Advanced Attention Mechanisms & MOE

CS XXX: Introduction to Large Language Models

Contents

Optimizing Attention

- The area that gets the most focus from the research community is the attention layer of the Transformer. This is because the attention calculation is the most computationally expensive part of the process.

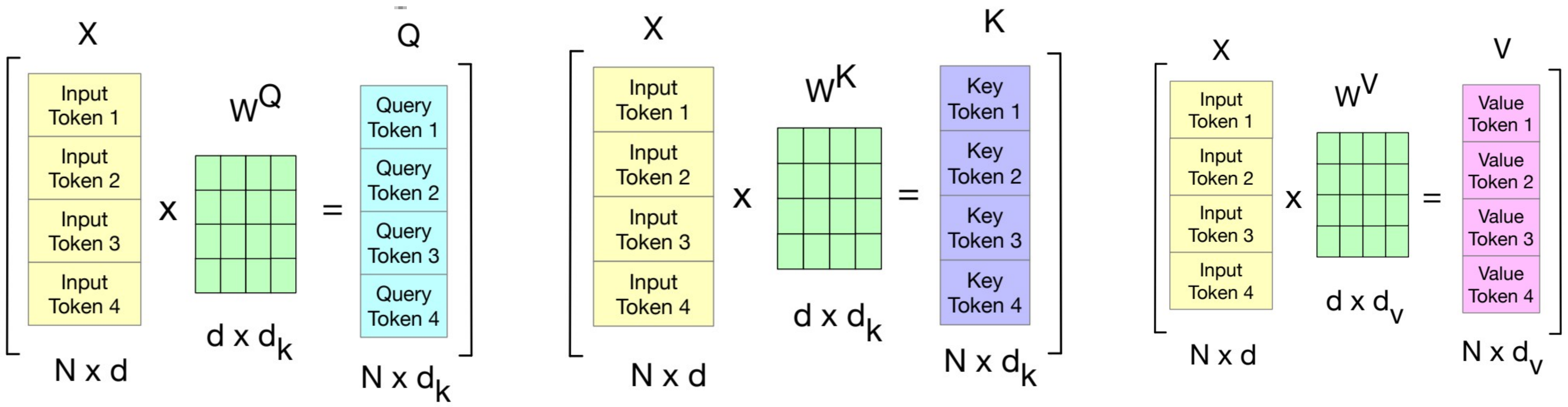
KV Cache

RECALL

- Attention Mechanism

$$\textbf{Self–Attention (A)} = \left(\text{softmax} \left(\textbf{mask} \left(\frac{\mathbf{QK^T}}{\sqrt{d_k}} \right) \right) \right) \mathbf{V}$$

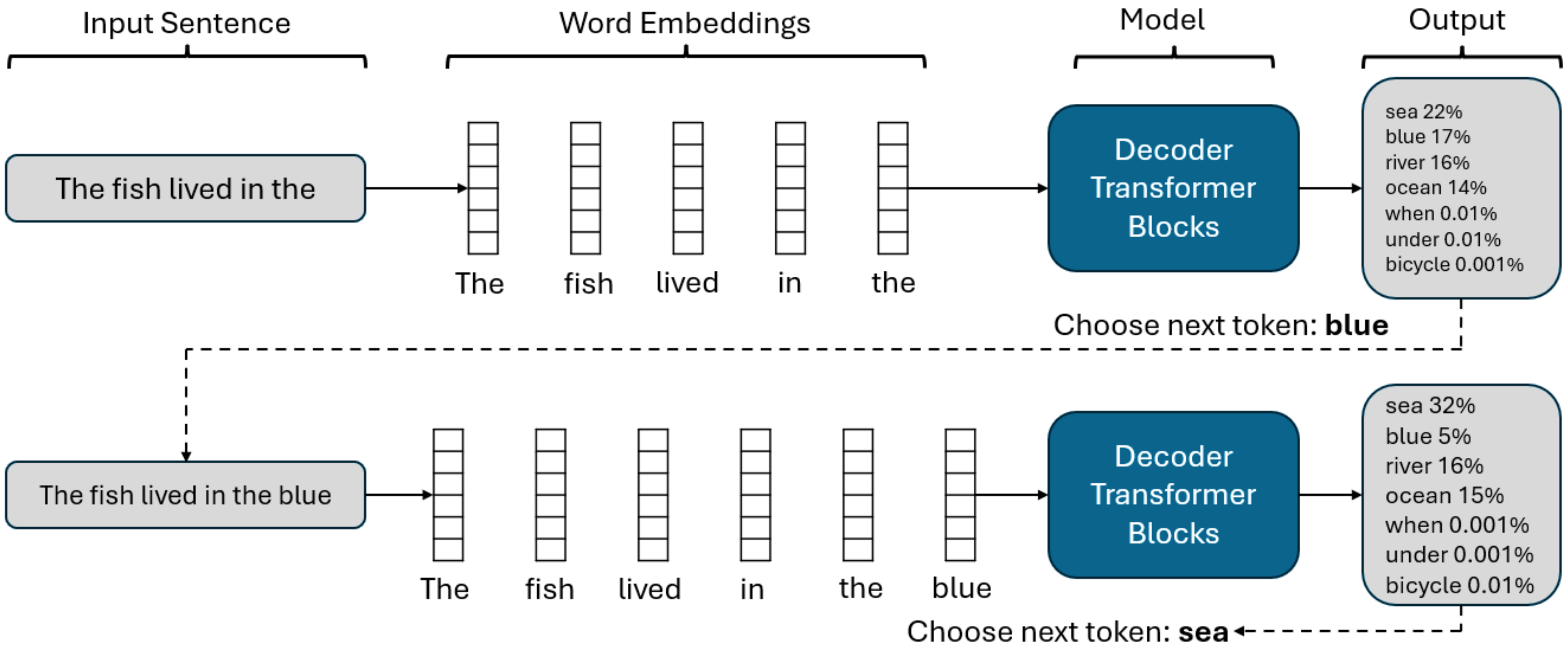
$$\mathbf{Q} = \mathbf{XW}^{\mathrm{Q}}, \qquad \mathbf{K} = \mathbf{XW}^{\mathrm{K}}, \qquad \mathbf{V} = \mathbf{XW}^{\mathrm{V}}$$



KV Cache

RECALL

- **Token by token generation**: Recall that when generating the second token, we simply append the output token to the input and do another forward pass through the model.



KV Cache

- At each time step several dot products to compute the matrix $\mathbf{QK}^{\mathbf{T}}$ are recomputed from the previous time step

- Consider the following computation at time step t for the sequence "John" let the composite embeddings for each word be

$$\text{John} = \begin{bmatrix} a \\ b \end{bmatrix}$$

- And the matrices $\mathbf{W}^K$ and $\mathbf{W}^Q$ are

$$\mathrm{W}^Q = \begin{bmatrix} q_1 & q_2 \\ q_3 & q_4 \end{bmatrix}, \qquad \mathrm{W}^K = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix}$$

KV Cache

- Then the matrices Q and K are

$$\mathrm{Q} = [aq_1 \; + bq_3 \quad aq_2 \; + bq_4 \;]$$

$$\mathrm{K} = [ak_1 \; + bk_3 \quad ak_2 \; + bk_4]$$

- The product $\mathbf{QK^T}$ is

$$\mathrm{QK^T} = [(aq_1 + bq_3)(ak_1 + bk_3) + (aq_2 + bq_4)(ak_2 + bk_4)]$$

- Assume that for this time step the next word predicted is “eats”. This output token is appended to the input at the next time step. The composite embeddings for this new token is

$$\mathrm{eats} = \begin{bmatrix} c \\ d \end{bmatrix}$$

KV Cache

- At time step $t + 1$

$$Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} q_1 & q_2 \\ q_3 & q_4 \end{bmatrix} = \begin{bmatrix} aq_1 + bq_3 & aq_2 + bq_4 \\ cq_1 + dq_3 & cq_2 + dq_4 \end{bmatrix}$$

$$K = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} = \begin{bmatrix} ak_1 + bk_3 & ak_2 + bk_4 \\ ck_1 + dk_3 & ck_2 + dk_4 \end{bmatrix}$$

$$QK^T$$

$$= \begin{bmatrix} (aq_1 + bq_3)(ak_1 + bk_3) + (aq_2 + bq_4)(ak_2 + bk_4) & (aq_1 + bq_3)(ck_1 + dk_3) + (aq_2 + bq_4)(ck_2 + dk_4) \\ (cq_1 + dq_3)(ak_1 + bk_3) + (cq_2 + dq_4)(ak_2 + bk_4) & (cq_1 + dq_3)(ck_1 + dk_3) + (cq_2 + dq_4)(ck_2 + dk_4) \end{bmatrix}$$

Was computed previously at time step t

KV Cache

- At each time step there are values that we re-calculate values already calculated in the previous time step.
- If Q and K are matrices of size 4×4096 at some time step t

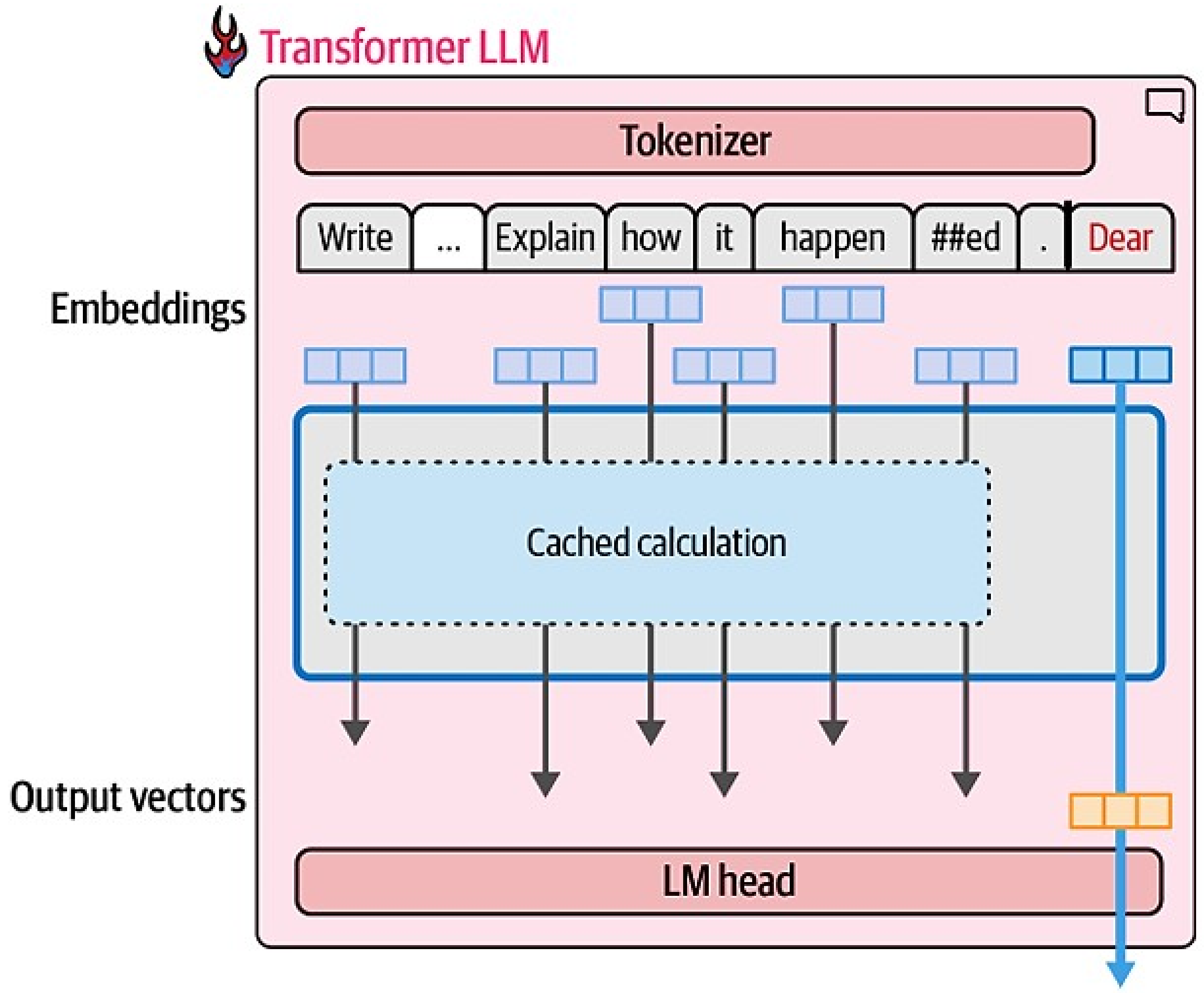
$$QK^T = \begin{bmatrix} {\color{red}\cdot} & \cdot & \cdot & \cdot \\ {\color{red}\cdot} & {\color{red}\cdot} & \cdot & \cdot \\ {\color{red}\cdot} & {\color{red}\cdot} & {\color{red}\cdot} & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

the red values were computed in the previous time step $t-1$.

- If we give the model the ability to cache the results of the previous calculation, we no longer need to repeat the calculations of the previous streams. This time the only needed calculation is for the last stream. This is an optimization technique called the keys and values (KV) cache and it provides a significant speedup of the generation process.

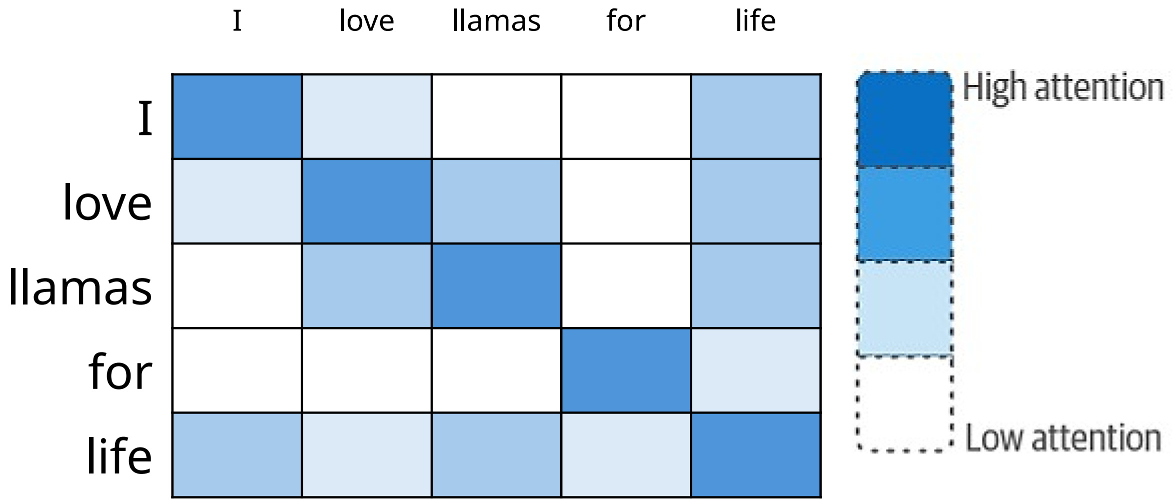
KV Cache

- When generating text, it’s important to cache the computation results of previous tokens instead of repeating the same calculation over and over again



Sliding Window Attention

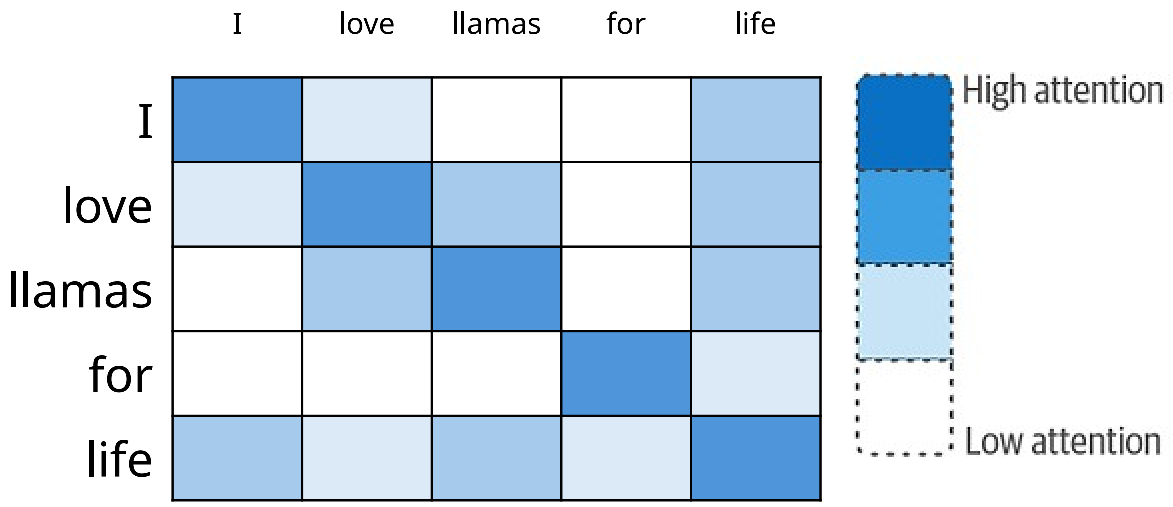
- Attention scores determine how strongly the model on the relationship between certain words



- "love" has high attention on "llamas," it means the model considers "love" important in relation to "llamas."

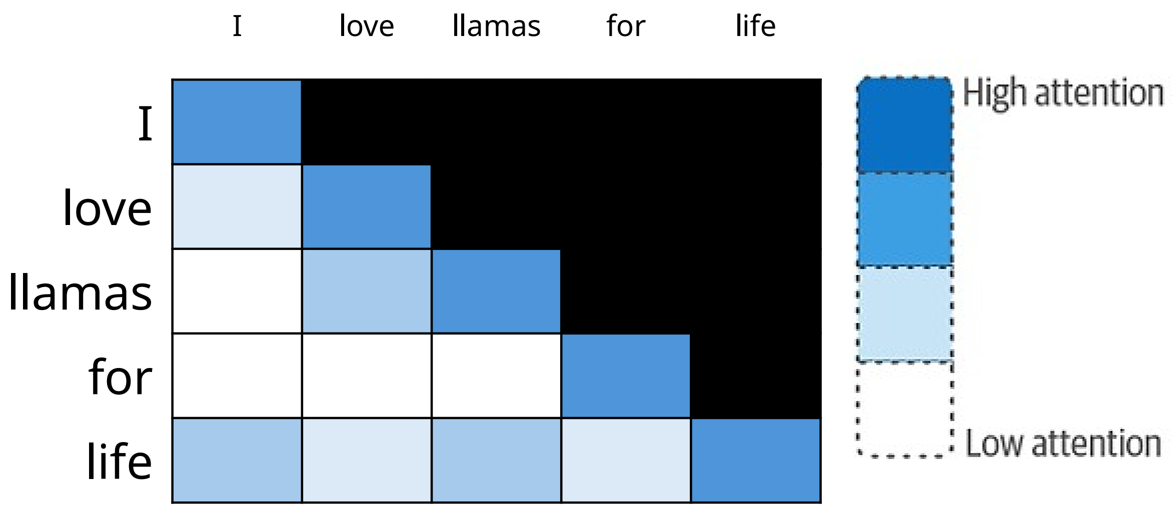
Sliding Window Attention

- Full n^2 attention: every token in the input sequence attends to every other token. This means that for an input sequence of length n, the attention mechanism computes relationships between all n tokens, leading to a complexity of $O(n^2)$ in both computation and memory.



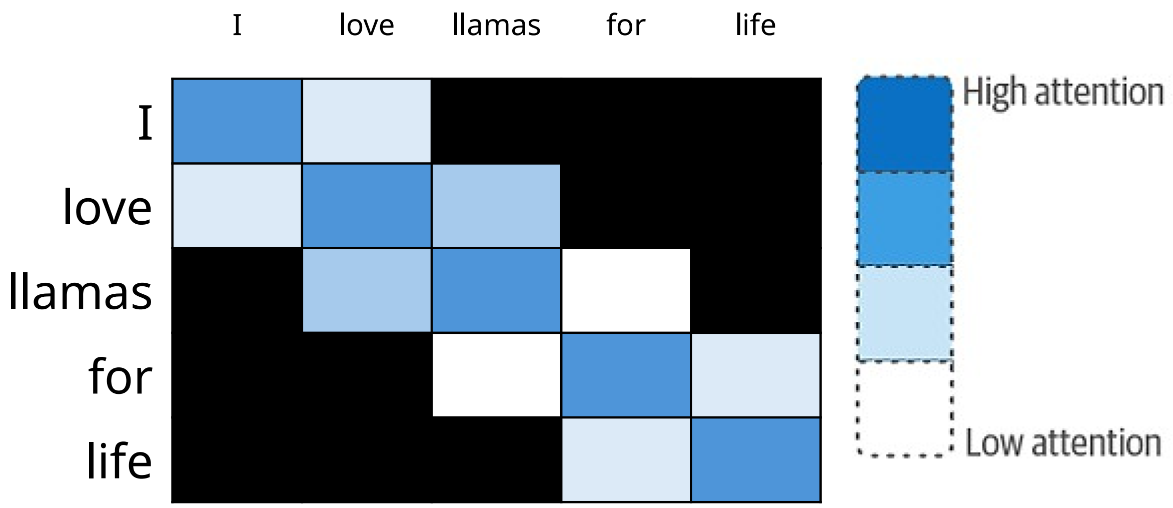
Sliding Window Attention

- Full $\frac{n^2}{2}$ attention: each token can only attend to itself and past tokens, preventing information leakage from future tokens. This introduces a triangular mask in the attention matrix, effectively halving the number of token pairs considered.



Sliding Window Attention

- Sliding Window Attention: Sliding Window Attention (SWA) is a more efficient alternative to full n^2 attention, designed to reduce computational complexity by limiting the number of attended tokens per position. Instead of attending to all past and future tokens, each token only attends to a fixed-size local window around it.
- Fixed Window Size w: Each token only attends to w tokens before and after it. If $w = 1$



- $O(nw)$ instead of $O(n^2)$ making it more scalable for long sequences

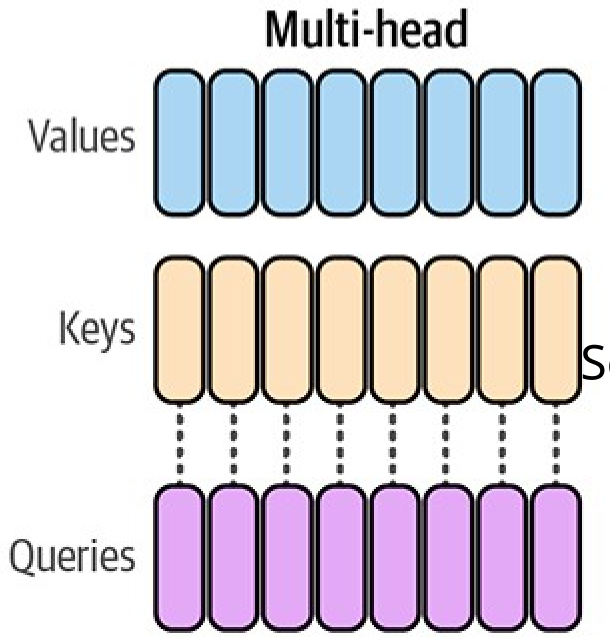
Multi Query Attention

- Multihead Attention

$$\mathbf{Q}^i = \mathbf{X}\mathbf{W}_i^{\mathbf{Q}} \qquad \mathbf{K}^i = \mathbf{X}\mathbf{W}_i^{\mathbf{K}} \qquad \mathbf{V}^i = \mathbf{X}\mathbf{W}_i^{\mathbf{V}}$$

$$\text{head}_i = \text{Self–Attention } (\mathbf{A}) = \left(\text{softmax}\left(\frac{\mathbf{Q}^i {\mathbf{K}^i}^{\mathbf{T}}}{\sqrt{d_k}} \right) \right) \mathbf{V}^i$$

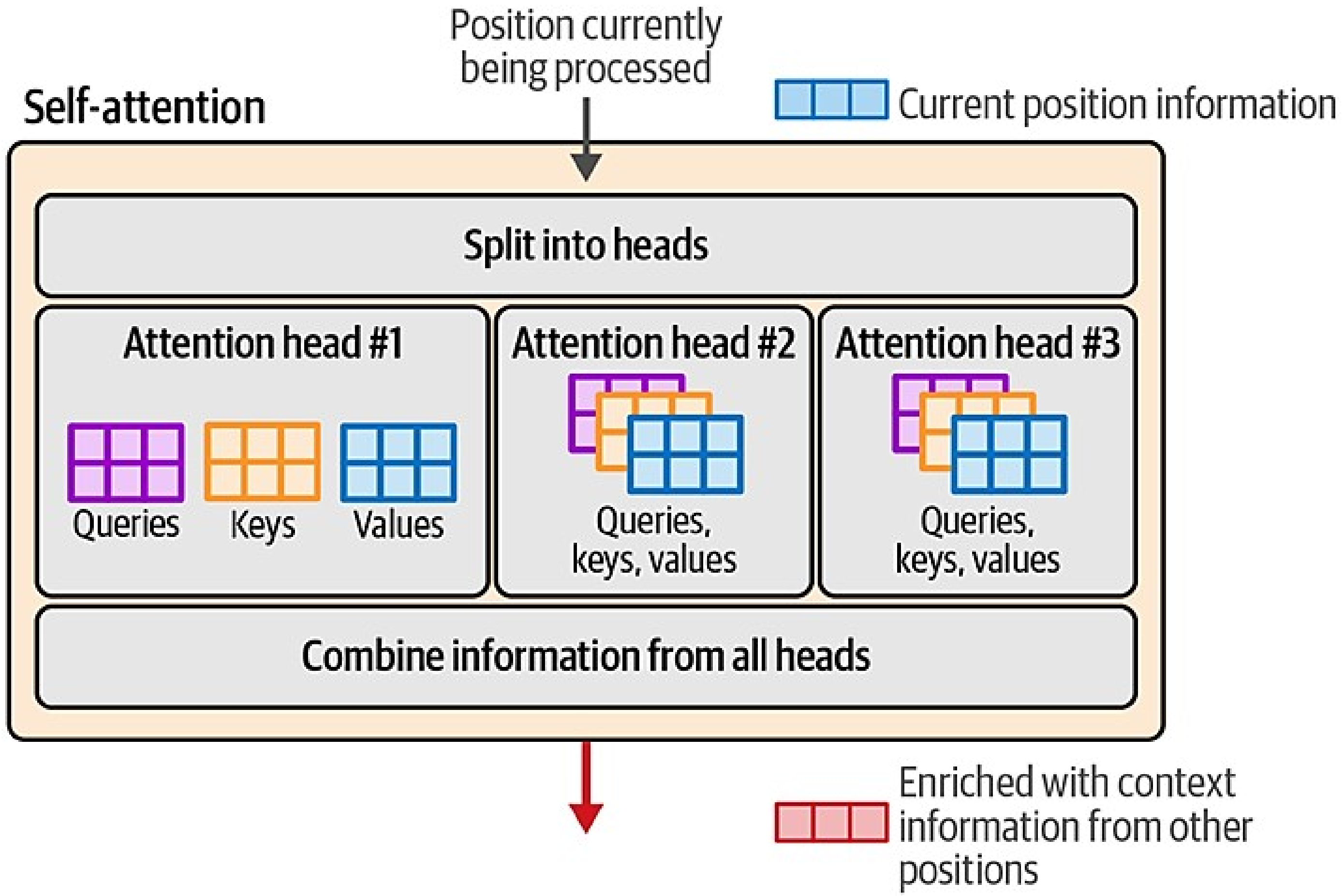
$$\text{MultiHead Attention } (\mathbf{M}) = (\text{head}_1 \oplus \text{ head}_2 \ \ldots \oplus \text{head}_h)\mathbf{W}^0$$



Separate key and value matrices for each head

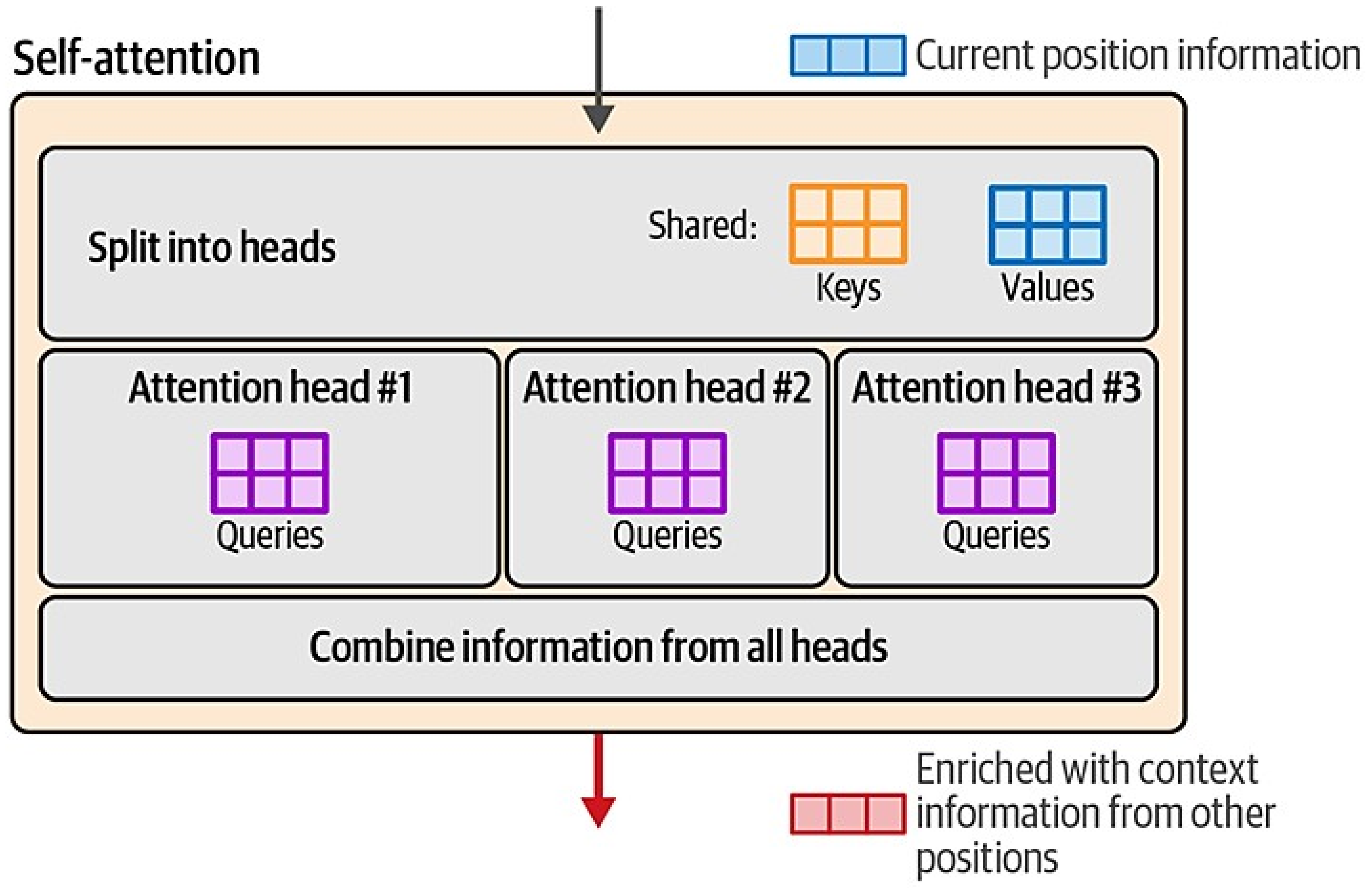
Multi Query Attention

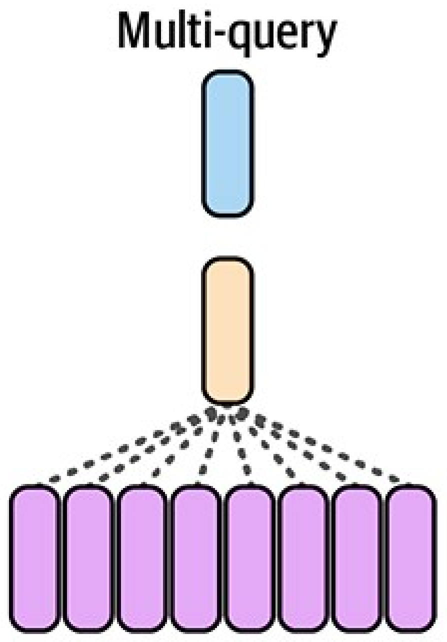
- Multihead Attention
  - Attention is conducted using matrices of queries, keys, and values. In multi-head attention, each head has a distinct version of each of these matrices.


Position currently being processed
Self-attention
Current position information
Split into heads
Attention head #1
Queries
Keys
Values
Attention head #2
Queries, keys, values
Attention head #3
Queries, keys, values
Combine information from all heads
Enriched with context information from other positions

Multi Query Attention

- Multi-query attention presents a more efficient attention mechanism by sharing the keys and values matrices across all the attention heads.



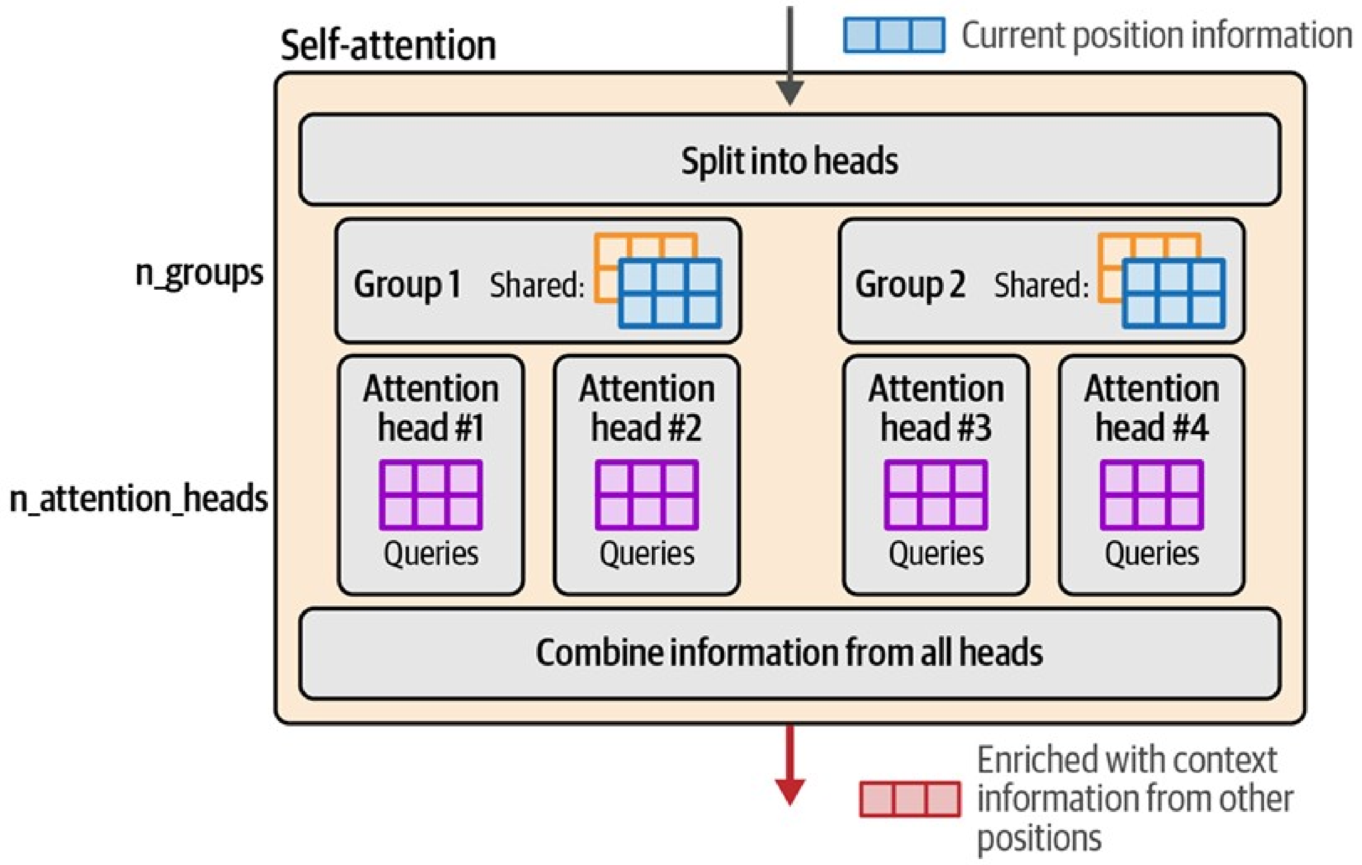


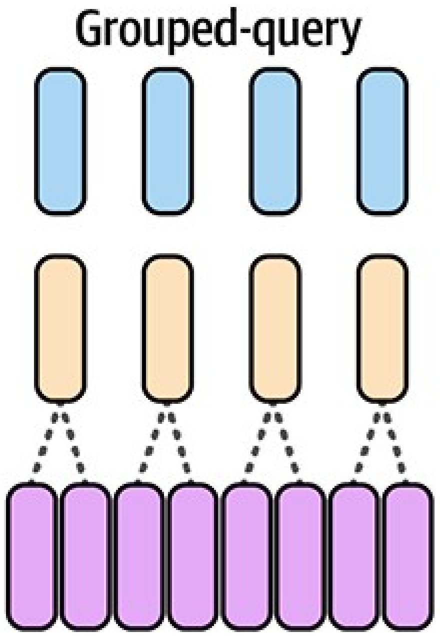
Grouped Query Attention

- As model sizes grow, however, multi-query optimization can be too punishing and we can afford to use a little more memory to improve the quality of the models. This is where grouped-query attention comes in. Instead of cutting the number of keys and values matrices to one of each, it allows us to use more (but less than the number of heads).

Grouped Query Attention

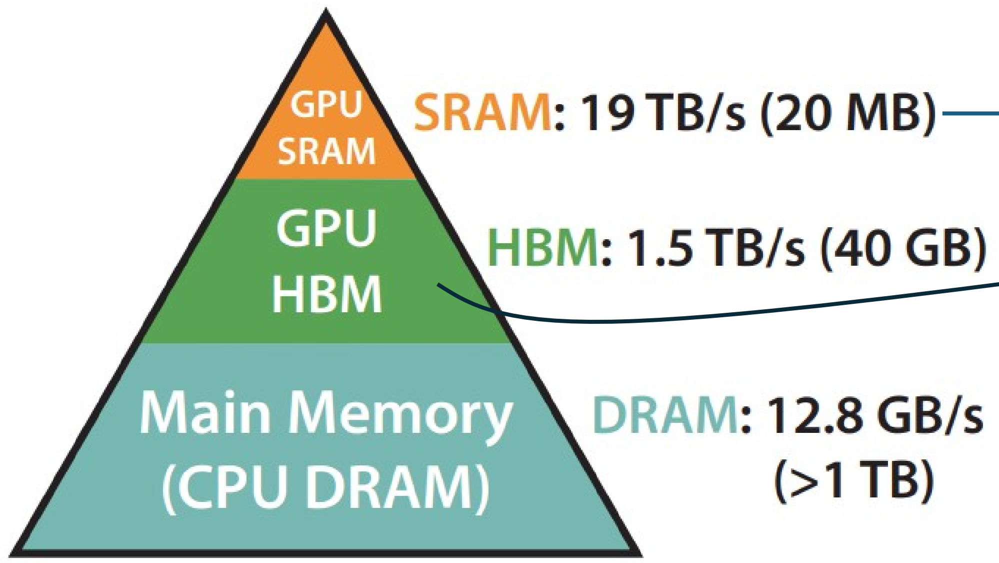
- Grouped-query attention sacrifices a little bit of the efficiency of multi query attention in return for a large improvement in quality by allowing multiple groups of shared key/value matrices; each group has its respective set of attention heads.





Flash Attention

- Flash Attention is a popular method and implementation that provides significant speedups for both training and inference of Transformer LLMs on GPUs. It speeds up the attention calculation by optimizing what values are loaded and moved between a GPU’s shared memory (SRAM) and high bandwidth memory (HBM).



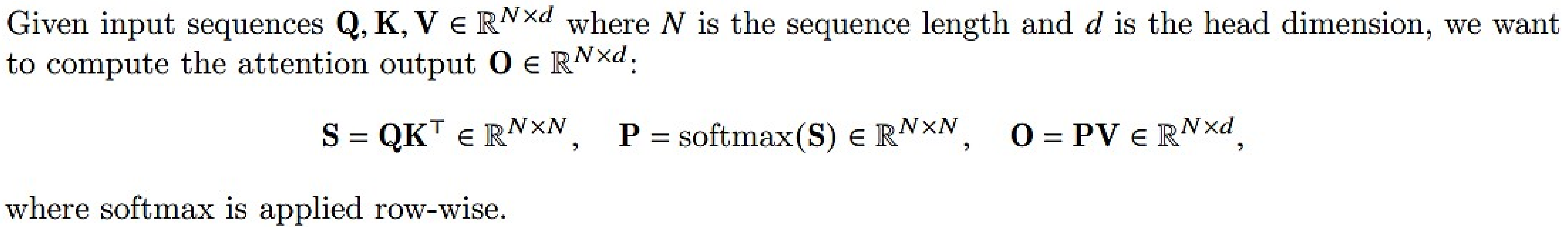
Memory Hierarchy with Bandwidth & Memory Size

SRAM: Speed (19 TB/s) / Storage (20 MB) – Faster computation but very little storage

HBM (GPU main memory): Speed (1.5 TB/s) / Storage (40 GB) – Slower computation and access (loading from and writing to) but lots of storage

Flash Attention

- Standard Attention



Given input sequences $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ where N is the sequence length and d is the head dimension, we want to compute the attention output $\mathbf{O} \in \mathbb{R}^{N \times d}$:

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^{\top} \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \text{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

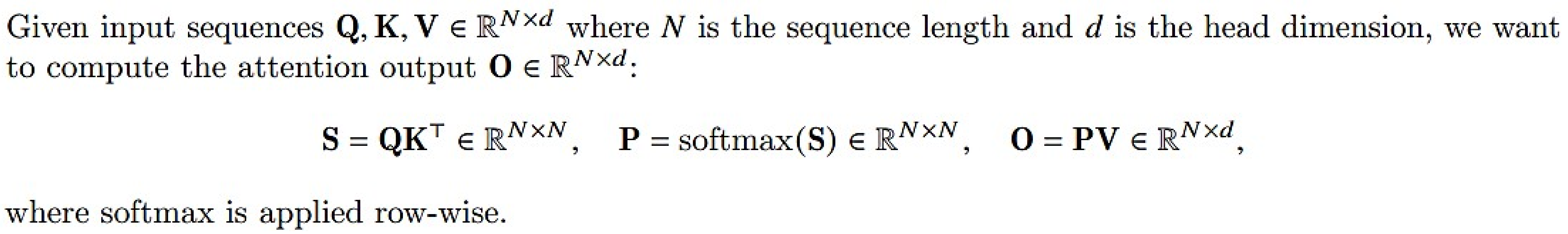
where softmax is applied row-wise.

Require: Matrices **Q**, **K**, **V** in HBM

```
1 Load Q, K from HBM
2 Compute S = QK^T
3 Write S to HBM.
4 Read S from HBM
5 Compute P = softmax(S)
6 write P to HBM.
7 Load P and V by blocks from HBM
8 Compute O = PV
9 write O to HBM.
```

Flash Attention

- Standard Attention



Given input sequences $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ where N is the sequence length and d is the head dimension, we want to compute the attention output $\mathbf{O} \in \mathbb{R}^{N \times d}$:

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

where softmax is applied row-wise.

```
Require: Matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ in HBM
Load $\mathbf{Q}$, $\mathbf{K}$ from HBM
compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$
Write $\mathbf{S}$ to HBM.
Read $\mathbf{S}$ from HBM
Compute $\mathbf{P}$ = softmax($\mathbf{S}$)
Write $\mathbf{P}$ to HBM.
Load $\mathbf{P}$ and $\mathbf{V}$ from HBM
Compute $\mathbf{O}$ = $\mathbf{P}\mathbf{V}$
Write $\mathbf{O}$ to HBM.
```

Flash Attention

- Standard Attention

```
Require: Matrices Q, K, V in HBM
Load Q, K from HBM
compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^T$
Write S to HBM.
Read S from HBM
Compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$
Write P to HBM.
Load P and V from HBM
Compute $\mathbf{O} = \mathbf{P}\mathbf{V}$
Write O to HBM.
```

- Number of reads from the HBM = 3
- Number of writes to HBM = 3

Goal of flash attention: Reduce HBM accesses, Do most of the computation using SRAM

Flash Attention

- Goal of Flash Attention

```
Require: Matrices Q, K, V in HBM
Load Q, K, V Block from HBM
Compute $S = QK^T$
Compute $P = \text{softmax}(S)$
Compute $O = PV$
write O to HBM.
```

$\}$ Do all the computation using SRAM without having to read and write to HBM at each step because of the limited storage of the SRAM

- Number of Reads: 1
- Number of Writes: 1
- Total Accesses = 2

Flash Attention

- Goal of Flash Attention

```
Require: Matrices Q, K, V in HBM
Load Q, K, V from HBM
Compute $S = QK^T$
Compute $P = \text{softmax}(S)$
Compute $O = PV$
write O to HBM.
```

Do all the computation using SRAM without having to read and write to HBM at each step because of the limited storage of the SRAM

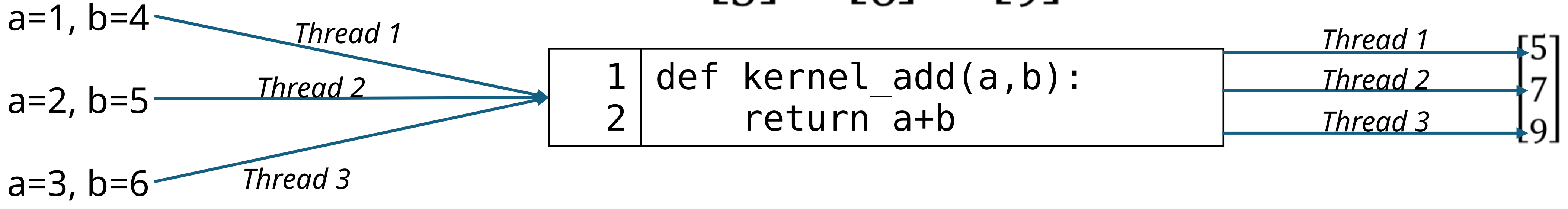
- To utilize SRAM memory efficiently Flash attention uses tiling: The Q, K, V matrices are split into blocks and then attention is computed by blocks.
- Tiling enables computation to be in one CUDA kernel, loading input from HBM, performing all the computation steps (matrix multiply, softmax, optionally masking and dropout, matrix multiply), then write the result back to HBM

Flash Attention

CUDA Kernel

- A kernel is a function that runs in parallel on multiple threads, with each thread executing the function independently on its own portion of the data.
- Consider the example of adding two vectors

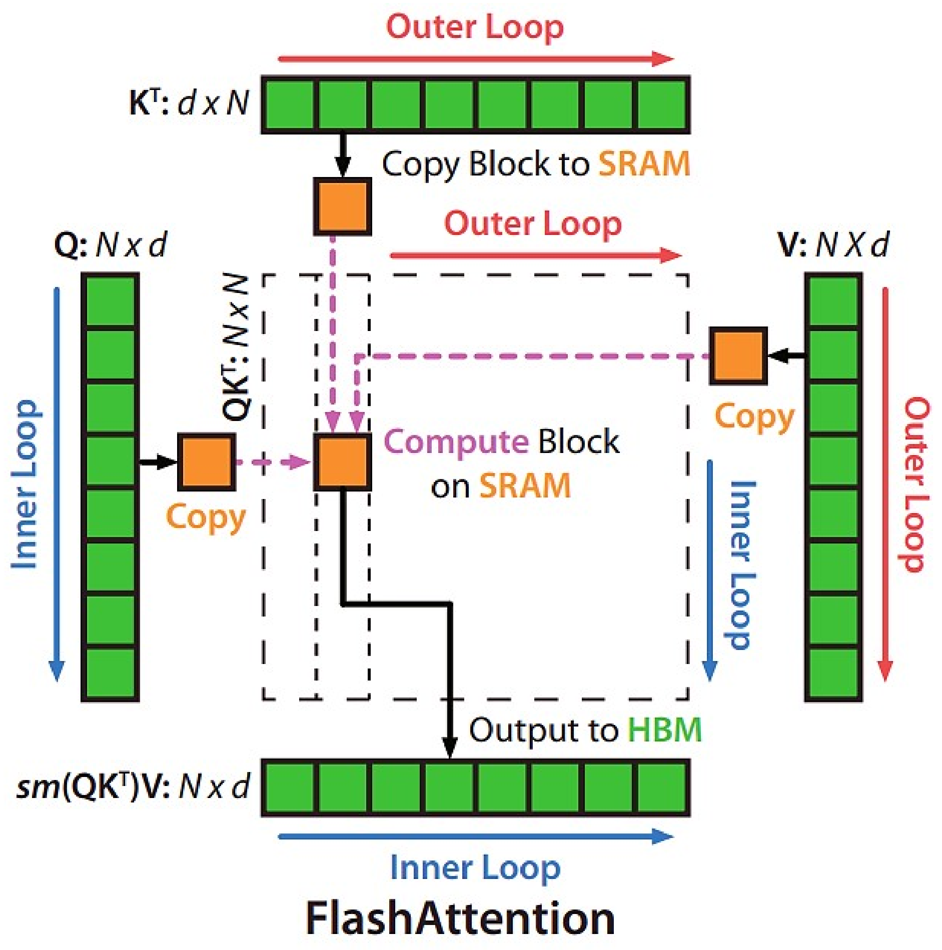
$$\begin{bmatrix}1\\2\\3\end{bmatrix}+\begin{bmatrix}4\\5\\6\end{bmatrix}=\begin{bmatrix}5\\7\\9\end{bmatrix}$$



```
def kernel_add(a,b):
    return a+b
```

Flash Attention

- Tiling: Attention Computation by blocks



Flash Attention

Attention Computation by blocks

$$Q = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 5 & 6 & 5 & 6 \\ 7 & 8 & 7 & 8 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

- Divide the matrices into blocks

$$Q = \begin{array}{cccc} \multicolumn{2}{c}{1} & \multicolumn{2}{c}{2} \\ \hline \boxed{\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}} & & \boxed{\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}} & \\ \boxed{\begin{matrix} 5 & 6 \\ 7 & 8 \end{matrix}} & & \boxed{\begin{matrix} 5 & 6 \\ 7 & 8 \end{matrix}} & \\ \hline \multicolumn{2}{c}{3} & \multicolumn{2}{c}{4} \end{array}, \qquad K = \begin{array}{cccc} \multicolumn{2}{c}{1} & \multicolumn{2}{c}{2} \\ \hline \boxed{\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}} & & \boxed{\begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}} & \\ \boxed{\begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}} & & \boxed{\begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix}} & \\ \hline \multicolumn{2}{c}{3} & \multicolumn{2}{c}{4} \end{array}$$

$$V = \begin{array}{cccc} \multicolumn{2}{c}{1} & \multicolumn{2}{c}{2} \\ \hline \boxed{\begin{matrix} 1 & 2 \\ 5 & 6 \end{matrix}} & & \boxed{\begin{matrix} 3 & 4 \\ 7 & 8 \end{matrix}} & \\ \boxed{\begin{matrix} 9 & 10 \\ 13 & 14 \end{matrix}} & & \boxed{\begin{matrix} 11 & 12 \\ 15 & 16 \end{matrix}} & \\ \hline \multicolumn{2}{c}{3} & \multicolumn{2}{c}{4} \end{array}$$

Flash Attention

Attention Computation by blocks

- For each K, V block iterate over all blocks of Q computing

$$S_{block} = Q_{block} K_{block}^T$$

- Example

  For $K_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $V_1 = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$:

  initialize $m_1 = -\infty$, $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix

  Iterate over all Q_i

  compute $S_{1,i} = Q_i K_1^T$

Flash Attention

Attention Computation by blocks

For $K_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $V_1 = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$:

initialize $m_1 = -\infty$, $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix

Iterate over all Q_i

compute $S_{1,i} = Q_i K_1^T$

$$S_{1,1} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

- For K_1 and V_1:
  - initialize $m_1 = -\infty$, $l_1 = 0,\ O_1 = 2 \times 2$ zero matrix
  - Iterate over all Q_i
    - compute $S_{1,i} = Q_i K_1^T$
    - compute row max $m_{1,i}$

$$S_{1,1} = \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \cdot \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}3 & 3\\7 & 7\end{bmatrix} \begin{array}{l} \rightarrow \max(3,3) = 3 \\ \rightarrow \max(7,7) = 7 \end{array}$$

$$m_{1,1} = \begin{bmatrix}3\\7\end{bmatrix}$$

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$ , $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,\mathrm{i}} = \exp(S_{1,i} - m_{1,i})$
```

$$\exp(S_{1,1} - m_{1,1}) = \begin{bmatrix} e^{3-3} & e^{3-3} \\ e^{7-7} & e^{7-7} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$ , $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,\mathrm{i}} = \exp\left(S_{1,i} - m_{1,i}\right)$
        compute denominator for softmax: $l_{1,i}$ by summing row entries of $P_{1,i}$
```

$$\exp\left(S_{1,1} - m_{1,1}\right) = \begin{bmatrix} e^{3-3} & e^{3-3} \\ e^{7-7} & e^{7-7} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Summing across rows:

$$l_{1,1} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$ , $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,\mathrm{i}} = \exp\left(S_{1,i} - m_{1,i}\right)$
        compute denominator for softmax: $l_{1,i}$ by summing row entries of $P_{1,i}$
        Compute partial output $O_{1,i} = P_{1,i} \cdot V_1$
```

$$O_{1,1} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 6 & 8 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$ , $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,\mathrm{i}} = \exp\left(S_{1,i} - m_{1,i}\right)$
        compute denominator for softmax: $l_{1,i}$ by summing row entries of $P_{1,i}$
        Compute partial output $O_{1,i} = P_{1,i} \cdot V_1$
        update global max value statistic $m_1$
```

$$m_1^{new} = \max\left(m_1, m_{1,1}\right) = \max\left(m_1, \begin{bmatrix} 3 \\ 7 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$$

Note that max is taken element wise. So for –infty and a vector the max is the vector. If the comparison is between a vector and a vector we take the max entry in each row. So max([2, 7], [3, 6]) = [3, 7]

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$, $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,i} = \exp(S_{1,i} - m_{1,i})$
        compute denominator for softmax: $l_{1,i}$ by summing row entries of $P_{1,i}$
        Compute partial output $O_{1,i} = P_{1,i} \cdot V_1$
        update global max value statistic $m_1$
        update softmax normalization factor $l_1$
```

$$l_1^{new} = e^{m_1 - m_1^{new}} l_1 + e^{m_{1,1} - m_1^{new}} l_{1,1}$$

$$m_1 - m_1^{new} = -\infty - \begin{bmatrix} 3 \\ 7 \end{bmatrix} = -\infty$$

$$e^{-\infty} = 0, \qquad l_1 = 0$$

$$l_1^{new} = e^{m_1 - m_1^{new}} l_{1,1} = e^{\begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 3 \\ 7 \end{bmatrix}} \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = e^{\begin{bmatrix} 0 \\ 0 \end{bmatrix}} \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

```
For $K_1$ and $V_1$:
    initialize $m_1 = -\infty$, $l_1 = 0$, $O_1 = 2 \times 2$ zero matrix
    Iterate over all $Q_i$
        compute $S_{1,i} = Q_i K_1^T$
        compute row max $m_{1,i}$
        compute numerator for softmax: $P_{1,i} = \exp(S_{1,i} - m_{1,i})$
        compute denominator for softmax: $l_{1,i}$ by summing row entries of $P_{1,i}$
        Compute partial output $O_{1,i} = P_{1,i} \cdot V_1$
        update global max value statistic $m_1$
        update softmax normalization factor $l_1$
        update output $O_1$ before $Q_{i+1}$ to accumulate across all query blocks. The division with $l_1$ applies softmax implicitly.
```

$$\boxed{O_1^{new} = \frac{1}{l_1^{new}} \left(e^{m_1 - m_1^{new}} O_1 + e^{m_{1,1} - m_1^{new}} O_{1,1} \right)}$$

Simplified

$$O_1^{new} = \frac{1}{\begin{bmatrix} 2 \\ 2 \end{bmatrix}} \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 6 & 8 \\ 6 & 8 \end{bmatrix} \right) = \frac{\begin{pmatrix} 1 \times 6 & 1 \times 8 \\ 1 \times 6 & 1 \times 8 \end{pmatrix}}{\begin{pmatrix} 2 \\ 2 \end{pmatrix}} = \begin{bmatrix} 6/2 & 8/2 \\ 6/2 & 8/2 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 3 & 4 \end{bmatrix}$$

Note that multiplication by $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and division by $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ are row wise scaling operations

Flash Attention

Attention Computation by blocks

The actual computation to update O_i involves taking the inverse of the diagonal matrix constructed from l_i^{new} and l_i.

$$O_1^{new} = \mathrm{diag}(l_1^{new})^{-1}\,(\mathrm{diag}(l_i)e^{m_1 - m_1^{new}}O_1 + e^{m_{1,1} - m_1^{new}}O_{1,1})$$

$$\mathrm{diag}(l_1^{new}) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \mathrm{diag}(l_1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Inverse of a diagonal matrix

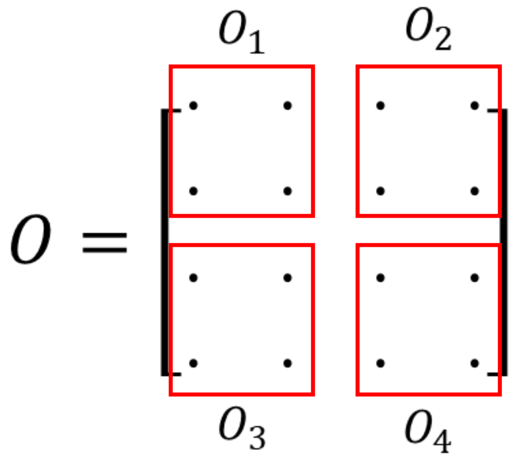
$$D = \begin{bmatrix} d_1 & 0 & 0 & \dots & 0 \\ 0 & d_2 & 0 & \dots & 0 \\ 0 & 0 & d_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_n \end{bmatrix} \qquad D^{-1} = \begin{bmatrix} 1/d_1 & 0 & 0 & \dots & 0 \\ 0 & 1/d_2 & 0 & \dots & 0 \\ 0 & 0 & 1/d_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1/d_n \end{bmatrix}$$

Flash Attention

Attention Computation by blocks

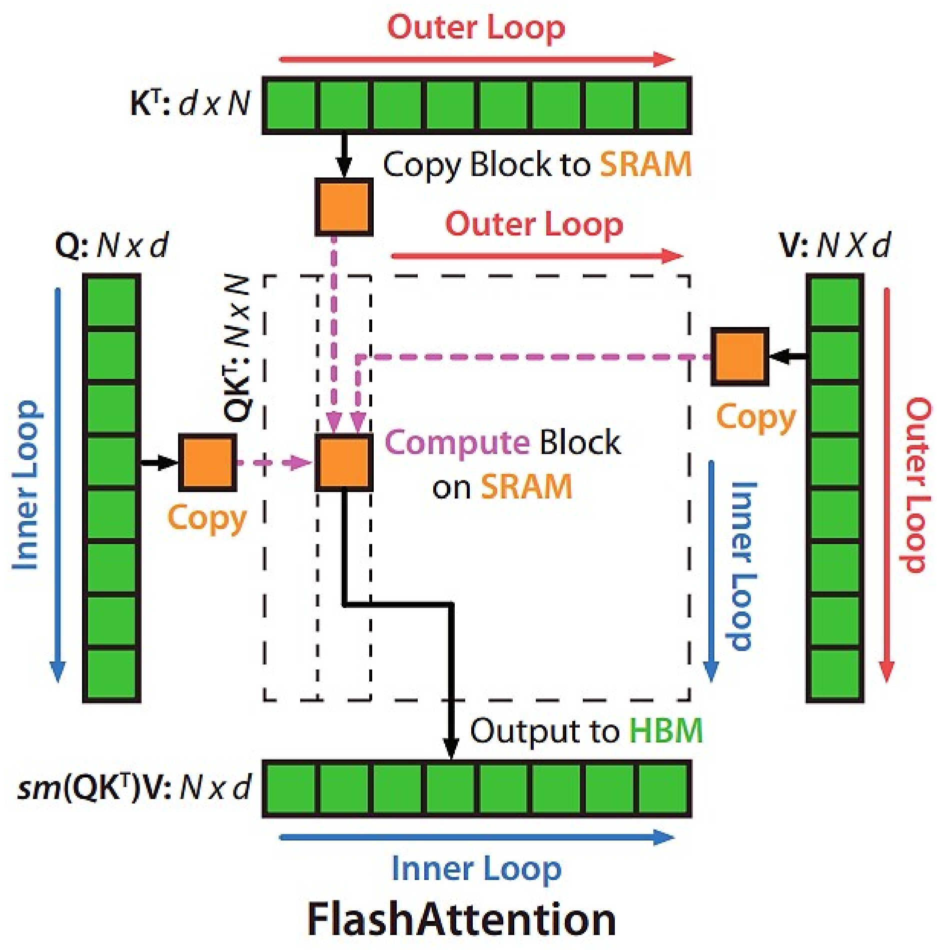
- We similarly update O_1 in each iteration of the inner loop. At the end of the inner loop we have the block 1 entries of the final attention output matrix O. This O_1 is now written to HBM

- The final attention output O will have four blocks.



Flash Attention

- Tiling: Attention Computation by blocks

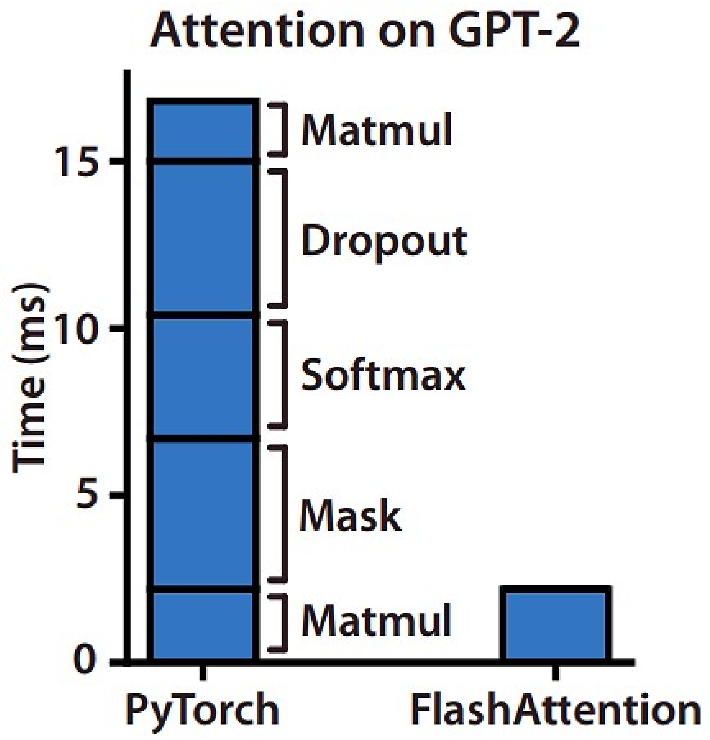


```
For all $K_j$ and $V_j$:
    initialize $m_j = -\infty$ , $l_j = 0$, $O_j =$ zero matrix
    For all $Q_i$:
        load $m_j$, $l_j$ from HBM
        compute $S_{j,i} = Q_i K_j^T$
        compute row max $m_{j,i}$
        $P_{j,i} = \exp(S_{j,i} - m_{j,i})$
        $l_{j,i}$ by summing row entries of $P_{j,i}$
        Compute partial output $O_{j,i} = P_{j,i} \cdot V_j$
        update global max value statistic $m_j$
        update softmax normalization factor $l_j$
        update output $O_j$
        write $m_j$, $l_j$ to HBM
    Output $O_j$ to HBM
```

Flash Attention

Speedup over the PyTorch implementation of attention on GPT-2.

- FlashAttention does not read and write the large $N \times N$ attention matrix to HBM, resulting in an 7.6X speedup on the attention computation.



Mixture of Experts

- We have explored how advanced attention mechanisms enhance efficiency. However, the majority of parameters in a Large Language Model (LLM) reside in the Feed-Forward Network (FFN) layers.

Mixture of Experts

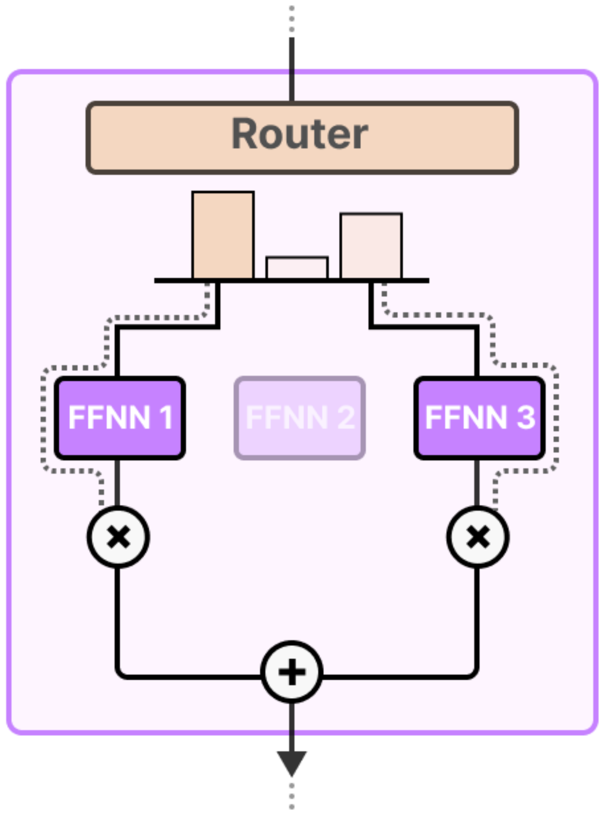
- We have explored how advanced attention mechanisms enhance efficiency. However, the majority of parameters in a Large Language Model (LLM) reside in the Feed-Forward Network (FFN) layers.
- Increasing the number of parameters in a model improves performance by capturing more complex patterns.
- More parameters, however, lead to higher computational costs, longer training times, and slower inference.

Mixture of Experts

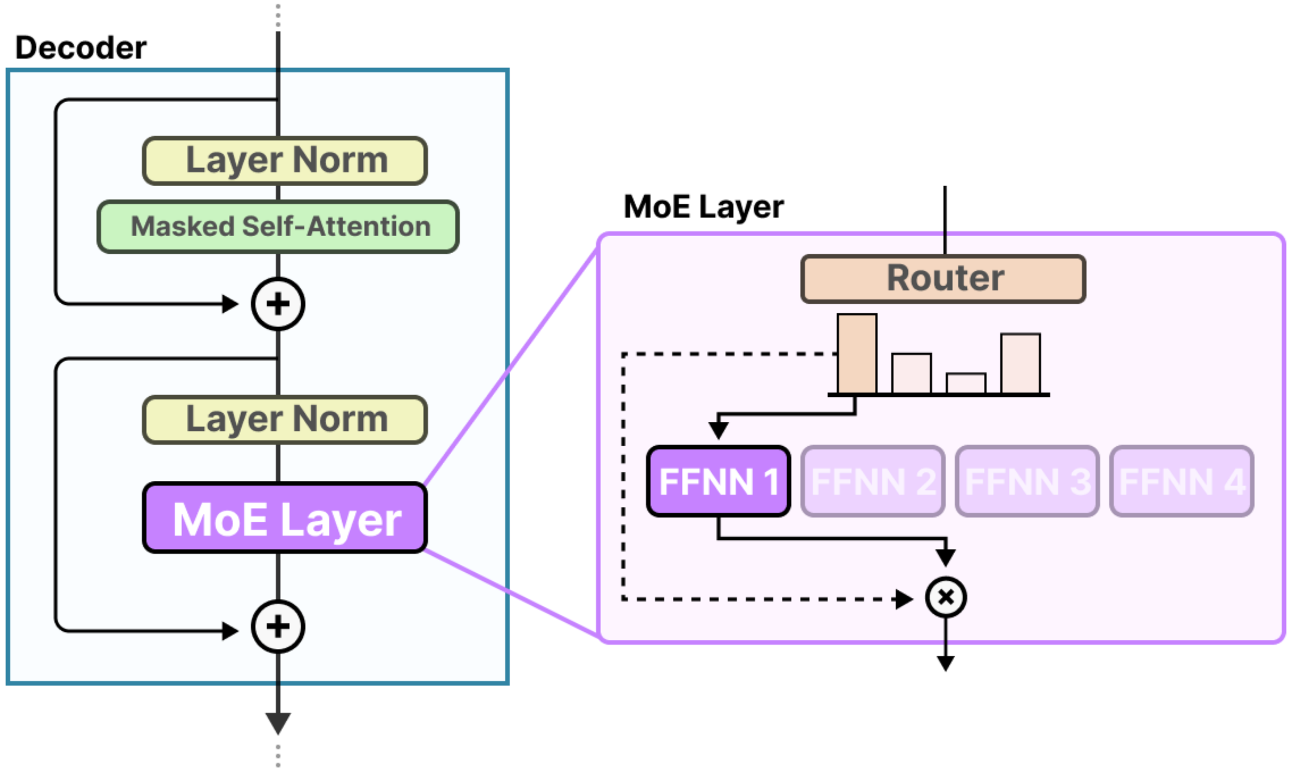
- How can we increase parameters without higher computational cost and training time?
  - The Mixture-of-Expert (MoE) structure (Jacobs et al., 1991; Jordan and Jacobs, 1994) is a classic design that substantially scales up the model capacity and only introduces small computation overhead.
  - Mixture of Experts (MoE) increases model capacity while maintaining efficiency by using sparse activation of subnetworks.
  - MoE consists of multiple expert networks, but only a subset is activated per input, reducing computation.

Mixture of Experts

- MoE layer contains many experts (non linear neural networks) that share the same network architecture and are trained by the same algorithm, with a gating (or routing) network that routes individual inputs to a few experts among all the candidates.



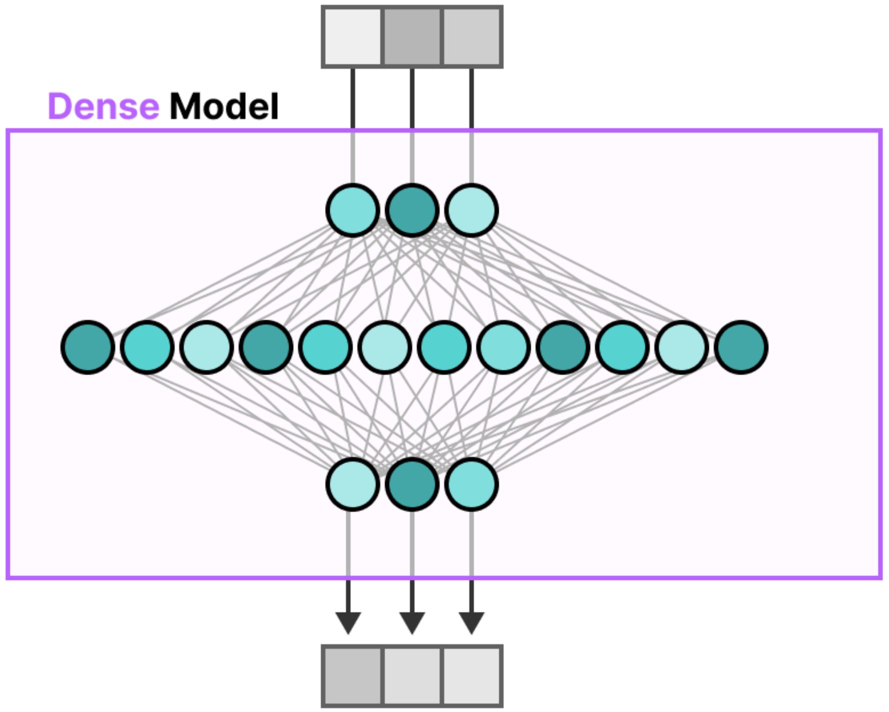
MoE layer



Mixture of Experts

Sparse Layers

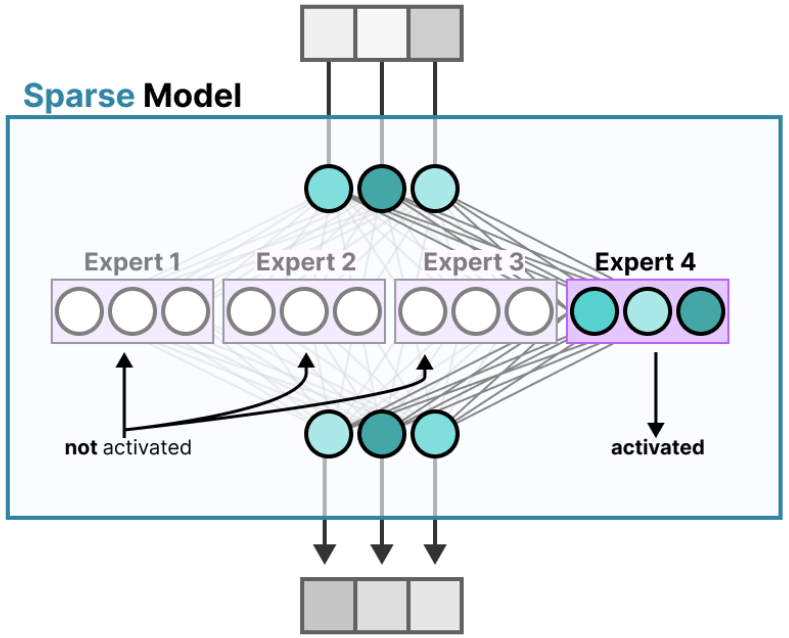
- The FFNN in a traditional Transformer is called a dense model since all parameters (its weights and biases) are activated. Nothing is left behind and everything is used to calculate the output.
- The input activates all parameters to some degree:



Mixture of Experts

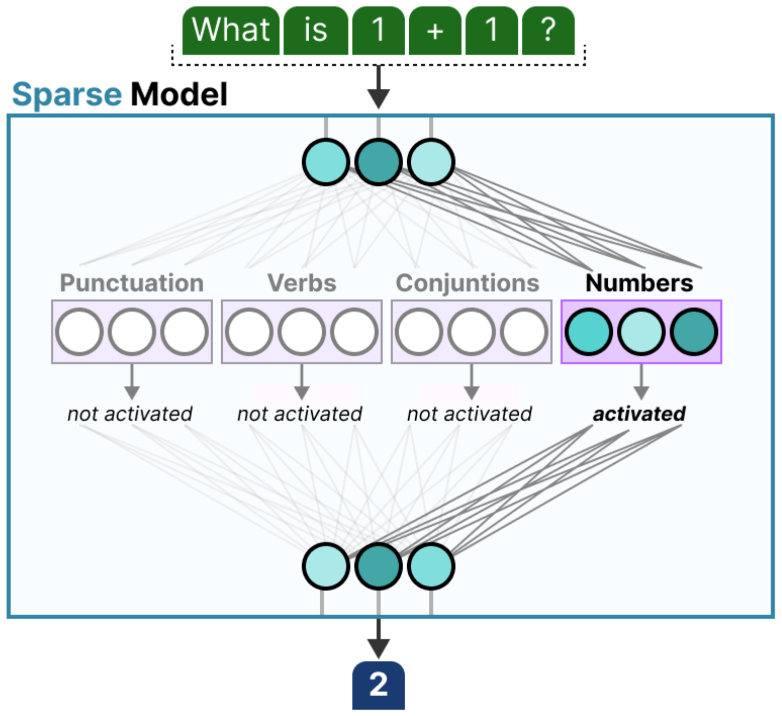
Sparse Layers

- Sparse models only activate a portion of their total parameters and are closely related to Mixture of Experts.
- In MoE, we can chop up our dense model into pieces (so-called experts), retrain it, and only activate a subset of experts at a given time:



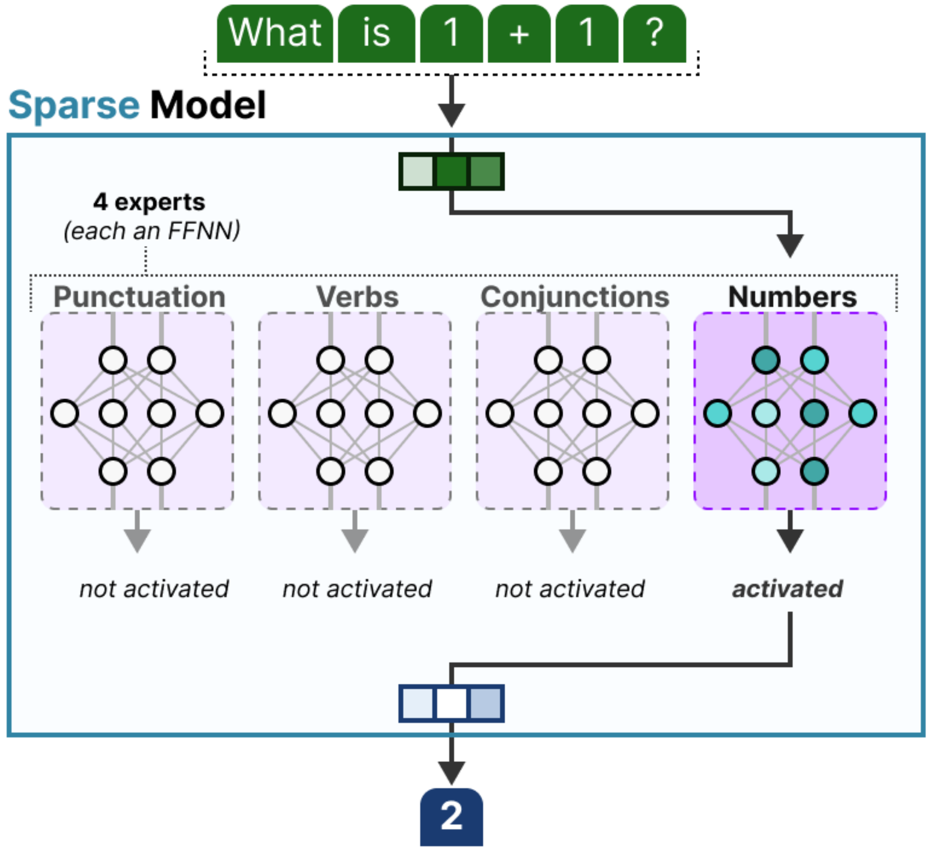
Mixture of Experts

- The underlying idea is that each expert learns different information during training. Then, when running inference, only specific experts are used as they are most relevant for a given task.
- When asked a question, we can select the expert best suited for a given task:



Mixture of Experts

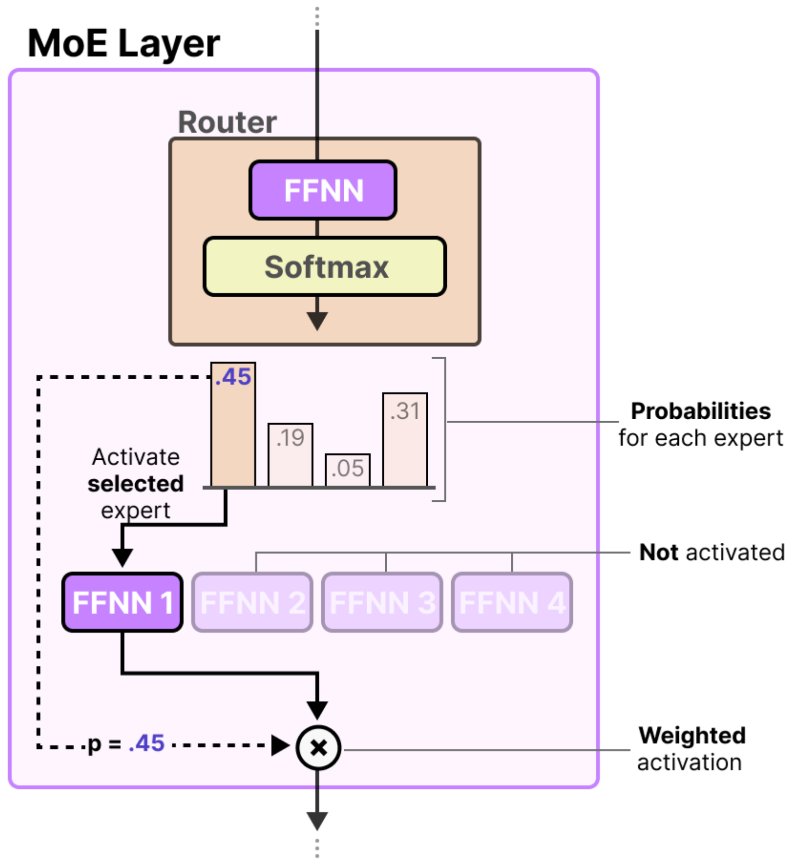
- All Experts in the MoE layer are FFNNs themselves. All experts have the same structure, initialized from the same weight distribution and are trained with the same optimization configuration.



Mixture of Experts

The Routing Mechanism

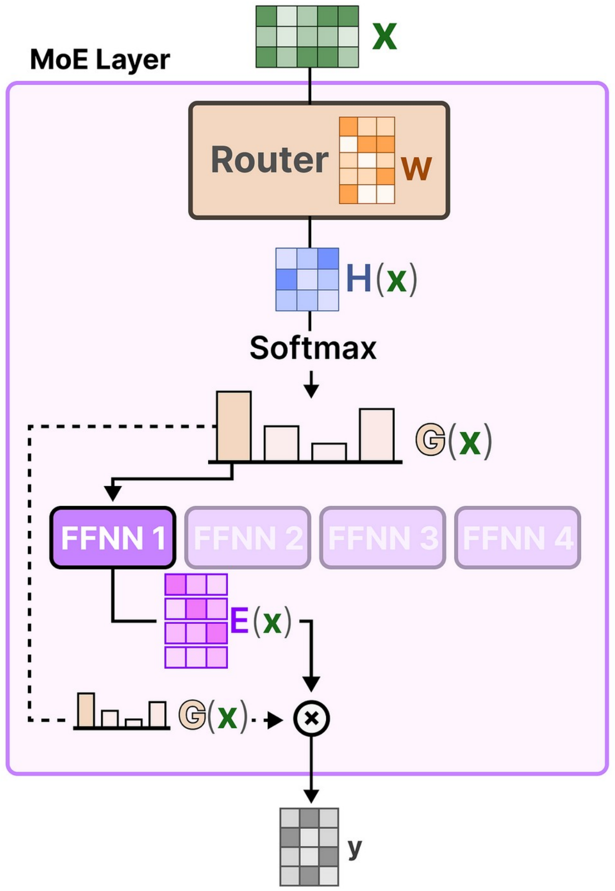
- The router (or gate network) is also an FFNN and is used to choose the expert based on a particular input. It outputs probabilities which it uses to select the best matching expert:

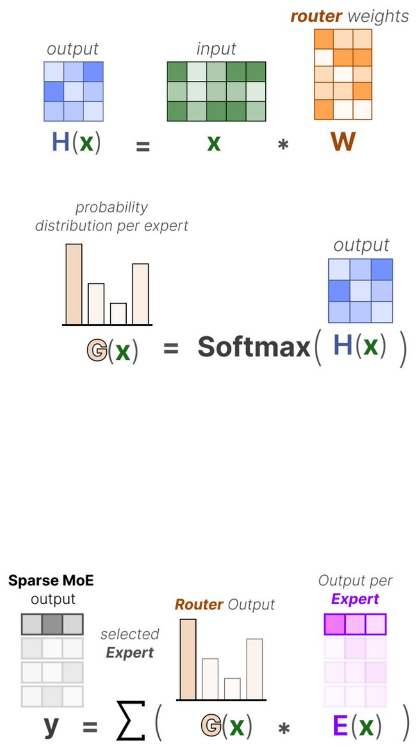


Mixture of Experts

The Routing Mechanism

- The gating network is arguably the most important component of any MoE as it not only decides which experts to choose during inference but also training.

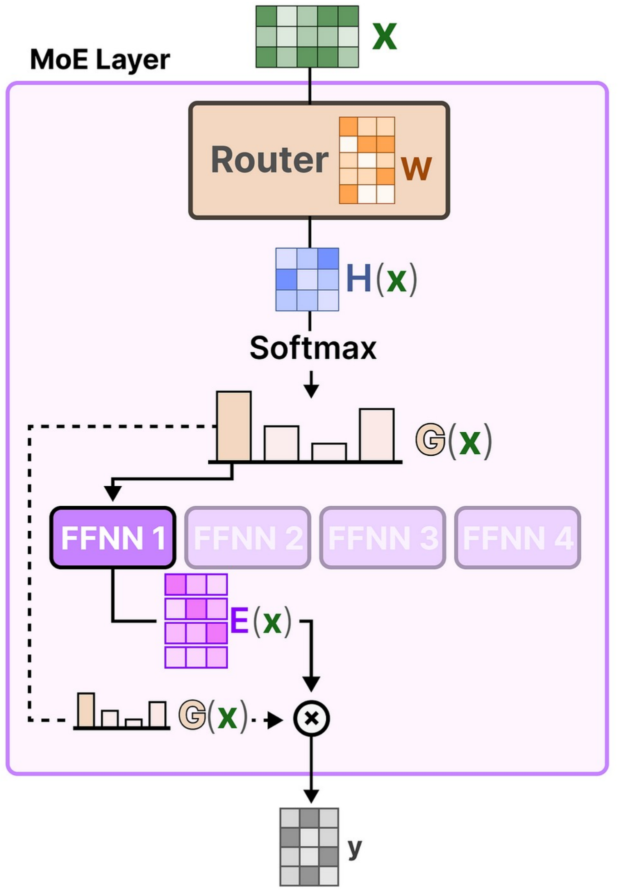


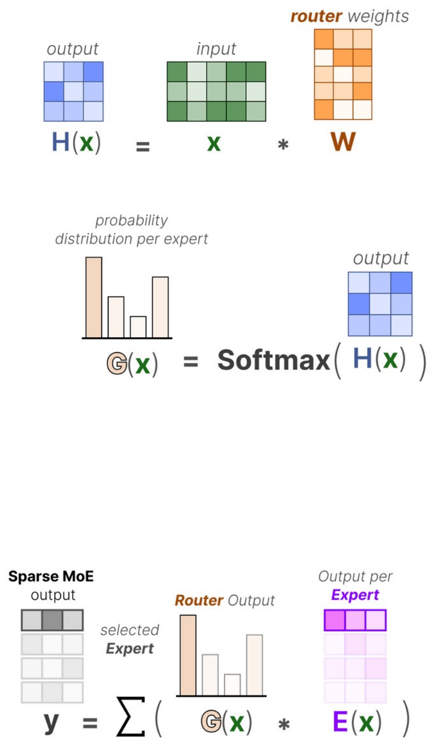


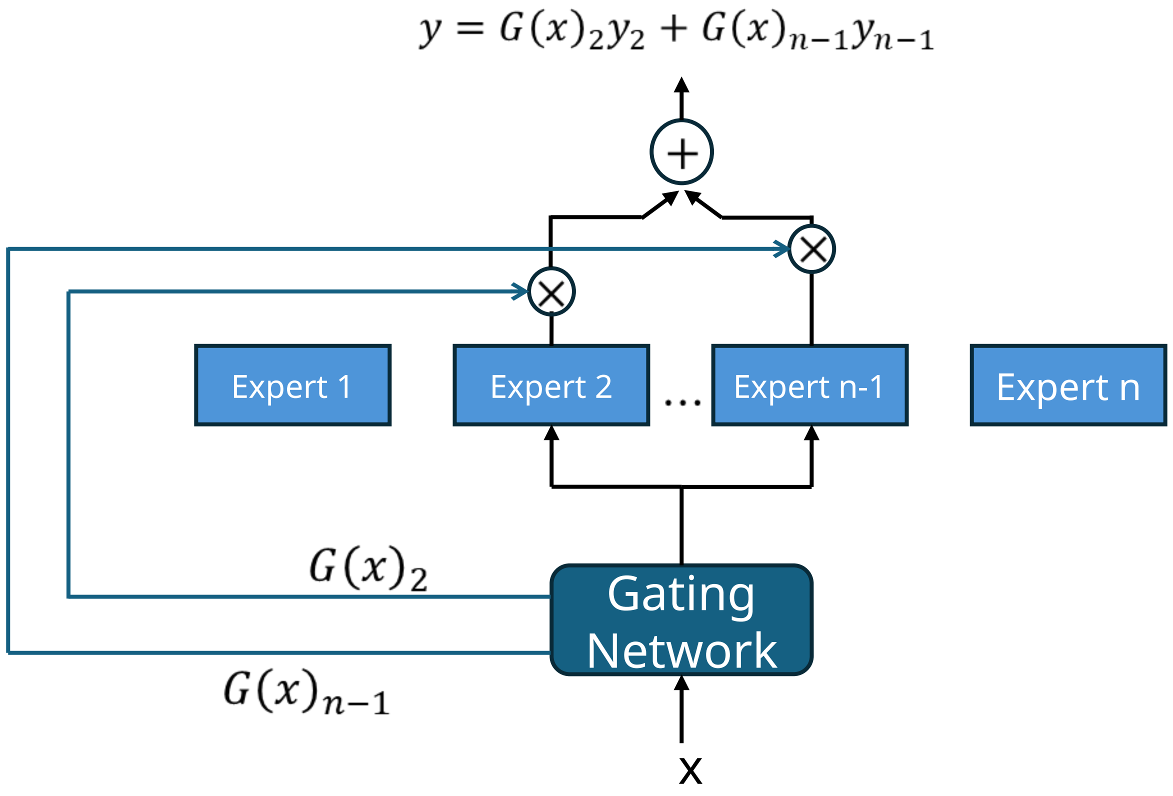
Mixture of Experts

The Routing Mechanism

- The gating network is arguably the most important component of any MoE as it not only decides which experts to choose during inference but also training.



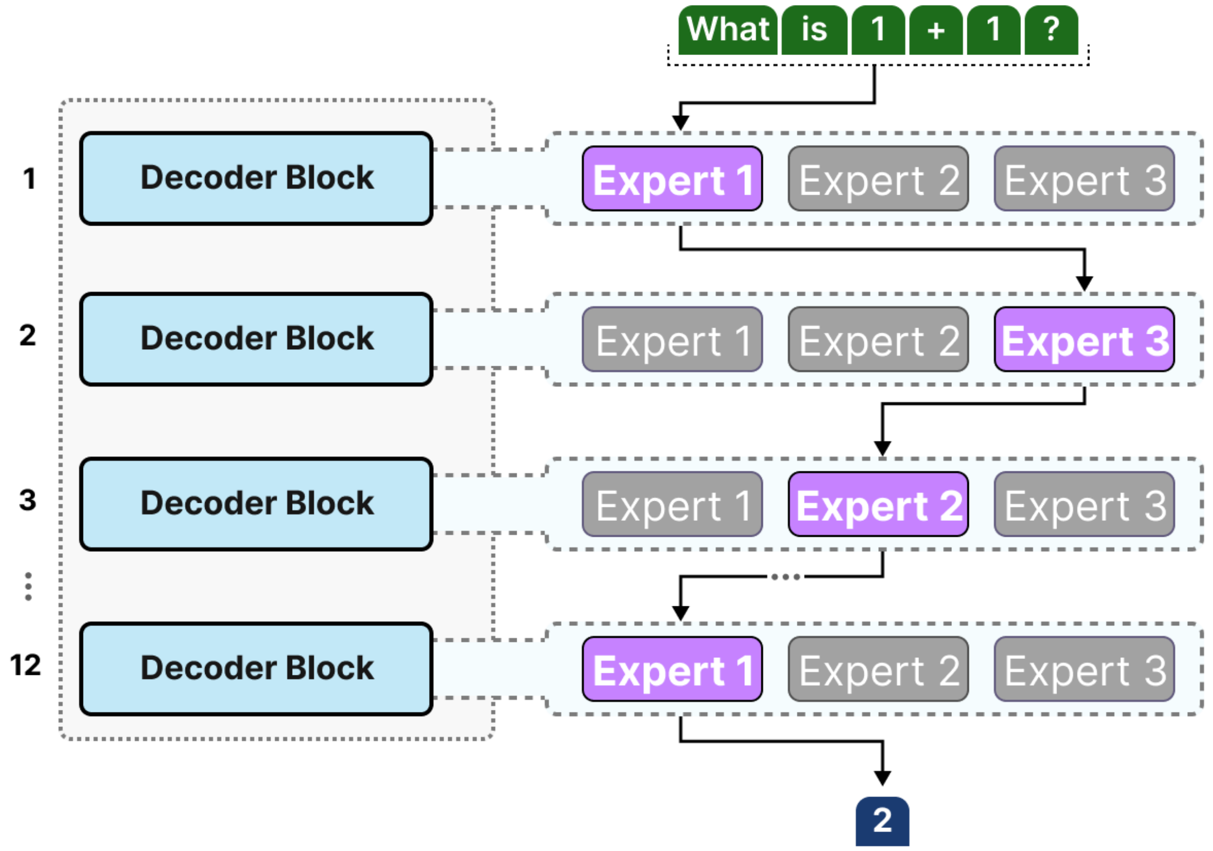




Mixture of Experts

The Routing Mechanism

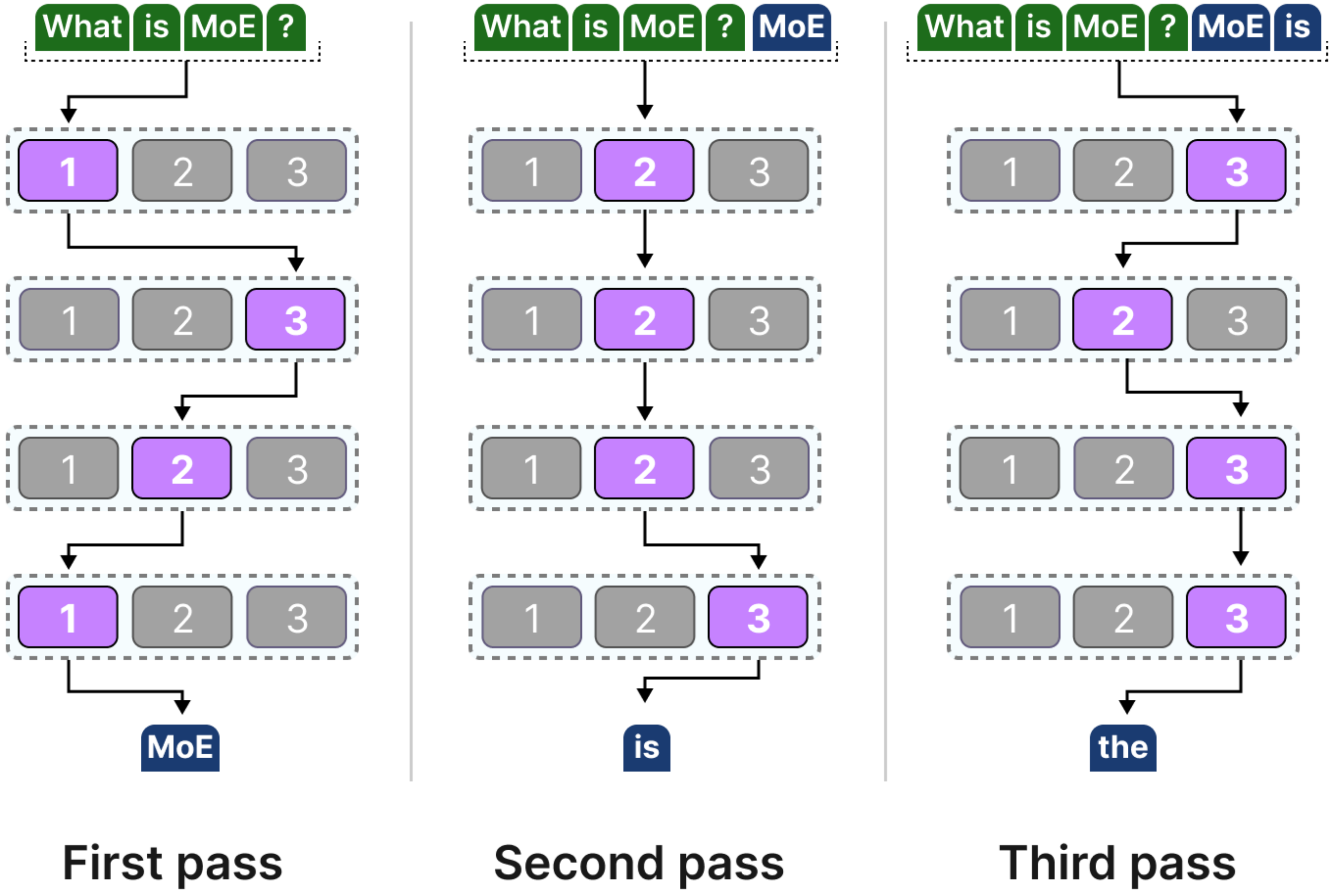
- Since most LLMs have several decoder blocks, a given text will pass through multiple experts before the text is generated:



Mixture of Experts

The Routing Mechanism

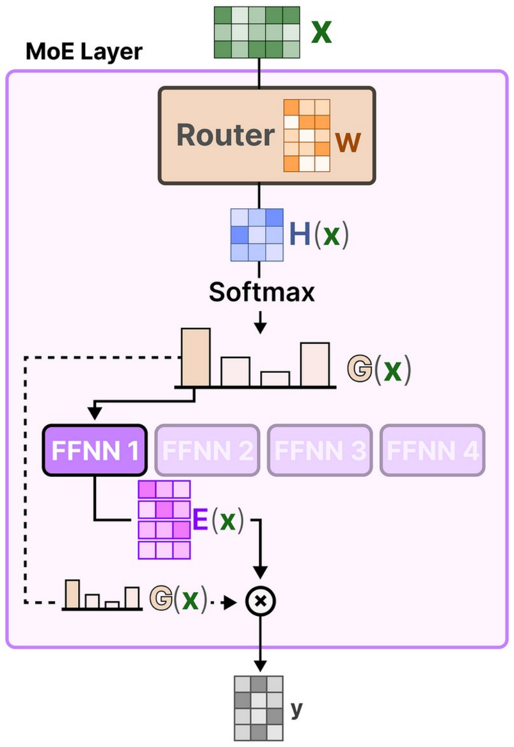
- The chosen experts likely differ between tokens which results in different "paths" being taken:

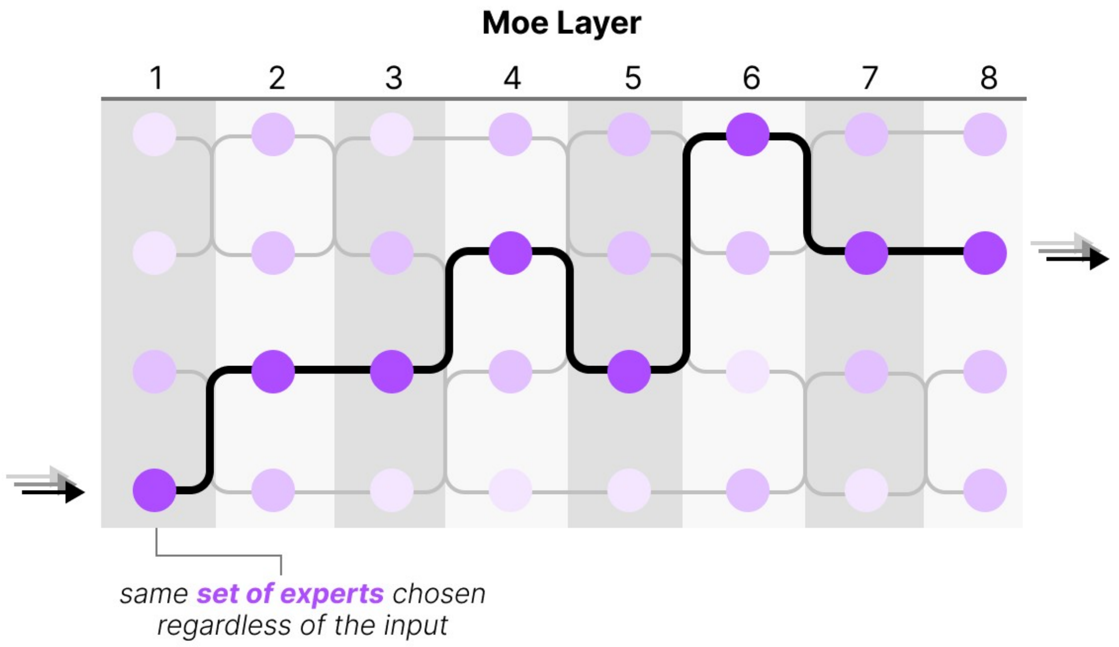


Mixture of Experts

Load Balancing

- The simple routing function often results in the router choosing the same expert since certain experts might learn faster than others Not only will there be an uneven distribution of experts chosen, but some experts will hardly be trained at all. This results in issues during both training and inference.

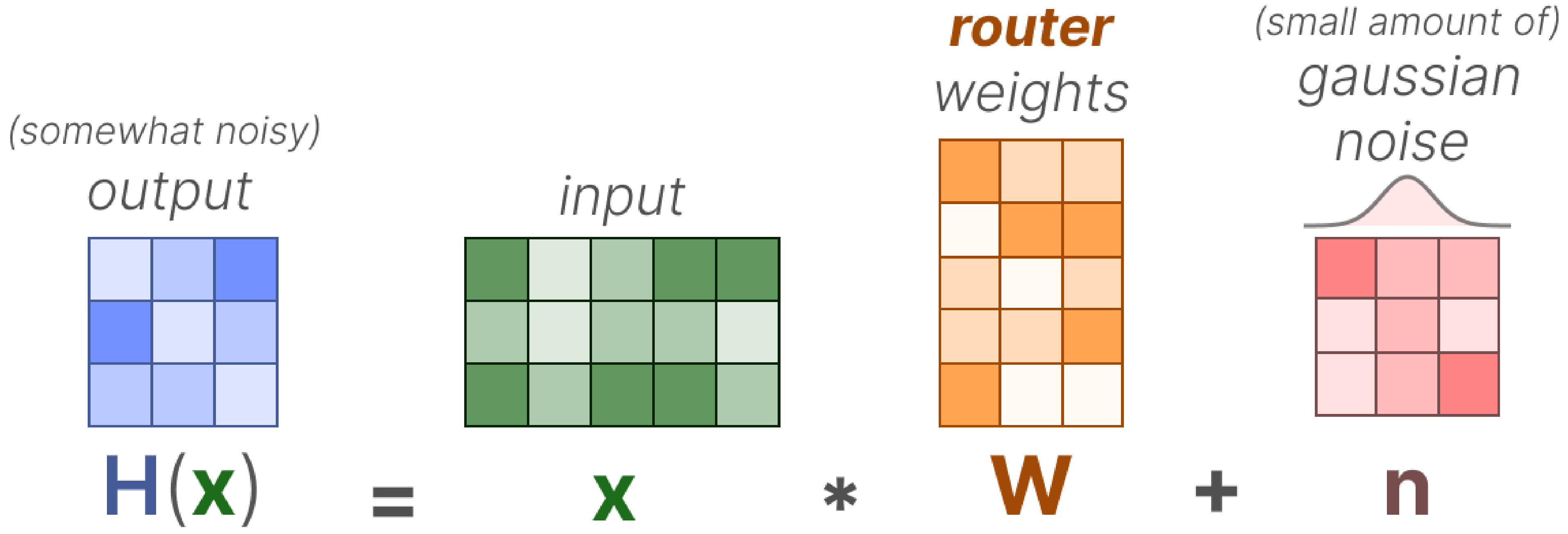




Mixture of Experts

Load Balancing

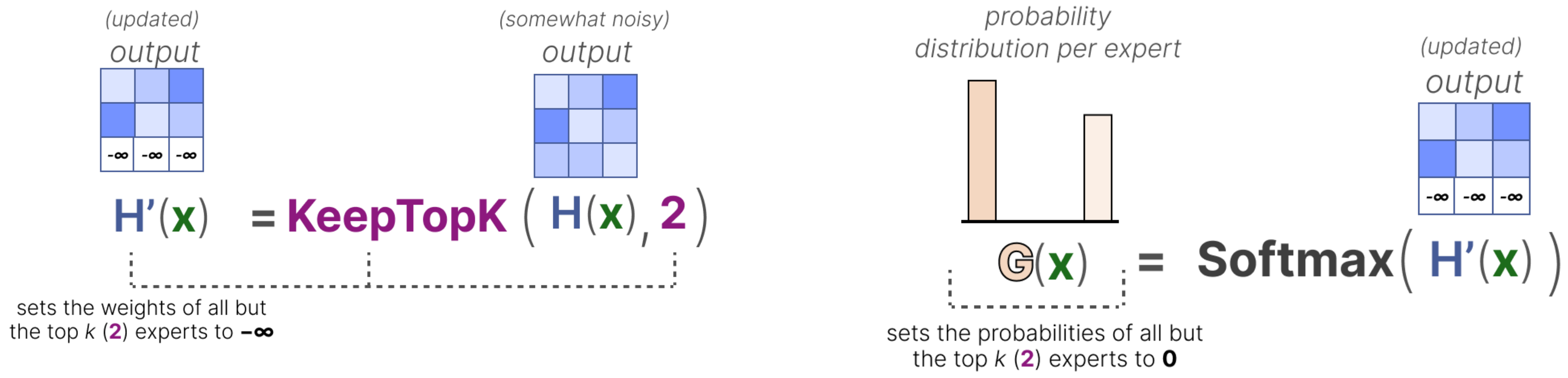
- To balance the importance of experts, we will need to look at the router as it is the main component to decide which experts to choose at a given time. One method of load balancing the router is through a straightforward extension called KeepTopK2. By introducing trainable (gaussian) noise, we can prevent the same experts from always being picked:



Mixture of Experts

Load Balancing

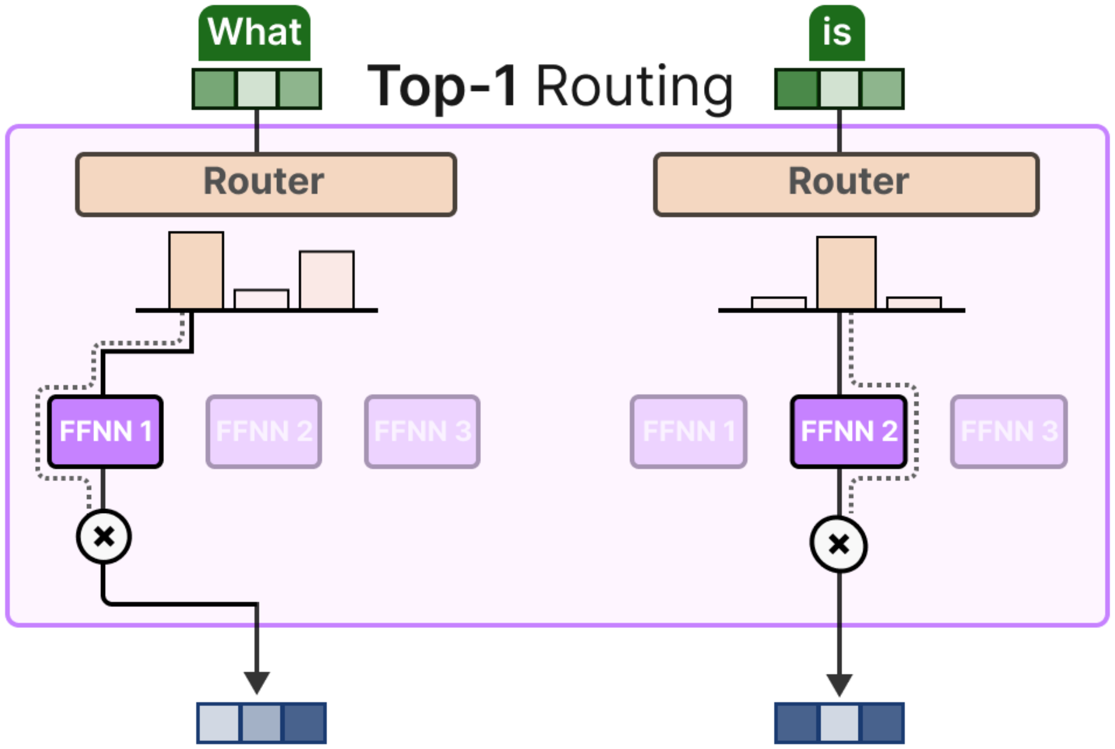
- Then, all but the top k experts that you want activating (for example 2) will have their weights set to $-\infty$. By setting these weights to $-\infty$, the output of the SoftMax on these weights will result in a probability of 0:

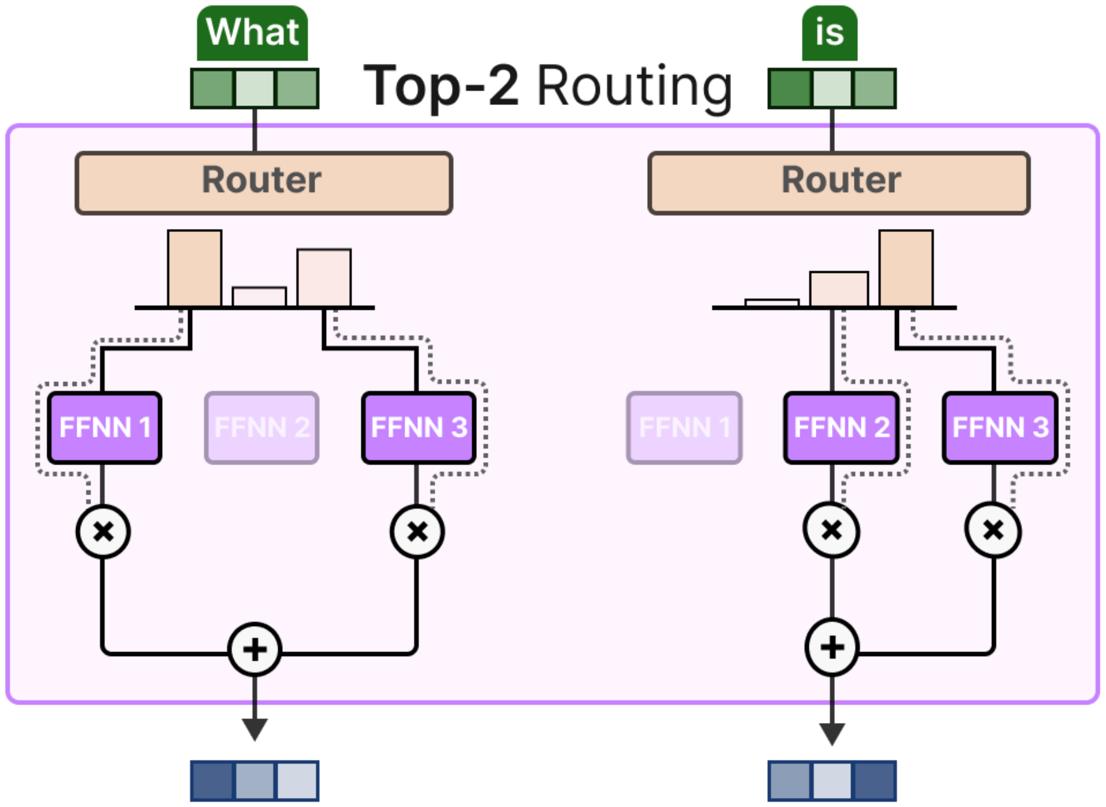

(updated)
output
-∞ -∞ -∞
(somewhat noisy)
output
H'(x) = KeepTopK (H(x), 2)
sets the weights of all but the top k (2) experts to −∞
probability distribution per expert
(updated)
output
-∞ -∞ -∞
G(x) = Softmax(H'(x))
sets the probabilities of all but the top k (2) experts to 0

Mixture of Experts

Load Balancing

- The KeepTopK strategy routes each token to a few selected experts. This method is called Token Choice3 and allows for a given token to be sent to one expert (top-1 routing) or to more than one expert (top-k routing):





Mixture of Experts

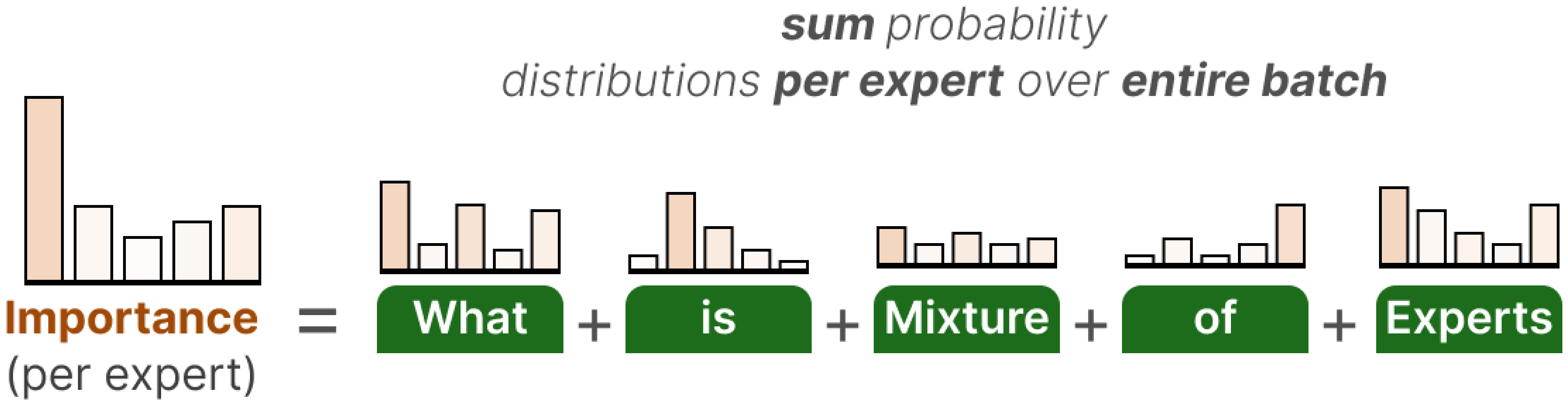
Auxiliary Loss

- To get a more even distribution of experts during training, the auxiliary loss (also called load balancing loss) was added to the network’s regular loss.
- It adds a constraint that forces experts to have equal importance.

Mixture of Experts

Auxiliary Loss

- The first component of this auxiliary loss is to sum the router values for each expert over the entire batch:



- This gives us the importance scores per expert which represents how likely a given expert will be chosen regardless of the input.

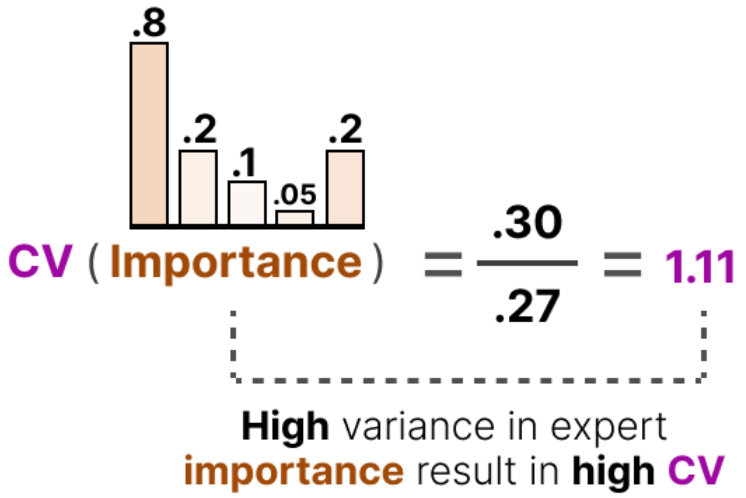
Mixture of Experts

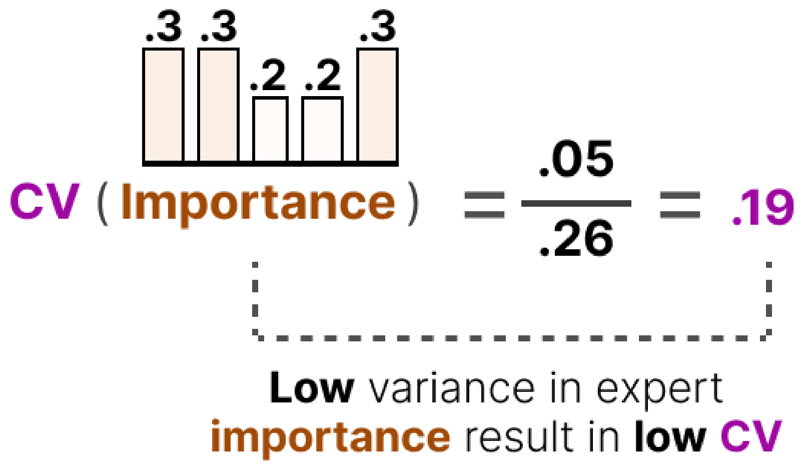
Auxiliary Loss

- We can use this to calculate the coefficient variation (CV), which tells us how different the importance scores are between experts.

$$\text{Coefficient Variation } (\mathbf{CV}) = \frac{\text{standard deviation } (\boldsymbol{\sigma})}{\text{mean } (\boldsymbol{\mu})}$$

- If there are a lot of differences in importance scores, the CV will be high. In contrast, if all experts have similar importance scores, the CV will be low (which is what we aim for):

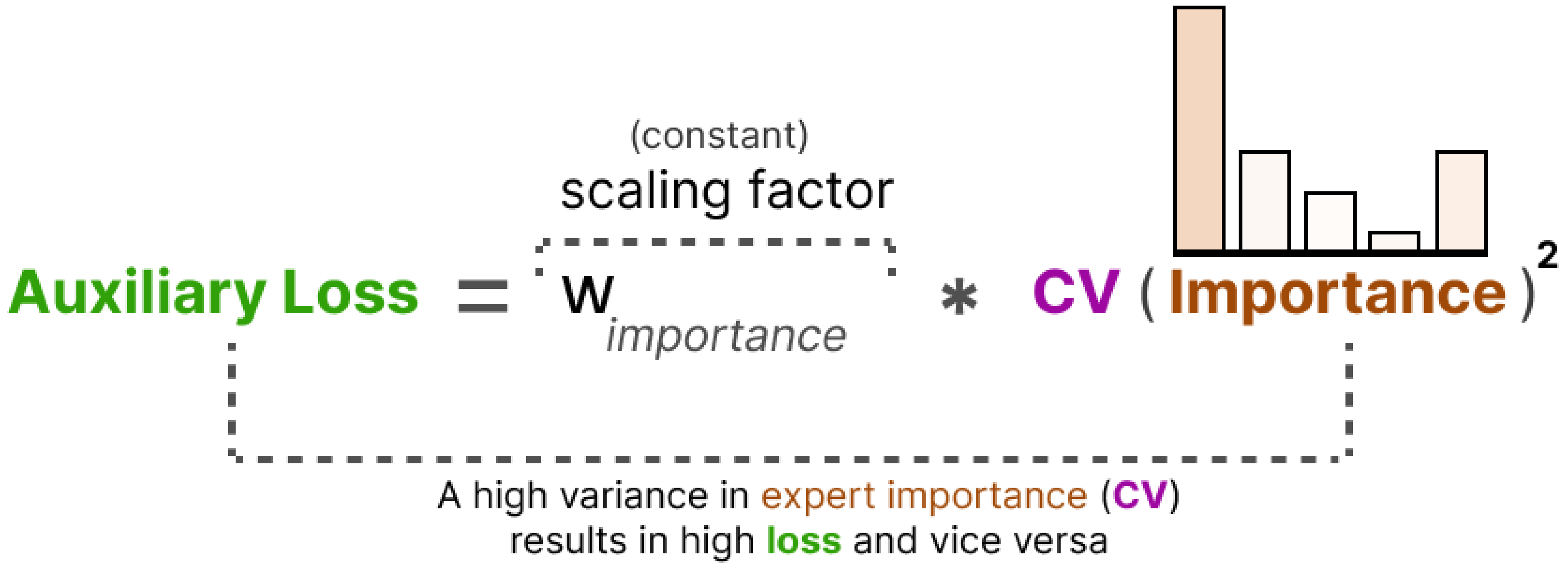




Mixture of Experts

Auxiliary Loss

- Using this CV score, we can update the auxiliary loss during training such that it aims to lower the CV score as much as possible (thereby giving equal importance to each expert):



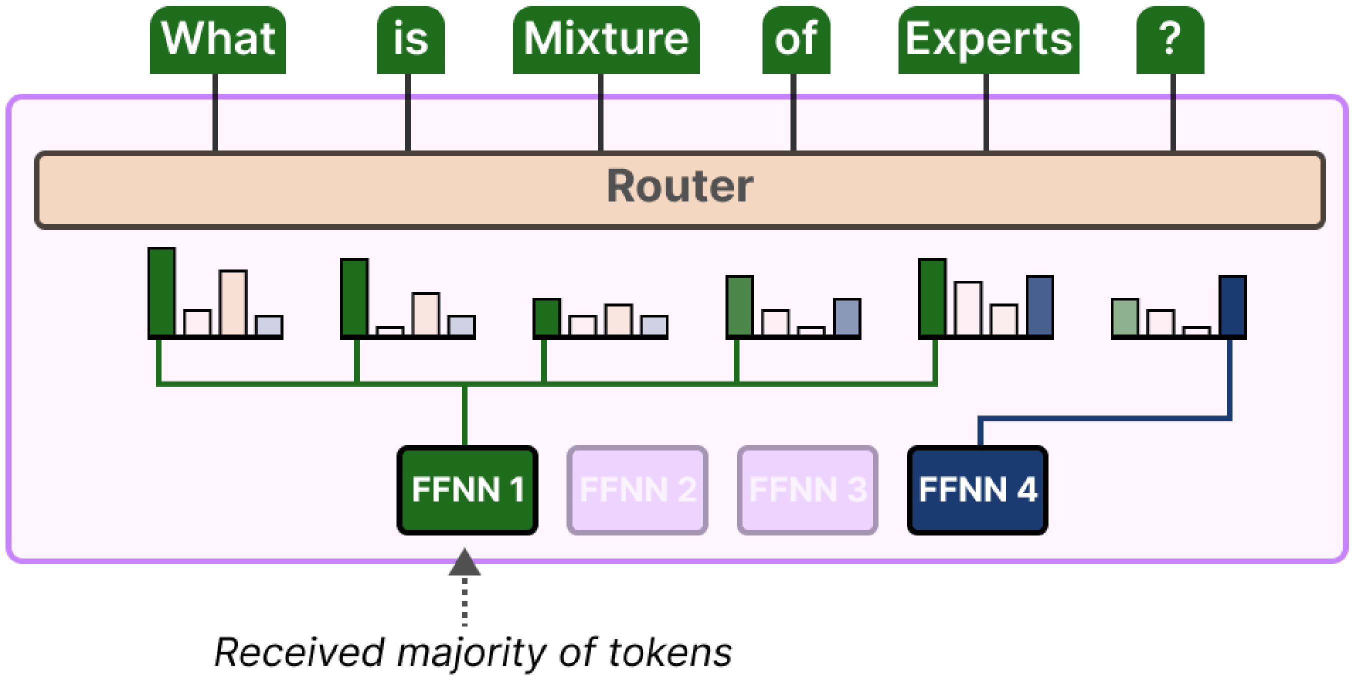
(constant)
scaling factor
Auxiliary Loss = w importance * CV (Importance)²
A high variance in expert importance (CV)
results in high loss and vice versa

- Finally, the auxiliary loss is added as a separate loss to optimize during training.

Mixture of Experts

Expert Capacity

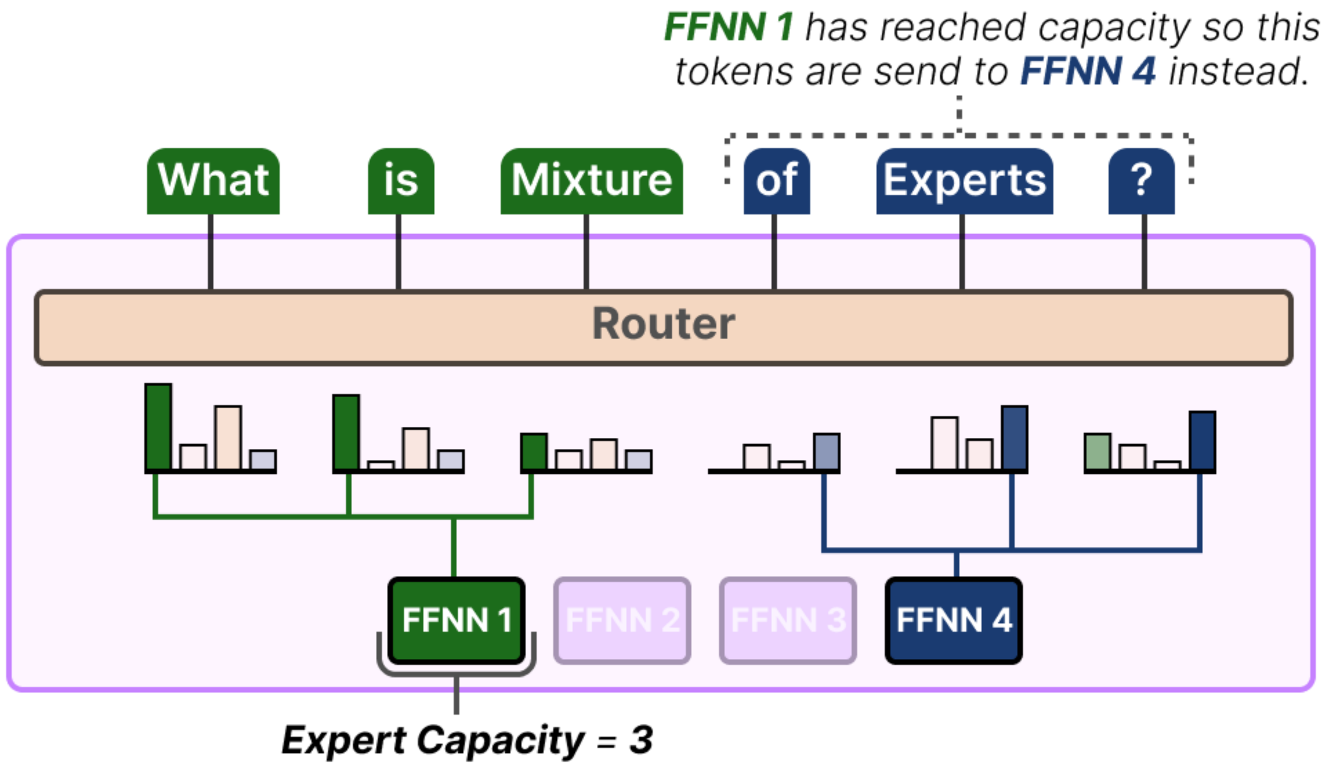
- Imbalance is not just found in the experts that were chosen but also in the distributions of tokens that are sent to the expert. For instance, if input tokens are disproportionally sent to one expert over another then that might also result in undertraining:



Mixture of Experts

Expert Capacity

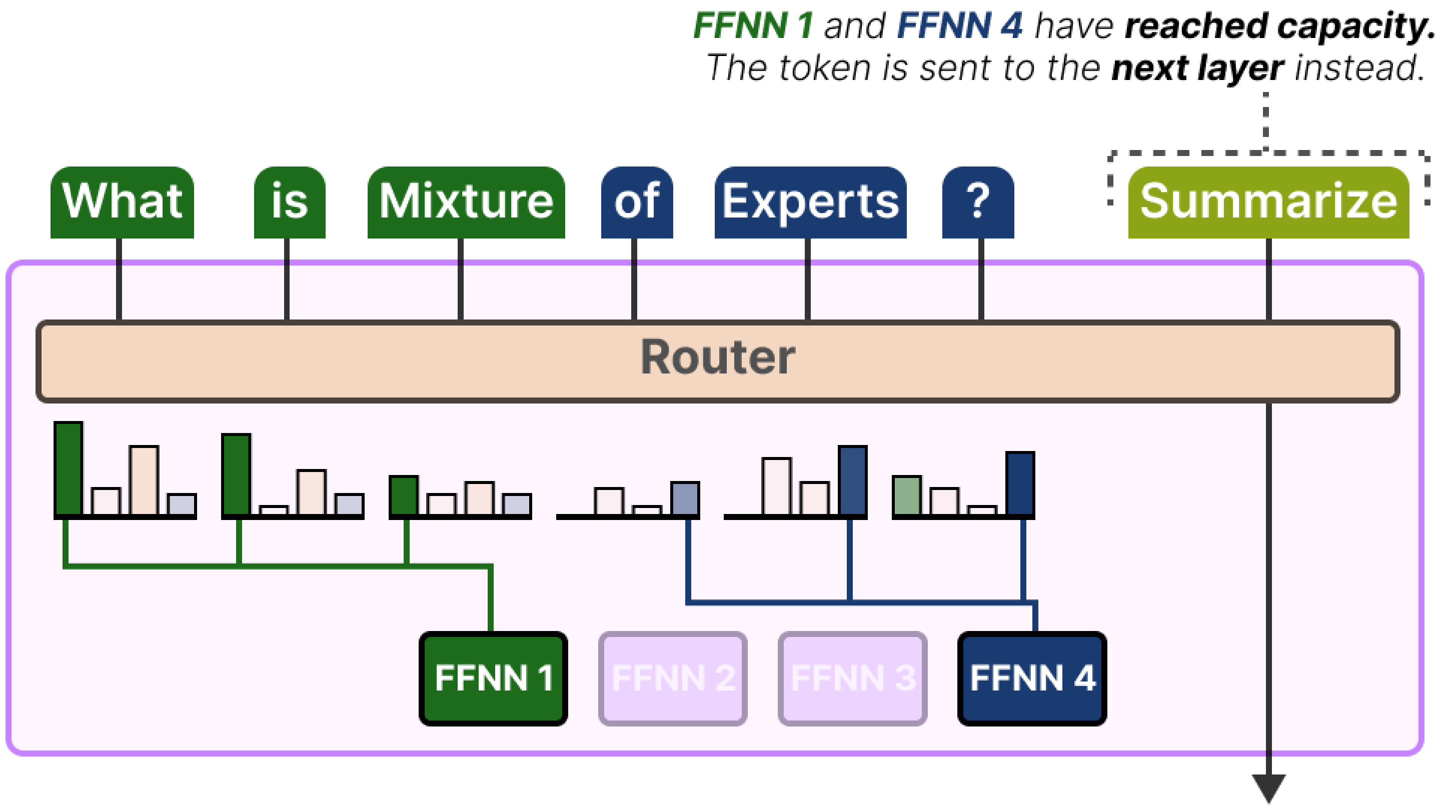
- A solution to this problem is to limit the amount of tokens a given expert can handle, namely Expert Capacity. By the time an expert has reached capacity, the resulting tokens will be sent to the next expert:



Mixture of Experts

Expert Capacity

- If both experts have reached their capacity, the token will not be processed by any expert but instead sent to the next layer. This is referred to as token overflow.


FFNN 1 and FFNN 4 have reached capacity.
The token is sent to the next layer instead.
What
is
Mixture
of
Experts
?
Summarize
Router
FFNN 1
FFNN 2
FFNN 3
FFNN 4

References

- Dao, Tri, et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. Advances in Neural Information Processing Systems (NeurIPS), 2022
- Alammar, Jay, and Maarten Grootendorst. Hands-On Large Language Models: Language Understanding and Generation. O’Reilly Media, 2024.
- Grootendorst, M. (2024, October 7). A Visual Guide to Mixture of Experts (MoE): Demystifying the role of MoE in Large Language Models. Exploring Language Models.