to retrieve long-range historical memory. **Union explosion** caused by cross-query and cross-head selection divergence can undermine the memory benefit of sparse retrieval. Although each query retrieves only a small number of KV blocks, chunk-based AR video generation processes thousands of query tokens (*e.g.*, 4–5K) simultaneously at each AR step, unlike LLM decoding, which typically generates one query token per step. Different queries within the same chunk may select different historical regions due to the diverse visual content in the chunk. Different attention heads may further prefer different historical regions because of their distinct attention patterns. In this case, sparse retrieval reduces the attention computation for each query, but the union of distinct selected KV blocks across all queries and heads can still be large. Although the method remains sparse in computation, it can still require a large KV memory footprint for the current AR step. We refer to this memory-side blow-up as Union Explosion.

To address these issues, we propose OmniMem, an explicit memory retrieval framework for chunk-based AR video generation. OmniMem addresses local bias with Adaptive Window Exclusion, which removes the KV blocks already covered by the sliding-window branch from the selection candidates, and thus preserves the sparse selection budget for informative long-range retrieval. To address Union Explosion, OmniMem introduces Query-Shared KV Selection and Per-Head Scattered KV Access. Query-Shared KV Selection reduces cross-query selection diversity by letting nearby query tokens share the same sparse KV selection. Per-Head Scattered KV Access avoids materializing a large merged cache across attention heads as the video grows longer. Instead, each attention head directly retrieves its own selected non-contiguous KV blocks from the historical cache. Together, these components make explicit long-range retrieval practical for chunk-based AR video generation. Extensive experiments demonstrate that OmniMem sets a new state of the art for long video generation, simultaneously improving motion dynamics and long-range consistency over prior designs, while incurring only 1.7% additional memory over LongLive [18].

Our main contributions are summarized as follows:

- We analyze the query regime in chunk-based autoregressive video generation and identify two issues that limit explicit memory retrieval: local bias in block selection and Union Explosion caused by cross-query and cross-head divergence.
- We propose OmniMem, combining Adaptive Window Exclusion, Query-Shared KV Selection, and Per-Head Scattered KV Access to enable explicit and scalable long-range memory retrieval.

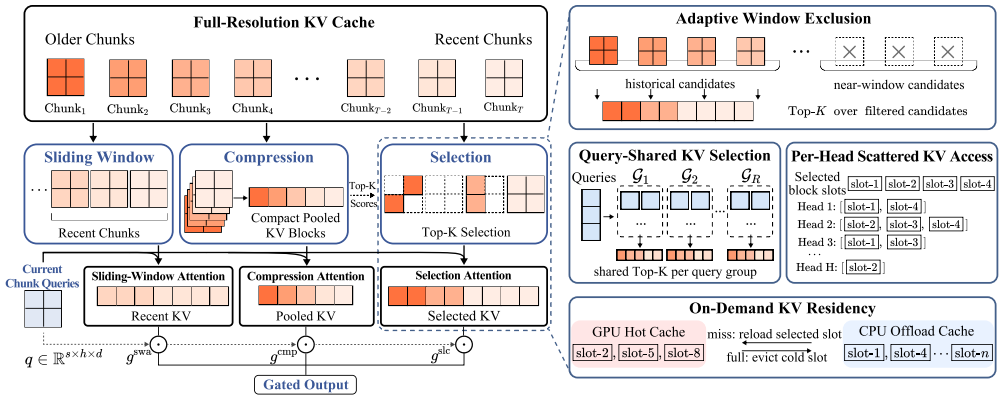


Figure 2: **Overview of OmniMem.** Current-chunk queries attend to recent KV blocks, pooled historical KV blocks, and retrieved full-resolution KV blocks through sliding-window, compression, and selection attention, respectively. The right panels summarize the key retrieval and access designs: filtering near-window candidates before Top-K selection, sharing Top-K selection within query groups, and accessing per-head block slots with on-demand GPU residency.

- Our framework improves the Dynamic Degree score by 52.3% over the strongest baseline on VBench-Long, while preserving visual consistency with only 1.7% additional memory overhead.

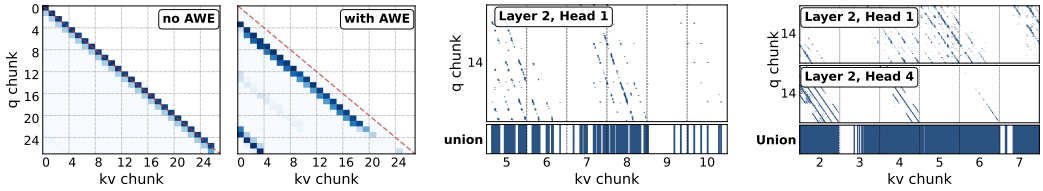
2 Related Work

Long Video Generation. Recent video generation models increasingly adopt causal, AR, or streaming formulations to improve generation throughput by generating videos sequentially and reusing intermediate states across frames. Prior efforts [24, 25] employ chunk-based AR generation and diffusion forcing for long-video synthesis, while distillation-based methods [12, 26] convert bidirectional models into few-step AR generators with distribution matching. To reduce error accumulation over long rollouts, LongLive [18] and Self-Forcing++ [27] introduce rollout-aware training, Rolling Forcing [19] denoises within a rolling window using attention sinks as global anchors, and MMM [28] combines long-context flow matching with sliding-window distribution matching in a non-AR setting. These methods improve long-horizon consistency through advanced training and denoising designs, while OmniMem explores a different scheme enabling explicit memory retrieval over the historical KV cache.

KV Cache Compression and Memory Retrieval in Long Video Generation. KV cache management is a key bottleneck in long-horizon video generation. In the LLM and streaming-video domains, strategies such as token eviction [29, 30], quantization [31], and streaming inference [32, 33] have been extensively studied and are now being adapted to autoregressive video generation, where the spatiotemporal cache grows even more aggressively. Recent approaches tackle this through low-bit KV-cache quantization [34], history compression via learned embeddings [22] or SSM-based global memory [21], and cache sparsification, retaining only informative tokens across chunks [35] or activating relevant memory on demand [20]. Further refinements exploit head-wise context redundancy [36] and salience-based policies that distill bidirectional knowledge for token importance estimation [37].

3 OmniMem: Explicit Memory Retrieval and Access

We investigate chunk-based Autoregressive (AR) video generation with a causal DiT backbone. Unlike token-wise AR generation in LLMs, DiT-based chunk generation processes thousands of tokens simultaneously in each chunk, leading to high memory cost especially in long video generation. The preliminary for AR video generation and KV cache are detailed in Appendix Sec. A.



(a) Local bias in Top-K selection (b) Cross-Query Divergence (c) Cross-Head Divergence

Figure 3: **Local bias and Union Explosion in selection attention.** (a) Top-K selection focuses near the current chunk without AWE, and shifts to long-range blocks with AWE. (b) Different query chunks in one head select different blocks. (c) Different heads also select different regions. (b) and (c) together cause Union Explosion. Note that each chunk contains a number of tokens (*e.g.*, 4-5K).

3.1 Framework

Problem Formulation. Long-video generation with causal DiTs is bottlenecked by the growing historical KV cache. As shown in the visualization (Appendix Sec. D) of the attention maps from Self-Forcing [26], many heads rely on scattered long-range historical positions, suggesting that the memory cannot be discarded. This motivates us to treat long-range memory access as an explicit retrieval problem over historical context. The goal is to preserve informative long-range dependencies while avoiding dense attention over the full history. This formulation raises two questions: which historical blocks should be retrieved for the current chunk, and how should these selected blocks be efficiently organized across thousands of queries with multiple attention heads.

Retrieval over Historical KV Cache. Building on recent advances in sparse attention for LLMs, particularly Native Sparse Attention (NSA) [23], we formulate long-range memory access as explicit block-level retrieval over the historical KV cache:

$$o = \sum_{c \in \mathcal{C}} g^c \cdot \mathrm{Attn}(Q, K^c, V^c), \quad \mathcal{C} = \{\mathrm{CMP}, \mathrm{SLC}, \mathrm{SWA}\}, \tag{1}$$

where compression attention (CMP), selection attention (SLC), and sliding-window attention (SWA) attend to the pooled historical KV cache, the retrieved full-resolution historical KV blocks, and the recent KV cache, respectively. Their outputs are fused by coefficients $g^c \in [0, 1]$. Selection attention uses compression-attention scores to identify the top-K relevant historical KV blocks for each query. Unlike NSA [23], our compression pools over 2D spatial neighborhoods to align with the visual semantics, implemented via a one-time token reorder (details and proof in Appendix Sec. J).

Challenges. In practice, we find that it is non-trivial to apply sparse memory retrieval to chunk-based AR video generation models, owing to two specific challenges, including local bias and union explosion, which we analyze in the following sections. The two challenges correspond to two fundamental questions, respectively: which historical blocks should be retrieved, and how should these selected blocks be efficiently organized. To answer these fundamental questions, OmniMem addresses local bias with Adaptive Window Exclusion (Sec. 3.2), and union explosion through Query-Shared KV Selection and Per-Head Scattered KV Access (Sec. 3.3).

3.2 Addressing Local Bias with Adaptive Window Exclusion

We first examine which blocks are actually selected by the top-K mechanism in Eq. (1). As shown in Fig. 3a, the selected blocks are heavily concentrated near the current chunk, leaving most of the long-range history unselected. This local bias makes selection attention redundant with the sliding window and weakens the purpose of explicit retrieval of historical memory. We analyze its cause below and present an effective solution for selecting long-range blocks and jumping out of local bias.

Local Bias in Top-K Selection. Without an explicit candidate mask, the compressed attention scores driving selection attention are dominated by nearby KV blocks. We attribute local bias to three compounding factors. (i) *Strong spatiotemporal correlation among nearby video tokens*. While RoPE imposes no explicit locality constraint, its relative-position parameterization can interact with this inherent visual continuity, systematically inflating attention scores between proximate tokens. (ii) *Amplification under AR video DiTs*. The bias is further exacerbated in AR DiTs, where the current chunk is denoised from a noisy latent while historical chunks supply clean, fully generated visual context. The asymmetry in signal quality between the noisy query and its immediate (clean) neighbors reinforces the tendency of attention to anchor locally. (iii) *Granularity mismatch between*

KV blocks and video chunks. The KV block size is typically far smaller than a single video chunk. Let S denote the number of tokens per chunk, B the tokens per KV block, and N_w the number of chunks covered by the sliding window. The local window then spans approximately $\lceil N_w S/B \rceil$ candidate blocks. Absent any mechanism to exclude these blocks, a top-K selector must satisfy $K > \lceil N_w S/B \rceil$ to guarantee, in the worst case, the retrieval of even a single block outside the local window. For video chunks comprising thousands of tokens, this lower bound far exceeds practical sparse budgets, causing selection attention to collapse onto the sliding-window region rather than surface genuinely long-range memory[3].

Adaptive Window Exclusion for Selection Attention. To mitigate local bias in top-K selection, we narrow the candidate set for selection attention to historical KV blocks outside the sliding window. Since recent chunks are already accessed by SWA, selecting them again wastes the sparse budget of selection attention. By removing the blocks covered by SWA from the selection candidates, selection attention is encouraged to retrieve complementary long-range memory outside the local window.

Let $\mathcal{H}^{(n)}$ denote all historical KV blocks at the nth AR step, and $\mathcal{W}^{(n)} \subset \mathcal{H}^{(n)}$ denote the blocks covered by SWA. Since selection attention retrieves top-K blocks, Adaptive Window Exclusion removes $\mathcal{W}^{(n)}$ only when at least K blocks remain outside the window, and top-K selection is performed after Adaptive Window Exclusion:

$$\Omega_{\mathrm{SLC}}^{(n)} = \begin{cases} \mathcal{H}^{(n)} \setminus \mathcal{W}^{(n)}, & \text{if } |\mathcal{H}^{(n)} \setminus \mathcal{W}^{(n)}| \geq K, \\ \mathcal{H}^{(n)}, & \text{otherwise.} \end{cases} \qquad \mathcal{I}_{h,q}^{(n)} = \mathrm{TopK}_{b \in \Omega_{\mathrm{SLC}}^{(n)}} s_{h,q,b}. \tag{2}$$

For each query token q and attention head h, selection attention uses compression-attention score $s_{h,q,b}$ for historical block b to select block indices. The selected indices are used to fetch the corresponding full-resolution KV blocks.

Applying exclusion unconditionally can destabilize selection attention when the historical cache is short. If few or no valid blocks exist outside the recent window, the selection output may shift abruptly, which can perturb the fused representation of all attention branches and impede training stability. To mitigate this, we make exclusion *conditional on the available history*. Before a sufficient number of outside-window blocks have accumulated, selection attention operates over the full historical KV cache. Once this threshold is reached, blocks already covered by SWA are excluded.

3.3 Mitigating Union Explosion with Query-Shared KV Selection and Per-Head KV Access

While Adaptive Window Exclusion governs *which* explicit memory blocks are selected, the efficiency of long-video retrieval equally depends on *where* these blocks reside. We now turn to the management and access of selected blocks. Under a memory budget, we maintain only a bounded set of explicit memory blocks on the GPU and offload less recently used historical blocks to CPU memory, as illustrated in Fig. 2. Under this design, the selected-block union becomes the critical quantity governing efficiency: prior to computing selection attention, all blocks chosen for the current chunk must be resident on the GPU. A larger union not only increases the volume of KV entries on GPU, but also inflates the temporary buffer required to materialize the selected KV, in a naïve implementation.

In LLM decoding, each step generates a single query token, so top-K selection produces a small and temporally stable set of blocks. Chunk-based AR video generation departs from this regime in a fundamental way: thousands of query tokens are processed in parallel, and distinct queries or attention heads may attend to disparate historical regions. Consequently, the union of selected blocks can grow rapidly within a single chunk—a phenomenon we term **Union Explosion**. We analyze this growth along two complementary axes: (i) cross-query divergence across queries, and (ii) cross-head divergence across attention heads.

Cross-Query Divergence. LLM decoding generates one query per step, so the top-K selection naturally yields a small set of K historical KV blocks. AR video generation jointly produces a sequence of latent tokens within each chunk. Different queries in the sequence carry different preferences and may select different historical KV blocks, so the chunk-level set of selected blocks is the union of selections across all queries (Fig. 3b), which can grow up to $S \cdot K$ in the worst case [4].

[3]For example, with $S = 4680$, $N_w = 3$, and $B = 60$, the local window spans $\lceil 3 \times 4680/60 \rceil = 234$ KV blocks, far above the top-K budget in practice.

[4]As a simple illustration, consider $S = 4$ query tokens, each selecting one block from a history of 4 blocks. If the four queries select distinct blocks, the chunk-level union covers the full history despite each per-query budget is one.

Cross-Head Divergence. Popular video DiTs [38, 39, 6] use multi-head attention (MHA), where each query head has its own KV head. This one-to-one structure makes both the CMP scores and the retrieved KV blocks head-specific. Empirically, as shown in Fig. 3c, different heads exhibit distinct selection patterns in chunk-based AR video generation. Some heads retrieve long-range context, while others focus on recent chunks. This motivates head-specific retrieval rather than relying only on fixed global memory, such as sink tokens [18, 19].

To effectively address the above challenges, we propose Query-Shared KV Selection for cross-query divergence, and Per-Head Scattered KV Access for cross-head divergence, as detailed below.

Query-Shared KV Selection. Cross-query divergence manifests at the query-token level. The spatially and temporally adjacent query tokens exhibit strong local correlation, tending to retrieve overlapping sets of historical KV blocks. To effectively mitigate this, we propose Query-Shared KV Selection, which exploits this local correlation explicitly by assigning a single shared KV-block selection to each group of adjacent query tokens, with independent selection across various groups.

Formally, the S query tokens are partitioned into $R = \lceil S/G_q \rceil$ adjacent query groups, where G_q denotes the number of query tokens per group. All query tokens within the same group share a unified top-K block selection, effectively constraining intra-group selection diversity without sacrificing the representational flexibility of inter-group retrieval. Thus, the per-query selection in Eq. (2) becomes

$$\bar{s}_{h,r,b} = \mathrm{AvgPool}_{q \in \mathcal{G}_r} s_{h,q,b}, \quad \mathcal{I}^{(n)}_{h,r} = \mathrm{TopK}_{b \in \Omega^{(n)}_{\mathrm{SLC}}} \bar{s}_{h,r,b}, \tag{3}$$

where $\mathcal{G}_r$ denotes the set of token indices belonging to group r. This reduces the worst-case selected block union along the query dimension from $S \cdot K$ to $R \cdot K$, while preserving diversity across query groups. It also makes the selected KV access more regular, allowing the selection-attention kernel to use larger dense tiles and improve GPU Tensor Core utilization (details in Appendix Sec. B).

Per-Head Scattered KV Access. As shown in Fig. 3c, different heads often prefer different historical regions, so forcing all heads to share the same selected blocks under a fixed sparse budget may reduce the expressiveness.Therefore, we perform selection independently for each head.

The challenge is how to represent these head-specific selections efficiently. Under the offload-and-reload setting, selected full-resolution KV blocks must be resident on GPU before selection attention is computed. A straightforward implementation gathers these blocks into a temporary selected-KV buffer for the attention kernel. Specifically, following the query sharing above, let $\mathcal{I}^{(n)}_{h,r}$ denote the top-K block indices selected by query group r for head h. The blocks accessed by head h are

$$\mathcal{U}^{(n)}_h = \bigcup_{r=1}^{R} \mathcal{I}^{(n)}_{h,r}, \qquad R = \left\lceil \frac{S}{G_q} \right\rceil. \tag{4}$$

The cross-head union determines which physical KV blocks must be resident on GPU or reloaded from CPU memory. The bottleneck here is not the attention computation itself, but rather the selected-KV representation required by the attention kernel. Materializing this union uniformly across all heads, however, introduces a large number of unused KV slots, since each head in practice attends to only a subset of the union. If implemented by materializing a head-aligned selected-KV buffer, the footprint scales as $M_{\mathrm{merge}} \propto N_h \left| \bigcup_{h=1}^{N_h} \mathcal{U}^{(n)}_h \right|$.

OmniMem instead uses Per-Head Scattered KV Access, as shown in Fig. 2. The full-resolution KV cache is organized as per-head, chunk-level tensors, and a pointer table maps the indices of selected blocks for each head to addresses within this native storage. The selection attention kernel consumes the per-head selected block indices, reading the corresponding non-contiguous KV blocks directly. This design strictly bounds memory access to the exact blocks required by each head. In the offload-and-reload setting, avoiding the materialization of a padded, cross-head dense tensor saves massive, redundant PCIe data transfers and prevents GPU memory exhaustion. As a result, the selected-KV representation scales with the sum of per-head selections, rather than with the union across heads (see Appendix Sec. C for details):

$$M_{\mathrm{scatter}} \propto \sum_{h=1}^{N_h} \left| \mathcal{U}^{(n)}_h \right|. \tag{5}$$

This avoids materializing a cross-head selected-KV buffer while preserving head-specific retrieval.

We highlight that OmniMem is specifically designed for chunk-based AR video generation, which significantly differs from sparse retrieval [23] in LLMs, with more details above and in Appendix Sec. I.



Figure 4: **Qualitative comparison on long-video generation.** Red boxes highlight repetitive frames where LongLive [18] collapses back to early content. Full videos and additional results are provided in the supplementary material.

Table 1: **Quantitative comparison on 60-second long video generation under VBench-Long [41]. Bold** denotes the best. VRAM measured on the Nvidia H100.

Model	VRAM (GB)	Total Score	Quality Score	Semantic Score	Subject Consistency	Imaging Quality	Dynamic Degree
SkyReels-V2 [25]	28.7	79.70	82.89	66.97	94.99	60.01	53.19
RollingForcing [19]	25.3	82.46	84.34	74.95	94.97	**72.22**	47.64
LongLive [18]	22.9	82.28	83.79	76.23	95.00	69.94	53.47
MemFlow [20]	26.2	82.56	84.14	76.24	94.49	70.67	54.03
OmniMem	23.3	**83.08**	**84.74**	**76.46**	**95.09**	71.49	**82.29**

4 Experiments

Implementation Details. OmniMem is built on Wan2.1-T2V-1.3B [39]. Following prior autoregressive video generation pipelines [12, 26], we add our memory module and fine-tune the model with ODE initialization on VidProM prompts [40]. We then continue training with Self-Forcing [26] and long-video tuning, following LongLive [18]. For the memory module, we use 15×2 pooling blocks, top-12 retrieval per query group, and a query-group size of 15. Adaptive Window Exclusion removes the most recent 3 chunks from selection candidates once enough history is available. The selection-attention kernel is implemented in Triton. More details are provided in Appendix Sec. B.

4.1 Main Results

Long Video Generation. We evaluate 60-second single-prompt video generation under the official VBench-Long [41] setting to measure long-horizon quality, consistency, and motion dynamics. We compare against the representative methods that target for solving long video problems as shown in Tab. 1. OmniMem achieves the best overall score and improves most long-video metrics, with a particularly large gain in Dynamic Degree. Prior methods obtain Dynamic Degree scores below 55, indicating that they tend to produce limited motion under long rollouts. In contrast, OmniMem achieves 82.29 of Dynamic Degree, indicating stronger motion dynamics under the same long-video setting. In terms of memory, OmniMem uses only 1.7% more VRAM than LongLive (sink+SWA) and remains more memory-efficient than other baselines, while preserving strong consistency. The qualitative comparison in Fig. 4 shows a similar trend. Compared with LongLive [18], OmniMem maintains more stable visual details and richer motion over time, while LongLive shows limited temporal variation in the highlighted regions. Additional qualitative results are provided in Appendix Sec. K.

Table 2: **Quantitative comparison on multi-prompt 60-second long video generation.** Imaging Quality (IQ), Subject Consistency (SC), and Background Consistency (BC) are computed under VBench-Long [41]. CLIP scores are computed per 10-second segment to assess prompt adherence.

Model	Imaging Quality	Subject Consistency	Background Consistency	CLIP Score↑					
				0–10 s	10–20 s	20–30 s	30–40 s	40–50 s	50–60 s
LongLive [18]	71.94	97.09	96.08	**27.51**	25.85	25.67	25.26	25.15	24.78
MemFlow [20]	71.44	**97.78**	96.51	27.35	25.93	25.79	25.69	25.11	24.67
OmniMem	**73.85**	97.72	**96.55**	27.43	**26.07**	**26.03**	**25.89**	**25.72**	**24.99**

Table 3: **Human preference rates (%).** Higher is better.

Method	Text alignment	Overall quality	Long-range consistency
LongLive	14.9	12.4	25.9
MemFlow	24.3	21.9	18.0
OmniMem	**60.8**	**65.7**	**56.1**

Table 4: **Adapted NSA on 60-second videos.** Peak inference VRAM is measured on an H200.

Method	VRAM (GB)	IQ	SC	BC
Adapted NSA	138.4	71.32	95.23	95.74
OmniMem	**23.3**	**73.85**	**97.72**	**96.55**

Long Video Generation with Prompt Switching. We further evaluate OmniMem under a multi-prompt long video generation setting, where the input is a sequence of 6 successive prompts, each spanning 10 seconds, that together specify a 60-second video. Following LongLive [18], we construct an evaluation set of 100 multi-prompt scripts. We then evaluate the methods that support prompt switching on this set, following VBench-Long [41]. To assess prompt adherence over time, we additionally compute CLIP scores between each generated 10-second segment and its corresponding prompt. In Tab. 2, OmniMem achieves the highest IQ and BC, with competitive SC. More importantly, OmniMem maintains stable CLIP scores as the video length grows, with only a 2.44 drop from 0–10s to 50–60s. The results indicate that OmniMem preserves quality, consistency, and video-text alignment throughout long generation. At 120 seconds, OmniMem also outperforms LongLive on imaging quality, subject consistency, and background consistency under prompt switching (Appendix Sec. O).

Results on short video generation are provided in Appendix Sec. E.

Human Evaluation. We compare OmniMem, LongLive, and MemFlow on 50 randomly sampled prompt-switching cases, each producing a 60-second video. Fifteen evaluators each assess all 50 anonymous, randomly shuffled video triplets, selecting the best video separately for text alignment, overall quality, and long-range consistency (750 judgments per criterion). OmniMem receives the highest preference rate on all three criteria (Tab. 3); the protocol and criterion definitions are provided in Appendix Sec. M.

Comparison with Adapted NSA. To isolate the retrieval design, we compare with an NSA baseline using the same backbone and training recipe on 60-second prompt-switching generation. We replace NSA's 1D pooling with spatial 2D pooling and enable per-head KV management and CPU offloading during training to avoid out-of-memory failures on H200. The baseline retains NSA's block selection, without Adaptive Window Exclusion or Query-Shared KV Selection. OmniMem improves all three quality metrics in Tab. 4. For inference memory profiling, NSA runs without CPU offloading on H200. Full protocol details are in Appendix Sec. L.

4.2 Ablation Study

We ablate individual components on 20-second videos with $K = 4$, reporting Imaging Quality (IQ), Subject Consistency (SC), Background Consistency (BC), and CLIP score.

Impact of Memory Branches. We ablate the contribution of each attention branch by progressively disabling CMP and SLC, with results reported in Tab. 5. Using SWA alone yields the weakest results, while adding either CMP or SLC brings consistent improvements. Combining all three achieves the best performance, showing that coarse global memory (CMP) and fine-grained retrieval (SLC) are both essential complements to the local window (SWA).

Table 5: **Ablation on the three attention branches of the fused design.**

Variant	IQ	SC	BC	CLIP
SWA	66.74	96.76	95.85	24.05
CMP + SWA	69.31	97.89	96.80	24.89
SLC + SWA	71.13	97.85	97.02	25.37
CMP + SLC + SWA	**71.38**	**98.34**	**97.30**	**26.53**

Table 6: **Comparison of different window exclusion chunk size.** † means the default setting.

Excluded Chunks	IQ	SC	BC	CLIP
0 chunks	66.13	96.78	96.52	24.51
1 chunk	68.95	97.77	96.58	24.67
2 chunks	**71.41**	97.90	96.71	26.38
3 chunks †	71.38	**98.34**	**97.30**	**26.53**

Table 7: **Ablation on query-group size G_q.** †: default.

G_q	IQ	SC	BC	CLIP
1	71.43	98.52	97.24	26.47
15 †	71.38	98.34	97.30	26.53
30	70.14	98.34	97.01	25.98
60	69.52	98.26	96.93	24.80

Table 8: **Ablation on head-group size G_h.**

G_h	IQ	SC	BC	CLIP
12	69.34	97.13	96.56	23.64
3	70.80	97.85	96.82	26.29
1	**71.38**	**98.34**	**97.30**	**26.53**

Table 9: **Ablation on sink tokens and selection attention.**

Variant	IQ	SC	BC	CLIP
SWA	66.74	96.76	95.85	24.05
SWA+Sink	69.14	96.97	96.90	25.32
SWA+SLC+Sink	70.50	97.76	**97.25**	25.29
SWA+ SLC	**71.13**	**97.85**	97.02	**25.37**

Impact of Window Exclusion Size. As discussed in Sec. 3.2, Adaptive Window Exclusion reduces the local bias in AR video generation. We vary the number of excluded chunks to study how much of the sliding-window region should be removed from the selection candidates. As shown in Tab. 6, excluding more recent chunks substantially improves performance over no exclusion. Removing the full SWA window (3-chunk) yields the best overall result and is used as our default. This supports our design of narrowing the selection candidates to historical KV blocks outside the local window.

Impact of Query-Shared KV Selection. We vary the group size G_q to demonstrate the effect of query-shared selection. Tab. 7 indicates that increasing G_q from 1 to 15 leaves all metrics nearly unchanged, confirming that adjacent queries within a chunk share similar context needs and can be safely grouped. Therefore, query-shared KV selection mitigates the cross-query Union Explosion without sacrificing generation quality, and additionally improves training efficiency by reducing redundant selection computation. Larger groups reduce IQ, BC, and CLIP; SC is unchanged at 30 and decreases at 60, suggesting that one shared selection becomes less effective for larger groups.

Impact of Per-Head KV Access. An alternative way to eliminate cross-head divergence is to group the selection across different heads, so that grouped heads retrieve the same historical blocks. We ablate this design in Tab. 8, where the head-group size G_h denotes the number of heads sharing a common KV selection. Sharing the selection across all 12 heads significantly degrades all metrics, and even a moderate size of $G_h = 3$ still leaves a clear gap to per-head selection. This indicates that different attention heads attend to genuinely different historical regions, and forcing them to share a single selection hurts generation quality.

Additional ablations on pooling size, Top-K, and LRU cache size are deferred to Appendix Sec. F.

4.3 Scalability of Long-Range Memory Access

Fig. 5 compares OmniMem with full-history attention and a Naive Sparse baseline, which applies per-query, per-head Top-K retrieval without Query-Shared KV Selection or Per-Head Scattered KV Access. Although Naive Sparse keeps the attention computation sparse, its selected-block union spans a large fraction of historical chunks. To preserve low latency, these chunks must stay resident on GPU, causing memory to grow close to full-history attention. In contrast, OmniMem keeps the active KV set bounded through query sharing and per-head scattered access. At 717 frames, OmniMem uses $3.3\times$ less memory and runs $2.7\times$ faster than full-history attention, matching Naive Sparse latency.

5 Discussion and Further Analysis

Dynamic Retrieval over Fixed Sink Tokens. Sink-based long-video methods keep fixed early tokens or chunks as global anchors, motivated by the attention-sink pattern. However, Fig. D1 shows that many heads use historical context beyond the sink region. In contrast, OmniMem dynamically retrieves full-resolution history, so sink-like regions can still be selected when useful without restricting retrieval to fixed anchors. Adding fixed sink anchors lowers IQ, SC, and CLIP while slightly improving BC (Tab. 9), showing no consistent benefit across metrics.

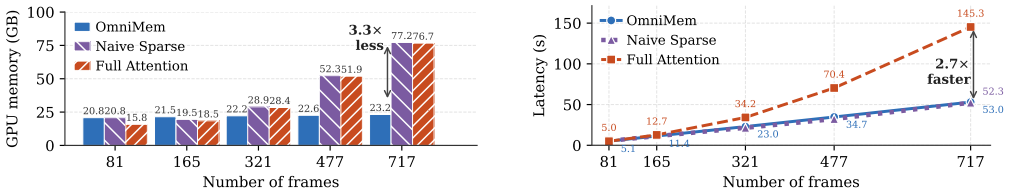


Figure 5: **Memory access scalability.** Naive Sparse reduces latency but loses the memory benefit due to Union Explosion. OmniMem maintains memory usage nearly constant while remaining efficient.

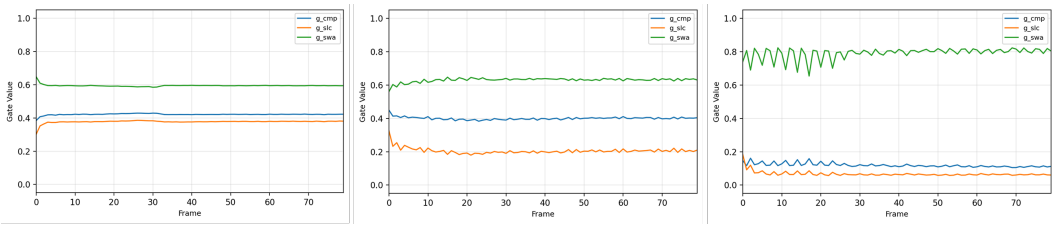


Figure 6: **Fusion gate weights across attention heads.** We visualize the gate values of the compression (g_{cmp}), selection (g_{slc}), and sliding-window (g_{swa}) branches across frames for three representative attention heads.

Selection Attention Uses Historical Tokens More Effectively. We further compare SWA+SLC with SWA+Sink under the same recent-window context. SWA+SLC selects full-resolution historical blocks per query and head, rather than always reserving fixed anchors. As shown in Tab. 9, it achieves better performance with fewer historical tokens, indicating a more effective use of historical memory.

Semantic Relevance of Retrieved History. We analyze retrieval on 40 prompt-switching videos at layers 2, 15, and 29. Frequently retrieved historical frames have higher CLIP alignment with the current prompt than lightly retrieved frames. Within the same historical frame, selected blocks also have higher DINOv2 similarity to the current query group than unselected blocks (e.g., 0.776 versus 0.367 at layer 15). These observations support the semantic relevance of retrieved history; the protocol and mean $\pm$ standard deviation results are reported in Appendix Sec. N.

Different Heads Prefer Different Memory Branches. Compression summarizes global history, selection retrieves full-resolution details, and sliding-window attention retains recent context. The fusion gates of three representative heads in Fig. 6 show distinct preferences: some balance the three branches, while others favor sliding-window attention. These patterns suggest head-level specialization in the use of historical information.

Length Extrapolation without Long-Video Finetuning. Explicit retrieval applies the same selection rule as the historical candidate pool grows. Fig. G1 shows a 20-second video generated by a model trained only on 5-second clips, without long-video finetuning. The example provides qualitative evidence of visual consistency beyond the training horizon. Our main 60-second evaluations use long-video finetuning.

6 Conclusion

We presented OmniMem, an explicit memory retrieval framework for chunk-based AR video generation. Instead of relying solely on SWA and sink tokens, OmniMem retrieves selected full-resolution historical KV blocks to enable long-range memory access. To make this practical, OmniMem addresses local-biased block selection and Union Explosion with Adaptive Window Exclusion, Query-Shared KV Selection, and Per-Head Scattered KV Access. Experiments on 60-second generation show that OmniMem improves long-video quality, temporal consistency, and motion dynamics over SOTA baselines while maintaining practical inference latency and memory footprint.

Acknowledgments and Disclosure of Funding

This work is partially supported by the National Science Foundation under Award 2414652. We would like to express our sincere gratitude to the reviewers for their invaluable feedback and constructive comments to improve the paper.

References

[1] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

[2] Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

[3] Bin Lin, Yunyang Ge, Xinhua Cheng, Zongjian Li, Bin Zhu, Shaodong Wang, Xianyi He, Yang Ye, Shenghai Yuan, Liuhan Chen, et al. Open-sora plan: Open-source large video generation model. *arXiv preprint arXiv:2412.00131*, 2024.

[4] Xuan Shen, Weize Ma, Yufa Zhou, Enhao Tang, Yanyue Xie, Zhengang Li, Yifan Gong, Quanyi Wang, Henghui Ding, Yiwei Wang, Pu Zhao, Jun Lin, and Jiuxiang Gu. Fastcar: Cache attentive replay for fast auto-regressive video generation on the edge. In *The Fourteenth International Conference on Learning Representations*, 2026.

[5] Xuan Shen, Chenxia Han, Yufa Zhou, Yanyue Xie, Yifan Gong, Quanyi Wang, Yiwei Wang, Yanzhi Wang, Pu Zhao, and Jiuxiang Gu. Draftattention: Fast video diffusion via low-resolution attention guidance. *arXiv preprint arXiv:2505.14708*, 2025.

[6] Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

[7] Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, et al. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

[8] Lin Zhao, Yushu Wu, Aleksei Lebedev, Dishani Lahiri, Meng Dong, Arpit Sahni, Michael Vasilkovsky, Hao Chen, Ju Hu, Aliaksandr Siarohin, et al. S2dit: Sandwich diffusion transformer for mobile streaming video generation. *arXiv preprint arXiv:2601.12719*, 2026.

[9] Yuqing Wang, Tianwei Xiong, Daquan Zhou, Zhijie Lin, Yang Zhao, Bingyi Kang, Jiashi Feng, and Xihui Liu. Loong: Generating minute-level long videos with autoregressive language models. *arXiv preprint arXiv:2410.02757*, 2024.

[10] Haoge Deng, Ting Pan, Haiwen Diao, Zhengxiong Luo, Yufeng Cui, Huchuan Lu, Shiguang Shan, Yonggang Qi, and Xinlong Wang. Autoregressive video generation without vector quantization. *arXiv preprint arXiv:2412.14169*, 2024.

[11] Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, Kun Xu, Hao Jiang, Nan Zhuang, Quzhe Huang, Yang Song, Yadong Mu, and Zhouchen Lin. Pyramidal flow matching for efficient video generative modeling. *arXiv preprint arXiv:2410.05954*, 2024.

[12] Tianwei Yin, Qiang Zhang, Richard Zhang, William T Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22963–22974, 2025.

[13] Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. Gamegen-x: Interactive open-world game video generation. In *International Conference on Learning Representations*, 2025.

[14] Chaorui Deng, Deyao Zhu, Kunchang Li, Shi Guang, and Haoqi Fan. Causal diffusion transformers for generative modeling. *arXiv preprint arXiv:2412.12095*, 2024.

[15] Yushu Wu, Yanyu Li, Anil Kag, Ivan Skorokhodov, Willi Menapace, Ke Ma, Arpit Sahni, Ju Hu, Aliaksandr Siarohin, Dhritiman Sagar, et al. Taming diffusion transformer for efficient mobile video generation in seconds. *arXiv preprint arXiv:2507.13343*, 2025.

[16] Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

[17] Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. *Forty-first International Conference on Machine Learning*, 2024.

[18] Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Yingcong Chen, Yao Lu, Song Han, and Yukang Chen. Longlive: Real-time interactive long video generation. *arXiv preprint arXiv:2509.22622*, 2025.

[19] Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling forcing: Autoregressive long video diffusion in real time. *arXiv preprint arXiv:2509.25161*, 2025.

[20] Sihui Ji, Xi Chen, Shuai Yang, Xin Tao, Pengfei Wan, and Hengshuang Zhao. Memflow: Flowing adaptive memory for consistent and efficient long video narratives. *arXiv preprint arXiv:2512.14699*, 2025.

[21] Yifei Yu, Xiaoshan Wu, Xinting Hu, Tao Hu, Yangtian Sun, Xiaoyang Lyu, Bo Wang, Lin Ma, Yuewen Ma, Zhongrui Wang, et al. Videossm: Autoregressive long video generation with hybrid state-space memory. *arXiv preprint arXiv:2512.04519*, 2025.

[22] Lvmin Zhang, Shengqu Cai, Muyang Li, Chong Zeng, Beijia Lu, Anyi Rao, Song Han, Gordon Wetzstein, and Maneesh Agrawala. Pretraining frame preservation in autoregressive video memory compression. *arXiv preprint arXiv:2512.23851*, 2025.

[23] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025.

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

[25] Guibin Chen, Dixuan Lin, Jiangping Yang, Chunze Lin, Junchen Zhu, Mingyuan Fan, Hao Zhang, Sheng Chen, Zheng Chen, Chengcheng Ma, Weiming Xiong, Wei Wang, Nuo Pang, Kang Kang, Zhiheng Xu, Yuzhe Jin, Yupeng Liang, Yubing Song, Peng Zhao, Boyuan Xu, Di Qiu, Debang Li, Zhengcong Fei, Yang Li, and Yahui Zhou. Skyreels-v2: Infinite-length film generative model, 2025.

[26] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025.

[27] Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Self-forcing++: Towards minute-scale high-quality video generation. *arXiv preprint arXiv:2510.02283*, 2025.

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

[29] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[30] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

[31] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37, 2024.

[32] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[33] Xinru Jiang, Lin Zhao, Xi Xiao, Yunbei Zhang, Janet Wang, Chenrui Ma, Haolin Li, Yanzhi Wang, Yifan Gong, and Octavia Camps. Dynamic hub-and-spoke memory for streaming video understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, pages 47455–47475, 2026.

[34] Haocheng Xi, Shuo Yang, Yilong Zhao, Muyang Li, Han Cai, Xingyang Li, Yujun Lin, Zhuoyang Zhang, Jintao Zhang, Xiuyu Li, et al. Quant videogen: Auto-regressive long video generation via 2-bit kv-cache quantization. *arXiv preprint arXiv:2602.02958*, 2026.

[35] Zeyu Zhang, Shuning Chang, Yuanyu He, Yizeng Han, Jiasheng Tang, Fan Wang, and Bohan Zhuang. Blockvid: Block diffusion for high-quality and consistent minute-long video generation. *arXiv preprint arXiv:2511.22973*, 2025.

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

[37] Xu Yang et al. Past- and future-informed kv cache policy with salience estimation in autoregressive video diffusion. *arXiv preprint arXiv:2601.21896*, 2026.

[38] Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, et al. Ltx-video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2024.

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

[40] Wenhao Wang and Yi Yang. Vidprom: A million-scale real prompt-gallery dataset for text-to-video diffusion models. *Advances in Neural Information Processing Systems*, 37:65618–65642, 2024.

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

[42] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21807–21818, 2024.

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

[44] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[45] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. arXiv:2304.07193.

Limitation and Broader Impact

Our Triton selection-attention kernel is tuned for current GPUs, and the latency and memory gains may vary across hardware platforms.

By making long-video generation more efficient, OmniMem may reduce the compute cost of creative tools, simulation, and world-model research. At the same time, higher-quality long video generation can be misused for disinformation or impersonation, a risk shared with general text-to-video systems. Practical deployments should be paired with safeguards such as content filtering, watermarking, and dataset controls.

A Preliminaries

Autoregressive Video Generation. We consider chunk-based AR video generation with a causal DiT backbone. Given a conditioning signal c (*e.g.*, a text prompt), the model generates a video as a sequence of latent chunks $\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(N)}\}$. At the nth AR step, the next chunk is generated conditioned on c and all previously generated chunks,

$$\mathbf{z}^{(<n)} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n-1)}\}, \qquad \mathbf{z}^{(n)} \sim p_\theta\left(\mathbf{z}^{(n)} \mid c, \mathbf{z}^{(<n)}\right), \tag{A1}$$

where p_θ denotes the conditional generation model. Each chunk contains T_c latent frames with spatial resolution $H_s \times W_s$. After flattening the spatiotemporal dimensions, one chunk yields $S = T_c H_s W_s$ latent tokens. Unlike token-wise autoregressive generation in LLMs, DiT-based chunk generation processes all S tokens in the current chunk jointly at each denoising step. As a result, a single autoregressive step introduces thousands of query tokens simultaneously, leading to a substantially different attention and memory access pattern from single-token decoding.

KV Cache. At the nth AR step, attention is computed over both the current chunk and all previously generated chunks, with causality enforced across chunks. For efficiency, the keys and values of previous chunks are cached and reused. For attention layer l and head h, we denote the full causal KV context as

$$\mathbf{K}_{l,h}^{(\leq n)} = [\mathbf{K}_{l,h}^{(1)}; \ldots; \mathbf{K}_{l,h}^{(n)}], \quad \mathbf{V}_{l,h}^{(\leq n)} = [\mathbf{V}_{l,h}^{(1)}; \ldots; \mathbf{V}_{l,h}^{(n)}]. \tag{A2}$$

Here, the first $n-1$ entries correspond to the historical KV cache, while the last entry comes from the current chunk. As n increases, the historical cache grows linearly with the number of generated chunks. Dense attention over the full historical cache incurs increasing computation and memory footprint, making long-video generation difficult to scale.

B Implementation Details

Base Model and Training. We build OmniMem on top of Wan2.1-T2V-1.3B [39] and follow a three-stage recipe. We first attach the memory module and fine-tune with ODE initialization on the VidProM prompts, following CausVid [12]. We then run Self-Forcing [26] to reduce the train-test gap during AR rollout, and finally apply the long-video tuning recipe from LongLive [18].

Training Hyperparameters. The ODE stage runs for 10K steps with a global batch size of 128 and a learning rate of 1×10^{-5}. AdamW is used with $\beta = (0.9, 0.999)$. The Self-Forcing stage runs for 600 steps and the long-video tuning stage runs for 500 steps, both with a global batch size of 64. For these two stages, we follow the optimizer settings of Self-Forcing and LongLive, using a learning rate of 4×10^{-7} for the generator and 2×10^{-6} for the fake score, with AdamW $\beta = (0, 0.999)$ and 0.01 weight decay. Training for the main OmniMem model uses 4 nodes with $8\times$ NVIDIA H100 GPUs, with gradient accumulation. The adapted NSA training protocol is described separately in Appendix Sec. L.

Memory Module. The pooling block size is 15×2, and the compression branch is a plain mean pool over each spatial neighborhood. The query-group size is 15, and each group selects the top-12 historical blocks per head for selection attention. The sliding window covers the current chunk and the two most recent historical chunks. Adaptive Window Exclusion removes sliding-window blocks from the selection candidates once sufficient historical context is available (Eq. (2)). The compression, selection, and sliding-window outputs are combined through learned per-branch sigmoid gates. The gate parameters are initialized with a normal distribution and trained jointly with the rest of the memory module, starting from the ODE stage.

Triton Kernel and Tile Padding. We implement both the forward and backward selection attention as Triton kernels (Triton 3.5.1), so that the entire training pipeline runs on the same sparse path as inference. The kernel reads non-contiguous KV blocks via the pointer table described in Sec. 3.3 and computes attention using an online softmax. For efficient use of the H100 tensor cores, the tile size along the sequence axis should be a power of two to match the matrix-multiply shapes the tensor cores expect. Our pooling block covers $15 \times 2 = 30$ tokens, which is not a power of two. We therefore pad each tile to 32 tokens inside the kernel and mask the two extra positions in the softmax, so that the padded entries do not affect the attention output. Padding adds $2/30 \approx 6.7\%$ more token positions. The resulting tile utilization is $30/32 \approx 93.8\%$, where utilization denotes the fraction of padded positions occupied by valid tokens. The head dimension is already 128 and needs no padding. A full sweep over other pooling shapes is reported in Tab. H1.

Inference and Offloading. The efficiency numbers in Sec. 4.3 are measured on a single H100 GPU. The GPU hot cache keeps 7 chunks resident and evicts cold ones with an LRU policy. We ablate this size in Sec. F.3. Offloaded KV blocks live in pinned host memory and are reloaded through PCIe when their chunks are selected. To keep PCIe transfers efficient, we explicitly bind each GPU process to the NUMA (Non-Uniform Memory Access) node that owns its PCIe root, thereby avoiding cross-socket traffic during reload.

C Shared Chunk-level KV cache and Per-Head Selection Forward

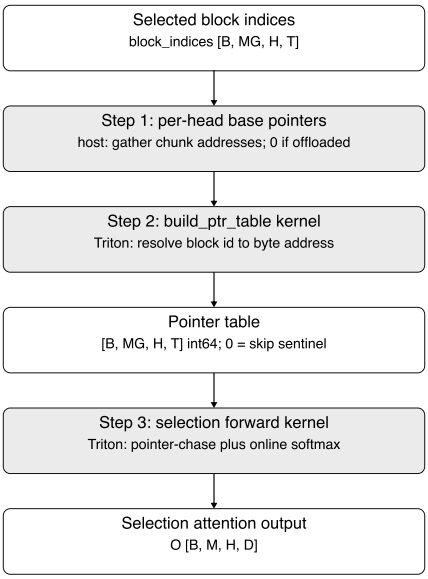


Figure C1: **Implementation pipeline of selection attention.** Step 1 runs on the host and collects the GPU addresses of the KV chunks each head needs. Step 2 is a Triton kernel that turns each selected block index into its final byte address, building the pointer table. Step 3 is the selection forward kernel. It follows each pointer in the table to read the corresponding KV block from its original storage, and skips any entry marked with 0. Only addresses pass through the pipeline; the KV data itself stays in its per-head chunk storage.

OmniMem maintains two levels of KV cache. The first level is a compact compressed cache for the compression branch, and the second level is a full-resolution chunk-level cache shared by the sliding-window and selection branches.

Compressed Cache. For each generated chunk, we pool its KV states into compressed KV blocks and append them to a contiguous compressed history. Since this cache is much shorter than the full-resolution KV cache, keeping it as a contiguous sequence makes compression attention simple and efficient.

Full-resolution Chunk-level Cache. The second level is a full-resolution chunk-level KV cache shared by the sliding-window branch and the selection branch. For each layer and head, the cache is

stored as a list of chunk tensors, $\{\mathbf{K}_{l,h}^{(1)}, \ldots, \mathbf{K}_{l,h}^{(n)}\}$ and $\{\mathbf{V}_{l,h}^{(1)}, \ldots, \mathbf{V}_{l,h}^{(n)}\}$. Sliding-window attention accesses the most recent W chunks from this cache. Because the window is contiguous and fixed-size, this requires concatenating only W large chunk tensors. A block-level physical cache would instead require concatenating $W\lceil S/B \rceil$ small tensors for the same window, where S is the number of tokens per chunk and B is the KV block size. Both layouts access the same tokens and require no padding, but the many-small-tensor concatenation introduces higher overhead in our list-based implementation.

Selection Access via Pointer Table. Selection attention uses the same full-resolution chunk-level cache, but accesses it through block indices. Each selected block index is mapped to a chunk index and an intra-chunk block offset. For each attention head, the sparse attention kernel receives a variable-length list of selected block indices and directly reads the corresponding KV blocks through a pointer table. The pointer table itself is built by a lightweight GPU kernel that maps each (chunk index, intra-chunk offset) pair to the physical address of the corresponding KV blocks. Only `int64` addresses are gathered, so this step carries address information rather than the raw KV data. An entry of 0 in the pointer table means that the chunk is on CPU, and the selection forward kernel skips these blocks with a single integer check, as shown in Algorithm 1. Thus, selection attention does not concatenate selected KV blocks into a temporary tensor, and different heads do not need to be padded to the same selected length. This two-level layout keeps compression attention compact, sliding-window attention contiguous, and selection attention fine-grained.

D The Visualization of Attention Map

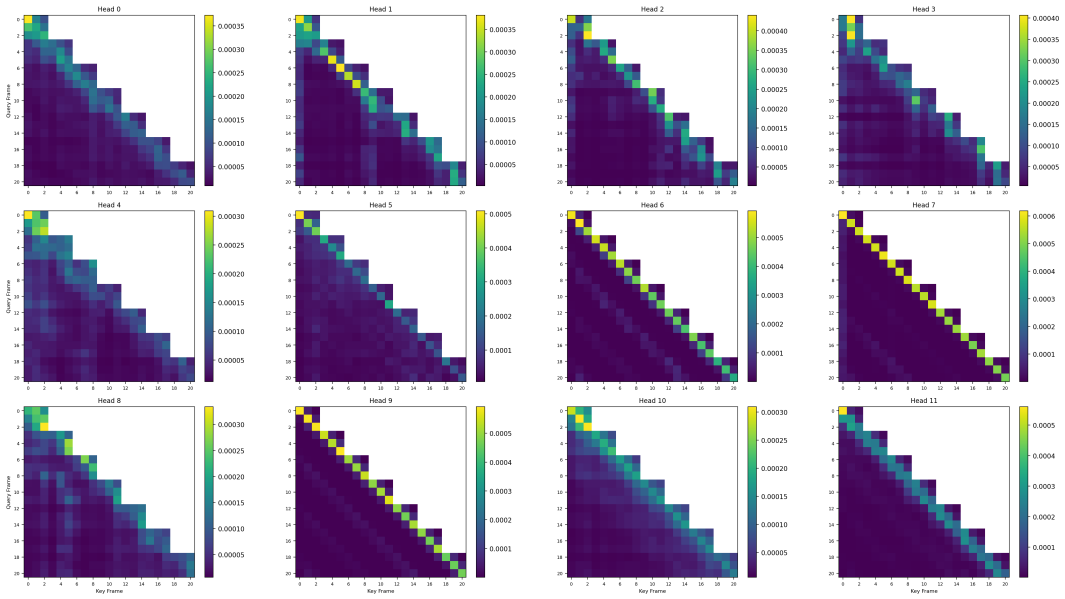


Figure D1: **Attention Map across different heads of Self-Forcing [26] with full memory.**

We visualize the per-head attention patterns of an autoregressive video generation model to reveal how different heads attend to the historical context. We adopt Self-Forcing [26], which retains the full historical KV cache without any compression. Therefore, the resulting attention maps faithfully reflect the intrinsic preference of each head. We generate a 5-second video and show the chunk-level attention map of each head in Fig. D1.

We observe two clear patterns: while some heads concentrate on recent frames, many others spread their attention across scattered positions in the history (*e.g.*, Heads 0, 3, 4, 8, 10). This confirms that (i) different heads exhibit distinct retrieval preferences (cross-head divergence), and (ii) a substantial fraction of heads rely on non-local historical context, which would be lost under sliding-window or sink-only memory.

Algorithm 1 Per-Head Selection Forward via Pointer Table

Require: Queries Q, selected block ids $I[b, g, h, 1{:}K]$
Require: Selection mask $M[b, g, h, 1{:}K]$ ▷ $M = 1$ if the block is selected
Require: Per-head GPU ptr table Π_ℓ, blocks per chunk B_{pc}
Require: Query-group size G_q, softmax scale σ

// Step 1. Build per-(b, g, h, t) pointer table.
for each (b, g, h, t) in parallel **do**
  if $M[b, g, h, t] = 0$ **then**
    $P[b, g, h, t] \leftarrow 0$ ▷ not selected; skip in attention
    continue
  end if
  $c \leftarrow I[b, g, h, t] \div B_{pc}$ ▷ chunk id of the selected block
  $o \leftarrow I[b, g, h, t] \bmod B_{pc}$ ▷ block offset within the chunk
  if $\Pi_\ell[h, c] = 0$ **then**
    WAITUNTILRELOAD(h, c) ▷ selected chunk is on CPU; wait until resident
  end if
  $P[b, g, h, t] \leftarrow \Pi_\ell[h, c] + \text{offset}(o)$
end for

// Step 2. Selection forward with online softmax.
for each (b, g, h) in parallel **do**
  $\mathbf{q} \leftarrow Q[b,\ g\text{-th group},\ h]$ ▷ G_q query tokens
  $m \leftarrow -\infty; \quad d \leftarrow 0; \quad \mathbf{a} \leftarrow \mathbf{0}$ ▷ online softmax state
  for $t = 1, \ldots, K$ **do**
    $p \leftarrow P[b, g, h, t]$
    if $p = 0$ **then**
      continue ▷ not selected or padded entry
    end if
    Load $\mathbf{K}_t, \mathbf{V}_t$ from address p
    $\mathbf{u} \leftarrow \sigma \cdot \mathbf{q}\mathbf{K}_t^\top$
    $m' \leftarrow \max(m,\ \max \mathbf{u})$
    $\mathbf{w} \leftarrow \exp(\mathbf{u} - m'); \quad \alpha \leftarrow \exp(m - m')$
    $d \leftarrow d \cdot \alpha + \sum \mathbf{w}$
    $\mathbf{a} \leftarrow \mathbf{a} \cdot \alpha + \mathbf{w}\mathbf{V}_t$
    $m \leftarrow m'$
  end for
  $O[b,\ g\text{-th group},\ h] \leftarrow \mathbf{a}/d$
end for

E Short Video Generation.

For completeness, we also evaluate OmniMem on base 5-second video generation in Tab. E1 using the standard VBench [42] prompt suite. Although our method is primarily designed for long-range memory retrieval, we verify that the proposed components also benefit for the short video quality.

Among these baselines, Self-Forcing [26] retains the full historical KV cache and serves as a strong reference for upper-bound quality on short videos. OmniMem achieves a Total VBench score of 84.29, matching Self-Forcing (84.31) within 0.02 points, while surpassing all other baselines that rely on memory compression or restricted context. Notably, OmniMem obtains the highest Quality score across the methods, suggesting that explicit per-head retrieval avoids the quality degradation.

Table E1: **Quantitative comparison on short video generation under the standard VBench [42] prompt suite.** We compare against other autoregressive models.

Model	#Params	Resolution	Vbench ↑		
			Total	Quality	Semantic
CausVid [12]	1.3B	832×480	81.20	84.05	69.80
Pyramid Flow [11]	2B	640×384	81.72	84.74	69.62
MAGI-1 [24]	4.5B	832×480	79.18	82.04	67.74
SkyReels-V2 [25]	1.3B	960×540	82.67	84.70	74.53
NOVA [10]	0.6B	768×480	80.12	80.39	79.05
LongLive [18]	1.3B	832×480	83.26	85.05	76.13
MemFlow [20]	1.3B	832×480	83.13	84.94	75.89
Self-Forcing [26]	1.3B	832×480	**84.31**	85.07	**81.28**
OmniMem	1.3B	832×480	84.29	**86.28**	76.31

Table F1: **Ablation of different KV block size** $h \times w$. h and w denote the height and width of each block.

Pooling	IQ	SC	BC	CLIP
15×2 (Top-8, 240 tokens)	71.42	98.41	97.71	27.12
15×4 (Top-4, 240 tokens)	71.02	98.31	97.04	26.28
one frame (Top-1, 1560 tokens)	69.98	98.30	97.01	26.71

Table F2: **Ablation of Top-K Selection.**

Top-K	IQ	SC	BC	CLIP
$K = 1$	71.24	97.86	96.56	25.98
$K = 4$	71.38	98.34	97.30	26.53
$K = 8$	71.42	98.41	97.71	27.12
$K = 12$	72.31	98.96	97.69	27.44

F Extended Ablation Studies

F.1 Impact of Pooling Size

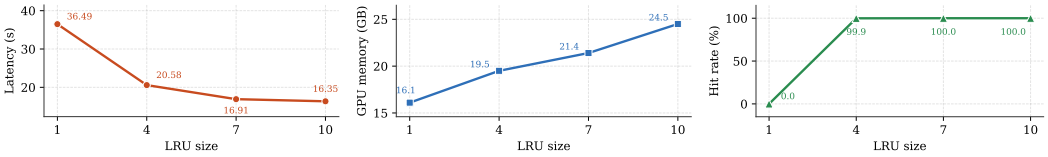
We ablate the pooling size in Tab. F1. Since our selection attention is implemented with Triton block-sparse kernels, the pooling size directly affects the fraction of valid tokens in each padded tile. As discussed in Tab. H1 of the appendix, we restrict the search to configurations whose utilization exceeds 90% for efficiency. We compare three pooling configurations. The first two (15×2 with Top-8 and 15×4 with Top-4) are matched to access the same total number of historical tokens (240), isolating the effect of pooling granularity from the token budget. The third uses one frame as the pooling unit (Top-1), which corresponds to the coarsest granularity and accesses substantially more tokens (1560). Under the same token budget, 15×2 outperforms 15×4 on each metric, indicating that finer pooling provides more accurate selection than coarser pooling. The frame-level configuration has the lowest IQ, SC, and BC despite accessing $6.5\times$ more tokens; its CLIP score lies between the two block-level configurations.

F.2 Impact of Top-K Selection

Top-K determines how many compressed KV blocks each head selects from the candidate pool per query group. We sweep $k \in \{1, 4, 8, 12\}$ and finetune each variant for long video generation from the same short video base model. The results are reported in Tab. F2. Increasing K generally improves quality. $K = 12$ achieves the highest IQ, SC, and CLIP scores, while BC is slightly higher at $K = 8$ (97.71 versus 97.69). The gains vary across metrics and intervals: CLIP increases by 0.55, 0.59, and 0.32 points for $K = 1 \to 4$, $4 \to 8$, and $8 \to 12$, respectively. These results show the quality benefit of enlarging the retrieval budget over the tested range.

F.3 The Efficiency–Memory Trade-off of LRU Size

The LRU cache stores recently retrieved KV blocks on the GPU, allowing them to be reused without reloading from CPU memory. Fig. F1 reports the impact of LRU size G_{LRU} on inference latency, GPU memory, and cache hit rate. With $G_{\text{LRU}} = 1$, the cache is too small to retain frequently selected blocks, leading to a 0% hit rate and a $2.2\times$ higher latency than the optimum due to constant CPU-GPU swapping. Increasing G_{LRU} to 4 already reaches a 99.9% hit rate, after which both latency and hit rate saturate while GPU memory keeps growing. From G_{LRU}=7 to G_{LRU}=10, latency improves by



(a) Inference Latency (b) Memory Consumption (c) Cache Hit Rate

Figure F1: **Impact of LRU cache size on 15-second video generation.** We compare (a) inference latency, (b) GPU memory consumption, and (c) cache hit rate under different LRU cache sizes.



Prompt: *"A dynamic over-the-shoulder perspective of a chef meticulously plating a dish in a bustling kitchen. ..."*

Figure G1: **Long-video generation by OmniMem without long-video finetuning.** A 20-second video generated by OmniMem, which is trained only on 5-second clips.

only $0.56\,\mathrm{s}$ ($16.91 \rightarrow 16.35\,\mathrm{s}$), while memory inflates by over $3\,\mathrm{GB}$ ($21.4 \rightarrow 24.5\,\mathrm{GB}$). This rapid saturation is a direct consequence of our explicit retrieval design. Since each query group selects only a small set of historical blocks, the working set of frequently accessed blocks is naturally compact, so a moderate LRU cache is sufficient to cover it.

G Additional Visualization for Length Extrapolation

Fig. G1 provides the qualitative example discussed in Sec. 5: a 20-second video generated by a model trained only on 5-second clips, without long-video finetuning. The fusion-gate visualization and the corresponding analysis are presented in the main text (Fig. 6).

H Padding Efficiency for Different Pooling Sizes

Our Triton selection-attention kernel pads each block to a power-of-two tile. Tile utilization is the ratio of the number of valid tokens to the padded tile size. We ablate block size choices in Tab. H1. While smaller blocks (*e.g.*, 12) provide finer-grained selection, they have a lower fraction of valid tokens per padded tile. The larger blocks (*e.g.*, 60, 120) reduce sequence length more aggressively but may lose temporal/spatial granularity.

I Explicit Retrieval in Chunk-Based Video Generation

We highlight that our OmniMem is specifically designed for chunk-based AR video generation, which significantly differs from sparse retrieval of full-resolution KV blocks [23] in LLMs. Specifically, in standard LLM decoding, each step involves a single query token, and modern architectures frequently employ Grouped Query Attention (GQA), wherein multiple query heads share a single KV head and can therefore share retrieval decisions with negligible overhead. By contrast, each AR step in video generation processes thousands of query tokens simultaneously, and prevalent video DiTs adopt Multi-Head Attention (MHA), where each query head maintains a dedicated KV head. These architectural differences fundamentally shift the computational bottleneck of sparse retrieval.

Therefore, in token-wise decoding, sparse retrieval primarily reduces the number of historical tokens attended to at each step. By contrast, in our chunk-based video generation, sparse retrieval must additionally govern the union of selected blocks across all query tokens and attention heads, which is substantially more complex with unique challenges including local bias and union explosion. OmniMem addresses these challenges to make explicit memory retrieval practical at both the algorithmic and system levels. Specifically, Adaptive Window Exclusion refines block selection by

Table H1: **Best pooling block shapes for the Wan-2.1-1.3B[39] 480P configuration with latent shape $h = 30, w = 52$.** The reported configurations use power-of-two padded tiles of at least 16 tokens. Tile utilization (util.) is block size divided by padded size. The best tile-fit group achieves 93.8% utilization with low padding overhead.

tile fit	h	w	block size	padded	util.	blocks
best	15	2	30	32	93.8%	52
	15	4	60	64	93.8%	26
good	2	13	26	32	81.3%	60
	2	26	52	64	81.3%	30
	2	52	104	128	81.3%	15
ok	3	4	12	16	75.0%	130
	6	2	12	16	75.0%	130
	6	4	24	32	75.0%	65
poor	5	2	10	16	62.5%	156
	5	4	20	32	62.5%	78
	10	2	20	32	62.5%	78
	3	13	39	64	60.9%	40
	10	4	40	64	62.5%	39
	6	13	78	128	60.9%	20
terrible	3	2	6	16	37.5%	260
	5	13	65	128	50.8%	24
	10	13	130	256	50.8%	12

discounting recency bias in top-K scoring; query sharing mitigates cross-query selection diversity by aggregating query tokens into representative proxies before retrieval; and Per-Head Scattered KV Access circumvents the need to expand head-specific selections into a large, head-aligned KV buffer, substantially reducing memory overhead.

J Permutation Equivariance of Reorder with RoPE

We justify the token reordering step in our compression pipeline. Our compression operates on *2D spatial neighborhoods*: tokens that are spatially adjacent in the original frame layout are pooled together so that each compressed block summarizes a semantically coherent region. However, when video tokens are flattened into a 1D sequence following the standard $f \times h \times w$ raster order, the tokens belonging to the same 2D spatial block are *not* contiguous in memory, making block-wise pooling difficult to express as an efficient operation.

To address this, we introduce a one-time token reorder π that places all tokens of each 2D spatial block into a contiguous range, so that compression becomes a simple contiguous reshape and mean reduction. The natural concern is whether this reorder with the corresponding RoPE positions changes the attention output. We show below that reordering tokens and their RoPE positions [43] jointly leaves the attention output unchanged up to the same reordering, so the entire reorder–compress–attend–restore pipeline is mathematically equivalent to the unreordered computation.

Setup. Let $X \in \mathbb{R}^{N\times d}$ be a token sequence and $\Theta = (\theta_1, \ldots, \theta_N)$ the corresponding RoPE positional indices, where each θ_i encodes the spatiotemporal position (f_i, h_i, w_i). RoPE-augmented attention is

$$\mathrm{Attn}(X, \Theta) \;=\; \mathrm{softmax}\Big(\tfrac{Q'(K')^\top}{\sqrt{d}}\Big)\, V, \tag{J1}$$

where the per-token queries, keys, and values are

$$Q'_i = R(\theta_i)\, W_Q X_i, \quad K'_j = R(\theta_j)\, W_K X_j, \quad V_j = W_V X_j, \tag{J2}$$

and $R(\theta)$ is the RoPE rotation associated with position θ. Let $\pi \in S_N$ be a permutation, and $P_\pi \in \{0,1\}^{N\times N}$ its permutation matrix, satisfying $(P_\pi X)_i = X_{\pi(i)}$ and $(P_\pi \Theta)_i = \theta_{\pi(i)}$.

Proposition 1 (Joint reorder equivariance)**.** *For any permutation $\pi \in S_N$,*

$$\mathrm{Attn}(P_\pi X,\, P_\pi \Theta) \;=\; P_\pi \cdot \mathrm{Attn}(X, \Theta). \tag{J3}$$

Proof. The reordered query satisfies

$$Q'^{(\pi)}_i \;=\; R\big((P_\pi\Theta)_i\big)\, W_Q\, (P_\pi X)_i \;=\; R(\theta_{\pi(i)})\, W_Q\, X_{\pi(i)} \;=\; Q'_{\pi(i)}, \tag{J4}$$

so $Q'^{(\pi)} = P_\pi Q'$. The same holds for K' and V, giving $K'^{(\pi)} = P_\pi K'$ and $V^{(\pi)} = P_\pi V$. Using $(P_\pi K')^\top = (K')^\top P_\pi^\top$, the score matrix becomes

$$Q'^{(\pi)} \big(K'^{(\pi)}\big)^\top \;=\; P_\pi\, Q'(K')^\top\, P_\pi^\top. \tag{J5}$$

Since row-wise softmax commutes with row permutation,

$$\mathrm{softmax}\big(P_\pi M P_\pi^\top\big) \;=\; P_\pi\, \mathrm{softmax}(M)\, P_\pi^\top \tag{J6}$$

for any matrix M. Combining this with $V^{(\pi)} = P_\pi V$ and $P_\pi^\top P_\pi = I$,

$$\mathrm{Attn}(P_\pi X, P_\pi \Theta) = \mathrm{softmax}\Big(\tfrac{P_\pi Q'(K')^\top P_\pi^\top}{\sqrt{d}}\Big)\, P_\pi V \tag{J7}$$

$$= P_\pi\, \mathrm{softmax}\Big(\tfrac{Q'(K')^\top}{\sqrt{d}}\Big)\, P_\pi^\top P_\pi\, V \tag{J8}$$

$$= P_\pi\, \mathrm{Attn}(X, \Theta). \qquad \square$$

Corollary 1 (Reorder–compress equivalence)**.** *Let the reorder π be chosen such that, for every 2D spatial compression block b, the indices $\{i : (h_i, w_i) \in b\}$ form a contiguous range under π. Then mean-pooling every $|b|$ consecutive tokens of $P_\pi K'$ and $P_\pi V$ yields the same compressed key/value sequence as applying 2D spatial block-pooling to the original K' and V. Combined with Proposition 1, the entire* (reorder → contiguous compress → attention → inverse reorder) *pipeline is equivalent to* (2D-block compress → attention) *on the unreordered sequence.*

Remark 1. *The corollary justifies our implementation: we apply* `token_reorder` *once at the input and* `token_restore` *once at the output. All transformer blocks (compression / selection / sliding-window attention) operate on reordered tokens internally, yielding strictly equivalent computation while reducing the spatial compression to a contiguous* `reshape+mean` *operation on GPU memory.*

K More Qualitative Results

Long Video Generation. Fig. K1 shows 60-second single-prompt results. SWA suffers from gradual subject drift as the kitten's appearance changes across frames, since distant context is discarded. LongLive [18] preserves identity via attention sinks but produces repetitive frames (red boxes). OmniMem maintains both subject consistency and natural scene evolution.

Multi-Prompt Long Video Generation. Figs. K2 and K3 show multi-prompt 60-second results. LongLive [18] and MemFlow [20] both exhibit repetition (red boxes) for the same underlying reason: neither can attach to the full-resolution historical context. LongLive uses fixed sink tokens, while MemFlow maintains a fixed memory bank, both losing the fully fine-grained details needed to follow prompt transitions. OmniMem retrieves full-resolution historical context on demand, faithfully following prompt transitions while preserving consistency.



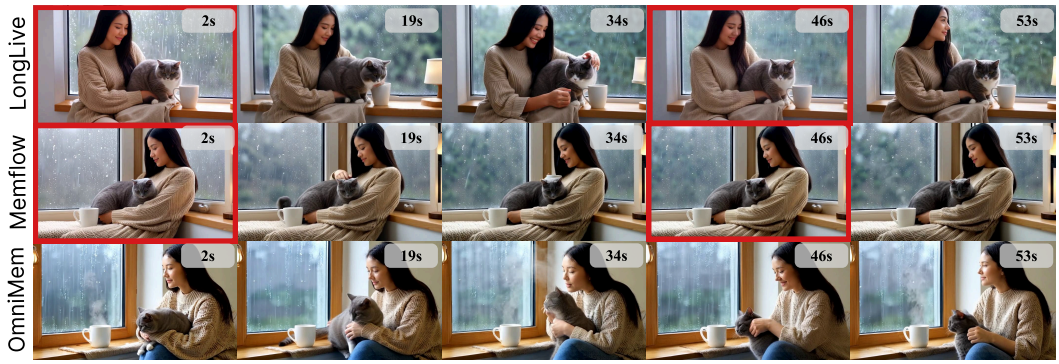
Prompt: *"A fluffy brown tabby kitten walking slowly through an autumn park covered with colorful fallen leaves ..."*

Figure K1: **Qualitative comparison on 60-second long video generation.** SWA denotes the sliding-window-only baseline. Red boxes highlight visible artifacts.



Prompt: *"A young male astronomer with short dark hair and glasses looks through a large brass telescope inside a warm observatory dome ..."*

Figure K2: **Qualitative comparison on multi-prompt 60-second video generation.** Red boxes mark repetitive frames in LongLive [18] and MemFlow [20].



Prompt: *"A cozy rainy afternoon inside a warm apartment. A young Asian woman with long straight black hair wearing an oversized beige ..."*

Figure K3: **Qualitative comparison on multi-prompt 60-second video generation.** Red boxes mark repetitive frames in LongLive [18] and MemFlow [20].

L Protocol for the NSA Comparison

We compare OmniMem with an adapted NSA [23] baseline on 60-second video generation with prompt switching. Both methods use the same backbone and training configurations for ODE initialization, Self-Forcing, and long-video tuning. To accommodate the visual token layout, we replace NSA's one-dimensional pooling with two-dimensional spatial pooling. The adapted baseline retains NSA's selection rule without Adaptive Window Exclusion or Query-Shared KV Selection.

Training memory management. The adapted NSA baseline exceeds the memory capacity of an NVIDIA H200 during training before reaching a rollout equivalent to 60 seconds of video. Training must retain gradient-related activations in addition to the historical KV cache, while the union of selected blocks can grow across the queries in each video chunk. To make this comparison feasible, we equip the baseline with the same per-head chunked KV management and asynchronous CPU offloading infrastructure used by OmniMem. This infrastructure manages KV residency and data movement while retaining the baseline's attention and block-selection rules. The quality comparison therefore evaluates the retrieval designs under matched training configurations.

Inference memory measurement. Peak inference VRAM is measured on a single NVIDIA H200. For this measurement, the adapted NSA baseline runs *without* CPU offloading, since its inference computation fits on the H200. The measured peaks are 138.4 GB for the adapted NSA baseline and 23.3 GB for OmniMem. Under this 60-second prompt-switching protocol, the baseline and OmniMem obtain Imaging Quality scores of 71.32 and 73.85, Subject Consistency scores of 95.23 and 97.72, and Background Consistency scores of 95.74 and 96.55, respectively.

M Human Evaluation Protocol

We evaluate 60-second videos generated from 50 randomly sampled prompts under the prompt-switching setting. For each prompt, we compare OmniMem with LongLive [18] and MemFlow [20]. The three videos are presented anonymously in a randomly shuffled order. Fifteen evaluators each assess all 50 triplets and select the best video separately for each of three criteria, yielding 750 judgments per criterion.

Evaluation criteria. For *text alignment*, evaluators judge whether the video follows the supplied prompts. For *overall quality*, they assess the completeness of generated objects, meaningful and dynamic motion, and the presence of flickering or other artifacts. For *long-range consistency*, they assess whether subject identity, scene layout, and visual style remain consistent throughout the 60-second video, including when a subject reappears after an extended absence.

Preference rates. A method's preference rate is the fraction of the 750 judgments selecting that method for the corresponding criterion. These are three-way preference rates, rather than pairwise win rates. OmniMem receives 60.8% of preferences for text alignment, 65.7% for overall quality, and 56.1% for long-range consistency. The corresponding rates are 14.9%, 12.4%, and 25.9% for LongLive, and 24.3%, 21.9%, and 18.0% for MemFlow.

N Semantic Relevance of Retrieved History

We examine whether the selected historical content is semantically relevant at both the frame and block levels. The analysis uses 40 generated videos, each comprising 27 chunks, with prompt switches at chunks 9 and 18. During every clean-context pass, we record the raw Top-12 block selections, using the default selection budget, from all 12 attention heads in layers 2, 15, and 29. We evaluate chunks indexed 4–26, which have historical context outside the local window.

Frame-level prompt alignment. For each generated chunk, we aggregate block-selection counts by historical frame, excluding frames within the local window. Heavily selected frames receive approximately six times as many block selections per frame as lightly selected frames. We evaluate alignment with the current segment's prompt using the CLIP [44] text–image score, computed from the decoded RGB frame and the prompt text. As shown in Tab. N1, heavily selected frames have higher prompt alignment at each inspected layer.

Block-level semantic similarity. We next compare selected and unselected blocks within the same historical frame. We apply DINOv2-B/14 [45] to decoded RGB frames to extract patch features.

Table N1: **Prompt alignment of retrieved historical frames.** CLIP text–image scores are reported as mean ± standard deviation. Higher values indicate stronger alignment with the current prompt.

Layer	Heavily selected frames	Lightly selected frames
2	0.2090 ± 0.0055	0.1953 ± 0.0055
15	0.2111 ± 0.0051	0.1962 ± 0.0060
29	0.2089 ± 0.0055	0.1942 ± 0.0055

For each current query block and historical candidate block, we average the patch features whose spatial receptive fields fall within its corresponding pixel region, then apply ℓ_2 normalization. We compute cosine similarity between the query-block representation and the selected historical blocks, and compare it with the similarity to the remaining, unselected blocks of the same historical frame. Tab. N2 shows higher similarity for selected blocks across all three layers. Together, these analyses support the semantic relevance of the retrieved history.

Table N2: **Semantic similarity of retrieved historical blocks.** DINOv2 cosine similarities are reported as mean ± standard deviation. Unselected blocks are drawn from the same historical frame as the selected blocks.

Layer	Selected blocks	Unselected blocks
2	0.731 ± 0.011	0.352 ± 0.010
15	0.776 ± 0.009	0.367 ± 0.009
29	0.757 ± 0.010	0.357 ± 0.010

O Generation Beyond 60 Seconds

We compare OmniMem with LongLive [18] on 120-second video generation with prompt switching. As shown in Tab. O1, OmniMem achieves higher Imaging Quality, Subject Consistency, and Background Consistency in this longer setting. The improvements are 1.87, 1.28, and 1.57 points, respectively.

Table O1: **120-second video generation with prompt switching.** Higher values are better for all three metrics.

Method	Imaging Quality	Subject Consistency	Background Consistency
LongLive	70.86	96.10	94.37
OmniMem	**72.73**	**97.38**	**95.94**

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 abstract and introduction clearly state the contributions and scope of the paper, which match the theoretical and experimental results presented.

   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: Limitations of our approach are discussed in the conclusion/discussion section.

   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: [Yes]

Justification: No theoretical results state their assumptions, and no proofs are needed.

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: We describe the model architecture, training procedure, hyperparameters, and datasets used in the main paper and appendix in sufficient detail to reproduce the main results.

   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: [Yes]

Justification: Code will be provided and published along with instructions for reproducing the main results. All datasets used are publicly available.

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: Training and evaluation details, including data splits, optimizer, hyperparameters, and selection procedure, are specified in the experiments section and appendix.

   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: [N/A]

   Justification: We report results averaged over multiple random seeds.

   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: Compute resources, including GPU type and approximate runtime, are reported.

   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: We have reviewed the NeurIPS Code of Ethics and confirm that the research conforms with it in every respect.

   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: Potential positive and negative societal impacts of the work are discussed in the broader impacts section.

    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: The paper does not release data or models with a high risk for misuse.

    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: All datasets, code, and models used are properly cited, and their licenses and terms of use are respected.

    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: [N/A]

    Justification: No new assets introduced in the paper.

    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: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    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: [N/A]

    Justification: The paper does not involve research with human subjects.

    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 not used as an important, original, or non-standard component of the core methods in this research.

    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.