APB-V: Accelerating Long-Video Understanding via Sequence-Parallelism-aware Approximate Attention

Yuxiang Huang[1], **Mingye Li**[2], **Xu Han**[1]*, **Chaojun Xiao**[1]*, **Weilin Zhao**[1], **Ao Sun**[3], **Ziqi Yuan**[1], **Hao Zhou**[4], **Fandong Meng**[4], **Zhiyuan Liu**[1]
[1]NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing, China.
[2]Department of CS&T, Central South University, Changsha, China.
[3]BUPT, Beijing, China. [4]Pattern Recognition Center, WeChat AI, Tencent Inc.
huang-yx21@mails.tsinghua.edu.cn, {han-xu, xcj}@tsinghua.edu.cn

Abstract

The efficiency of long-video inference remains a critical bottleneck, mainly due to the dense computation in the prefill stage of Large Multimodal Models (LMMs). Existing methods either compress visual embeddings or apply sparse attention on a single GPU, yielding limited acceleration or degraded performance and restricting LMMs from handling longer, more complex videos. To overcome these issues, we propose APB-V, a sequence-parallel framework with optimized attention that accelerates long-video inference across multiple GPUs. By distributing approximate attention, APB-V reduces computation and increases parallelism, enabling efficient processing of more visual embeddings without compression and thereby improving task performance. System-level optimizations, such as load balancing and fused forward passes, further unleash the potential of APB-V, delivering speedups of 12.72×, 1.70×, and 1.18× over FLASHATTN, ZIGZAGRING, and APB, without notable performance loss. Code available at https://github.com/thunlp/APB.

1 Introduction

The rapid advances of Large Language Models (LLMs) (Achiam et al., 2023; Anthropic, 2025; DeepSeek-Team, 2024; Google, 2025) in long-context inference have catalyzed the concurrent development of Large Multimodal Models (LMMs) (Tang et al., 2023; Qwen-Team, 2025; InternVL-Team, 2025), endowing LMMs with the capacity to understand extended video sequences. This capability is typically realized through the synergistic integration of visual encoders (Dosovitskiy et al., 2020) with long-context LLM backbones, yielding remarkable performance on long-video benchmarks such as LongVideoBench (Wu et al., 2024) and VNBench (Zhao et al., 2025). Despite the potential in advancing long-video processing, LMMs encounter severe efficiency bottlenecks when handling ultra-long videos. An increased number of video frames results in a larger batch size for the visual encoder, leading to a rise in both compute cost and the number of generated video embeddings. Given the quadratic complexity of the attention layers employed by LLM backbones, these additional video embeddings extend the input length and slow down the inference speed.

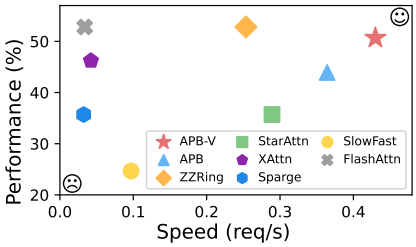

Figure 1: APB-V's performance and inference speed using `Qwen2.5VL-3B` as the base LMM on VNBench (processing 64-frame 1440p videos).

To alleviate the above efficiency issue, existing efforts can be categorized into two main approaches: ***intrinsic attention optimization*** and ***explicit input reduction***. *Intrinsic attention optimization* (Xu et al., 2025b; Li et al., 2025; Xiao et al., 2024b) methods aim to reduce the computational burden of the LLM backbone within LMMs by optimizing the attention mechanism or minimizing the key-value (KV) cache size, but cannot address the computational cost of the feed-forward networks (FFNs) and the visual encoder. *Explicit input reduction* methods (Xu et al., 2024; Choudhury et al., 2024; Yao et al., 2025) aim to reduce the number of video embeddings by applying adaptive selection or pooling operations within or after the visual encoder, thereby lowering the overall computational cost. Since long-video processing typically involves computation-intensive inference, neither approach can achieve satisfactory efficiency improvements while preserving fine-grained

* indicates corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2010–2025
July 2-7, 2026 ©2026 Association for Computational Linguistics

video details. Consequently, *existing approaches inevitably face a trade-off between* ***efficiency*** *and* ***performance*** (e.g., in our experiments, the typical method SLOWFAST achieves less than $3\times$ speedup but suffers around 25% accuracy degradation in Figure 7 and Table 1). As single-GPU methods usually suffer from excessive compute reduction and cannot well overcome the efficiency-performance trade-off, a third scaling dimension is necessary.

Scaling computational power while suppressing quadratic complexity, i.e., by running approximate attention with reduced compute across more GPUs (referred to as *hosts*), offers a promising pathway, directly addressing the core challenges and providing the following advantages.

Advantage 1: Supporting strong compute density through well designed parallelism. Long-video inference lends itself naturally to parallelization when guided by effective design. The video encoding process is inherently independent across frames, allowing distribution across multiple hosts. Meanwhile, the LLM backbone can be parallelized via sequence parallelism, with context segments placed and processed on different hosts. However, the dense computation and communication in exact sequence parallelism algorithms hinder scalability to larger host counts, resulting in limited efficiency gain (ZZRING in Figure 1). In contrast, approximate attention improves scalability through optimized computation and compressed communication, enabling faster inference.

Advantage 2: Preserving task performance. Conventional optimizations often sacrifice fine-grained details in long-video inputs, especially when token pruning is applied for explicit input reduction. By replacing input reduction with approximate attention compute reduction, the full set of visual embeddings is preserved, thereby safeguarding task performance.

Building on this perspective, we propose APB-V. Our main contributions are summarized as follows:

(I) We introduce APB-V, a sequence-parallelism-aware approximate attention framework for accelerating long-video understanding. Unlike embedding compression or single-GPU attention optimization, APB-V adopts local KV cache compression with a distributed approximate attention mechanism, well striking an efficiency-performance balance.

(II) We optimize APB-V from a system-level perspective by designing a load-balancing strategy across hosts, along with a two-stage attention mechanism where communication perfectly overlapped with computation.

(III) We evaluate APB-V on extensive benchmarks. As shown in Table 1, the experimental results show that APB-V exhibits the best tradeoff between performance and efficiency. More specifically, APB-V achieves speedup of $12.72\times$, $1.70\times$, and $1.18\times$ over FLASHATTN, ZIGZAGRING, and APB, without significant performance degredation.

2 Preliminaries

We introduce the basic notations of Transformers and long-video inference in this section.

Transformers. For an L-layer Transformer-based model, each layer consists of an attention block and an FFN block (Vaswani et al., 2017). The model takes a sequence of d-dimensional embeddings $\mathbf{E} \in \mathbb{R}^{n\times d}$ as the input, where n is the input sequence length. For each layer, given the input hidden $\mathbf{H}$, the query, key, and value matrices are first computed as $\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \mathbf{K} = \mathbf{H}\mathbf{W}_K, \mathbf{V} = \mathbf{H}\mathbf{W}_V$, and then the attention is applied as

$$\begin{aligned}\mathbf{A} &= \mathtt{Attn}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right)\\ &= \mathtt{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}}{\sqrt{d}}\right)\cdot \mathbf{V},\end{aligned} \tag{1}$$

where $\mathbf{M}$ is the attention mask. Based on the attention block, the FFN block computation is

$$\mathbf{H}' = \mathtt{FFN}\left(\mathbf{A}\right). \tag{2}$$

We omit the notations for attention heads, position embeddings, and normalizations for simplicity.

Long-Video Inference. An LMM $\mathcal{M}$ typically consists of a visual encoder $\mathcal{M}_v$, a connector $\mathcal{M}_c$, an embedding layer $\mathcal{M}_e$, and an LLM backbone $\mathcal{M}_l$. During the long-video inference, the video is first divided into F frames, each containing $H \times W$ d_v-dimensional patches. We denote the patchified input as a fourth-order tensor $\mathbf{I} \in \mathbb{R}^{F\times H\times W\times d_v}$. For the i-th frame $\mathbf{I}_i$, it is first reshaped into a second-order embedding sequence $\tilde{\mathbf{I}}_i \in \mathbb{R}^{HW\times d_v}$, and then encoded by the visual encoder, producing the output $\mathbf{O}_i = \mathcal{M}_v(\tilde{\mathbf{I}}_i) \in \mathbb{R}^{HW\times d_v}$. By encoding each frame output $\mathbf{O}_i$ to n_v d-dimensional embeddings through the connector $\mathcal{M}_c$, we get the whole video embeddings $\mathbf{E}_v = \mathcal{M}_c\left([\mathbf{O}_1,\ldots,\mathbf{O}_F]\right) \in \mathbb{R}^{Fn_v\times d}$. Finally, the video embeddings and the text embeddings (encoded from the textual context by $\mathcal{M}_e$) are concatenated to form the input for the LLM $\mathcal{M}_l$, denoted

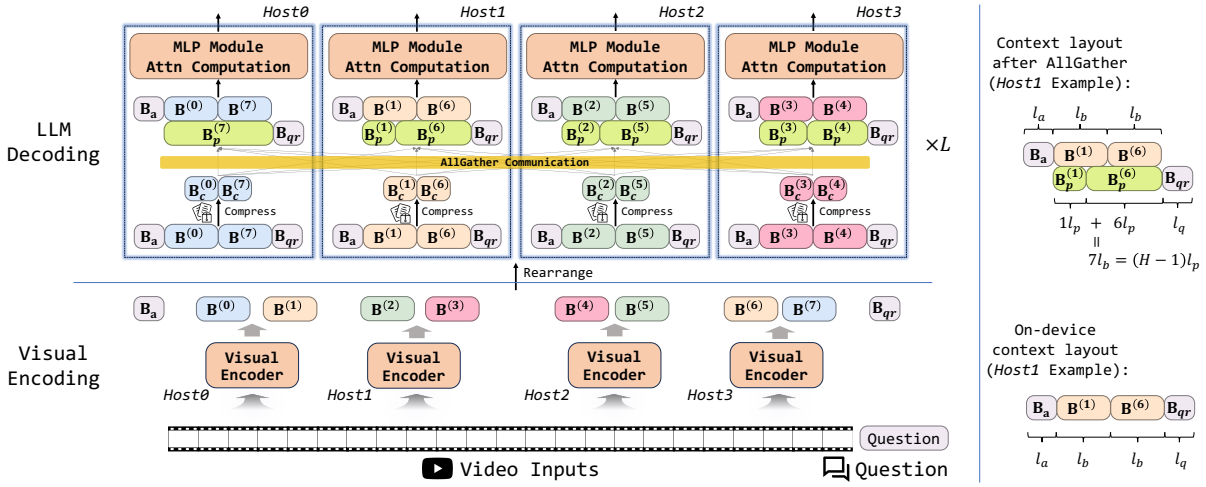

Figure 2: The framework of APB-V. The anchor block and passing block are denoted as $\mathbf{B}_a$ and $\mathbf{B}_p$, while $\mathbf{B}^{(h)}$ denotes the context block on virtual host h. $\mathbf{B}_{qr}$ represents the query block. The video input is first encoded into embeddings using frame parallelism across hosts. After context splitting, each physical host (containing two virtual hosts) holds the anchor block, query block, and corresponding context blocks. In approximate attention, each context block is first compressed then communicated. Attention is then computed over $\mathbf{B}_a$, $\mathbf{B}^{(h)}$, $\mathbf{B}^{(2H-1-h)}$, $\mathbf{B}_p^{(h)}$, $\mathbf{B}_p^{(2H-1-h)}$, and $\mathbf{B}_{qr}$. The passing blocks are discarded immediately after attention.

as $\mathbf{E} = [\mathbf{E}_v, \mathbf{E}_t] \in \mathbb{R}^{n \times d}$, where $\mathbf{E}_t \in \mathbb{R}^{n_t \times d}$ is the text embeddings and $n = n_v + n_t$.

3 APB-V's Framework Design

APB-V is a long-video inference framework that leverages specially designed sequence parallelism and approximate attention, as shown in Figure 2. In this section, we present the main components in the order they operate during inference. We assume the input to an LMM $\mathcal{M}$ is $[\mathbf{I}, \mathbf{T}_\mathrm{Q}]$, where $\mathbf{T}_\mathrm{Q}$ denotes the token sequences of the query, and $\mathbf{I}$ represents the patchified video input. We define a ***host*** as a group of GPUs that maintains a full replica of the LMM. Inference is performed across H such hosts.

3.1 Frame Parallelism

Initially, an LMM's visual encoder transforms the visual inputs into embeddings, as shown in Section 2. Since each frame is encoded independently, we adopt frame parallelism to address the high computational intensity, inspired by LongVILA (Chen et al., 2025). On the h-th host, we obtain the input embeddings of the entire text query and a subset of video frames as

$$\mathbf{E}_v^{(h)}, \mathbf{E}_Q = \mathcal{M}_c\left(\mathcal{M}_v\left(\mathbf{I}^{(h)}\right)\right), \mathcal{M}_e\left(\mathbf{T}_\mathrm{Q}\right), \quad (3)$$

where $\mathbf{I}^{(h)}$ is a subset of $\mathbf{I}$ for the h-th host.

3.2 Context Splitting

We begin with a collective communication step that enables each host to access the complete sequence: $\mathbf{E}_v = \texttt{AllGather}\left(\{\mathbf{E}_v^{(h)}\}_{h=1}^{H}\right)$. Then, the input embeddings are partitioned into three components:

- ***Anchor Block***: $\mathbf{B}_a = [e_1, \cdots, e_{l_a}]$, comprising the initial l_a embeddings of $\mathbf{E}_v$;
- ***Query Block***: $\mathbf{B}_{qr} = \mathbf{E}_Q$, consisting of all query embeddings at the end of the sequence;
- ***Context Block***: $\mathbf{B}^{(h)}$, formed by equally dividing the remaining sequence of $\mathbf{E}_v$ across hosts.

In decoder-only models, different context blocks attend to varying amounts of context, and naive host assignment thus leads to imbalanced attention computation. We introduce a ZigZag-style arrangement to solve this issue. Specifically, we introduce $2H$ virtual hosts during context splitting. A ***virtual host*** is a logical construct that does not correspond directly to any physical GPU assignment, whereas a ***physical host*** corresponds to an actual set of GPUs. Each physical host contains two complementary virtual hosts to ensure better load balancing. More details of the mapping from physical to virtual hosts are introduced in Section 4.3. For simplicity in notations, we use virtual hosts to describe the following attention process.

3.3 Approximate Attention

To reduce both the computational and communication overhead of attention, we design an approximate attention mechanism based on the observation that only the most essential KV pairs need to be visible to subsequent tokens, whereas the others can remain confined within their local blocks, avoiding

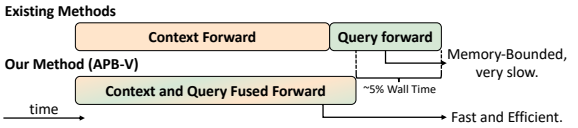

Figure 3: Fused context and query forward pass.

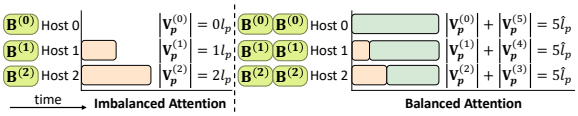

Figure 4: Attention load balancing. "$|\mathbf{V}_c^{(h)}|$" is the total length of previous essential KVs of virtual host h.

unnecessary cross-host attention.

The h-th virtual host maintains the anchor block $\mathbf{B}_a$, the context block $\mathbf{B}^{(h)}$, and the query block $\mathbf{B}_{qr}$, and the QKV states of these blocks are $\{\mathbf{Q}_a, \mathbf{K}_a, \mathbf{V}_a\}$, $\{\mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)}\}$, $\{\mathbf{Q}_{qr}, \mathbf{K}_{qr}, \mathbf{V}_{qr}\}$, respectively. On the h-th host, we first use the query-to-context attention scores $\mathbf{Q}_{qr}\mathbf{K}^{(h)\top}$ to identify the most l_p important KV pairs $(\mathbf{K}_c^{(h)}, \mathbf{V}_c^{(h)})$ from $(\mathbf{K}^{(h)}, \mathbf{V}^{(h)})$. Then, we build the ***passing blocks*** that contain the essential KV pairs on previous hosts whose host index is no larger than h, similar to APB (Huang et al., 2025), by $\mathbf{K}_p^{(h)}, \mathbf{V}_p^{(h)} = \texttt{AllGather}\left(\{\mathbf{K}_c^{(h)}, \mathbf{V}_c^{(h)}\}_{h=1}^{H}\right)_{\leq h}$. Finally, we compute the attention scores for the anchor and context blocks:

$$\begin{aligned}\mathbf{A}_a &= \texttt{Attn}\left(\mathbf{Q}_a, \mathbf{K}_a, \mathbf{V}_a\right), \\ \mathbf{A}_h &= \texttt{Attn}\left(\mathbf{Q}^{(h)}, \left[\mathbf{K}_a, \mathbf{K}_p^{(h)}, \mathbf{K}^{(h)}\right], \left[\mathbf{V}_a, \mathbf{V}_p^{(h)}, \mathbf{V}^{(h)}\right]\right),\end{aligned} \tag{4}$$

where $\mathbf{A}_a, \mathbf{A}^{(h)}$ correspond to the attention results of the anchor and context blocks.

We simultaneously compute the attention result $\mathbf{A}^{qr}$ for the query block, utilizing FLASHATTN's `lse` to merge the results from different hosts:

$$\begin{aligned}&\mathbf{A}_{qr}^{(h)}, \texttt{lse}^{(h)} = \texttt{Attn}(\mathbf{Q}_{qr}, \tilde{\mathbf{K}}_{qr}, \tilde{\mathbf{V}}_{qr}), \\ &\mathbf{A}_{qr} = \texttt{Merge}(\texttt{AllGather}(\{\mathbf{A}_{qr}^{(h)}, \texttt{lse}^{(h)}\})_{h=1}^{H}),\end{aligned} \tag{5}$$

where $\tilde{\mathbf{K}}_{qr} = [\mathbf{K}_a^{(h)}, \mathbf{K}^{(h)}, \mathbf{K}_{qr}]$, with $\mathbf{K}_a^{(h)}$ denoting the sliced anchor block for load balancing such that $[\mathbf{K}_a^{(h)}]_{h=1}^{H} = \mathbf{K}_a$. (The same applies to $\tilde{\mathbf{V}}_{qr}$.) Further details are provided in Appendix A.

4 APB-V's System Optimizations

The efficiency in existing methods (Acharya et al., 2025; Huang et al., 2025) that adopt sequence-parallelism inference is usually bounded by:

- Imbalanced video encoding across hosts;
- Memory-bound forward pass for query prefill;

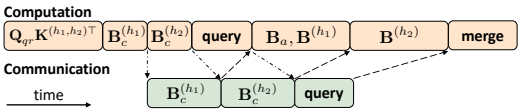

Figure 5: Overlapping communication with computation on virtual host h. "$\mathbf{Q}_{qr}\mathbf{K}^{(h_1,h_2)\top}$" estimates KV importance; "$\mathbf{B}_c^{(h_1,h_2)}$" are essential KVs; "`query`" and "`merge`" denote query attention and its merging; "$\mathbf{B}_a$" and "$\mathbf{B}^{(h_1,h_2)}$" indicate anchor and context attention.

- Imbalanced attention computation across hosts;
- Inefficient communication design.

We briefly introduce the following system optimizations to alleviate these obstacles, with more details introduced in Appendix B.

4.1 Visual Load Balancing

To enable balanced frame parallelism in the visual encoder, we design a visual load-balancing strategy by evenly distributing the number of frames across virtual hosts. Given F frames and H virtual hosts, the number of frames assigned to the h-th host is:

$$F^{(h)} = \left\lfloor \frac{F}{H} \right\rfloor + \mathbb{I}\left[h < F \bmod H\right], \tag{6}$$

where $\mathbb{I}$ denotes the indicator function.

4.2 Fused Context and Query Forward

Previous methods execute the prefill stage for context and query separately. Since queries are typically short, a dedicated forward pass for the query often becomes memory-bound and inefficient, as shown in Figure 3. To overcome this, we fuse context and query blocks into a single forward pass, computing attention jointly. The partial query result $\mathbf{A}_{qr}^{(h)}$ is merged using the `lse` output from FLASHATTN at the end of each attention layer.

4.3 Approximate Attention Load Balancing

Adopting vanilla sequence parallelism for approximate attention incurs imbalanced attention computation: for host h, the number of passing blocks is $h \cdot l_p$, leading to uneven workloads. To mitigate this, we adopt a ZigZag-style (Zhu, 2024) load-balancing strategy, assigning the h-th and $(2H-1-h)$-th virtual hosts to the same physical host, so that the total length of passing blocks is balanced across hosts, as shown in Figure 4.

4.4 Overlapped Communication

Existing methods like STARATTN (Acharya et al., 2025) avoid inter-host communication but fail to capture long-term dependencies. In contrast,

Method	Retrieval				Ordering				Counting				Overall
	E	I-1	I-2	Avg.	E	I-1	I-2	Avg.	E-1	E-2	I	Avg.	
`InternVL3-2B`													
FullAttn	90.00	90.67	36.00	72.22	64.67	24.00	24.67	37.78	40.67	4.67	28.67	24.67	44.89
XAttn	90.00	**90.67**	**38.00**	**72.89**	54.00	**23.33**	15.33	30.89	**37.33**	**7.33**	**28.67**	**24.44**	42.74
SpargeAttn	**91.33**	89.33	33.33	71.33	46.67	11.33	18.67	25.56	26.67	4.67	24.67	18.67	38.52
SlowFast	48.00	56.67	32.00	45.56	15.33	6.67	8.67	10.22	24.67	5.33	26.00	18.67	24.81
StarAttn	90.00	88.67	34.67	71.11	29.33	6.00	10.00	15.11	18.00	6.00	22.00	15.33	33.85
APB	89.33	89.33	36.00	71.56	59.33	17.33	18.00	31.56	33.33	3.33	24.00	20.22	41.11
APB-V	90.67	89.33	32.67	70.89	**64.00**	22.00	**21.33**	**35.78**	**37.33**	5.33	26.67	23.11	**43.26**
`Qwen2.5VL-3B`													
FullAttn	90.67	84.00	48.00	74.22	72.67	51.33	37.33	53.78	53.33	8.00	30.00	30.44	52.81
XAttn	**90.00**	77.33	**48.00**	71.78	65.33	39.33	23.33	42.67	36.67	**10.00**	26.00	24.22	46.22
SpargeAttn	89.33	75.33	37.33	67.33	36.67	13.33	10.00	20.00	26.00	8.00	25.33	19.78	35.70
SlowFast	41.33	64.00	25.33	43.56	12.00	10.00	12.67	11.56	23.33	5.33	28.00	18.89	24.67
StarAttn	**90.00**	83.33	46.67	**73.33**	18.67	10.67	10.67	13.33	27.33	6.67	27.33	20.44	35.70
APB	**90.00**	**82.67**	44.00	72.22	47.33	29.33	22.00	32.89	44.67	8.00	27.33	26.67	43.93
APB-V	**90.00**	**82.67**	46.67	73.11	**66.00**	**47.33**	**33.33**	**48.89**	**52.00**	9.33	**28.67**	**30.00**	**50.67**
`Qwen2.5VL-7B`													
FullAttn	90.67	82.00	59.33	77.33	74.67	59.33	57.33	63.78	54.67	13.33	34.67	34.22	58.44
XAttn	**90.67**	79.33	**60.00**	**76.67**	70.67	53.33	44.67	56.22	47.33	12.00	**32.00**	30.44	54.44
SpargeAttn	88.67	79.33	50.67	72.89	63.33	35.33	38.00	45.56	44.00	10.67	26.67	27.11	48.52
SlowFast	43.33	69.33	42.67	51.78	14.00	12.00	9.33	11.78	22.00	9.33	28.67	20.00	27.85
StarAttn	**90.67**	**82.00**	57.33	**76.67**	27.33	20.00	18.67	22.00	25.33	12.00	28.67	22.00	40.22
APB	89.33	**82.00**	58.67	**76.67**	58.00	35.33	36.67	43.33	48.00	11.33	30.00	29.78	49.93
APB-V	**90.67**	80.67	58.00	76.44	**72.00**	**56.67**	**48.00**	**58.89**	**54.00**	**14.00**	**32.00**	**33.33**	**56.22**

Table 1: Accuracies (%) of VNBench. "E" and "I" represent the edited and inserted data subset, respectively.

APB (Huang et al., 2025) incorporates inter-host communication to model such dependencies, at the cost of higher communication overhead. To address this, we overlap communication with computation, as illustrated in Figure 5. The transfer of passing blocks and partial query attention results are performed concurrently with attention calculation.

5 Experiments

We now present our empirical analysis of APB-V, focusing on the following questions.
(***Q1***) Can APB-V achieve similar or better task performance compared with other baselines?
(***Q2***) Can APB-V obtain a faster inference speed under various video lengths and resolutions?
(***Q3***) How does each component of APB-V contribute to overall performance and efficiency?

We also provide a case study in Section 5.5.

5.1 Experimental Settings

Benchmarks. We evaluate two long-video benchmarks: LongVideoBench (Wu et al., 2024) and VNBench (Zhao et al., 2025). LongVideoBench contains 1337 real-world videos in four duration ranges (8s-15s, 15s-60s, 180s-600s, and 900s-3600s). VNBench offers 1350 synthetic videos featuring challenging retrieval, ordering, and counting tasks. We set the frame number to 64 for both benchmarks.

Models. To examine the effect of APB-V on various model architectures and model sizes, we conduct experiments based on three LMMs: `InternVL3-2B` (InternVL-Team, 2025), and `Qwen2.5VL-3B/7B` (Qwen-Team, 2025). `InternVL3-2B` resizes frames to 448×448, while `Qwen2.5VL` models supports native-resolution processing using the patch size of 14×14.

Metrics. We use the benchmarks' original metrics for performance evaluation. For speed measurement, we define inference speed as requests processed per second (req / s) and compute relative speedup against FullAttn.

Baselines. Here are our four baseline categories:

- Accurate attention including FullAttn (Dao, 2024) and ZigZagRingAttn (denoted as ZZRing) (Zhu, 2024);
- Token pruning SlowFast-Llava (denoted as SlowFast) (Xu et al., 2024);
- Sparse attention XAttn (Xu et al., 2025b) and SpargeAttn (Zhang et al., 2025c);
- Approximate attention with sequence parallelism including StarAttn (Acharya et al., 2025) and APB (Huang et al., 2025).

For methods using sequence parallelism (ZZRing, StarAttn, and APB), we employ frame paral-

Method	8-15s	15-60s	180-600s	900-3600s	Overall	P. D.
InternVL3-2B						
FullAttn	61.90	67.44	54.61	50.00	55.35	-
XAttn	62.43	71.51	53.40	48.58	54.97	Yes
SParge	56.08	64.53	50.49	42.55	49.74	Yes
SlowFast	60.85	66.86	50.24	44.50	51.46	Yes
StarAttn	**63.49**	66.86	52.91	48.40	54.30	Yes
APB	61.90	**67.44**	**55.83**	48.76	55.20	Yes
APB-V	62.43	**67.44**	54.85	**49.82**	**55.42**	**No**
Qwen2.5VL-3B						
FullAttn	69.31	70.93	52.18	43.79	53.47	-
XAttn	66.14	69.19	52.18	44.33	53.03	Yes
SParge	59.79	60.47	42.48	43.97	47.87	Yes
SlowFast	65.61	64.53	50.49	46.45	52.73	Yes
StarAttn	**68.78**	70.93	53.64	45.39	54.52	**No**
APB	68.25	**72.09**	52.18	**47.34**	**54.97**	**No**
APB-V	68.25	70.93	**53.88**	44.86	54.30	**No**
Qwen2.5VL-7B						
FullAttn	73.81	73.84	56.44	49.82	58.38	-
XAttn	72.49	73.84	57.77	47.52	57.59	Yes
SParge	65.08	66.86	47.09	40.43	49.37	Yes
SlowFast	72.49	68.60	53.16	49.82	56.47	Yes
StarAttn	74.07	72.67	57.28	50.53	58.26	Yes
APB	75.13	73.84	**58.01**	50.18	59.16	**No**
APB-V	**77.78**	**74.42**	57.77	**50.71**	**59.76**	**No**

Table 2: Accuracies (%) of LongVideoBench. "Overall" stands for complete dataset's accuracy, and "P. D." is the performance degredation compared with FullAttn.

lelism on the visual encoder to ensure a fair competition and set the host number to 8 with 1 GPU in each host. Other methods (FlashAttn, XAttn, SParge, and SlowFast) use a single GPU. We train APB's retaining heads to select KV blocks on NextQA (Xiao et al., 2021) with other settings identical to the official configuration.

Hyperparameters and Environments. Given n input embeddings, we set the anchor length $l_a = \frac{n}{64}$ and the passing length $l_p = \frac{n}{128}$ for APB-V, while APB uses $l_a = \frac{n}{64}$ and $l_p = \frac{n}{64}$. With APB-V's $2H$ virtual hosts, APB-V and APB finally maintain the same amount of compute. When adapting SlowFast to `InternVL-2B`, we remove all the text labels of each frame; we gather the corresponding ids for 3D-RoPE (Qwen-Team, 2025) when adapting to `Qwen2.5-VL`. For SParge, we set the threshold of similarity and cumulative distribution to 0.3 and 0.96, respectively. Other baseline methods use default configurations. All experiments are conducted on an 8×A800-40GB cluster with 3rd-generation NVLink interconnects.

5.2 Benchmarking Task Performance

In Table 1 and Table 2, to investigate (***Q1***), we conduct evaluations of APB-V and competitive baselines on both LongVideoBench and VNBench.

Performance variations are more pronounced on synthetic videos compared with real-world inputs. As shown in Table 1, all methods exhibit some accuracy degradation compared to FullAttn. Among them, XAttn, APB, and APB-V exhibit better performance. SlowFast suffers significant accuracy drops, revealing the limitations of token pruning for complex and long retrieval tasks. StarAttn also underperforms as it cannot capture long-range dependencies. In contrast, APB-V achieves the highest accuracy among all baselines.

For real-world long videos (Table 2), XAttn and SlowFast show significant performance degradation across all models. While StarAttn outperforms sparse attention and token pruning methods, it remains less accurate than FullAttn on `InternVL3-2B` and `Qwen2.5VL-7B`. Both APB and APB-V achieve near-lossless accuracy, with APB-V surpassing APB on two models.

Overall, APB-V achieves better results among approximate attention methods for long-video understanding, surpassing all baselines.

5.3 Benchmarking Inference Speed

To address (***Q2***), we evaluate the speedup of all methods relative to FlashAttn across various resolutions and video lengths, with results shown in Figure 6 and 7. The speedup results show that: (1) sequence parallelism effectively enhances inference speed, where methods without sequence parallelism (XAttn and SlowFast) show significant limited speedup; (2) APB-V outperforms all baselines across all scenarios regardless of base model, video resolution or video length; (3) APB-V exhibits extraordinary speedup for high resolutions or long videos, achieving 12.72×, 1.70×, and 1.18× speedup compared with FlashAttn, ZZRing, and APB, respectively, when processing 64-frame 1440p videos on `Qwen2.5-VL-3B`.

5.4 Ablation Studies

To address (***Q3***), we present an ablation study that highlights the contributions of each optimization, from both the algorithm and system perspectives.

Block Composition. To analyze the impact of anchor block $\mathbf{A}$ and passing block $\mathbf{P}$, we conduct an ablation study on VNBench using the LMM `InternVL3-2B`. Table 3 demonstrates that: removing either component leads to significant performance degradation, with $\mathbf{P}$ particularly crucial for the overall accuracy. This validates the effective-

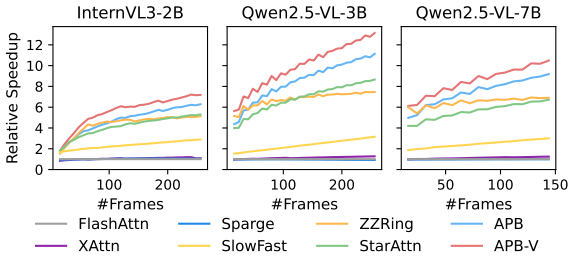

Figure 6: The relative speedup of APB-V and baselines compared to FLASHATTN under various number of frames. We use 720p for `Qwen2.5-VL` models.

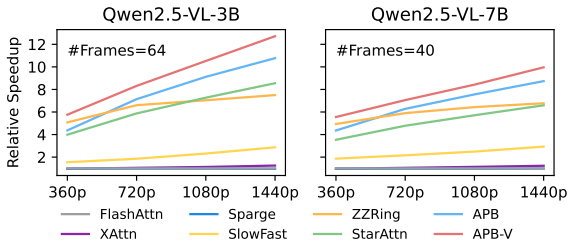

Figure 7: The relative speedup of APB-V and baselines compared to FLASHATTN under various resolutions.

ness of our design on applying an anchor block and a passing block for attention at each host.

System Optimizations. We conduct an ablation study on APB-V's system optimizations to evaluate their effectiveness, focusing on four key components: (1) overlapping communication with computation in attention, (2) fused context-query forward pass, (3) ZigZag-style load balancing, and (4) visual encoding frame parallelism. We systematically disable each optimization and report the inference speed (req / s) in Table 4. We configure the anchor length as $l_a = \frac{n}{64}$ and passing length as $l_p = \frac{n}{128}$, using $H = 8$ physical hosts. For ZigZag ablation, we revert to APB's passing length ($l_p = n/64$). When the frame parallelism is removed, we encode all frames for each host. The ablation results demonstrate that all four system optimizations effectively accelerate the inference process. The fused context-query forward pass and frame parallelism yield the most significant speed improvements. When removing all optimizations, the system degrades to APB-like inference with all frames encoded on each host, remaining faster than FLASHATTN but 4× slower than APB-V. Overall, APB-V exhibits the fastest inference speed.

Host Scalability. To evaluate the host scalability of APB-V in terms of efficiency and performance, we conduct an ablation study comparing it with sequence-parallel baselines under different host counts $H \in \{2, 4, 6, 8\}$. Results are reported in Table 5 and Table 6. Table 5 shows that APB-V achieves the fastest inference across all host counts for most frame numbers, whereas other baselines encounter severe efficiency bottlenecks as frame numbers grow. In terms of performance, Table 6 shows that APB-V maintains stable task accuracy across different host counts.

Since APB-V introduces communication, we further study the impact of different communication media on efficiency. As mainstream GPU clusters are typically connected via NVLINK or InfiniBand (IB), we use $H = 8$ GPUs with NVLINK as the baseline, and then split them into two groups: NVLINK for intra-group links and IB for inter-group links, to measure the drop in inference speed. Thus, the IB link emerges as the bottleneck for the balanced `AllGather` operation. As shown in Table 7, introducing the IB bottleneck degrades efficiency for all methods. APB-V experiences the smallest drop and sustains stable inference speed, owing to its reduced communication design.

Hyperparameter Analysis. we provide a detailed hyperparameter analysis on the anchor length l_a and passing length l_p. The block length is determined by the input length and the number of hosts H. We test APB-V with different hyperparameters on the Ordering-E subset of VNBench, using

Method	Counting			
	E-1	E-2	I	Avg.
w/o **A**	50.00	7.33	27.33	25.33
w/o **P**	21.33	5.33	25.33	17.33
APB-V	**52.00**	**9.33**	**28.67**	**30.00**

Table 3: Ablation study on the anchor block **A** and passing block **P**, using `Qwen2.5VL-3B` as the tested LMM.

Method	#Frames					
	16	24	32	40	48	56
APB-V	**1.846**	**1.050**	**0.916**	**0.658**	**0.594**	**0.471**
-O	1.827	1.049	0.911	0.656	0.593	0.470
-O-F	1.646	0.980	0.854	0.624	0.565	0.450
-O-F-Z	1.618	0.947	0.813	0.590	0.523	0.415
-O-F-Z-V	0.381	0.253	0.189	0.151	0.125	0.107
FLASHATTN	0.226	0.136	0.092	0.068	0.052	0.042

Table 4: Ablation on system optimizations. Inference speed (req/s) of 1440p videos of various lengths are tested on `Qwen2.5VL-3B`. "-" indicates the baseline without certain optimization. "O", "F", "Z", and "V" represent overlapping communication-computation, fused context-query forward, ZigZag load-balancing strategy, and frame parallelism.

Method	#Frames 16	24	32	40	48	56
$H = 2$						
ZIGZAGRING	**2.090**	1.335	0.994	0.779	0.626	0.525
STARATTN	1.401	0.908	0.666	0.519	0.416	0.345
APB	1.788	1.208	0.909	0.726	0.592	0.503
APB-V	2.049	**1.375**	**1.043**	**0.824**	**0.678**	**0.572**
$H = 4$						
ZIGZAGRING	**3.824**	2.418	1.853	1.446	1.155	0.988
STARATTN	2.633	1.792	1.390	1.113	0.903	0.769
APB	3.131	2.227	1.740	1.436	1.176	1.008
APB-V	3.748	**2.608**	**1.981**	**1.628**	**1.327**	**1.125**
$H = 6$						
ZIGZAGRING	4.143	3.410	2.436	1.858	1.671	1.347
STARATTN	3.157	2.605	1.942	1.537	1.379	1.146
APB	3.505	3.070	2.273	1.867	1.702	1.437
APB-V	**4.222**	**3.672**	**2.653**	**2.133**	**1.946**	**1.633**
$H = 8$						
ZIGZAGRING	5.595	3.550	3.143	2.301	2.000	1.666
STARATTN	4.485	2.888	2.571	1.975	1.762	1.493
APB	4.891	3.290	2.985	2.348	2.118	1.766
APB-V	**6.171**	**4.060**	**3.612**	**2.756**	**2.521**	**2.013**

Table 5: Ablation across various host number "H". Inference speed (req/s) of 720p videos of various lengths are tested on `Qwen2.5VL-3B`.

Video Length	H 2	4	6	8
8-15s	68.78	69.84	70.37	68.25
15-60s	69.77	70.93	71.51	70.93
180-600s	51.94	52.91	53.64	53.88
90-3600s	43.79	43.44	44.33	44.86
Overall	53.18	53.63	54.38	54.30

Table 6: Performance under various host number H, where we select $H \in \{2, 4, 6, 8\}$. We evaluate LongVideoBench's performance using `Qwen2.5VL-3B`.

InternVL3-2B. In our original setup, l_p was set to $n/128$, which equals 256 in the VNBench experiment; $l_a = n/64$, which equals 512. This setting is consistent across all samples, since InternVL3-2B resizes input frames to 448×448 and we use 64 frames in the experiment.

For passing length l_p, we conduct experiments with $l_p \in 0, 128, 256, 512$, as well as a "no-compression" setting. Note that due to implementation constraints in our modified FLASHATTN kernel, l_p must be a multiple of 128. Therefore, in the "no-compression" setting, we set l_p to the largest possible value. The number of missing KVs is fewer than 256 per physical host (fewer than 128 on each virtual host). The results in Table 8 show a 1.12× speedup when compressing $\mathbf{B}$ to $\mathbf{B}_c$ with $l_p = n/128$. When the compression rate is higher (i.e., $l_p = 0$ or 128), we observe a notable degradation in model performance. Therefore, we choose

Method	Host Setting (8)	(4, 4)
ZIGZAGRING	1.221	1.179 (-3.44%)
STARATTN	0.959	0.939 (-2.09%)
APB	1.231	1.214 (-1.38%)
APB-V	**1.461**	**1.450 (-0.75%)**

Table 7: Inference speed (req/s) across different communication media. "(8)" denotes 8 GPUs interconnected via NVLINK, while "(4, 4)" indicates two groups of 4 GPUs each, with intra-group NVLINK connections and inter-group communication over InfiniBand. We use `Qwen2.5-VL-7B` as the tested LMM.

l_p	0	128	256	512	No Comp
Ordering-E (%)	12.67	34.00	37.33	40.00	40.67
Throughput (req/s)	5.67	5.63	4.78	4.39	4.26

Table 8: Hyperparameter analysis on l_p.

$l_p = n/128$ as the default setting to strike a balance between efficiency and performance.

For anchor length l_a, we conduct experiments with $l_a \in \{0, 128, 256, 512, 1024\}$. The results in Table 9 show that the sensitivity of l_a is lower than that of l_p. However, when l_a is small (0 or 128), some performance degradation occurs. When l_a is large (1024), the throughput decreases significantly. Therefore, we select $l_p = 512$.

5.5 Case Study

To fully demonstrate how the approximate attention works in our proposed algorithm, we conduct a case study. We select one video from the Retrieval-E subset of VNBench. This video has 256 frames and is a first-person recording taken inside a car, capturing a group of friends on a road trip along a rural country road as shown in Figure 8. A subtitle reading "The secret word is Nick" is inserted from frames 63 to 67. The query for this video is "What is the secret word in this video?". We count how many times each video block is selected into the passing blocks (i.e., the essential KVs) and visualize this frequency as yellow intensity. As shown in Figure 8, the region containing the answer exhibits noticeably higher yellow intensity than other regions, indicating that it is selected into the passing blocks more frequently. Consequently, the relevant evidence of the given query remains consistently available to each host, enabling the model to produce the correct answer.

6 Related Works

Existing methods on optimizing the efficiency of long-video inference can be categorized into two

Figure 8: A case study from VNBench. Yellow intensity indicates how frequently a spatial position is selected into the passing block. Given the text query "What is the secret word in this video?", the correct answer is "Nick". The region corresponding to "Nick" exhibits noticeably higher yellow intensity than other regions.

l_a	0	128	256	512	No Comp
Ordering-E (%)	36.67	35.33	38.00	37.33	39.33
Throughput (req/s)	4.73	4.60	4.77	4.78	4.36

Table 9: Hyperparameter analysis on l_a.

major types: *intrinsic attention optimization* and *explicit input reduction*.

Intrinsic attention optimization. To mitigate the challenge of intense computational demands caused by increasing input sequence lengths, various optimizations target the attention mechanisms of these models. These optimizations can be achieved through KV cache-centric strategies, memory offloading, and approximate attention designs. *KV cache–centric strategies* (Zhang et al., 2023; Li et al., 2024b; Xiao et al., 2024b; Liu et al., 2024; Huang et al., 2024; Kim et al., 2024) focus on reducing the KV cache size to lower memory usage and boost inference throughput. *Memory offloading techniques* (Lee et al., 2024b; Xiao et al., 2024a; Sun et al., 2025) reduce both memory and computation by selectively loading only the most relevant cache blocks onto the GPU for attention computation. *Approximate attention methods* (Jiang et al., 2024; Li et al., 2025; Xu et al., 2025b; Zhang et al., 2025c,b,a) compute only a subset of attention scores to lower computational cost while preserving model performance. Approximate attention methods can also be integrated with sequence parallelism (Acharya et al., 2025; Huang et al., 2025) to further accelerate inference using multi-GPU systems. Recently, trainable approximate attention methods (Yuan et al., 2025; Lu et al., 2025; Gao et al., 2024; MiniCPM-Team, 2025) have also shown potential in achieving both high performance and efficiency.

Explicit input reduction. Accelerating long-video inference can also be accomplished by reducing the number of input tokens before these tokens are processed by the visual encoder or the LLM backbone. *LLM token reduction* methods reduce the number of input tokens before the LLM backbone to alleviate the computational burden. These techniques (Xu et al., 2024; Lee et al., 2024a; Shi et al., 2025; Luo et al., 2025; Wang et al., 2024) typically select the most important or merge video embeddings while minimizing performance loss. Other methods (Xu et al., 2025a; Yao et al., 2025; Yu et al., 2025) perform token reduction-aware multimodal training. Although these methods effectively compress video tokens for the LLM, they fail to address the cost of visual encoding, which remains a non-negligible bottleneck in the long-video inference process. *Video token reduction* (Choudhury et al., 2024; Zhou et al., 2024, 2023; Hao et al., 2024; Jang et al., 2025; Ataiefard et al., 2024) aims to reduce the computational cost of visual encoding by exploiting temporal redundancy and spatial sparsity immediately after video patchification. While effective in accelerating both the visual encoder and the LLM backbone, such methods may suffer from notable performance degradation on complex videos, as they often overlook fine-grained details.

7 Conclusion

We propose APB-V, a sequence parallelism framework featuring an approximate attention mechanism for LMM long-video inference. Through local KV cache compression and passing blocks, APB-V simultaneously reduces communication and computation costs while maintaining long-range dependencies. Carefully crafted system optimizations ensure APB-V's efficiency. Extensive evaluations demonstrate APB-V's superiority, achieving 12.72×, 1.70×, and 1.18× speedup over FLASHATTN, ZIGZAGRING, and APB, respectively, without notable performance loss.

Limitations

Since APB-V primarily optimizes attention mechanisms, our benchmarks focus on decoder-only transformer-based LMMs. Other architectures, such as convolutional networks, are incompatible with the proposed methods. We also restrict APB-V to multi-GPU inference, as on a single GPU it degenerates to FLASHATTN. APB-V mainly focuses on accelerating long-video inference with a scalable number of GPUs under scenarios constrained by end-to-end time (i.e., time-to-first-token, TTFT). A wide range of real-world applications for long-video understanding align with these requirements, e.g., surveillance camera video processing, autonomous driving, etc. Running long-video inference on a single GPU is heavily limited by the total compute power of that GPU. For example, if a video is encoded into 512K tokens for an 8B LMM, the theoretical wall-time for prefilling such a sequence on a single NVIDIA A100 would exceed 5 minutes, which is unacceptable in many scenarios. Therefore, accelerating long-video inference with more compute power, i.e., more GPUs, is the only solution. Therefore, we restrict our discussion to scenarios where multiple GPUs are available.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2024YFB4505603) and National Natural Science Foundation of China (No. 62576186) and a grant from the Guoqiang Institute, Tsinghua University.

References

Shantanu Acharya, Fei Jia, and Boris Ginsburg. 2025. Star attention: Efficient llm inference over long sequences. *Proceedings of ICML*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv:2303.08774*.

Anthropic. 2025. Claude 4 Sonnet.

Foozhan Ataiefard, Walid Ahmed, Habib Hajimolahoseini, Saina Asani, Farnoosh Javadi, Mohammad Hassanpour, Omar Mohamed Awad, Austin Wen, Kangling Liu, and Yang Liu. 2024. Skipvit: Speeding up vision transformers with a token-level skip connection. *arXiv:2401.15293*.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2024. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. *Proceedings of ECCV*.

Yukang Chen, Fuzhao Xue, Dacheng Li, Qinghao Hu, Ligeng Zhu, Xiuyu Li, Yunhao Fang, Haotian Tang, Shang Yang, Zhijian Liu, and 1 others. 2025. Longvila: Scaling long-context visual language models for long videos. *Proceedings of ICLR*.

Rohan Choudhury, Guanglei Zhu, Sihan Liu, Koichiro Niinuma, Kris Kitani, and László Jeni. 2024. Don't look twice: Faster video transformers with run-length tokenization. *Proceedings of NeurIPS*.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. *Proceedings of ICLR*.

DeepSeek-Team. 2024. Deepseek-v3 technical report. *arXiv:2412.19437*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv:2010.11929*.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and 1 others. 2024. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv:2410.13276*.

Google. 2025. Gemini 2.5 Pro.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Xinyue Hao, Gen Li, Shreyank N Gowda, Robert B Fisher, Jonathan Huang, Anurag Arnab, and Laura Sevilla-Lara. 2024. Principles of visual tokens for efficient video understanding. *arXiv:2411.13626*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Proceedings of COLM*.

Yuxiang Huang, Mingye Li, Xu Han, Chaojun Xiao, Weilin Zhao, Ao Sun, Hao Zhou, Jie Zhou, Zhiyuan Liu, and Maosong Sun. 2025. Apb: Accelerating distributed long-context inference by passing compressed context blocks across gpus. *Proceedings of ACL*.

Yuxiang Huang, Binhang Yuan, Xu Han, Chaojun Xiao, and Zhiyuan Liu. 2024. Locret: Enhancing eviction in long-context llm inference with trained retaining heads on consumer-grade devices. *arXiv:2410.01805*.

InternVL-Team. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv:2504.10479*.

Huiwon Jang, Sihyun Yu, Jinwoo Shin, Pieter Abbeel, and Younggyo Seo. 2025. Efficient long video tokenization via coordinate-based patch reconstruction. *Proceedings of CVPR*.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, and 1 others. 2024. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Proceedings of NeurIPS*.

Minsoo Kim, Kyuhong Shim, Jungwook Choi, and Simyung Chang. 2024. Infinipot: Infinite context processing on memory-constrained llms. *Proceedings of EMNLP*.

Seon-Ho Lee, Jue Wang, Zhikang Zhang, David Fan, and Xinyu Li. 2024a. Video token merging for long-form video understanding. *Proceedings of NeurIPS 2024*.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024b. Infinigen: Efficient generative inference of large language models with dynamic {KV} cache management. *Proceedings of OSDI*.

Yanwei Li, Chengyao Wang, and Jiaya Jia. 2024a. Llama-vid: An image is worth 2 tokens in large language models. *Proceedings of ECCV*.

Yucheng Li, Huiqiang Jiang, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and 1 others. 2025. Mminference: Accelerating pre-filling for long-context vlms via modality-aware permutation sparse attention. *Proceedings of ICML*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. Snapkv: Llm knows what you are looking for before generation. *Proceedings of NeurIPS*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *Proceedings of ICML*.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, and 1 others. 2025. Moba: Mixture of block attention for long-context llms. *arXiv:2502.13189*.

Yongdong Luo, Wang Chen, Xiawu Zheng, Weizhong Huang, Shukang Yin, Haojia Lin, Chaoyou Fu, Jinfa Huang, Jiayi Ji, Jiebo Luo, and 1 others. 2025. Quota: Query-oriented token assignment via cot query decouple for long video comprehension. *arXiv:2503.08689*.

MiniCPM-Team. 2025. Minicpm4: Ultra-efficient llms on end devices. *arXiv:2506.07900*.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, and 1 others. 2023. Dinov2: Learning robust visual features without supervision. *Proceedings of TMLR*.

Qwen-Team. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Xiaoqian Shen, Yunyang Xiong, Changsheng Zhao, Lemeng Wu, Jun Chen, Chenchen Zhu, Zechun Liu, Fanyi Xiao, Balakrishnan Varadarajan, Florian Bordes, Zhuang Liu, Hu Xu, Hyunwoo J. Kim, Bilge Soran, Raghuraman Krishnamoorthi, Mohamed Elhoseiny, and Vikas Chandra. 2024. Longvu: Spatiotemporal adaptive compression for long video-language understanding. *arXiv:2410.17434*.

Yumeng Shi, Quanyu Long, and Wenya Wang. 2025. Static or dynamic: Towards query-adaptive token selection for video question answering. *arXiv:2504.21403*.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *Proceedings of ICML*.

Yunlong Tang, Jing Bi, Siting Xu, Luchuan Song, Susan Liang, Teng Wang, Daoan Zhang, Jie An, Jingyang Lin, Rongyi Zhu, and 1 others. 2023. Video understanding with large language models: A survey. *arXiv:2312.17432*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Proceedings of NeurIPS*.

Han Wang, Yuxiang Nie, Yongjie Ye, Deng GuanYu, Yanjie Wang, Shuai Li, Haiyang Yu, Jinghui Lu, and Can Huang. 2024. Dynamic-vlm: Simple dynamic visual token compression for videollm. *arXiv:2412.09530*.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. 2024. Longvideobench: A benchmark for long-context interleaved video-language understanding. *Proceedings of NeurIPS*.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Proceedings of NeurIPS*.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024b. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *Proceedings of ICLR*.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. 2021. Next-qa: Next phase of question-answering to explaining temporal actions. *Proceedings of CVPR*.

Mingze Xu, Mingfei Gao, Zhe Gan, Hong-You Chen, Zhengfeng Lai, Haiming Gang, Kai Kang, and Afshin Dehghan. 2024. Slowfast-llava: A strong training-free baseline for video large language models. *arXiv:2407.15841*.

Mingze Xu, Mingfei Gao, Shiyu Li, Jiasen Lu, Zhe Gan, Zhengfeng Lai, Meng Cao, Kai Kang, Yinfei Yang, and Afshin Dehghan. 2025a. Slowfast-llava-1.5: A family of token-efficient video large language models for long-form video understanding. *arXiv:2503.18943*.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. 2025b. Xattention: Block sparse attention with antidiagonal scoring. *Proceedings of ICML*.

Linli Yao, Yicheng Li, Yuancheng Wei, Lei Li, Shuhuai Ren, Yuanxin Liu, Kun Ouyang, Lean Wang, Shicheng Li, Sida Li, and 1 others. 2025. Timechat-online: 80% visual tokens are naturally redundant in streaming videos. *arXiv:2504.17343*.

Tianyu Yu, Zefan Wang, Chongyi Wang, Fuwei Huang, Wenshuo Ma, Zhihui He, Tianchi Cai, Weize Chen, Yuxiang Huang, Yuanqian Zhao, and 1 others. 2025. Minicpm-v 4.5: Cooking efficient mllms via architecture, data, and training recipe. *arXiv:2509.18154*.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, and 1 others. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *Proceedings of ACL*.

Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. 2025a. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. *Proceedings of ICML*.

Jintao Zhang, Jia Wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. 2025b. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. *Proceedings of ICLR*.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. 2025c. Spargeattn: Accurate sparse attention accelerating any model inference. *Proceedings of ICML*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Proceedings of NeurIPS*.

Zijia Zhao, Haoyu Lu, Yuqi Huo, Yifan Du, Tongtian Yue, Longteng Guo, Bingning Wang, Weipeng Chen, and Jing Liu. 2025. Needle in a video haystack: A scalable synthetic evaluator for video mllms. *Proceedings of ICLR*.

Shangchen Zhou, Chongyi Li, Kelvin CK Chan, and Chen Change Loy. 2023. Propainter: Improving propagation and transformer for video inpainting. *Proceedings of CVPR*.

Xingyu Zhou, Leheng Zhang, Xiaorui Zhao, Keze Wang, Leida Li, and Shuhang Gu. 2024. Video super-resolution transformer with masked inter&intra-frame attention. *Proceedings of CVPR*.

Zilin Zhu. 2024. Ring flash attention. https://github.com/zhuzilin/ring-flash-attention.

A Pseudocode of APB-V's Core Attention

Algorithm 1: APB-V's Core Attention

Input: Host index h, Host number H;
Ancore Block $\{\mathbf{Q}_a, \mathbf{K}_a, \mathbf{V}_a\}$;
Context Block 1 $\{\mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)}\}$;
Context Block 2 $\{\mathbf{Q}^{(2H-1)}, \mathbf{K}^{(2H-1)}, \mathbf{V}^{(2H-1)}\}$;
Query Block $\{\mathbf{Q}_{qr}, \mathbf{K}_{qr}, \mathbf{V}_{qr}\}$.
Output: Attention score of anchor block $\mathbf{A}_a$, context block $\mathbf{A}^{(h)}$, $\mathbf{A}^{(2H-1)}$, and query block $\mathbf{A}_{qr}$.

// Identify Essential KVs
$[\mathbf{S}_h, \mathbf{S}_{2H-1}] \leftarrow \mathbf{Q}_{qr} \left[\mathbf{K}^{(h)}, \mathbf{K}^{(2H-1)}\right]^\top$
// Send Block 1's Essential KVs
$\mathbf{K}_c^{(h)}, \mathbf{V}_c^{(h)} \leftarrow (\mathbf{K}^{(h)}, \mathbf{V}^{(h)}).\texttt{gather}(\texttt{ArgTop}_{l_p}(\mathbf{S}_h))$
$\texttt{handle}_1 \leftarrow \texttt{AsyncAllGather}(\mathbf{K}_c^{(h)}, \mathbf{V}_c^{(h)})$
// Send Block 2's Essential KVs
$\mathbf{K}_c^{(2H-1)}, \mathbf{V}_c^{(2H-1)} \leftarrow (\mathbf{K}^{(2H-1)}, \mathbf{V}^{(2H-1)}).\texttt{gather}(\texttt{ArgTop}_{l_p}(\mathbf{S}_{2H-1}))$
$\texttt{handle}_2 \leftarrow \texttt{AsyncAllGather}(\mathbf{K}_c^{(2H-1)}, \mathbf{V}_c^{(2H-1)})$
// Query Attention
$\mathbf{A}_{qr}^{(h)}, \texttt{lse}^{(h)} \leftarrow \texttt{QueryAttention}(\mathbf{Q}_{qr}, [\mathbf{K}_a, \mathbf{K}^{(h,2H-1)}, \mathbf{K}_{qr}], [\mathbf{V}_a, \mathbf{V}^{(h,2H-1)}, \mathbf{V}_{qr}])$
$\texttt{handle}_3 \leftarrow \texttt{AsyncAllGather}(\mathbf{A}_{qr}^{(h)}, \texttt{lse}^{(h)})$
// Anchor and Block 1's Attention
$\mathbf{K}_p^{(h)}, \mathbf{V}_p^{(h)} \leftarrow \texttt{MakePassingBlock}(\texttt{handle}_1)$
$\mathbf{A}_a, \mathbf{A}^{(h)} \leftarrow \texttt{Attn}([\mathbf{Q}_a, \mathbf{Q}^{(h)}], [\mathbf{K}_a, \mathbf{K}_p^{(h)}, \mathbf{K}^{(h)}], [\mathbf{V}_a, \mathbf{V}_p^{(h)}, \mathbf{V}^{(h)}])$
// Block 2's Attention
$\mathbf{K}_p^{(2H-1)}, \mathbf{V}_p^{(2H-1)} \leftarrow \texttt{MakePassingBlock}(\texttt{handle}_2)$
$\mathbf{A}^{(2H-1)} \leftarrow \texttt{Attn}(\mathbf{Q}^{(2H-1)}, [\mathbf{K}_a, \mathbf{K}_p^{(2H-1)}, \mathbf{K}_p^{(2H-1)}], [\mathbf{V}_a, \mathbf{V}_p^{(2H-1)}, \mathbf{V}_p^{(2H-1)}])$
// Merging Query Attention Result
$\mathbf{A}_{qr} \leftarrow \texttt{Merge}(\texttt{handle}_3)$
return $\mathbf{A}_a, \mathbf{A}^{(h)}, \mathbf{A}^{(2H-1)}, \mathbf{A}_{qr}$

B Details of System Optimizations

B.1 Fused Context and Query Forward

Previous methods, including STARATTN (Acharya et al., 2025) and APB (Huang et al., 2025), perform the prefill stage using two separate forward passes: one for the document context and another for the query. Since queries are typically short, a single forward pass dedicated to the query can become memory-bound and inefficient. To address this issue, we fuse the prefill processes of both the context (video) and the query into a single forward pass, enabling faster prefill execution by reducing memory I/O of model's parameters. Apart from reading the model's parameters twice in existing methods, we concatenate the query block after each host's context block and conduct all linear projections together for both blocks to reduce redundant memory I/O. As described in the inference process section, we conduct an online `softmax` on query attention's partial result $\mathbf{A}_{qr}^{(h)}$ and $\texttt{lse}^{(h)}$ at the end of each attention module to enable the query attention to be fully completed within the same forward pass as the video context.

B.2 Approximate Attention Load Balancing

APB (Huang et al., 2025) introduces imbalanced computation across hosts, as the number of context blocks involved in the attention computation varies from host to host. This imbalance limits overall efficiency, since the attention computation time is ultimately bounded by the last host. To address this issue, we develop a ZigZag-style load balancing strategy inspired by RINGFLASHATTN (Zhu, 2024), enabling more balanced APB computation across hosts.

Given H physical hosts, we instantiate $2H$ virtual hosts and assign virtual hosts h and $2H-1-h$ to physical host h. Since anchor blocks are identical, each host only holds one replica of the anchor block, and so is the query block. In this setup, each host processes one anchor block of length l_a, two context blocks of length l_b, and a total of $2H-1$ passing blocks of length l_p, ensuring an h-independent and equal amount of compute for all physical hosts. The amout of FLOPs of one attention computation on physical host h is

$$2l_a^2 d + 4l_b^2 d + 4(2H-1)l_p l_b d, \tag{7}$$

which is independent of h and identical across all physical hosts.

Since the anchor block introduces a significant amount of compute in the query attention, we also design a load balancing strategy specifically for this stage. As each physical host holds a full copy of the anchor block, we evenly divide the anchor block into H slices. The query on physical host h attends only to the h-th slice of the anchor block. The numerical accuracy is ensured by the online `softmax` technique.

B.3 Overlapping Communication with Computation

Existing methods such as STARATTN (Acharya et al., 2025) avoid communication across hosts and therefore cannot effectively capture long-term dependencies. In contrast, APB (Huang et al., 2025)

addresses long-term dependencies by introducing inter-host communication; however, this comes with additional overhead, as attention computation becomes tightly coupled with communication results. To mitigate this issue, we design an overlapping strategy to eliminate GPU compute bubbles during communication by structuring the attention process in a two-stage manner. Specifically, for the h-th host, we first perform attention over the anchor block $\mathbf{B}_a$ and the local context block $\mathbf{B}^{(h)}$, followed by a second-stage attention over $\mathbf{B}^{(2H-1-h)}$ separately. Since attention computation is highly compute-bound, dividing it into two stages introduces minimal overhead while allowing communication to be overlapped with useful computation.

When the attention mechanism begins, we first compute the multiplication $\mathbf{Q}_{qr}\mathbf{K}^{(h,2H-1-h)\top}$ to obtain the importance scores, followed by gathering the compressed context blocks for virtual hosts h and $2H-1-h$. Once the $\mathbf{B}_c^{(h)}$ and $\mathbf{B}_c^{(2H-1-h)}$ are generated, we perform `AllGather` communication for them while simultaneously computing the query attention. Then, we perform the first-stage attention over the anchor block $\mathbf{B}_a$ and context block $\mathbf{B}^{(h)}$, leveraging the fact that the passing block for virtual host h is received during query attention. Upon receiving the passing block from virtual host $2H-1-h$, we proceed with the second stage of attention for $\mathbf{B}_c^{(2H-1-h)}$. The communication of the partial query attention results and `lse` takes place immediately after the communication of the two passing blocks, ensuring that the merging of the query attention can proceed without delay following the second stage. As a result, all communication steps can be effectively overlapped with computation, without any compute waiting bubbles. The detailed overlapping routine is illustrated in Figure 5.

C Baselines and More Related Works

We first introduce our baselines, followed by a brief discussion on the potentially related methods.

C.1 Baselines

FLASHATTN. This is an accurate attention implementation without sequence parallelism. Since the attention computation must be performed on a single GPU, only data parallelism or pipeline parallelism can be applied during deployment. However, these forms of parallelism do not reduce the TTFT (time-to-first-token) for long-video inference requests.

SPARGEATTN and XATTN. These two methods introduce sparse attention with different sparsity patterns but do not incorporate sequence parallelism. They reduce computation by eliminating redundant attention operations that naturally arise from sparse attention scores, and thus can speed up long-video inference to some extent. However, without sequence parallelism, the computational capability of a single GPU remains insufficient to achieve acceptable TTFT.

SLOWFAST-LLAVA. This method introduces sparsity by pruning video embeddings. The attention module in the LLM backbone remains unchanged, and the speedup is mainly attributed to the reduced number of input video embeddings. However, removing embeddings introduces significant information loss, as the pruned embeddings cannot be recovered, resulting in notable performance degradation.

ZIGZAGRING. ZIGZAGRING serves as a vanilla baseline of sequence parallelism based on ring-style communication. The input context is evenly partitioned across hosts, and a zigzag load-balancing strategy is adopted to balance computation and communication during attention evaluation. For H hosts, $H-1$ communication rounds are required, where each virtual host h receives partial attention results from $h-1$ and sends to $h+1$. The outputs are aggregated using an online softmax. Because ZIGZAGRING does not introduce attention sparsity, it still suffers from dense computation when processing long-video inputs.

STARATTN. STARATTN is an early approach that integrates sparsity into sequence parallelism. Each host attends only to the anchor block (the beginning of the sequence) and its local block, with no communication during document prefill. Query prefill uses online softmax to merge partial results across hosts. STARATTN is fast and scales well with the number of hosts, but it cannot capture long-range dependencies due to its strictly local sparse attention pattern. In contrast, APB-V maintains long-context dependencies through training-free passing block construction and further improves efficiency through system-level optimizations.

APB. APB is proposed to mitigate the performance degradation observed in StarAttn. It incorporates passing blocks into each attention computation and selects essential KVs using a trained retaining head. Unlike APB, APB-V is completely

Method	Retrieval				Ordering				Counting				Overall	Thru.
	E	I-1	I-2	Avg.	E	I-1	I-2	Avg.	E-1	E-2	I	Avg.	(%)	(req/s)
`InternVL3-2B`														
FullAttn	90.00	90.67	36.00	72.22	64.67	24.00	24.67	37.78	40.67	4.67	28.67	24.67	44.89	1.40
FastV	84.00	90.00	**39.33**	**71.11**	27.33	8.67	16.00	17.33	31.33	**5.33**	24.67	20.44	36.30	1.89
APB-V	**90.67**	**89.33**	32.67	70.89	**64.00**	**22.00**	**21.33**	**35.78**	**37.33**	**5.33**	**26.67**	**23.11**	**43.26**	**4.78**

Table 10: Comparing with FastV on VNBench. "E" and "I" represent the edited and inserted data subset.

Method	SG1	SG2	MK1	MK2	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg.	Thru. (tok/s)
FullAttn	100.00	100.00	90.00	90.00	90.00	100.00	64.00	66.00	65.00	75.00	40.00	80.00	4314.89
APB	**100.00**	**100.00**	80.00	85.00	93.75	98.75	64.00	69.00	71.67	**70.00**	**30.00**	78.29	41704.68
APB-V	**100.00**	**100.00**	**90.00**	**90.00**	88.75	**100.00**	**73.00**	**71.00**	**75.00**	**70.00**	**30.00**	**80.70**	**60376.24**

Table 11: Comparing APB-V with APB on RULER.

training-free. Moreover, APB still requires two separate forward passes for the document and the query, and its lack of load-balancing strategy limits efficiency. With carefully designed system optimizations, APB-V achieves significantly higher speed than APB.

C.2 Potentially Related Methods

FastV. FastV (Chen et al., 2024) is a token-pruning method that reduces the number of hidden states after a certain layer of the LLM backbone. Because some hidden states are discarded to improve efficiency, substantial information is lost and the removed content cannot be recovered. In addition, the original FastV uses an H2O-like (Zhang et al., 2023) pruning function that requires access to full attention scores. This design is incompatible with FlashAttn, forcing FastV to fall back to vanilla matrix multiplication and resulting in high GPU memory usage (a limitation shared with H2O). For this reason, we replace it with a SnapKV-like pruning function in the experiments above, otherwise there would be out-of-memory errors.

LongVU. LongVU (Shen et al., 2024) introduces frame reduction, feature fusion, and cross-modal video token pruning techniques to reduce computational cost when processing long video inputs. However, it requires large-scale training and incorporates prior knowledge from DINOv2 (Oquab et al., 2023), making it unsuitable as a plug-and-play method for arbitrary pretrained LMMs. Another potential weakness compared with APB-V is that LongVU removes video tokens during the prefill stage, which may harm multi-turn QA performance. Once irrelevant tokens are pruned in the first turn, they cannot be retrieved in subsequent dialogue.

LlamaVID. LLaMAVID (Li et al., 2024a) is an LMM specifically designed for high compression ratios in long-video processing, encoding each frame into only two tokens for the LLM backbone. This design requires large-scale multimodal pretraining and a carefully crafted pipeline for long-video understanding. In current LMMs, the information flow is typically simple: a ViT encodes long videos into embeddings, a connector compresses them into a compact representation, and the LLM processes them. Because aggressive compression is not always used during pretraining, a plug-and-play method (such as APB-V) offers a more practical solution that works broadly with existing models.

D Supplementary Experiments

D.1 Comparing with FastV

FastV is an efficient inference method designed for LMMs where a SnapKV-like (Li et al., 2024b) pruning function is used to reduce model's KV cache. We test FastV on VNBench using `InternVL3-2B`, with K set to 2 and R set to 0.5. For APB-V we set the host number H to 8. The results in Table 10 show that FastV can accelerate long-video understanding by 1.35×, but at the cost of a substantial performance degredation. In contrast, APB-V achieves both higher performance and greater efficiency at the same time.

D.2 Extending to Long-Context NLP Tasks

APB-V can be applied to any Transformer-based decoder-only model. To demonstrate its transferability, we evaluate APB-V on long-context NLP tasks. We compare APB-V, APB, and FullAttn on RULER (Hsieh et al., 2024) using `Llama-3.1-8B-Instruct` (Grattafiori

et al., 2024), and the results are listed below. We run 20 entries on the following tasks: Single-NIAH-1/2, MultiKey-NIAH-1/2, MultiValue-NIAH, MultiQuery-NIAH, VT, CWE, FWE, and QA1/2, using 8 GPUs. The input length is set to 128K. We also report the prefill throughput for reference. FULLATTN is implemented in FLASHATTN and is executed on a single GPU. The results in Table 11 show that APB-V is also able to handle long-context NLP tasks. APB-V achieves 13.99× and 1.45× speedups compared with FULLATTN and APB, respectively.