APB: Accelerating Distributed Long-Context Inference by Passing Compressed Context Blocks across GPUs

Yuxiang Huang*, Mingye Li*, Xu Han, Chaojun Xiao, Weilin Zhao, Sun Ao, Hao Zhou,

Jie Zhou, Zhiyuan Liu, Maosong Sun

Presenter: Yuxiang Huang

Dept. of CST, Tsinghua University; Dept. of CST, Central South University;

Beijing University of P&T; Wechat AI Tencent Inc.

2025.07.15

Also Mentioning:

- Locret: Enhancing Eviction in Long-Context LLM Inference with Trained Retaining Heads on Consumer-Grade Devices. Yuxiang Huang, et. al.
- APB-V: Accelerating Long-Video Understanding via Sequence Parallelism-aware Approximate Attention. Yuxiang Huang, et al.

At the very Beginning …

Are we able to prefill 512K tokens within 15s?

Similar Question from Zhihu: Prefill 1M tokens in 20s
https://zhuanlan.zhihu.com/p/709928421 (by Hui Qiang)

- Common Scenario:
  - Model: Llama-3.1-8B-Instruct, Float16/BFloat16 precision
  - GPU: A100/A800
  - Task: Context retrieval / Needle-in-a-Haystack Retrieval / Summary

Vanilla Prefill

- Long-context prefill is compute-bounded.
- Calculating the total FLOPs (Floating point operations):

$$L \times \left(4nd^2 + \frac{4}{g}nd^2 + 2n^2d + 6ndI\right)$$

  - n: input length, d: model dimension, I: FFN dim, L: num layers
- 8B model (Llama-3.1-8B-Instruct) with 512K input tokens:
  - **7.94×10^{16} FLOPs**
- A100/A800 can provide **312 TFLOPs per second**
- Prefilling 512K tokens on single GPU: **$254.51\text{s} \approx 4.24\ \text{min}$!**

Unbearable!!! 🙄

Existing methods: Overall

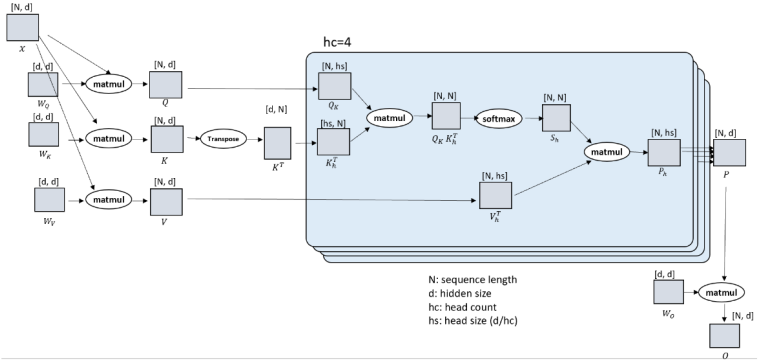
- Existing methods focus on **Reducing Computation** or **Enhancing Parallelism**
- **Reducing Computation: Basically Impossible!!!**
  - Sparsity: $\mathbf{15s / 254s = 6\%}$
  - LLM = Attention + FFN. Both need 6% sparsity.
  - FFN: very hard. **DeepseekMoE-16B: activate 8 of 64 experts = 12.5%**
  - Attention: also very hard. **Deepseek NSA: 8.62% sparsity at 64K input**
- **Enhancing Parallelism: Very Expensive!!!**
  - We need $\mathbf{254s / 15s = 17}$ **A100/A800 GPUs**
  - An A100 GPU worth $20,000. That needs **340,000$**

Existing methods: Combination

- One natural optimization is to combine both of them. But How?

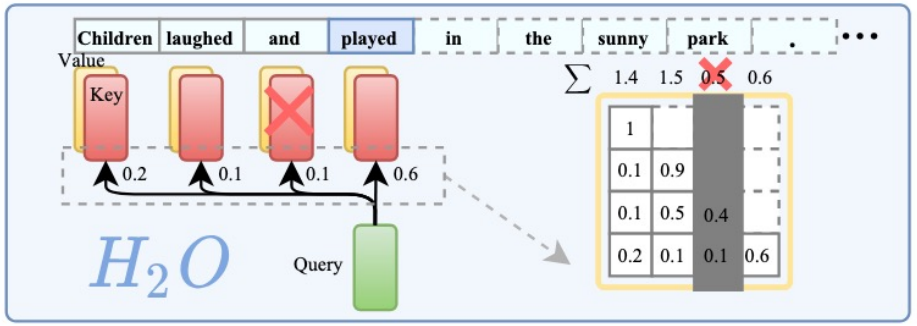
Reducing Computation + Enhancing Parallelism

- **Sequence Parallelism** (Enhancing Parallelism)
  - Designed for long-context training
  - No sparsity at all (Full Attention)



Ulysses (Sequence Parallelism)

- **Sparse/Approximate Attention** (Reducing Computation)
  - Full sequence information → sparse pattern
  - Can't be satisfied in sequence parallelism



H2O (Approximate Attention)

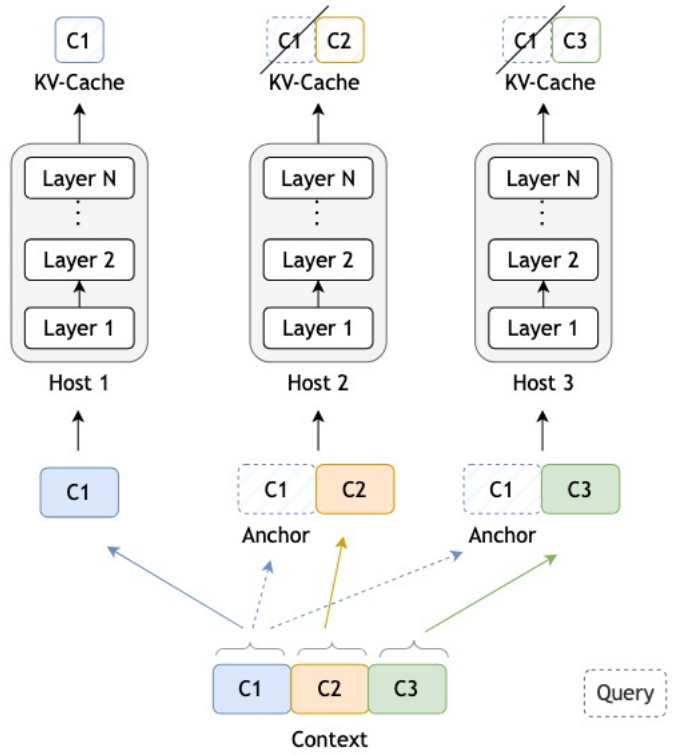
SP-Aware Approximate Attention

Reducing Computation + Enhancing Parallelism

- Initial attempt: **StarAttn (Acharya, 2024)**
- Equally split context to GPUs
- Apply the **Anchor Block**
  - The first context block on GPU0
- Each block only attends to itself and the anchor block

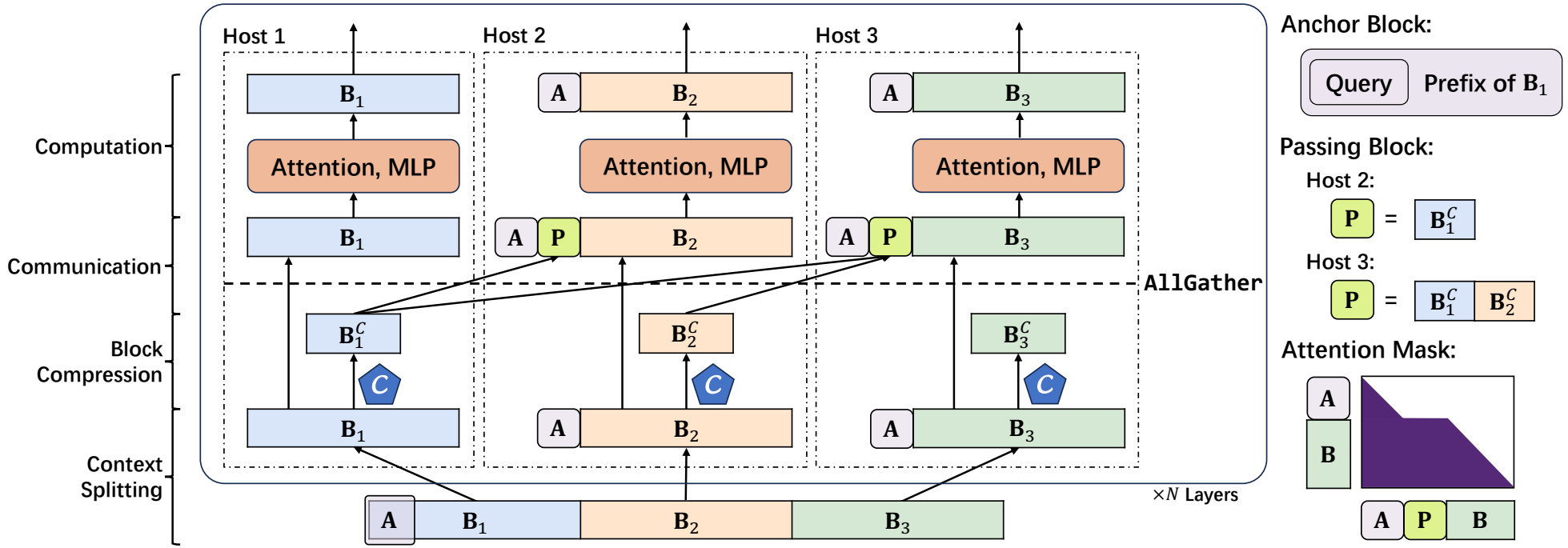
$$\frac{L}{H} \times \left[(8H-4)nd^2 + \frac{8H-4}{g}nd^2 + \frac{8H-6}{H}n^2d + (12H-6)ndI\right]$$

- Total FLOPs: $\mathbf{4.63\times10^{16}}$ **FLOPs**
- Execute on 8 GPUs: $\mathbf{4.63\times10^{17} / (8\times312T) = 18.58s > 15s}$ 😢



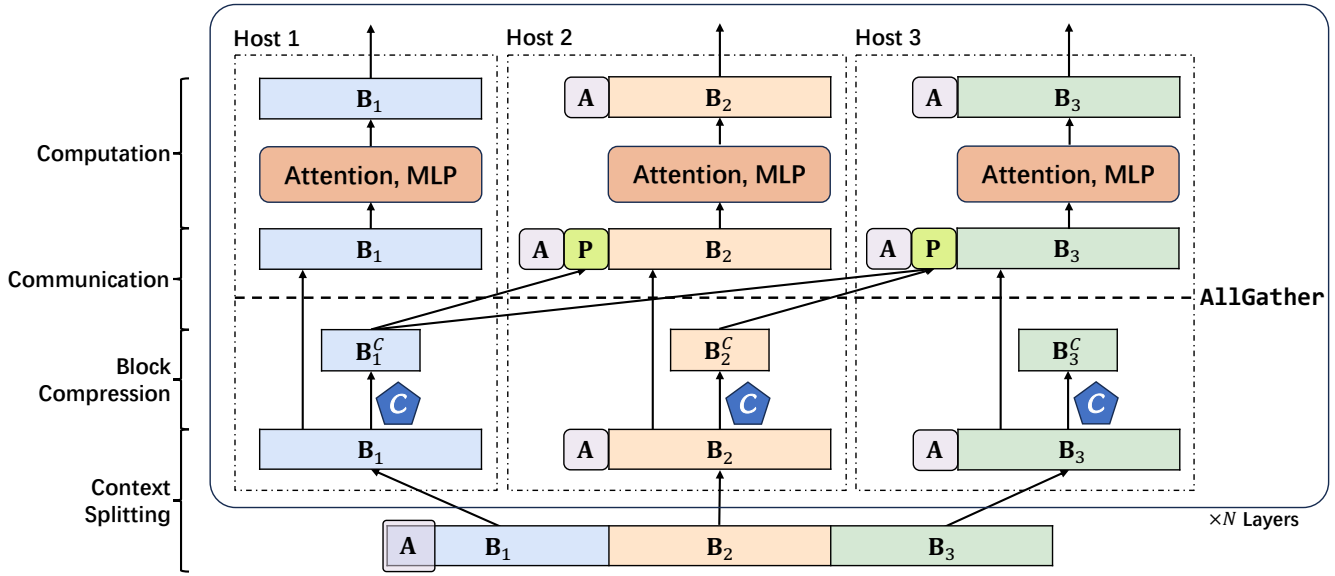
APB Framework: Overall

- Smaller **Anchor Blocks**: 1/4 or 1/8 of StarAttn's → shorter local context length
- Solving long-distance dependencies: **Passing Blocks**
- Local KV compression, light communication
- Provide query information：Embed the query into the head of the anchor block



APB Framework: Inference

- **Context Splitting:**
  - Equally Split to each host
  - Prepend an anchor block with query embedded



Anchor Block:

Query | Prefix of $\mathbf{B}_1$

$$\mathbf{A} = \{q_1, \cdots, q_{l_q}, d_1, \cdots, d_{l_a}\}.$$

$$\{\mathbf{A}, \mathbf{B}_h\} = \{q_1, \cdots, q_{l_q}, d_1, \cdots, d_{l_a}; \quad d_{(h-1)l_b+1}, \cdots, d_{hl_b}\}$$

$$l_a = \frac{1}{4} l_b \text{ or } l_a = \frac{1}{8} l_b$$

APB Framework: Inference

- **Block Compression:**
  - Train retaining heads to score KV pairs
  - Retain important KV pairs to compress the KV cache
- **Communication:**
  - AllGather and form Passing Blocks

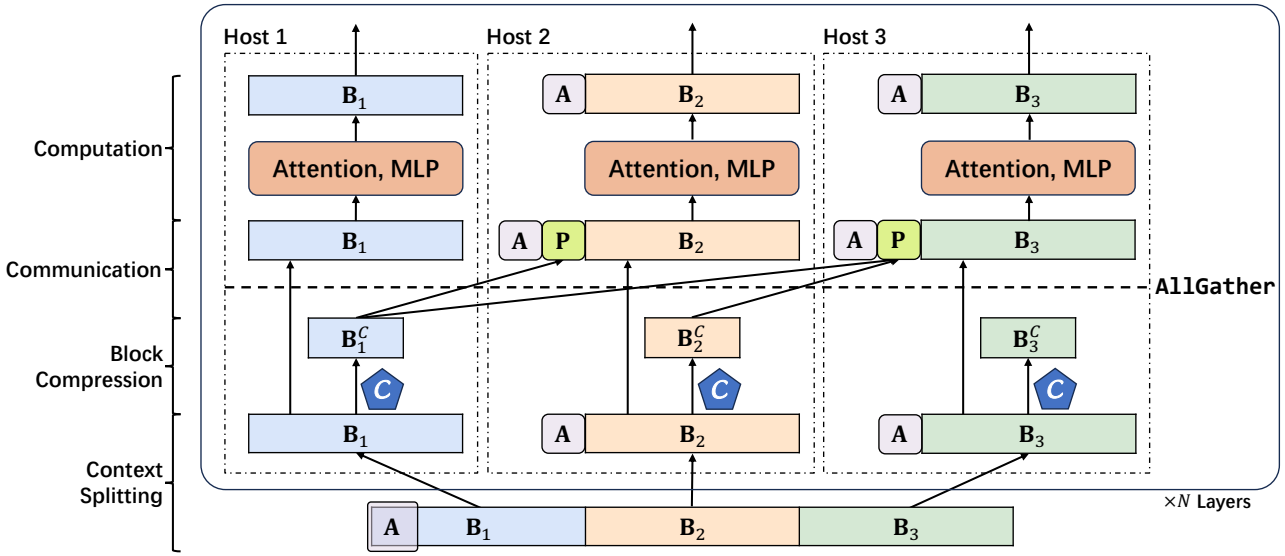
Passing Block:

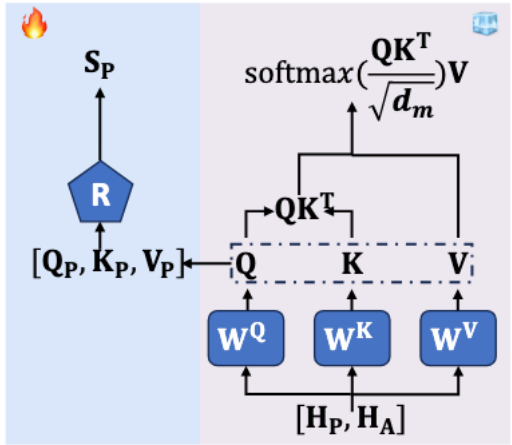
Host 2:

$\mathbf{P} = \mathbf{B}_1^C$

Host 3:

$\mathbf{P} = \mathbf{B}_1^C \; \mathbf{B}_2^C$





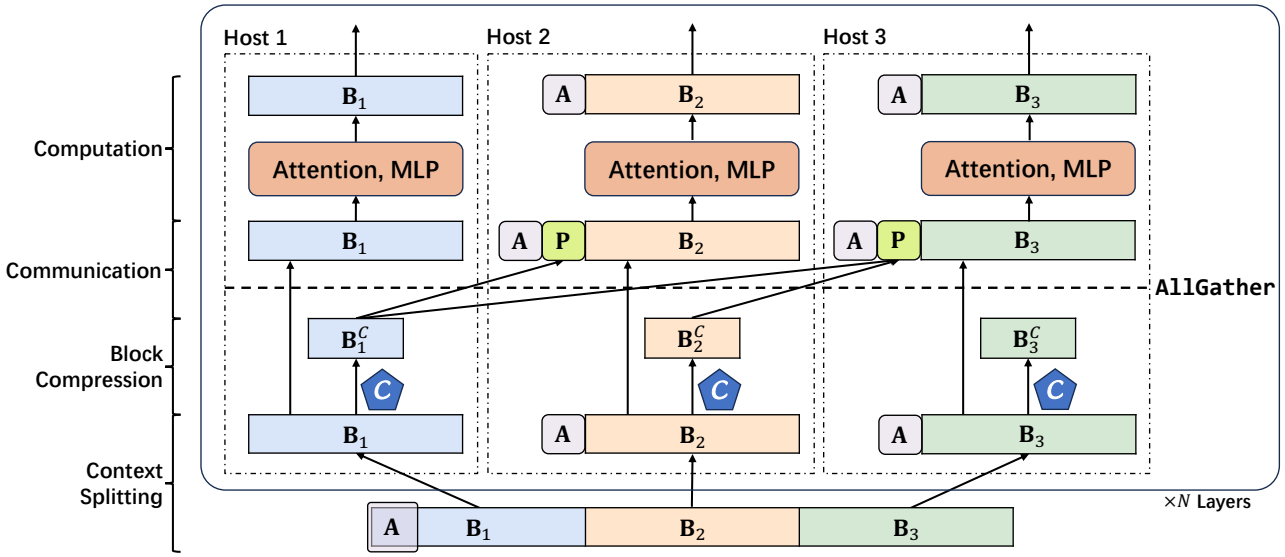
Attention with Retaining Head

Retaining Heads → Locret

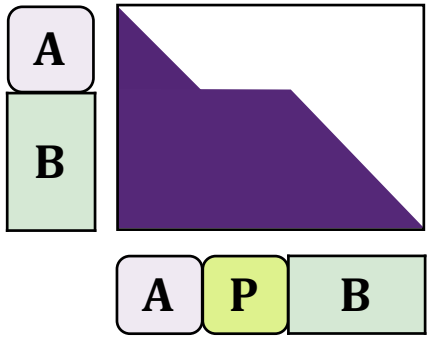
APB Framework: Inference

- **Computation：**
  - Anchor + Passing + Context Blocks
  - using a specific Flash Attn kernel
  - special attention mask
  - Discard passing blocks after attention computation

$$
\begin{aligned}
\mathbf{Q}^{(i)} &= \left[\mathbf{Q}_a^{(i)}, \mathbf{Q}_h^{(i)}\right], \\
\mathbf{K}^{(i)} &= \left[\mathbf{K}_a^{(i)}, \mathbf{K}_p^{C}, \mathbf{K}_h^{(i)}\right], \\
\mathbf{V}^{(i)} &= \left[\mathbf{V}_a^{(i)}, \mathbf{V}_p^{C}, \mathbf{V}_h^{(i)}\right], \qquad (2) \\
\left[\mathbf{A}_a^{(i)}, \mathbf{A}_h^{(i)}\right] &= \texttt{softmax}\left(\mathbf{M}' \odot \frac{\mathbf{Q}^{(i)}\mathbf{K}^{(i)\top}}{\sqrt{d_m}}\right) \cdot \mathbf{V}^{(i)}, \\
\left[\mathbf{H}_a^{(i)}, \mathbf{H}_h^{(i)}\right] &= \texttt{FFN}\left(\left[\mathbf{A}_a^{(i)}, \mathbf{A}_h^{(i)}\right]\right).
\end{aligned}
$$

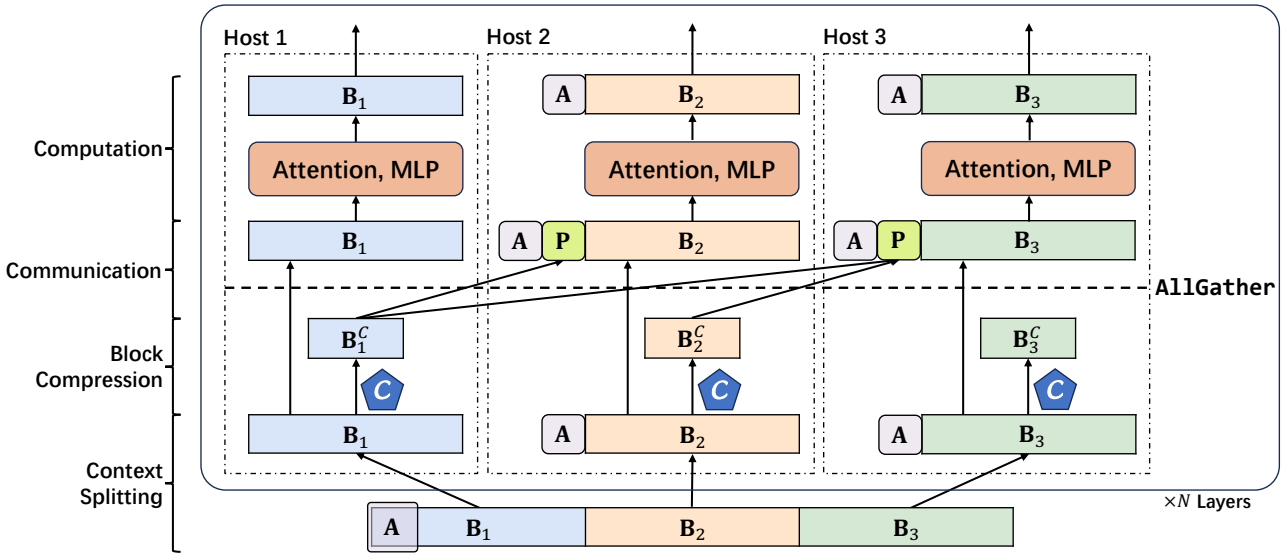


Attention Mask:



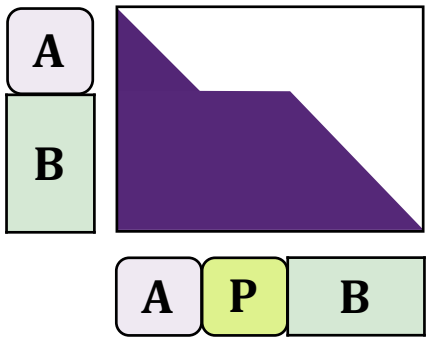
APB Framework: Inference

- **Computation：**
  - Anchor + Passing + Context Blocks
  - using a specific Flash Attn kernel
  - special attention mask
  - Discard passing blocks after attention computation



$$L \times \left[4\left(1+\frac{1}{g}+\frac{0.5n}{Hd}+\frac{1.5I}{d}\right)\frac{n}{H}\bar{d}^2 + 4(H-1)\left(1+\frac{1}{g}+\frac{0.5\left(\frac{n}{H}+l_a\right)}{d}+\frac{1.5I}{d}\right)\left(\frac{n}{H}+l_a\right)\bar{d}^2 + l_pH(H-1)\left(\frac{n}{H}+l_a\right)d \right]$$

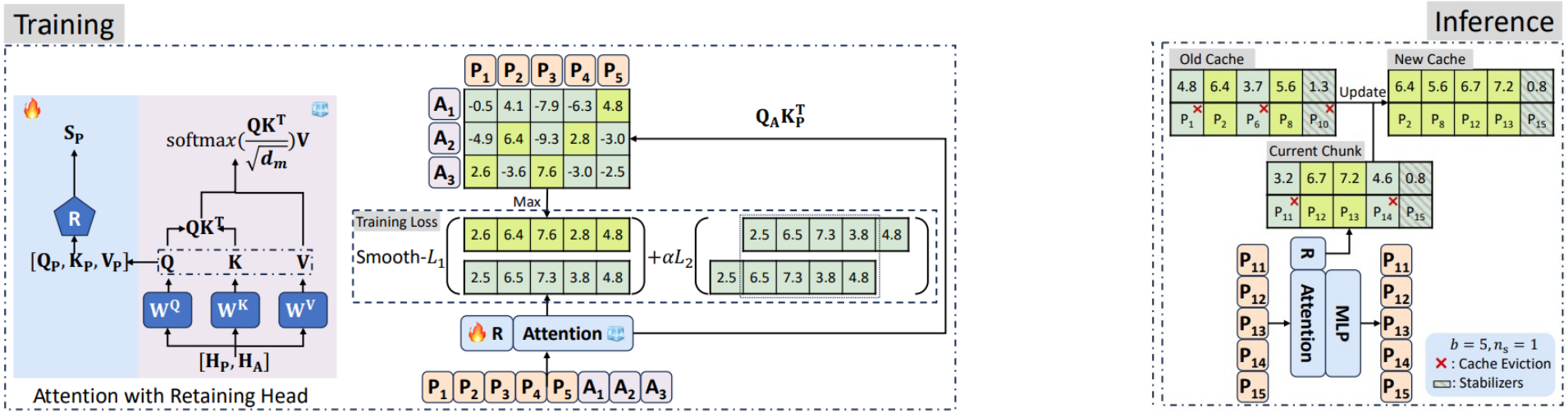
Attention Mask:



- Total FLOPs: 1.88×10^{16} **FLOPs**
- Execute on 8 GPUs: **7.53s** 😊

How Retaining Heads Work

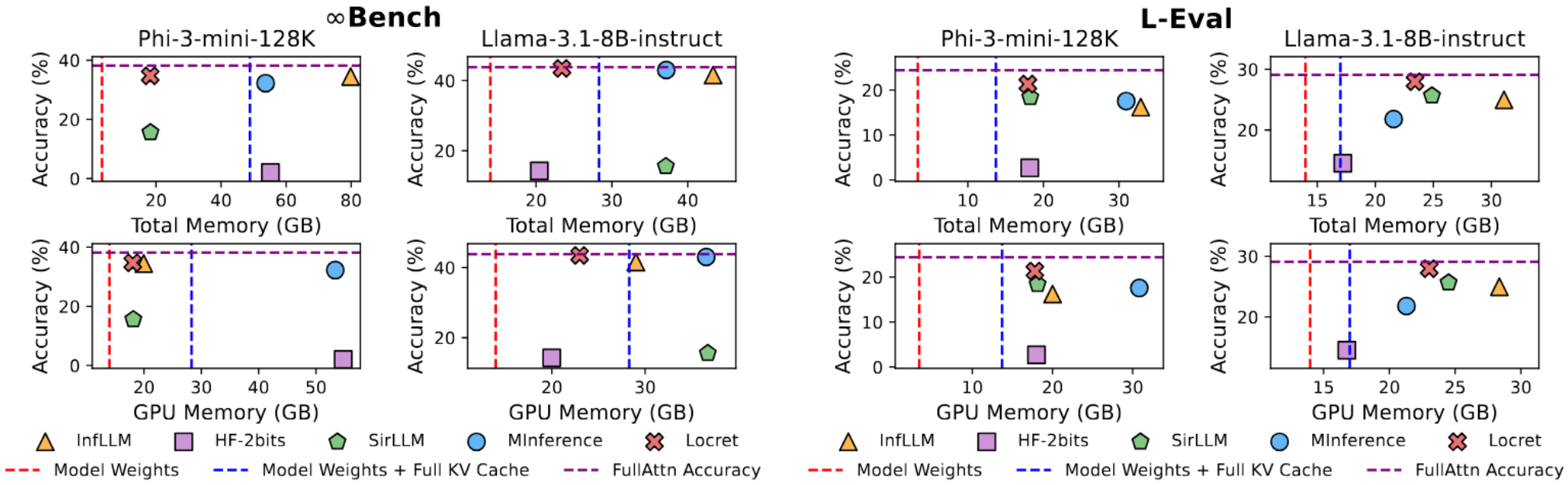
- **Purpose:**
  - Select important KV pairs from each block for compression
- **Input:**
  - Triplet: Q,K,V from the current context block $s_1, \cdots, s_{l_b} = \mathcal{R}([\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h])$
- **Mechanism:**
  - A lightweight MLP outputs importance scores
- **output：**
  - Top-k KV pairs are selected as compressed cache



Locret: Enhancing Eviction in Long-Context LLM Inference with Trained Retaining Heads on Consumer-Grade Devices

Locret: Retaining Heads + Chunked Prefill

- **Key Scenario: Deciding essential KV pairs without full sequence access**
- Chunked Prefill + Eviction: Lower GPU memory usage



Lower GPU memory usage & comparable performance

Locret: Retaining Heads + Chunked Prefill

- **Key Scenario: Deciding essential KV pairs without full sequence access**
- Chunked Prefill + Eviction: Lower GPU memory usage

- Challenging Tasks: RULER
- Needs a small modification
- Putting the query in the front
- Much better than SnapKV

Table 3. Performance, prefill speed, and decode speed on RULER. The best and second-highest scores among eviction-based methods in each column are highlighted in **bold** and underlined. "†" indicates only testing the first 20 entries per subtask due to poor performance. FULLATTN is implemented using FLASH-ATTENTION.

Method	RULER-128K (%)	Prefill (tok/s)	Decode (tok/s)
FULLATTN	82.20	4319.95	12.32
MINFERENCE	72.97	7205.06	2.61
LOCRET-Q	**75.54**	9587.84	40.11
SNAPKV	48.76	4203.34	36.49
$H_2O^\dagger$	15.04	464.73	**44.70**
SIRLLM†	13.23	**9717.41**	40.86
LOCRET†	34.33	9587.09	37.38

Running RULER with KV cache eviction!

Locret: Enhancing Eviction in Long-Context LLM Inference with Trained Retaining Heads on Consumer-Grade Devices

APB Experiment Setting

	Without SP	With SP
Full Attention	FlashAttn	RingAttn Ulysses
Approximate Attention	MInference	StarAttn APB

Performance

FlashAttn: Vanilla; Ulysses, RingAttn: only SP; Minference: only sparse; StarAttn: sparse + SP

Method	R.PassKey	R.Number	R.KV	E.Sum	E.QA	E.MC	E.Dia	Z.QA	C.Debug	M.Find	Avg.
Llama-3.1-8B-instruct											
FullAttn	100.00	99.49	51.00	30.59	29.04	63.76	11.00	36.18	24.62	28.82	47.45
MInference	98.47	98.81	17.40	30.06	26.42	55.46	12.50	36.51	28.17	**32.29**	43.61
StarAttn	**100.00**	**98.98**	40.60	30.55	**30.66**	61.57	**15.00**	36.26	26.65	24.57	46.48
APB	**100.00**	98.81	**81.8**	**30.63**	29.25	**63.32**	11.00	**36.99**	**30.46**	26.86	**50.91**
Qwen-2.5-14B-instruct											
FullAttn	100.00	100.00	17.80	27.80	10.40	52.84	28.00	10.21	38.07	42.57	42.68
MInference	**100.00**	**100.00**	5.20	25.63	12.04	61.57	16.50	11.06	41.62	**48.57**	42.22
StarAttn	**100.00**	**100.00**	18.00	**27.48**	12.44	62.88	23.50	11.03	**43.40**	37.71	43.69
APB	**100.00**	**100.00**	**62.20**	26.51	**12.74**	**66.38**	**25.00**	**11.22**	37.31	35.71	**47.34**
Yi-34B-200K											
FullAttn	100.00	100.00	49.00	5.83	17.57	47.60	2.00	18.77	25.13	28.00	39.39
MInference	**100.00**	**100.00**	52.40	**7.19**	**20.07**	**63.32**	**2.50**	25.40	**28.17**	**31.40**	43.05
StarAttn	**100.00**	**100.00**	24.60	4.38	19.71	60.26	0.00	23.40	28.17	21.14	38.17
APB	**100.00**	**100.00**	**65.20**	5.96	19.81	62.45	1.00	**25.42**	27.92	30.00	**43.78**

Table 1: The results of APB compared with all the baselines on ∞Bench, where higher score represents better performance. "Avg." represents the average score. The highest score in each column is marked in **bold**. FullAttn represents FlashAttn, RingAttn, and Ulysses, as their computational results remain unchanged.

Best Overall Performance on InfiniteBench

Performance

FlashAttn: Vanilla; Ulysses, RingAttn: only SP; Minference: only sparse; StarAttn: sparse + SP

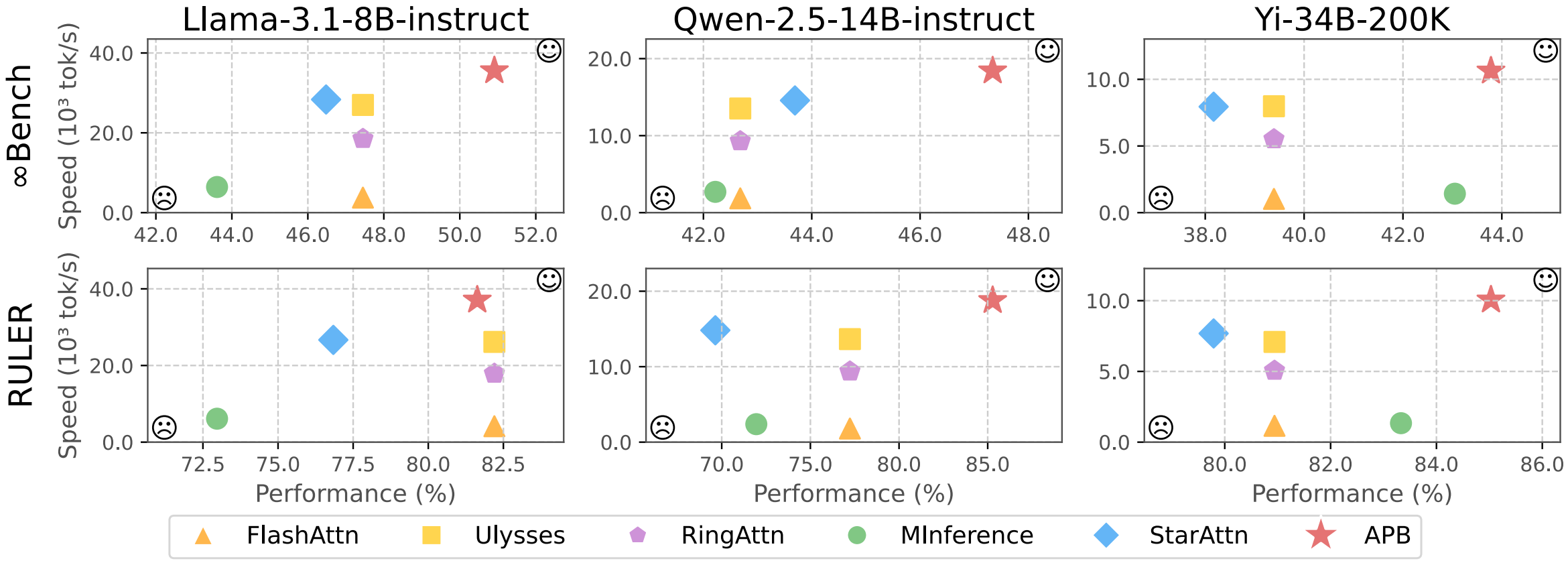
Method	SG1	SG2	SG3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg.
Llama-3.1-8B-instruct														
FullAttn	99.40	99.80	99.60	98.20	87.60	67.00	94.65	98.00	60.98	71.40	72.20	78.20	41.6	82.20
MInference	**100.00**	98.60	99.00	95.40	58.20	23.80	84.35	95.70	**66.40**	45.94	74.67	67.80	38.80	72.97
StarAttn	**100.00**	99.60	99.60	**95.80**	73.60	53.00	72.80	94.45	59.40	**65.72**	76.53	67.40	41.00	76.84
APB	**100.00**	**100.00**	**99.80**	85.60	**91.00**	**89.00**	**95.05**	**96.40**	51.96	63.82	**77.33**	**70.00**	**41.20**	**81.63**
Qwen-2.5-14B-instruct														
FullAttn	**100.00**	99.20	99.80	94.20	47.80	27.20	75.10	94.60	89.52	93.88	76.13	63.20	43.40	77.23
MInference	**100.00**	99.60	**100.00**	89.60	32.60	6.94	61.25	**93.29**	89.44	84.36	77.33	56.60	44.40	71.95
StarAttn	**100.00**	99.20	97.40	80.00	38.00	20.20	48.75	77.10	82.44	93.86	**77.53**	52.80	38.00	69.64
APB	**100.00**	**100.00**	99.80	**93.00**	**87.20**	**85.80**	**66.55**	93.00	**96.40**	**94.94**	76.33	**60.00**	**55.60**	**85.28**
Yi-34B-200K														
FullAttn	100.00	100.00	99.60	95.20	76.00	55.40	92.10	97.05	85.56	51.84	84.27	65.20	50.00	80.94
MInference	**100.00**	**100.00**	**100.00**	**96.00**	87.00	59.60	**90.15**	96.05	73.68	79.20	**84.87**	**67.40**	49.40	83.33
StarAttn	**100.00**	**100.00**	99.60	93.20	78.20	48.00	77.05	88.35	**83.96**	80.58	78.33	61.20	48.80	79.79
APB	**100.00**	**100.00**	**100.00**	92.80	**88.00**	**92.60**	86.05	**96.45**	60.48	**88.46**	84.53	63.80	**52.20**	**85.03**

Table 2: The results of APB compared with all the baselines on RULER, where higher score represents better performance. "Avg." represents the average score. The highest score in each column is marked in **bold**. FullAttn represents FlashAttn, RingAttn, and Ulysses, as their computational results remain unchanged.

Best Overall Performance on RULER

End-to-End Benchmark

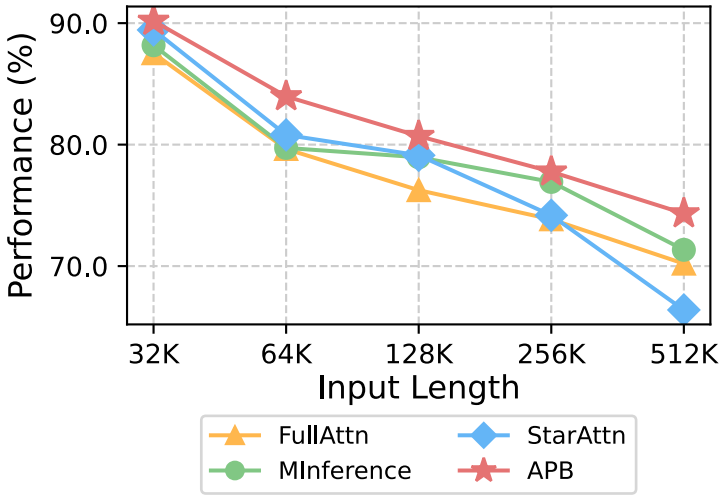
FlashAttn: Vanilla; Ulysses, RingAttn: only SP; Minference: only sparse; StarAttn: sparse + SP

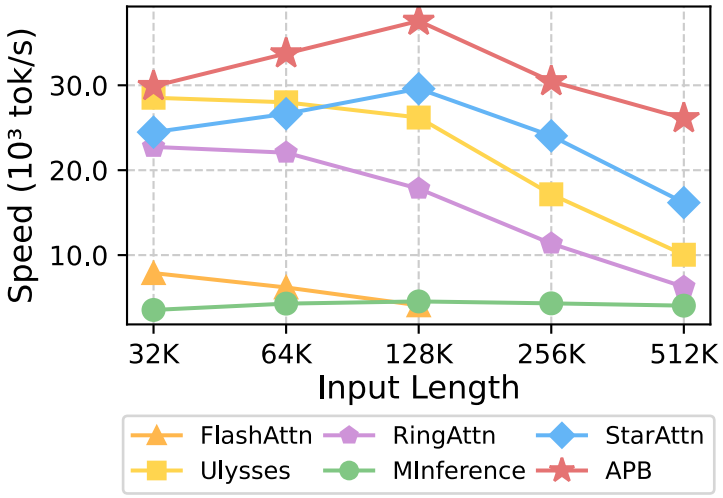


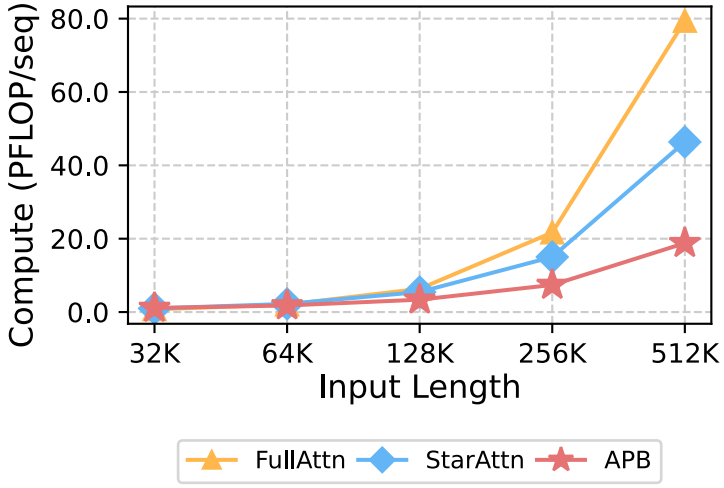
APB Achieves best performance-speed balance!

Various Input Lengths

FlashAttn: Vanilla; Ulysses, RingAttn: only SP; Minference: only sparse; StarAttn: sparse + SP







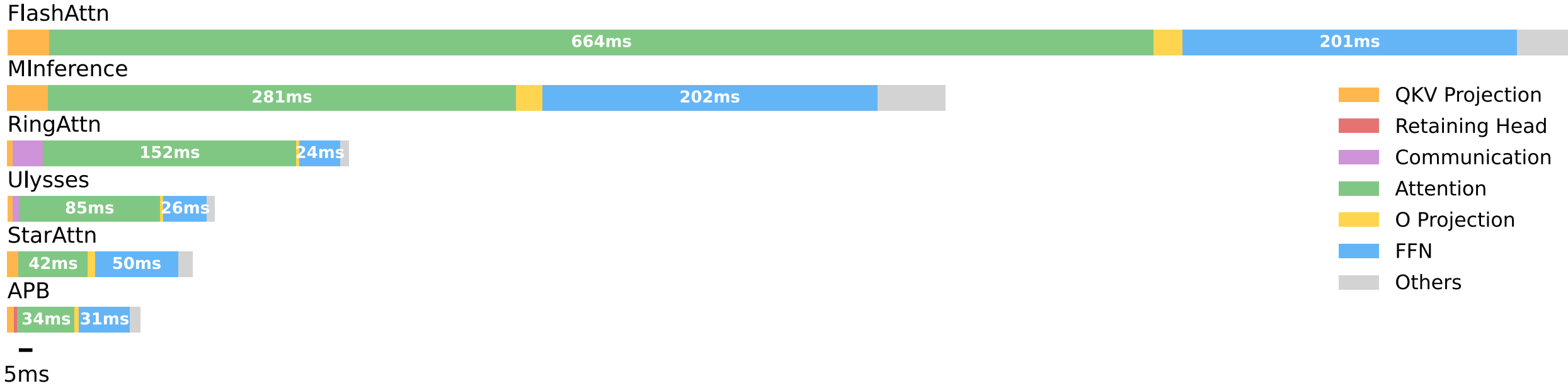
Best performance

Highest Speed

Lowest Compute

Wall-Time Breakdown Analysis

FlashAttn: Vanilla; Ulysses, RingAttn: only SP; Minference: only sparse; StarAttn: sparse + SP



Both faster Attention & faster FFN!

Ablation Studies

- All components are effective (1, 2, 4, 6)
- Anchor block is critical (6, 7)
- Removing passing block causes 8% performance degredation (4, 5)
- A good compressor is essential (2)
- Putting the query in the front is important (1, 2, 3)

No.	**A**	**P**	$\mathcal{C}$	Q	E.MC
0	✓	✓	$\mathcal{R}$	✓	**72.00**
1	✓	✓	$\mathcal{R}$	✗	68.00
2	✓	✓	Rd.	✓	66.00
3	✓	✓	Rd.	✗	66.00
4	✓	✗	Rd.	✓	64.00
5	✓	✗	Rd.	✗	62.00
6	✗	✓	$\mathcal{R}$	✗	28.00
7	✗	✓	Rd.	✗	28.00
8	✗	✗	Rd.	✗	22.00

Table 3: Ablation studies of APB on E.MC. "**A**" and "**P**" represent the anchor block and passing block. We implement the compressors $\mathcal{C}$ as the retaining heads $\mathcal{R}$ or random selectors "Rd.". "Q" indicates embedding the query in the anchor block.

Short context & Quantization

Method	RULER-4K	Speed(tok/s).
FLASHATTN	94.54	6247.58
APB	**94.89**	**6597.47**

Table 5: The task performance and inference speed of APB and FLASHATTN on RULER, where context length set to 4K tokens. We report the speed in "tok/s".

Method	Original (M)	Quantized (M-4bits)
FULLATTN	82.20	81.73
MINFERENCE	72.97	-
STARATTN	76.84	-
APB	81.63	79.13

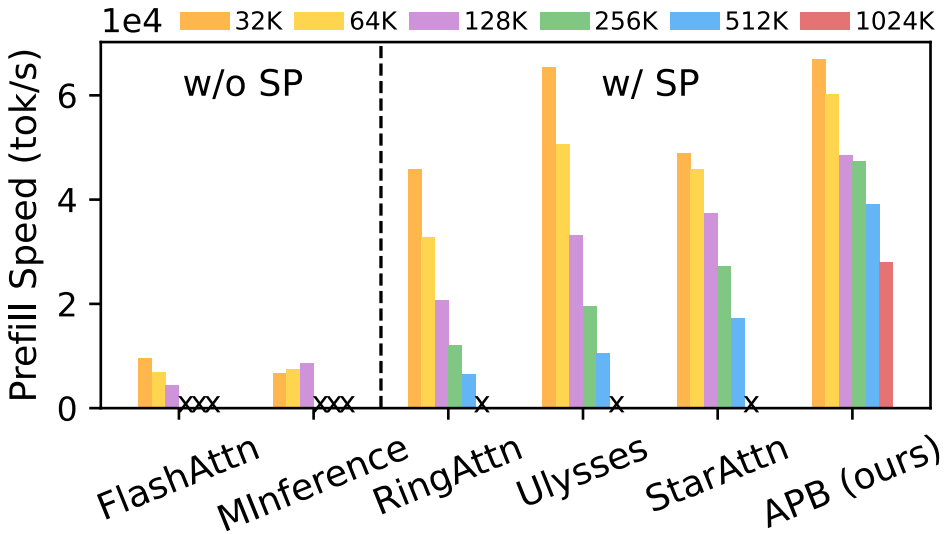
Table 6: Performance of FULLATTN, MINFERENCE, STARATTN, APB, and their quantized variants on RULER-128K. "M" denotes the original method and "M-4bits" represents its 4-bit quantized version. Since MINFERENCE and STARATTN cause more degradation than quantization, we skip their quantized versions.

Short Context Friendly

Quantization Friendly

Takeaways

- Reducing Computation + Enhancing Parallelism
  - APB is an attention designed for sequence parallelism framework
- APB achieves up to 9.2x, 4.2x, 1.6x speedup compared with FlashAttn, RingAttn, and StarAttn.
- Are we able to prefill 512K tokens in 15s?
  - Yes!
  - Theoretically: **7.53s**
  - Measured result: **13.39s < 15s** (Unavoidable costs: synchronization, etc.)



APB-V: Multimodal, Faster, Stronger

Can we use APB to accelerate multimodal models?

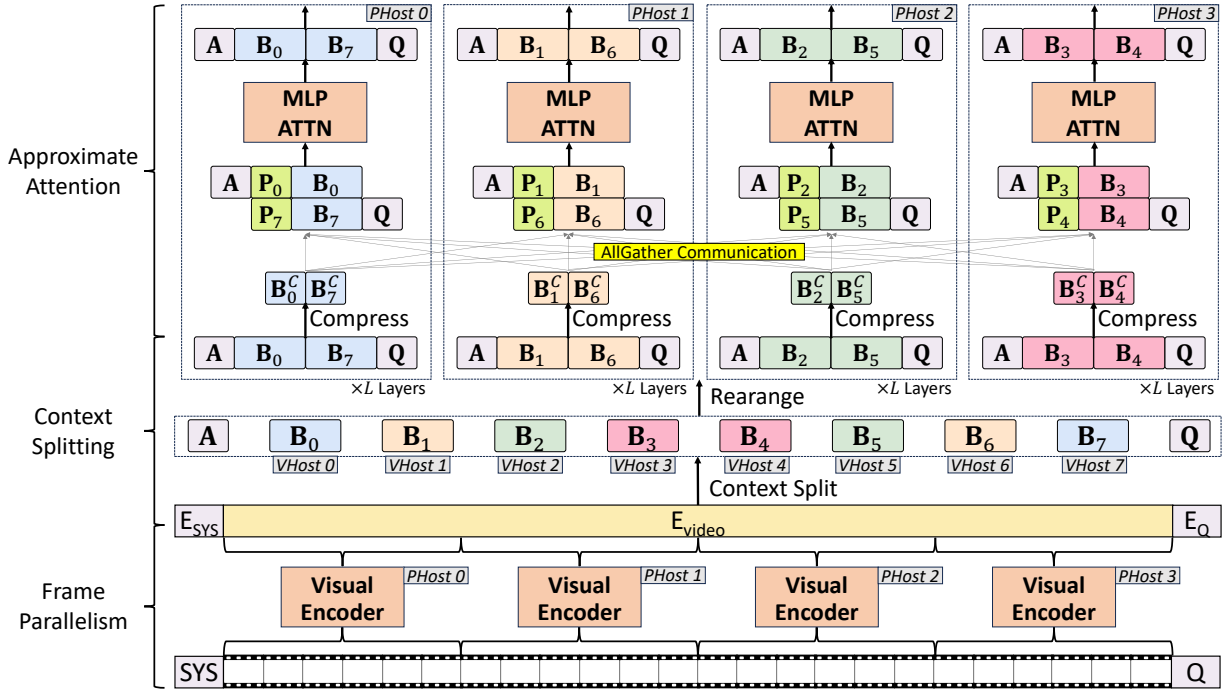
Can APB be even faster?

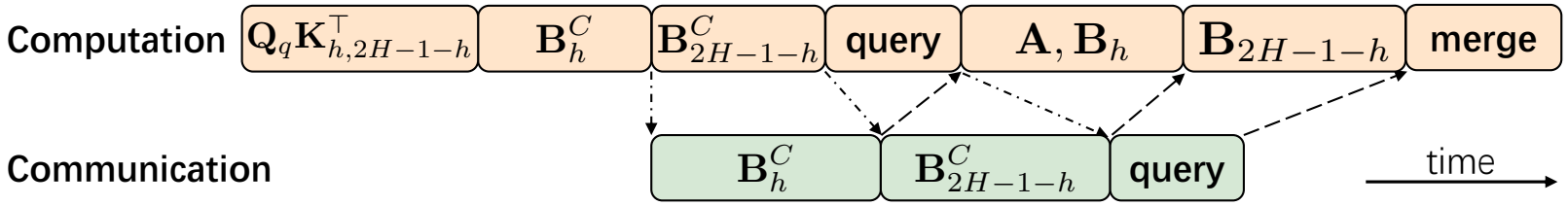
We are working towards it...

- Distribute the computation of the ViT module evenly across all GPUs
- Load Balancing by ZigZag
- Optimize the pipeline to fully overlap communication and computation
- Will release in early August!

APB-V: Multimodal, Faster, Stronger

- Training Free
- Frame Parallelism
- ZigZag style APB for load balancing
- Fused forward pass of document and query
- Overlapping communication and computation





APB-V: Multimodal, Faster, Stronger

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
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
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
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 1: LongVideoBench results of APB-V and the baselines. Higher score indicates better performance, "Overall" stands for the accuracy on the complete dataset, and "P. D." represents performance degredation compared with FullAttn. The highest score in each column is marked in **bold**.

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
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
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
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 2: VNBench results of APB-V and all the baselines. Higher score indicates better performance, and "Overall" stands for the overall accuracy on the complete dataset. The highest score in each column is marked in **bold**.

Training Free & Better Performance!

APB-V: Multimodal, Faster, Stronger

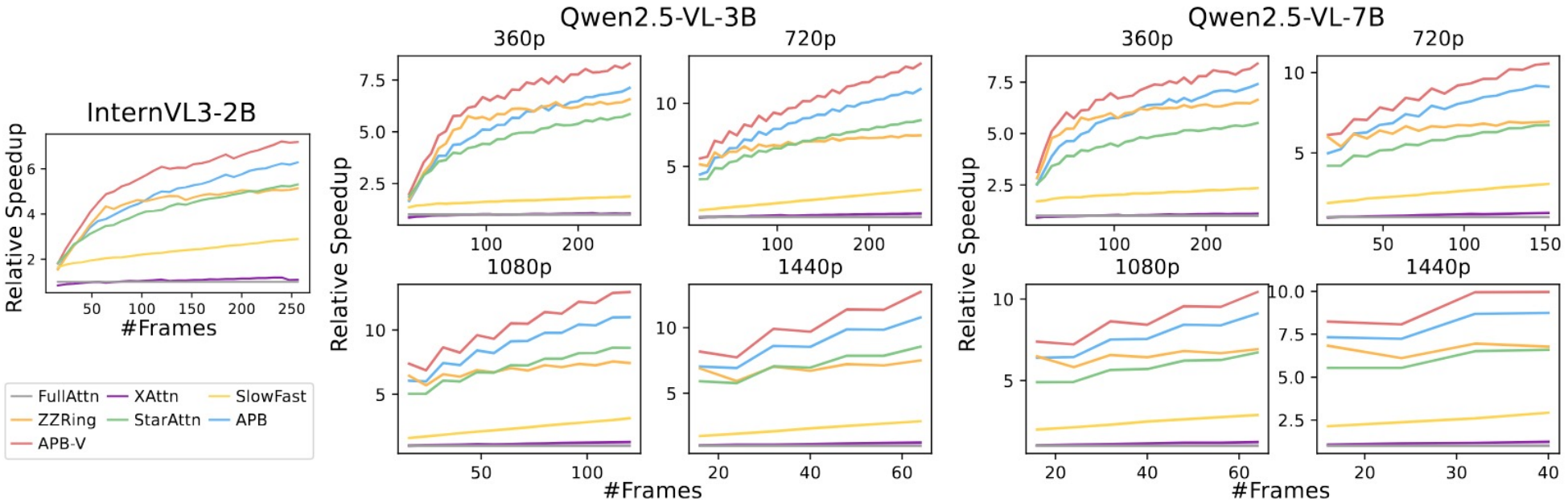


Figure 3: The relative speedup of APB-V and baselines compared to FULLATTN, evaluated across four input resolutions: 360p, 720p, 1080p, and 1440p. Note that `InterVL3-2B` resizes all inputs to 448×448, so resolution has no impact on speedups.

Faster on all resolutions and length!


Thanks

Q&A