StreamingTOM: Streaming Token Compression for Efficient Video Understanding

Supplementary Material

A. Implementation Details

A.1. Terminology Clarification

Throughout this paper, *Pre-LLM* and *Post-LLM* refer to **pipeline location**, not inference stage. **Pre-LLM** denotes compression applied after the vision encoder and projector but before visual tokens enter the LLM transformer layers. **Post-LLM** denotes memory management applied after kv-cache formation within the LLM. Both operations occur during the prefill phase; the distinction captures *where* in the pipeline compression acts, not *when* during inference.

A.2. Streaming Attention and Token Saliency

CTR necessitates a saliency score for each visual token derived from the vision encoder. Rather than materializing full attention maps, we implement a streaming self-attention kernel inspired by memory-efficient chunked attention [9], integrated into the SigLIP encoder [68] to compute both the attention output and per-token importance scores simultaneously. Given queries Q, keys K, values V, and an optional mask M, the kernel processes keys in small chunks through two passes over the sequence. The first pass accumulates a numerically stable log-sum-exp of the attention scores along the key dimension, while the second pass recomputes the scores to obtain normalized probabilities. These probabilities are subsequently aggregated into the final output and an importance accumulator. Algorithm 1 summarizes this procedure.

Here H denotes the number of attention heads and L_q the number of query positions in the self-attention module. We enable this streaming kernel exclusively on the final layer of the vision encoder to expose the resulting importance scores as an additional one-dimensional tensor over visual tokens. CTR consumes these scores as attention-based saliency signals to rank, select, and compress visual tokens in a strictly streaming fashion without revisiting past frames.

Algorithm 1 Streaming attention with importance scores

Require: Query Q, keys K, values V, optional mask M, scale α, chunk size C
Ensure: Attention output O, importance scores s

```
Convert Q, K, V to float32 and initialize attn_max, attn_lse, O, imp
for each key chunk [s : e) of size at most C do
    Compute scores S = αQK^⊤_{s:e}
    if M is provided then
        Add corresponding mask slice: S ← S + M_{:,:,:,s:e}
    end if
    Update attn_max and attn_lse using log-sum-exp over the key dimension
end for
for each key chunk [s : e) do
    Recompute S = αQK^⊤_{s:e}
    if M is provided then
        Add corresponding mask slice: S ← S + M_{:,:,:,s:e}
    end if
    Compute probabilities P from attn_max and attn_lse
    O ← O + PV_{s:e}
    imp_{s:e} ← imp_{s:e} + Σ_{h,q} P
end for
s ← imp/(H · L_q)
return O, s
```

A.3. Text–Video Retrieval

Layer-wise vision memory. During video encoding, we extract key tensors $K^{(l)} \in \mathbb{R}^{H_k \times T^{(l)} \times d}$ from every transformer block prior to applying rotary embeddings to ensure they contain exclusively visual tokens. For each contiguous group of G visual tokens, we compute a summary key

$$k_t^{(l)} = \frac{1}{G} \sum_{i \in \mathcal{G}_t} K_i^{(l)}, \qquad t = 1, \ldots, \frac{T^{(l)}}{G},$$

and store the resulting sequence $\{k_t^{(l)}\}$ alongside the exact KV cache for that layer. This process yields a layer-aligned memory bank that preserves the original attention-head dimensionality while maintaining the chronological order of visual segments.

Question-guided single-pass retrieval. Upon receiving a question, we exclude vision placeholders and prompts to process the clean text through the language model in a single pass. At transformer layer l, we map the resulting query tensor $Q^{(l)} \in \mathbb{R}^{H_q \times T_q \times d}$ to the key–value head space by averaging query heads within each GQA group. Subsequently, we average across the token dimension to obtain a layer-specific question embedding

$$q^{(l)} = \frac{1}{T_q} \sum_{j=1}^{T_q} Q_{:,j,:}^{(l)} \in \mathbb{R}^{H_k \times d}.$$

Before retrieval, we flatten both $q^{(l)}$ and each $k_t^{(l)}$ into vectors, ℓ_2-normalize them, and perform a cosine-similarity

top-K search within that layer. We determine the number of retrieved segments via

$$K = \min\Big(\Big\lceil \frac{B}{G} \Big\rceil,\ \frac{T^{(l)}}{G}\Big),$$

where B represents the global token budget and $T^{(l)}$ denotes the total number of visual tokens stored at layer l. We maintain the selected indices in sorted order to ensure the temporal coherence of the retrieved segments.

Cache reconstruction for decoding. Following this single forward pass, each layer identifies the necessary visual groups. We retrieve the corresponding KV slices and system prompt tokens from the memory bank to assemble a `DynamicCache`, reusing these per-layer indices throughout the decoding process. Consequently, the system incurs the question–video matching cost only once per question while permitting each layer to independently select the video segments contributing to its attention computation.

A.4. RVS Streaming Evaluation

While most benchmarks employ a conventional offline evaluation via the lmms-eval framework [70] where one video–question pair is processed at a time with full clip access, RVS operates under a strict streaming protocol. We sort all questions for a given video by their end timestamps. With frames sampled at a fixed rate such as 0.5 fps, the model processes the video linearly and observes the content only once. Given a question ending at time t, we encode only the new frames arriving between the last encoding step and time t to update the streaming KV cache without re-encoding earlier frames. Should a subsequent question fall within the timeframe of the previously encoded frames, we answer it using the existing cache and pass an empty image batch to indicate no additional visual input. This protocol ensures that the model generates predictions solely from video content observed up to the question time, highlighting the streaming nature of our method on RVS.

B. Additional Results

B.1. LongVideoBench

Table 5. Results on LongVideoBench.

Method	Setting	Accuracy
LLaVA-OV-7B	32 frames	56.4
+StreamingTOM (ours)	0.5 FPS	56.3

We evaluate StreamingTOM on LongVideoBench [57], which spans a wide duration range (8s–60min). StreamingTOM retains 99.8% of the baseline accuracy (56.3 vs. 56.4). The marginal gap is expected, as LongVideoBench's duration distribution skews shorter than VideoMME-Long and MLVU, where StreamingTOM's streaming temporal coverage provides the greatest advantage (Table 1).

B.2. Generalization to Other VLMs

To validate backbone-agnostic generalization, we apply StreamingTOM to Qwen2.5-VL-7B-Instruct [3] on VideoMME. The core CTR and OQM mechanisms require no modification; only minor adjustments to the input format and 3D rotary position encoding are needed.

Table 6. Generalization results on Qwen2.5-VL-7B-Instruct.

Method	Setting	VideoMME				Retain
		Short	Medium	Long	Avg	
Qwen2.5-VL-7B	32 frames	72.4	61.3	50.3	61.3	100%
+StreamingTOM	32 frames	69.3	57.2	50.1	58.9	96.1%
+StreamingTOM	0.5 FPS	69.9	65.2	54.0	63.0	102.8%

At 32 frames, StreamingTOM retains 96.1% of baseline accuracy with substantially fewer tokens. Under streaming settings (0.5 FPS), temporal coverage boosts retention to 102.8%, consistent with our LLaVA-OV-7B findings (Table 1), confirming backbone-agnostic generalization without modifying the core pipeline.

C. Discussion

C.1. Adaptivity of StreamingTOM

StreamingTOM capitalizes on the observation that long video sequences possess strong temporal locality and redundancy where nearby frames share content and few tokens drive downstream reasoning. CTR and OQM harness this structure within a strictly training-free paradigm. Since both modules operate exclusively on token representations and KV caches without assuming specific architectural details beyond a patch-based vision encoder and a transformer-style language model, StreamingTOM seamlessly integrates with diverse video LLMs. This compatibility applies to both offline and streaming settings and requires no modification to the original training procedure. The configurable per-frame token budget and global KV-cache budget serve as precise control mechanisms to balance accuracy, latency, and memory usage across varied deployment scenarios.

C.2. Deployment Scenarios

StreamingTOM's bounded memory growth and predictable latency make it applicable to several practical streaming scenarios. **(1) Long-duration monitoring.** In applications such as surveillance and meeting assistants, the kv-cache remains bounded regardless of stream length (e.g., 1.2 GB for one hour of video), enabling continuous operation on a single GPU without memory overflow. **(2)**

Latency-sensitive interaction. For embodied AI and autonomous driving assistants, sub-0.4s TTFT (Table 3 in the main paper) supports real-time querying over live video streams. **(3) Resource-constrained deployment.** The fixed per-frame budget G ensures predictable compute and memory consumption per frame, facilitating capacity planning on edge devices with limited resources. Being training-free and backbone-agnostic, StreamingTOM can be integrated into existing VLM pipelines without retraining, lowering the barrier for practical adoption across diverse deployment environments.

D. Limitations and Future Work

Despite the compression and efficiency gains achieved in a training-free setting, the underlying multimodal backbone fundamentally constrains the overall performance of StreamingTOM. Compression or memory management algorithms alone cannot resolve intrinsic limitations regarding fine-grained perception or long-range reasoning. Adopting stronger or task-adapted backbones would directly raise the performance ceiling, making the joint tuning of the backbone with CTR and OQM a logical extension. Furthermore, while our design optimizes prefill and bounds the active KV cache, the vision encoder remains the dominant latency source. Consequently, dense feature extraction in the visual front-end limits end-to-end acceleration even when the language model executes with high efficiency.

Several avenues exist to extend StreamingTOM and address these limitations. One path moves beyond purely training-free compression to explore lightweight tuning of CTR and OQM alongside the backbone by learning content-aware token budgets or memory allocation policies atop a frozen or partially fine-tuned video LLM. Simultaneously, reducing the computational cost of the vision encoder warrants investigation. This could involve introducing lightweight pre-encoder modules for early downsampling and motion-aware frame selection or integrating encoder-side sparsity and quantization techniques. Integrating such backbone improvements with StreamingTOM offers the potential to further minimize the end-to-end cost of long-form video understanding while preserving robust performance.