CACHE LOCALLY, RECOMPUTE SELECTIVELY FOR STREAMING VIDEO UNDERSTANDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Streaming video understanding (SVU) requires video-LLMs to process incoming video continuously while answering user queries with low latency. Recent methods reduce query-time cost by prefilling frames into KV caches during idle time and reusing the cached KVs at query time. To bound per-frame computation over unbounded video streams, these methods restrict the context available during prefill. This creates a fundamental context mismatch: the restricted context may omit information needed to properly contextualize video tokens, yet, still contain information irrelevant to the eventual query, which is unknown during prefill. We propose **FrameBlend**, a simple yet effective plug-and-play method to address this mismatch through selective KV recomputation, without requiring any additional training. At query time, FrameBlend re-prefills selected video tokens using query-relevant context and replaces their cached KVs with recomputed ones. Crucially, this recomputation enables shorter attention windows during idle-time prefill, substantially reducing prefill cost while improving accuracy. Experiments across three SVU frameworks and five streaming and offline benchmarks demonstrate accuracy gains of up to 3.9 pp. On RVS-Ego, FrameBlend reduces per-frame prefill FLOPs by over 20% and total processing time by 14–59% for retrieval-based frameworks, including the cost of query-time recomputation. These results demonstrate that a simple change in how contextualization is distributed between idle time and query time can improve the accuracy-efficiency trade-off for SVU.

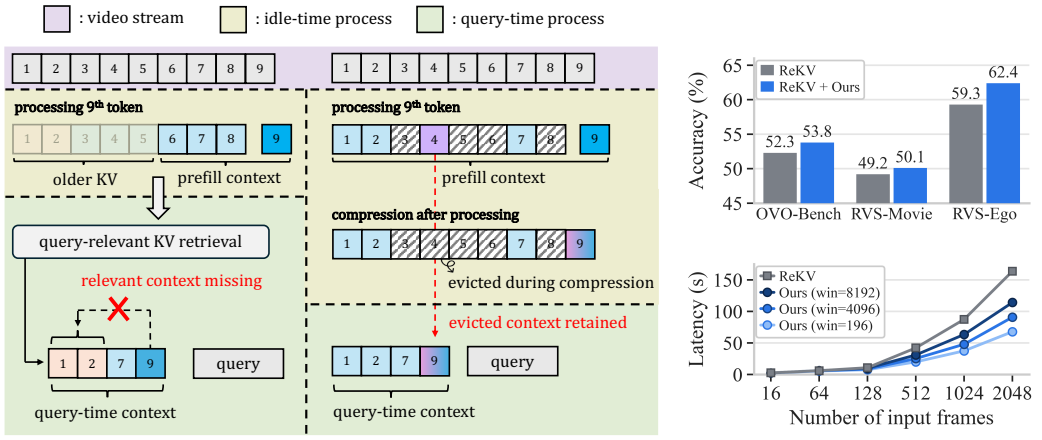
1 INTRODUCTION

Recent video large language models (video-LLMs) have demonstrated strong video understanding capabilities (Li et al., 2024; Bai et al., 2025a;b). These advances, together with growing demand for real-time applications such as robotics (Mu et al., 2023), autonomous driving (Shao et al., 2024), and surveillance systems (Rao et al., 2022), have spurred interest in streaming video understanding (SVU). In SVU, a video-LLM continuously processes an incoming video stream and answers user queries as they arrive. This setting imposes three constraints. First, during *idle time*, i.e., no query is pending, the model must keep pace with a potentially unbounded stream under limited compute and memory. Second, because future queries are unknown during idle-time processing, it is difficult to decide in advance which information to preserve. Third, at *query time*, the model must respond with minimal latency, leaving little room for additional computation before answering.

To meet these requirements, recent methods prefill incoming frames into key-value (KV) caches during idle time and directly reuse the cached KVs at query time (Di et al., 2025; Ning et al., 2026; Zhang et al., 2026). However, letting every incoming token attend to all preceding tokens incurs a cumulative attention cost that grows quadratically with stream length, which is prohibitive for long streams. Existing methods therefore limit the prefill context, either through sliding-window attention, which restricts attention to recent frames (Di et al., 2025; Chen et al., 2026a;b), or through KV cache compression, which condenses accumulated context into a fixed-size memory (Kim et al., 2025; Ning et al., 2026; Zhang et al., 2026).

Both strategies can create a *mismatch* between the context used to construct KVs during idle-time prefill and the context used at query time. With sliding-window attention, a token cannot attend to earlier tokens outside its window, even if those tokens are later retrieved together with it to answer a query. With KV cache compression, information from tokens that are later compressed or evicted

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) sliding-window attention (b) KV cache compression (c) accuracy and efficiency

Figure 1: **Motivation and performance of FrameBlend.** (a) Sliding-window attention can omit inter-token context needed at query time. (b) KV compression can retain unnecessary context from evicted tokens. (c) FrameBlend improves SVU framework accuracy while reducing total latency.

may remain encoded in retained KVs, even when it is irrelevant to the query (Pang et al., 2026). Reused KVs may thus both omit relevant context and retain irrelevant context. Direct KV reuse leaves these representations uncorrected, potentially degrading answer quality (Yao et al., 2025).

To resolve this mismatch, we propose **FrameBlend**, a training-free method that selectively recomputes cached KVs at query time. Inspired by recomputation techniques in LLM serving (Yao et al., 2025; Hu et al., 2025), FrameBlend re-prefills a selected subset of tokens under the query-time context and overwrites their cached KVs. Whereas prior methods either rely on costly operations to identify recomputation targets or select them in a query-agnostic manner, FrameBlend uses QTS (query-relevant token-level selection), a lightweight, frequency-based mechanism guided by query relevance. Moreover, in SVU, query-time recomputation fundamentally changes how idle-time prefill should be designed; rather than using large attention windows to preserve context in advance, FrameBlend instead uses shorter windows, as missing context can be recovered once the query arrives. This not only reduces prefill cost but can also improve accuracy over longer windows. We attribute this gain to reduced contextual interference: since future queries are unknown at prefill time, shorter windows limit how much query-irrelevant information is baked into the cached KVs. Together, *shorter attention windows* during idle-time prefill and *query-time KV recomputation* form the core principle of FrameBlend, improving both efficiency and accuracy without training, and with minimal modifications to existing SVU frameworks.

We evaluate FrameBlend on three SVU frameworks—ReKV (Di et al., 2025), HERMES (Zhang et al., 2026), and LiveVLM (Ning et al., 2026)—across two streaming benchmarks: RVS (Zhang et al., 2024) and OVO-Bench (Niu et al., 2025). We also assess general video understanding ability on three offline benchmarks (Fu et al., 2025; Zhou et al., 2025; Mangalam et al., 2023). FrameBlend matches or improves aggregate accuracy on every framework-benchmark pair, with gains of up to 3.2 pp on streaming benchmarks and 3.9 pp on offline benchmarks. For ReKV and LiveVLM, FrameBlend further reduces per-frame prefill FLOPs by 22.2–26.0% and prefill latency by 12.6–60.5% on RVS. Even after accounting for query-time recomputation, total FLOPs and latency decrease by 19.0–23.2% and 14.1–58.7%, respectively.

Our contributions are summarized as follows:

- We propose FrameBlend, a plug-and-play, training-free method that selectively recomputes KVs for relevant tokens at query time. To our knowledge, it is the first query-aware recomputation method for SVU.
- We show that query-time recomputation enables shorter attention windows during idle-time prefill, reducing prefill cost while matching or exceeding the accuracy of longer windows.
- Across ReKV, HERMES, and LiveVLM on two streaming and three offline benchmarks, FrameBlend consistently improves (or matches) aggregate accuracy for every framework–benchmark

pair, with gains of up to 3.2 pp and 3.9 pp, respectively. For ReKV and LiveVLM on RVS, it also reduces per-frame prefill FLOPs by 22.2–26.0% and total latency by 14.1–58.7%.

2 Related work

Streaming video understanding. The central challenge in SVU is efficiently storing and utilizing the information in potentially unbounded video streams, under memory and computation constraints. One line of work addresses this by maintaining an explicit memory of visual representations from incoming frames (Zhang et al., 2024; Qian et al., 2025; Wang et al., 2025; Zeng et al., 2025). However, these stored representations must still be processed by the backbone LLM to answer the incoming query, often leading to long query-time latency for LLM prefill computation.

Recently, a growing line of work instead prefills incoming frames into KV cache during idle time and directly reuses the cached KVs at query time, thereby shifting much of the prefill computation to idle time (Di et al., 2025; Chen et al., 2026b;a; Kim et al., 2025; Yang et al., 2025; Ning et al., 2026; Zhang et al., 2026). These approaches can be categorized according to how they manage the context during the prefill and decoding. During prefill, allowing each frame to attend to all preceding frames becomes prohibitively expensive as the stream grows. Therefore, some approaches restrict attention to recent frames using sliding-window attention (Di et al., 2025; Chen et al., 2026b;a), while others apply KV cache compression to reduce redundant tokens (Kim et al., 2025; Yang et al., 2025; Ning et al., 2026; Zhang et al., 2026). During decoding, retrieval-based methods independently select query-relevant KVs at each layer based on their similarity to the query (Di et al., 2025; Chen et al., 2026a; Ning et al., 2026). In contrast, non-retrieval methods use the entire bounded KV memory, which has been progressively compressed during prefill, to generate the final prediction at the query time (Kim et al., 2025; Yang et al., 2025; Zhang et al., 2026).

These works directly reuse the stored KVs for decoding, which can create a context mismatch between prefill and decoding. Our work mitigates this mismatch by selectively recomputing cached KVs with query-relevant context at query time.

Selective recomputation of KV cache. The idea of recomputing the cache for resolving context mismatch has been studied recently in the field of long-context LLMs. In these scenarios, the primary goal is to approximate the behavior of full-context KVs, by recomputing only a fraction of the KVs that have been previously prefilled by computing in short chunks. For example, CacheBlend dynamically selects and recomputes the KVs for tokens that deviate large by such recomputations (Yao et al., 2025). EPIC instead statically recomputes the first few tokens of each cached chunk (Hu et al., 2025). These ideas have been extended to vision-language model serving, where MPIC recomputes the first few image tokens (Zhao et al., 2025), while VLCache dynamically adjusts the recomputation ratio across layers (Qin et al., 2025).

These methods, however, are not specifically designed for SVU, where recomputation must be both query-aware and efficient under a tight query-time latency budget. Their recomputation targets are determined either by query-agnostic heuristics or by additional query-time operations. In contrast, FrameBlend selects query-relevant recomputation targets and leverages existing backbone retrieval results where applicable, thereby minimizing additional target-selection operations.

3 Preliminaries

In this section, we formally describe the problem of streaming video understanding and the idea of KV cache reuse. For the sake of simple exposition, we focus on describing the case of ReKV—a method with sliding-window prefill and decoding based on selective retrieval (Di et al., 2025).

Consider a standard streaming video question-answering task, where the model is given a stream of video frames. More concretely, define the *video token stream* as an unlimited sequence of tokens

$$\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots), \tag{1}$$

where the tokens are temporally ordered. That is, $\mathcal{X}$ consists of the tokenized version of each video frame, concatenated in the temporal order. The *user query* $\mathbf{q}$ may arrive in the middle of this stream, e.g., after the first N video tokens, asking a specific question about the video. Then, the video-LLM must generate an answer $\hat{y}$ using the observed token sequence $\mathcal{X}_{\leq N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$.

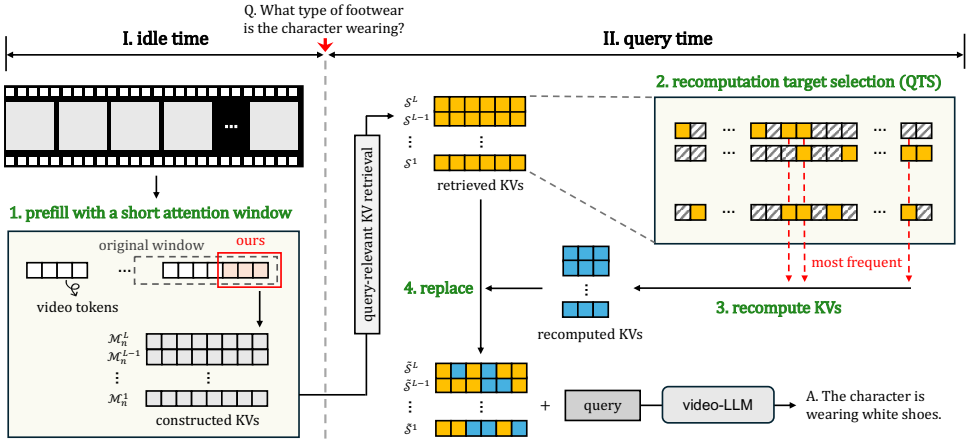


Figure 2: **Overview of FrameBlend.** FrameBlend uses a short attention window during idle-time prefill and selectively recomputes the KVs of query-relevant tokens at query time.

Because storing all tokens $\mathcal{X}_{\leq N}$ and processing them after the query arrives incurs too much latency and storage requirements, we consider KV cache reuse (Di et al., 2025; Ning et al., 2026; Zhang et al., 2026). Concretely, we divide the streaming inference into two phases: *idle time* and *query time*, corresponding to before and after the query arrives, respectively.

Idle time. During idle time, each incoming video token $\mathbf{x}_n$ is processed continually, forming the (layerwise) *memory* $\mathcal{M}_n^\ell$ for each transformer layer $\ell \in \{1, \ldots, L\}$. For standard sliding-window methods, e.g., ReKV (Di et al., 2025), this is simply done by prefilling w most recent tokens $(\mathbf{x}_{n-w+1}, \ldots, \mathbf{x}_n)$ to the video-LLM and storing the key-value (KV) cache $\mathbf{c}_n^\ell = (\mathbf{k}_n^\ell, \mathbf{v}_n^\ell)$ from each layer. That is, the memory will keep accumulating these KV caches as

$$\mathcal{M}_n^\ell = (\mathbf{c}_1^\ell, \mathbf{c}_2^\ell, \ldots, \mathbf{c}_n^\ell). \tag{2}$$

More recent methods employ mechanisms to continually update and compress the previously generated memory, keeping the total storage cost sublinear in n (Ning et al., 2026; Zhang et al., 2026).

Query time. When the query $\mathbf{q}$ arrives at time N, the model utilizes the current memory $\{\mathcal{M}_N^\ell\}_\ell$ to generate the response, rather than prefilling the entire observed sequence from scratch. To reduce the cost of loading the memory, one can first select a subset of KV caches $\mathcal{S}^\ell \subseteq \mathcal{M}_N^\ell$ for each layer ℓ, that is most relevant for answering the target query $\mathbf{q}$. This is typically done by applying some similarity metric $\rho(\cdot, \cdot)$ between the query and the keys of each KV, i.e.,

$$\mathcal{S}^\ell = \left\{ (\mathbf{k}_i^\ell, \mathbf{v}_i^\ell) \;\middle|\; \rho(\mathbf{k}_i^\ell, \mathbf{q}) \in \mathrm{ArgTopK}\left(\{\rho(\mathbf{k}_i^\ell, \mathbf{q})\}_{i=1}^N\right) \right\}. \tag{3}$$

For example, many prior works use the dot-product similarity between the stored key and the query representation derived from $\mathbf{q}$ (Di et al., 2025; Chen et al., 2026a; Ning et al., 2026). In contrast, the methods that compress-and-store the KV during idle time typically utilize the whole memory—i.e., $\mathcal{S}^\ell = \mathcal{M}_N^\ell$—as it is already compact (Kim et al., 2025; Yang et al., 2025; Zhang et al., 2026). Critically, we note that these KV subsets $\mathcal{S}^\ell$ do not need to correspond to an identical set of tokens across different layers; in fact, different layers commonly select different tokens.

4 Method

We propose **FrameBlend**, a simple yet effective plug-and-play method to reduce the prefill cost and improve the accuracy of SVU frameworks by selectively recomputing the KV cache at query time. We first formally describe how FrameBlend works (Section 4.1), and discuss how it allows us to use a shorter attention window (Section 4.2). By default, we focus on describing the case of applying on retrieval-based methods, e.g., ReKV (Di et al., 2025) or LiveVLM (Ning et al., 2026). Then, we additionally discuss how to adapt it for KV compression methods, e.g., Zhang et al. (2026).

4.1 Selective Recomputation of KV Cache

After the base SVU framework (e.g., ReKV) determines the layerwise KV sets $\{\mathcal{S}^\ell\}_{\ell=1}^L$ for answering the query $\mathbf{q}$, FrameBlend operates in two simple steps: (1) *Recomputation target selection:* We select a subset of the loaded KV sets to recompute; (2) *Recomputing and decoding:* We recompute the KVs and use them for decoding the answer. We describe each step below.

Step 1: Recomputation target selection. In this step, we select a subset of the retrieved KV cache $\mathcal{S}^\ell$ that will be recomputed, using the QTS (query-relevant token-level selection) criterion.

The design principle of QTS is twofold. First, we prioritize the most query-relevant KVs, because they will have the largest impacts on the final answer. Second, we select KVs in a way that recomputing can be done efficiently, in order to minimize the query-time latency. Typical recomputing methods for LLM serving, e.g., CacheBlend (Yao et al., 2025), are suboptimal in this sense; they focus on approximating the properties of full KV cache by recomputing KVs that get updated the most, gradually proceeding over the layers. This is not aligned well for SVU, being query-agnostic and requiring a tedious sequential selection procedure between transformer layers.

In short, QTS makes selection at the token-level, i.e., select KVs corresponding to the same tokens at every layer. This design is geared toward minimizing the latency, by enabling an allocation of constant computational budget over the layers and removing any inter-layer operations.

QTS selects the tokens that have been most frequently selected as $\mathcal{S}^\ell$ across all layers. More concretely, the set of *recomputation target tokens* is defined as

$$\mathcal{P} = \left\{\mathbf{x}_i \;\middle|\; \phi(\mathbf{x}_i) \in \mathrm{ArgTopK}\big(\{\phi(\mathbf{x}_i)\}_{i=1}^N\big)\right\}, \qquad \text{where} \quad \phi(\mathbf{x}_i) := \left|\left\{\ell : \mathbf{c}_i^\ell \in \mathcal{S}^\ell\right\}\right|. \tag{4}$$

Since this $\mathcal{P}$ is computed directly from the backbone framework's selections, it captures the cross-layer agreement on the query relevance of $\mathbf{x}_i$ without additional operations.

For frameworks without query-time retrieval, e.g., HERMES (Zhang et al., 2026), QTS simply applies the same similarity-based retrieval to the tokens in memory (Equation (3)), and chooses the highest-similarity tokens. The results of this retrieval only affect which tokens to recompute, and the backbone's original decoding context remains unchanged.

Step 2: Recomputing and decoding. The tokens in $\mathcal{P}$ are jointly re-prefilled through all LLM layers in temporal order, which requires a single forward pass over $|\mathcal{P}|$ video tokens. The recomputed KVs then replace the corresponding KVs in $\{\mathcal{S}^\ell\}_{\ell=1}^L$ to construct the final decoding context. Concretely, let $\hat{\mathbf{c}}_i^\ell$ denote the recomputed KV of token i at layer ℓ. The replacement is defined as

$$\tilde{\mathbf{c}}_i^\ell = \begin{cases} \hat{\mathbf{c}}_i^\ell, & \mathbf{x}_i \in \mathcal{P}, \\ \mathbf{c}_i^\ell, & \text{otherwise}, \end{cases} \qquad \tilde{\mathcal{S}}^\ell = \left\{\tilde{\mathbf{c}}_i^\ell \mid \mathbf{c}_i^\ell \in \mathcal{S}^\ell\right\}. \tag{5}$$

The answer is decoded using $\tilde{\mathcal{S}}^\ell$, following the same temporal ordering and positional encoding policy as the backbone SVU framework. Additionally, to avoid repeated vision-encoder computation during recomputation, FrameBlend retains each observed video token in CPU memory during idle-time prefill.

4.2 Prefilling with a Short Attention Window

With post-query recomputation, we revisit the amount of context required during prefill. Since recomputing provides inter-token context at the query time, FrameBlend may need less contextual information during the idle time. In other words, we may need a shorter window for the prefill.

Our experimental results confirm this intuition. In Figure 3, we report the accuracy of FrameBlend applied on ReKV with various window sizes, measured on the OVO-Bench for LLaVA-OV-7B. For simplicity, we have used $K = \infty$ for recomputing (i.e., recompute all retrieved cache).

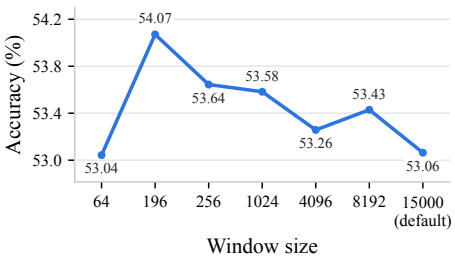


Figure 3: **Effect of window size.** Accuracy of ReKV with FrameBlend on OVO-Bench.

From Figure 3, we observe that reducing prefill windows can improve accuracy for FrameBlend, as short as 196 tokens—which correspond to a single video frame. This is in stark contrast with the observations of ReKV, where a longer window generally leads to better performance (Di et al., 2025). We attribute this gain to reduced contextual interference at prefill (Pang et al., 2026; Shen et al., 2026): since the future query is unknown, a longer context increases the risk of encoding unnecessary contextual information into the KVs. Results on additional benchmarks are provided in Appendix C.

FrameBlend thus reduces the attention window to 1–68% of the backbone's original window size, depending on the benchmark, reducing prefill compute and improving accuracy over longer ones.

5 Experiments

5.1 Experimental setup

Benchmarks. We evaluate FrameBlend on five streaming and offline video understanding benchmarks, covering both multiple-choice and open-ended question answering. For streaming evaluation, we use OVO-Bench (Niu et al., 2025) and RVS (Zhang et al., 2024). On RVS, following Zhang et al. (2026), we use GPT-3.5-turbo-0125 to evaluate open-ended answers. For offline evaluation, we use MLVU (Zhou et al., 2025), Video-MME (Fu et al., 2025), and EgoSchema (Mangalam et al., 2023). We use the official development set for MLVU, the full EgoSchema test set evaluated by the official evaluation server, and Video-MME without subtitles.

Base SVU methods. To evaluate FrameBlend as a plug-and-play method, we integrate it into three frameworks: ReKV (Di et al., 2025); LiveVLM (Ning et al., 2026); HERMES (Zhang et al., 2026). We retain the original configuration of each framework except for applying the query-time recomputation and the short prefill window introduced by FrameBlend.

Video-LLMs. We evaluate on two open-weight backbone video-LLMs: LLaVA-OneVision-7B (LLaVA-OV-7B) (Li et al., 2024) and Qwen2.5-VL-7B (Bai et al., 2025b).

Hyperparameters. All video inputs have been sampled at 0.5 FPS. We set the recomputation budget to $K = 7840$ video tokens, corresponding to 40 frames for LLaVA-OV-7B. For FrameBlend, we have tuned prefill window sizes for each framework-benchmark pair.

Hardware. All efficiency experiments have been conducted on a single NVIDIA A100 80GB GPU.

Omitted details. See Appendix A.

5.2 Main results

Streaming video understanding. As shown in Table 1, FrameBlend improves the understanding performance of the backbone frameworks. On LLaVA-OV-7B, it improves RVS-Ego accuracy by 1.3–3.1 pp, RVS-Movie accuracy by 0.2–1.8 pp, and OVO-Bench average accuracy by 0.2–1.5 pp across ReKV, HERMES, and LiveVLM. Similar improvements are observed on Qwen2.5-VL-7B, showing that FrameBlend consistently improves streaming performance across different backbone video-LLMs. These results demonstrate that FrameBlend is a simple yet effective method that can be applied to diverse SVU frameworks.

Offline video understanding. Following prior work (Di et al., 2025; Zhang et al., 2026), we further evaluate FrameBlend on offline benchmarks to demonstrate its general video understanding capability. As shown in Table 2, FrameBlend improves accuracy by 0.3–3.9 pp on MLVU and 0.4–2.9 pp on Video-MME across ReKV, HERMES, and LiveVLM. On EgoSchema, it maintains comparable performance or improves it, with changes ranging from -0.1 to $+1.6$ pp. Overall, these results demonstrate that FrameBlend improves the general video understanding capability of SVU frameworks beyond streaming settings.

5.3 Efficiency Analysis

Idle- and query-time efficiency. We evaluate the efficiency of FrameBlend on RVS-Ego by separately measuring idle-time and query-time costs. For idle-time efficiency, we report cost on a per-

Table 1: **FrameBlend on streaming benchmarks.** We compare the accuracy of various SVU frameworks with and without FrameBlend on streaming VQA benchmarks.

	RVS-Ego		RVS-Movie		OVO-Bench		
Method	Acc.	Score	Acc.	Score	Real-time	Backward	Avg.
LLaVA-OV-7B							
ReKV	59.3	3.93	49.2	**3.45**	61.5	43.2	52.3
ReKV + FrameBlend	**62.4**	**3.99**	**50.1**	3.41	**63.0**	**44.6**	**53.8**
LiveVLM	60.1	3.95	51.5	3.50	63.6	43.5	53.5
LiveVLM + FrameBlend	**62.8**	**4.01**	**51.7**	**3.53**	**65.8**	**43.6**	**54.7**
HERMES	60.7	3.97	54.4	3.56	65.7	**48.9**	57.3
HERMES + FrameBlend	**62.0**	**4.00**	**56.2**	**3.61**	**66.6**	48.4	**57.5**
Qwen2.5-VL-7B							
ReKV	66.6	4.02	60.1	3.63	62.5	**45.3**	53.9
ReKV + FrameBlend	**69.7**	**4.05**	**62.6**	**3.67**	**65.4**	44.3	**54.9**
HERMES	68.0	**4.08**	61.7	3.65	66.7	48.2	57.4
HERMES + FrameBlend	**71.2**	4.06	**64.6**	**3.70**	**67.8**	**48.2**	**58.0**

Table 2: **FrameBlend on offline benchmarks.** We compare the accuracy of various SVU frameworks with and without FrameBlend on offline VQA benchmarks.

	Video-MME				MLVU				EgoSchema
LLaVA-OV-7B	Short	Medium	Long	Total	Holistic	Single-Detail	Multi-Detail	Total	Total
ReKV	**71.2**	55.3	46.9	57.8	83.2	73.1	41.1	68.4	60.0
ReKV + FrameBlend	70.7	**57.0**	**50.7**	**59.4**	**83.6**	**73.4**	**41.3**	**68.7**	**60.3**
LiveVLM	68.4	**56.2**	48.4	57.7	82.7	70.8	41.1	67.0	**60.6**
LiveVLM + FrameBlend	**69.6**	56.1	**48.8**	**58.1**	**83.4**	**71.9**	**42.8**	**68.1**	60.5
HERMES	70.4	52.0	44.9	55.8	81.2	61.5	35.7	60.2	59.7
HERMES + FrameBlend	**71.1**	**57.4**	**47.6**	**58.7**	**83.4**	**65.4**	**41.3**	**64.1**	**61.3**

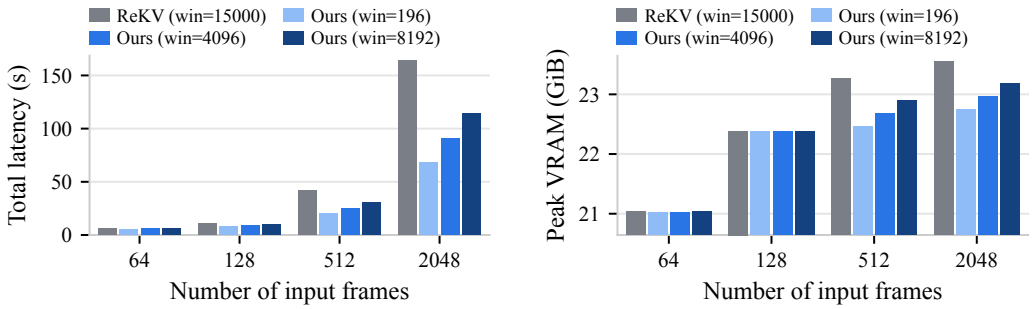
frame basis, as SVU continuously processes incoming frames before a query arrives. We analytically estimate FLOPs and measure wall-clock latency, reported as TFLOPs per frame and milliseconds per frame, respectively. For query-time efficiency, we report recomputation latency and time to first token (TTFT) per question. Since RVS-Ego contains multiple questions in each video, we evaluate each question independently using all video frames preceding that question as input.

As shown in Table 3, FrameBlend reduces per-frame prefill FLOPs by 12–26% and prefill latency by 13–61% across ReKV, HERMES, and LiveVLM. Although recomputation increases TTFT to approximately one second, the savings in idle-time computation exceed the additional query-time cost: total FLOPs decrease by 10–23% and total latency by 14–59%. These savings mainly result from the shorter prefill window enabled by recomputation, which reduces the attention computation required for each incoming frame.

Efficiency across different video lengths. Since the benefit of reducing per-frame prefill cost accumulates over a video stream, we further evaluate efficiency as the number of input frames increases. Figure 4a shows that FrameBlend increasingly reduces total latency over ReKV as the video grows longer, due to its shorter prefill window. Similarly, peak VRAM in Figure 4b exhibits greater memory savings as the video grows longer. This is because, in ReKV, the shorter prefill window keeps fewer KVs resident on the GPU while offloading older KVs to host memory. Additionally, peak VRAM is similar between ReKV and FrameBlend for short inputs, where memory usage is dominated by vision encoder activations.

Table 3: **Efficiency metrics.** We compare SVU methods with and without FrameBlend on RVS-Ego.

	Prefill (idle time)		Answering (query time)		Total	
LLaVA-OV-7B	TFLOPs/f	ms/f	Recomp. (s)	TTFT (s)	TFLOPs	Latency (s)
ReKV	4.38	81.1	–	0.17	4647.8	90.6
ReKV + FrameBlend	**3.24**	**32.0**	0.77	0.93	**3567.4**	**37.4**
LiveVLM	3.78	32.6	–	0.04	4007.2	37.6
LiveVLM + FrameBlend	**2.94**	**28.5**	0.77	0.84	**3247.6**	**32.3**
HERMES	4.07	46.4	–	0.03	4333.0	51.7
HERMES + FrameBlend	**3.57**	**37.7**	0.95	1.04	**3914.3**	**42.0**



(a) Total latency

(b) Peak VRAM

Figure 4: **Efficiency at varying numbers of input frames.** We analyze ReKV with FrameBlend under varying video lengths, and observe that the efficiency gain increases as the video gets longer.

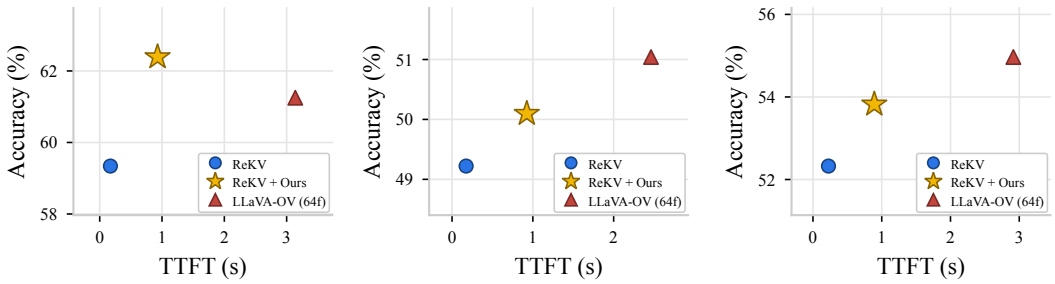
Comparison with query-time prefill. We also compare FrameBlend with an offline LLaVA-OV-7B baseline that uniformly samples and prefills 64 frames at query time. This places FrameBlend between two settings: ReKV minimizes TTFT by directly reusing cached KVs, whereas the offline baseline performs the full prefill after the query arrives. As shown in Figure 5, FrameBlend incurs additional TTFT over ReKV due to recomputation, but remains substantially faster than the offline baseline while achieving higher accuracy than ReKV. Thus, FrameBlend provides a balanced TTFT–accuracy trade-off by introducing selective recomputation to SVU frameworks.

5.4 Recomputation Target Selection Analysis

We compare QTS with CacheBlend (Yao et al., 2025) to evaluate the effect of recomputation target selection. For a controlled comparison, we follow the released CacheBlend implementation while using the same prefill window size and KV replacement policy as FrameBlend. This setup isolates the effect of target selection: the CacheBlend-based variant fully recomputes early layers to identify targets in a query-agnostic manner, whereas QTS directly selects query-relevant targets without such additional computation. As shown in Table 4, FrameBlend consistently achieves higher accuracy with lower added TTFT across all streaming benchmarks. These results show that QTS provides a more favorable accuracy–latency trade-off for selective recomputation in SVU.

5.5 Ablation Study

We ablate the two key components of FrameBlend on ReKV: a shorter attention window during idle-time prefill and selective recomputation at query time. As shown in Table 5, applying either component alone has mixed effects across benchmarks. A shorter prefill window removes part of the inter-token context, which can reduce unnecessary context but may also discard useful context for answering the query. Query-time recomputation complements this by selectively restoring query-relevant context among the selected tokens. Consequently, combining the shorter prefill window with recomputation achieves the highest overall accuracy across all benchmarks.



(a) RVS-Ego (b) RVS-Movie (c) OVO-Bench

Figure 5: **TTFT–accuracy trade-off.** We compare ReKV with and without FrameBlend, and the offline LLaVA-OV-7B using 64 uniformly sampled frames.

Table 4: **Comparison against CacheBlend.** Accuracy and TTFT when applied on ReKV.

	OVO-Bench		RVS-Movie		RVS-Ego	
LLaVA-OV-7B	Acc.	Added TTFT (s)	Acc.	Added TTFT (s)	Acc.	Added TTFT (s)
ReKV	52.3	–	49.2	–	59.3	–
ReKV + CacheBlend	52.7	0.90	48.0	1.51	61.4	2.26
ReKV + FrameBlend	**53.8**	**0.67**	**50.1**	**0.75**	**62.4**	**0.76**

Table 5: **Ablation studies.** We ablate two key components of FrameBlend.

	RVS-Ego		RVS-Movie		OVO-Bench		
LLaVA-OV-7B	Acc.	Score	Acc.	Score	Real-time	Backward	Avg.
ReKV	59.3	3.93	49.2	3.45	61.5	43.2	52.3
+ short window	60.7	3.94	46.8	3.30	**63.3**	43.0	53.1
+ recomputation	58.7	3.94	48.9	**3.46**	61.7	**44.8**	53.2
+ both	**62.4**	**3.99**	**50.1**	3.41	63.0	44.6	**53.8**

6 Conclusion

In this work, we introduced FrameBlend, a simple yet effective plug-and-play method for SVU. FrameBlend addresses the context mismatch in prior SVU frameworks by selectively recomputing cached KVs at query time without requiring any training. To enable efficient recomputation under the tight query-time latency budget of SVU, we propose QTS, which selects query-relevant recomputation targets with minimal additional overhead. We further show that, with query-time recomputation, using shorter attention windows during idle-time prefill can improve performance while substantially reducing computation. Evaluated across three SVU frameworks and five streaming and offline video understanding benchmarks, FrameBlend consistently improves (or matches) performance while reducing the overall computational cost.

Limitations. While FrameBlend improves accuracy and overall efficiency, several limitations remain and could be addressed in future work. Although we evaluate FrameBlend across multiple SVU frameworks, video-LLM backbones, and benchmarks, our experiments are limited to 7B-scale backbone models, and its effectiveness on both smaller and larger models remains to be explored. In addition, FrameBlend currently uses a fixed prefill window selected for each framework and benchmark; automatically adapting the window to different video streams could further improve its practicality. Finally, although FrameBlend reduces overall computation, selective recomputation introduces additional query-time latency. Dynamically adjusting the recomputation budget according to latency constraints could further improve its applicability to settings with stricter query-time latency requirements.

AI USE STATEMENT

In this work, we used generative AI tools to help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, and interpret experimental results. We have not used generative AI tools to generate synthetic datasets, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in writing proofs, clean or reformat datasets, or support qualitative and thematic data analysis. Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, create research artifacts, draft parts of the paper, summarize or analyze existing literature, discover research topics or identify gaps, brainstorm research ideas, source or search for information, improve the readability, and identify relevant literature. We have reviewed all AI-assisted work. We carefully reviewed AI-assisted hypotheses, interpretations of experimental results, scientific figures, code, and other research artifacts for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide framework-specific implementation details in Appendix A, prefill-window configurations in Appendix B. Section 5 describes the benchmarks, backbone models, hyperparameters, and hardware used in our experiments. Together, these details specify the experimental settings and evaluation procedures used to reproduce our results.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report, 2025b.

Xueyi Chen, Keda Tao, Kele Shao, and Huan Wang. StreamingTOM: Streaming token compression for efficient video understanding. In *Computer Vision and Pattern Recognition*, 2026a.

Yilong Chen, Xiang Bai, Zhibin Wang, Chengyu Bai, Yuhan Dai, and Ming Lu. StreamKV: Streaming video question-answering with segment-based KV cache retrieval and compression. In *AAAI Conference on Artificial Intelligence*, 2026b.

Shangzhe Di, Zhelun Yu, Guanghao Zhang, Haoyuan Li, ZhongTao, Hao Cheng, Bolin Li, Wanggui He, Fangxun Shu, and Hao Jiang. Streaming video question-answering with in-context video KV-cache retrieval. In *International Conference on Learning Representations*, 2025.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-MME: The first-ever comprehensive evaluation benchmark of multi-modal LLMs in video analysis. In *Computer Vision and Pattern Recognition*, 2025.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *International Conference on Machine Learning*, 2025.

Minsoo Kim, Kyuhong Shim, Jungwook Choi, and Simyung Chang. InfiniPot-V: Memory-constrained KV cache compression for streaming video understanding. *Advances in Neural Information Processing Systems*, 2025.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. LLaVA-OneVision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. EgoSchema: A diagnostic benchmark for very long-form video language understanding. *Advances in Neural Information Processing Systems*, 2023.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. EmbodiedGPT: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 2023.

Zhenyu Ning, Guangda Liu, Qihao Jin, Chengwei Li, Wenchao Ding, Minyi Guo, and Jieru Zhao. LiveVLM: Efficient online video understanding via streaming-oriented KV cache and retrieval. In *Design Automation Conference*, 2026.

Junbo Niu, Yifei Li, Ziyang Miao, Chunjiang Ge, Yuanhang Zhou, Qihao He, Xiaoyi Dong, Haodong Duan, Shuangrui Ding, Rui Qian, et al. OVO-Bench: How far is your video-LLMs from real-world online video understanding? In *Computer Vision and Pattern Recognition*, 2025.

Zhanzhong Pang, Dibyadip Chatterjee, Fadime Sener, and Angela Yao. Decouple and cache: KV cache construction for streaming video understanding. In *International Conference on Machine Learning*, 2026.

Rui Qian, Shuangrui Ding, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. Dispider: Enabling video LLMs with active real-time interaction via disentangled perception, decision, and reaction. In *Computer Vision and Pattern Recognition*, 2025.

Shengling Qin, Hao Yu, Chenxin Wu, Zheng Li, Yizhong Cao, Zhengyang Zhuge, Yuxin Zhou, Wentao Yao, Yi Zhang, Zhengheng Wang, et al. VLCache: Computing 2% vision tokens and reusing 98% for vision-language inference. *arXiv preprint arXiv:2512.12977*, 2025.

Aravinda S Rao, Marko Radanovic, Yuguang Liu, Songbo Hu, Yihai Fang, Kourosh Khoshelham, Marimuthu Palaniswami, and Tuan Ngo. Real-time monitoring of construction sites: Sensors, methods, and applications. *Automation in Construction*, 2022.

Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L Waslander, Yu Liu, and Hongsheng Li. LMDrive: Closed-loop end-to-end driving with large language models. In *Computer Vision and Pattern Recognition*, 2024.

Yujiao Shen, Shulin Tian, Jingkang Yang, and Ziwei Liu. A simple baseline for streaming video understanding. *arXiv preprint arXiv:2604.02317*, 2026.

Haibo Wang, Bo Feng, Zhengfeng Lai, Mingze Xu, Shiyu Li, Weifeng Ge, Afshin Dehghan, Meng Cao, and Ping Huang. StreamBridge: Turning your offline video large language model into a proactive streaming assistant. *Advances in Neural Information Processing Systems*, 2025.

Yanlai Yang, Zhuokai Zhao, Satya Narayan Shukla, Aashu Singh, Shlok Kumar Mishra, Lizhu Zhang, and Mengye Ren. StreamMem: Query-agnostic KV cache memory for streaming video understanding. *arXiv preprint arXiv:2508.15717*, 2025.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *European Conference on Computer Systems*, 2025.

Xiangyu Zeng, Kefan Qiu, Qingyu Zhang, Xinhao Li, Jing Wang, Jiaxin Li, Ziang Yan, Kun Tian, Meng Tian, Xinhai Zhao, et al. StreamForest: Efficient online video understanding with persistent event memory. *Advances in Neural Information Processing Systems*, 2025.

Haoji Zhang, Yiqin Wang, Yansong Tang, Yong Liu, Jiashi Feng, Jifeng Dai, and Xiaojie Jin. Flash-VStream: Memory-based real-time understanding for long video streams. *arXiv preprint arXiv:2406.08085*, 2024.

Haowei Zhang, Shudong Yang, Jinlan Fu, See Kiong Ng, and Xipeng Qiu. HERMES: KV cache as hierarchical memory for efficient streaming video understanding. In *Association for Computational Linguistics*, 2026.

Shiju Zhao, Junhao Hu, Rongxiao Huang, Jiaqi Zheng, and Guihai Chen. MPIC: Position-independent multimodal context caching system for efficient MLLM serving. *arXiv preprint arXiv:2502.01960*, 2025.

Junjie Zhou, Yan Shu, Bo Zhao, Boya Wu, Zhengyang Liang, Shitao Xiao, Minghao Qin, Xi Yang, Yongping Xiong, Bo Zhang, et al. MLVU: Benchmarking multi-task long video understanding. In *Computer Vision and Pattern Recognition*, 2025.

A FRAMEWORK-SPECIFIC IMPLEMENTATION DETAILS

A.1 REKV WITH FRAMEBLEND

ReKV (Di et al., 2025) stores the KVs of all incoming frames and independently retrieves the top 64 frames at each LLM layer when a query arrives. During idle-time prefill, the original ReKV lets each token attend to up to 15,000 preceding video tokens using a sliding-window attention. We apply the short prefill window of FrameBlend by changing only this window size; the retrieval and decoding procedures remain unchanged.

For recomputation target selection, we directly use the layer-wise retrieval results produced by ReKV. Because ReKV retrieves complete frame blocks, all tokens within a frame block share the same retrieval score; QTS therefore operates at the frame level. We first rank each frame by the number of layers in which it is retrieved. Ties are broken by the sum of its normalized retrieval scores over the layers that retrieve it. The selected frames are ordered temporally and jointly re-prefilled through all LLM layers. At each layer, only frames that are both selected for recomputation and present in that layer's original retrieval set are replaced by their recomputed KVs. Therefore, FrameBlend does not add frames to the layer-wise decoding context or modify ReKV's retrieval policy.

A.2 LIVEVLM WITH FRAMEBLEND

LiveVLM (Ning et al., 2026) processes incoming frames while maintaining a compressed KV cache. It applies compression when the cache exceeds its budget of 12,000 tokens. At query time, it independently retrieves 40% of 16-token pages for each layer and head. The resulting query-time context can therefore contain different video tokens across both layers and heads.

FrameBlend uses sliding-window attention with a separate, small KV buffer, allowing each incoming token to attend only to the most recent tokens of the original stream within the prefill window. The separate KV buffer ensures that the effective window size does not depend on how much the KV cache has been compressed. For QTS, we use the tokens selected by LiveVLM's retrieval mechanism as candidates. Because LiveVLM has different retrieval sets for each layer–head pair, we count how many layer–head pairs retrieve each token. We then select the most frequently retrieved tokens based on this count, using normalized retrieval scores for tie-breaking. The selected tokens are jointly re-prefilled, and only their corresponding slots in the temporary query-time cache are replaced.

A.3 HERMES WITH FRAMEBLEND

HERMES (Zhang et al., 2026) maintains a compressed KV cache instead of performing query-time retrieval. We set the maximum KV cache size to 6,000 video tokens, and HERMES independently compresses the cache at each layer to this budget. The original HERMES allows incoming tokens to attend to the entire compressed cache. For FrameBlend, as in LiveVLM, we restrict this prefill attention to the most recent video tokens within the prefill window using sliding-window attention. We also maintain a separate KV buffer containing the uncompressed KVs of these recent tokens for the same reason as in LiveVLM.

Because HERMES has no query-time retrieval, we perform an auxiliary retrieval forward pass only for recomputation target selection. At each layer, we retrieve the top-M video tokens using their query similarity and use these layer-wise sets as the voting sets for QTS. We use $M = 2{,}048$ for RVS and $M = 4{,}096$ for OVO-Bench and the offline benchmarks. We then rank tokens by the number of layers that select them and choose the most frequently retrieved tokens. This auxiliary retrieval affects only which KVs are recomputed; the decoding context remains the full compressed KV cache produced by HERMES. The selected tokens are re-prefilled in temporal order using the layer-specific position indices maintained by HERMES, and the recomputed KVs replace only the corresponding surviving slots in each layer.

Table 6: **Prefill window sizes used by FrameBlend.** Values denote the number of video tokens.

	Streaming benchmarks			Offline benchmarks		
Method	OVO-Bench	RVS-Ego	RVS-Movie	Video-MME	MLVU	EgoSchema
LLaVA-OV-7B						
ReKV	2048	196	196	8192	8192	384
HERMES	1024	1024	512	512	512	1024
LiveVLM	196	1024	4096	2048	4096	512
Qwen2.5-VL-7B						
ReKV	4096	4096	1024	–	–	–
HERMES	4096	512	512	–	–	–

B Prefill window size tuning

We tune the size of the short attention window for idle-time prefill separately for each framework and benchmark. The original attention ranges are 15,000 tokens for ReKV, 12,000 tokens for LiveVLM, and 6,000 tokens for HERMES in our experiments. Table 6 summarizes the window sizes used for the main and appendix results. Across the evaluated settings, FrameBlend uses only 1.3–54.6% of ReKV's original attention range, 1.6–34.1% of LiveVLM's, and 8.5–68.3% of HERMES's.

C Additional Prefill Window Size Analysis

To further examine how the prefill window affects performance, we sweep the window size under full recomputation ($K = \infty$) for ReKV with FrameBlend for simplicity. As shown in Figure 6, performance typically peaks at a particular window size rather than improving monotonically as the window grows. Overall, prefill windows shorter than ReKV's default window of 15K tokens tend to perform better, while the optimal window size varies across benchmarks. Even when the window is relatively large, such as 8K tokens, it is still only about half of the default window size. This also suggests that the appropriate amount of idle-time context is benchmark-dependent, making automatic adaptation of the window size to the video stream an interesting direction for future work.

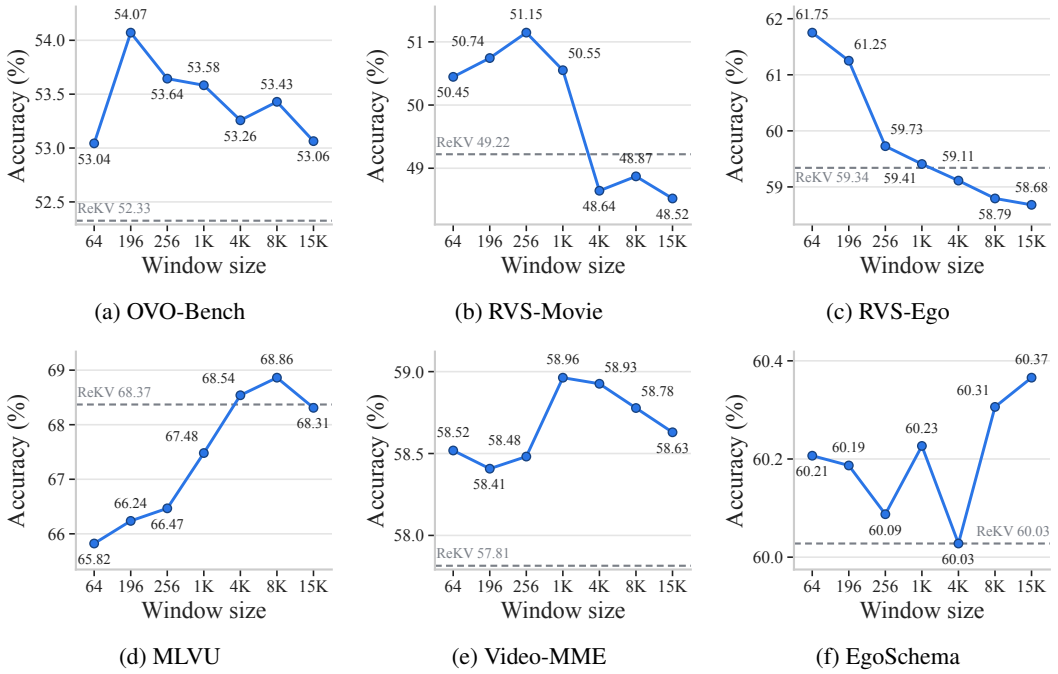
D Additional efficiency analysis

Here, we provide additional efficiency results for HERMES and LiveVLM as the video stream grows. As in the main experiments, wall-clock latency is measured on a single NVIDIA A100 80GB GPU. As shown in Figure 7, FrameBlend reduces latency for both frameworks, although the reduction is smaller than for ReKV. This is because HERMES and LiveVLM already use compressed KV caches and therefore spend less time on attention.

For peak GPU memory, FrameBlend with a window size of 196 uses comparable or lower memory than the corresponding baseline. With a window size of 4096, the additional KV buffer used by compression-based frameworks slightly increases peak memory, by approximately 0.38 GiB for HERMES and 0.22 GiB for LiveVLM.

E Additional CPU memory usage

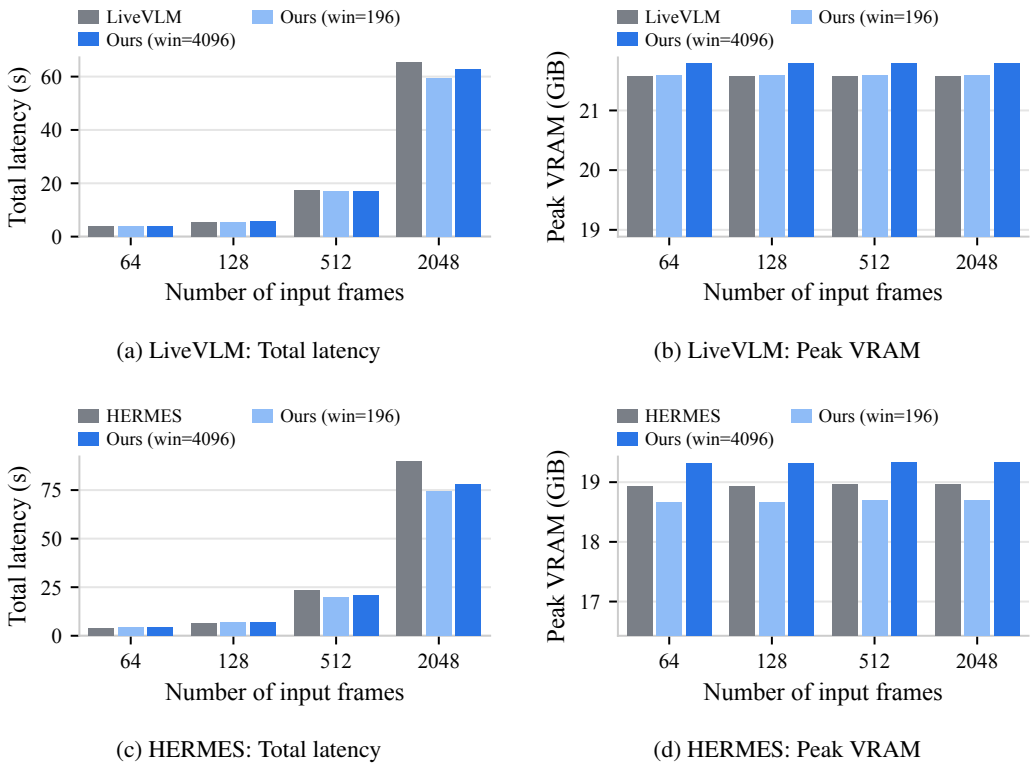
FrameBlend additionally stores projected visual embeddings in CPU memory for query-time recomputation. On RVS-Ego, whose videos are approximately one hour long, these embeddings require 2.36 GiB for ReKV, 0.25 GiB for HERMES, and 1.01 GiB for LiveVLM. For compression-based frameworks, only embeddings corresponding to tokens that remain in at least one layer can become recomputation candidates, resulting in lower CPU memory overhead.



(a) OVO-Bench (b) RVS-Movie (c) RVS-Ego

(d) MLVU (e) Video-MME (f) EgoSchema

Figure 6: **Prefill window size sweep with full recomputation.** We evaluate the accuracy of ReKV with FrameBlend across different prefill window sizes under full recomputation.



(a) LiveVLM: Total latency (b) LiveVLM: Peak VRAM

(c) HERMES: Total latency (d) HERMES: Peak VRAM

Figure 7: **Efficiency at varying numbers of input frames.** We analyze LiveVLM and HERMES with FrameBlend under varying video lengths and observe that the efficiency gain increases as the video becomes longer.