北京大学
PEKING UNIVERSITY

Efficient Attention

Boao Kong, Yutong He, Renjia Deng
Kun Yuan

Review: computational complexity for Transformer

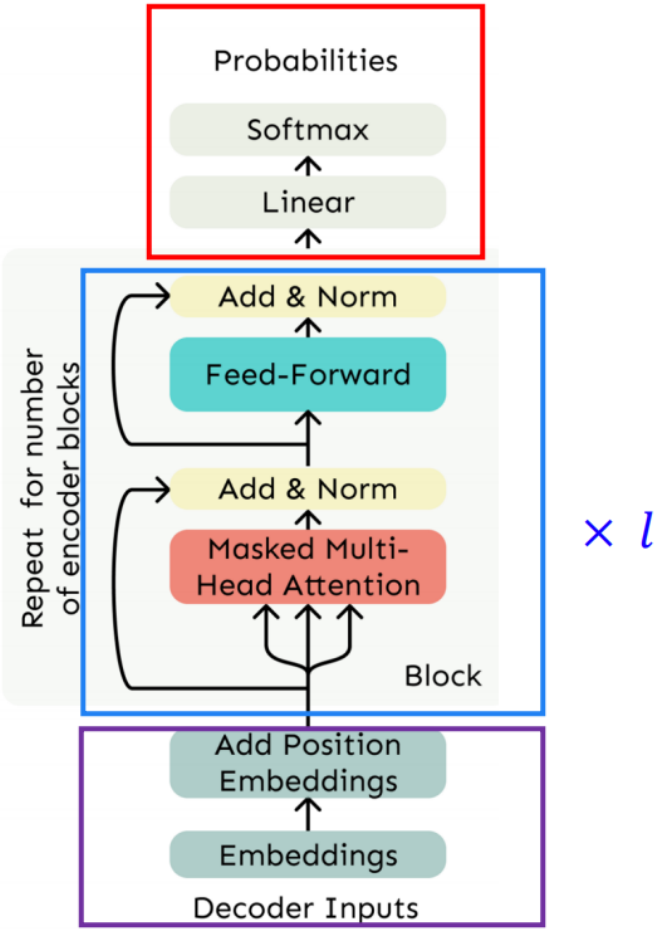
- Embeddings: $2svh$
- Attention blocks: $24lsh^2 + 4s^2lh$
- Probability predictions: $2svh$

Total forward FLOPs:

$$\ell(24sh^2 + 4s^2h) + 4svh$$

When using batch-size b, the total forward FLOPs:

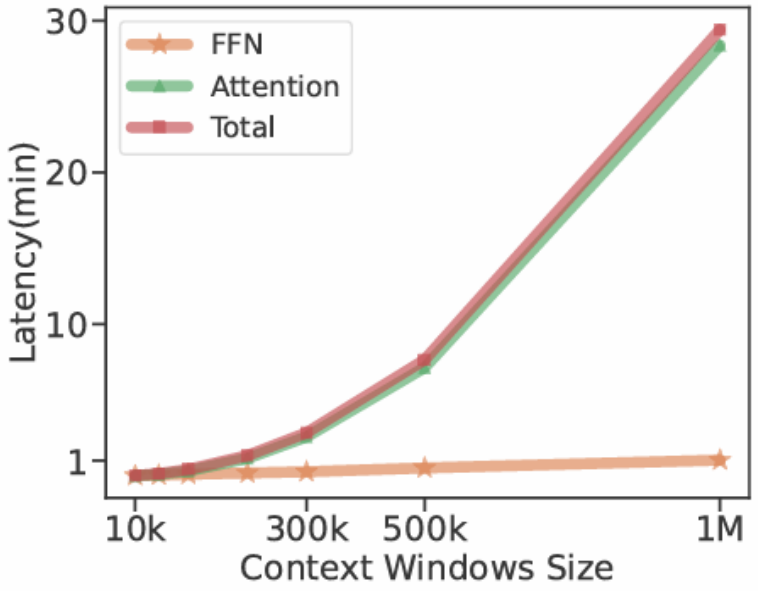
$$b\ell(24sh^2 + 4s^2h) + 4bsvh$$



Review: computational complexity for Transformer

- The pre-training stage for modern LLMs perfers long sequence length like 4096 or 8192.
- During inference period, it will be much higher to ~1M.
- With long sequence length, the computation for Attention blocks dominate the computation

- Embeddings: $2svh$
- Attention blocks: $24lsh^2 + 4s^2lh$
- Probability predictions: $2svh$



Outline

- Sparse attention
  - Mixture of Block Attention (MoBA)
  - Native Sparse Attention (NSA)
- Linear Attention
  - Kernelized Sparse Attention
  - Hardware-Efficient Training of Linear Attention
  - DeltaNet

北京大学
PEKING UNIVERSITY

PART 01

Mixture of Block Attention (MoBA)

Mixture of Block Attention (MoBA)

北京大学 PEKING UNIVERSITY

Kimi Mixture of Block Attention （MoBA）

MoBA: Mixture of Block Attention for Long-Context LLMs

Full Report

🚀 Introducing **MoBA --- Mixture of Block Attention**

- **Trainable Block Sparse Attention**: The full context is divided into blocks, where each query token learns to attend to the most relevant KV blocks, enabling efficient processing of long sequences.
- **Parameter-less Gating Mechanism**: A novel Parameter-less top-k gating mechanism is introduced to selects the most relevant blocks for each query token, ensuring that the model focuses only on the most informative blocks.
- **Seamlessly Transition between Full and Sparse Attention**: MoBA is designed to be a flexible substitute for full attention, allowing seamless transitions between full and sparse attention modes.

Block Attention for MoBA

- Recall the traditional Attention: $\mathrm{Attn}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Softmax}\left(\boldsymbol{q}\boldsymbol{K}^{\top}\right)\boldsymbol{V}$
- MoBA uses **a subset for the key and value** to obtain attention score **for each query**:

$$\mathrm{MoBA}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Softmax}\left(\boldsymbol{q}\boldsymbol{K}[I]^{\top}\right)\boldsymbol{V}[I]$$

  where $[I]$ denote the activated demension for key and value, which is a subset of $\{1, 2, \cdots, s\}$.

- To obtain the activated dimension, MoBA uses the TopK strategy for the affinity score

$$I_i = [(i-1) \times B + 1, i \times B]$$ — Split the small batch

$$s_i = \langle \boldsymbol{q}, \mathrm{mean\_pool}(\boldsymbol{K}[I_i]) \rangle$$ — Obtain query-key affinity score

$$g_i = \begin{cases} 1 & s_i \in \mathrm{Topk}\left(\{s_j | j \in [n]\}, k\right) \\ 0 & \text{otherwise} \end{cases}$$ — Top-K gate

$$I = \bigcup_{g_i > 0} I_i.$$ — Activated dimension

Computation complexity for MoBA

北京大学 PEKING UNIVERSITY

- Obtain the activated dimension for one query: s/B inner products for h/a -dimension vectors:

$$2sh/(aB) \quad \text{FLOPs}$$

- s querys and a heads in total:

$$sa \times 2sh/(aB) = 2s^2h/B \quad \text{FLOPs}$$

$$I_i = [(i-1) \times B + 1, i \times B]$$ — Split the small batch

$$s_i = \langle \boldsymbol{q}, \text{mean\_pool}(\boldsymbol{K}[I_i]) \rangle$$ — Obtain query-key affinity score

$$g_i = \begin{cases} 1 & s_i \in \text{Topk}(\{s_j | j \in [n]\}, k) \\ 0 & \text{otherwise} \end{cases}$$ — Top-K gate

$$I = \bigcup_{g_i > 0} I_i.$$ — Activated dimension

- Sparse Multi-head Attention:
  - QK: Multiplation between $s \times h/a$ and $h/a \times kB$: $2shkB/a \quad \text{FLOPs}$
  - KV: Multiplation between $s \times kB$ and $kB \times h/a$: $2shkB/a \quad \text{FLOPs}$
  - a heads in total: $4shkB \quad \text{FLOPs}$

- Totally: $4shkB + 2s^2h/B \quad \text{FLOPs}$
- MHA: $4s^2h \quad \text{FLOPs}$

Comparsion for computation complexity

- MOBA: $4shkB + 2s^2h/B \quad \text{FLOPs}$
- MHA: $4s^2h \quad \text{FLOPs}$

- In the case of LLMs pre-training, we set s=4096, B=512, k=3, MOBA takes only **37.60%** FLOPs of MHA.

- If s=32768, B=512, k=3, MOBA takes only **4.785%** FLOPs of MHA.

- A significant computation saving in long context scenarios.

MoBA with FlashAttention

北京大学 PEKING UNIVERSITY

- MoBA can be integrated into FlashAttention with the combination with MoBA attention and historical attention

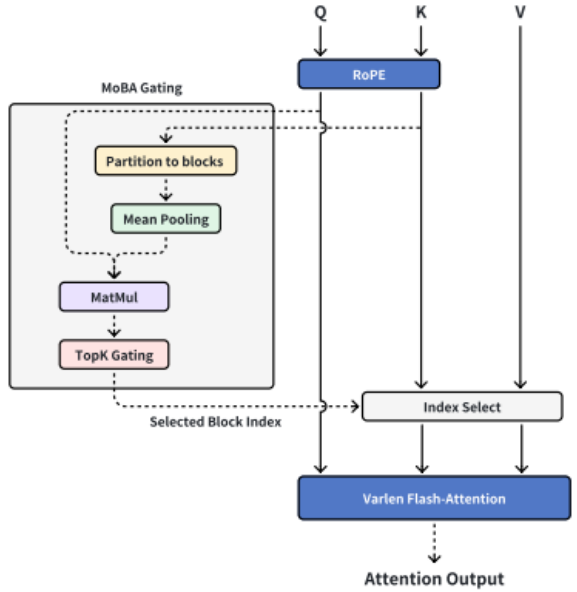
Algorithm 1 MoBA (Mixture of Block Attention) Implementation

Require: Query, key and value matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times h \times d}$; MoBA hyperparameters (block size B and top-k); h and d denote the number of attention heads and head dimension. Also denote $n = N/B$ to be the number of blocks.

// Split KV into blocks
$\{\tilde{\mathbf{K}}_i, \tilde{\mathbf{V}}_i\} = \text{split\_blocks}(\mathbf{K}, \mathbf{V}, B)$, where $\tilde{\mathbf{K}}_i, \tilde{\mathbf{V}}_i \in \mathbb{R}^{B \times h \times d}, i \in [n]$
// Compute gating scores for dynamic block selection
$\bar{\mathbf{K}} = \text{mean\_pool}(\mathbf{K}, B) \in \mathbb{R}^{n \times h \times d}$
$\mathbf{S} = \mathbf{Q}\bar{\mathbf{K}}^\top \in \mathbb{R}^{N \times h \times n}$
// Select blocks with causal constraint (no attention to future blocks)
$\mathbf{M} = \text{create\_causal\_mask}(N, n)$
$\mathbf{G} = \text{topk}(\mathbf{S} + \mathbf{M}, k)$
// Organize attention patterns for computation efficiency
$\mathbf{Q}^s, \tilde{\mathbf{K}}^s, \tilde{\mathbf{V}}^s = \text{get\_self\_attn\_block}(\mathbf{Q}, \tilde{\mathbf{K}}, \tilde{\mathbf{V}})$
$\mathbf{Q}^m, \tilde{\mathbf{K}}^m, \tilde{\mathbf{V}}^m = \text{index\_select\_moba\_attn\_block}(\mathbf{Q}, \tilde{\mathbf{K}}, \tilde{\mathbf{V}}, \mathbf{G})$
// Compute attentions seperately
$\mathbf{O}^s = \text{flash\_attention\_varlen}(\mathbf{Q}^s, \tilde{\mathbf{K}}^s, \tilde{\mathbf{V}}^s, \text{causal=True})$
$\mathbf{O}^m = \text{flash\_attention\_varlen}(\mathbf{Q}^m, \tilde{\mathbf{K}}^m, \tilde{\mathbf{V}}^m, \text{causal=False})$
// Combine results with online softmax
$\mathbf{O} = \text{combine\_with\_online\_softmax}(\mathbf{O}^s, \mathbf{O}^m)$
return $\mathbf{O}$

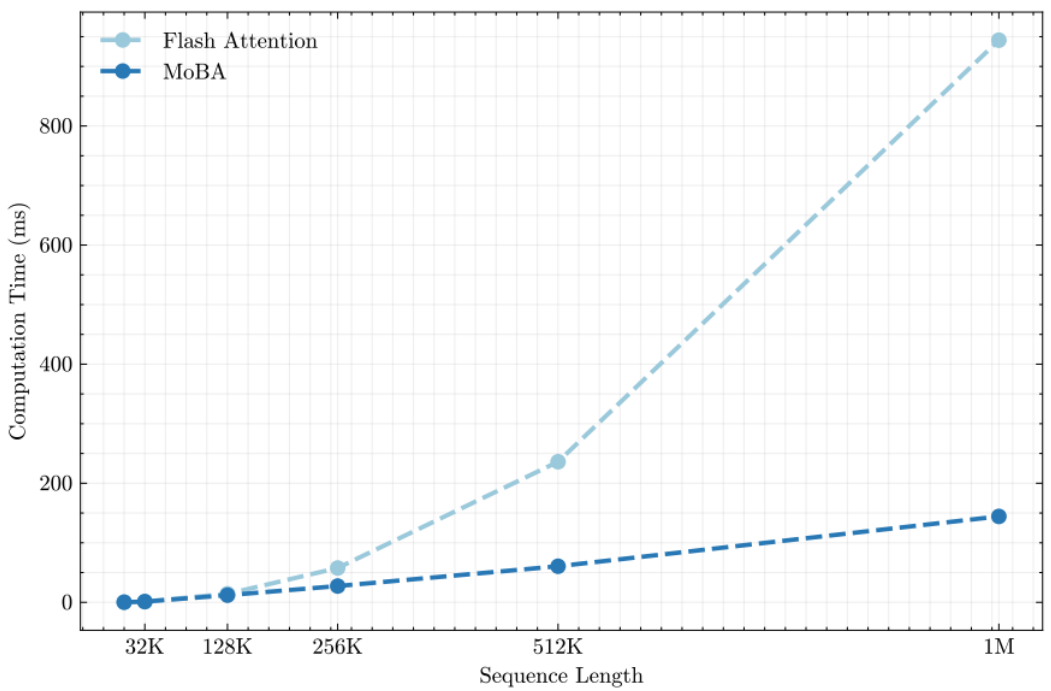
MoBA Gating

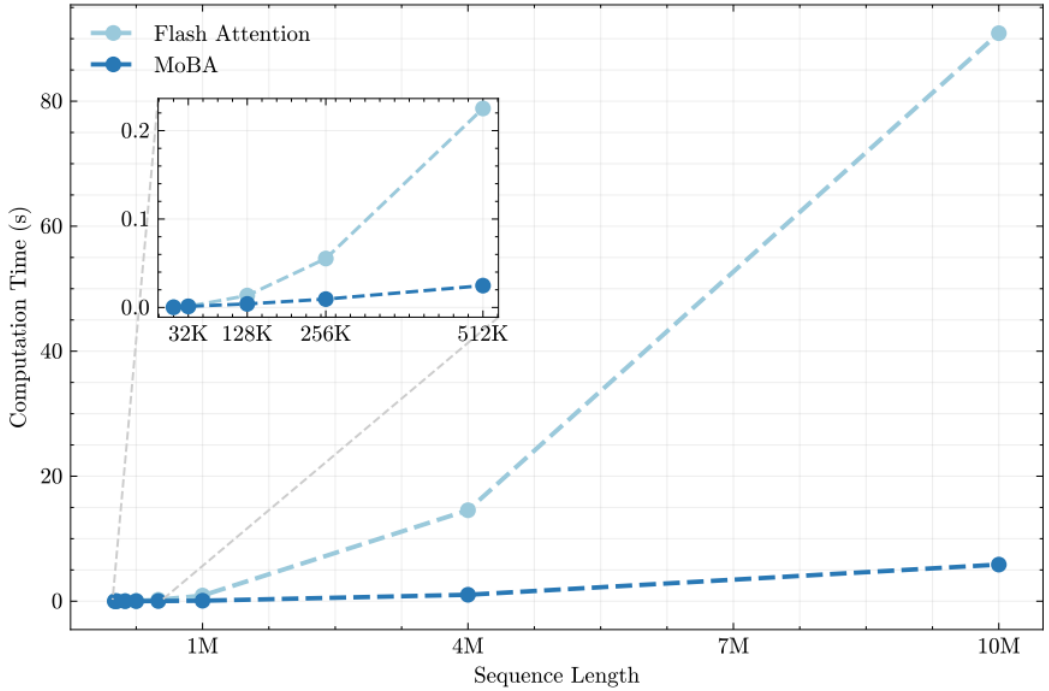
Obtain MoBA attention and historical attention seperately



MoBA with FlashAttention

- MoBA+FlashAttention achieves a significant acceleration compared to MHA+FlashAttention
- Computation time on 1M models with **increasing sequence length** and **fixed sparse ratio**:

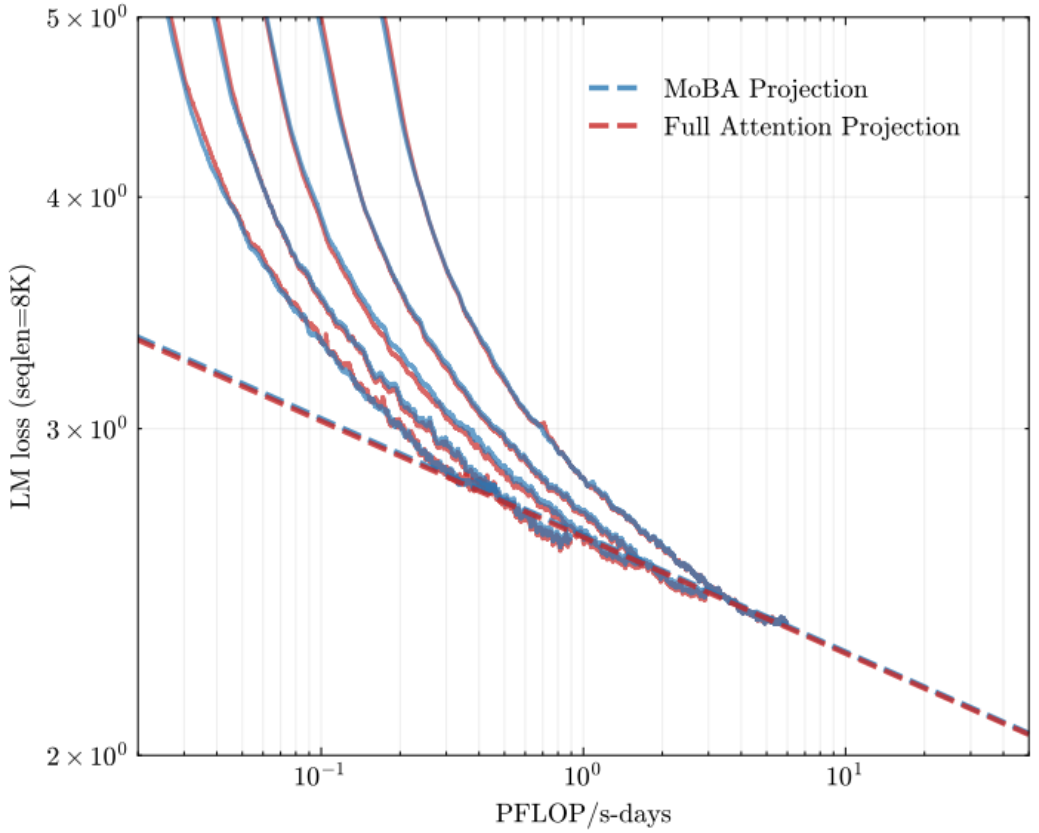




Validation performance

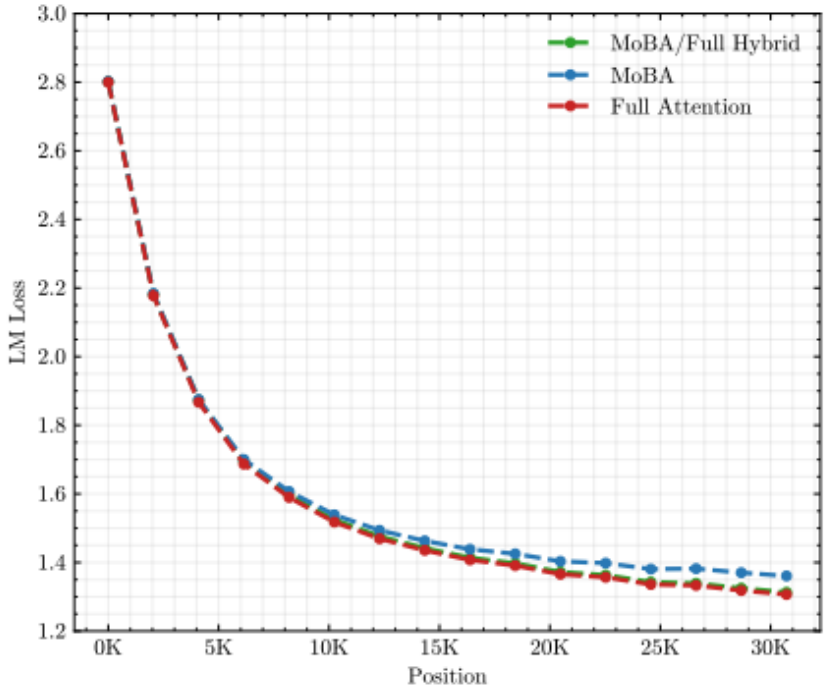
- MoBA can achieve a comparable performance as Full Attention in validation set during the training stage with scaled model size.
- PFLOP: A metric for the computation complexity.

Model Param	Head	Layer	Hidden	Training Token	Block size	TopK
568M	14	14	1792	10.8B	512	3
822M	16	16	2048	15.3B	512	3
1.1B	18	18	2304	20.6B	512	3
1.5B	20	20	2560	27.4B	512	3
2.1B	22	22	2816	36.9B	512	3



Hybrid training with MoBA and MHA

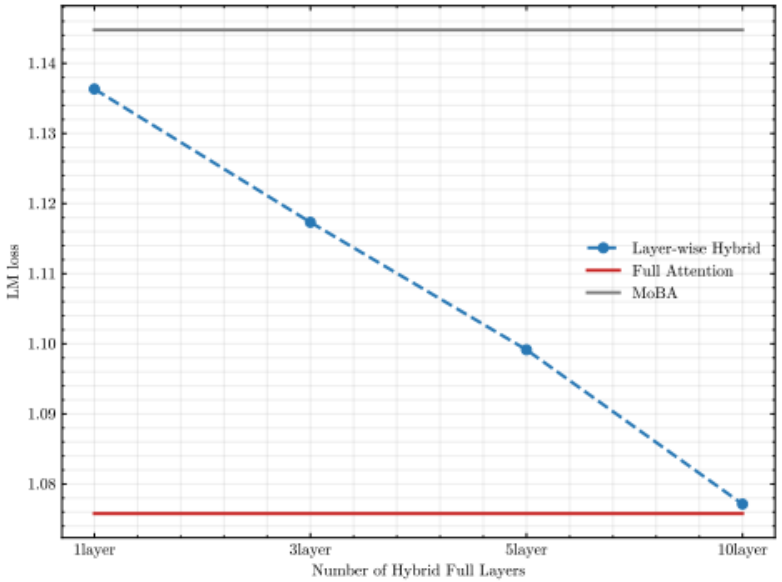
- In practical training tasks, MoBA can be combined with MHA
  - Use MoBA in the first 90% steps: reduce computation with long sequence length
  - Use MHA in the last 10% steps: improve the long context ability
- The **position-wise** LM Loss in long context scenario illustrates that the hybrid training for MoBA and MHA can improve the long context ability（这里position-wise指的是长文本输出下，某一个token位置的平均loss）

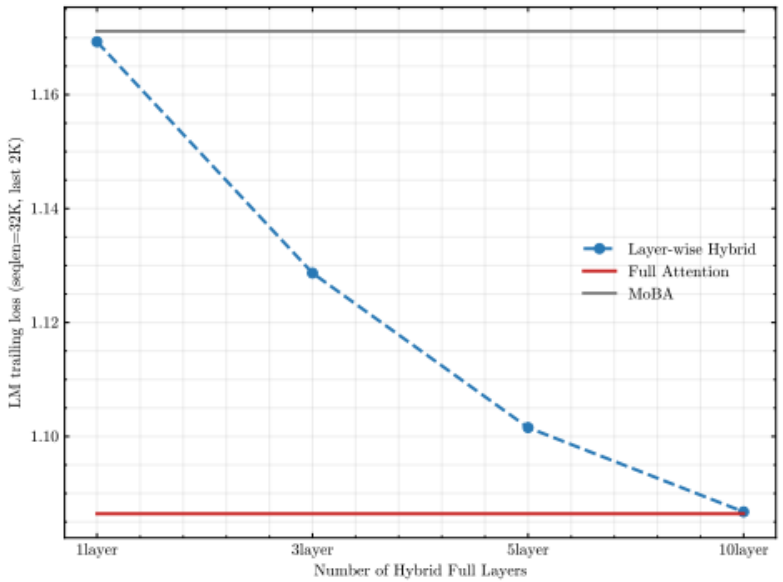


Layer-wise hybrid strategy for MoBA

北京大学 PEKING UNIVERSITY

- MoBA introduces the **layer-wise hybrid strategy** in SFT period.
  - Motivation: SFT typically applies loss masking to prompt tokens, which may hinder the backpropagation of gradients through sparse attention paths, affecting model optimization.
  - Method: Only the **last few layers** are replaced with **full attention**, while the remaining layers maintain MoBA sparse attention.
  - SFT in Transformer with 32 layers: Achieving full attention's performance with **10 full layers**.





MoBA: summary

北京大学 PEKING UNIVERSITY

- MoBA applies the idea of MoE to the attention blocks.
- Reduce the computation for the attention, especially in long sequence case.
- Can be combined with FlashAttention.
- Step-wise and layer-wise hybrid training can balance the efficiency and long context performance

Extension: MoH

北京大学 PEKING UNIVERSITY

- Mixture-of-heads (MoH) is another framework that applies MoE to the attention blocks.

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V},$$

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \boldsymbol{K} = \boldsymbol{X}'\boldsymbol{W}_K, \boldsymbol{V} = \boldsymbol{X}'\boldsymbol{W}_V,$$

$$\text{MoH}(\boldsymbol{X},\boldsymbol{X}') = \sum_{i=1} g_i \boldsymbol{H}^i \boldsymbol{W}_O^i,$$

MoH routing score

$$g_i = \begin{cases} \alpha_1 \text{Softmax}(\boldsymbol{W}_s \boldsymbol{x}_t)_i, & \text{if } 1 \le i \le h_s, \\ \alpha_2 \text{Softmax}(\boldsymbol{W}_r \boldsymbol{x}_t)_{i-h_s}, & \text{if Head } i \text{ is activated,} \\ 0, & \text{otherwise,} \end{cases}$$

TopK gating & learnable factor

- MoH can reduce the activated parameters and thus reduce the memory and computation in pre-training and inference.

Table 4. **Comparisons between MoH-LLaMA3-8B and LLaMA3-8B.** Please refer to Tab. G in the Appendix for the performance of the model at the end of the first stage of training.

Methods	#Activated Heads (%)	MMLU (5)	CEVAL (5)	CMMLU (5)	GSM8K(8)	TruthfulQA
LLaMA3-8B (Dubey et al., 2024)	100	65.2	52.3	50.7	49.5	35.4
MoH-LLaMA3-8B	75	65.8	61.5	64.4	56.9	44.0
Methods	**#Activated Heads (%)**	**MMLU (5)**	**CEVAL (5)**	**CMMLU (5)**	**GSM8K(8)**	**TruthfulQA**
LLaMA3-8B (Dubey et al., 2024)	100	81.9	30.0	83.9	75.5	94.0
MoH-LLaMA3-8B	75	80.1	30.3	84.0	76.4	92.2
Methods	**#Activated Heads (%)**	**MMLU (5)**	**CEVAL (5)**	**CMMLU (5)**	**GSM8K(8)**	**TruthfulQA**
LLaMA3-8B (Dubey et al., 2024)	100	81.0	72.5	31.5	59.0	61.6
MoH-LLaMA3-8B	75	78.8	72.9	28.3	60.1	**64.0**

Table 7. **Comparisons about inference time.** We convert the $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ features into sparse matrices using the mask generated by the router and replace the dense matrix multiplication in the attention mechanism with sparse matrix multiplication. To eliminate the impact of underlying operator optimizations, we replaced all matrix multiplications with sparse matrix multiplication when testing for speed.

Methods	#Head Num	#Head Dim	#Sequence Length	#Activated Heads (%)	Time (ms)
Multi-Head Attention	32	64	256	100	0.360
MoH (Ours)	32	64	256	90	0.352
MoH (Ours)	32	64	256	75	0.321
MoH (Ours)	32	64	256	50	**0.225**
Multi-Head Attention	32	64	512	100	1.376
MoH (Ours)	32	64	512	90	1.351
MoH (Ours)	32	64	512	75	1.180
MoH (Ours)	32	64	512	50	**0.863**

PART 02

Native Sparse Attention (NSA)

NSA: native sparse attention

北京大学 PEKING UNIVERSITY

deepseek

Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention

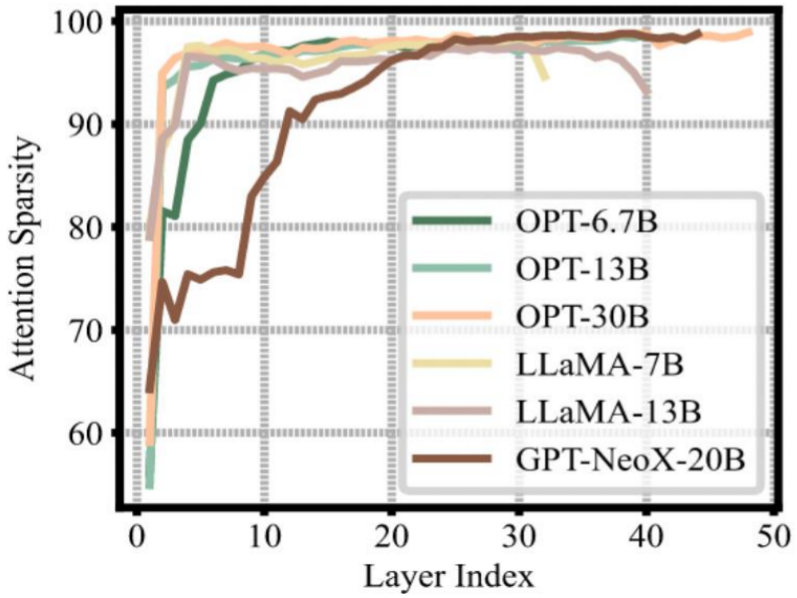
Jingyang Yuan*[1,2], Huazuo Gao[1], Damai Dai[1], Junyu Luo[2], Liang Zhao[1], Zhengyan Zhang[1], Zhenda Xie[1], Y. X. Wei[1], Lean Wang[1], Zhiping Xiao[3], Yuqing Wang[1], Chong Ruan[1], Ming Zhang[2], Wenfeng Liang[1], Wangding Zeng[1]

[1]DeepSeek-AI
[2]Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, PKU-Anker LLM Lab
[3]University of Washington
{yuanjy, mzhang_cs}@pku.edu.cn, {zengwangding, wenfeng.liang}@deepseek.com

Motivation: Attention is naturally sparse

北京大学 PEKING UNIVERSITY

- Key Observation: Although LLM is densely trained, **Attention is naturally sparse.**
  - In OPT family, almost KV cache of all layers have a sparsity over **95%**, which means potential **20X** KV cache reduction.



- Idea:

Approximate the attention score matrix with sparse matrix, only load/save tokens with large score.

- Challenges:

How to **efficiently find** the "important" tokens?

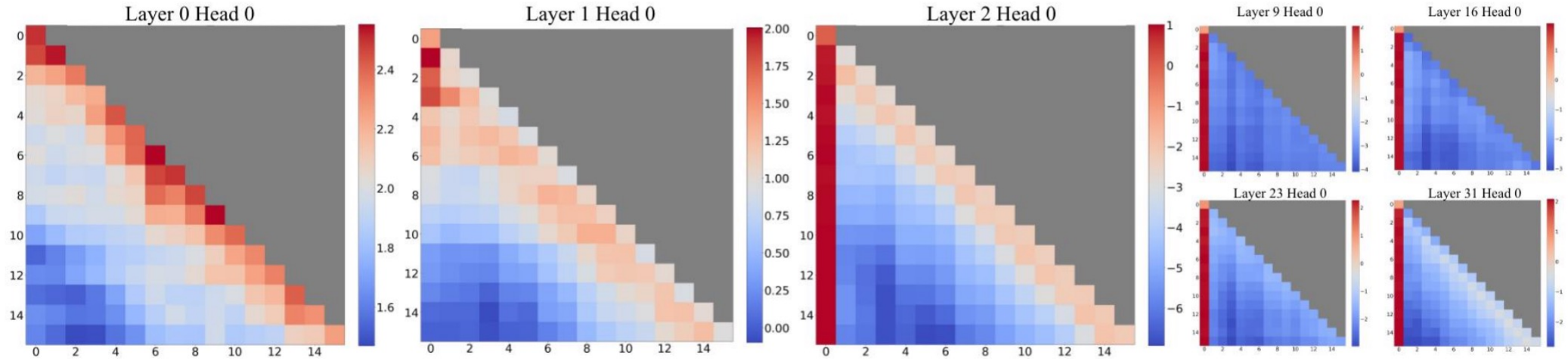
How to **maintain the performance** of LLM without heavy finetuning?

Motivation: Attention sink

北京大学 PEKING UNIVERSITY

- Interesting phenomenon: **Attention sink**

Large attention scores are given to initial tokens as a “sink” **even if they are not semantically important**, especially in the deeper layer.



average attention logits in Llama-2-7B over 256 sentences.

Understanding NSA: The "Long Article" Reading Analogy

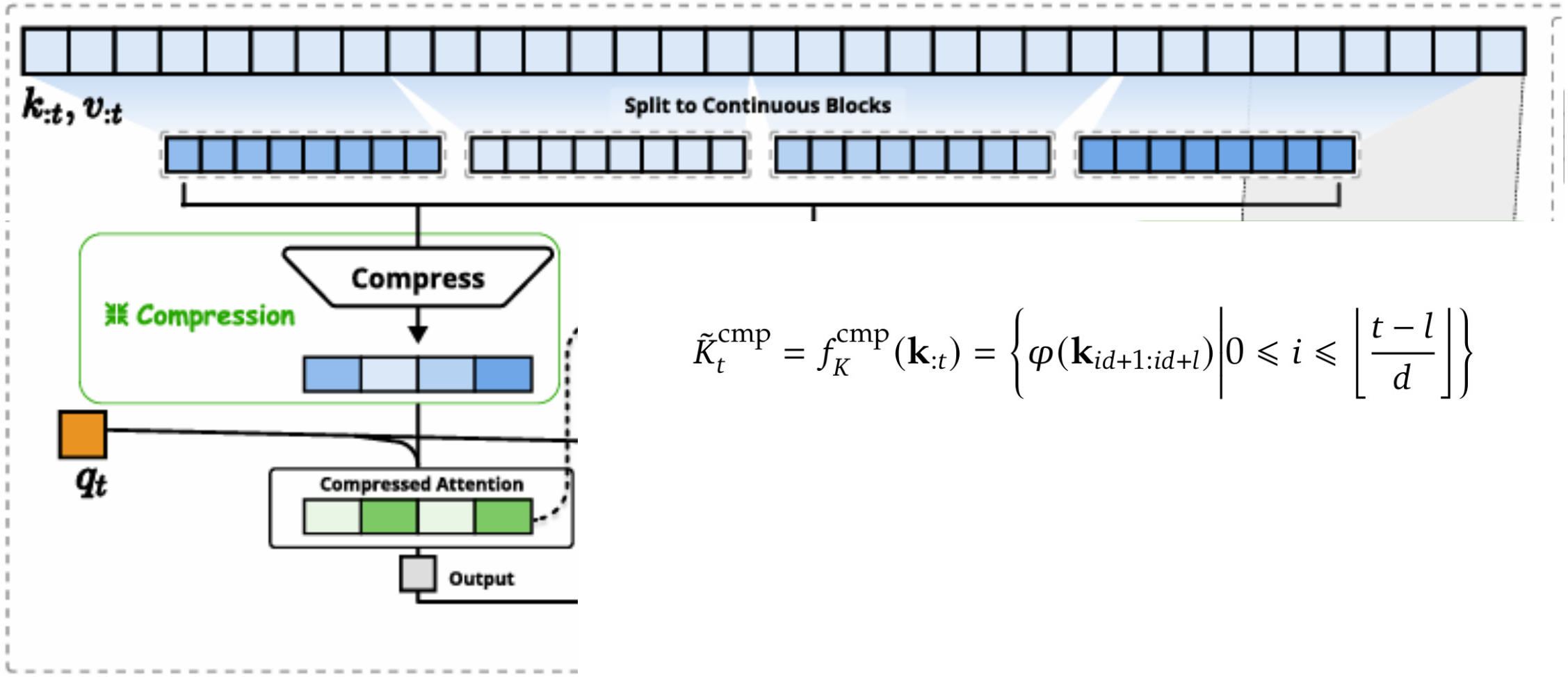
The Challenge: How do we efficiently process very long texts?

- **Step 1:** Roughly skim the text from beginning to end to identify the paragraphs of interest.
- **Step 2**: Read the previously marked key paragraphs with word-by-word precision.
- **Step 3:** Expand the scope around specific phrases of interest to the surrounding sentences to capture important context.

NSA utilizes this three-step philosophy to effectively solve the Sparse Attention problem.

NSA: Token compression + Token sclection + sliding attention

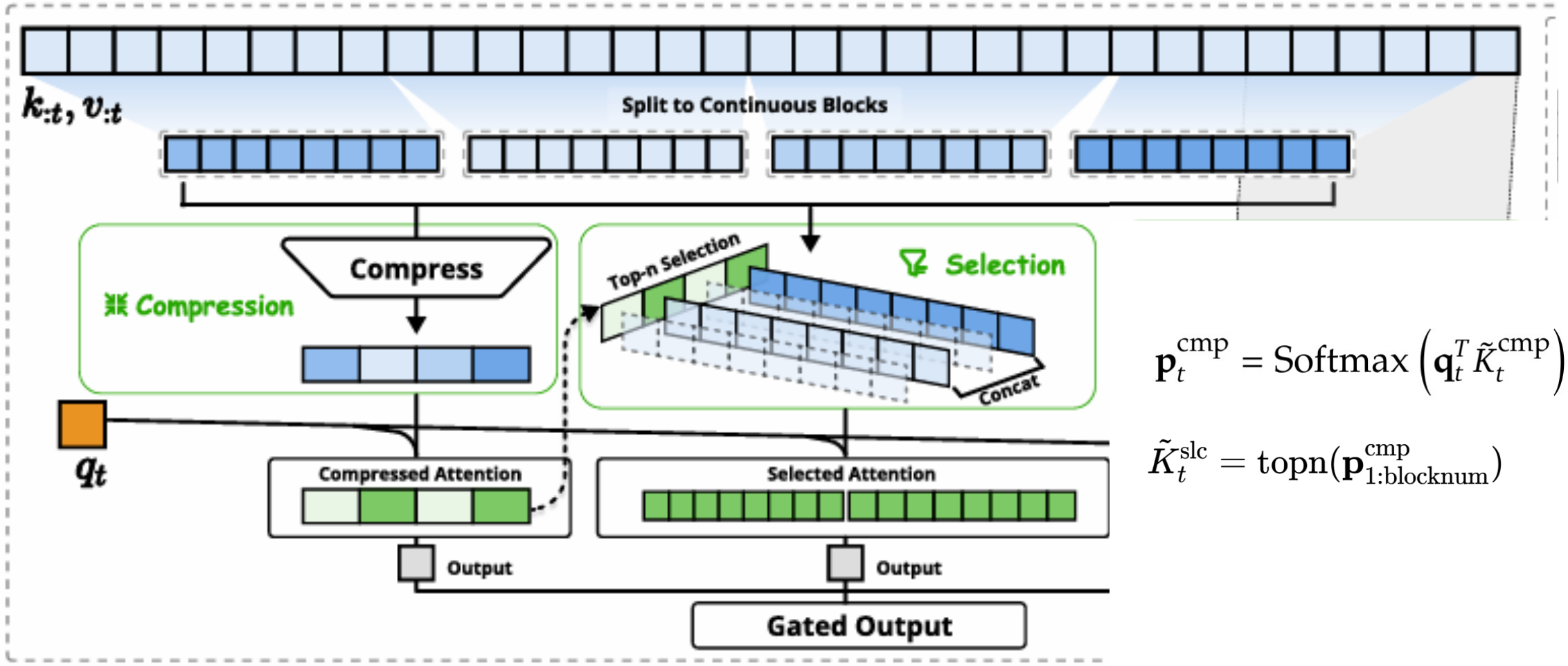
- With block-level representations for key and value, NSA compresses the block by a learnable MLP φ .



$$\tilde{K}_t^{\text{cmp}} = f_K^{\text{cmp}}(\mathbf{k}_{:t}) = \left\{ \varphi(\mathbf{k}_{id+1:id+l}) \middle| 0 \leqslant i \leqslant \left\lfloor \frac{t-l}{d} \right\rfloor \right\}$$

NSA: Token sclection

- After compression, NSA selects activated blocks by a TopK gate for the softmax score.

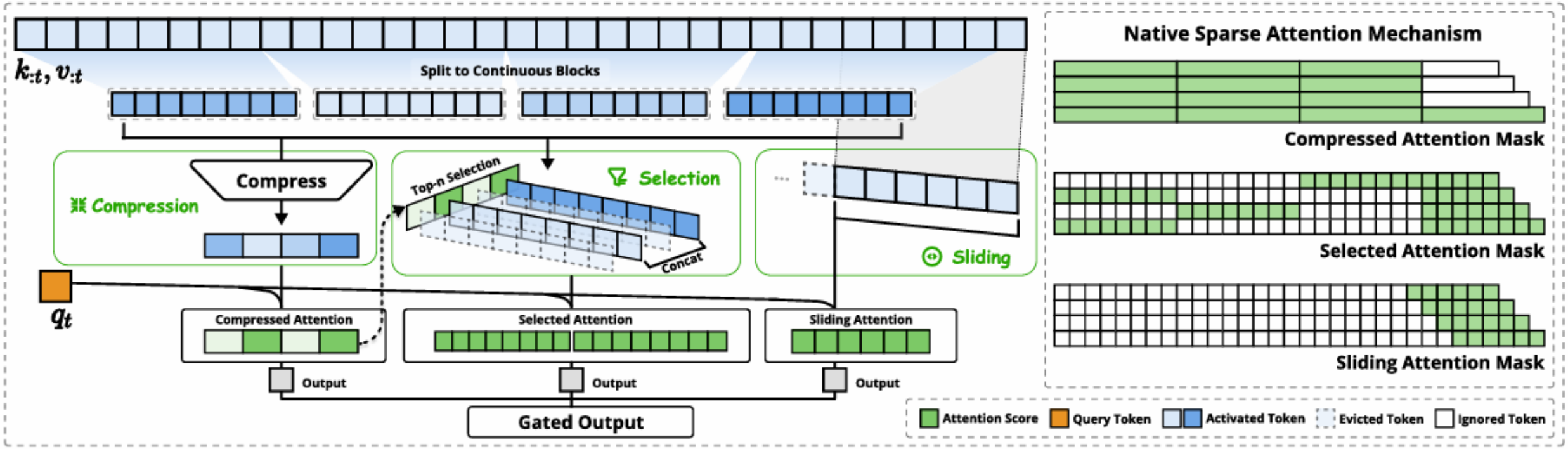


$$\mathbf{p}_t^{\text{cmp}} = \text{Softmax}\left(\mathbf{q}_t^T \tilde{K}_t^{\text{cmp}}\right)$$

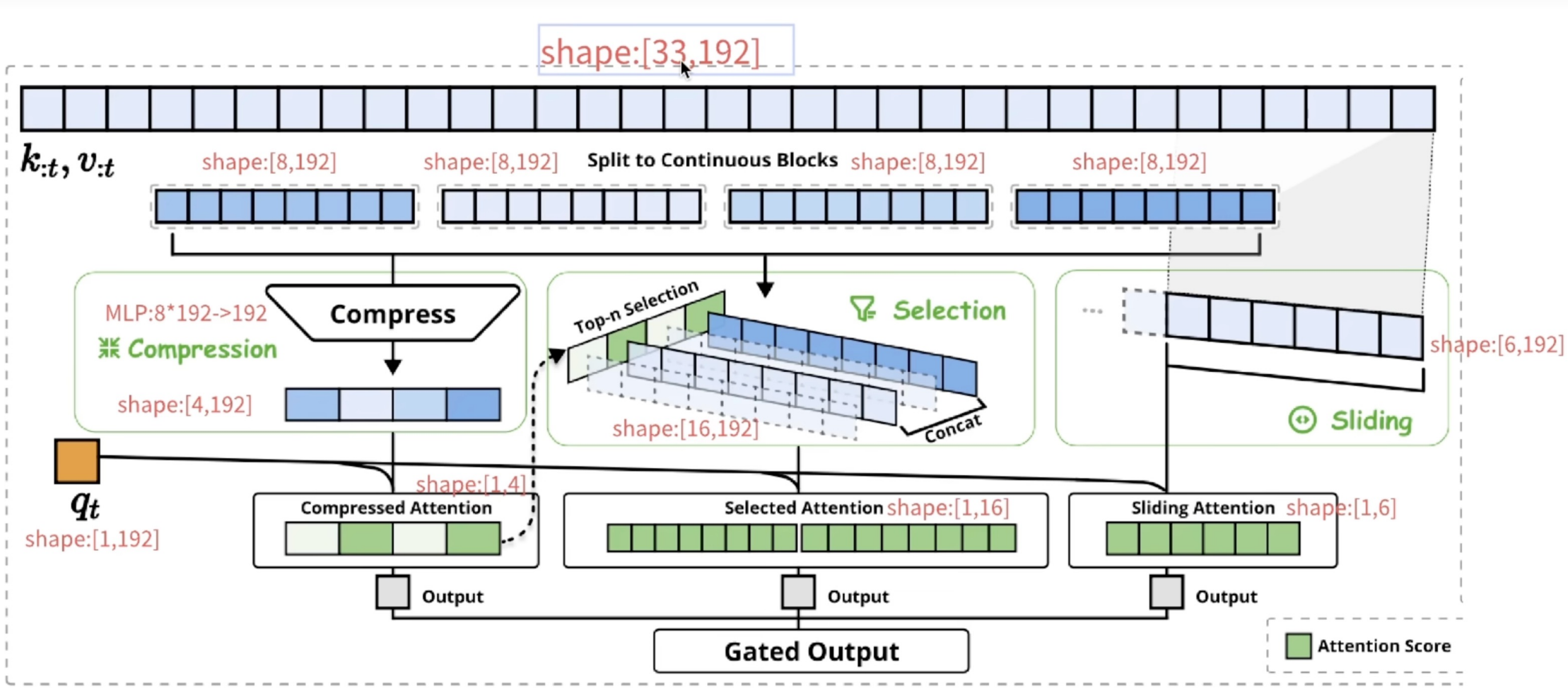
$$\tilde{K}_t^{\text{slc}} = \text{topn}(\mathbf{p}_{1:\text{blocknum}}^{\text{cmp}})$$

NSA: sliding attention

- NSA uses sliding attention due to the Attention sink



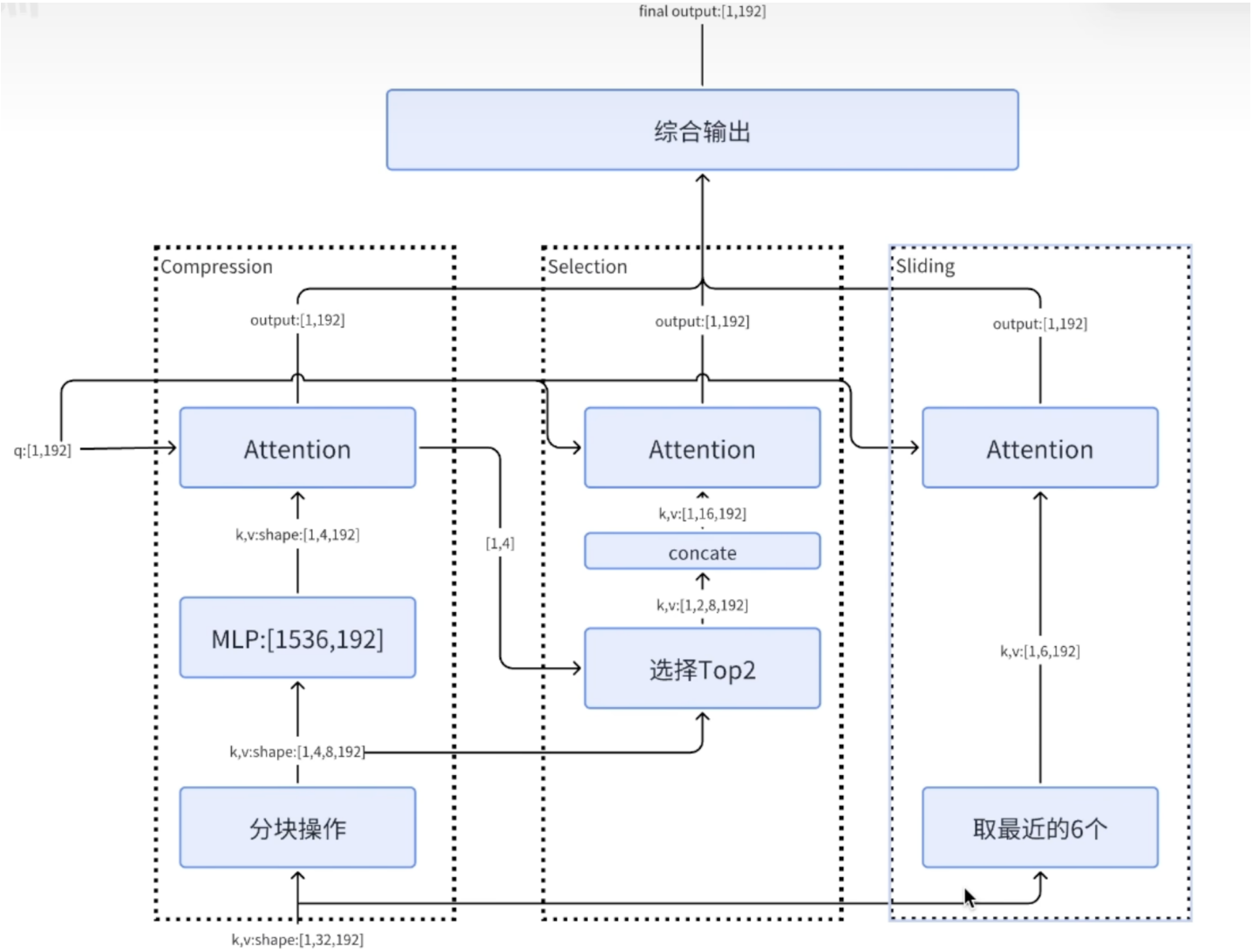
NSA: Token compression + Token sclection + sliding attention



Source: BiliBili Up主 “AI那些事儿”

$$\mathbf{o}_t^* = \sum_{c \in C} g_t^c \cdot \text{Attn}(\mathbf{q}_t, \tilde{K}_t^c, \tilde{V}_t^c). \quad \text{where} \quad C = \{\text{cmp}, \text{slc}, \text{win}\}$$

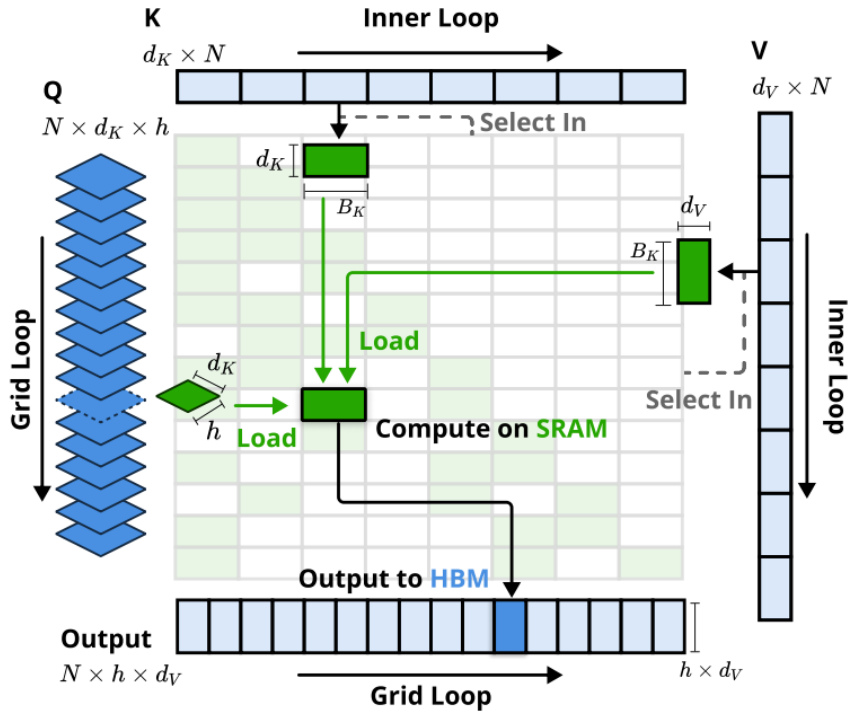
NSA: Token compression + Token sclection + sliding attention



Source: BiliBili Up主 “AI那些事儿”

NSA: kernel design

- NSA can be achieve the FlashAttention speedup with kernel design
  - Use GQA groups to load queries.
  - Taking block-wise operation in SRAM for selected key and value



NSA: single-step time

- NSA can achieve the step-wise speedup due to sparse architecture.
- Achieving 9.0× speedup than FlashAttention with long context

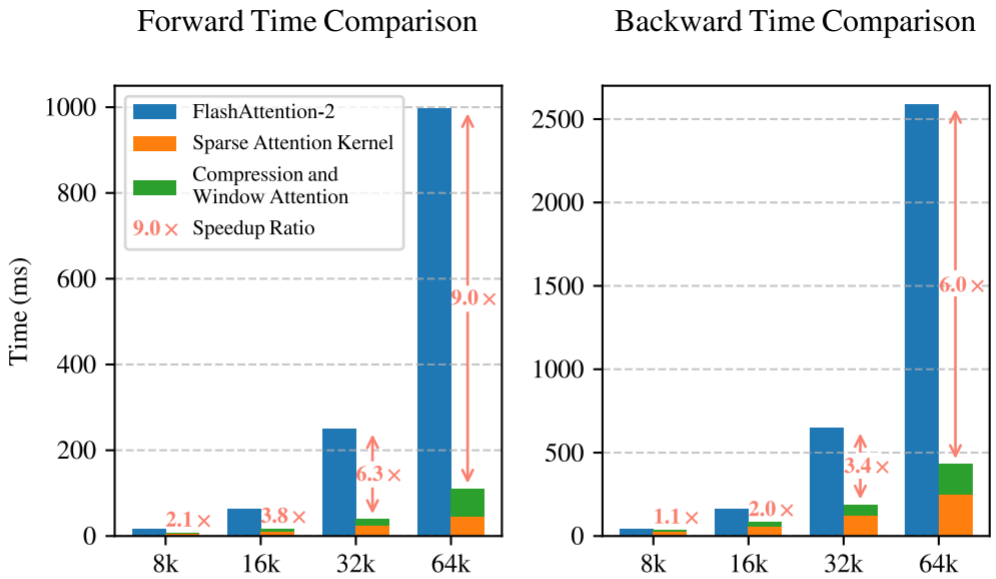
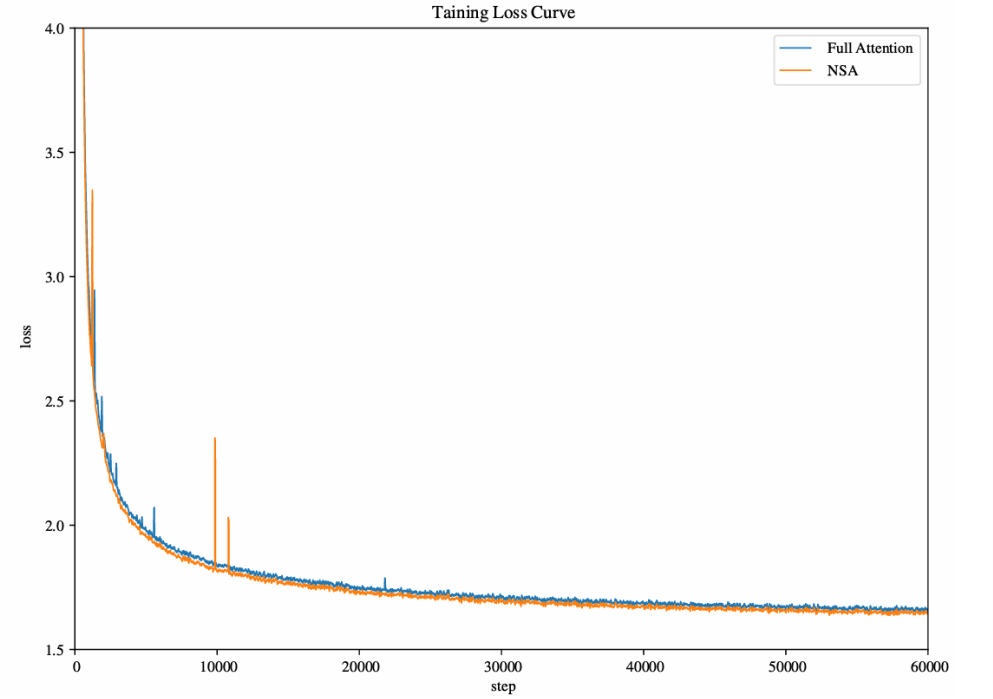


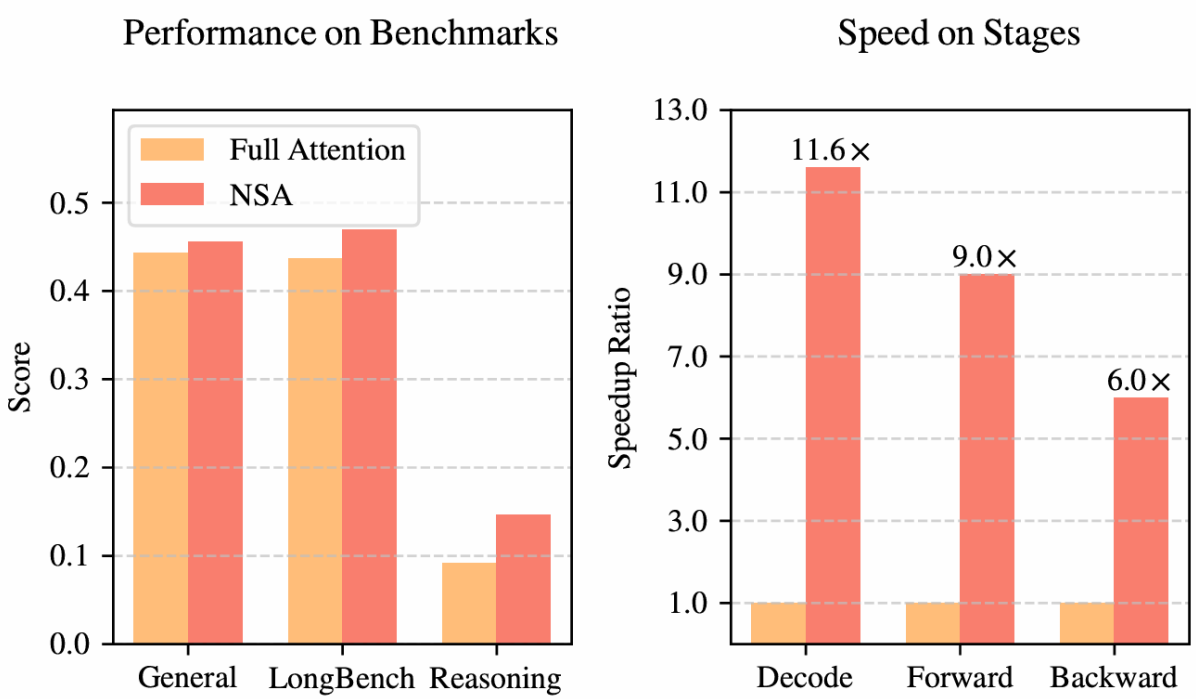
Figure 6 | Comparison of Triton-based NSA kernel with Triton-based FlashAttention-2 kernel. Our implementation significantly reduces latency across all context lengths, with the improvement becoming more pronounced as input length increases.

NSA: pre-training performance

- NSA can reduce the computation overhead due to the sparse architecture.
- NSA can achieve a better training-loss and downstream performance than full attention with the same steps.

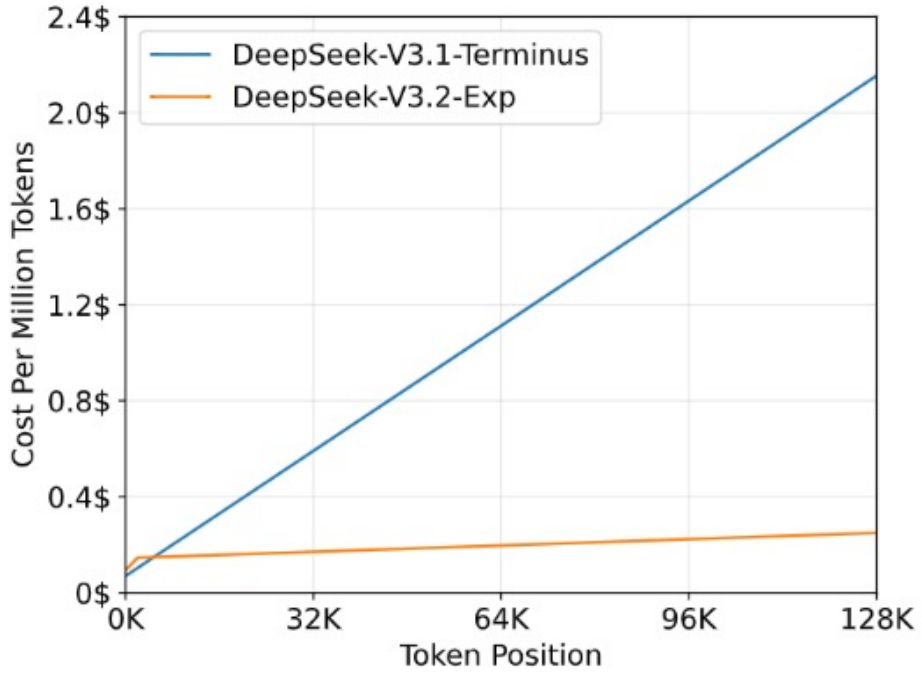


Training loss

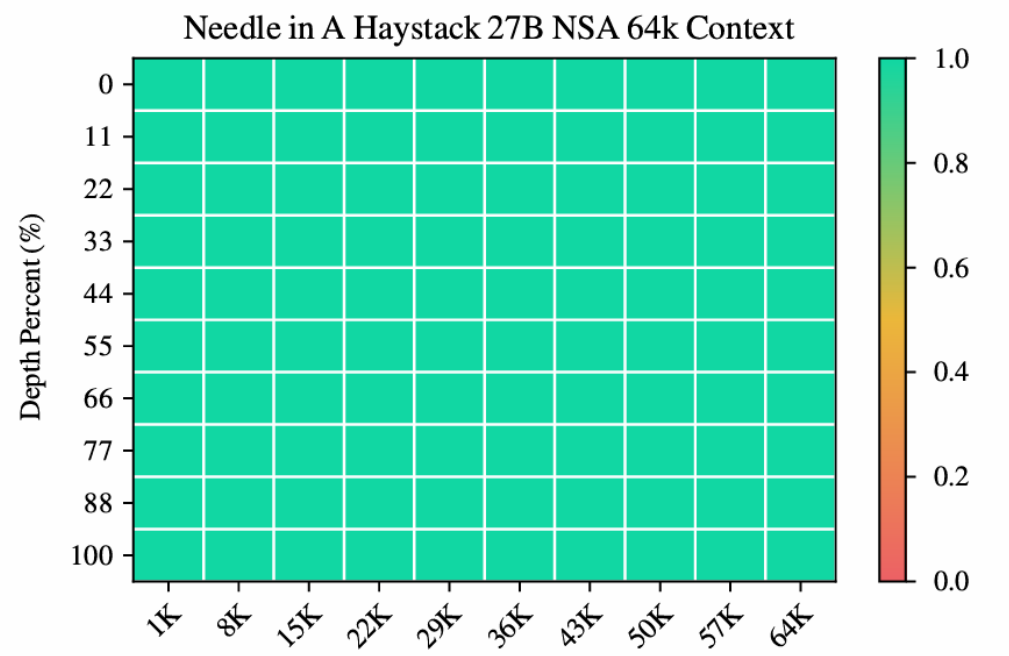


Downstream performance

NSA: inference performance



Cost for inference



Needle-in-a-Haystack performance

NSA: summary

北京大学 PEKING UNIVERSITY

- NSA is a sparse framework for Attention blocks.
- Three core design for sparsity: Token compression + Block wise sparsity + Sliding-window attention.
- Can be combined with FlashAttention.
- Reduce the computation for the attention, especially in long sequence case.

北京大学
PEKING UNIVERSITY

PART 03

Kernelized Linear Attention

Standard approach: Softmax attention

- Softmax attention:

$$O = \text{softmax}(QK^\top)V$$

- Here:
  - $Q = (q_1^\top, \cdots, q_L^\top)^\top \in \mathbb{R}^{L \times d_h}$
  - $K = (k_1^\top, \cdots, k_L^\top)^\top \in \mathbb{R}^{L \times d_h}$
  - $V = (v_1^\top, \cdots, v_L^\top)^\top \in \mathbb{R}^{L \times d_h}$
  - we consider a single attention head for convenience
  - we omit $\frac{1}{\sqrt{d_h}}$ for convenience.

Standard approach: Softmax attention

北京大学 PEKING UNIVERSITY

- Softmax Attention with causal mask:

$$O = \text{softmax}(QK^\top + M)V$$

- Here:
  - $M = \begin{pmatrix} 0 & -\infty & \cdots & -\infty \\ 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in (\mathbb{R} \cup \{-\infty\})^{L \times L}$ is an additive causal mask

- To compute the attention score for q_i, we must know all previous keys k_j, $j = 1,2,\cdots,i$. The **memory cost** scales linearly with the sequence length!

Breaking apart: What does softmax attention compute?

- For query q_i, softmax attention actually computes:

$$
\begin{aligned}
o_i =& \text{softmax}(q_i^\top k_1, \cdots, q_i^\top k_i) V_{:i} \\
=& \sum_{j \le i} \frac{\exp(q_i^\top k_j)}{\sum_{\ell \le i} \exp(q_i^\top k_\ell)} \cdot v_j \\
=& \frac{\sum_{j \le i} \exp(q_i^\top k_j) v_j}{\sum_{j \le i} \exp(q_i^\top k_j)}
\end{aligned}
$$

Replacing the exponential function

- Consider replace the exp operation with the following format:

$$\exp(x^\top y) \approx \phi(x)^\top \phi(y)$$

Here, $\phi : \mathbb{R}^{d_h} \to \mathbb{R}^{d_k}$ is called **kernel function**.

- For example, if we let $\phi(x) = x$, the kernel function is equivalent to identity mapping. It corresponds to simply removing the exponential operation, a.k.a. **linear attention**.

Constant memory for kernelized linear attention

- For query q_i, kernelized linear attention actually computes:

$$
\begin{aligned}
o_i &= \frac{\sum_{j \le i} \phi(q_i)^\top \phi(k_j) v_j}{\sum_{j \le i} \phi(q_i)^\top \phi(k_j)} \\
&= \frac{\sum_{j \le i} v_j \phi(k_j)^\top \phi(q_i)}{\sum_{j \le i} \phi(k_j)^\top \phi(q_i)} \\
&= \frac{\sum_{j \le i} v_j \phi(k_j)^\top \cdot \phi(q_i)}{\sum_{j \le i} \phi(k_j)^\top \cdot \phi(q_i)} \\
&= \frac{(\sum_{j \le i-1} v_j \phi(k_j)^\top + v_i \phi(v_i)^\top)\phi(q_i)}{(\sum_{j \le i-1} \phi(k_j)^\top + \phi(k_i)^\top)\phi(q_i)}
\end{aligned}
$$

Constant memory for kernelized linear attention

北京大学 PEKING UNIVERSITY

- Consequently, we only need to maintain

$$S_i = \sum_{j \le i} v_j \phi(k_j)^\top \quad \text{and} \quad T_i = \sum_{j \le i} \phi(k_j)^\top,$$

and compute o_i recursively:

$$\begin{cases} S_i = S_{i-1} + v_i \phi(k_i)^\top, \\ T_i = T_{i-1} + \phi(k_i)^\top, \\ o_i = \frac{S_i \phi(q_i)}{T_i \phi(q_i)}, \end{cases}$$

- Memory cost $O(d_k(d_h+1))$ is independent on sequence length L

Kernelized Linear Attention: Training v.s. Inference

- Inference: RNN-like **recursive** computation

$$\begin{cases} S_i = S_{i-1} + v_i \phi(k_i)^\top, \\ T_i = T_{i-1} + \phi(k_i)^\top, \\ o_i = \frac{S_i \phi(q_i)}{T_i \phi(q_i)}, \end{cases}$$

- Training: Transformer-like **parallel** computation

$$O = \mathrm{Normalize}((\phi(Q)\phi(K)^\top) \odot M')V$$

Here $M' = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \in \mathbb{R}^{L \times L}$ is a multiplicative causal mask

$$\phi(Q) = (\phi(q_1)^\top, \cdots, \phi(q_L)^\top)^\top, \qquad \phi(K) = (\phi(k_1)^\top, \cdots, \phi(k_L)^\top)^\top$$

and $[\mathrm{Normalize}(X)]_{ij} = \frac{x_{ij}}{\sum_k x_{ik}}$,

Kernelized Linear Attention v.s. Standard Softmax Attention

北京大学 PEKING UNIVERSITY

	Training	Inference
Softmax attention	$O = \text{softmax}(QK^\top + M)V$ $= \text{Normalize}(\exp(QK^\top) \odot M')V$	$o_i = \text{softmax}(q_i^\top k_1, \cdots, q_i^\top k_i)V_{:i}$
Kernelized Linear attention	$O = \text{Normalize}((\phi(Q)\phi(K)^\top) \odot M')V$	$\begin{cases} S_i = S_{i-1} + v_i \phi(k_i)^\top, \\ T_i = T_{i-1} + \phi(k_i)^\top, \\ o_i = \frac{S_i \phi(q_i)}{T_i \phi(q_i)}, \end{cases}$

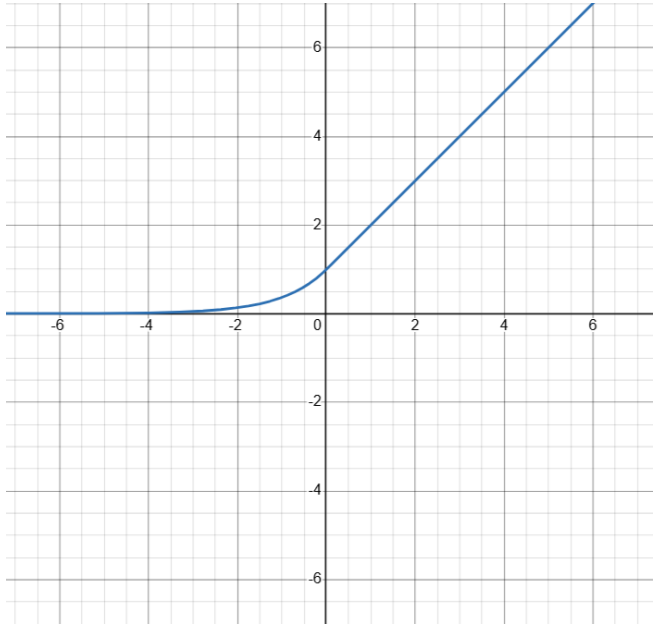
- The major benefit of kernelized linear attention lies in the reduction of **inference-time memory** cost

Kernel functions

- Katharopoulos kernel:

$$\phi(x) = \mathrm{elu}(x) + 1 = \begin{cases} \exp(x), & x \leq 0 \\ x + 1, & x > 0 \end{cases}$$

- Katharopoulos kernel is a heuristic kernel, not designed to mimic the exponential operation.



Kernel functions

- FAVOR+ (Performer):

$$\exp(x^\top y) = \mathbb{E}_{\omega\sim\mathcal{N}(0,I_d)}[\exp(\omega^\top x - \|x\|^2/2)\exp(\omega^\top y - \|y\|^2/2)]]$$

Proof hint: $\mathbb{E}\left[\exp\left(\omega^\top(x+y)\right)\right] = \exp\left(\frac{\|x+y\|^2}{2}\right).$

Thus, we have the following kernel function:

$$\phi(x) = \frac{1}{\sqrt{m}}(\exp(\omega_1^\top x - \|x\|^2/2), \cdots, \exp(\omega_m^\top x - \|x\|^2/2))^\top$$

- FAVOR+ kernel is a stochastic kernel that provides an **unbiased** estimation of softmax attention.

北京大学
PEKING UNIVERSITY

PART 04

Hardware-Efficient Training of Linear Attention

Linear Attention: Inference

北京大学 PEKING UNIVERSITY

- Inference (Recurrent Form):

$$S_t = S_{t-1} + v_t k_t^T \in \mathbb{R}^{d \times d}$$

$$o_t = S_t q_t \in \mathbb{R}^d$$

- Memory: $O(d^2)$, we only need to store S_t

  Softmax Attention: $O(Ld)$ KV Cache

- Computation: $O(Ld^2)$, L is the sequence length

  Softmax Attention: $O(L^2 d)$

Inference of Linear Attention is very efficient for long context sequences

Why not Train in Recurrent form

$$S_t = S_{t-1} + v_t k_t^T \in \mathbb{R}^{d \times d}$$
$$o_t = S_t q_t \in \mathbb{R}^d$$

- Strict sequential computation, lacking sequence parallelism.
- Lacking matmul operations, cannot be optmizes in modern GPUs.
- Saving all S_t for backward requires $O(Ld^2)$ memory, which is unacceptable.

Linear Attention: Training

- In training, we use parallel form:

$$S_t = S_{t-1} + v_t k_t^T \in \mathbb{R}^{d \times d}$$
$$o_t = S_t q_t \in \mathbb{R}^d$$

$$O = (QK^T \odot M)V \in \mathbb{R}^{L \times d}$$

Parallelizable

- We cannot remove the mask to make it be like $Q(K^T V)$

- Memory: $O(Ld)$, if we use flash attention.
- Computation: $O(L^2 d)$

The training complexity of linear attention is the same as softmax attention.

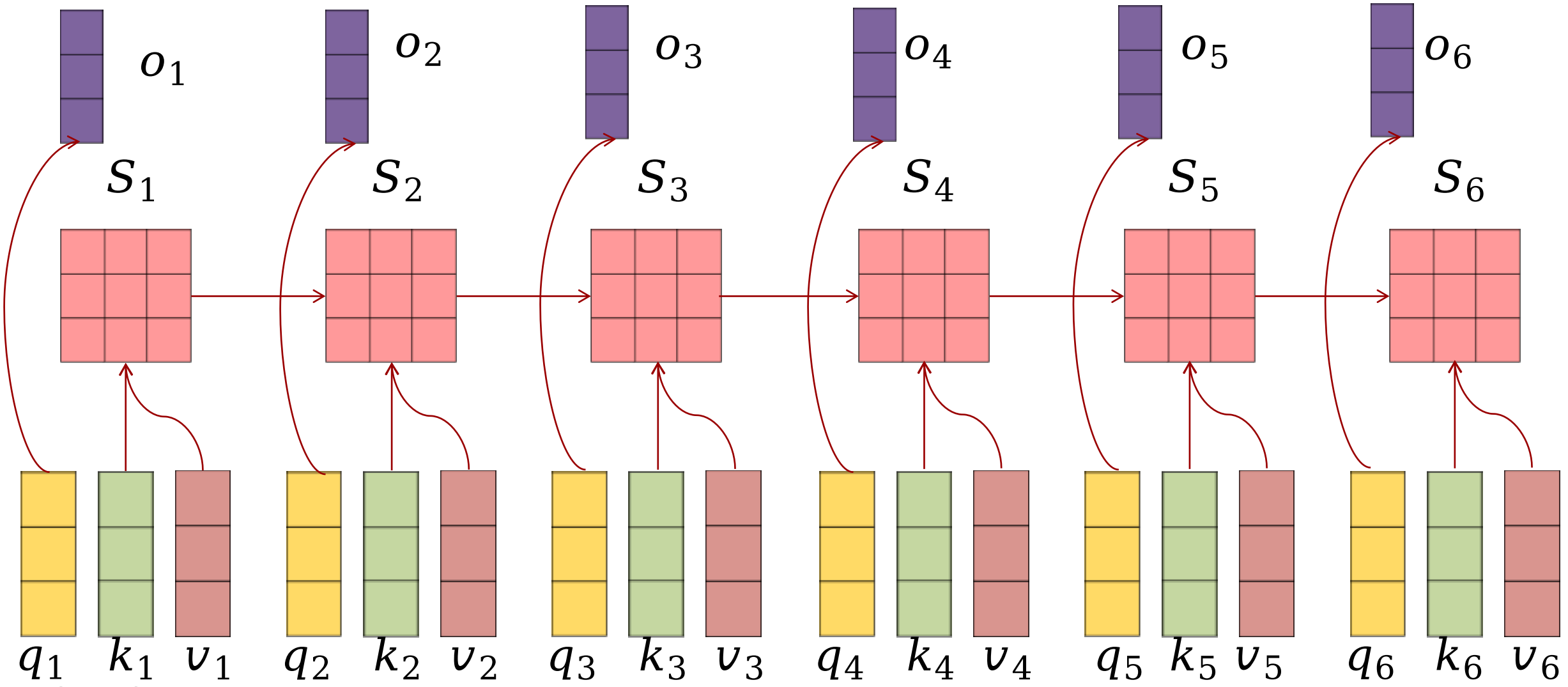
Chunkwise Form

$$S_L = \sum_{i=1}^{L} \mathbf{v}_i \mathbf{k}_i^\top = \mathbf{V}^\top \mathbf{K}$$

$$\text{where } \mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_L]^\top \in \mathbb{R}^{L \times d}, \quad \mathbf{K} = [\mathbf{k}_1, \ldots, \mathbf{k}_L]^\top \in \mathbb{R}^{L \times d}.$$

- Outer-product structure enables high parallelism by using matrix multiplication, which is highly optimized by tensor cores in modern GPUs.
- We can split the tokens into chunks, compute the intemediate state S, and compute the output of any position in parallel.

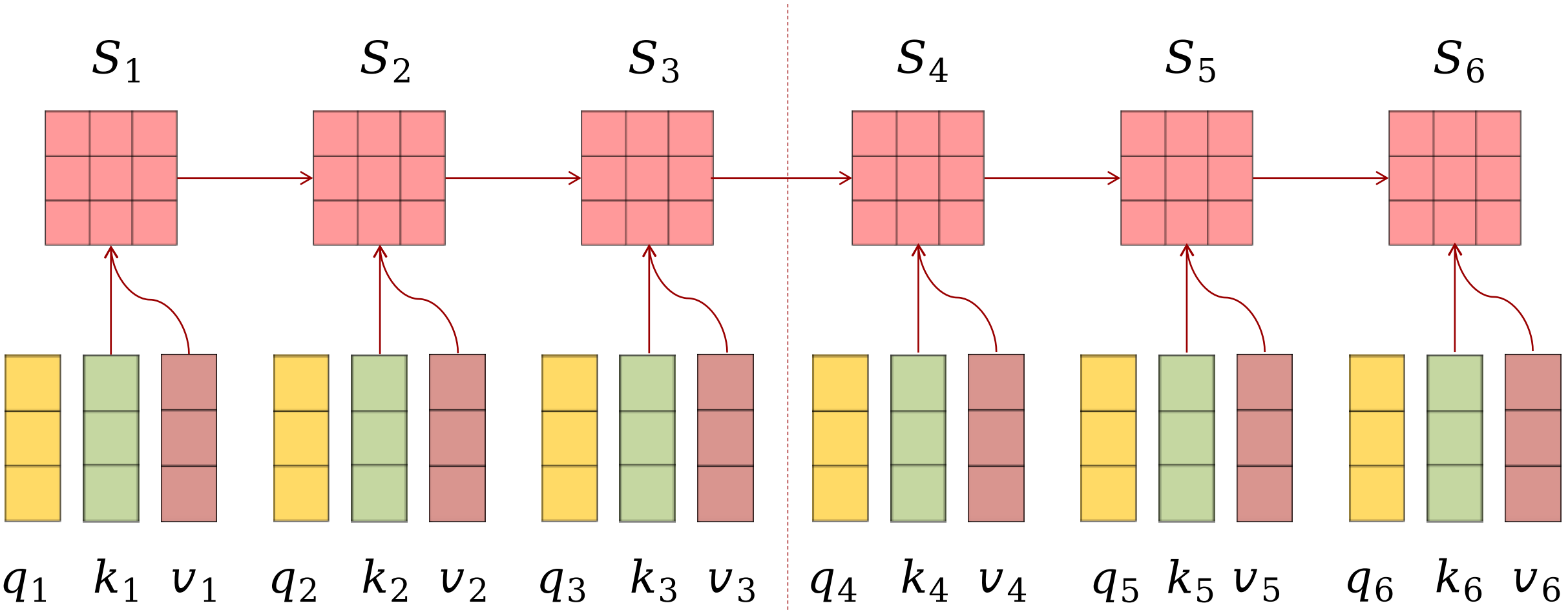
Chunkwise Form



Chunkwise Form

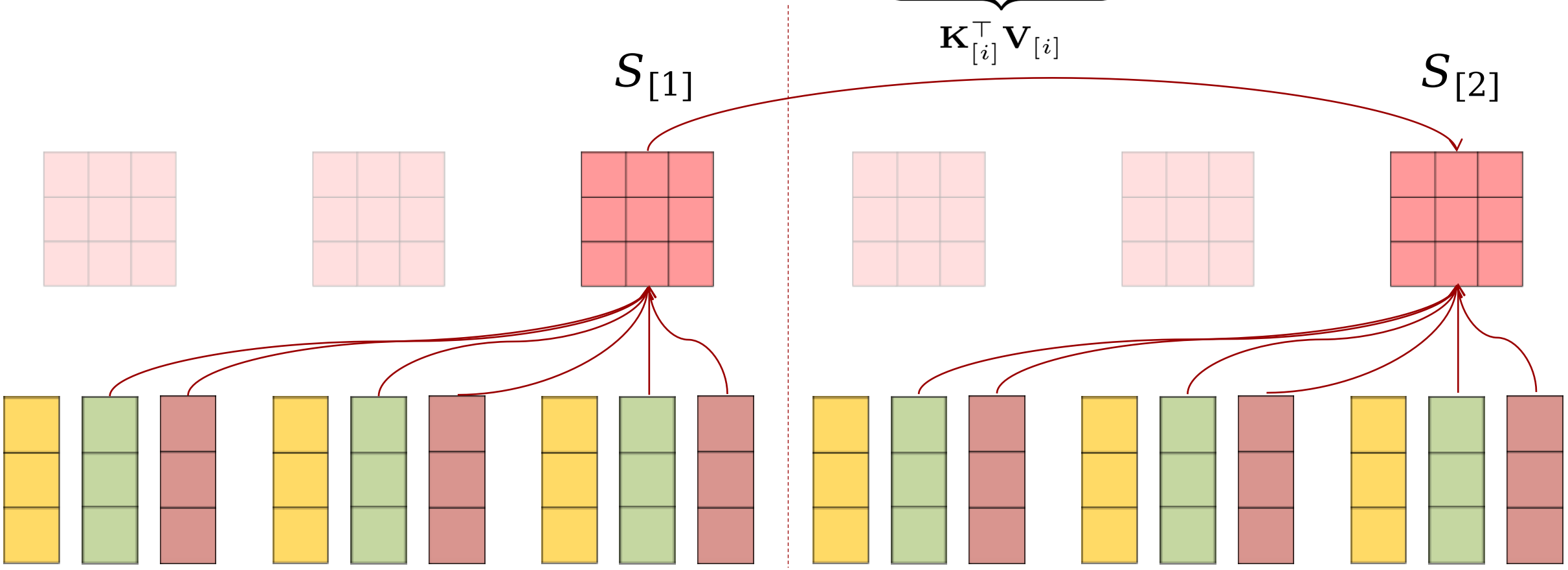
$$S_t = S_{t-1} + v_t k_t^T \in \mathbb{R}^{d \times d}$$

$$o_t = S_t q_t \in \mathbb{R}^d$$



Chunkwise Form: Step 1

$$\mathbf{S}_{[i+1]} = \mathbf{S}_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \mathbf{k}_j^\top \mathbf{v}_j}_{\mathbf{K}_{[i]}^\top \mathbf{V}_{[i]}}$$

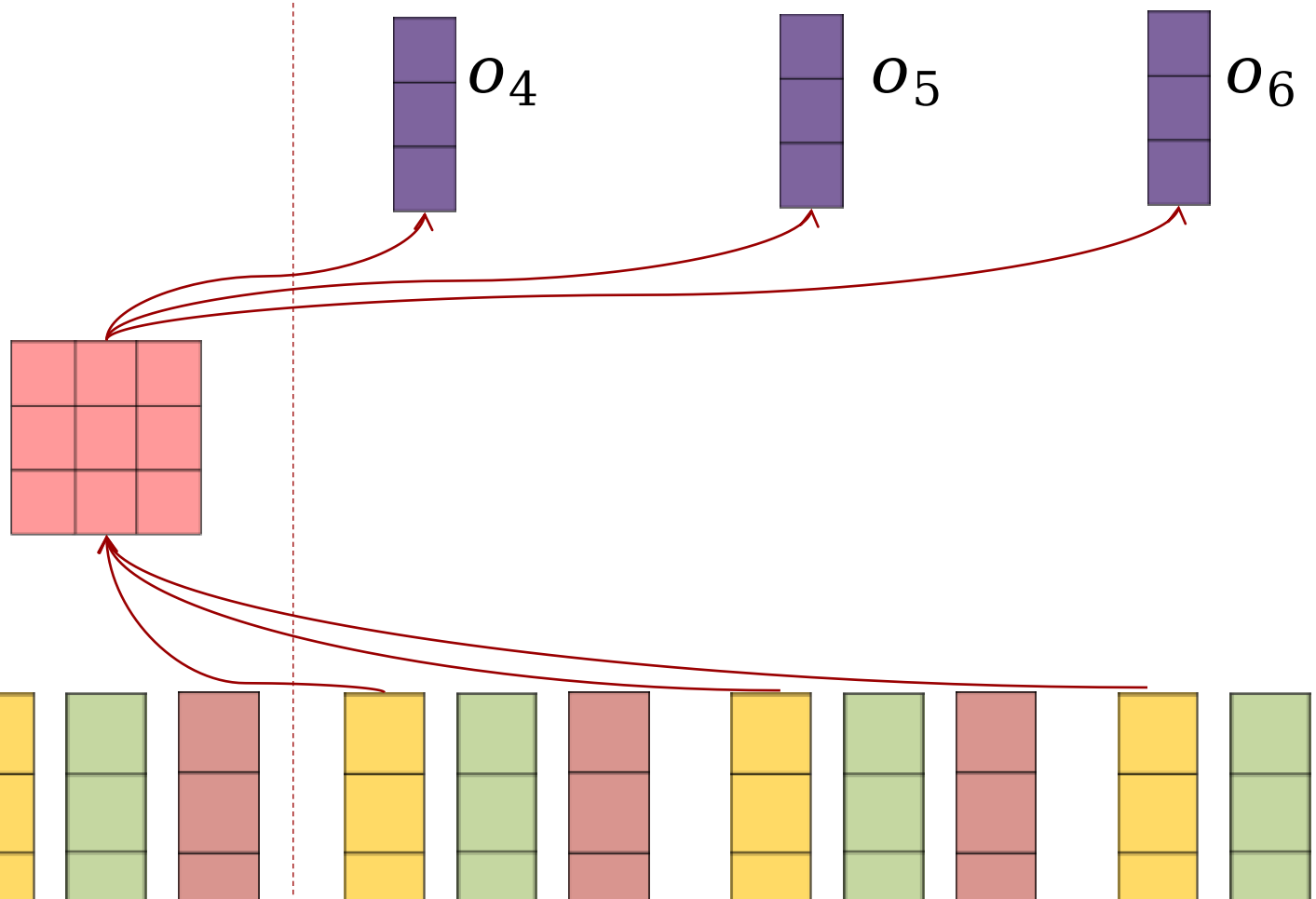


$S_{[i]} \in R^{d \times d}$ means the last state of the i-th chunk.

$Q_{[i]} = [q_{iC+1}, \ldots, q_{(i+1)C}]^T \in R^{C \times d}$ ($K_{[i]}$ *and* $V_{[i]}$ similarly), C is the chunk size.

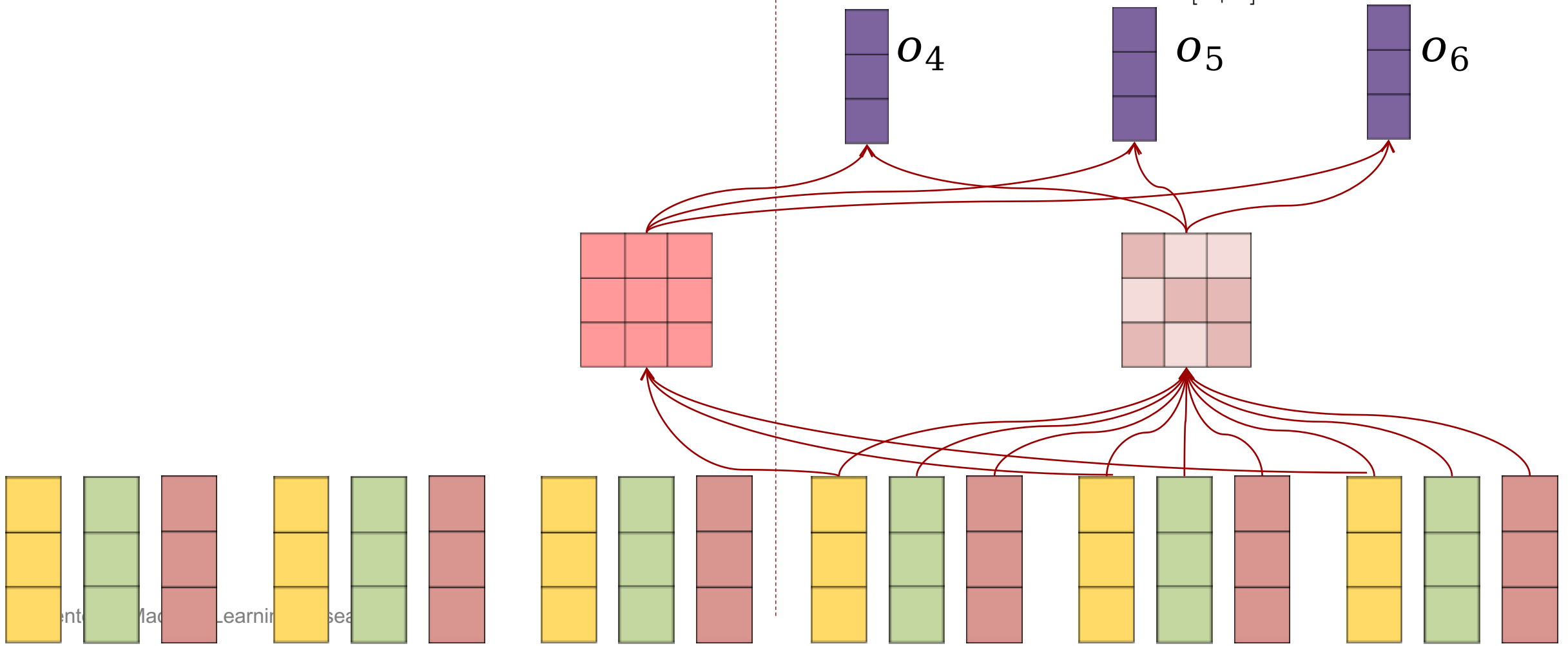
Chunkwise Form: Step2

$$\mathbf{O}_{[i+1]} = \underbrace{\mathbf{Q}_{[i+1]}\mathbf{S}_{[i]}}_{\text{inter-chunk: } \mathbf{O}^{\text{inter}}_{[i+1]}} + \mathbf{O}^{\text{intra}}_{[i+1]}$$



Chunkwise Form: Step2

$$\mathbf{O}_{[i+1]} = \underbrace{\mathbf{Q}_{[i+1]}\mathbf{S}_{[i]}}_{\text{inter-chunk: } \mathbf{O}^{\text{inter}}_{[i+1]}} + \underbrace{\left((\mathbf{Q}_{[i+1]}\mathbf{K}^{\top}_{[i+1]}) \odot \mathbf{M}\right)\mathbf{V}_{[i+1]}}_{\text{intra-chunk: } \mathbf{O}^{\text{intra}}_{[i+1]}}$$



Computational Complexity of Chunkwise Training

$$\mathbf{S}_{[t+1]} = \underbrace{\mathbf{S}_{[t]}}_{\mathbb{R}^{d\times d}} + \underbrace{\mathbf{V}_{[t]}^{\top}\mathbf{K}_{[t]}}_{\mathbb{R}^{d\times C}\quad \mathbb{R}^{C\times d}} \in \mathbb{R}^{d\times d}$$

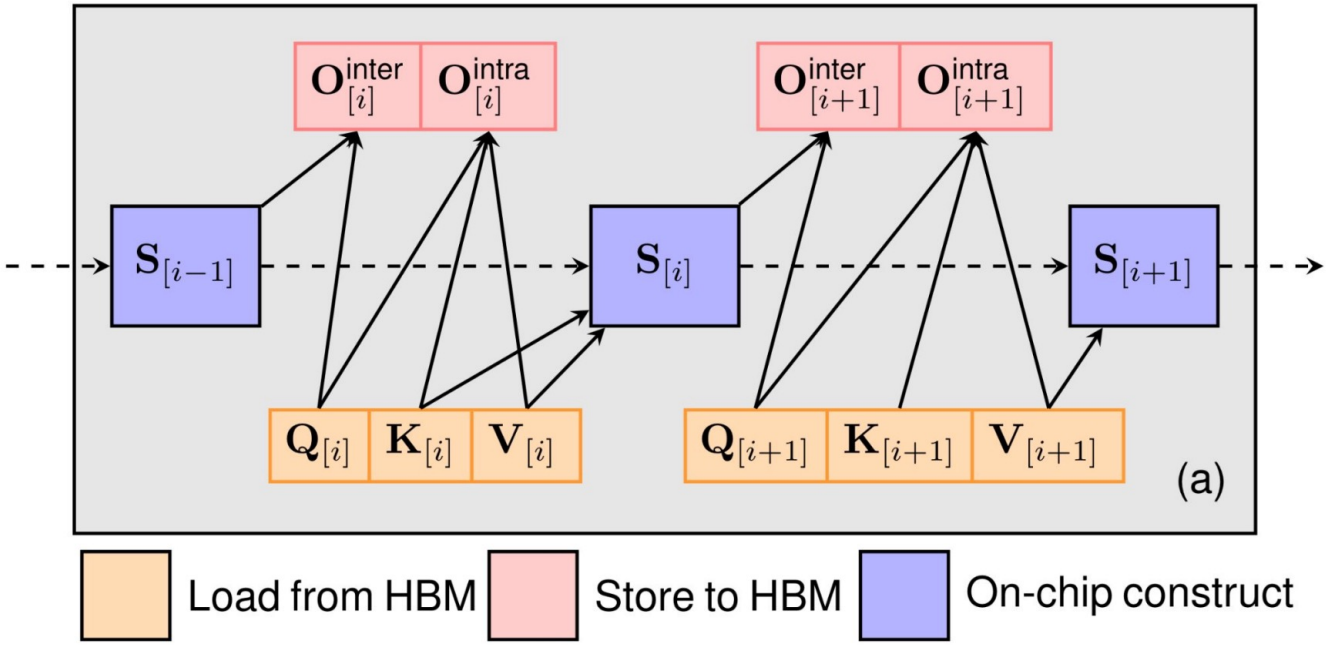
- $O(Cd^2)$ for each chunk, and $O(Ld^2)$ for the whole sequence.

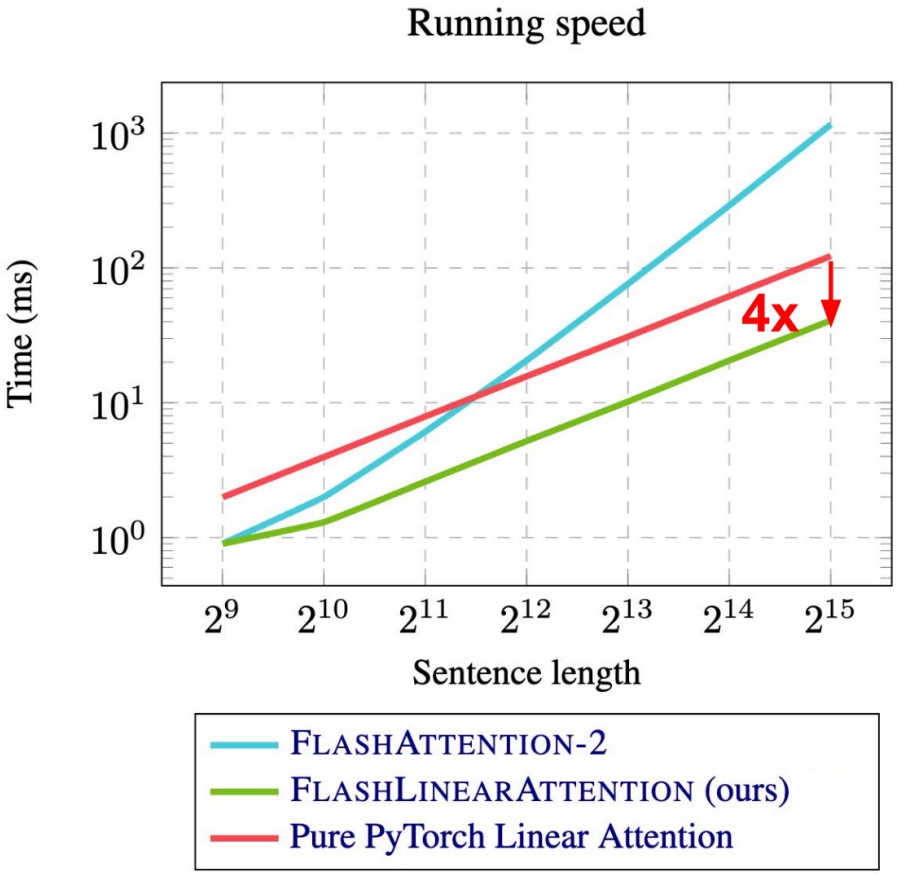
$$\mathbf{O}_{[t]} = \underbrace{\mathbf{Q}_{[t]}\mathbf{S}_{[t]}^{\top}}_{\text{inter-chunk: } \mathbf{O}_{[t]}^{\text{inter}}} + \underbrace{\left(\mathbf{Q}_{[t]}\mathbf{K}_{[t]}^{\top} \odot \mathbf{M}\right)\mathbf{V}_{[t]}}_{\text{intra-chunk: } \mathbf{O}_{[t]}^{\text{intra}}} \in \mathbb{R}^{C\times d}$$

- $O(C^2d + Cd^2)$ for each chunk, and $O(Ld^2 + LCd)$ for the whole sequence. C is the chunk size, and is in {64, 128, 256} in practice.

Hardware-Efficient Opmization

- Flash Linear Attention further optimizes chunkwise training through kernel fusion.





[FlashLinearAttention: Hardware-Efficient Algorithm for Linear Attention]

北京大学
PEKING UNIVERSITY

PART 05

DeltaNet

Test Time Training(TTT) View

- Linear Attention is using a constant memory S to memorize kv pairs : $(k_1, v_1), (k_2, v_2), \ldots, (k_t, v_t)$
- TTT idea: Design a model $v = f(S; k)$, use the above kv pairs to train the model weight S. Finally the model memorizes those kv pairs.
- Linear Attention is a type of TTT :
  - $f(S; k_t) = Sk_t$
  - $L_t(S) = -< Sk_t, v_t >$
  - SGD update: $S_t = S_{t-1} - \nabla L_t(S_{t-1}) = S_{t-1} + v_t k_t^T$

DeltaNet

- Vanilla Linear Attention :
  - $f(S; k_t) = Sk_t$
  - $L_t(S) = -< Sk_t, v_t >$
  - SGD update: $S_t = S_{t-1} - \nabla L_t(S_{t-1}) = S_{t-1} + v_t k_t^T$

- DeltaNet has a better objective :
  - $L_t(S) = \frac{1}{2}\|Sk_t - v_t\|^2$
  - SGD update: $S_t = S_{t-1} - \beta_t \nabla L_t(S_{t-1}) = S_{t-1}(I - \beta_t k_t k_t^T) + \beta_t v_t k_t^T$

Associative Memory View

- Linear Attention build key-value memory using outer products:

$$\mathbf{S} = \sum_i \mathbf{v}_i \mathbf{k}_i^\top$$

- To retrieve the value via a key (k_i are l2-normalized):

$$\mathbf{S}\mathbf{k}_j = \underbrace{\mathbf{v}_j}_{\text{desired}} + \underbrace{\sum_{i \neq j} \left(\mathbf{k}_i^\top \mathbf{k}_j\right) \mathbf{v}_i}_{\text{retrieval error}}$$

- The retrieval error grows unbounded as the sequence length increases.

[DeltaNet Explained, Songlin Yang]
[Understanding Transformer from the Perspective of Associative Memory]

Associative Memory View

北京大学 PEKING UNIVERSITY

- DeltaNet Idea: not only write new values, but also remove old values:

$$\mathbf{v}_t^{\text{old}} = \mathbf{S}_{t-1}\mathbf{k}_t$$

$$\mathbf{v}_t^{\text{new}} = \beta_t \mathbf{v}_t + (1-\beta_t)\mathbf{v}_t^{\text{old}}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \underbrace{\mathbf{v}_t^{\text{old}}\mathbf{k}_t^\top}_{\text{remove old}} + \underbrace{\mathbf{v}_t^{\text{new}}\mathbf{k}_t^\top}_{\text{write new}}$$

- Then We get $S_t = S_{t-1}(I - \beta_t k_t k_t^T) + \beta_t v_t k_t^T$

Gated Delta Net

- Just like traditional RNNs, Gated DeltaNet adds a gating mechanism on DeltaNet:

$$S_t = S_{t-1}(\alpha_t(I - \beta_t k_t k_t^T)) + \beta_t v_t k_t^T$$
$$\alpha_t = \sigma(W_g x_t) \in (0, 1)$$

- It is simple but effective.
- In-context retrieving tasks:

	S-NIAH-1 (pass-key retrieval)				S-NIAH-2 (number in haystack)				S-NIAH-3 (uuid in haystack)		
Model	1K	2K	4K	8K	1K	2K	4K	8K	1K	2K	4K
DeltaNet	97.4	96.8	**99.0**	**98.8**	98.4	45.6	18.6	14.4	85.2	47.0	22.4
Mamba2	**99.2**	**98.8**	65.4	30.4	99.4	98.8	56.2	17.0	64.4	47.6	4.6
Gated DeltaNet	98.4	88.4	91.4	91.8	**100.0**	**99.8**	**92.2**	**29.6**	**86.6**	**84.2**	**27.6**

[Gated Delta Networks: Improving Mamba2 with Delta Rule]

Gated Delta Net

北京大学 PEKING UNIVERSITY

- Just like traditional RNNs, Gated DeltaNet adds a gating mechanism on DeltaNet:

$$S_t = S_{t-1}(\alpha_t(I - \beta_t k_t k_t^T)) + \beta_t v_t k_t^T$$
$$\alpha_t = \sigma(W_g x_t) \in (0, 1)$$

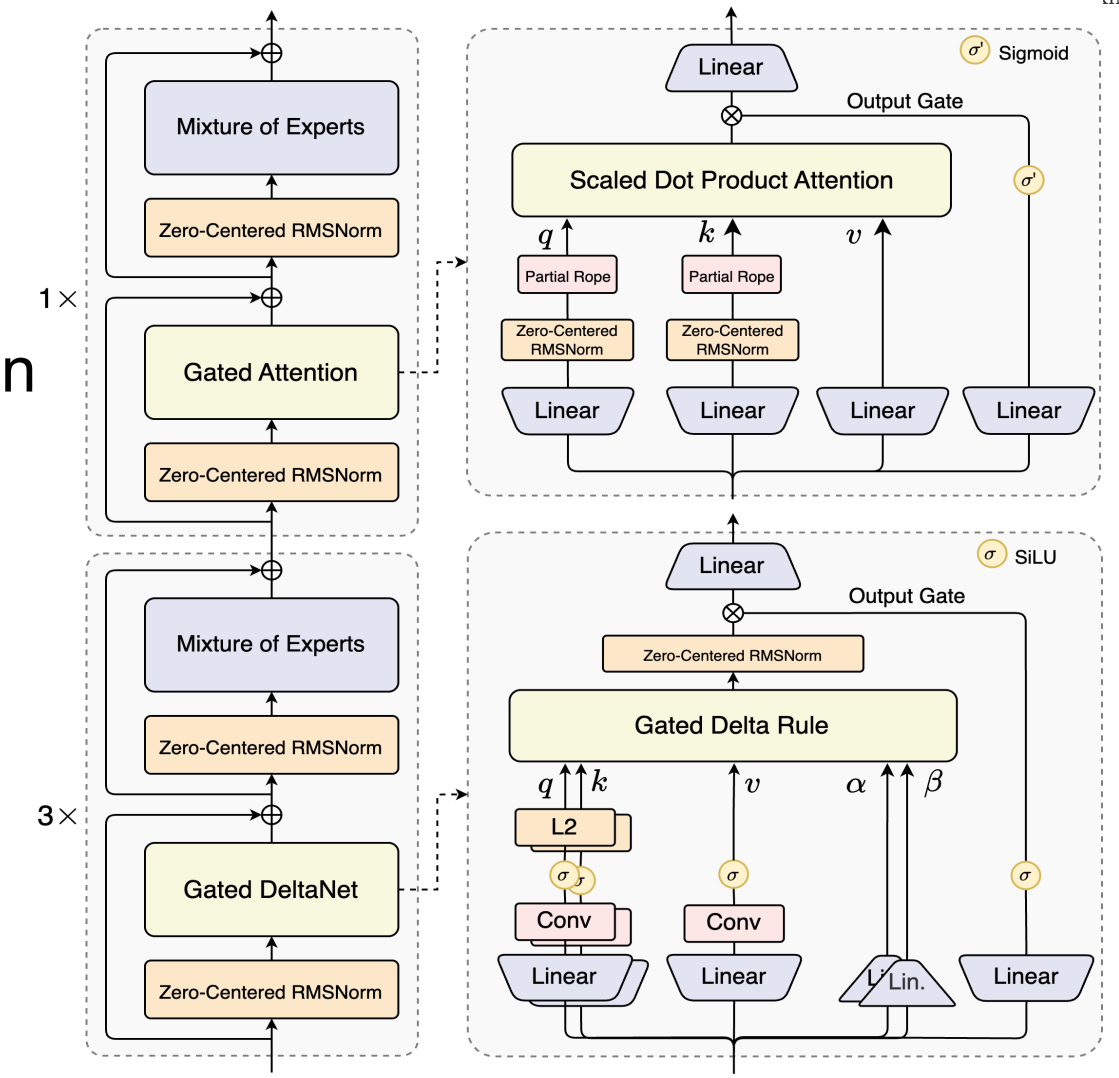
- Language Modelling

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	SIQA acc ↑	BoolQ acc ↑	Avg.
Recurrent models											
RetNet	19.08	17.27	40.52	70.07	49.16	54.14	67.34	33.78	**40.78**	60.39	52.02
HGRN2	19.10	17.69	39.54	70.45	49.53	52.80	69.40	35.32	40.63	56.66	51.79
Mamba	17.92	15.06	43.98	71.32	52.91	52.95	69.52	35.40	37.76	**61.13**	53.12
Mamba2	16.56	12.56	45.66	71.87	55.67	55.24	**72.47**	37.88	40.20	60.13	54.89
DeltaNet	17.71	16.88	42.46	70.72	50.93	53.35	68.47	35.66	40.22	55.29	52.14
Gated DeltaNet	**16.42**	**12.17**	**46.65**	**72.25**	**55.76**	**57.45**	71.21	**38.39**	40.63	60.24	**55.32**
Attention or hybrid models											
Transformer++	18.53	18.32	42.60	70.02	50.23	53.51	68.83	35.10	40.66	57.09	52.25
Samba	16.13	13.29	44.94	70.94	53.42	55.56	68.81	36.17	39.96	62.11	54.00
Gated DeltaNet-H1	16.07	**12.12**	47.73	**72.57**	56.53	**58.40**	71.75	**40.10**	41.40	**63.21**	**56.40**
Gated DeltaNet-H2	**15.91**	12.55	**48.76**	72.19	**56.88**	57.77	71.33	39.07	**41.91**	61.55	56.18

Hybrid Attention

Qwen3-Next

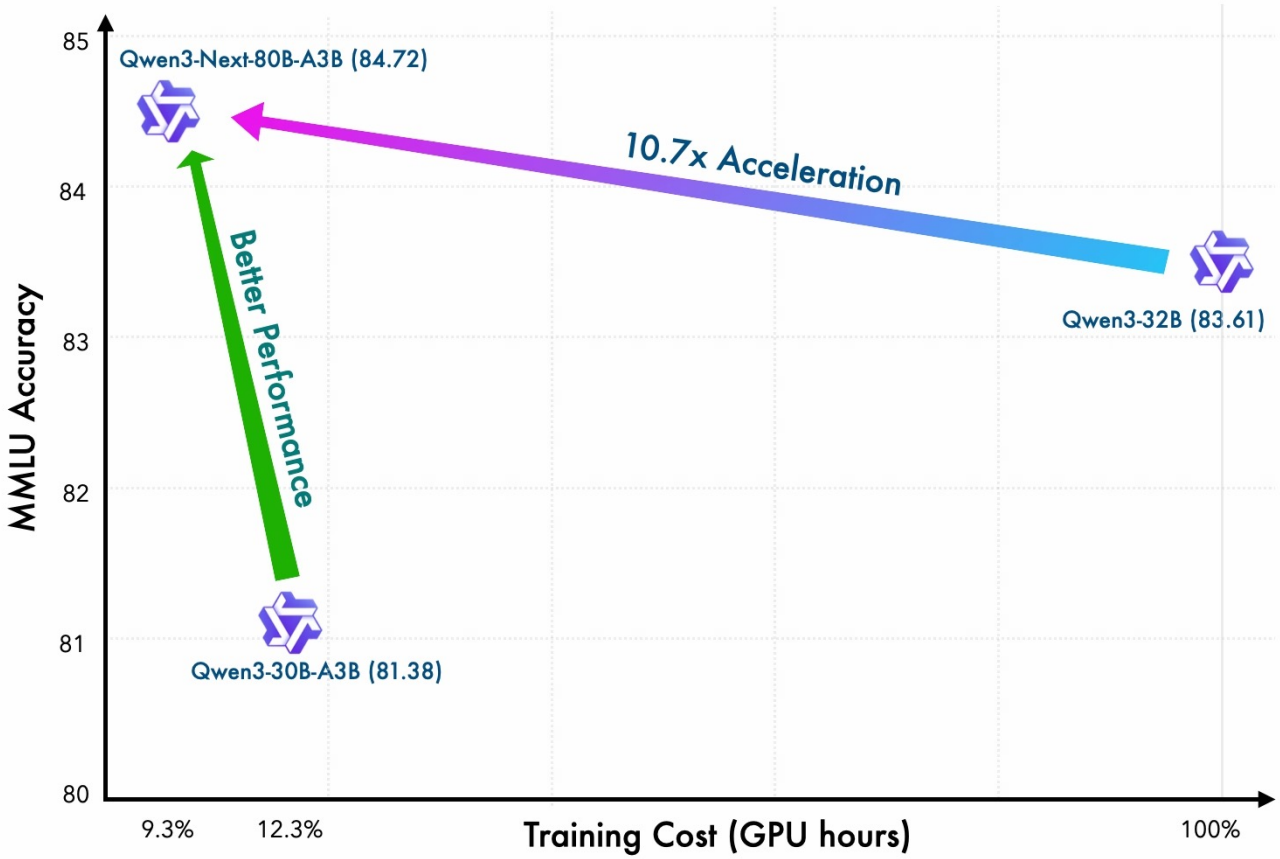
3 Gated DeltaNet : 1 Softmax Attention

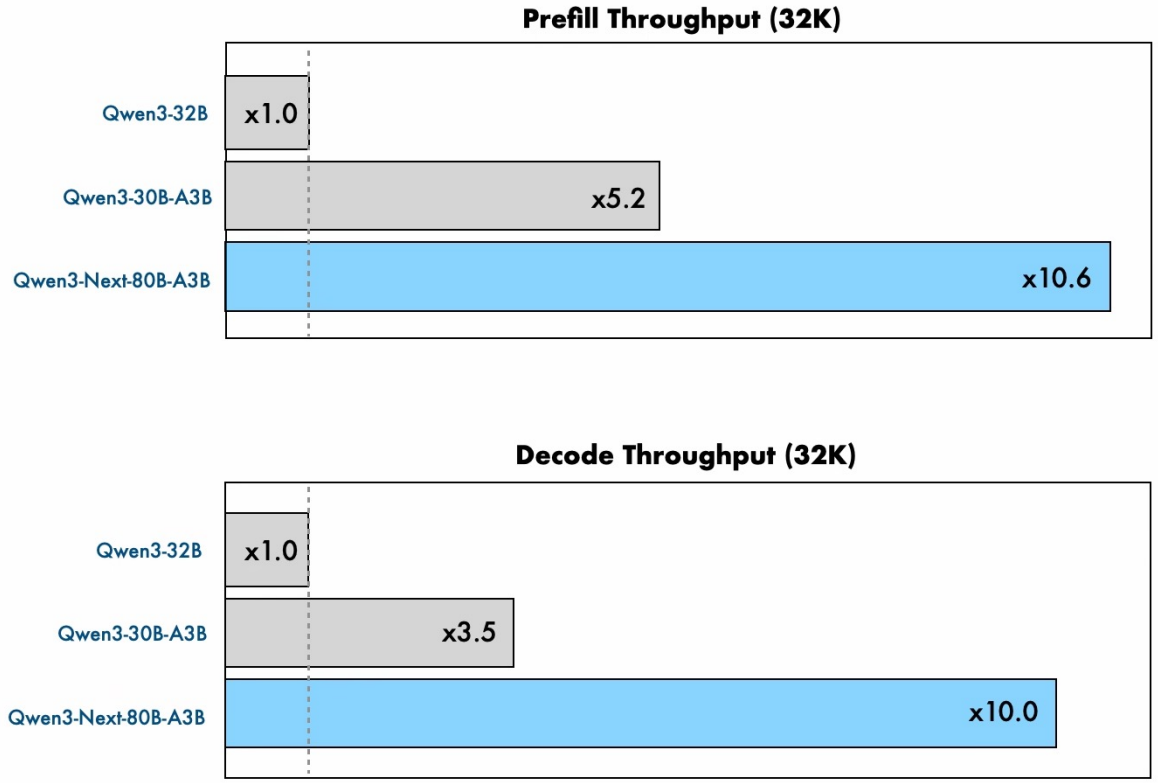


[Qwen3-Next：迈向更极致的训练推理性价比]

Hybrid Attention

Qwen3-Next is extremly efficient





[Qwen3-Next：迈向更极致的训练推理性价比]

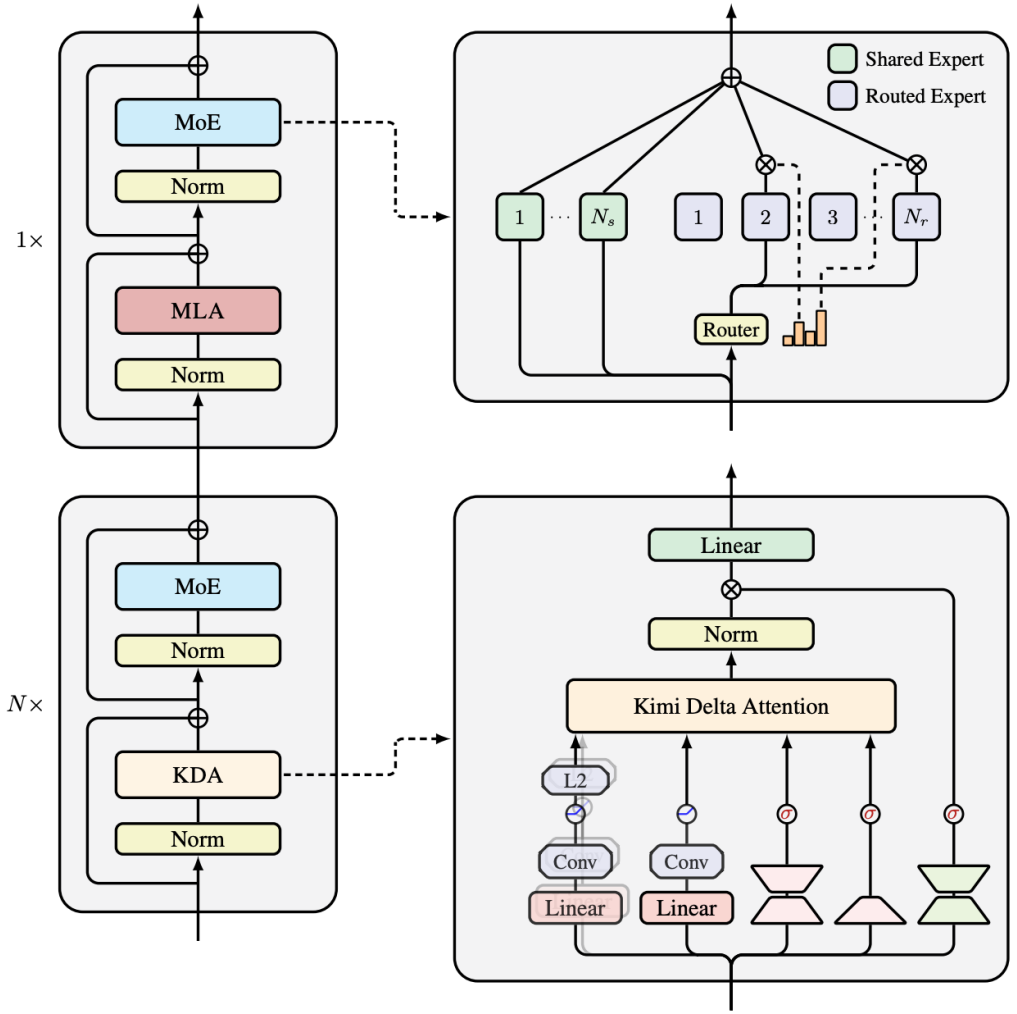
Hybrid Attention

Kimi-Linear

N Kimi DeltaNet : 1 MLA

		Training PPL (↓)	Validation PPL (↓)
Hybrid ratio	3:1	**9.23**	**5.65**
	0:1	9.45	5.77
	1:1	9.29	5.66
	7:1	9.23	5.70
	15:1	9.34	5.82

3:1 is the best



[Kimi Linear: An Expressive, Efficient Attention Architecture]