ECS 289D: Datacenter Systems for LLMs

# FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

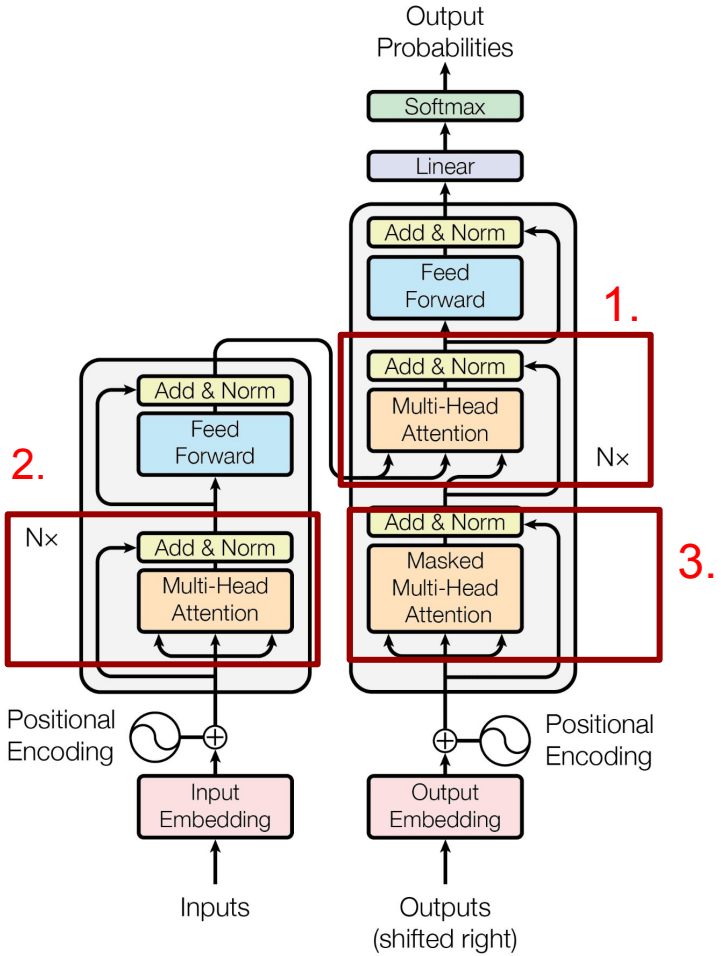
**Authors: Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré**

Presented by Ansha Prashanth

# Introduction

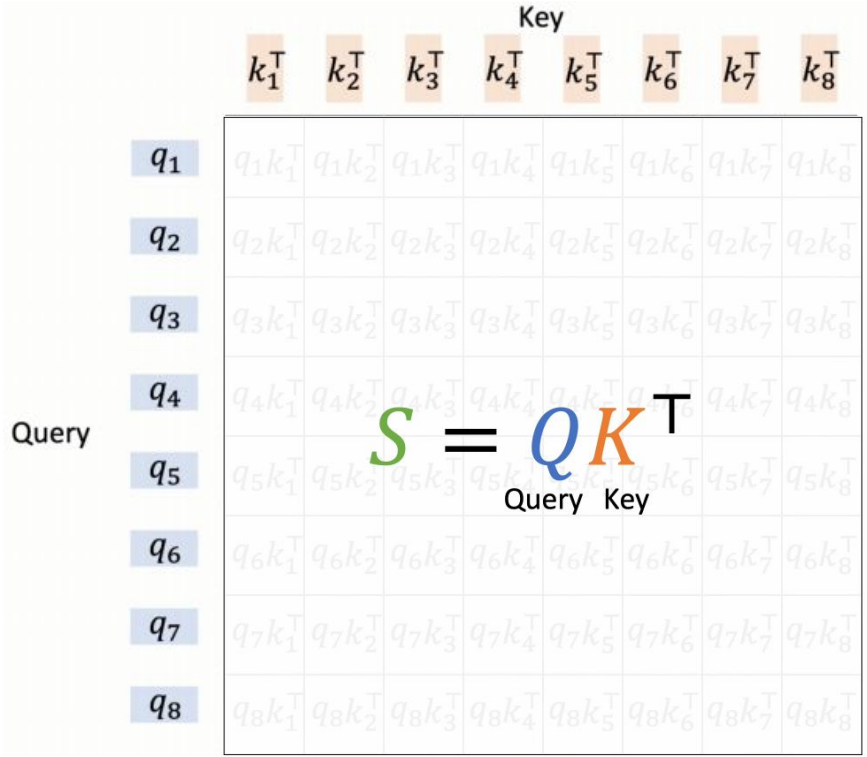
A transformer model contains many base attention layers:

1. Cross-Attention layer: Decoder-encoder attention
2. Global Self-attention layer: Encoder self-attention
3. Causal self-attention layer: Decoder self-attention



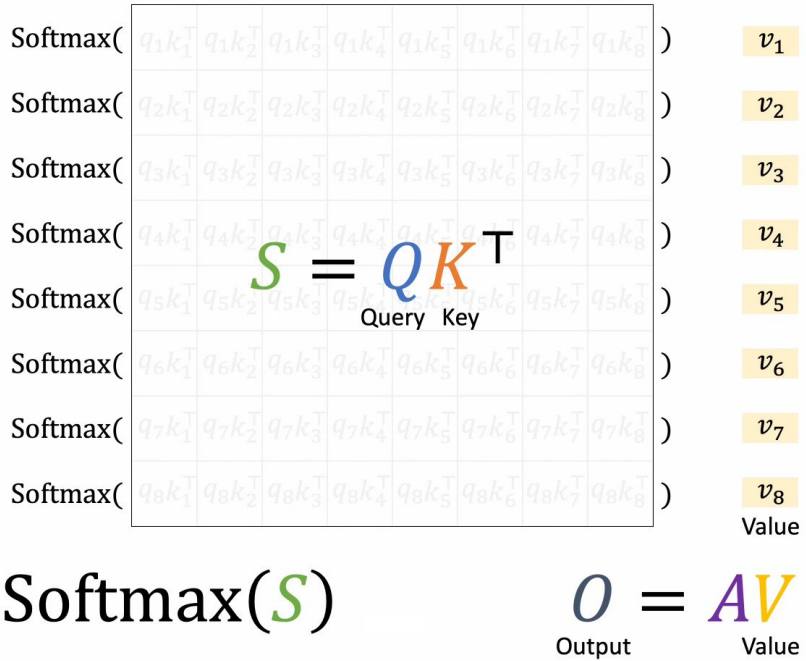
# Self Attention

- **Query (Q)**: Represents the current state or part of the sequence we are focusing on.
- **Key (K)**: Represents all parts of the sequence we compare the query against.
- **Value (V)**: Represents the actual information we want to use in the output.
- **N** is the sequence length and **d** is the head dimension
- **Output (O)**

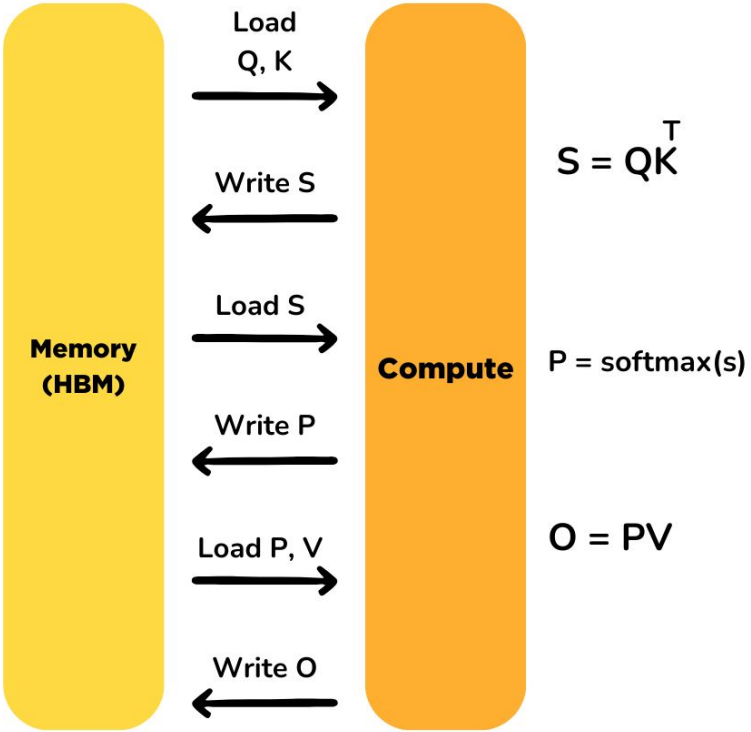


# Self Attention

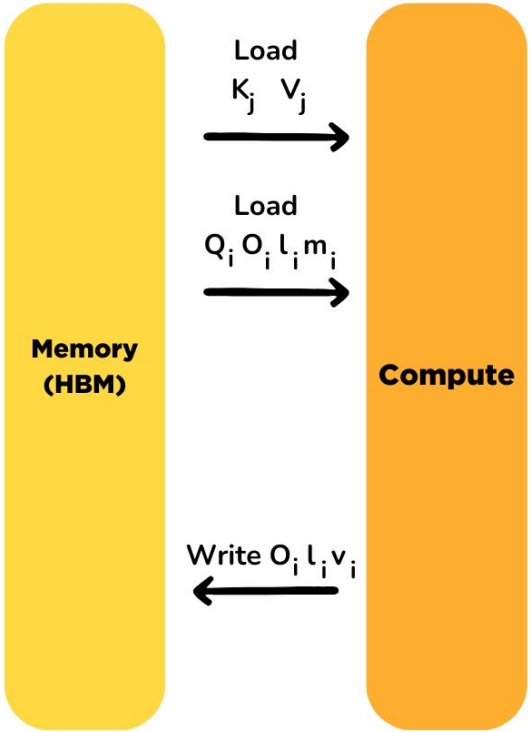
- Attention Score Matrix (S): Calculated via the dot product of Q and K
- Probability Matrix (P or A): Softmax is applied row-wise to S
- Final Output (O): Computed as the weighted sum of V
- The intermediate matrices, S and P, are N×N, requiring **O($N^2$)** memory, which is the root of the memory bottleneck.



## Standard Attention Implementation




## Flash Attention




Kernel operations fused together, reducing reads & writes

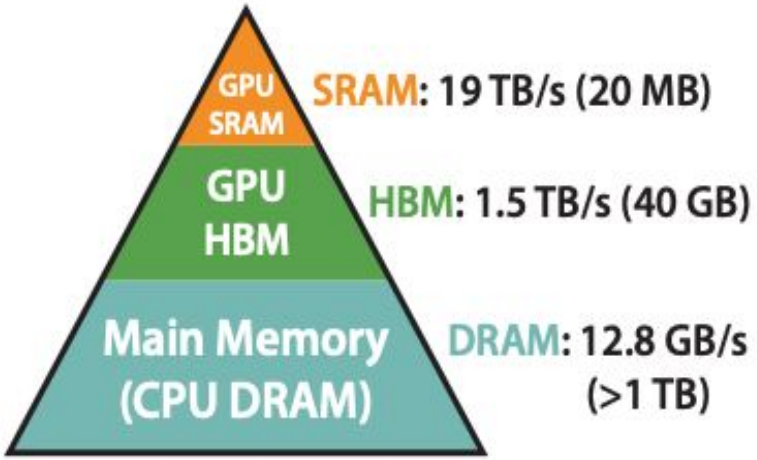
$S_{ij} = Q_i K_j^T$
$\tilde{m}$ = rowmax of S
P = exp(s - m)
l = rowsum of P
$m = \max(\tilde{m}_{ij}, m)$
calculate O from l & m

Initialize O, l and m matrices with zeroes. m and l are used to calculate cumulative softmax. Divide Q, K, V into blocks (due to SRAM's memory limits) and iterate over them, for i is row & j is column.

# GPU Memory Hierarchy

- **SRAM**: fast, on-chip, small
- **HBM (High Bandwidth Memory)**: slower than SRAM, large size. That's what we usually address as GPU memory.

- "*On modern GPUs, compute speed has out-paced memory speed, and most operations in Transformers are bottlenecked by memory accesses*". This is the **main motivation for FlashAttention.**




Memory Hierarchy with Bandwidth & Memory Size

# FlashAttention

Tiling and recomputation are applied to overcome the technical challenge of computing exact attention with sub-quadratic HBM access

**Tiling**:

- Split Q, K, V into blocks and load them from HBM to SRAM
- Compute attention block by block without storing full S,P matrices
- Use online softmax statistics m(x), $\ell$(x) to combine block results.

**Recomputation**:

- Only store the final output O and softmax statistics (m, $\ell$)
- Recompute S, P on the fly during backpropagation using Q, K, V blocks

# Tiling

$C = A \times B$

$B$

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

$A$

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

| | | | |
|---|---|---|---|
| $C_{1,1}$ | $C_{1,2}$ | | |
| $C_{2,1}$ | $C_{2,2}$ | | |
| | | | |
| | | | |

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 9 & 10 \\ 13 & 14 \end{bmatrix}$$

Without tiling: 32 memory accesses

With tiling: 16 memory accesses

NxN block → 1/N memory access

# Tiling

$B$

| | |
|---|---|
| $B_{11}$ | $B_{12}$ |
| $B_{21}$ | $B_{22}$ |

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$
$$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$
$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$
$$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

$A$

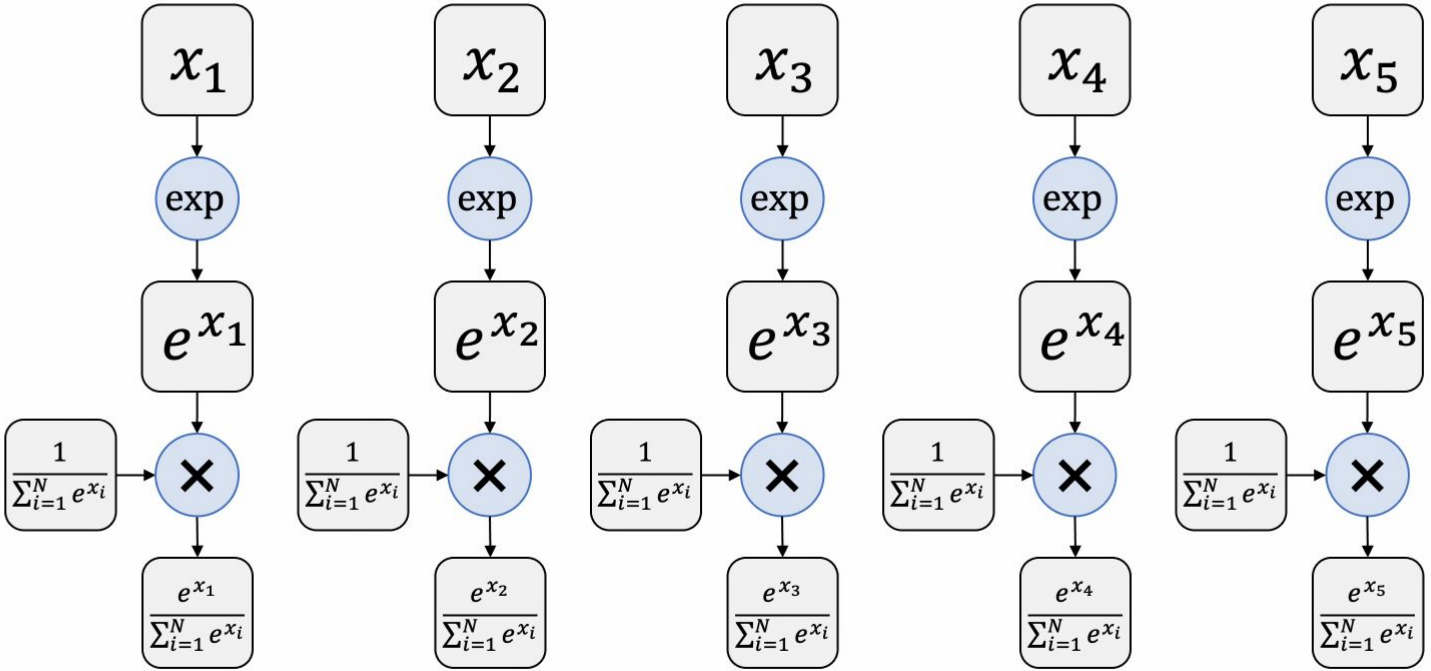
| | |
|---|---|
| $A_{11}$ | $A_{12}$ |
| $A_{21}$ | $A_{22}$ |

| | |
|---|---|
| $C_{11}$ | $C_{12}$ |
| $C_{21}$ | $C_{22}$ |

$C = A \times B$

$$S = QK^{\top}$$
$$A = \text{Softmax}(S)$$
$$O = AV$$

$$A = \text{Softmax}(S) \qquad S = \{x_1, x_2, \cdots, x_N\}$$



# Safe Softmax in FlashAttention

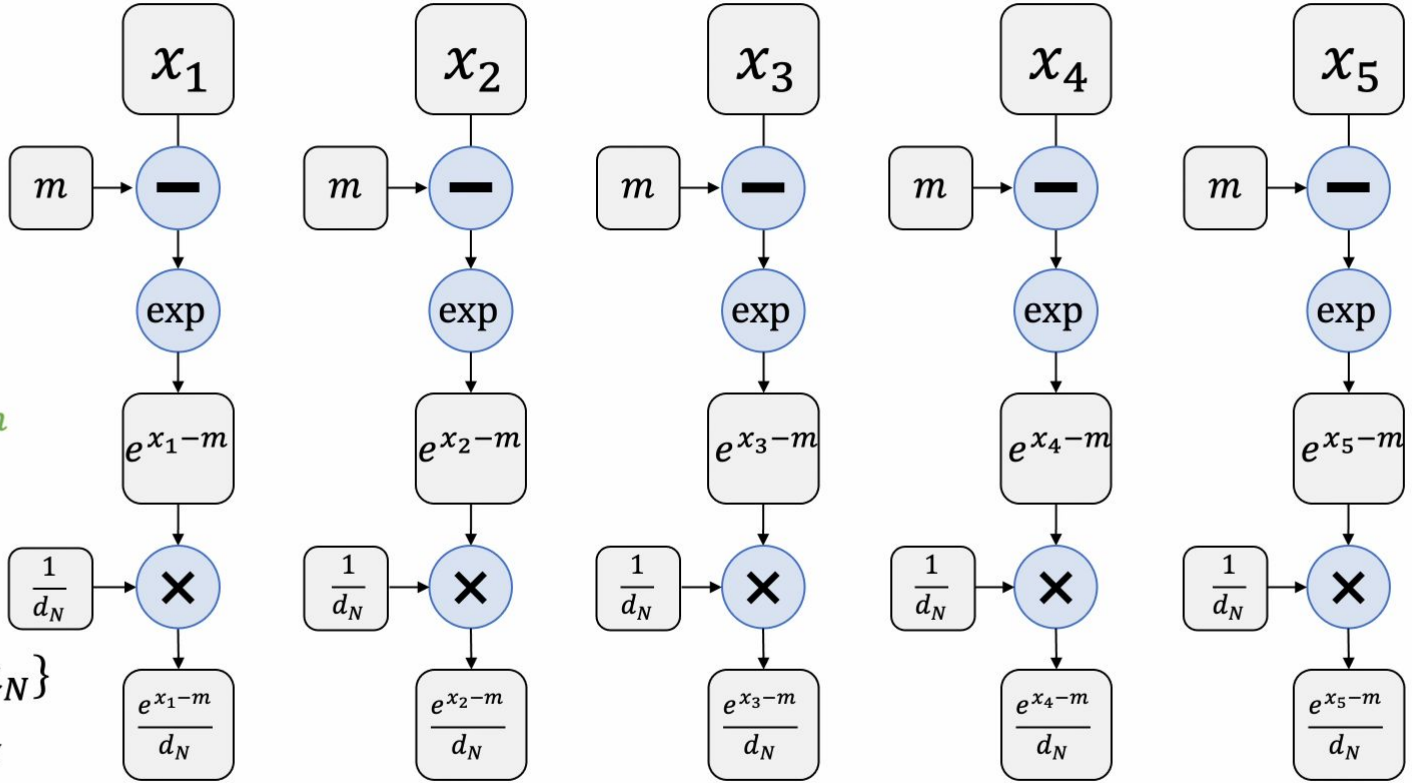
- Standard softmax computation involves exponentiating values, which may cause numerical instability, especially in FP16 due to dynamic range.
  - Direct exponentiation of large numbers (eg. $e^{12}$ =162754) may exceed FP16 limits (65504) leading to numerical errors.
- Safe Softmax mitigates this by shifting all inputs by subtracting the maximum value:

$$m = \max(x_i), \quad \text{softmax}(x_i) = \frac{e^{x_i - m}}{\sum_j e^{x_j - m}}$$

$A = \text{Softmax}(S)$

$S = \{x_1, x_2, \cdots, x_N\}$

$m = \max(\{x_1, x_2, \cdots, x_N\})$




$d_N = \sum_{i=1}^{N} e^{x_i-m}$

$A = \{a_1, a_2, \cdots, a_N\}$

$a_i = e^{x_i-m_N}/d_N$

$$A = \text{Softmax}(S) \qquad S = \{x_1, x_2, \cdots, x_N\}$$

$$m_N = \max(\{x_1, x_2, \cdots, x_N\})$$

$$m_0 = -\infty$$

**for** $1 \le i \le N$ **do**

$$m_i = \max(m_{i-1}, x_i)$$

$$d_N = \sum_{i=1}^{N} e^{x_i - m_N}$$

$$d_0 = 0$$

**for** $1 \le i \le N$ **do**

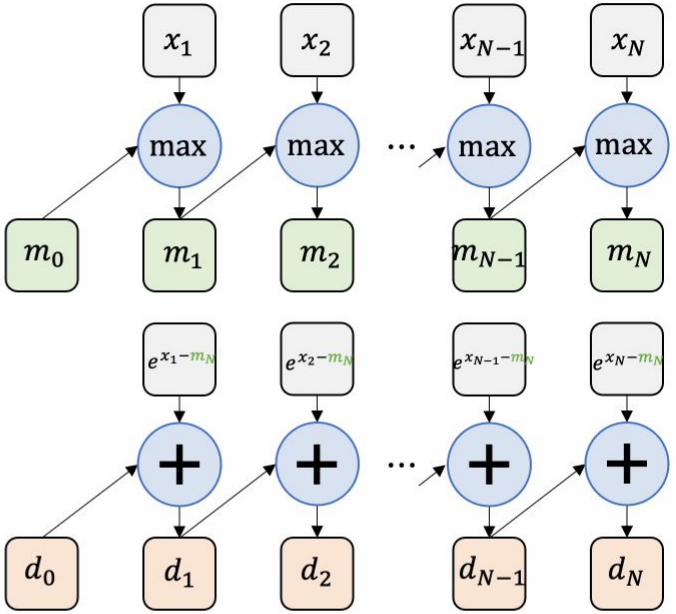
$$d_i = d_{i-1} + e^{x_i - m_N}$$

$$A = \{a_1, a_2, \cdots, a_N\}$$

$$a_i = e^{x_i - m_N} / d_N$$

**for** $1 \le i \le N$ **do**

$$a_i = e^{x_i - m_N} / d_N$$



$$A = \text{Softmax}(S) \qquad S = \{x_1, x_2, \cdots, x_N\}$$

$m_0 = -\infty$

**for** $1 \le i \le N$ **do**

$m_i = \max(m_{i-1}, x_i)$

$$d_i = \sum_{j=1}^{i} e^{x_j - m_N} \qquad d'_i = \sum_{j=1}^{i} e^{x_j - m_i} \qquad d'_N = d_N$$

$d_0 = 0$

**for** $1 \le i \le N$ **do**

$d_i = d_{i-1} + e^{x_i - m_N}$

$$d'_i = \underbrace{\left(\sum_{j=1}^{i-1} e^{x_j - m_{i-1}}\right)}_{d'_{i-1}} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d_N$

$$A = \text{Softmax}(S) \qquad S = \{x_1, x_2, \cdots, x_N\}$$

$m_0 = -\infty$

**for** $1 \le i \le N$ **do**

$m_i = \max(m_{i-1}, x_i)$

$d_0 = 0$

**for** $1 \le i \le N$ **do**

$d_i = d_{i-1} + e^{x_i - m_N}$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d_N$

$d'_i = d'_{i-1}\, e^{m_{i-1} - m_i} + e^{x_i - m_i} \qquad d'_N = d_N$

$m_0 = -\infty$

$d_0 = 0$

**for** $1 \le i \le N$ **do**

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d'_N$

Online normalizer calculation for softmax, 2018

# Online Softmax

- FlashAttention uses the mathematical properties of the softmax function to **decompose** the **normalization.**
- This allows us to compute 'local' softmax over each block and then rescale them to get the correct output incrementally.

$S = QK^\top \qquad A = \text{Softmax}(S) \qquad O = AV \qquad S = \{x_1, x_2, \cdots, x_N\}$

$x_i = qk_i^\top$

$d'_i = d'_{i-1}\, e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$m_0 = -\infty$

$d_0 = 0$

**for** $1 \le i \le N$ **do**

$x_i = qk_i^\top$

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$o_0 = 0$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d'_N$

$o_i = o_{i-1} + a_i\, v_i$

**return** $o_N$

$$o_i = \sum_{j=1}^{i} \frac{e^{x_j - m_N}}{d'_N} v_j \qquad o_N = o'_N$$

$$o'_i = \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} v_j$$

$S = QK^\top \qquad A = \text{Softmax}(S) \qquad O = AV \qquad S = \{x_1, x_2, \cdots, x_N\}$

$x_i = qk_i^\top$

$d'_i = d'_{i-1}\, e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$m_0 = -\infty$

$d_0 = 0$

$$o'_i = \sum_{j=1}^{i-1} \frac{e^{x_j - m_i}}{d'_i} \frac{d'_{i-1}}{d'_{i-1}} \frac{e^{-m_{i-1}}}{e^{-m_{i-1}}} v_j + \frac{e^{x_i - m_i}}{d'_i} v_i$$

**for** $1 \le i \le N$ **do**

$x_i = qk_i^\top$

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$o_0 = 0$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d'_N$

$o_i = o_{i-1} + a_i v_i$

**return** $o_N$

$S = QK^\top \qquad A = \text{Softmax}(S) \qquad O = AV \qquad S = \{x_1, x_2, \cdots, x_N\}$

$x_i = qk_i^\top$

$d'_i = d'_{i-1}\, e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$m_0 = -\infty$

$d_0 = 0$

**for** $1 \le i \le N$ **do**

$x_i = qk_i^\top$

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

$o_0 = 0$

**for** $1 \le i \le N$ **do**

$a_i = e^{x_i - m_N} / d'_N$

$o_i = o_{i-1} + a_i\, v_i$

**return** $o_N$

$$o'_i = \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} v_j\right) \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} v_i$$

$o'_{i-1}$

$S = QK^\top$ $A = \text{Softmax}(S)$ $O = AV$ $S = \{x_1, x_2, \cdots, x_N\}$

$x_i = qk_i^\top$

**for** $1 \leq i \leq N$ **do**

$x_i = qk_i^\top$

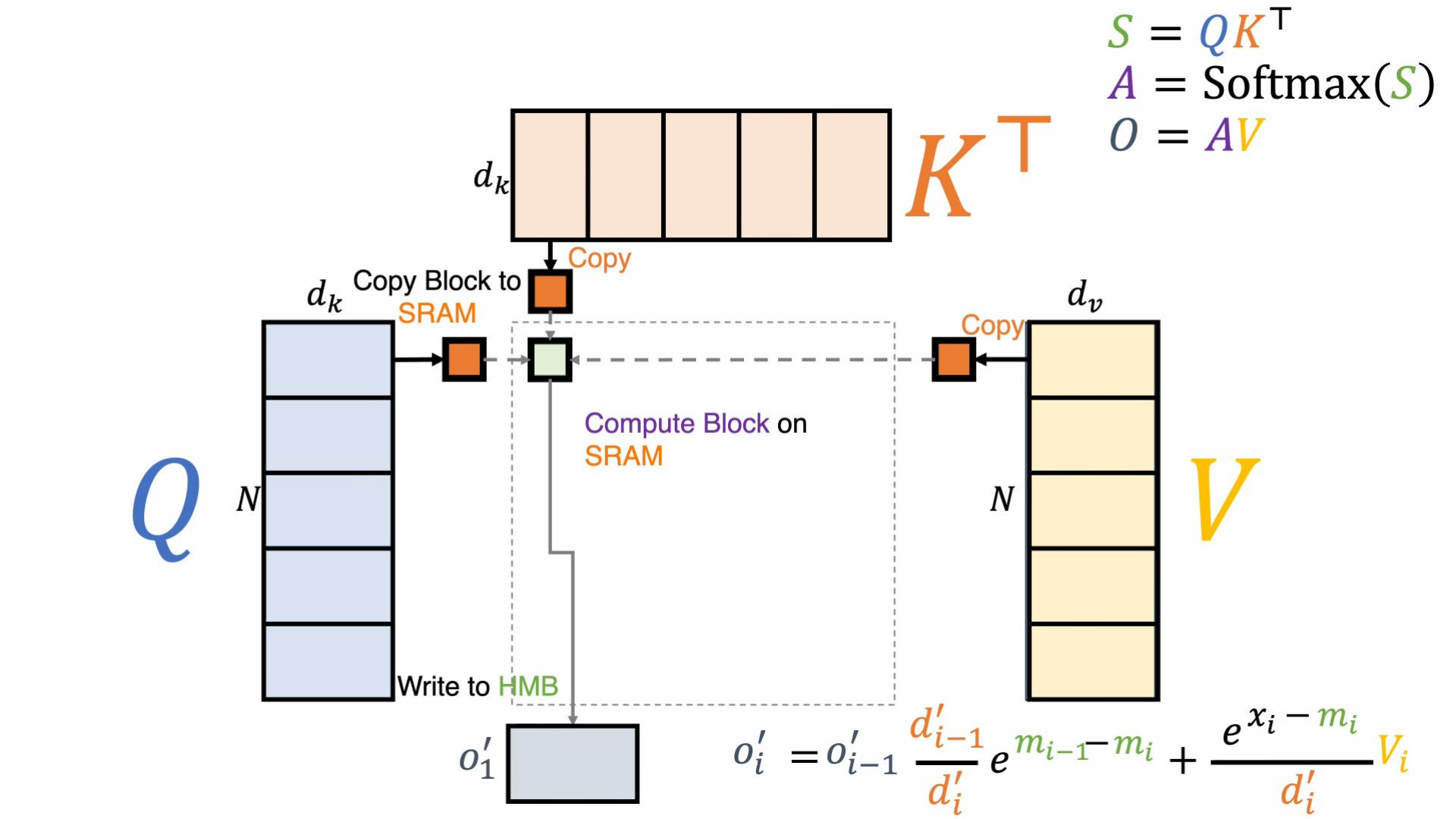
$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$

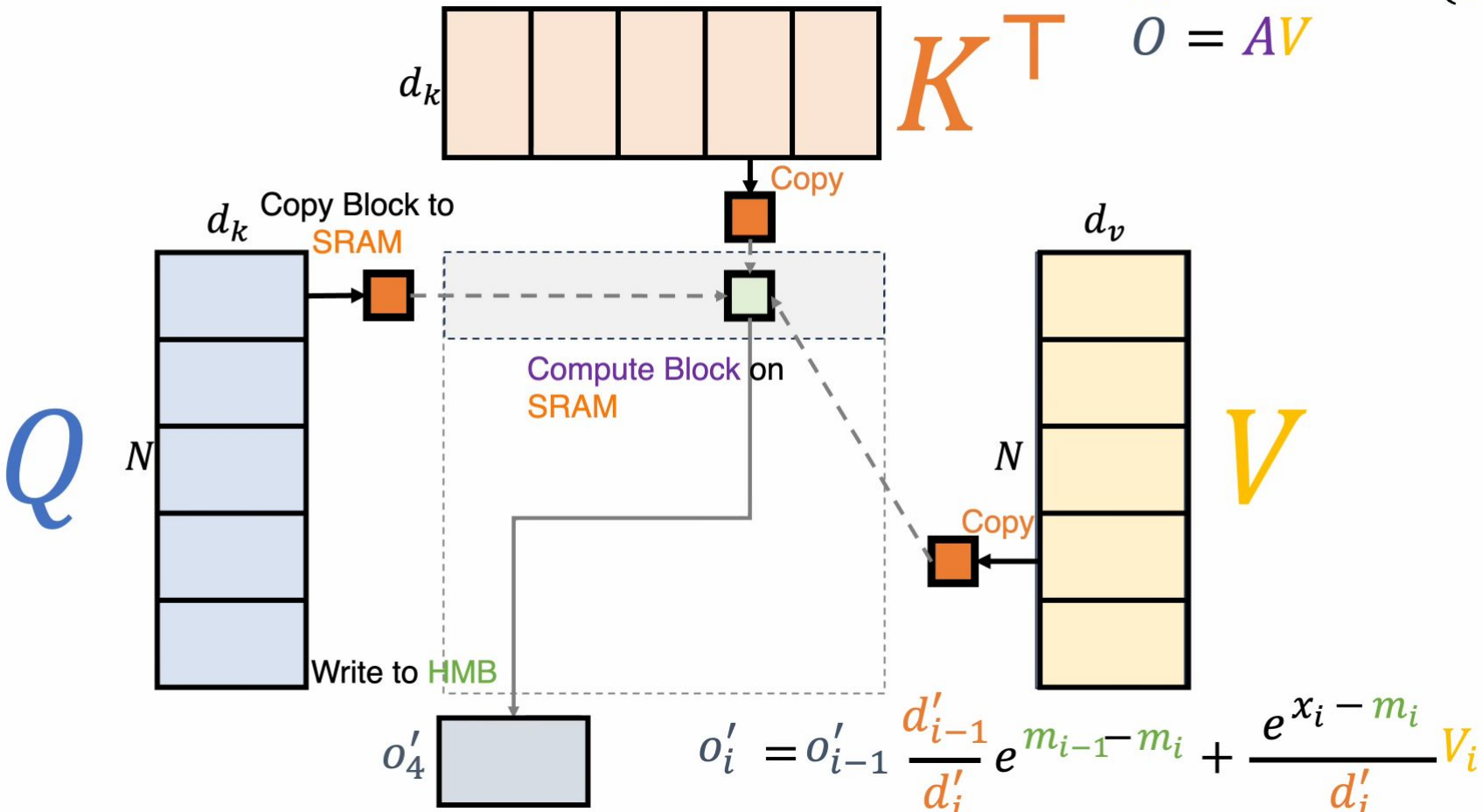
$o'_i = o'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} v_i$

**return** $o'_N$

Fast and Memory-Efficient Exact Attention with IO-Awareness, 2022

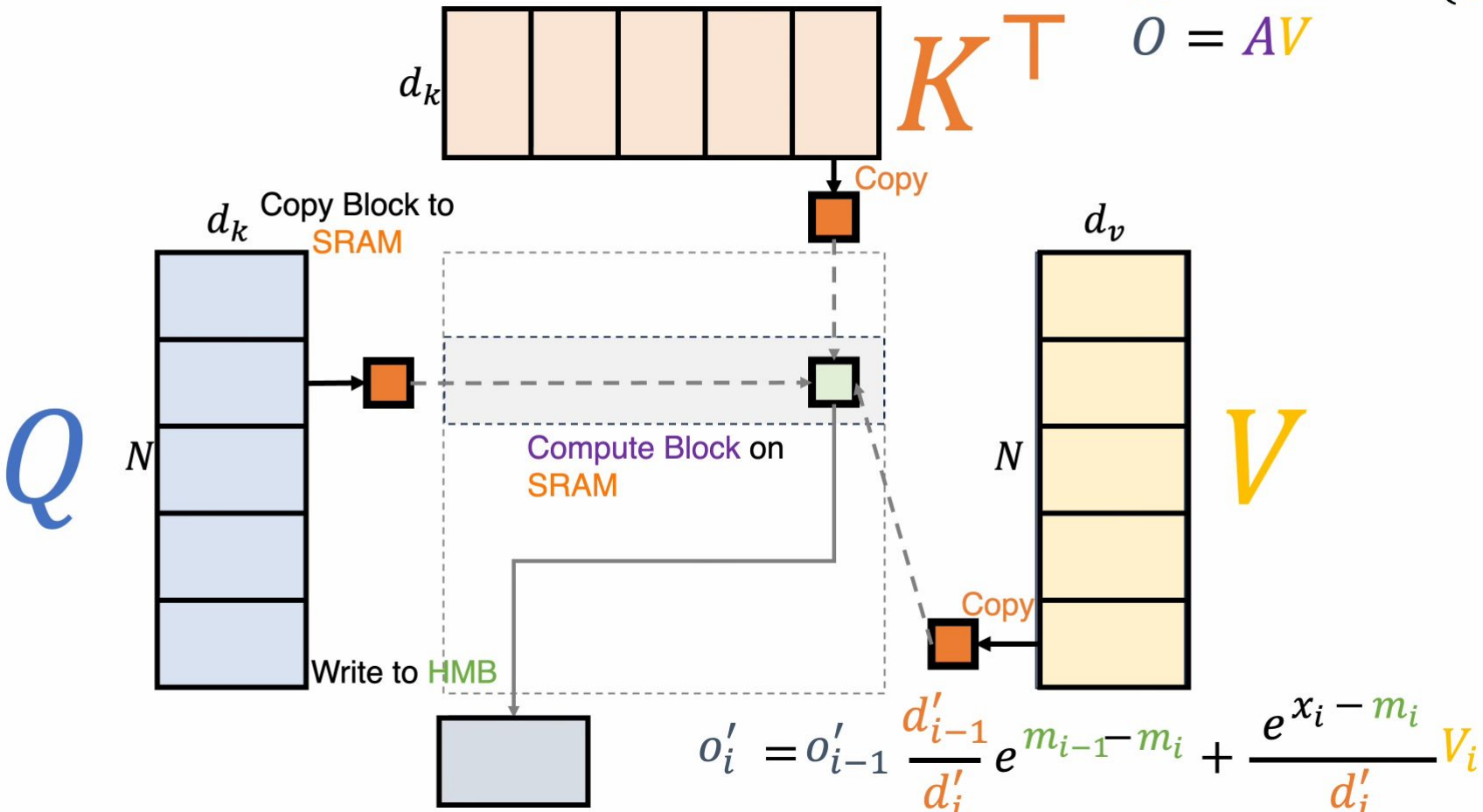

$S = QK^\top$
$A = \mathrm{Softmax}(S)$
$O = AV$
$d_k$
$K^\top$
Copy
$d_k$
Copy Block to SRAM
$d_v$
Copy
Compute Block on SRAM
$Q$
$N$
$N$
$V$
Write to HMB
$o_1'$
$o_i' = o_{i-1}' \frac{d_{i-1}'}{d_i'} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d_i'} V_i$

$$S = QK^{\top}$$
$$A = \mathrm{Softmax}(S)$$
$$O = AV$$




$$o'_i = o'_{i-1}\frac{d'_{i-1}}{d'_i}e^{m_{i-1}-m_i} + \frac{e^{x_i - m_i}}{d'_i}V_i$$

$$S = QK^{\top}$$
$$A = \mathrm{Softmax}(S)$$
$$O = AV$$




$$o_i' = o_{i-1}' \frac{d_{i-1}'}{d_i'} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d_i'} V_i$$

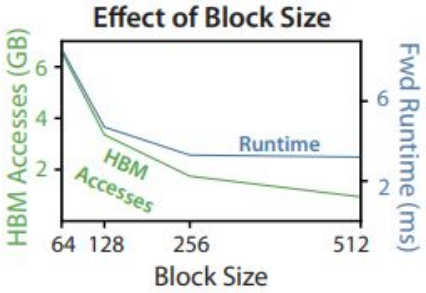
# Backward Pass and Recomputation

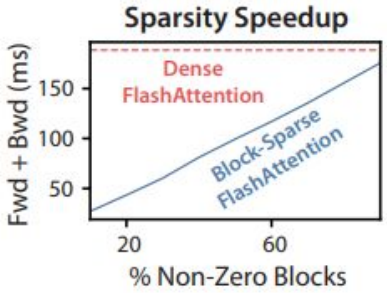
- Standard Attention:
  - The backward pass requires the original N×N attention score matrix (S) and probability matrix (P) to compute the gradients dQ, dK, and dV using the **chain rule**.
- FlashAttention Solution: **Recomputation**
  - This strategy is a form of selective gradient checkpointing.
  - We only store the output O and the compact, linear-sized softmax normalization statistics (m and ℓ) from the forward pass.
- How it works:
  - We load blocks of *Q, K, V, dO, O, m, and ℓ* into SRAM
  - We recompute the attention matrix block $(S_{ij})$ and probability matrix block $(P_{ij})$ on-chip using the stored m and ℓ and the original input Q and K
  - We then use this recomputed $P_{ij}$ along with the output gradient (dO) and the original O to compute the input gradients dQ, dK, and dV locally.

# How impactful is FlashAttention?

| Attention | Standard | FlashAttention |
|---|---|---|
| GFLOPs | 66.6 | 75.2 |
| HBM R/W (GB) | 40.3 | 4.4 |
| Runtime (ms) | 41.7 | 7.3 |

**Forward + backward runtime for GPT-2 on A100 GPU**




| BERT Implementation | Training time (minutes) |
|---|---|
| Nvidia MLPerf 1.1 [58] | 20.0 ± 1.5 |
| FlashAttention (ours) | **17.4** ± 1.4 |

| Models | ListOps | Text | Retrieval | Image | Pathfinder | Avg | Speedup |
|---|---|---|---|---|---|---|---|
| Transformer | 36.0 | 63.6 | 81.6 | 42.3 | 72.7 | 59.3 | - |
| FlashAttention | 37.6 | 63.9 | 81.4 | 43.5 | 72.7 | 59.8 | 2.4× |
| Block-sparse FlashAttention | 37.0 | 63.0 | 81.3 | 43.6 | 73.3 | 59.6 | **2.8**× |

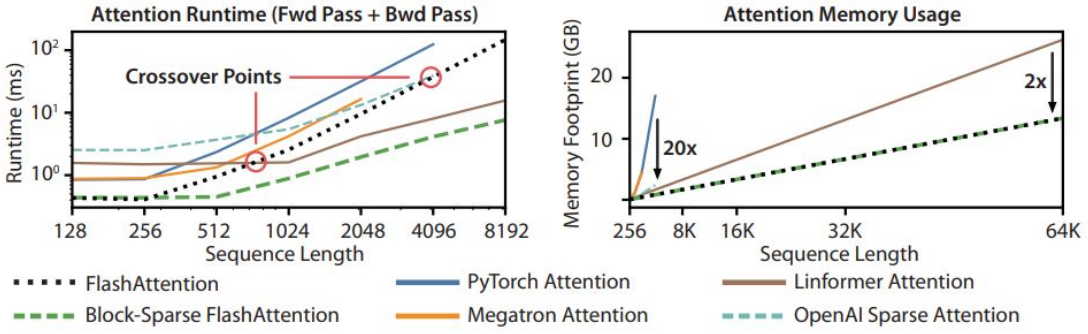
| Model implementations | OpenWebText (ppl) | Training time (speedup) |
|---|---|---|
| GPT-2 small - Huggingface [87] | 18.2 | 9.5 days (1.0×) |
| GPT-2 small - Megatron-LM [77] | 18.2 | 4.7 days (2.0×) |
| GPT-2 small - FlashAttention | 18.2 | **2.7 days (3.5×)** |
| GPT-2 medium - Huggingface [87] | 14.2 | 21.0 days (1.0×) |
| GPT-2 medium - Megatron-LM [77] | 14.3 | 11.5 days (1.8×) |
| GPT-2 medium - FlashAttention | 14.3 | **6.9 days (3.0×)** |

**15% wall-clock speed-up on BERT-large (seq. Length 512);**
**3x on GPT-2 (seq. Length 1K);**
**2.4x on long-range arena (seq. length 1K-4K)**

# How impactful is FlashAttention?

| Model implementations | Context length | OpenWebText (ppl) | Training time (speedup) |
|---|---|---|---|
| GPT-2 small - Megatron-LM | 1k | 18.2 | 4.7 days (1.0×) |
| GPT-2 small - FlashAttention | 1k | 18.2 | **2.7 days (1.7×)** |
| GPT-2 small - FlashAttention | 2k | 17.6 | 3.0 days (1.6×) |
| GPT-2 small - FlashAttention | 4k | **17.5** | 3.6 days (1.3×) |

| **Model** | Path-X | Path-256 |
|---|---|---|
| Transformer | ✗ | ✗ |
| Linformer [84] | ✗ | ✗ |
| Linear Attention [50] | ✗ | ✗ |
| Performer [12] | ✗ | ✗ |
| Local Attention [80] | ✗ | ✗ |
| Reformer [51] | ✗ | ✗ |
| SMYRF [19] | ✗ | ✗ |
| FlashAttention | **61.4** | ✗ |
| Block-sparse FlashAttention | 56.0 | **63.1** |



# Summary

- FlashAttention: IO-aware exact attention
- Reduces HBM access. Avoids storage of $QK^T$ for both forward and backward passes.
- Softmax is an algebraic operation
- Huge gains across GPT-2, BERT, and Transformer

# Thank You

References:


1. Dao, T., Fu, D., Ermon, S., Rudra, A. and Ré, C., 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing systems, 35, pp.16344-16359.
2. Dao, T., 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691.
3. https://www.youtube.com/@jbhuang0604