To Infinity and Beyond: An Exploration of KV Cache Compression for Streamed Video in vLLM

Arnav Shah
University of Michigan
Ann Arbor, United States
arnshah@umich.edu

Mihir Arya
University of Michigan
Ann Arbor, United States
msarya@umich.edu

Om Joshi
University of Michigan
Ann Arbor, United States
omjoshi@umich.edu

***Abstract*—Long-context video understanding is constrained by the linear memory scaling of Key-Value (KV) caches in Transformer architectures, where extended streams frequently trigger Out-Of-Memory (OOM) errors. This work explores architectural strategies to enable effectively infinite video streaming within the vLLM inference engine. We document an iterative design process, transitioning from logical masking and physical block eviction to a hybrid attention mechanism that integrates Sliding Window Attention for high-fidelity local context with a State Space Model (SSM) history branch for compressed global context. We describe the systems-level challenges of managing "dual memory"—integrating PagedAttention blocks with fixed SSM states—and optimizing stream latency via client-side orchestration. Our evaluation demonstrates that this approach reduces peak memory usage by over 50% and achieves $O(1)$ memory growth with negligible latency overhead. We also identify trade-offs in long-term semantic detail and narrative closure. We present these results as a validation of the mechanical viability of infinite streaming on commodity hardware, highlighting the critical gap between memory capacity and memory fidelity as a primary direction for future Large Multimodal Model research.**

***Index Terms*—Large Multimodal Models, Video Understanding, State Space Models, KV Cache, Efficient Inference, vLLM, Mamba**

I. Introduction

The ability to process and understand long-form video is essential for next-generation AI assistants. Users desire to interact with real-time video streams using natural language, but this is extremely difficult to achieve in long video sequences using current architectures. As a Vision Language Model (VLM) processes a video, token representations from every single frame accumulate in the Key-Value (KV) cache, effectively loading the entire video into memory. This creates a severe memory bottleneck where the cache grows linearly ($O(N)$) with video length, quickly exhausting GPU memory for videos exceeding a few minutes [1].

The computational dynamics of this problem are orders of magnitude more severe than in text processing. If we consider a standard Vision Transformer (ViT) strategy where a single frame yields 196 tokens, a 60-minute video sampled at 1 FPS generates approximately 705,600 tokens. Computing a self-attention matrix of size $700k \times 700k$ is computationally intractable for current hardware accelerators due to both FLOPs and memory capacity [1].

The consequences of this bottleneck include:

- **High Latency:** The model slows down significantly as the cache grows, becoming memory-bandwidth bound as it fetches massive KV caches from High Bandwidth Memory (HBM).
- **Low Throughput:** The growing memory footprint reduces the number of concurrent requests the system can handle.
- **System Failures:** The system inevitably runs out of memory (OOM) and crashes for streams exceeding the hardware limit.

While techniques like token pruning and compression exist, they often degrade performance or require complex heuristics. We propose a hybrid architecture that combines two distinct mechanisms to solve this "infinite context" problem:

1) **Sliding Window Attention:** Maintains a precise, uncompressed buffer of the most recent frames (e.g., last 4096 tokens) to handle immediate context and high-frequency details [3].
2) **Mamba-style SSM:** A parallel "history branch" that compresses the entire video history into a fixed-size recurrent state [2].

This hybrid approach allows the model to retain infinite historical context (in compressed form) with constant memory overhead, while preserving the high-resolution awareness of recent events. We implement this architecture in vLLM, introducing a `HybridSSMAdapter` and `HybridAttentionBackend` that seamlessly fuse SSM computations with optimized PagedAttention kernels.

All work can be found on this GitHub repository.

II. Background and Related Work

A. The KV Cache Bottleneck

In standard autoregressive decoding, the Key and Value matrices for all previous tokens are stored in GPU High Bandwidth Memory (HBM). For visual inputs, where a single second of video can generate hundreds of tokens, this cache grows rapidly. For a 7B parameter model with a 128k context window, the KV cache alone may surpass 100 GB of memory, forcing the use of multi-GPU setups merely for capacity rather than compute power [1].

B. StreamingLLM and Attention Sinks

Xiao et al. introduced *StreamingLLM* [3], demonstrating that model stability relies disproportionately on the initial few tokens (the "attention sink") and the most recent tokens. Their work proved theoretically that models can generate infinite sequences by retaining only the sink and a sliding window of recent tokens. However, their original implementation did not address the systems-level challenges of integrating this policy into paged memory managers or JIT-compiled inference engines.

C. vLLM and PagedAttention

Our work is implemented within vLLM, a high-throughput inference engine that utilizes PagedAttention. PagedAttention mitigates memory fragmentation by storing KV states in non-contiguous physical blocks, managed via a virtual block table. Modern inference stacks (e.g., TorchDynamo, CUDA Graphs) rely on static tensor shapes for optimization. Dynamic changes in sequence length can trigger "guard failures," forcing costly re-compilation.

D. State Space Models (SSMs)

To circumvent the quadratic bottleneck, the research community has revisited Recurrent Neural Networks via Structured State Space Models (SSMs). Mamba [2] discretizes a continuous-time state space equation

$$h'(t) = Ah(t) + Bx(t)$$

into a recurrent form ($h_t = \bar{A}h_{t-1} + \bar{B}x_t$) suitable for deep learning. Crucially, the size of $h(t)$ is fixed and independent of the sequence length N. This offers $O(1)$ inference memory. However, pure SSMs often struggle with "needle-in-a-haystack" retrieval tasks—scenarios requiring the recall of specific, high-fidelity details from the distant past—necessitating a hybrid approach.

III. Methodology

Our core research objective is to enable natural language interaction with real-time, potentially infinite, video streams. However, as noted in the introduction, this is fundamentally constrained by the mechanics of the Transformer KV cache. As a VLM processes a video stream, token representations accumulate linearly. This accumulation results in high latency due to memory bandwidth saturation, low throughput due to reduced batch capacity, and inevitable system crashes (OOM) when the stream length exceeds physical HBM. Separating the video into multiple request chunks results in the streaming ability required; however, under the default vLLM implementation, the KV cache is not reused.

To address this, we frame the problem of infinite streaming along two primary research dimensions:

A. Dimension 1: Efficient Storage Architecture

The first dimension questions the necessity of storing the full history: *What is the optimal architecture for storing visual history?* We seek a balance between memory compression, task accuracy, and end-to-end latency.

- **Compression vs. Accuracy:** Can we compress the history into a fixed-size state without catastrophic forgetting?
- **Degradation:** How does this compressed context degrade over extremely long streams (e.g., hours of video)?

Our hypothesis is that a hybrid architecture, combining a fixed-size memory pool for history and an Attention window for recent frames, provides the optimal point on this trade-off curve, offering $O(1)$ memory growth with minimal accuracy loss for streaming tasks.

B. Dimension 2: Semantic Relevance

The second dimension questions the necessity of processing every frame. *What does "relevance" mean for a VLM?*

- **Filtering:** How do we detect and filter relevant frames before they even reach the model?
- **Memory Interaction:** How does frame selection interact with the KV cache structure?

While this paper primarily focuses on the architectural solution (Dimension 1), we also explore a single client-side orchestration strategy to optimize frame relevance and minimize redundant computation (Dimension 2).

C. Design Strategy

Guided by these two dimensions, our approach to enabling infinite video streaming evolved through four distinct phases. Each phase addressed a specific failure mode discovered when moving from theory to production systems.

1) Phase 1: Logical Masking: We first attempted to solve the storage problem via software-level masking, inspired by StreamingLLM [3]. The goal was to logically ignore "middle" tokens while retaining "sink" and "recent" tokens. However, this failed to solve the underlying system resource usage. Logical masking did not prevent the physical block allocator from reserving memory, leading to leaks and compiler instability.

2) Phase 2: Physical Block Eviction: To address the failures of Phase 1, we moved the solution to the allocator level. We implemented a mechanism to physically free memory blocks corresponding to the middle of the video stream. This successfully capped memory usage and enforced a static shape for the compiler. However, this introduced the issue of permanently losing context stored in evicted blocks.

3) Phase 3: Hybrid Attention: To solve the context loss problem introduced in Phase 2, we arrived at our final architecture: Hybrid Attention. We reintroduced the evicted history not as heavy KV blocks, but as a lightweight, fixed-size SSM state. This allows the system to physically evict KV blocks (solving the OOM and compiler issues) while retaining a compressed global context in the SSM branch (solving the context loss issue).

4) Phase 4: Client-Side Stream Orchestration: We encountered latency issues due to redundant computation of overlapping frames in continuous streams. To address this, we implemented a client-side orchestration layer that optimizes request structure for vLLM's prefix caching. By managing a local "smart" sliding window, the client ensures that the server reuses the KV cache for static history (sink) and shared recent frames, computing attention only for new visual data. This complementary layer minimizes computational overhead and maximizes throughput without requiring further server-side architectural changes.

IV. Implementation

We integrated our solution into the vLLM ecosystem, modifying the model executor and attention backends to support this dual-path computation. This required rethinking how state is managed in a paged memory environment.

A. Phase 1: Logical Masking

Our initial implementation attempted to enforce the StreamingLLM policy [3] through logical masking. We modified the attention kernel to accept a custom mask M where indices corresponding to the "middle" tokens were set to $-\infty$.

$$M_{i,j} = \begin{cases} 0 & \text{if } j \in \text{Sink} \cup \text{Recent} \\ -\infty & \text{if } j \in \text{Middle} \end{cases} \tag{1}$$

However, this approach failed at the system level due to the decoupling of compute and memory in vLLM:

- **Memory Leak:** The vLLM `BlockAllocator` operates on sequence length, ignoring the attention mask. As the video progressed, physical blocks continued to be allocated for masked tokens, leading to linear memory growth and OOM errors.
- **Compiler Thrashing:** The masking strategy created a "dynamic shape" problem ($L \to L + 1$). This triggered `TorchDynamo` guard failures (e.g., `tensor['x'] size mismatch`), forcing the engine to discard optimized kernels and re-compile the graph at every step.

B. Phase 2: Physical Block Eviction

To resolve the Phase 1 failures, we implemented the `StreamingVideoKVCacheManager`. This component interfaces directly with the `BlockSpaceManager` to enforce physical eviction.

The manager intercepts requests at the end of each generation step and executes the following logic (see Listing 1):

1) **Topology Identification:** Slices the request's block table into `sink_blocks`, `middle_blocks`, and `recent_blocks`.
2) **Physical Eviction:** Calls `GLOBAL_POOL.free(middle_blocks)`. This explicitly decrements the reference count of the physical pages in the GPU allocator, reclaiming memory immediately.
3) **Static Shape Enforcement:** We overwrite `request.block_table` with only the sink and recent blocks. This ensures the attention kernel perceives a constant sequence length (e.g., `NUM_SINK + NUM_RECENT`), satisfying compiler guards and enabling CUDA Graph reuse.

To note, we also explored and ultimately abandoned a compression-driven eviction strategy that coupled token-level StreamingLLM-style KV compression with block-level physical eviction. At the attention layer, a `KVCacheCompressor` generated a StreamingLLM keep policy (retain sink and recent tokens) and emitted a `keep_mask` plus `compression_info` containing candidate `blocks_to_free`. Instead of using the manager to deterministically free the middle slice and rewrite the block table, this approach propagated eviction decisions through the runtime: the worker collected `compression_info`, the scheduler validated reference counts and called `block_pool.free_blocks(...)` on `blocks_to_free`, and `freed_block_ids` were returned to workers for device-side zeroing.

Empirically, this compression-driven physical eviction path failed the cost–benefit test: compared to a masking-only baseline, it regressed end-to-end latency (TTFT/TPOT) and throughput without producing a demonstrated memory win in our local profiling artifacts. We attribute the regression to the following overheads:

1) **Static-shape enforcement was not guaranteed.** Phase 2's primary win depends on deterministically rewriting `request.block_table` to `sink + recent` In the compression pipeline, some blocks may get freed, but the attention-visible block-table shape isn't consistently forced into the constant-length form needed for stable compiler behavior and maximum CUDA Graph reuse.
2) **Token masks do not align cleanly to block boundaries.** Compression decisions are made at token granularity, while the allocator operates at block granularity. Many KV blocks are partially occupied by retained tokens, so `blocks_to_free` is conservative and often sparse, limiting reclaimable pages while preserving per-step overhead.
3) **Scheduler-mediated freeing added overhead of coordination and synchronization.** The additional handoffs (attention to worker to scheduler to worker), ref-count checks, block-pool churn, and post-free device zeroing introduced synchronization on the critical path, dominating any limited gains from page reuse.
4) **Batching reduced how much memory we could effectively reclaim.** Conservative request attribution and ref-count validation in multi-request batches further reduced the set of blocks eligible for safe freeing, lowering the benefit term without reducing the fixed coordination cost.

While physical eviction is required to prevent allocator-driven KV growth, driving frees indirectly from token-level compression events can be net-negative unless it also enforces

TABLE I: Coupling token-level KV compression (**V1**) with block-level physical eviction (**V2**) degrades performance

Metric	**V1**	**V2**	**△ (V2–V1)**
Total generation time (s)	5.570	6.269	+0.699 (+12.6%)
Time per token (ms)	27.8	31.3	+3.5 (+12.6%)
Throughput (tok/s)	35.91	31.90	-4.01 (-11.2%)
Speedup vs. baseline (x)	0.98	0.87	-0.11
Peak CPU RSS (MB, main proc)	85.2	144.6	+59.4 (+69.7%)
Peak CPU VMS (MB, main proc)	82.2	142.4	+60.2 (+73.2%)
Output length (chars)	886	886	0
Output match vs. baseline	No	No	—

the deterministic block-table rewrite (static-shape invariant) and amortizes eviction overhead. Accordingly, we rely on the explicit `StreamingVideoKVCacheManager` slice–free–rewrite loop (Listing 1) as the primary Phase 2 mechanism, and treat compression-driven eviction as an instructive negative result.

Listing 1: Phase 2 Physical Eviction Pseudocode

```
function apply_kv_cache_compression(request):
  if request.length < MAX_TOKENS: return

  # 1. Identify Topology
  sink_blocks = request.block_table[:NUM_SINK]
  middle_blocks = request.block_table[NUM_SINK:-
      NUM_RECENT]
  recent_blocks = request.block_table[-NUM_RECENT:]

  # 2. Physical Eviction (Reclaim HBM)
  GLOBAL_POOL.free(middle_blocks)

  # 3. Static Shape Reconstruction
  request.block_table = sink_blocks + recent_blocks
```

C. Phase 3: Hybrid Attention

We implemented a `HybridSSMAdapter` as a PyTorch module that replicates the Mamba 1.0 architecture [2]. The adapter functions as the bridge between the vLLM engine and Mamba:

- **Input Projection:** Expands the hidden dimension
- **Conv1d:** Applies a short temporal convolution
- **Selective Scan:** Performs the core recurrence mechanism
- **Output Projection:** Projects back to the hidden dimension

Crucially, the adapter implements the `AttentionLayerBase` interface, allowing it to interact directly with vLLM's memory manager while utilizing a `MambaSpec` to request fixed state buffers rather than growing KV blocks.

A major system challenge was vLLM's Chunked Prefill. In a Transformer, calculating attention for chunk k requires reading the KV cache of previous chunks. In an SSM, calculating chunk k requires the exact numerical value of the hidden state h at the end of chunk $k-1$. Because vLLM splits large prefills into smaller batches to minimize time to first token (TTFT) [1], we implemented a state-passing mechanism. The inference engine materializes the SSM state at the boundary of every chunk, stores it in HBM, and reloads it when the next chunk is scheduled. This required specific logic to handle `cu_seqlens` (cumulative sequence lengths) to ensure states are correctly reset or passed when batches contain multiple heterogeneous sequences.

We introduced `HybridAttentionBackend`, a custom attention backend that orchestrates the parallel execution:

1) Delegates the sliding-window attention computation to vLLM's optimized `TritonAttentionImpl`
2) Delegates the SSM history computation to `HybridSSMAdapter`
3) Fuses the results by summing the outputs: `output += ssm_out`

To prevent memory fragmentation, we utilized a `mamba_page_size_padded` configuration. This ensures that the memory allocated for continuous Mamba states aligns with the page boundaries of the Paged KV cache, preventing the block manager from creating gaps or illegal memory accesses.

We enabled this architecture for Llama 3.2 (`HybridLlama`) [4] and Qwen 2.5 VL (`HybridQwen2_5_VL`) [5]. Users can enable the feature via the `use_hybrid_attention` configuration flag.

The core of our history branch is the `HybridSSMAdapter`. Below is a simplified snippet of its implementation showing the integration with vLLM's attention metadata:

Listing 2: HybridSSMAdapter implementation

```
class HybridSSMAdapter(nn.Module, AttentionLayerBase
    ):

    def forward_history_branch_prefill(self,
        hidden_states, attn_metadata):

        # Integration with Mamba 1.0 kernels
        if self.ssm_mode == "disabled":
            return torch.zeros_like(hidden_states)

        # 1. Convolution
        x_conv = causal_conv1d_fn(
            x_t, self.conv1d.weight, self.conv1d.
                bias,
            conv_states=conv_state,
            activation="silu"
        )

        # 2. SSM Scan (Selective State Space)
        # Handles state passing for chunked prefill
        y = selective_scan_fn(
            x, ssm_state, dt, self.A, B, C, self.D,
                z,
            self.dt_proj.bias, delta_softplus=True
        )

        return self.out_proj(y)
```

D. Phase 4: Client-Side Stream Orchestration

While our `StreamingVideoKVCacheManager` enforces memory limits at the server level, we developed a complementary client-side driver to optimize request latency using prefix caching. This two-layer architecture ensures that the application layer actively cooperates with the inference engine to minimize redundant computation. This was discovered as we ran into CUDA out-of-memory errors on large videos.

The client maintains two local frame buffers: a fixed set of *Sink Frames* (S) representing the video start, and a sliding window of *Recent Frames* (R). As the video streams, the client orchestrates the prompt construction through two phases:

1) **Accumulate Phase:** New frames are appended to R. The input prompt is $P_t = [S, R_{old}, R_{new}]$. Since $[S, R_{old}]$ is identical to the previous request P_{t-1}, vLLM's prefix caching mechanism detects the shared hash and reuses the existing KV cache, computing attention only for R_{new}.
2) **Shift Phase:** When $|R|$ exceeds the window size W, the oldest frames are evicted ($R \leftarrow R[-W:]$). This creates a partial cache miss, necessitating a re-computation of the modified window, but ensures the context length remains bounded.

This client-side logic ensures that the prompt length sent to the server never exceeds $N_{sink} + N_{window}$, effectively clamping the memory footprint to $O(1)$ regardless of the stream duration, while Prefix Caching minimizes the latency of the accumulation steps.

V. Evaluation

Our evaluation focuses on three critical dimensions: memory efficiency, computational throughput and latency, and temporal stability.

A. Quantitative Analysis

1) Memory Efficiency: We first validated the impact of our Physical Eviction strategy (Phase 2) on system scalability when processing a 9s video ("baby reading" example in vLLM repository). As shown in Table II, enforcing a static block limit reduces per-stream footprints by **89.4%** and enables a **9.5x** increase in concurrent stream capacity.

TABLE II: Per-Stream KV Cache Footprint (Excluding Weights)

Metric	Baseline (Full)	Phase 2 (Eviction)	Delta
Cache/Stream	223.1 MB	**23.6 MB**	**-89.4%**
Max Concurrency	968 Streams	**9,150 Streams**	**9.5x**

We further compared the full Hybrid approach (Phase 3) against a Standard Attention baseline on `Llama-3.2-1B` (64 frames). The Hybrid architecture demonstrates a flat memory profile (~0 MiB/frame growth) after the sliding window fills, whereas Standard Attention exhibits linear growth until OOM (Table III).

TABLE III: Standard vs. Hybrid Attention (Llama-3.2-1B, 64 Frames)

Metric	Standard	Hybrid	Delta
Peak Memory	18.4 GB	**8.2 GB**	**-55%**
Growth Rate	120 MiB/frame	**~0 MiB/frame**	**Flat**

2) Throughput & Latency: A common concern with State Space Models (SSMs) is the potential computational overhead introduced by recurrent state updates. To address this, we benchmarked `Qwen2.5-VL-3B-Instruct` across varying context lengths and durations. In particular, `video_benchmark.py` measures end-to-end latency, token throughput, and generation stability. This benchmark isolates the computational overhead of the attention mechanism by running controlled inference loops on `Qwen2.5-VL-3B-Instruct`.

It is important to note that for these evaluations, the SSM parameters were randomly initialized. This approach was chosen to validate the architecture and computational overhead within the bounds of our time and resource constraints. The `HybridQwen2ForCausalLM` implementation relies on the standard Qwen2 weight loading mechanism, which does not include SSM-specific parameters (such as the state transition matrices A, D, and the convolutional weights). Consequently, these parameters retain their initial random values during inference, ensuring that the performance metrics reflect the cost of the SSM computation path rather than the quality of the generated text.

Our results indicate that Hybrid Attention achieves parity with Standard Attention. Namely, at 48 frames (5 iterations), Hybrid Attention maintained a throughput of 4027 tokens/s against a throughput of 4011 tokens/s for the baseline (Fig. 1).

This suggests that the O(1) state management of the Hybrid approach effectively neutralizes the overhead of the SSM recurrence, delivering the memory benefits of sparse attention without the typical latency penalty.

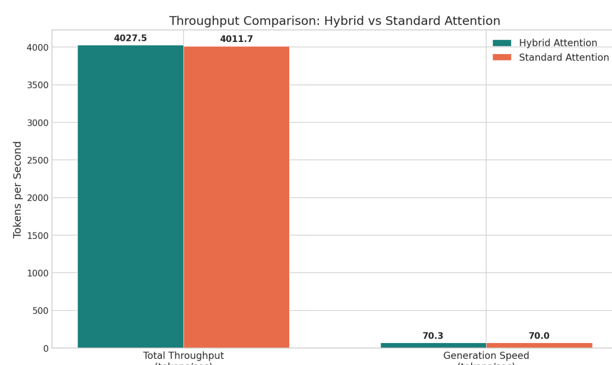

Fig. 1: Throughput comparison at 48 frames (5 iterations). Hybrid Attention (4027 tok/s) nearly matches the Standard Attention baseline (4011 tok/s), demonstrating little to no compute overhead.

3) Long-Horizon Stability: To verify stability for long-running video, we executed a 300-iteration stress test.

As illustrated in Figure 2, the Hybrid model exhibits no significant cumulative latency drift. The average latency difference over 300 iterations was less than **0.4%** (1.87s vs 1.86s).

To further investigate the jitter characteristics, we analyzed the latency distribution for both short-term (50 iterations) and

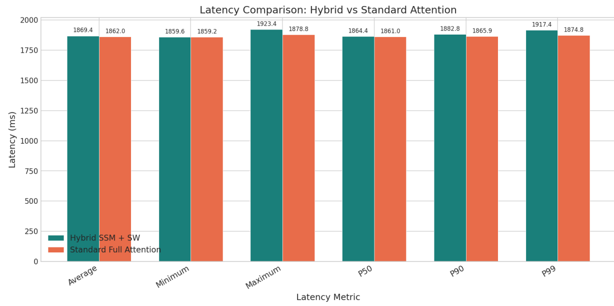

Fig. 2: Latency stability over 300 iterations. The Hybrid model tracks the Standard model closely, with no cumulative drift.

long-term (300 iterations) scenarios.

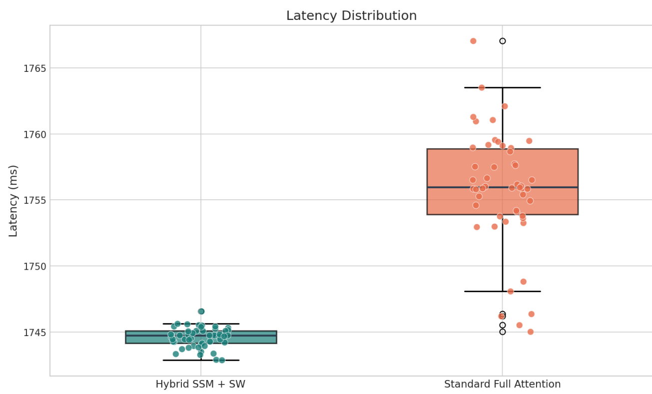

Fig. 3: Latency distribution (50 iters).

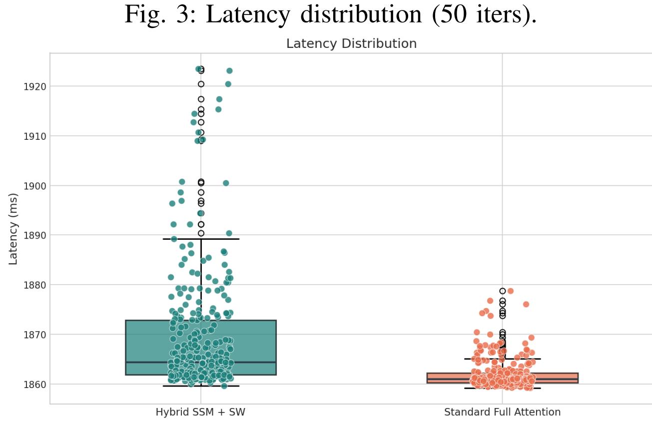

Fig. 4: Latency distribution (300 iters).

Figures 3 and 4 visualize the latency distributions using box plots. In the short-term (16-frame, 50-iteration) scenario, both models show near-identical tight distributions. In the long-term (48-frame, 300-iteration) stress test, the Standard Attention model remains highly stable (IQR $<$ 5ms). The Hybrid model maintains a comparable median latency (P50: 1.86s vs 1.86s) but exhibits a slightly wider tail (P99: 1.92s). This minor increase in jitter is a known characteristic of the state transition overhead in hybrid architectures but remains well within acceptable limits for real-time applications, especially given the 55% memory reduction. These results suggest the viability of Hybrid Attention for long-running video agents.

B. Qualitative Analysis on Short-form Video

We tested the model's ability to retain information using a "baby reading" video (approx. 9s duration). The model was prompted with: *"Describe what happens in this video in detail."* The following comparison highlights the impact of compression on generation quality.

Control (Full Cache):

> *The video depicts a young child sitting on a bed, wearing glasses and holding a book. The child appears to be engaged in reading, as they turn the pages of the book with their hands. The bed is made up with a white comforter and a few blankets, and there is a wooden headboard visible in the background. The child's attire consists of a light blue shirt and pink pants. The setting appears to be a cozy, home environment, with a few items of furniture and decor visible in the background. The video captures the child's focused expression and the act of reading, highlighting the innocence and curiosity of childhood.*

Treatment (KV Cache Compression):

> *The video depicts a young child sitting on a bed, wearing glasses and holding a book. The child appears to be engaged in reading, as indicated by their focused gaze and the way they hold the book. The child is wearing glasses, which adds a touch of cuteness to the scene. The child is seated on a bed, holding a book in their lap. The book appears to be a standard-sized book, possibly a children's book or a storybook. ... The setting seems to be a bedroom, with a bed and some furniture visible in the background. The child is wearing glasses and appears to be engaged in some activity, possibly reading a book. The setting seems to be a*

The Control model provides a fluid, cohesive narrative with specific background details ("wooden headboard", "light blue shirt"). The Treatment model successfully retains the critical semantic elements—"child", "glasses", "bed", "book"—but exhibits some repetition (mentioning the glasses multiple times) and consistently fails to output an end-of-sequence token. This suggests that while the hybrid architecture successfully preserves the semantic substance of the video, the eviction of exact history degrades the cues required for the model to recognize narrative closure.

C. Qualitative Analysis on Longer-form Video

To evaluate the system's performance on longer, high-entropy content, we tested the client-side rolling window strategy (Phase 4) on a fast-paced gameplay video (*Fortnite*, approx. 17s duration). Again using `Qwen2.5-VL-3B-Instruct` The model was prompted with: *"Describe what happens in this video in detail."*. We compared two configurations to analyze the impact of window size on generation quality and latency.

Scenario 1: 20-Frame Rolling Window

- **Configuration:** 32k Context Window, 20-frame sliding buffer
- **Total Streaming Time:** 130.06s
- **Model Output:** *"In the video, a player is standing on a wooden platform near the water. The player is holding a weapon and appears to be preparing to shoot. The platform is elevated above the water, and the player is facing towards the water. The player's character is visible in the top left corner of the screen."*
- **Observation:** The model successfully identified key spatial elements ("wooden platform", "water", "top left corner") but limited the narrative to the immediate visual buffer.

Scenario 2: 100-Frame Rolling Window

- **Configuration:** 128k Context Window, 100-frame sliding buffer
- **Total Streaming Time:** 456.36s
- **Model Output:** *"In the video, a player is seen standing on a wooden platform with a gun in their hand. They are aiming the gun and shooting it at a building. The player is wearing a backpack and is surrounded by a large crowd of people. The player continues to shoot the building while the crowd watches."*
- **Observation:** The larger window allowed for action continuity ("shooting at a building"), but introduced significant latency (3.5x slower). Notably, the model began to hallucinate details not present in the sparse frames ("large crowd of people"), likely due to the attention head struggling with the disconnected temporal jumps in the larger rolling buffer.

VI. Discussion

The transition from inference on finite-context video to infinite-context streaming requires rearchitecting memory management in Large Multimodal Models. Our results suggest that the bottleneck for long-form video understanding is not the model's ability to reason, but the inference engine's ability to manage state. By decoupling recent high-fidelity visual data from historical context, we demonstrate that infinite streaming is achievable on commodity hardware.

Our failure analysis of Phase 1 (Logical Masking) and Phase 2 (Physical Eviction) suggests logical masking is insufficient for production environments, as the physical allocator and JIT compiler (TorchDynamo/CUDA Graphs) must be synchronized. The regression in latency observed in combining Phase 1 and Phase 2 features (Table I, +12.6% generation time) highlights the hidden costs of memory management. That is, the overhead of constantly freeing and rewriting block tables can outweigh the benefits of smaller caches if not implemented as a fused kernel. The Hybrid approach (Phase 3) succeeds because it shifts the burden from the allocator (slow, CPU-bound) to the compute kernel (fast, GPU-bound).

The most significant finding of this study is the validation of the $O(1)$ memory growth hypothesis without incurring the expected latency penalties associated with recurrent state updates. As shown in Table III, the Hybrid Architecture reduces peak memory usage by 55% compared to standard attention while maintaining a flat growth rate. This flat profile is the prerequisite for "infinite" streaming; without it, systems may crash due to an OOM error.

Crucially, this memory efficiency does not come at the cost of speed or throughput. Our throughput analysis (Fig. 1) reveals that the overhead of the SSM scan and convolution operations is effectively negligible. This result may be attributed to the reduction in memory bandwidth pressure. By avoiding the fetch of gigabytes of KV cache from HBM, the system avoids the memory-bound regime that typically throttles large batch inference.

While the system solves the mechanical problem of streaming, the qualitative analysis reveals limits in semantic retention. The "Baby Reading" video evaluation demonstrates that while the Hybrid model retains high-level concepts ("child," "glasses"), it suffers from detail erosion (loss of background specifics) and repetition. Further, the model consistently failed to output an end-of-sequence token. This suggests that the compressed SSM state may lack the high-frequency distinctiveness required to prevent the decoding loop from falling into repetitive modes. Further, the eviction of exact history may degrade the context required for the model to recognize narrative closure.

The "Fortnite" scenario demonstrates the efficacy of our client-side prefix caching strategy in facilitating long-form video understanding. By leveraging the KV cache of the static sink and shared history, the system successfully maintained narrative continuity over an extended stream. However, this capability comes with a caveat regarding semantic precision. In Scenario 2, while the larger window allowed for better action tracking, it occasionally resulted in hallucination (e.g., the "large crowd of people"). We hypothesize that when the attention window spans a large temporal distance with sparse visual cues, the model may struggle to resolve causal dependencies, filling gaps with confabulation. Thus, while prefix caching effectively solves the computational bottleneck of long videos, the window size W remains a critical hyperparameter that must be tuned to balance context length with narrative coherence.

VII. Conclusion

In this work, we presented a hybrid attention architecture integrated into vLLM that enables inference on video streams for Large Multimodal Models. By fusing a sliding window of recent tokens with a Mamba-style State Space Model for history, we achieved a system that maintains $O(1)$ memory growth.

Our evaluation confirms that this architecture solves the catastrophic memory scaling of standard VLMs, reducing peak usage by over 50% and enabling stable, drift-free inference over thousands of tokens (Fig. 2). However, we also identified the semantic limits of this approach: while the system never crashes, the quality of the historical recall degrades into a generalized semantic summary rather than a photographic record.

Future work must address the gap between "memory capacity" and "memory fidelity." Potential directions include training the SSM branch end-to-end rather than using it as an adapter to qualitatively analyze the output of hybrid attention-based models. We also see value in developing dynamic policies for the client-side orchestrator that can expand or contract the sliding window based on the entropy of the current scene. Ultimately, this work seeks to move the field one step closer to models that can watch, understand, and react to the world without bound.

Acknowledgment

We thank Dr. Jason Mars (University of Michigan, Ann Arbor) for his guidance throughout this project and effort in introducing us to the field of efficient AI systems. We also thank the vLLM open-source community for their robust repository and responsiveness on public forums.

References

[1] W. Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proc. 29th Symp. Operating Syst. Principles (SOSP)*, 2023.

[2] A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[3] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient Streaming Language Models with Attention Sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[4] AI@Meta, "The Llama 3 Herd of Models," *arXiv preprint arXiv:2407.21783*, 2024.

[5] Qwen Team, "Qwen2-VL: To See the World More Clearly," *arXiv preprint arXiv:2408.16700*, 2024.

[6] O. Lieber, B. Lenz, H. Hofstatter, et al., "Jamba: A Hybrid Transformer-Mamba Language Model," *arXiv preprint arXiv:2403.19887*, 2024.

[7] P. Glorioso et al., "Zamba: A Compact 7B SSM-Hybrid Model," *arXiv preprint arXiv:2405.16712*, 2024.

[8] K. Li et al., "VideoMamba: State Space Model for Efficient Video Understanding," *arXiv preprint arXiv:2403.06977*, 2024.

[9] A. Hatamizadeh and J. Kautz, "MambaVision: A Hybrid Mamba-Transformer Vision Backbone," in *Proc. CVPR*, 2024.

[10] T. Dao and A. Gu, "Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality," *arXiv preprint arXiv:2405.21060*, 2024.

[11] L. Fu et al., "Video-MME: The First-Ever Comprehensive Evaluation Benchmark of Multi-modal LLMs in Video Analysis," *arXiv preprint arXiv:2405.21075*, 2024.

[12] Y. Mangalam et al., "EgoSchema: A Diagnostic Benchmark for Very Long-Form Video Language Understanding," in *Proc. NeurIPS*, 2024.

[13] M. M. Moradi, W. Ahmed, S. Wen, S. Mudur, W. Zhang, and Y. Liu, "Balancing Computation Load and Representation Expressivity in Parallel Hybrid Neural Networks," *arXiv preprint arXiv:2505.19472*, 2025.