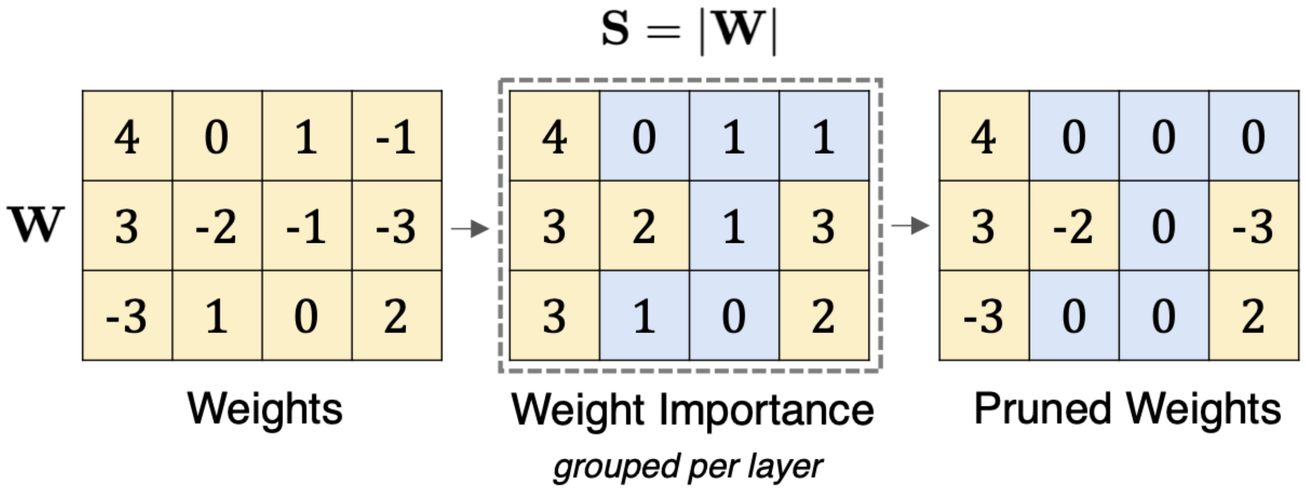
Efficient Inference: Sparsity & Hardware-aware Optimizations

May 19, 2025

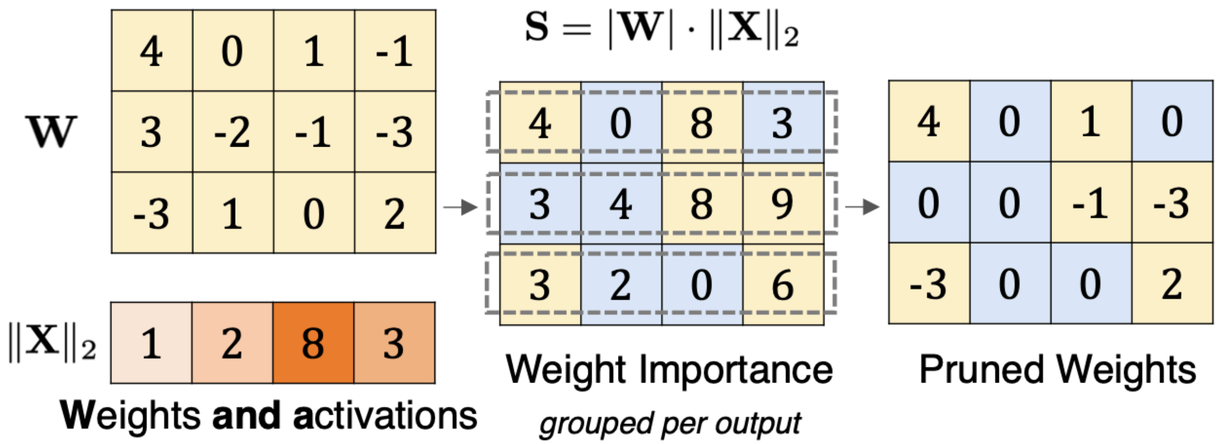
Recap: Parameter Pruning



- **Key idea:** Prune weights that are lower than a certain threshold

[Sun et al.]

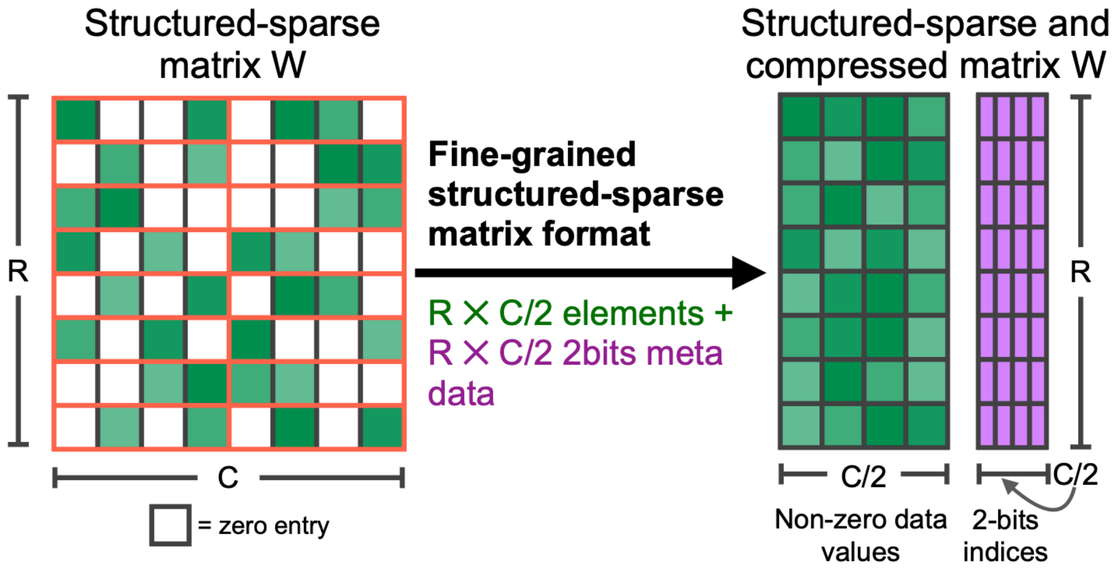
Recap: Activation-aware pruning



- **Key idea:** Prune weights that have a low importance based on the $|\mathbf{W}| \cdot ||\mathbf{X}||_2$ metric
- Similar idea to AWQ and LLM.int8(): Weight importance is determined by the activations too

[Sun et al.]

Recap: Structured pruning



- **Key idea:** Fixed ratio of weights (e.g., 2:4) is pruned between each structure
- Can leverage hardware acceleration

[Mishra et al.]

Up next

Other techniques for enhancing sparsity

Mixture of Experts (MoE)

- **Key idea:** Block level sparsity — only certain weight blocks are active for a given token

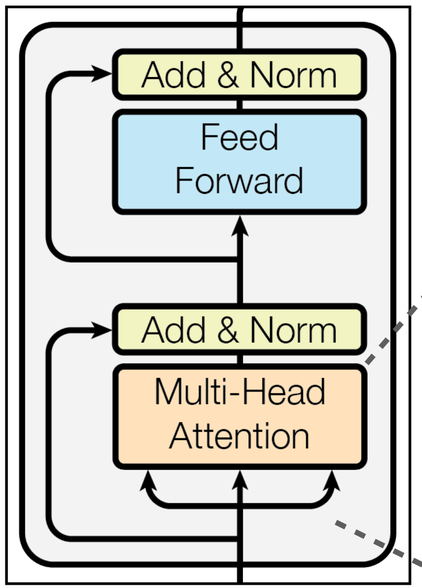
Mixture of Experts (MoE)

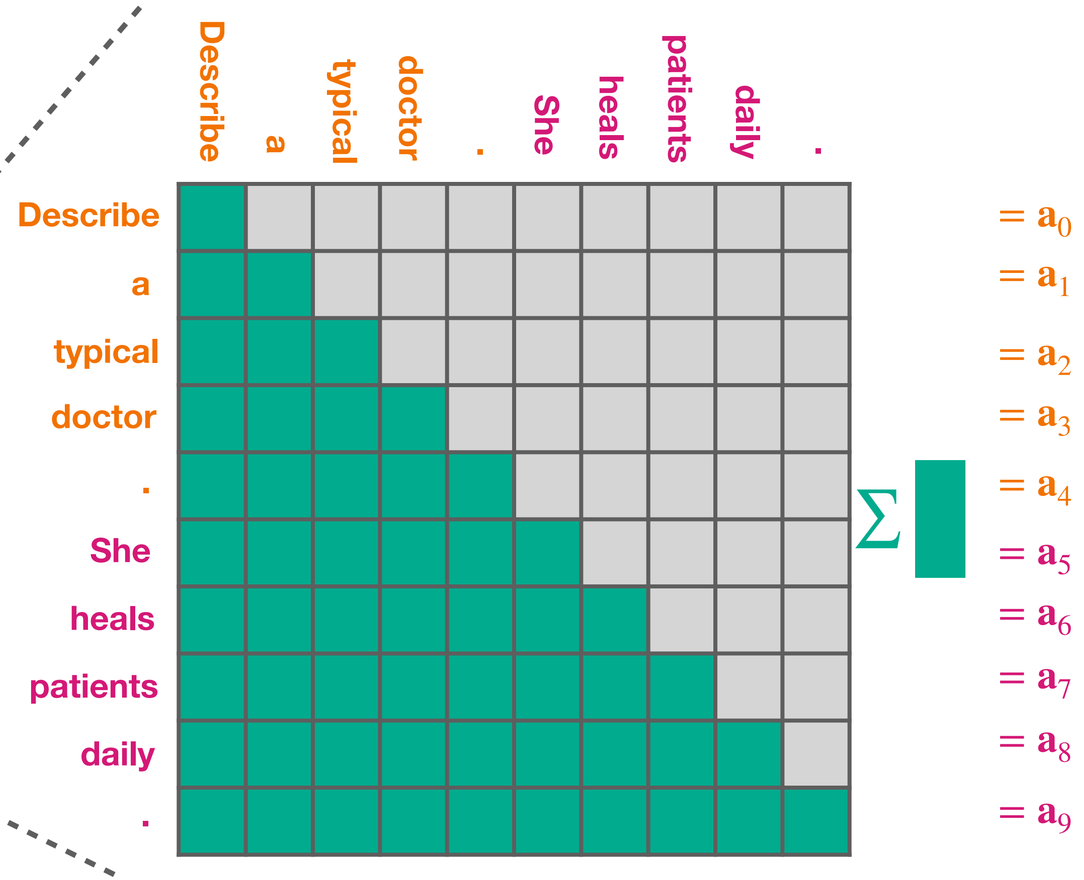
- **Key idea:** Block level sparsity — only certain weight blocks are active for a given token

Let us revisit the Transformer layer

Expensive operation # 1: Attention

When processing *token i*, how much each *token j* should contribute?





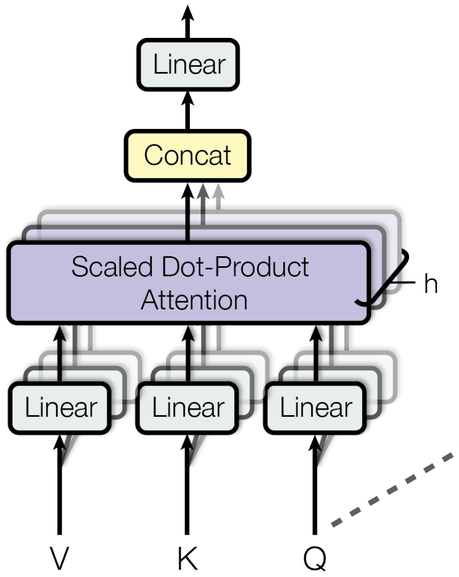
Causal LLMs

Attention of *token i* depends only on *current or previous* tokens

$$\mathbf{a}_i = \sum_{j \le i} \text{attn}(i, j)$$

[Vaswani et al.]

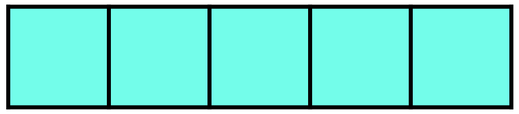
Inside the attention computation



$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

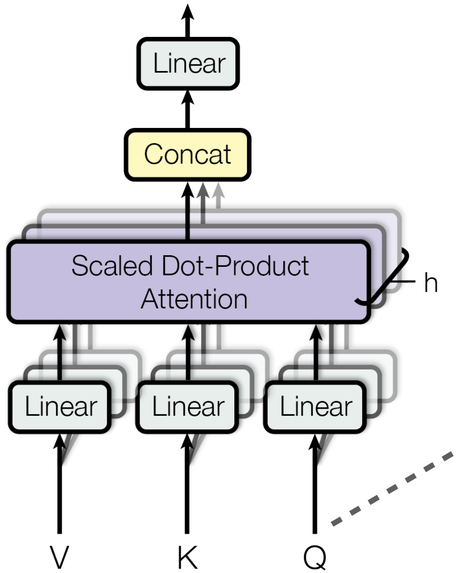
$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



Recall: $\mathbf{X}$ represents a single token

Vaswani et al.

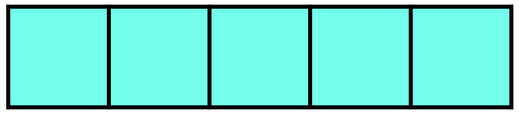
Inside the attention computation



$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

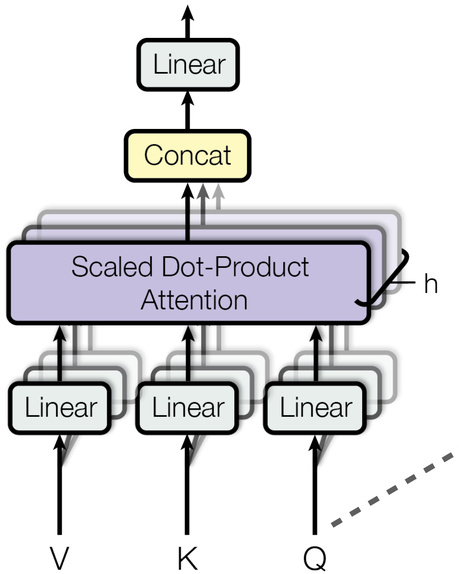
$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



Vaswani et al.

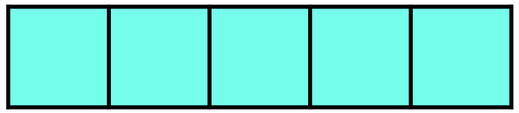
Inside the attention computation



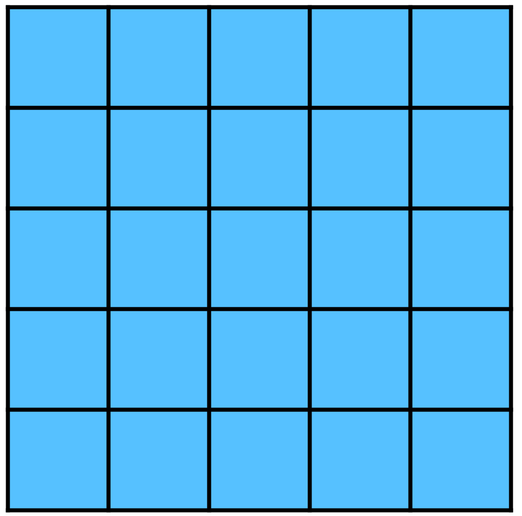
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



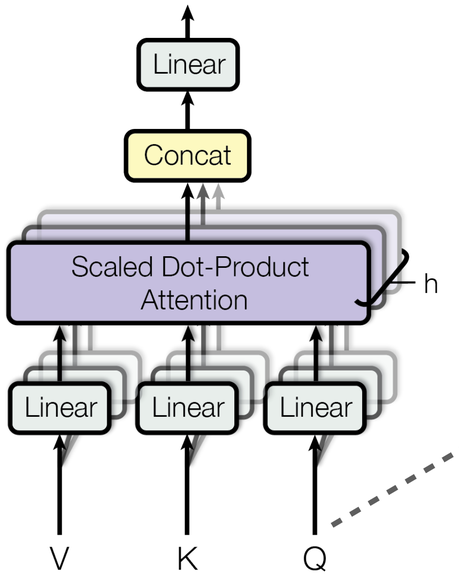
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

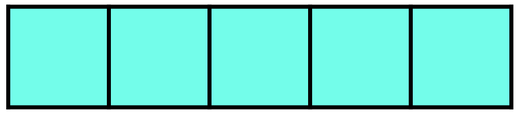
Inside the attention computation



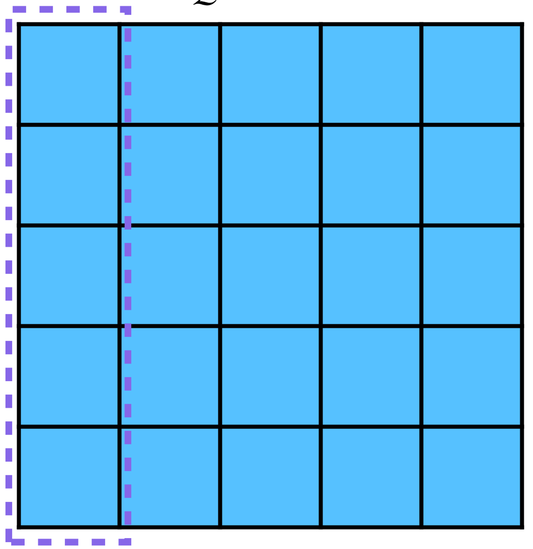
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



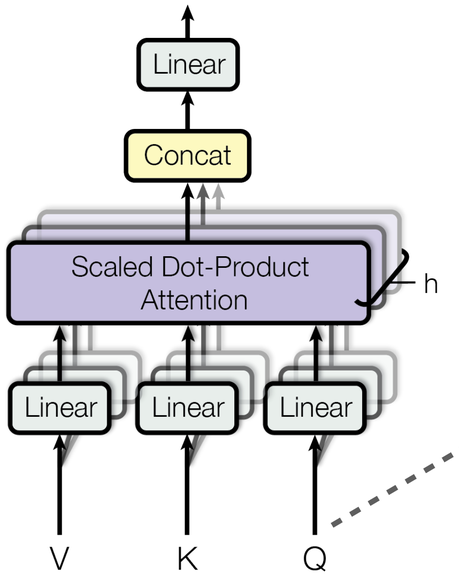
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

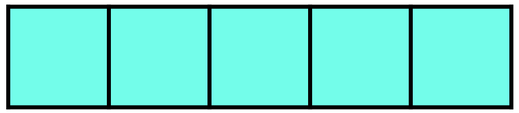
Inside the attention computation



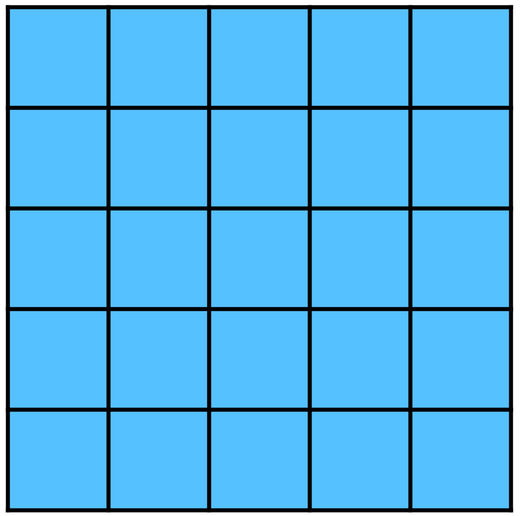
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



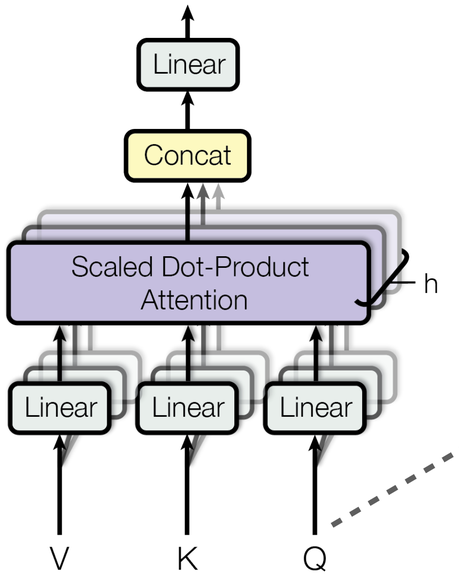
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

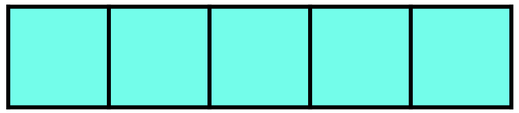
Inside the attention computation



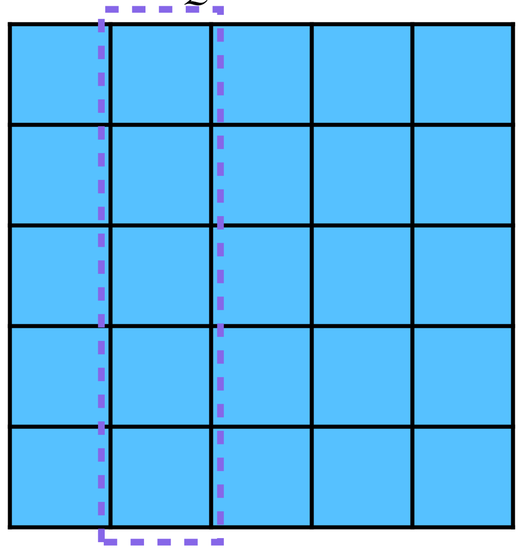
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



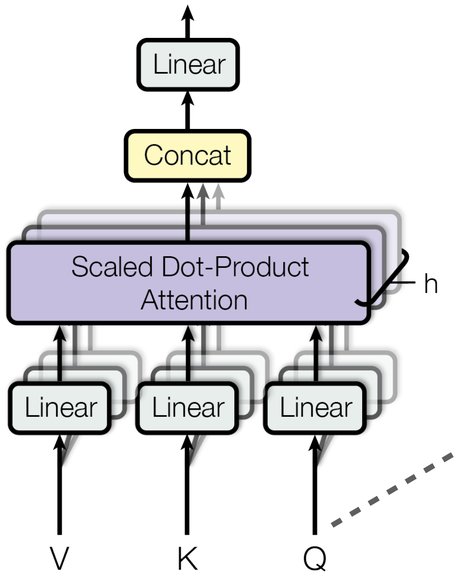
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

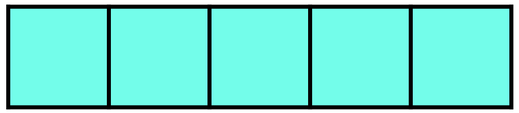
Inside the attention computation



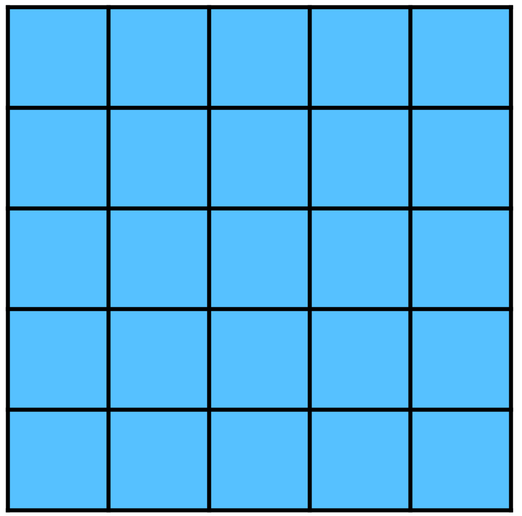
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



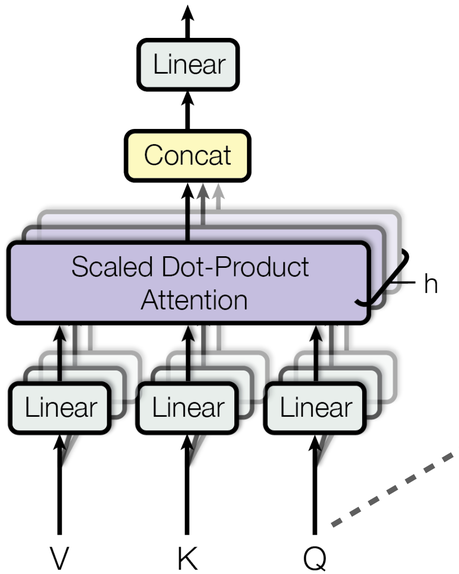
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

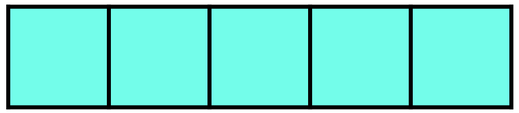
Inside the attention computation



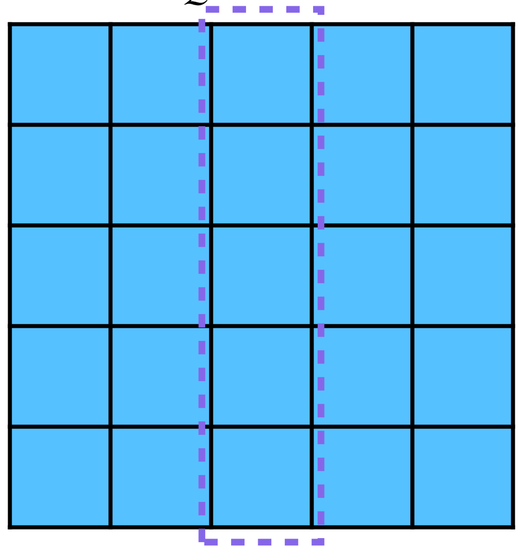
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



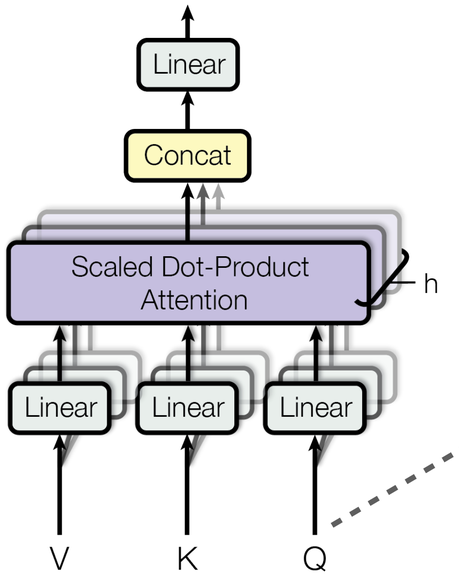
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

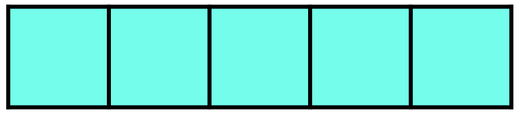
Inside the attention computation



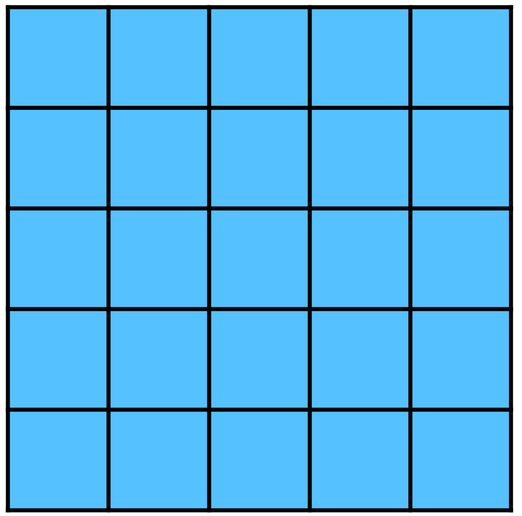
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



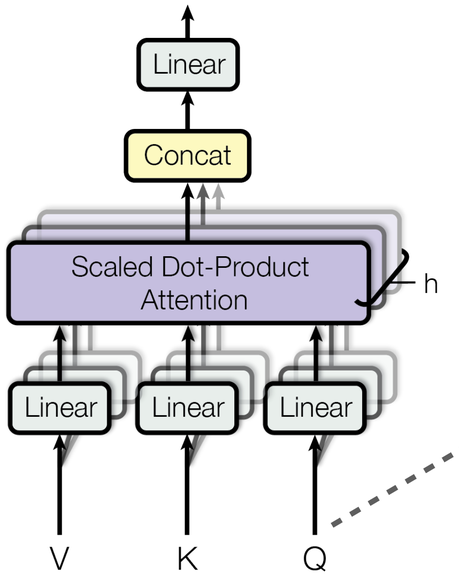
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

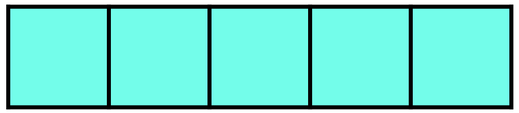
Inside the attention computation



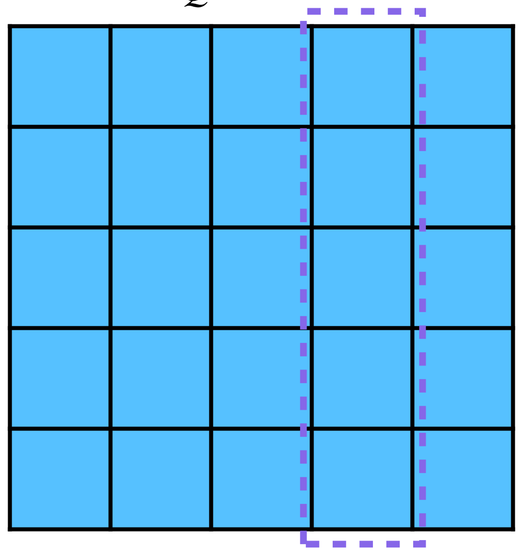
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



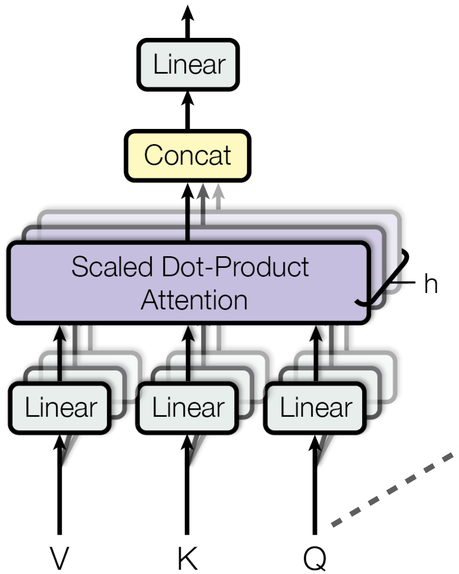
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

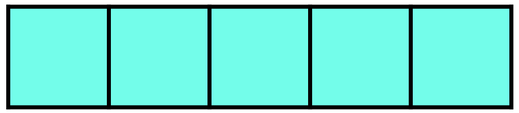
Inside the attention computation



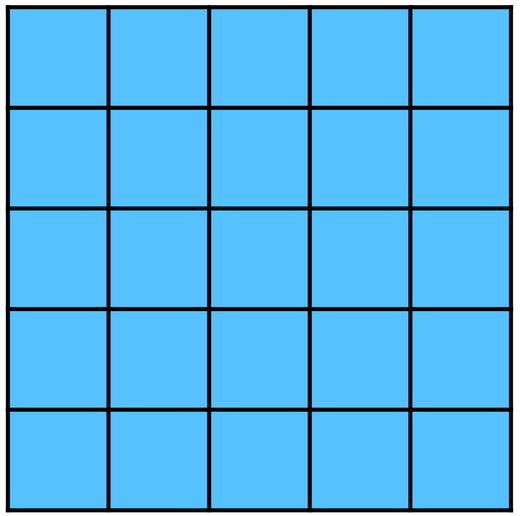
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



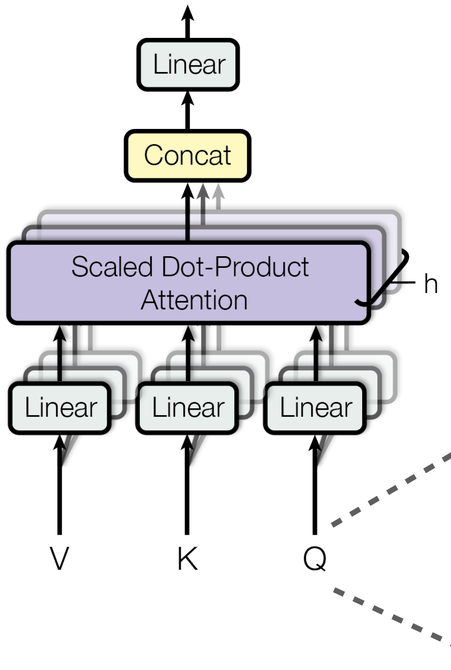
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

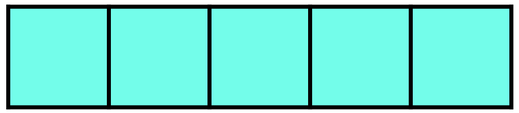
Inside the attention computation



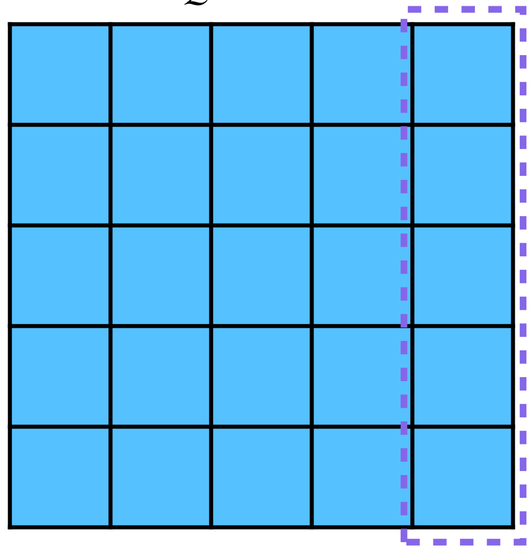
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



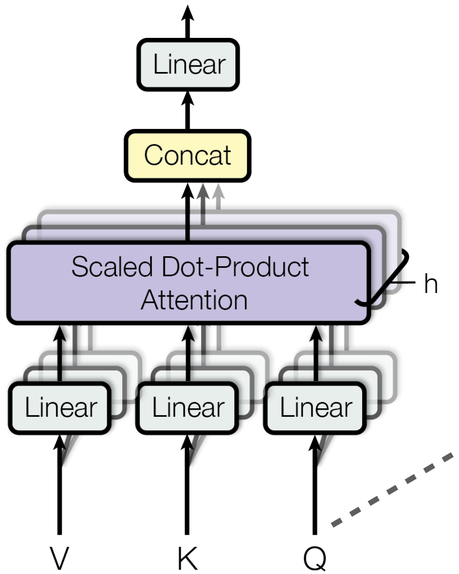
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

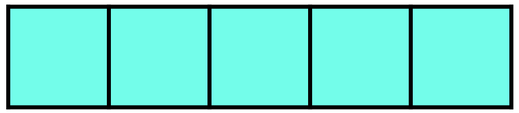
Inside the attention computation



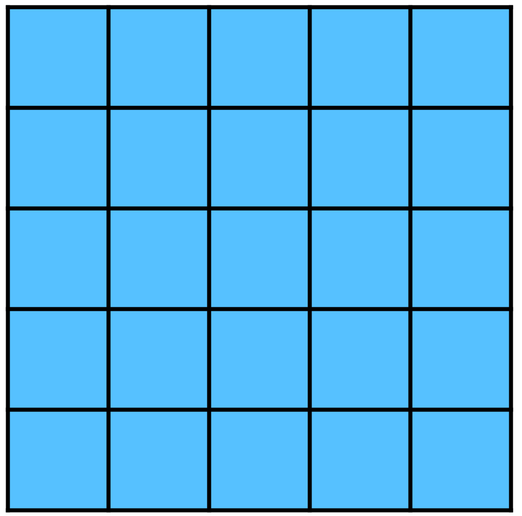
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



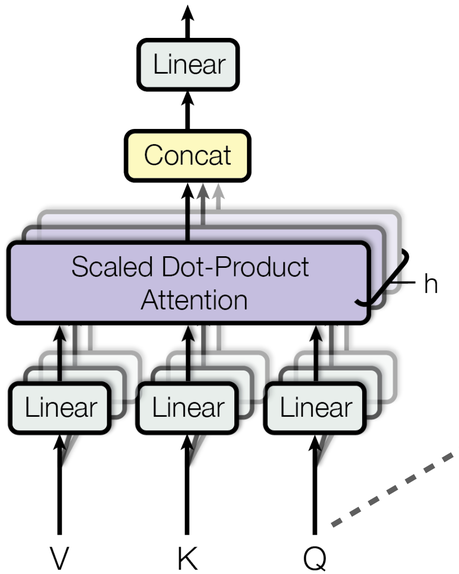
$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

Vaswani et al.

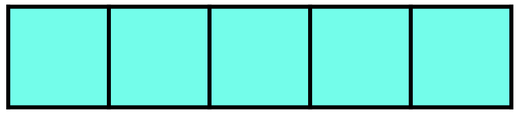
Inside the attention computation



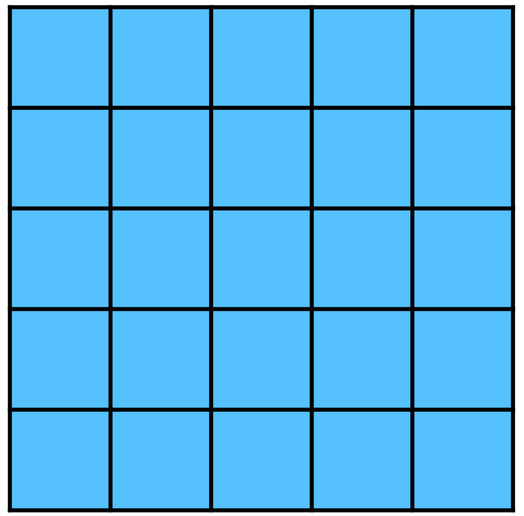
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{W}_Q \in \mathbb{R}^{D \times D}$$

$$\mathbf{X} \in \mathbb{R}^{B \times D}$$



$$\mathbf{W}_Q \in \mathbf{R}^{D \times D}$$



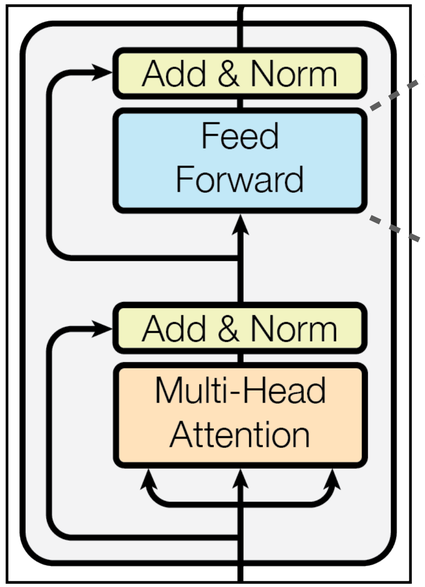
Multiply $\mathbf{X}$ with each column of $\mathbf{W}_Q$

D^2 operations per token

Vaswani et al.

Expensive operation # 2: The fully connected net

Project to 4D dimensions, then project back to D



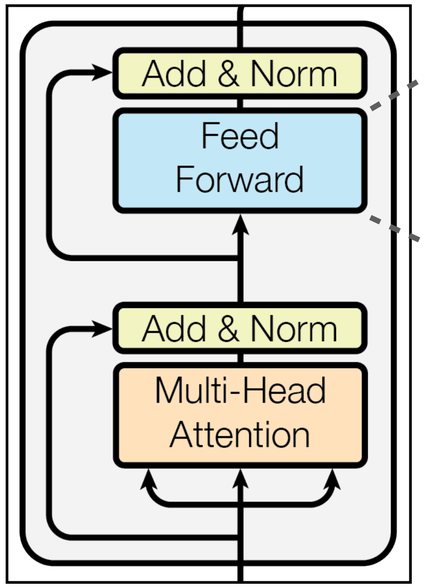
$$\max(0, \mathbf{X}\mathbf{W}_u)\mathbf{W}_d$$

* Skipping the nitty gritty like biases

[Vaswani et al.]

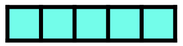
Expensive operation # 2: The fully connected net

Project to 4D dimensions, then project back to D

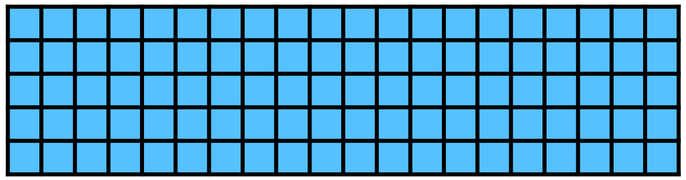


$$\max(0, \mathbf{X}\mathbf{W}_u)\mathbf{W}_d$$

$\mathbf{X} \in \mathbb{R}^{B \times D}$



$\mathbf{W}_u \in \mathbf{R}^{D \times 4D}$

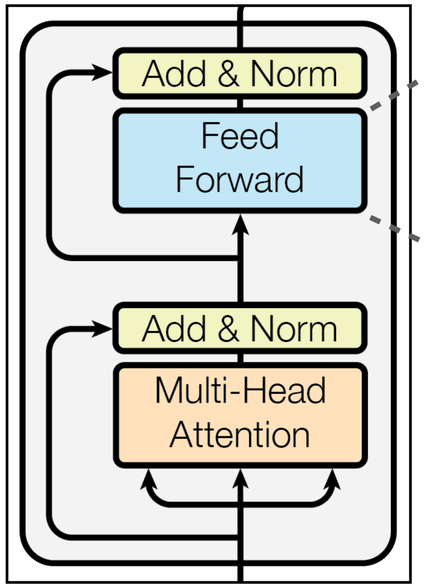


* Skipping the nitty gritty like biases

[Vaswani et al.]

Expensive operation # 2: The fully connected net

Project to 4D dimensions, then project back to D

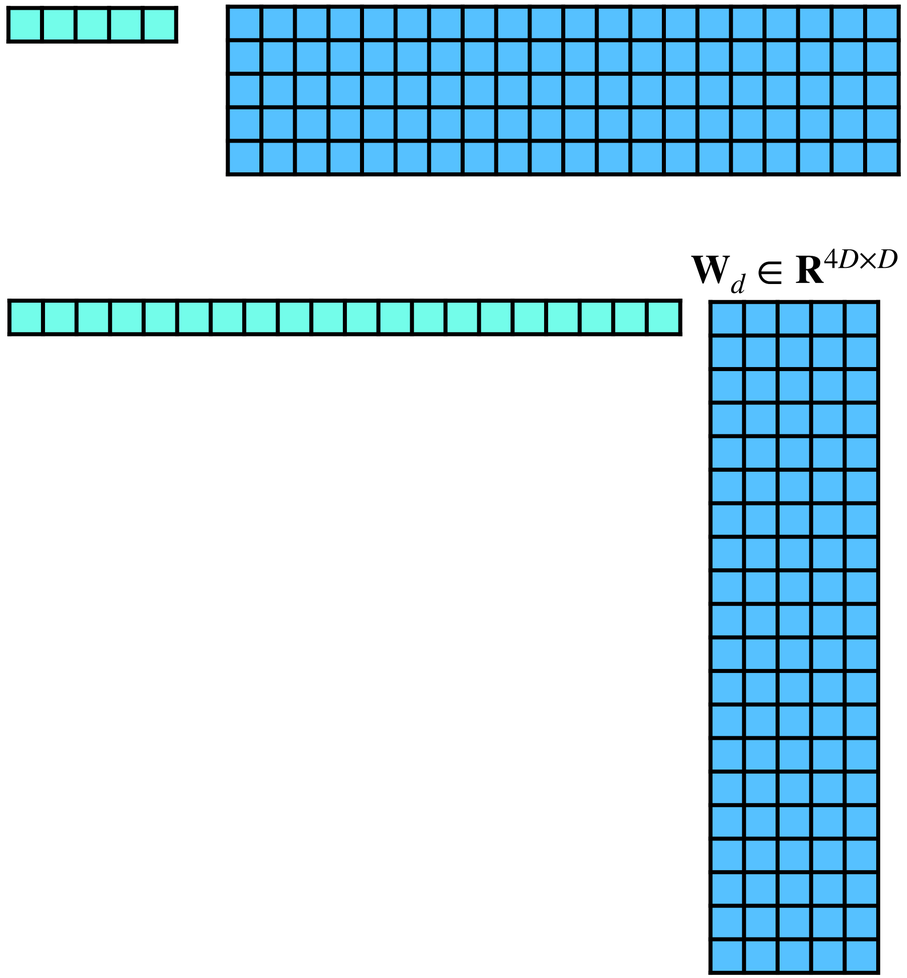


$$\max(0, \mathbf{X}\mathbf{W}_u)\mathbf{W}_d$$

$\mathbf{X} \in \mathbb{R}^{B \times D}$

$\mathbf{W}_u \in \mathbf{R}^{D \times 4D}$

$\mathbf{W}_d \in \mathbf{R}^{4D \times D}$

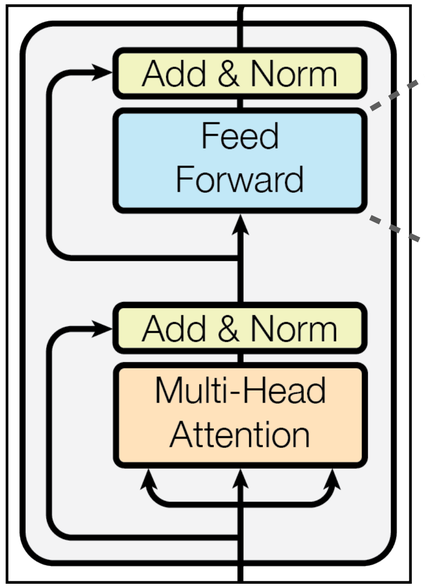


* Skipping the nitty gritty like biases

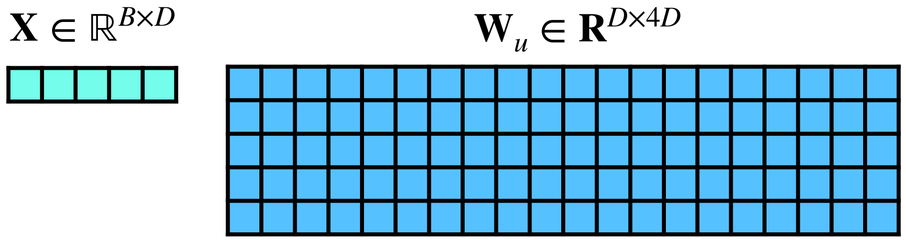
[Vaswani et al.]

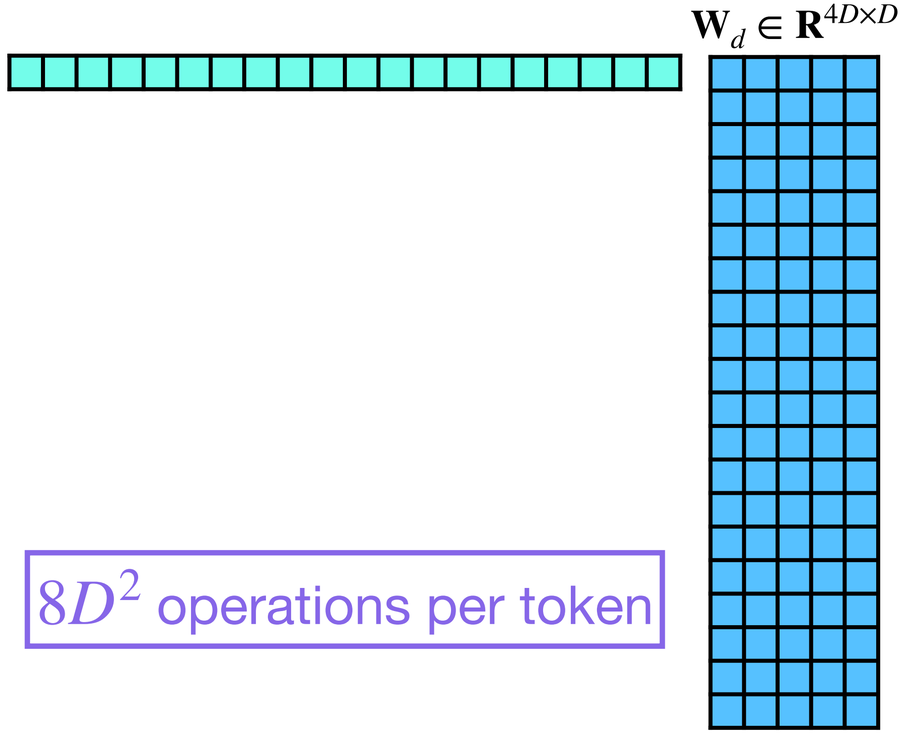
Expensive operation # 2: The fully connected net

Project to 4D dimensions, then project back to D



$$\max(0, \mathbf{X}\mathbf{W}_u)\mathbf{W}_d$$





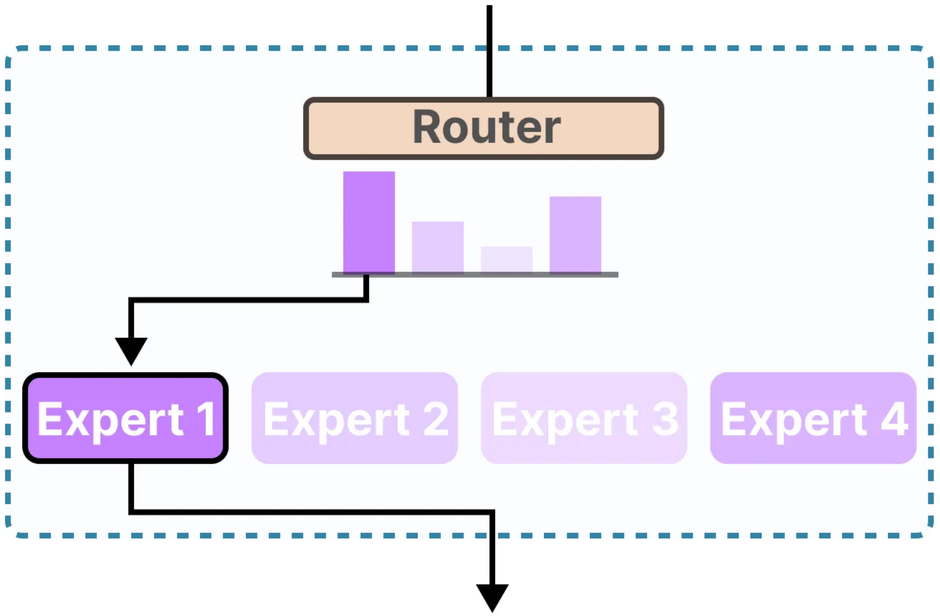
$8D^2$ operations per token

* Skipping the nitty gritty like biases

[Vaswani et al.]

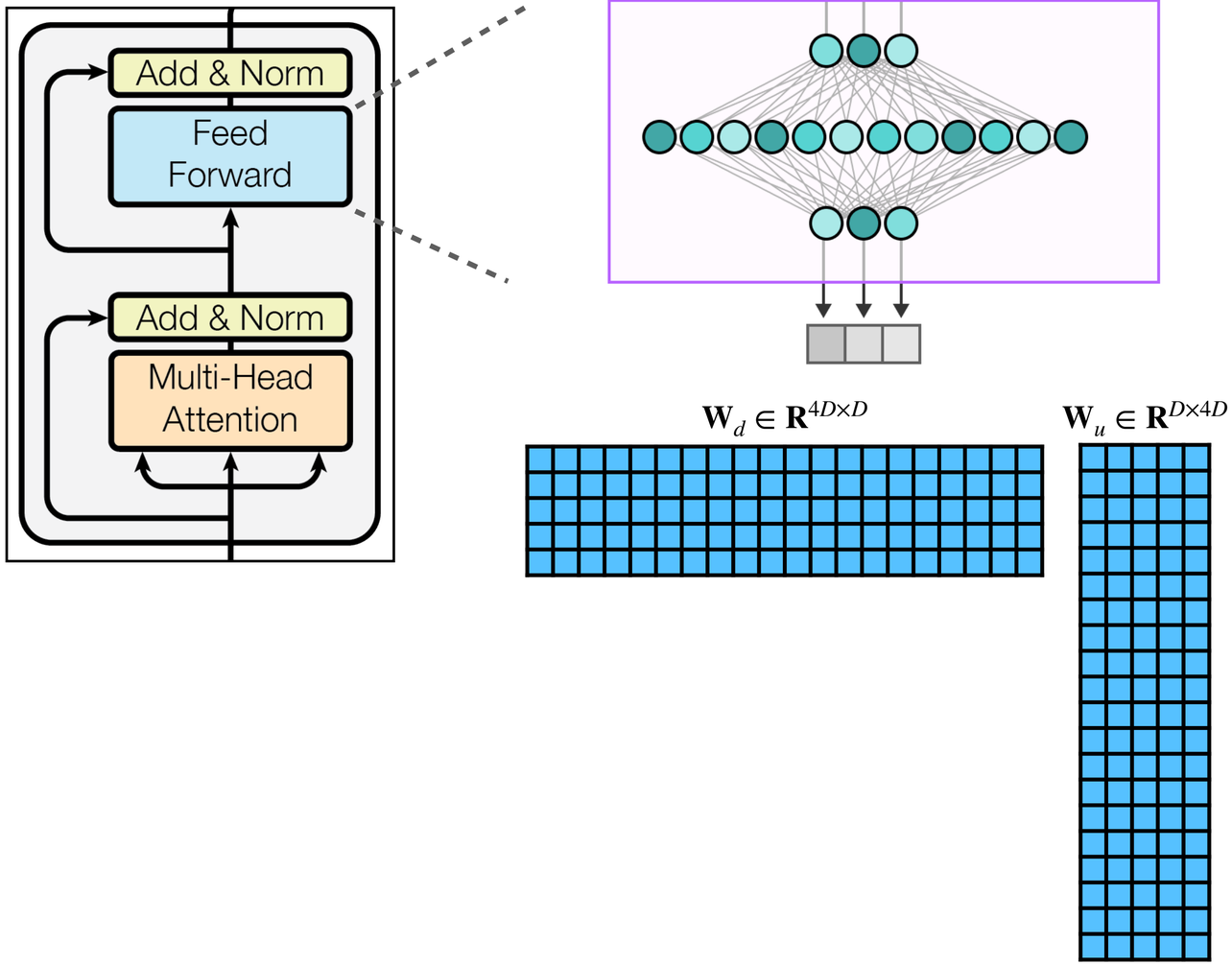
Mixture of experts

- **Key idea:** Each token processed by a small portion of the model
- Parameters split into disjoint blocks
- Each input passed through a single (or few) blocks
- Router: Decodes which block should process the token
  - Itself a fully connected layer



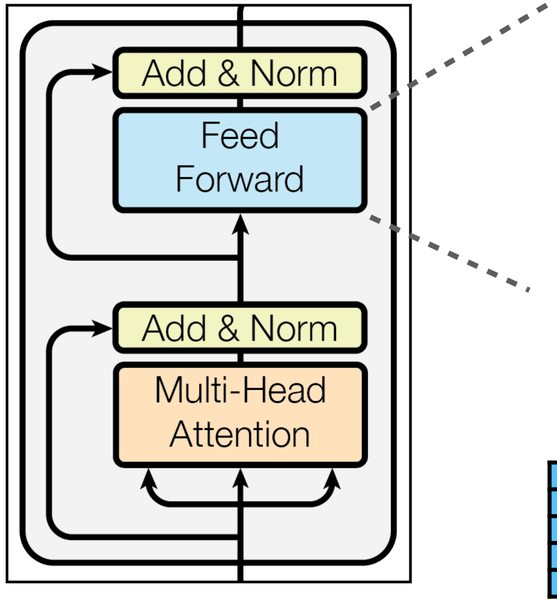
[Grootendorst]

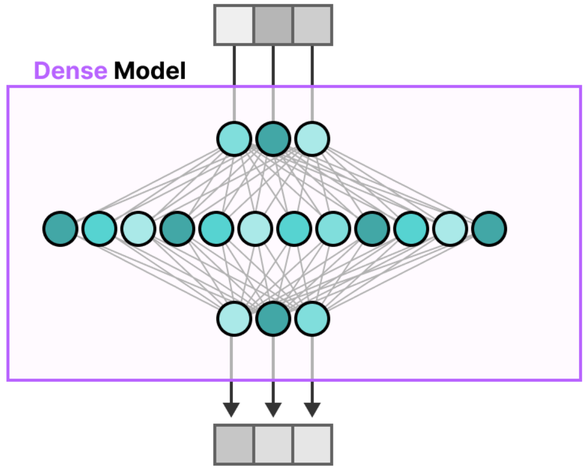
MoE in the transformer layer

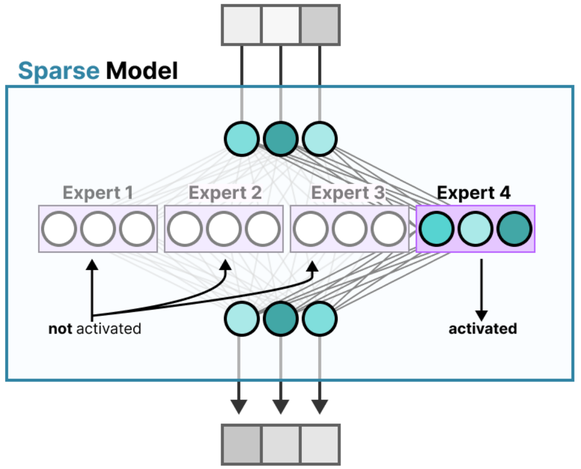


[Grootendorst]
[Vaswani et al.]

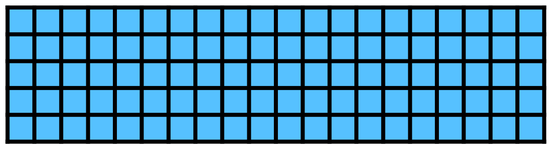
MoE in the transformer layer



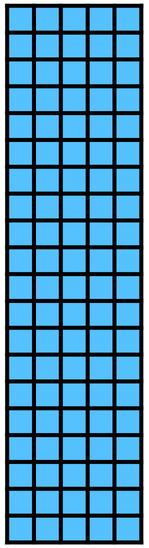




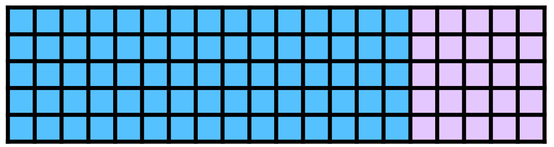
$\mathbf{W}_d \in \mathbf{R}^{4D \times D}$



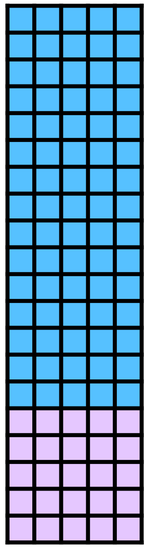
$\mathbf{W}_u \in \mathbf{R}^{D \times 4D}$



$\mathbf{W}_d \in \mathbf{R}^{4D \times D}$

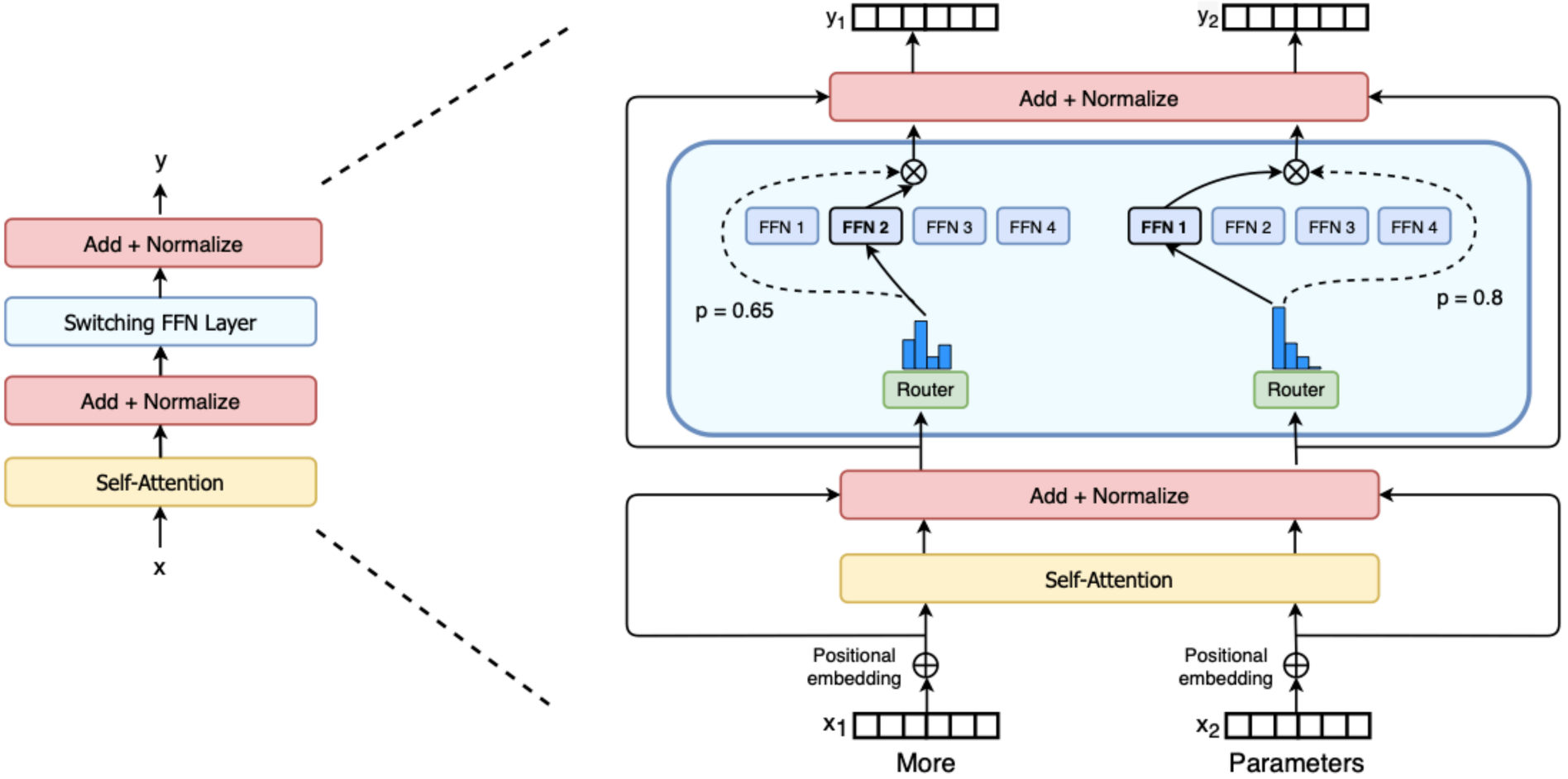


$\mathbf{W}_u \in \mathbf{R}^{D \times 4D}$



[Grootendorst]
[Vaswani et al.]

Expert switching happens for every token



[Jiang et al.]

Why MoE?

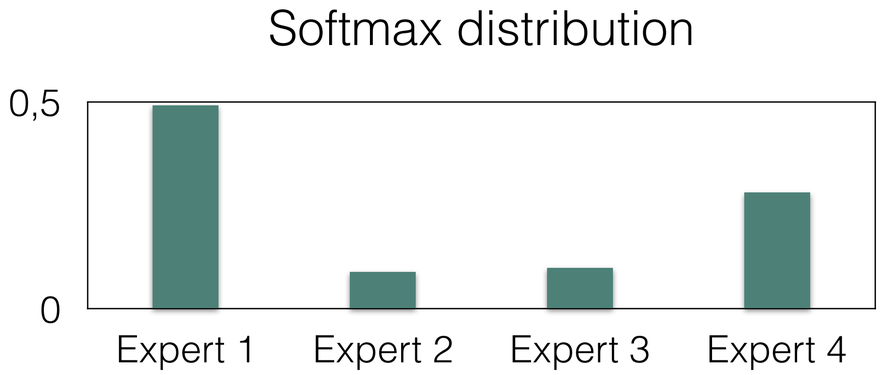
- Intuition
  - Very high overall capacity (combined parameters of all experts)
  - Fast inference: Each input token sees one or few, but not all, experts (2 in the case of Mixtral 8x7B)

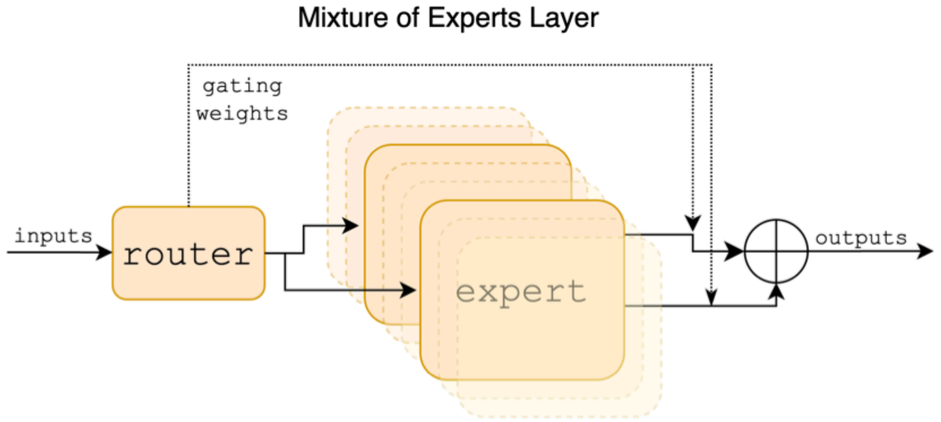
- No (direct) memory saving: All experts need to be loaded in on the GPU

- Better parallelization: Each token could go to a different expert

How to route

- Router: Decodes which block should process the token
  - Router($\mathbf{X}$) = $[p_1, p_2, \ldots, p_M]$
  - Itself a fully connected model
- Pick the K highest ranking experts
- Mixtral: Select top-2 logits, weigh the output by resulting softmax probability
  - Expert Logits: [1, **2**, 0.5, **3**]
  - Top-2: [**2**, 3]
  - Softmax: [0.27, 0.73]
  - Output = 0.27 $\text{expert}_2(x)$ + 0.73 $\text{expert}_4(x)$





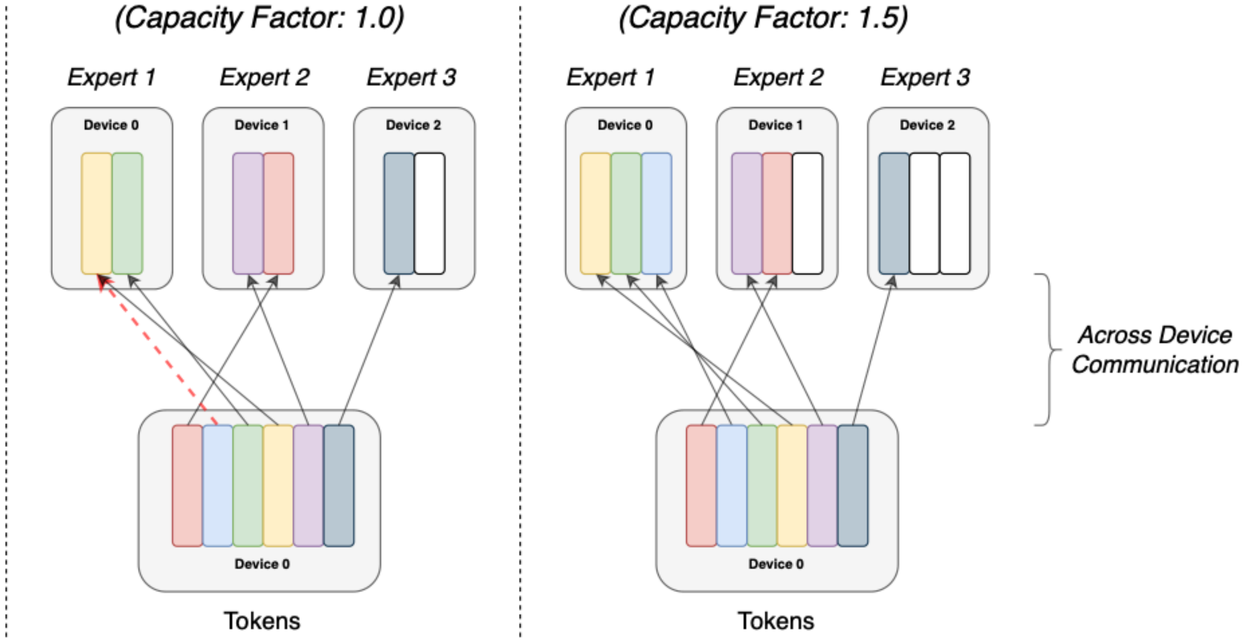
[Jiang et al.]

Load balancing

$$\text{expert capacity} = \left(\frac{\text{tokens per batch}}{\text{number of experts}}\right) \times \text{capacity factor}.$$

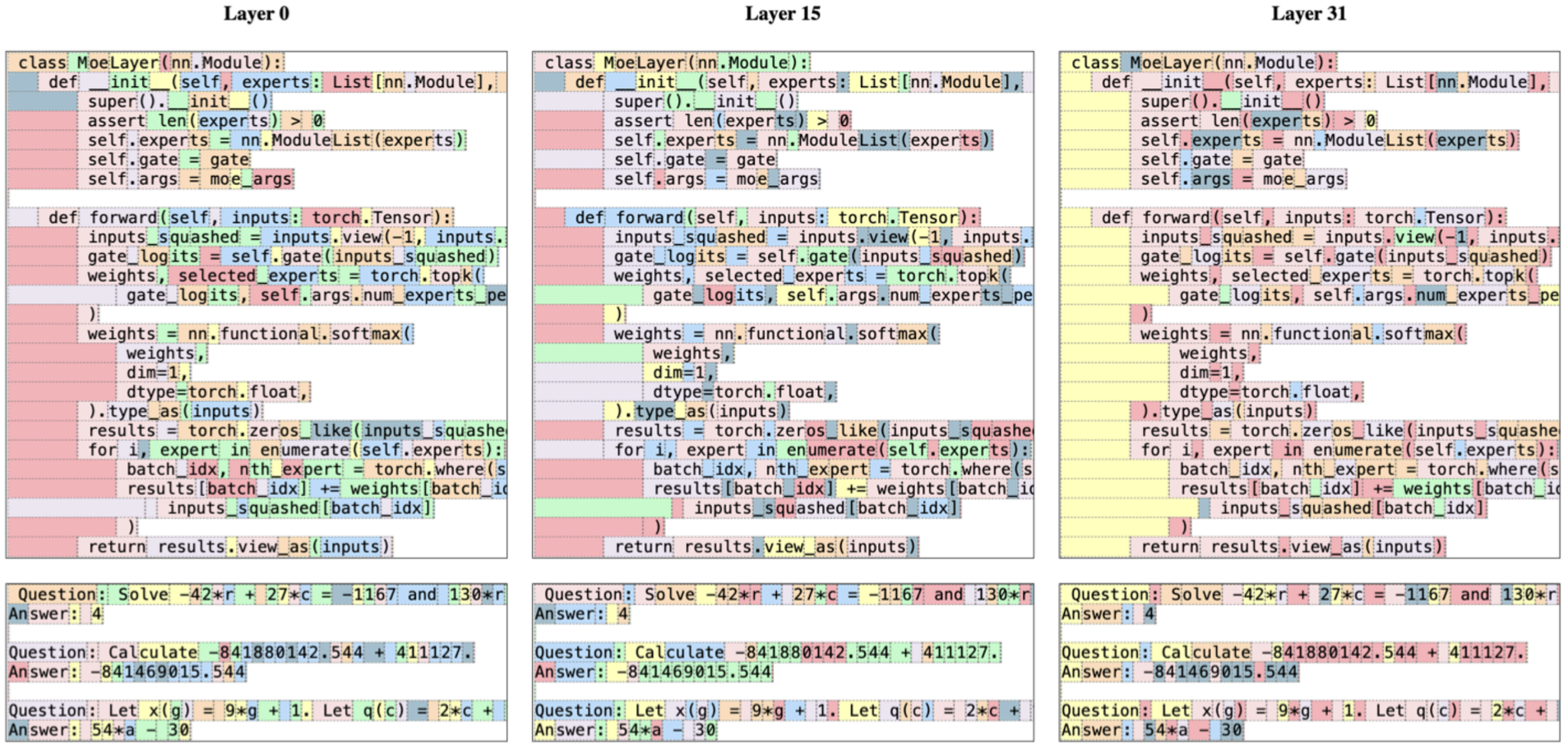
Terminology

- **Experts:** Split across devices, each having their own unique parameters. Perform standard feed-forward computation.
- **Expert Capacity:** Batch size of each expert. Calculated as
- (tokens_per_batch / num_experts) * capacity_factor
- **Capacity Factor:** Used when calculating expert capacity. Expert capacity allows more buffer to help mitigate token overflow during routing.



- Low capacity: Tokens will get dropped (the red arrow in the left example)
- High capacity: Wasted compute (right example: experts 2 and 3 are underutilized)

[Fedus et al.]

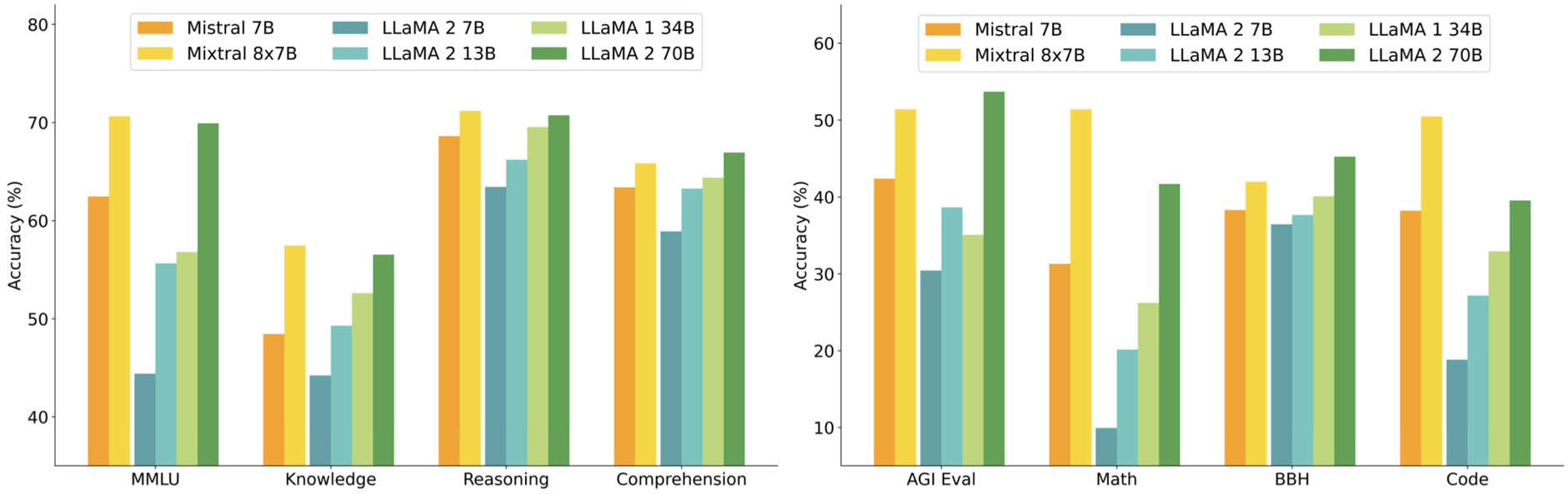


- Different experts responsible for different patterns
  - `self` keyword goes to a single expert even though it consists of multiple tokens
  - Indentations often get assigned to the same expert

[Jiang et al.]

Performance gains

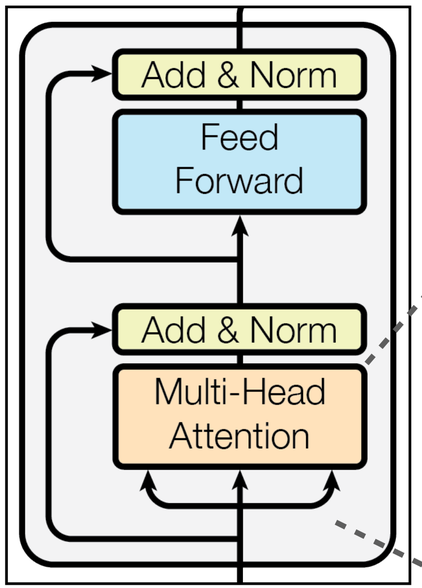
- Mixtral 8x7B
  - 46.7B total parameters
  - Only two experts active at a time

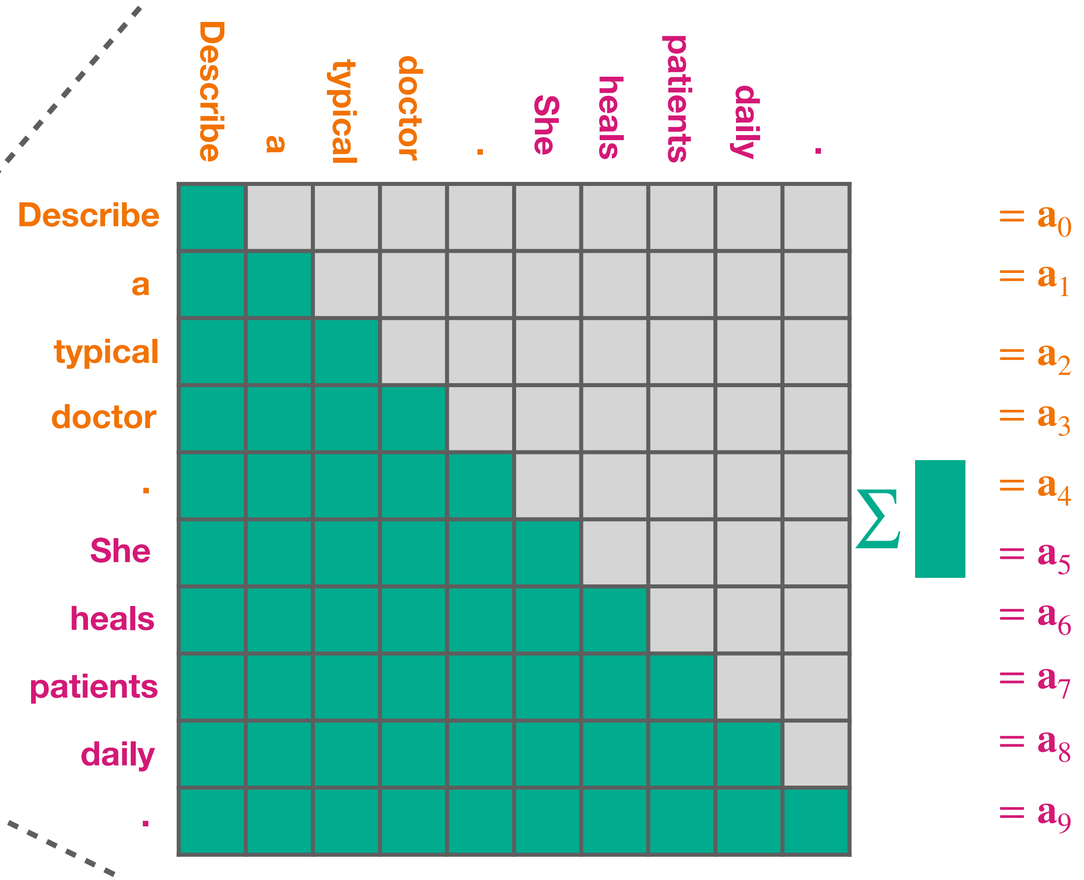


[Jiang et al.]

Expensive operation # 1: Attention

When processing *token i*, how much each *token j* should contribute?





Causal LLMs

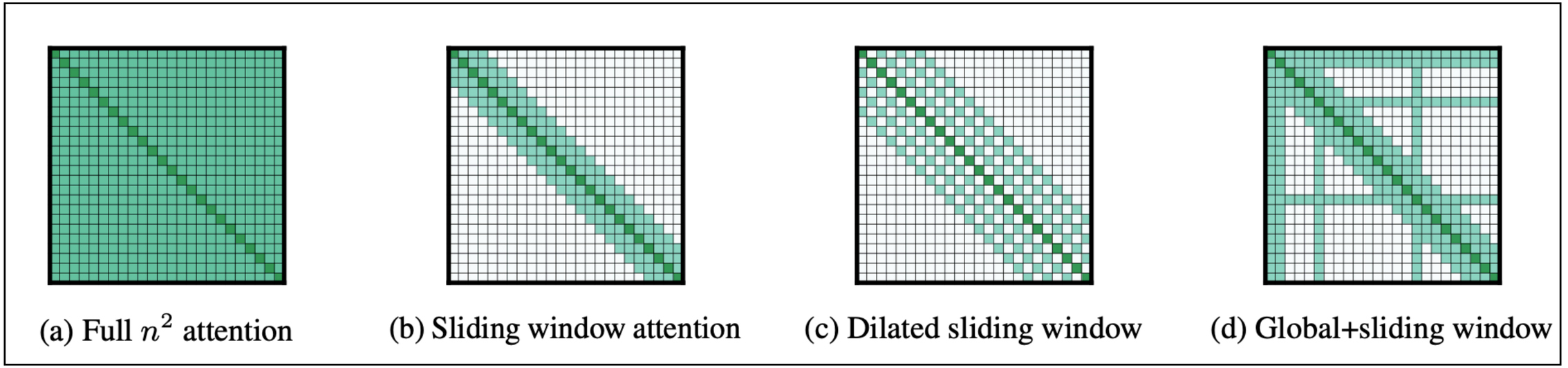
Attention of *token i* depends only on *current or previous* tokens

$$\mathbf{a}_i = \sum_{j \le i} \text{attn}(i, j)$$

[Vaswani et al.]

Sparse attention with Longformer

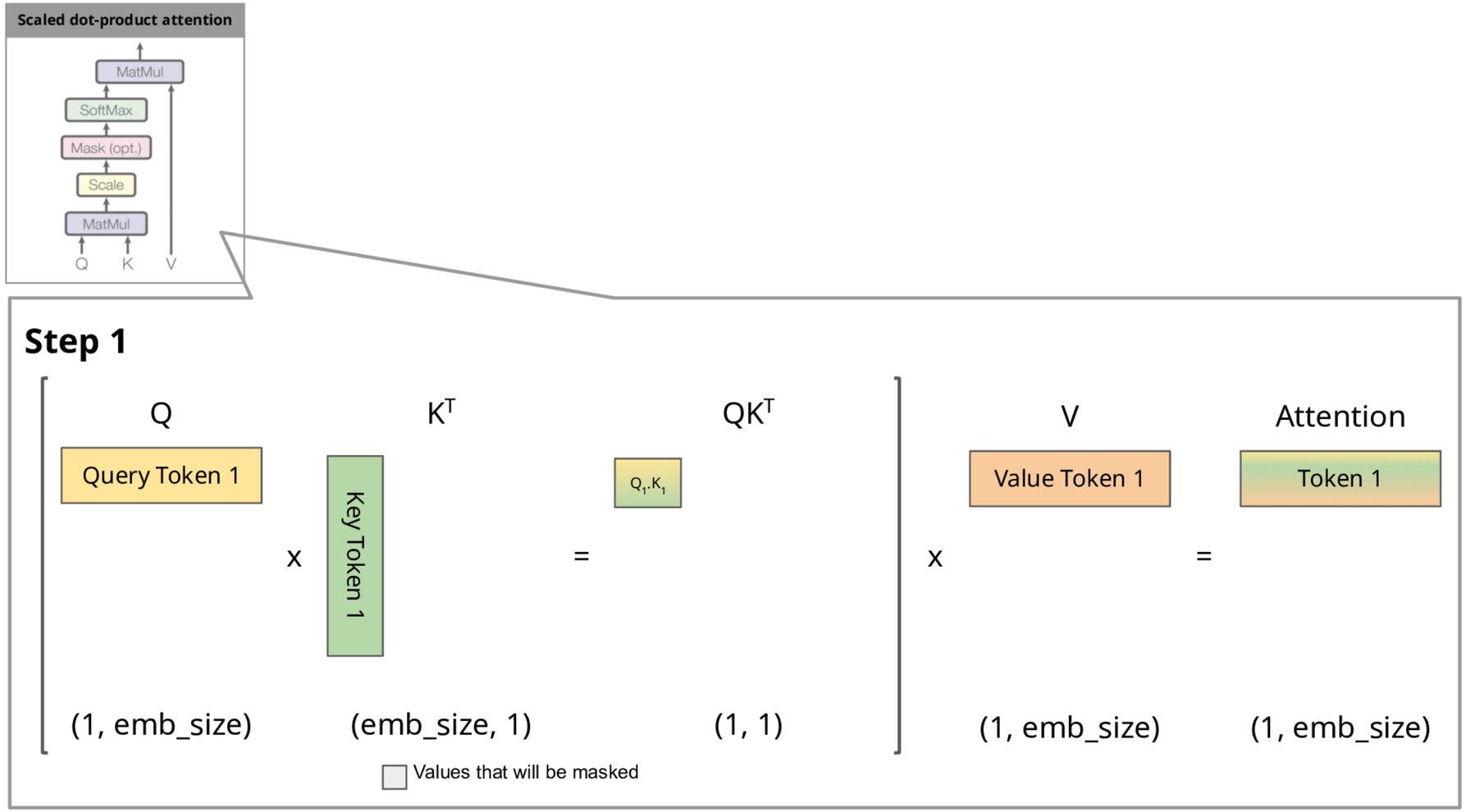
Key idea: Reduce the $O(N^2)$ attention cost



- Dilated sliding window: Increase the receptive field
- Global + Sliding window: Sliding window but few preselected tokens like [CLS] attend to all tokens

[Beltagy et al.]

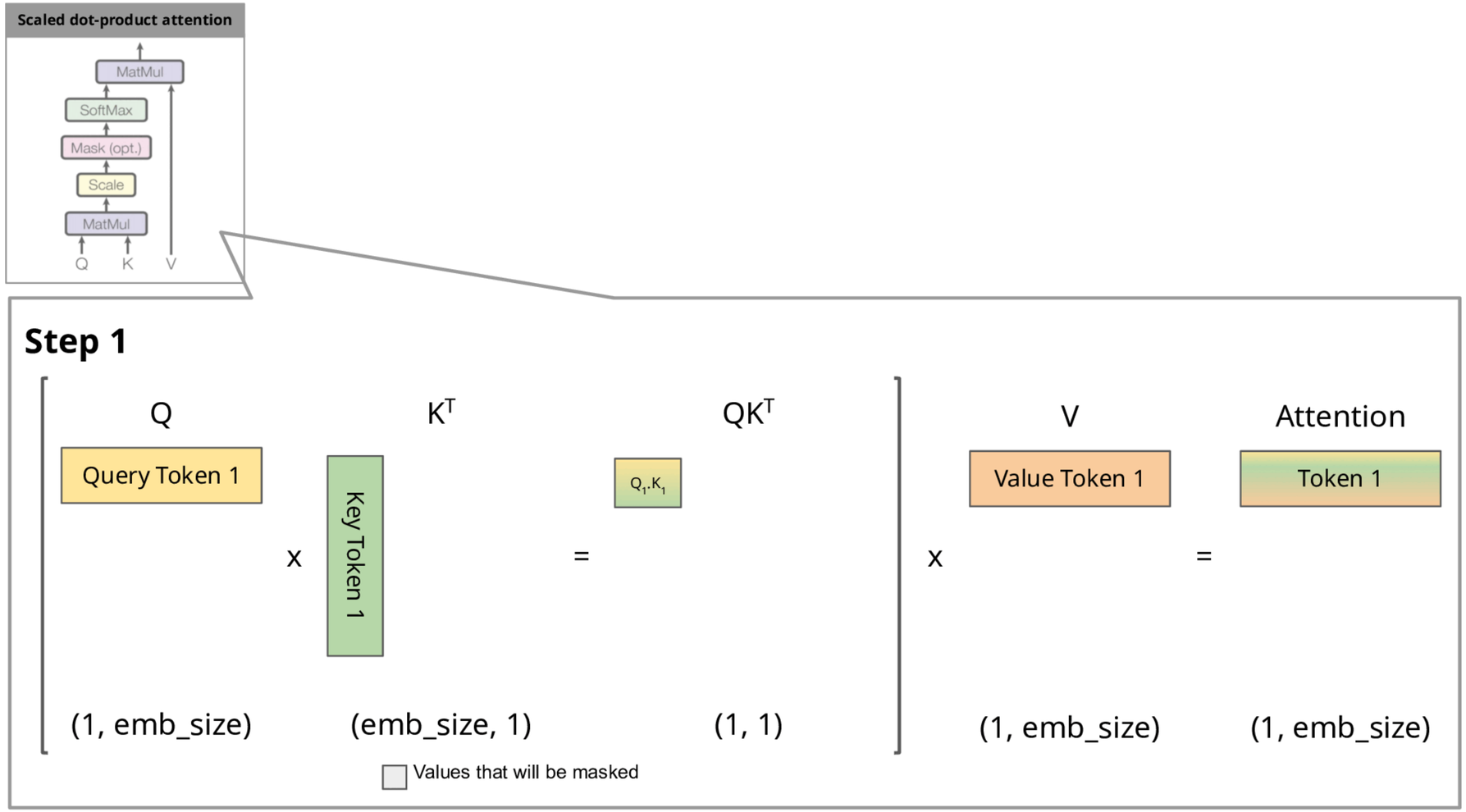
KV-caching: Optimizing attention computation



Zoom-in! (simplified without Scale and Softmax)

[Lages]

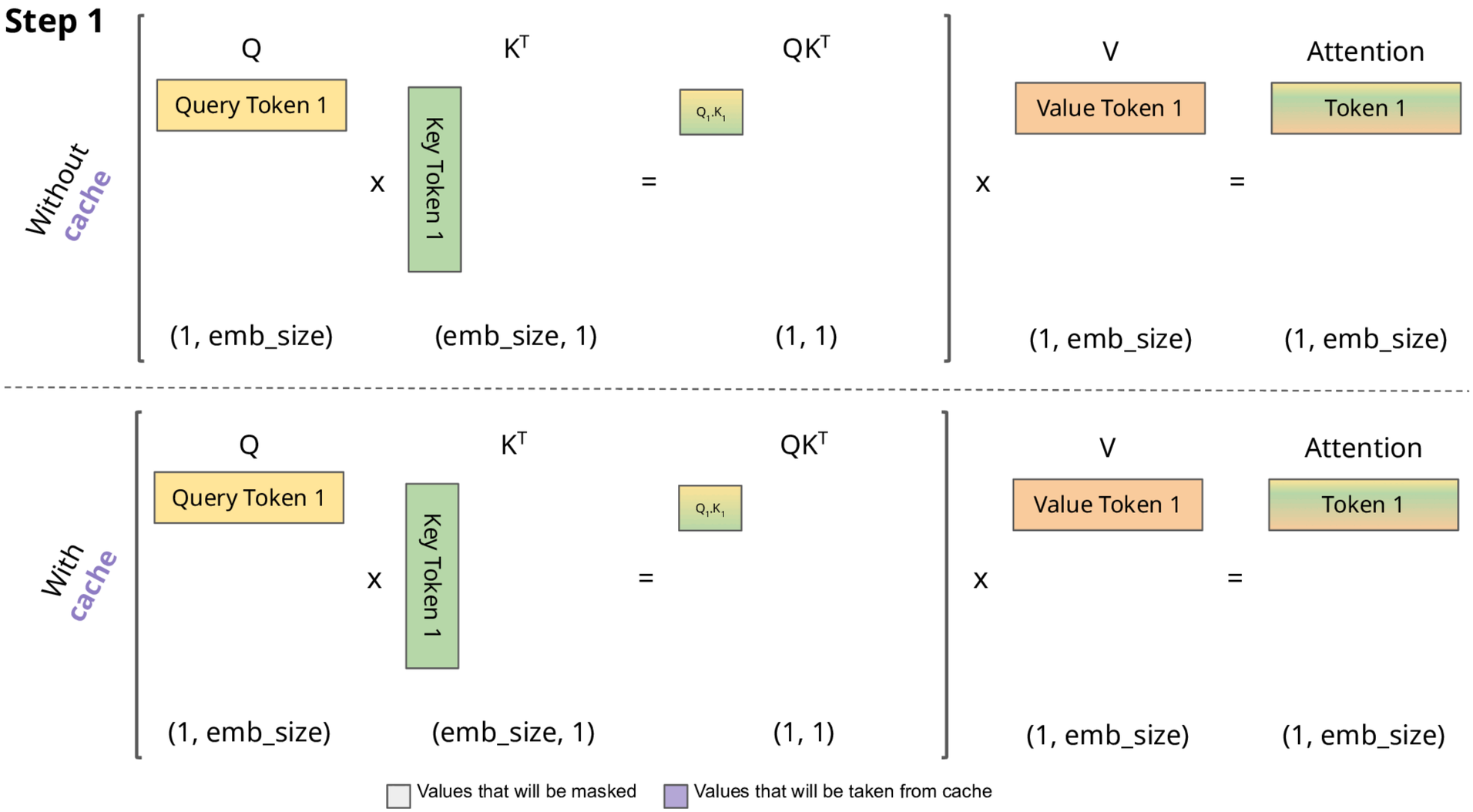
KV-caching: Optimizing attention computation



Zoom-in! (simplified without Scale and Softmax)

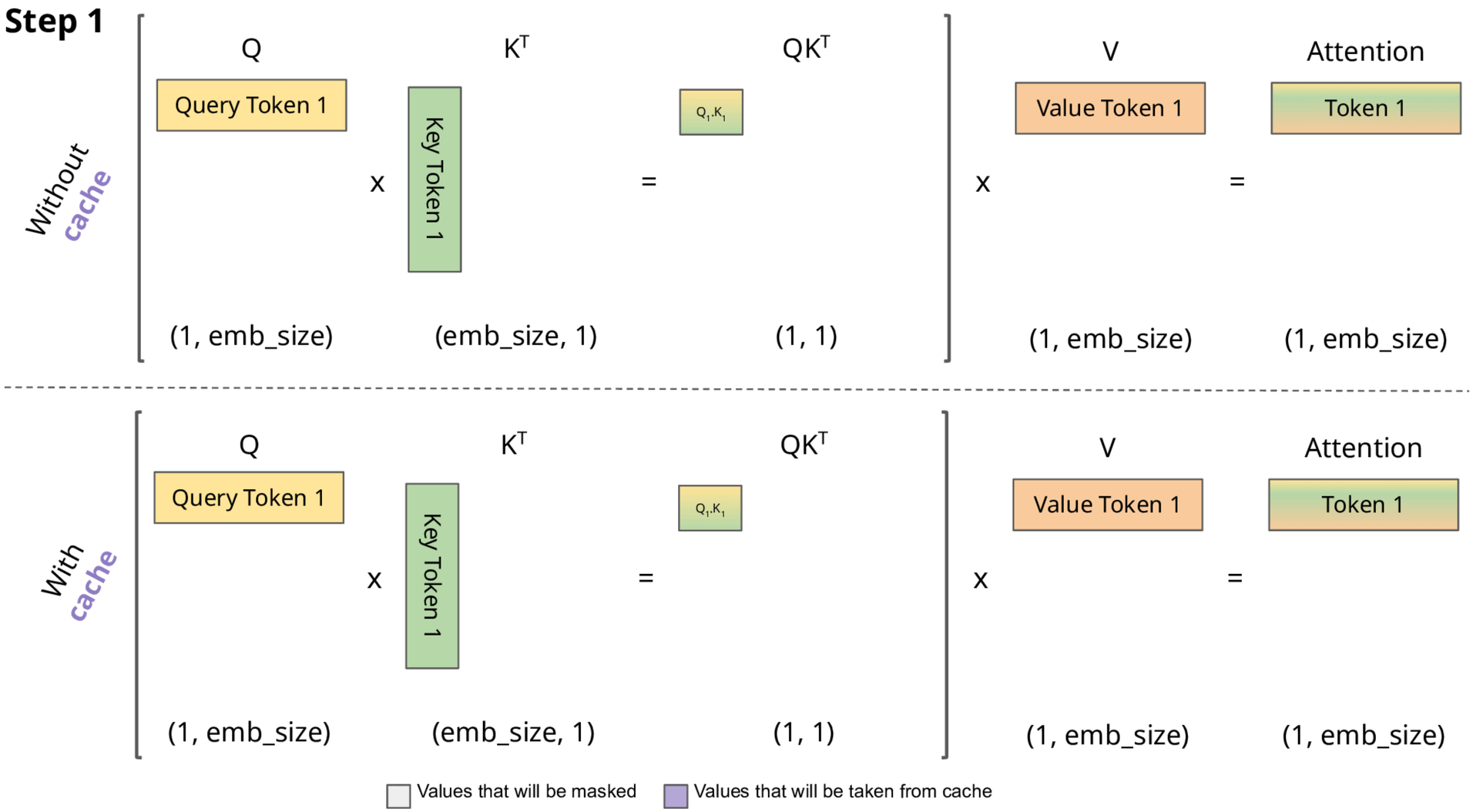
[Lages]

KV-caching: Optimizing attention computation



[Lages]

KV-caching: Optimizing attention computation

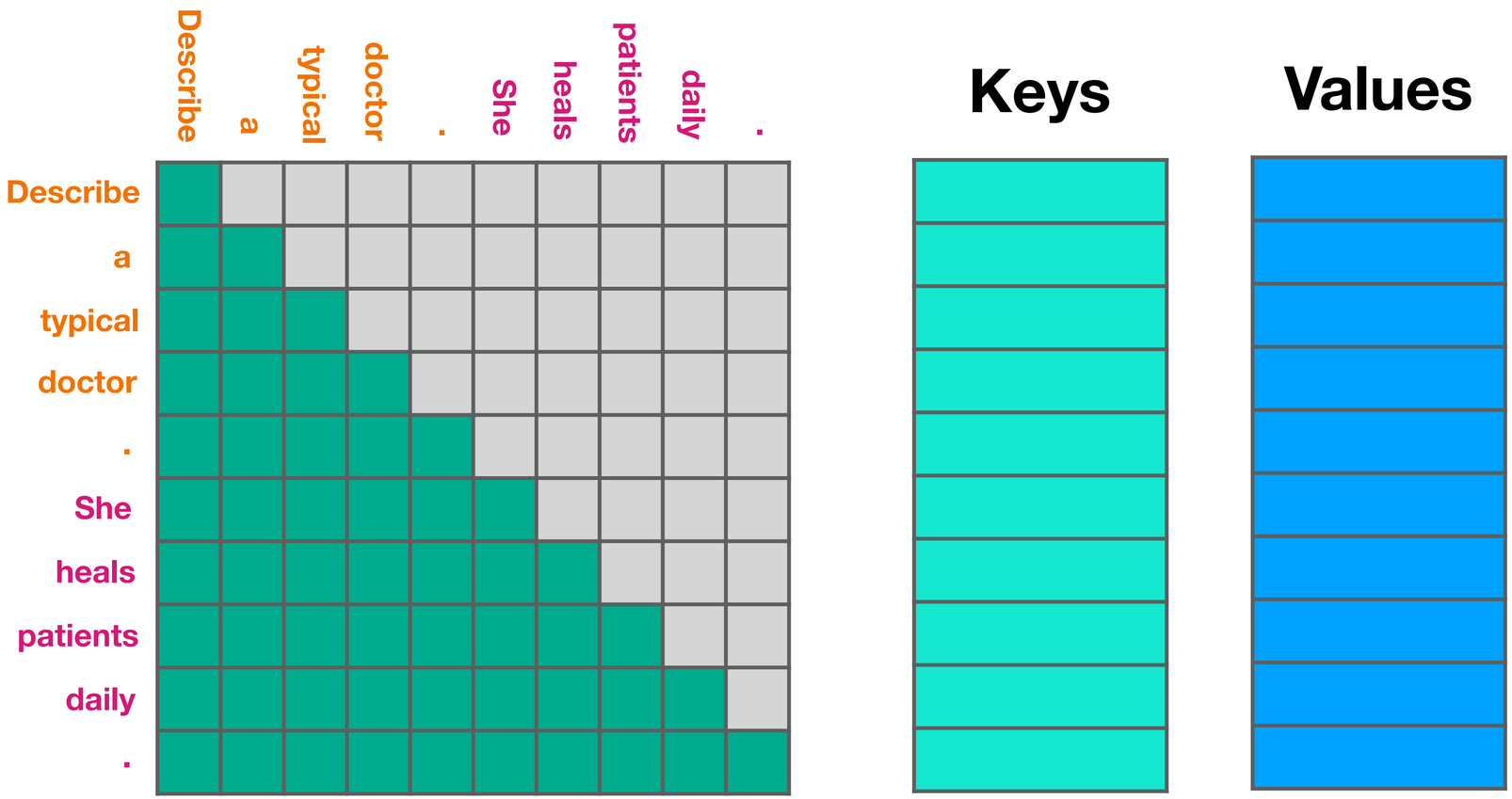


[Lages]

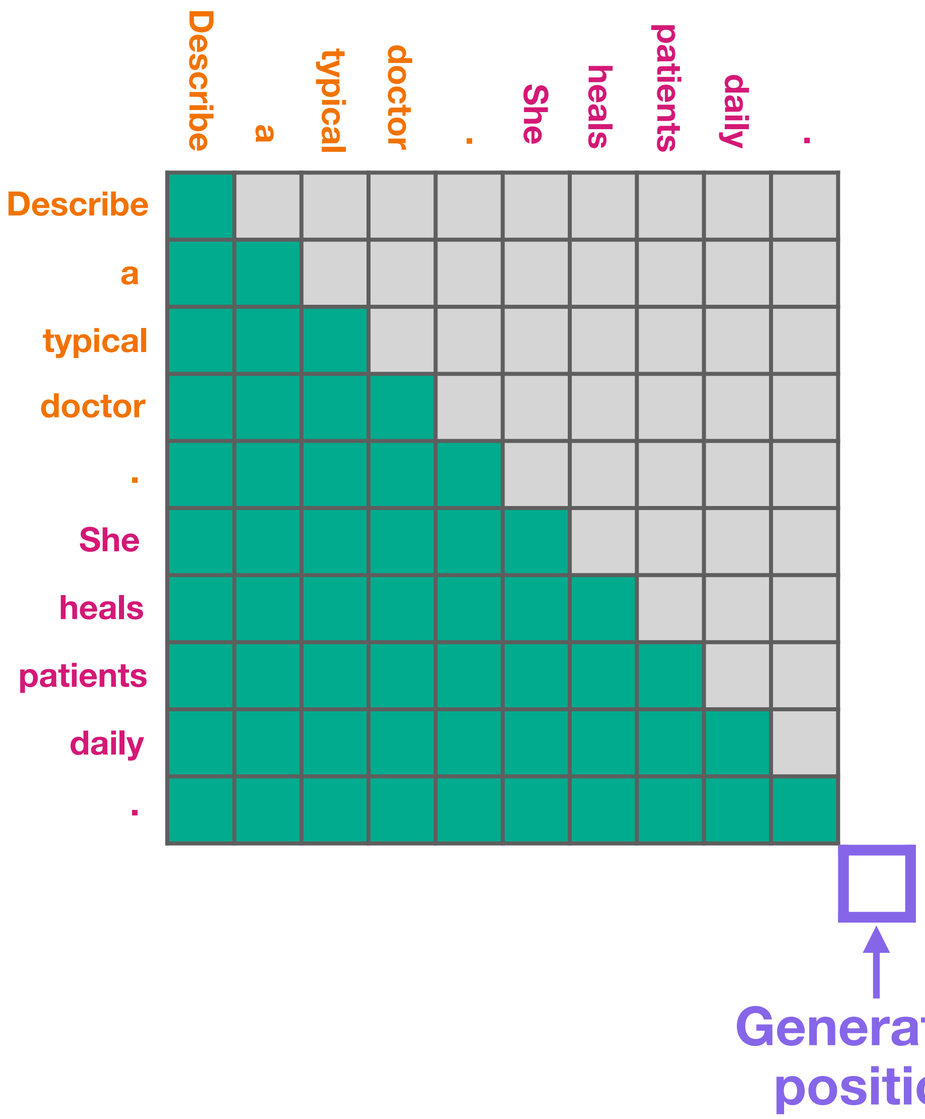
KV Cache size

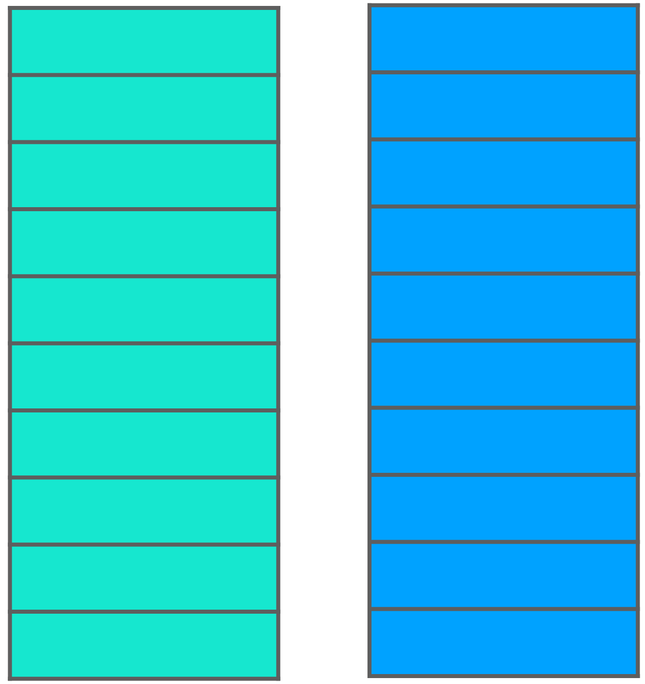
- Large sequences: Cache can get really large
- 30B model, 128 batch size, 1024 sequence length = 180GB KV cache [Zhang et al.]
- Idea: Evict tokens from cache

Which tokens to evict?



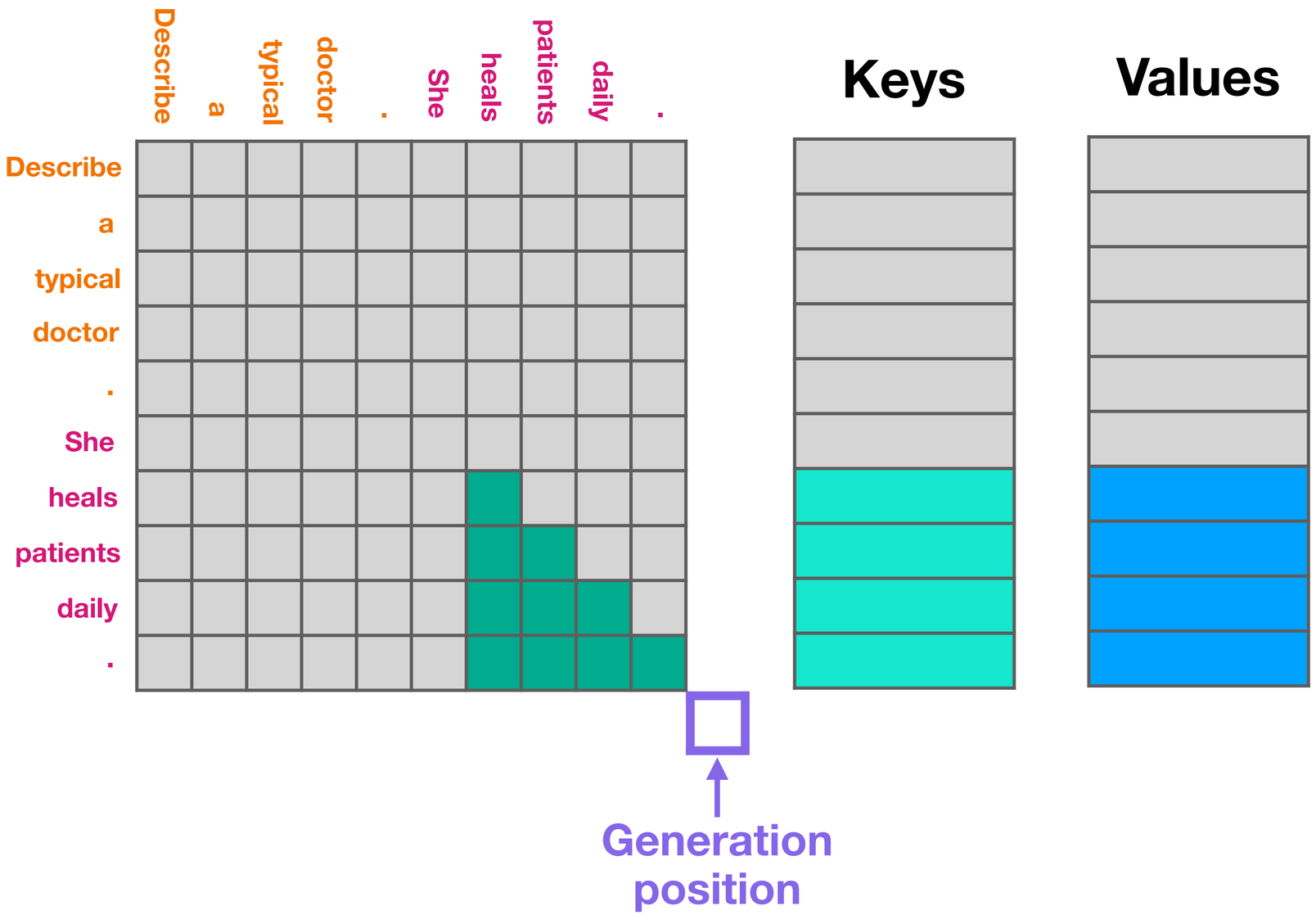
Which tokens to evict?





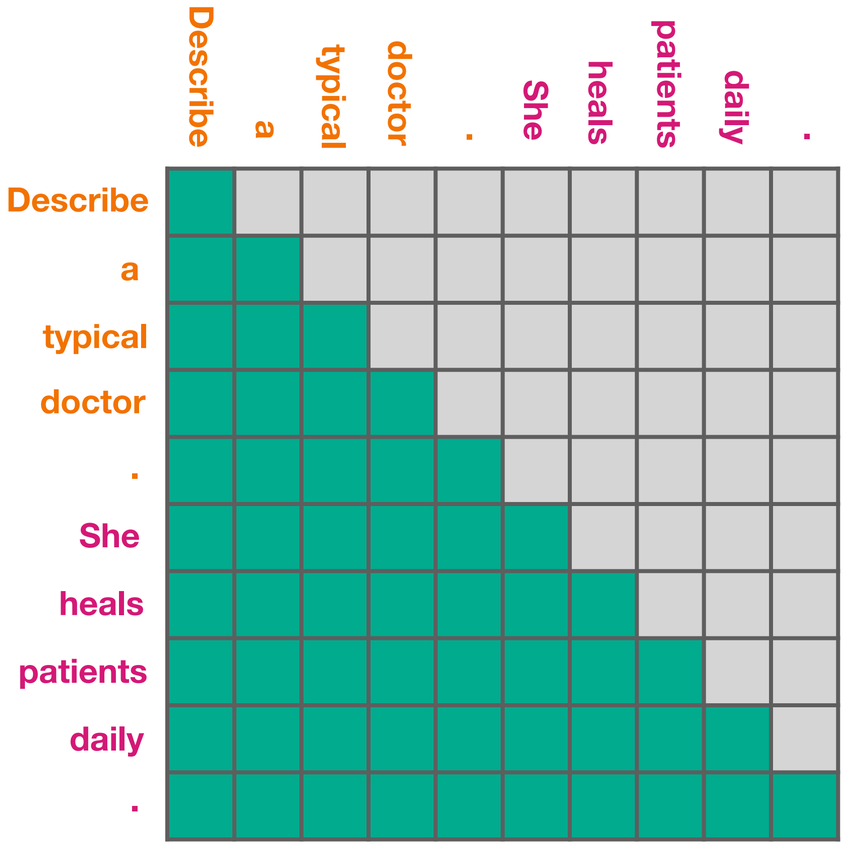
- Oldest?
  - Could have the most information

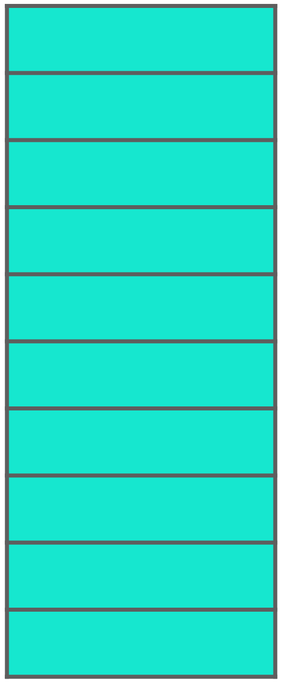
Which tokens to evict?

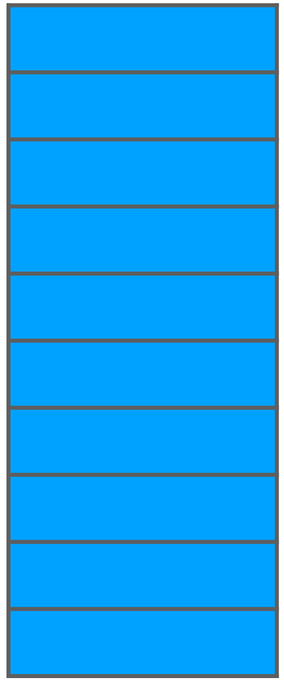


- Oldest?
  - Could have the most information

Which tokens to evict?



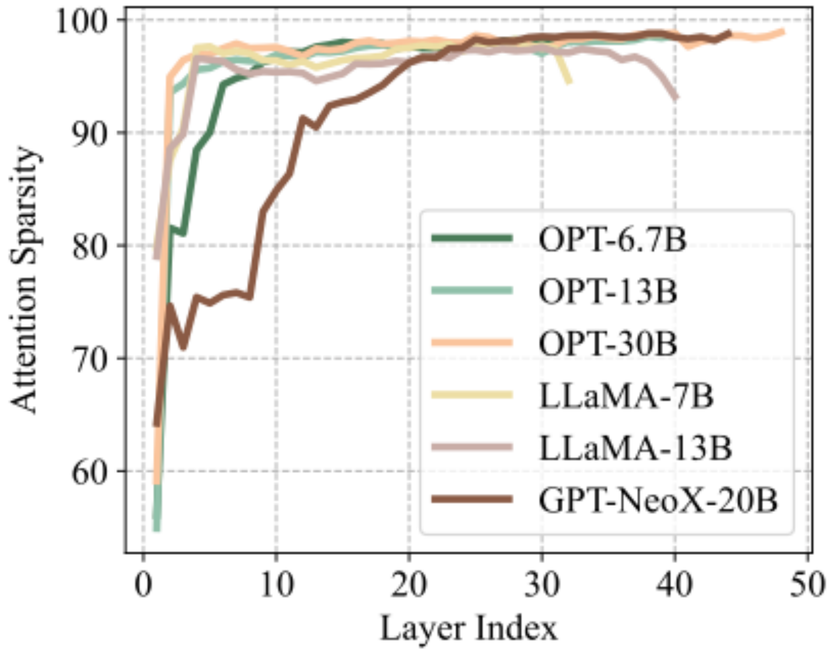




- Oldest?
  - Could have the most information
- At random?
  - Could evict a really important one

Eviction strategies: H2O

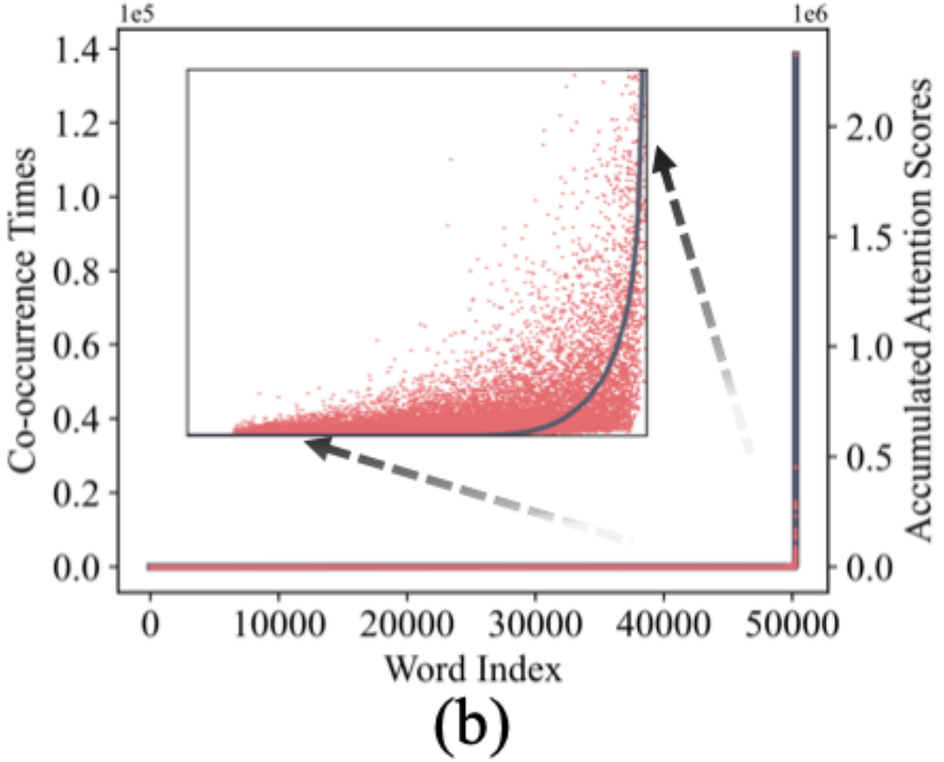
- A few tokens are responsible for most of the attention score [Zhang et al.]
- Attention sparsity score
  - Compute softmax(QK^T)
  - Zero out values that are less than 1% of the max value
  - Plot the fraction of zero values
- Very high sparsity in most layers
- Takeaway: Can drop most previous tokens from cache



[Zhang et al.]

Identifying heavy hitters

- Attention sparsity score
  - Compute softmax(QK^T)
- Plot tokens with highest accumulated attention score
  - How much a token in cache contributes to attention
- Takeaway: Very few tokens contribute most attention
- Proposes an algorithms for identifying heavy hitters



[Zhang et al.]

Exercise

Sink attention

- **Key idea:** Some tokens act as *attention sinks*
  - Take up a large attention score
  - Generally tokens in the beginning, even if semantically unimportant

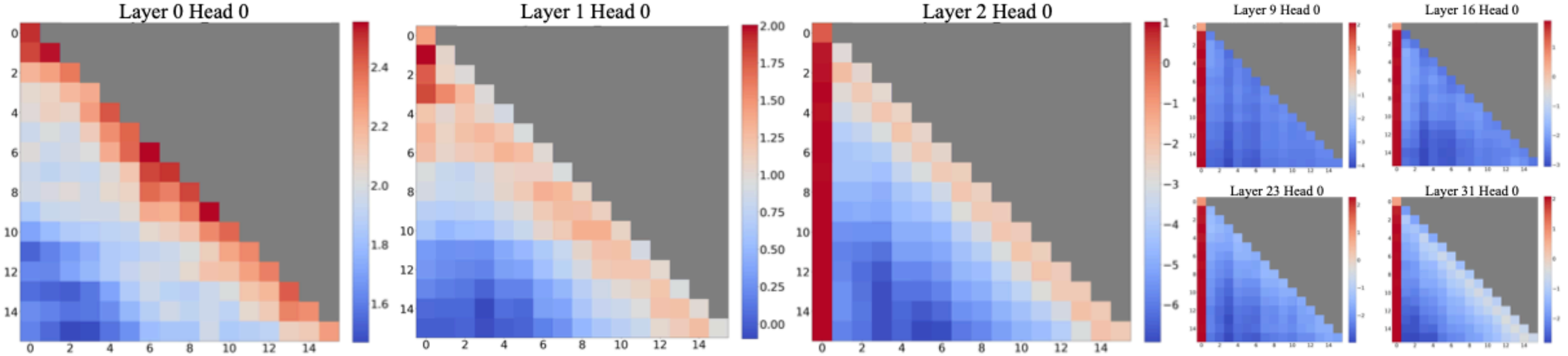


Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two layers, the model heavily attends to the initial token across all layers and heads.

[Xiao et al.]

Why do attention sinks exist?

- An artifact of the softmax computation in attention
  - Softmax needs to sum to 1
  - Ends up assigning disproportionate weight to initial tokens
  - Because initial tokens are visible from every other token

$$\text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{D}}\right)\mathbf{V}$$

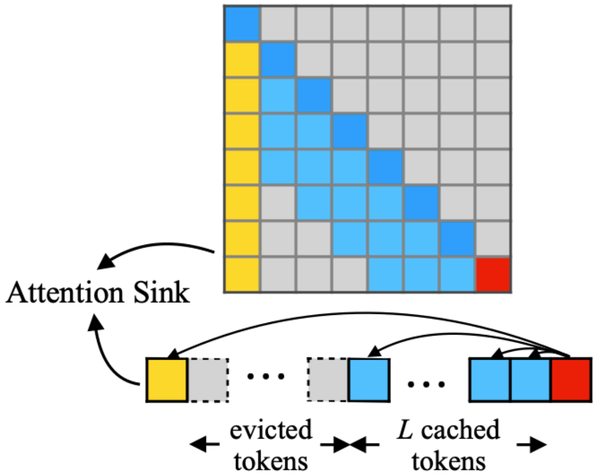
- It is not the *semantics* of the initial tokens, but the *position* that results in this artifact
  - Tested the hypothesis by replacing initial four tokens with `\n`
  - Still got very high attention on initial tokens

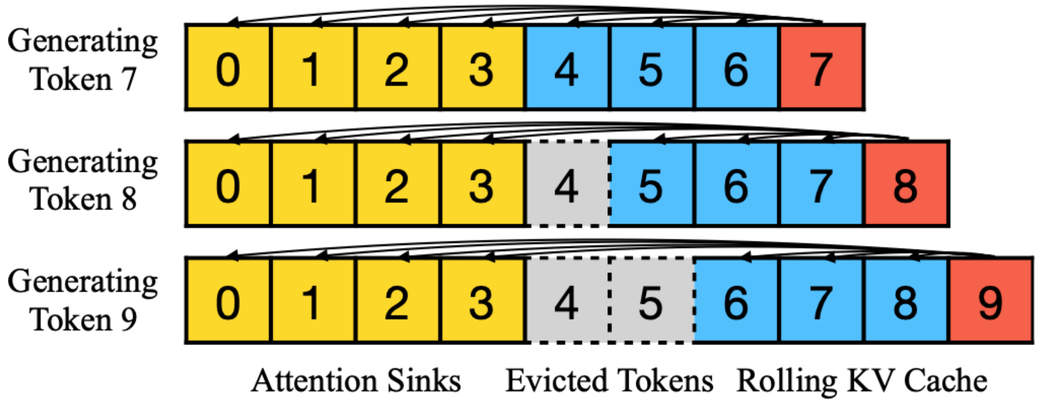
- **Takeaway:** Do not evict initial tokens

[Xiao et al.]

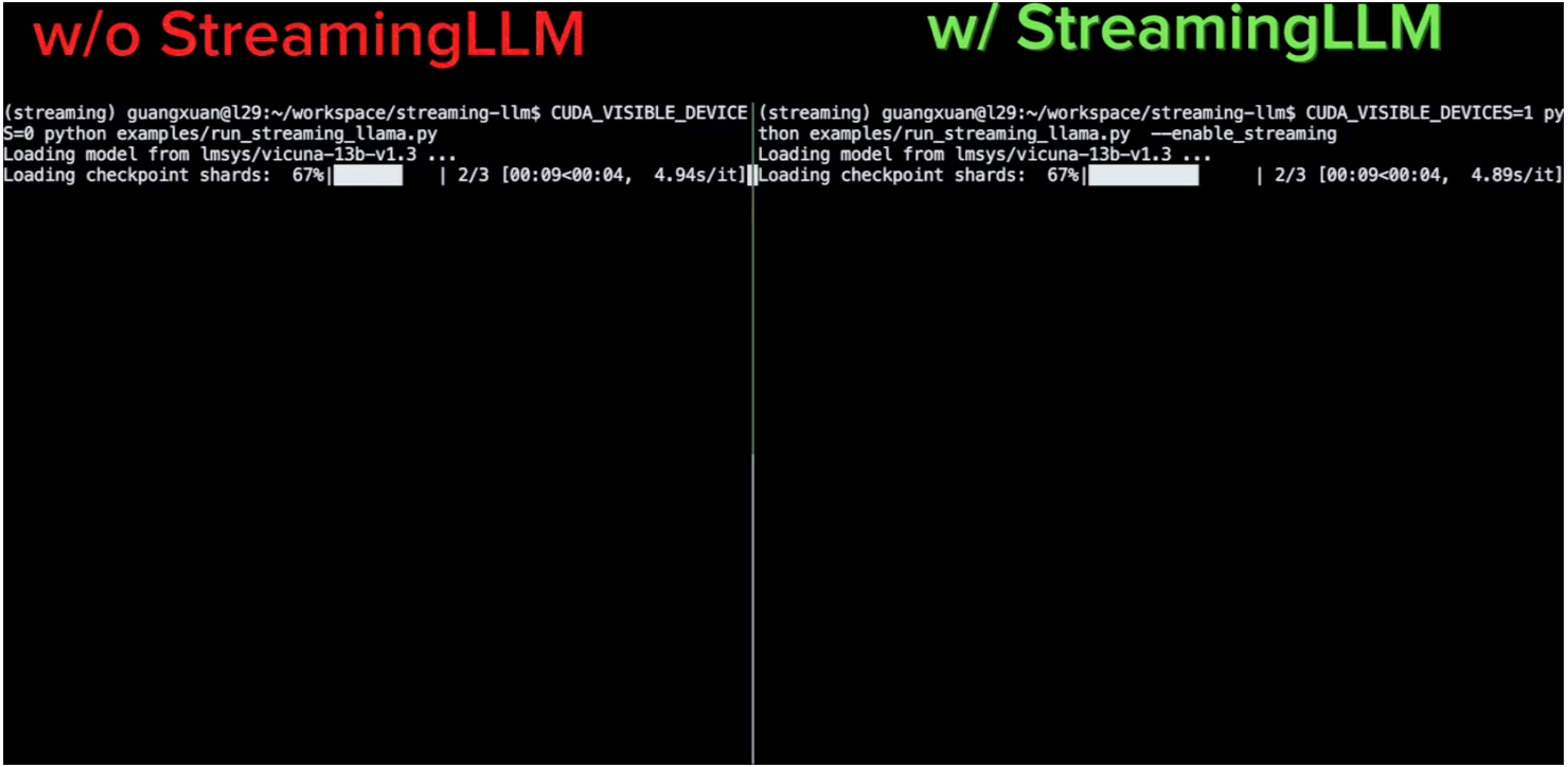
Streaming LLM

- **Key ideas**
  - Keep the initial tokens in KV cache
  - Keep the most recent tokens (sliding window)

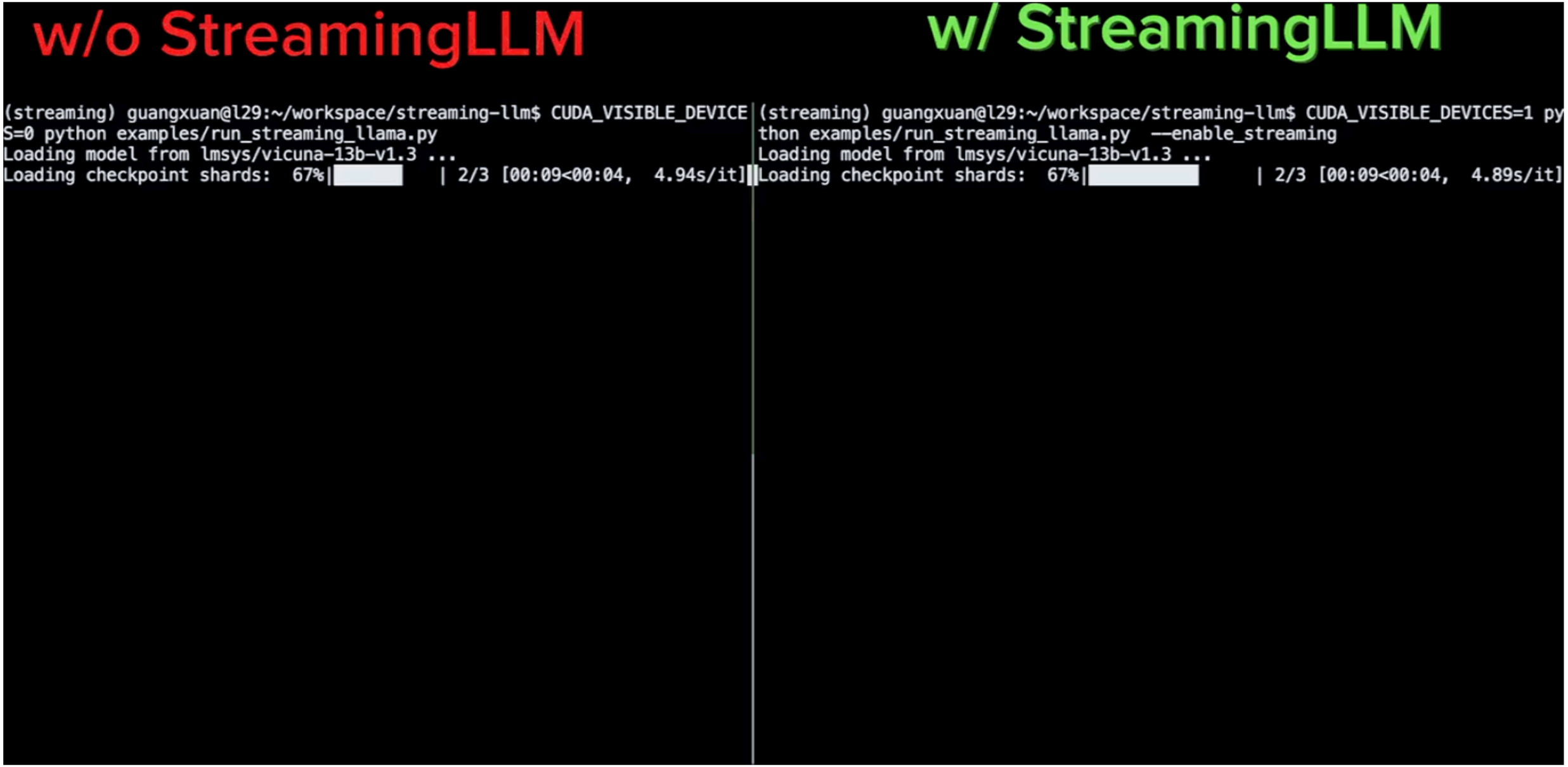




[Xiao et al.]



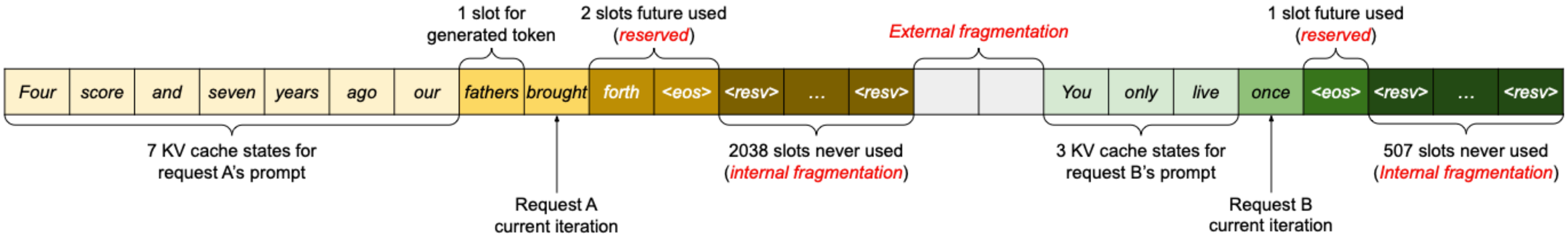
[StreamingLLM]



[StreamingLLM]

Hardware-aware optimization

Waste of space in KV caching

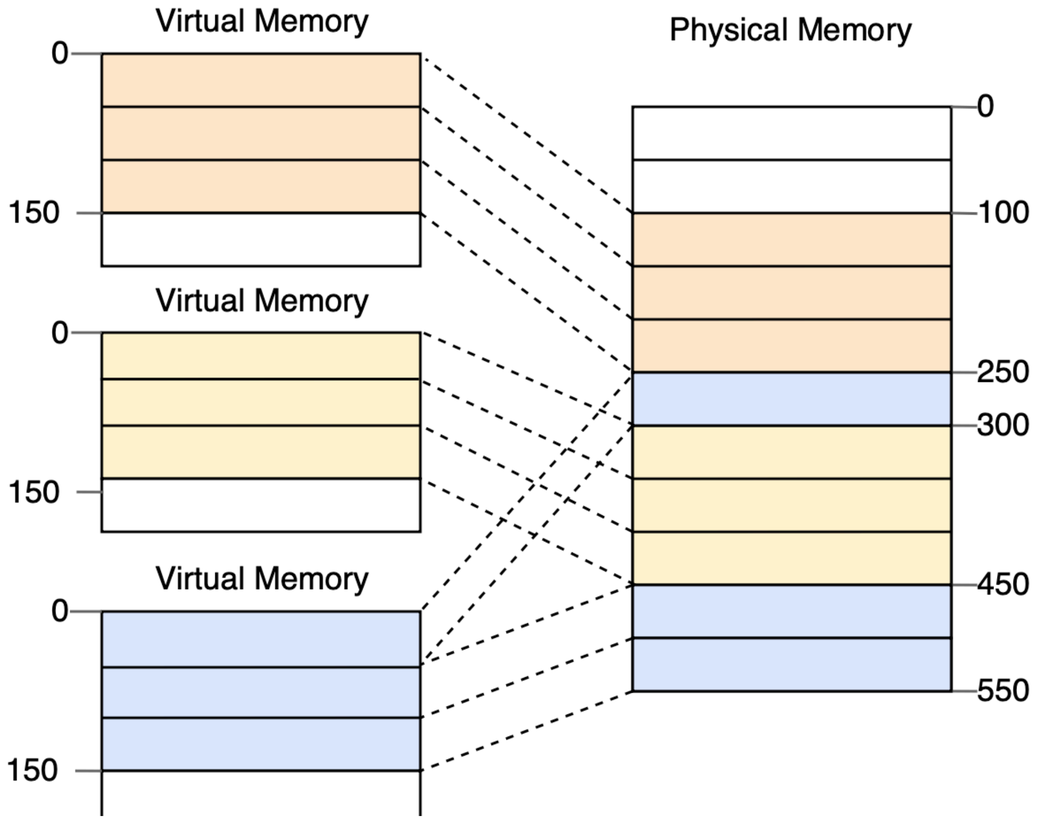


- Two requests: First with max_sequence length = 2048 and second with 512
- Generations much smaller than expected. We end up wasting a lot of space through fragmentation

[Kwon et al.]

Paged Attention

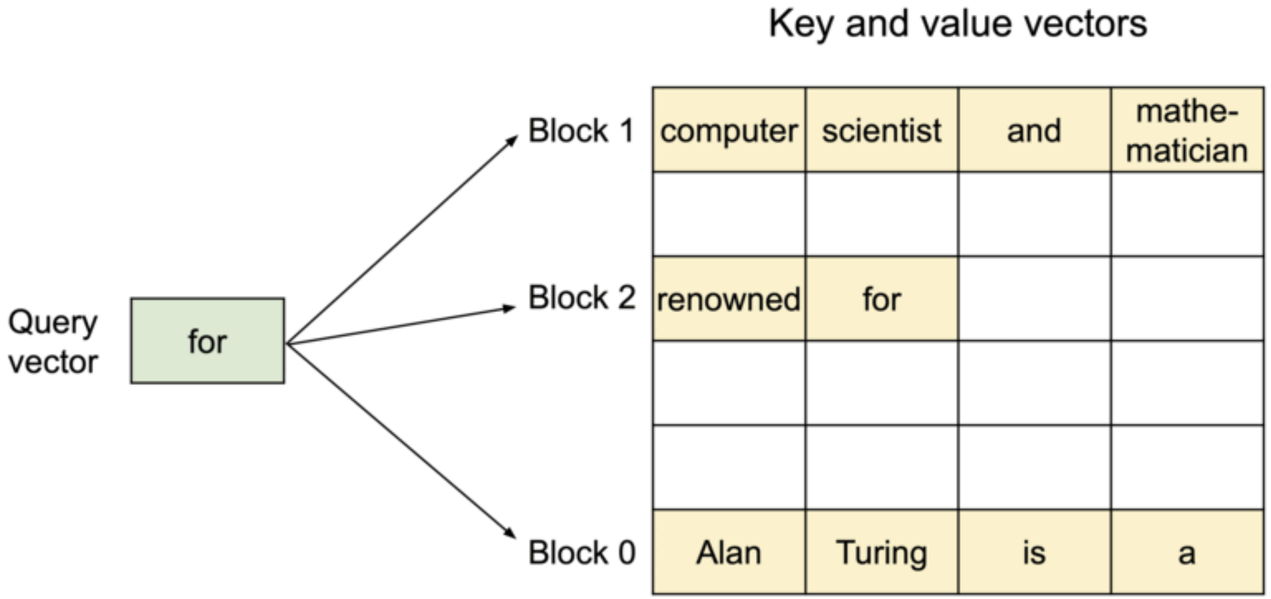
- Inspired by how operating systems manage RAM via Memory Paging
- Flexibly allocating memory for different processes
- The memory looks contiguous to each process but in practice is non-contiguous



[Oppermann]

Paged Attention

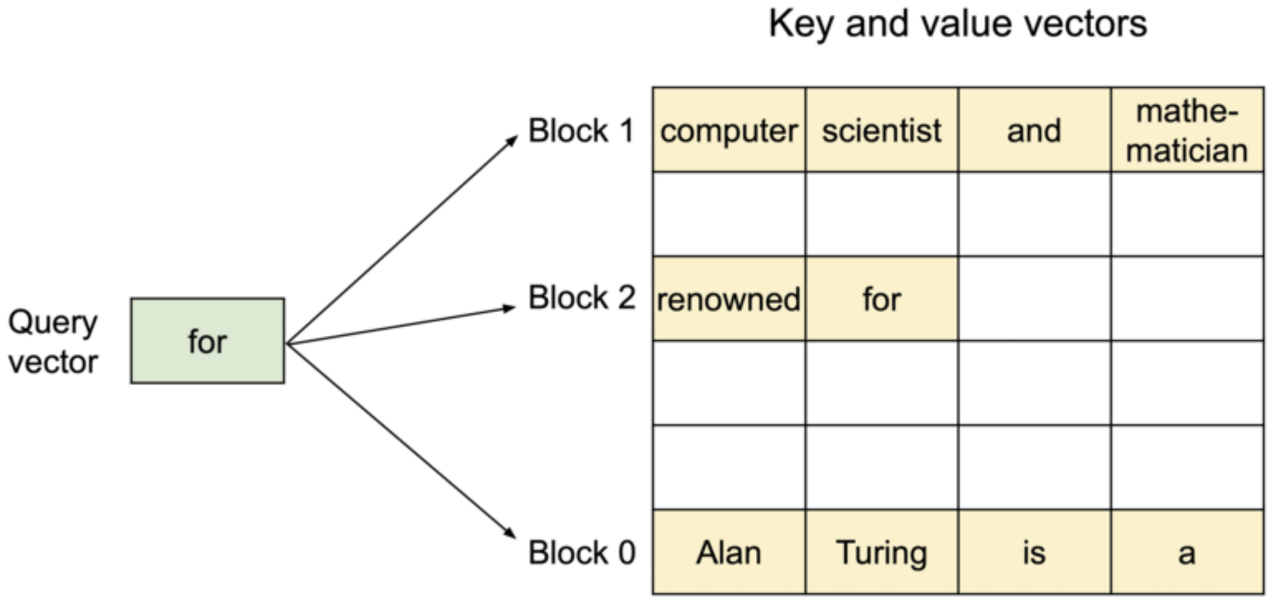
- Split KV cache of each sequence into a block
- Store the block in a non-contiguous, on-demand manner, less fragmentation



[vLLM Blog]

Paged Attention

- Split KV cache of each sequence into a block
- Store the block in a non-contiguous, on-demand manner, less fragmentation



[vLLM Blog]

Allocating KV Cache on demand

0. Before generation.

Seq A

Prompt: “Alan Turing is a computer scientist”
Completion: “”

Logical KV cache blocks

Block 0				
Block 1				
Block 2				
Block 3				

Block table

Physical block no.	# Filled slots
–	–
–	–
–	–
–	–

Physical KV cache blocks

Block 0				
Block 1				
Block 2				
Block 3				
Block 4				
Block 5				
Block 6				
Block 7				

[vLLM Blog]

Allocating KV Cache on demand

0. Before generation.

Seq A

Prompt: “Alan Turing is a computer scientist”
Completion: “”

Logical KV cache blocks

Block 0				
Block 1				
Block 2				
Block 3				

Block table

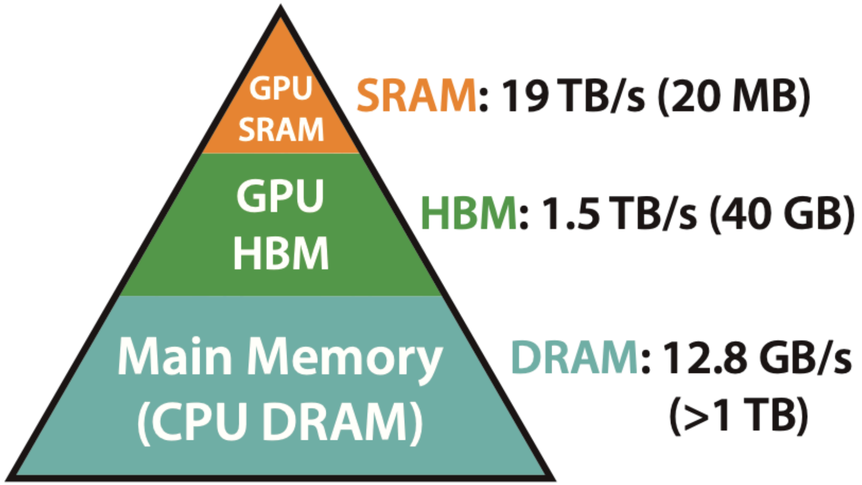
Physical block no.	# Filled slots
–	–
–	–
–	–
–	–

Physical KV cache blocks

Block 0				
Block 1				
Block 2				
Block 3				
Block 4				
Block 5				
Block 6				
Block 7				

[vLLM Blog]

The memory hierarchy

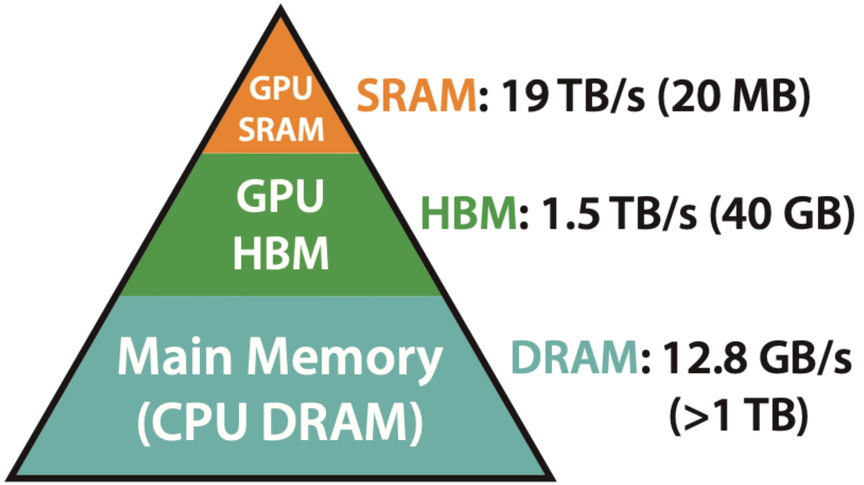


Memory Hierarchy with Bandwidth & Memory Size

Big: Can (often) store a model
Slow in moving data like parameters and activations

[Dao et al.]

The memory hierarchy

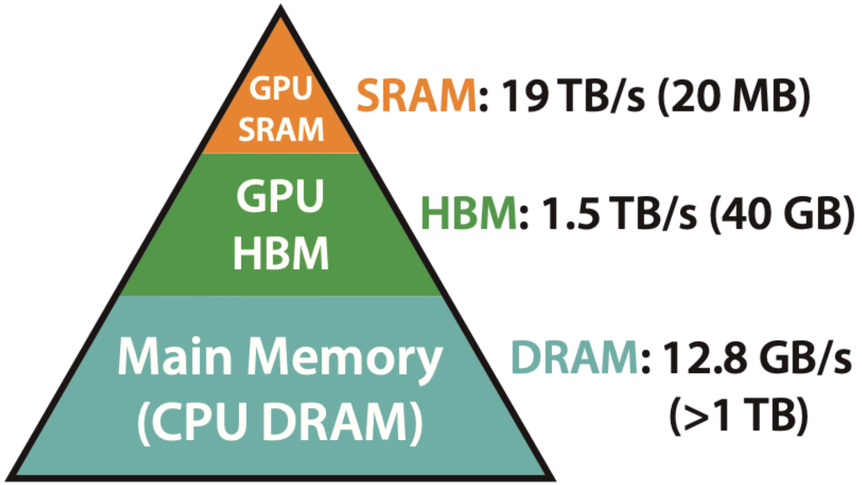


Memory Hierarchy with Bandwidth & Memory Size

Medium: Can store several layers
Somewhat fast in moving data

[Dao et al.]

The memory hierarchy



Memory Hierarchy with Bandwidth & Memory Size

Small: Can store a handful of params
Fast: Parallel computation

[Dao et al.]

Standard attention implementation is slow

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

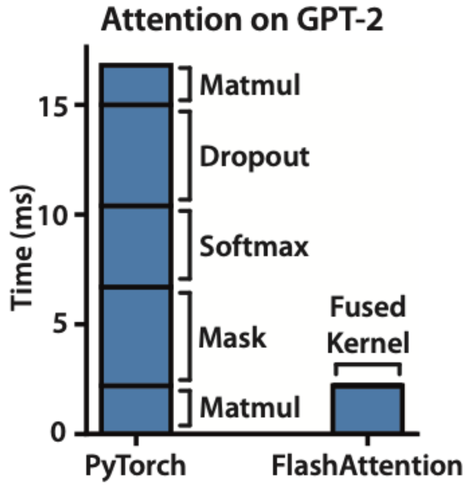
1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

- Needs a lot of back and forth between HBM and SRAM
- Bottlenecked by memory bandwidth

[Dao et al.]

Flash Attention

- **Key Idea 1: Fused Kernels**
  - Use Fused kernels perform all attention operations at once
  - No longer need to bring intermediate attention computations back to HBM



- **Key Idea 2: Matrix Tiling**
  - No need to realize the NxN attention matrix
  - Load parts of Q, K and V matrices into SRAM
  - Maintain some extra stats to compute softmax on the fly
  - No need to access all logits at once

- **Homework:** Read Algorithm 1 in the FlashAttention paper (Dao et al.)

[Dao et al.]

Exercise

References

- Attention is All you Need
- A Visual Guide to Mixture of Experts (MoE)
- Mixtral of Experts
- A Simple and Effective Pruning Approach for Large Language Models
- MIT 6.5940: TinyML and Efficient Deep Learning Computing
- Transformers KV Caching Explained
- Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity
- H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models
- Efficient Streaming Language Models with Attention Sinks
- StreamingLLM GitHub
- Efficient Memory Management for Large Language Model Serving with PagedAttention
- vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention
- FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness