ICML International Conference On Machine Learning

Long-Context Modeling with Dynamic Hierarchical Sparse Attention for Memory-Constrained LLM Inference

Siheng Xiong[1], Joe Zou[2], Faramarz Fekri[1], Yae Jee Cho[2]

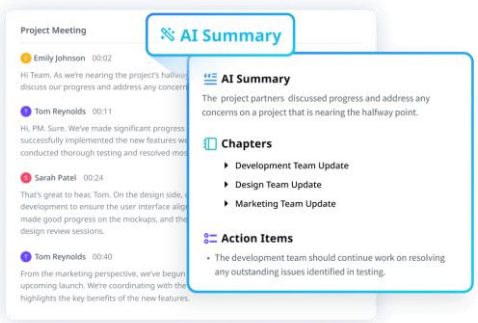
[1]Georgia Institute of Technology, [2]Google

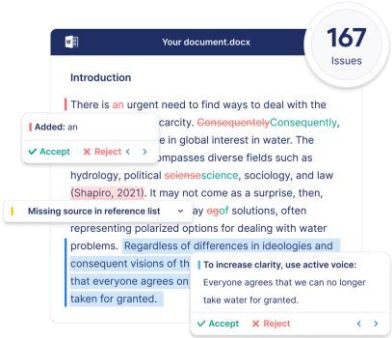
Paper

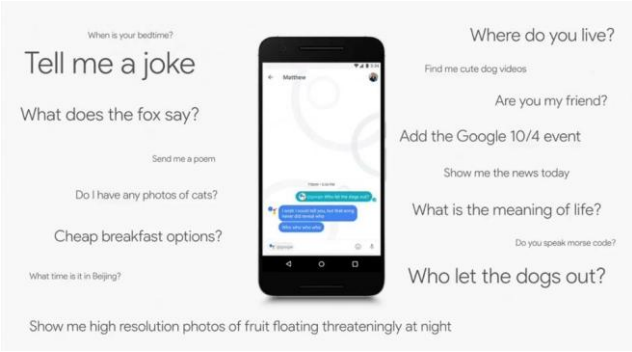
Code

Long Context Modeling for Memory-Constrained LLM Inference

- On-device LLM applications often need to process large amounts of text, for example:
  - Proofreading & summarizing long documents
  - Searching & retrieving meeting transcripts
  - Analyzing personal histories (emails, messages)
  - Maintaining extended context in personal assistants







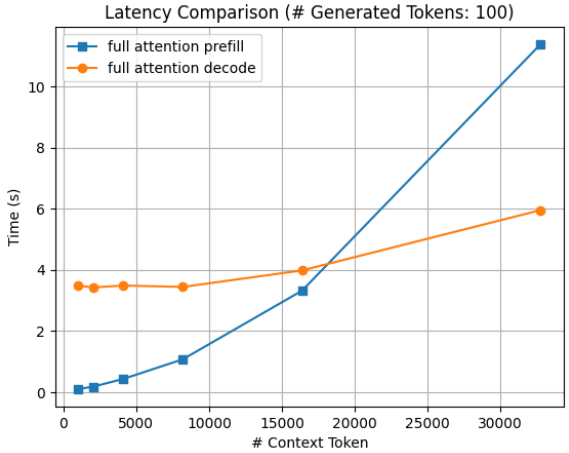
Challenges of Long-Context LLM Inference under Memory Constraints

Naive dense attention becomes impractical for long-context on-device LLM inference.

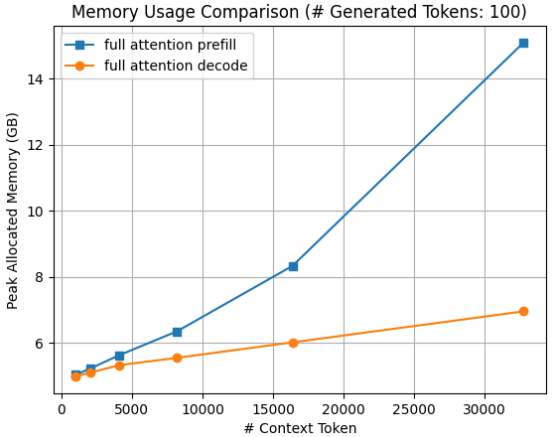
- **Quadratic Attention Cost**: latency & memory usage grow rapidly with context length L.

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^{\cdot}}{\sqrt{d_k}}\right)V \qquad \text{Time complexity: } O(L^2 \cdot d)$$

- **Memory Constraints:** limited VRAM, RAM, compute.



Prefilling latency grows rapidly with context length.

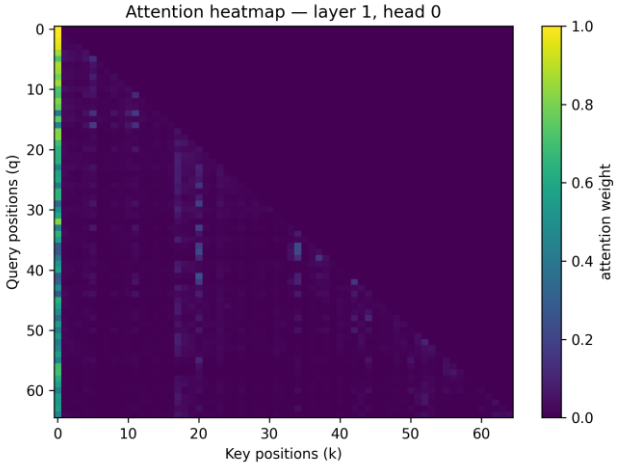


Prefilling memory usage increases sharply.

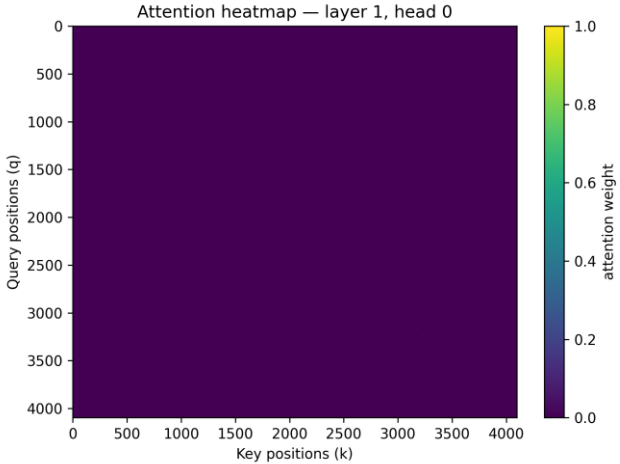
Gemma2-2b-it, bf16, torch.sdpa

Sparse Attention for LLMs

Attention matrices are sparse. Skipping unimportant interactions can preserve accuracy while reducing cost.



Short context → denser attention



Long context → sparser attention

Gemma-2-2b-it,
bf16, torch.sdpa

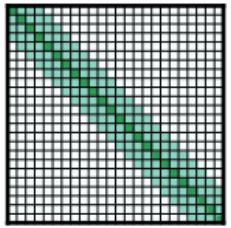
Longer contexts naturally become sparser → opportunity for selective computation.

Limitations of Existing Methods:

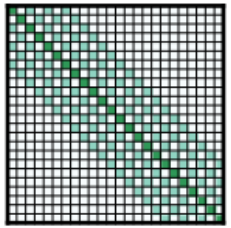
Fixed sparsity patterns:

Static patterns: Longformer, BigBird, streamLLMs

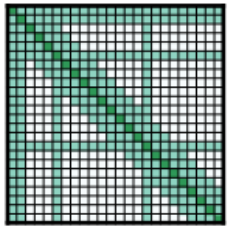
Predefined templates / heuristics: Minference, H2O, Scissorhands, Pyramid KV



Sliding window



Dilated sliding window



Global + Sliding window

Goal of the Project:

- **Enable efficient long-context inference** under memory constraints **without retraining** the base model.
- **Dynamically select and attend to the most relevant tokens** instead of computing the full attention matrix.
- **Maintain accuracy comparable to dense attention** while significantly reducing:
  - **Latency** in prefilling
  - **Memory usage** to fit constraints

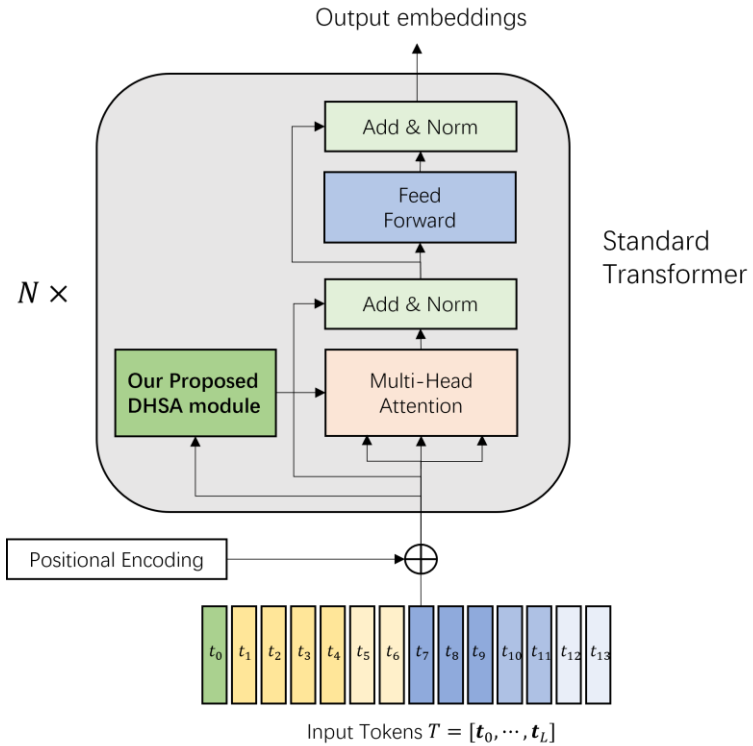
Dynamic Hierarchical Sparsity Attention (DHSA)

- **Role:** DHSA is a plug-in module integrated inside each Transformer layer.
- **Input:** The token embeddings at the current layer.
- **Output:** Selected Key Token Indices.
- **Mechanism**: Guides the multi-head attention to skip unimportant token pairs.

Q

$[0]$
$[0, 1, 2, 3, 4]$
$[0, 1, 2, 3, 4, 5, 6]$
$[0, 1, 2, 5, 6, 7, 8, 9]$
$[0, 1, 2, 7, 8, 9, 10, 11]$
$[0, 1, 2, 3, 4, 7, 10, 11]$

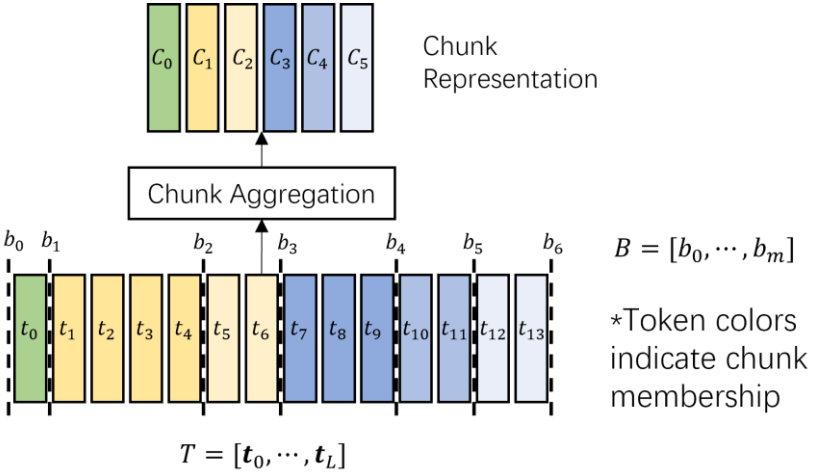
K



$T = [t_0, \cdots, t_L]$: Input tokens
N: Number of layers

Core Idea of DHSA

- Leverage **chunk-level similarity** to guide **token-level sparsity** prediction.

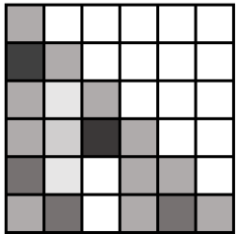


$B = [b_0, \cdots, b_m]$

*Token colors indicate chunk membership

Chunk-level Similarity

$S_C = Q_C K_C^T$

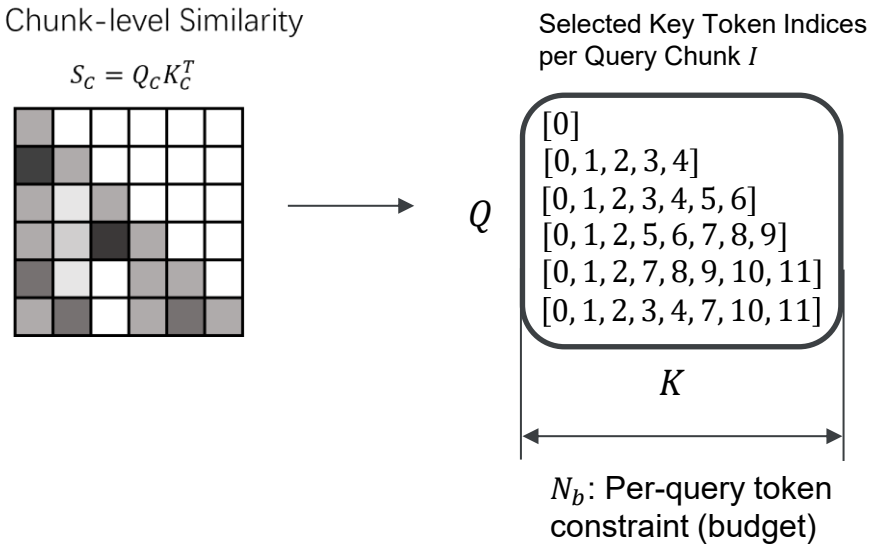


$B = [b_0, \cdots, b_m]$: Boundary indices
$C_0, C_1, \cdots, C_{m-1}$: Chunks

Q_c : Chunk query matrix
K_c: Chunk key matrix
S_c: Chunk-level similarity

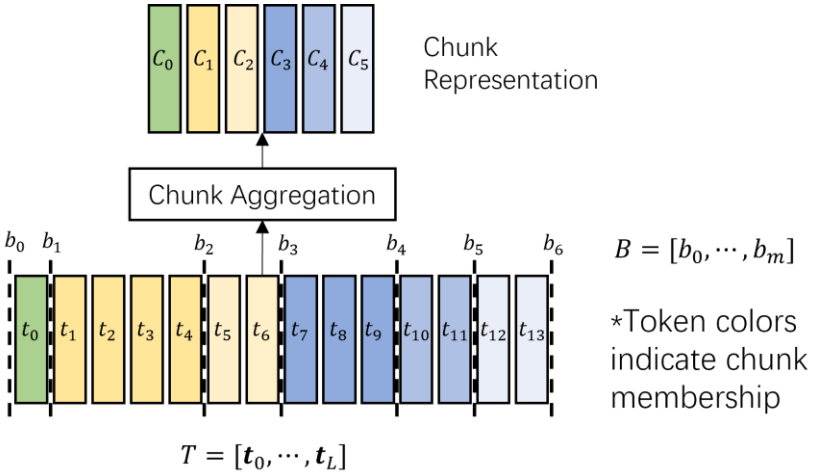
Core Idea of DHSA

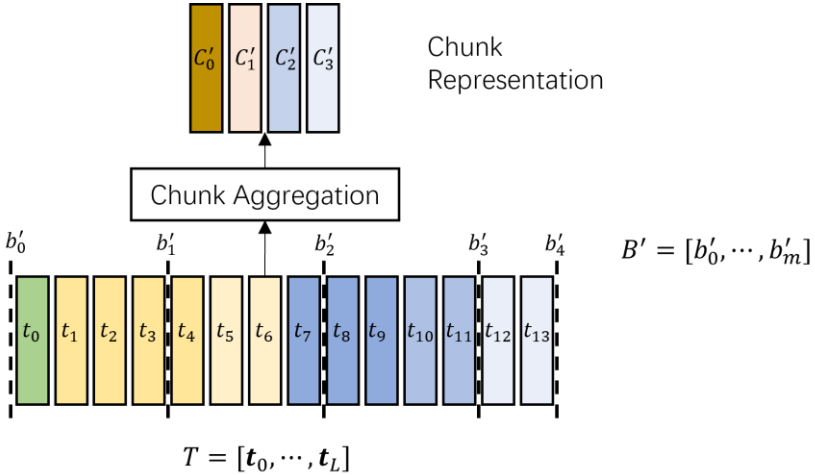
- Leverage **chunk-level similarity** to guide **token-level sparsity** prediction.



Our Contributions

- Dynamic Chunking with Boundary Prediction

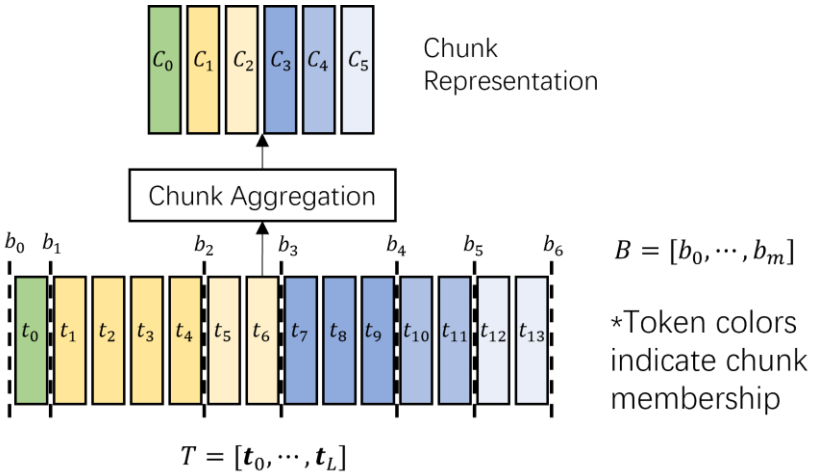




Why is static chunking not enough?

Our Contributions

- Robust Chunk Representation with Variable Sizes



Why is average pooling not enough?

It is sensitive to chunk length.

-> Important tokens can be buried by many low-importance ones, weakening the overall representation.

Computational Cost Comparison

Method	Complexity	Notes
Dense Attention	$O(L^2)$	Every token attends to all tokens
Block Sparse Attention	$O(L \cdot N_b + N_c^2)$	Only attends within fixed-size blocks
DHSA	$O(L \cdot N_b + L + N_c^2)$	Adds boundary prediction cost

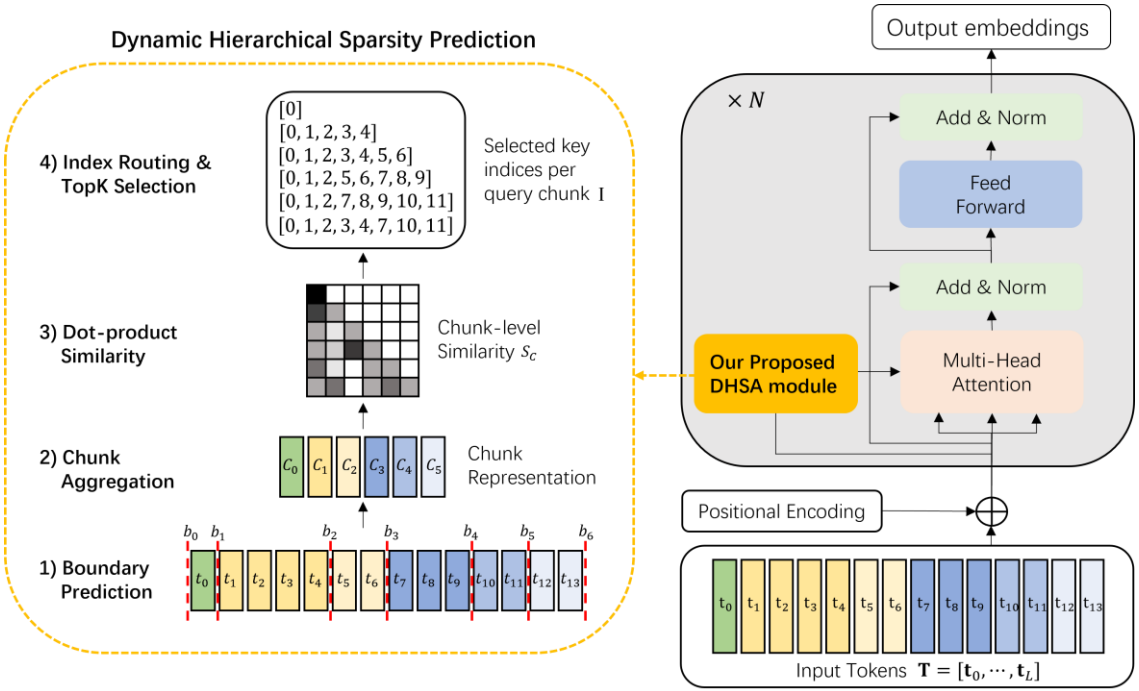
Static chunking + average pooling -> Block Sparse Attention
L: Context length
N_b: Per-query token constraint (budget)
N_c: Number of chunks

Cost:

Block Sparse Attention < DHSA < Dense Attention

Overview of DHSA

Given the **token embeddings** at the current layer, DHSA first predicts **boundaries** and obtains **chunk representations**, then computes **chunk-level similarities** to generate **selected key token indices**.



Dynamic Chunking with Boundary Prediction

- Problem Formulation
- Architecture of Boundary Predictor
- Automatic Labelling
- Inference of Boundary Predictor

Problem Formulation: Boundary Classification

Input: Local keys $[\boldsymbol{k}_{i-w+1}, \cdots, \boldsymbol{k}_i]$, $[\boldsymbol{k}_{i+1}, \cdots, \boldsymbol{k}_{i+w}]$ (w is the receptive field)

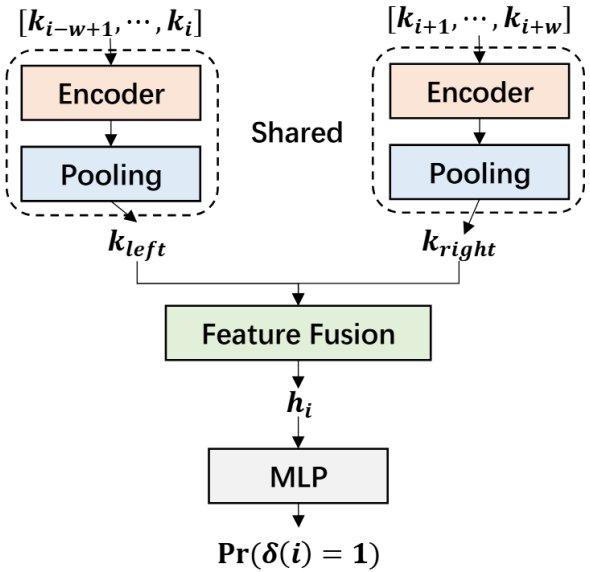
Output: The probability that token i is a chunk end $\Pr(\delta(i = 1))$

Rationales:

- **Local context is sufficient** as high similarity between preceding and succeeding tokens suggests continuity, whereas low similarity indicates a contextual shift.

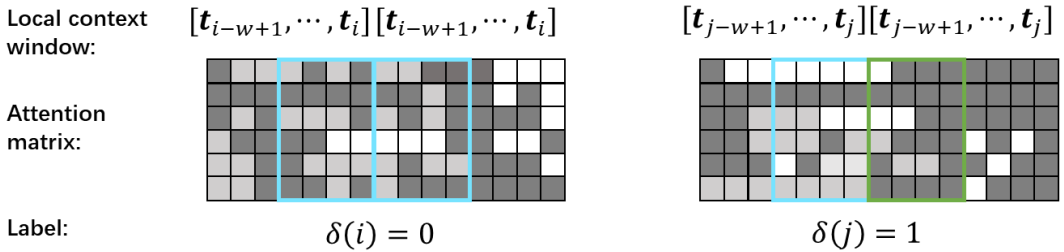
- Limiting to local windows greatly **reduces computation**.

Architecture of Boundary Predictor



Automatic Labelling

- End-to-end optimization is computationally intractable.
- Automatically derived from attention scores: Compare accumulated attention mass



Inference of the Boundary Predictor

Non-Maximum Suppression (NMS): well-separated, reliable boundaries

Empirical Findings: Applying NMS with a small window size (e.g., 8) yields better results than using no NMS. For tasks requiring broader context segmentation, a larger NMS window size (e.g., 64) further improves performance.

Robust Chunk Representation with Variable Sizes

Average pooling is sensitive to chunk length.

Add length normalization:

$$\mathbf{q}_c = \sqrt{|\mathbf{C}|} \cdot \bar{\mathbf{q}}, \quad \mathbf{k}_c = \sqrt{|\mathbf{C}|} \cdot \bar{\mathbf{k}}$$

$|C|$: Chunk size
$\bar{q}$: Average of token queries within the chunk
$\bar{k}$: Average of token keys within the chunk

Alternative: max pooling

$$\mathbf{q}_c = \mathrm{maxPool}(\{\mathbf{q}_j | \mathbf{t}_j \in \mathbf{C}\}), \quad \mathbf{k}_c = \sqrt{|\mathbf{C}|} \cdot \bar{\mathbf{k}}$$

Evaluation: Experiment Setup

PyTorch & Hugging Face Transformers
Backend: PyTorch SDPA / Tiled online-softmax

GPU:
- RTX 3090 (24GB) × 1
- Ubuntu 22.04.4 LTS
- Python 3.12
- CUDA 12.4
- PyTorch 2.5.1+cu124
- Transformers 4.52.3

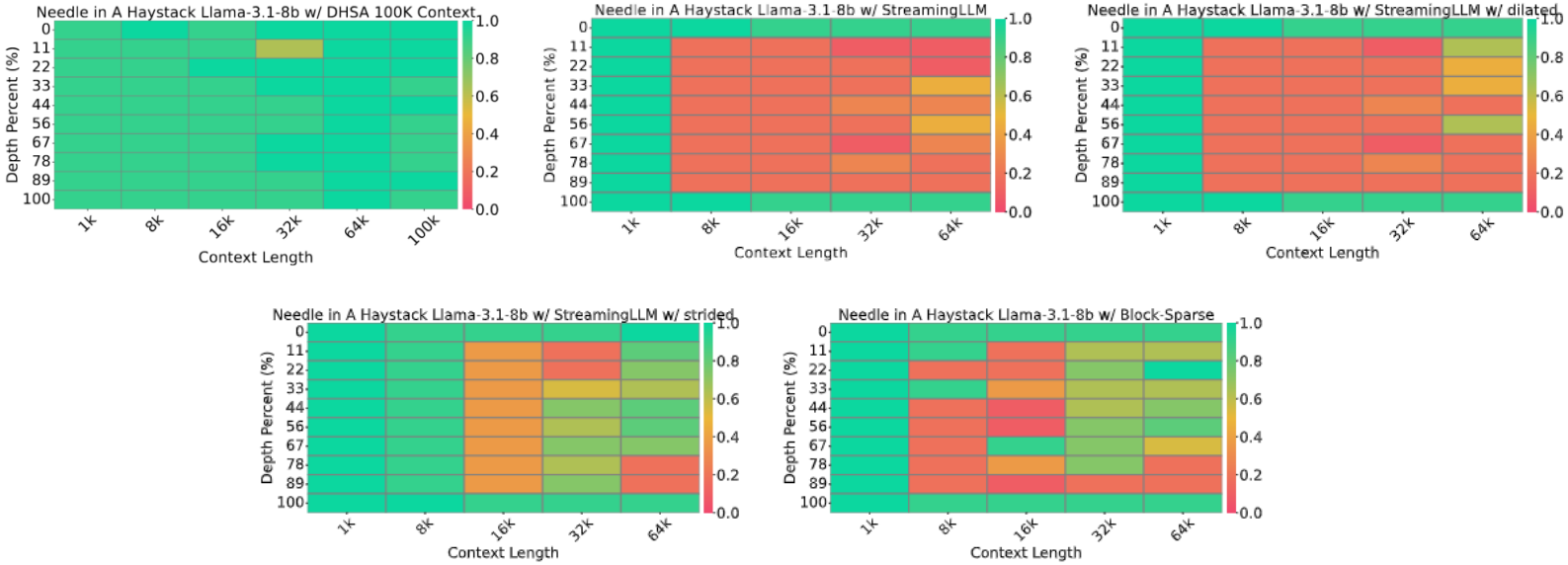
CPU:
- Intel Core 5 120U × 1
- Windows 11
- Python 3.11
- PyTorch 2.9.1+cpu
- Transformers 4.52.3

Models:
- Llama-3.1-8B-Instruct (4-bit)
- Qwen2.5-3B-Instruct
- gemma-2-2b-it

- Qwen2.5-14B-Instruct
- Llama-3.1-8B-Instruct (BF16)

Baselines: StreamingLLM, Minference, Block-Sparse, DuoAttention, SeerAttention, Quest

Needle-in-a-Haystack Test

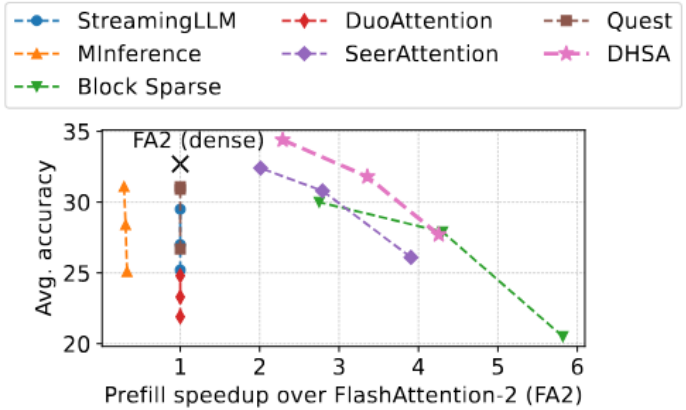


LongBench

Methods	Single Doc QA	Multi Doc QA	Summ.	Few-shot Learning	Synth.	Code	Avg.
Llama-3.1-8B-Instruct (4-bit)	22.0	10.5	29.4	68.3	44.0	22.3	32.7
StreamingLLM	15.0	6.8	**27.7**	62.3	26.0	22.0	27.0
StreamingLLM w/ dilated	15.1	6.7	27.6	62.3	26.0	22.1	27.0
StreamingLLM w/ strided	12.0	5.0	25.5	58.9	11.3	**24.1**	23.4
MInference	17.6	8.2	26.7	67.8	24.3	22.1	28.4
Block Sparse	16.3	7.4	22.3	59.7	44.2	20.5	27.9
DuoAttention	13.0	6.5	26.4	60.9	3.5	22.5	23.3
SeerAttention	19.9	10.4	25.6	68.7	40.1	19.5	30.8
Quest	**22.4**	**10.7**	27.1	60.3	45.2	21.4	30.9
DHSA	18.3	10.1	26.9	**68.8**	**45.7**	22.7	**31.8**
Qwen2.5-3B-Instruct	12.7	6.9	25.5	66.9	20.0	19.8	26.0
StreamingLLM	9.2	6.0	**25.7**	54.8	2.5	19.8	20.7
StreamingLLM w/ dilated	9.7	6.2	25.0	54.0	3.3	19.6	20.7
StreamingLLM w/ strided	8.1	7.0	24.7	47.2	2.1	18.1	18.8
MInference	10.1	5.5	24.1	57.0	2.5	19.6	21.1
Block Sparse	9.6	4.6	21.3	56.1	10.0	17.5	20.6
DHSA	**11.3**	**7.9**	24.7	**67.2**	**14.0**	**21.6**	**25.3**
gemma-2-2b-it	27.0	26.2	24.7	65.0	7.5	25.5	30.9
StreamingLLM	13.6	15.2	22.9	53.4	7.5	26.2	23.9
StreamingLLM w/ dilated	14.8	16.5	22.3	52.2	**10.0**	**28.7**	24.7
StreamingLLM w/ strided	14.1	15.1	**23.2**	51.2	**10.0**	24.4	23.7
MInference	17.3	20.8	22.3	58.0	5.0	27.5	26.3
Block Sparse	11.4	17.4	16.3	51.6	2.5	25.1	21.6
DHSA	**27.6**	**24.5**	23.1	**64.4**	7.5	25.4	**30.3**

LongBench

Method	Prefill	Model-Agnostic	HW
StreamingLLM		✓	✓
MInference	✓	✓	
Block-Sparse	✓		
DuoAttention		✓†	
SeerAttention	✓		
Quest		✓†	
DHSA	✓	✓	✓

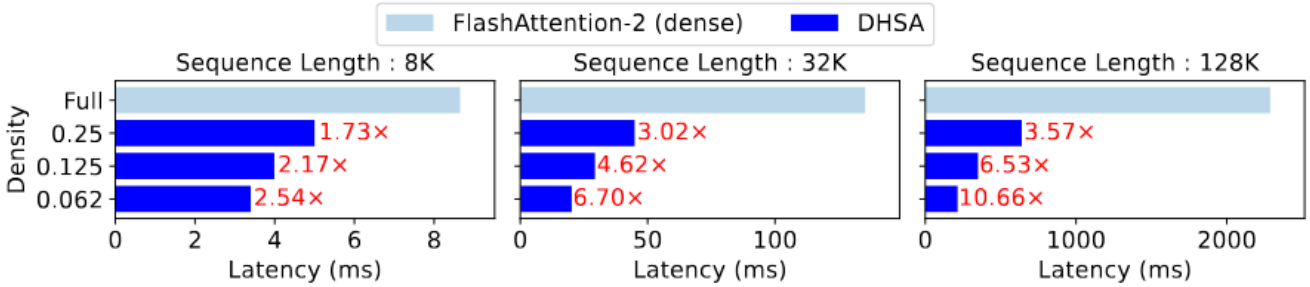


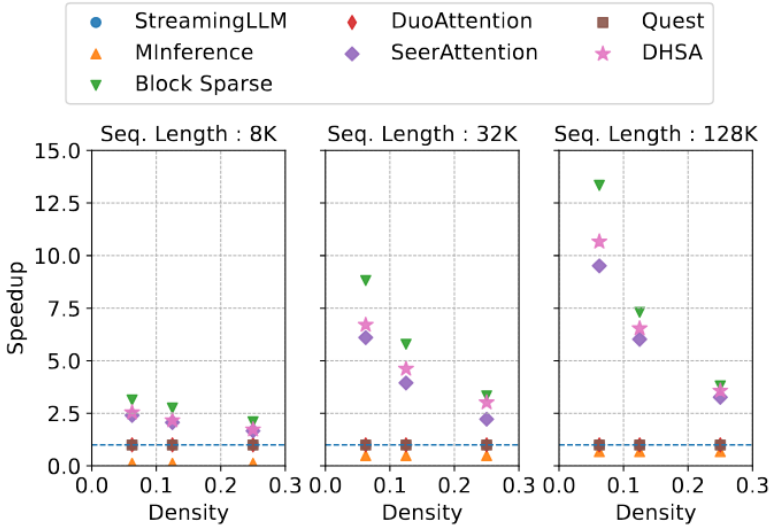
RULER

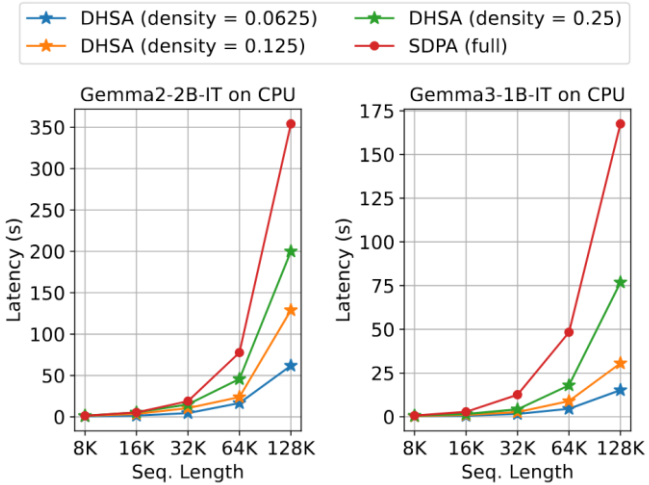
Method	4K	8K	16K	32K	48K
Llama-3.1-8B-Instruct (4-bit)	95.8	93.8	93.2	81.5	75.2
MInference	91.7	88.4	88.6	74.7	64.2
BlockSparse	60.3	65.4	69.3	53.0	43.1
DuoAttention	92.0	**89.3**	**89.7**	72.9	69.1
SeerAttention	86.9	89.0	89.1	75.0	53.6
Quest	87.7	86.0	87.2	73.9	68.2
DHSA	**92.1**	88.5	88.7	**76.2**	**71.5**

Method	**4K**				**8K**				**16K**				**32K**			
	QA-1	**QA-2**	**VT**	**Avg.**	**QA-1**	**QA-2**	**VT**	**Avg.**	**QA-1**	**QA-2**	**VT**	**Avg.**	**QA-1**	**QA-2**	**VT**	**Avg.**
Llama-3.1-8B-Instruct (4-bit)	83.8	72.0	100.0	95.8	78.5	72.0	99.6	93.8	72.2	66.0	100.0	93.2	70.5	66.0	67.6	81.5
BlockSparse	23.0	32.0	85.2	60.3	37.2	34.0	84.0	65.4	33.7	44.0	82.4	69.3	24.2	28.0	24.0	53.0
DHSA	**74.3**	**65.8**	**99.4**	**92.1**	**69.1**	**63.3**	**97.6**	**88.5**	**63.5**	**61.1**	**97.9**	**88.7**	**59.2**	**56.8**	**61.4**	**76.2**

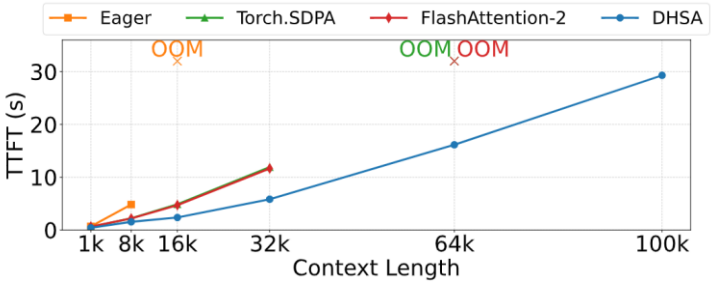
Latency


FlashAttention-2 (dense)
DHSA
Sequence Length : 8K
Sequence Length : 32K
Sequence Length : 128K
Density
Full
0.25
0.125
0.062
1.73×
2.17×
2.54×
3.02×
4.62×
6.70×
3.57×
6.53×
10.66×
Latency (ms)
Latency (ms)
Latency (ms)


StreamingLLM
MInference
Block Sparse
DuoAttention
SeerAttention
Quest
DHSA
Seq. Length : 8K
Seq. Length : 32K
Seq. Length : 128K
Speedup
Density
Density
Density


DHSA (density = 0.0625)
DHSA (density = 0.125)
DHSA (density = 0.25)
SDPA (full)
Gemma2-2B-IT on CPU
Gemma3-1B-IT on CPU
Latency (s)
Latency (s)
Seq. Length
Seq. Length

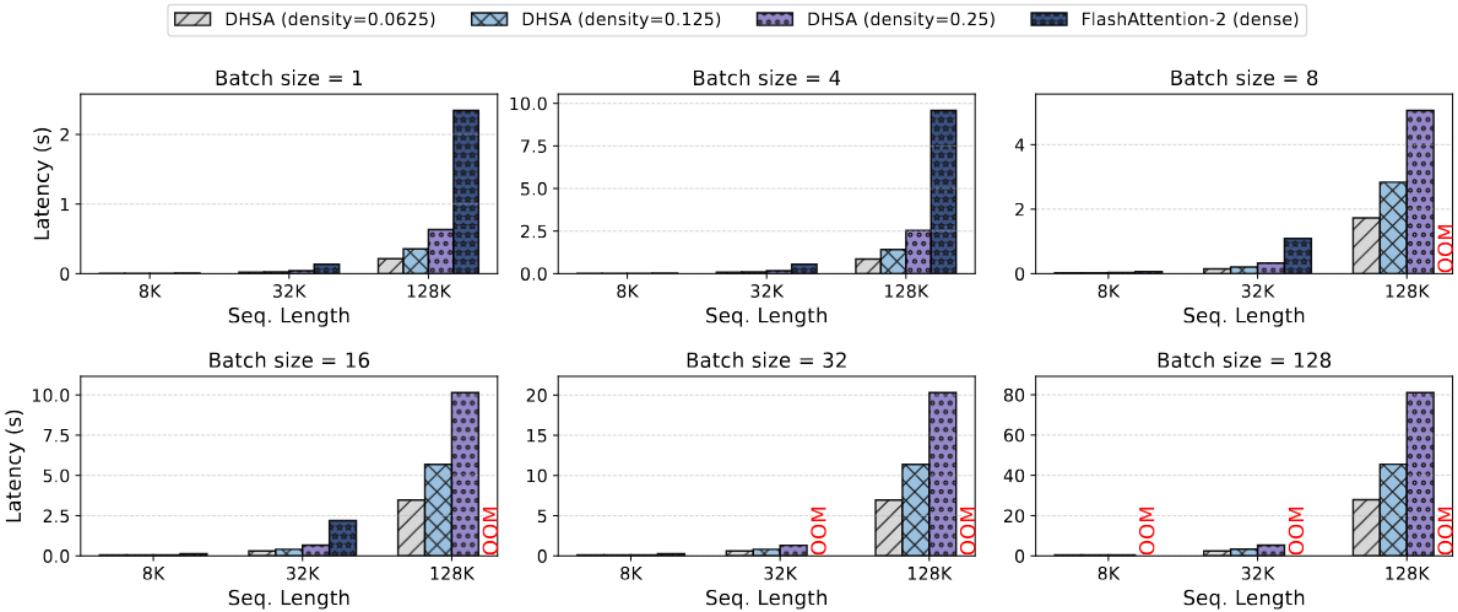
Latency



Component	8K (ms)	16K (ms)	32K (ms)
Boundary prediction	45	97	201
Routing / chunk selection	237	379	944
Sparse attention kernel	128	316	939
DHSA TTFT	1,550	2,360	5,830
Dense FA2 TTFT	2,170	4,710	11,680

Dataset	Avg. Tokens	Dense FA2 (s)	DHSA (s)
NarrativeQA	29,869	10.73	5.37
HotpotQA	12,854	3.71	2.04
Musique	15,617	4.59	2.32
QMSum	13,917	4.05	2.15
PassageCount	14,970	4.38	2.26
RepoBench-P	10,818	3.07	1.84
Avg.	–	3.28	1.88

Latency


DHSA (density=0.0625)
DHSA (density=0.125)
DHSA (density=0.25)
FlashAttention-2 (dense)
Batch size = 1
Batch size = 4
Batch size = 8
Batch size = 16
Batch size = 32
Batch size = 128
Latency (s)
Seq. Length
8K
32K
128K
OOM

Takeaways

- Dense attention is the main bottleneck for memory-constrained long-context LLM inference.
- Sparse attention must be input-adaptive: static or heuristic patterns can miss important tokens.
- DHSA preserves near-dense accuracy while reducing prefill latency, enabling long-context inference on limited hardware.

Future Work

- Long-generation support: extend DHSA from prefill-dominated workloads to decoding and KV-cache management.
- Broader generalization: evaluate DHSA on more reasoning, code, and multi-modal long-context tasks.
- Hybrid efficiency stack: combine DHSA with quantization, KV-cache compression, and prompt compression.

Thank you!