

# Long-context LLMs

**CS 5624: Natural Language Processing**  
*Spring 2025*

<https://tuvllms.github.io/nlp-spring-2025>

**Tu Vu**



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

Tri Dao<sup>†</sup>, Daniel Y. Fu<sup>†</sup>, Stefano Ermon<sup>†</sup>, Atri Rudra<sup>‡</sup>, and Christopher Ré<sup>†</sup>

<sup>†</sup>Department of Computer Science, Stanford University

<sup>‡</sup>Department of Computer Science and Engineering, University at Buffalo, SUNY

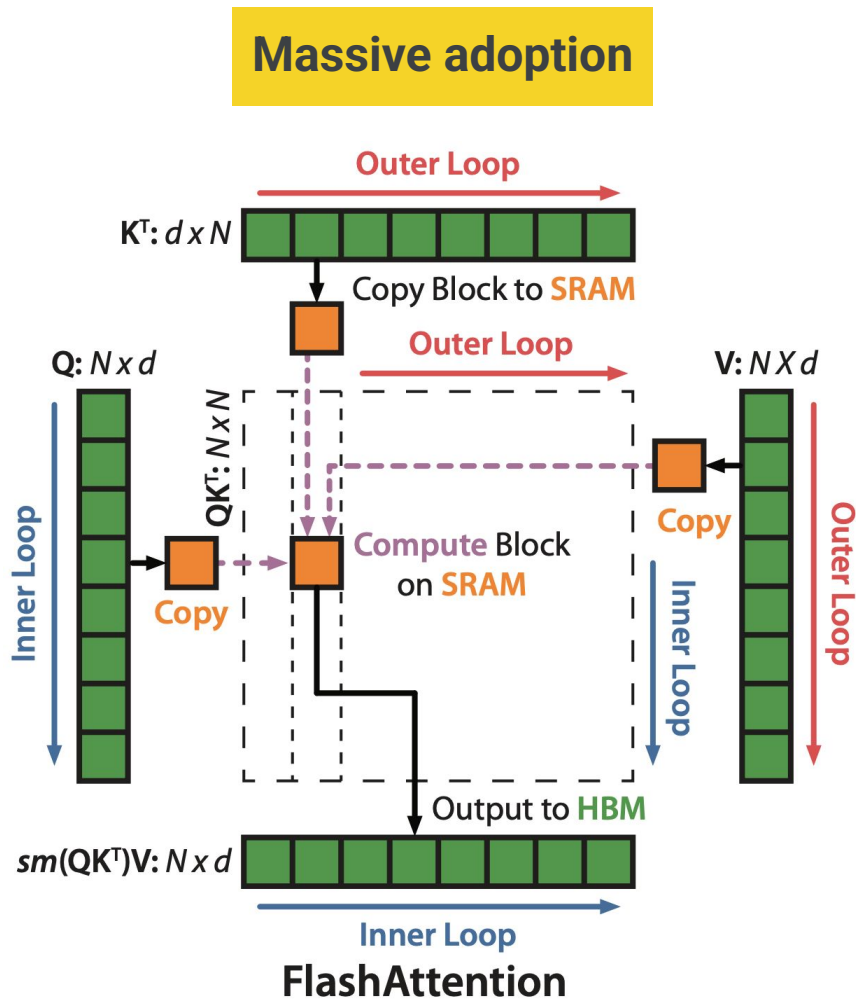
{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu,  
chrismre@cs.stanford.edu

# Why do we need to model longer sequences?

# How to model longer sequences?

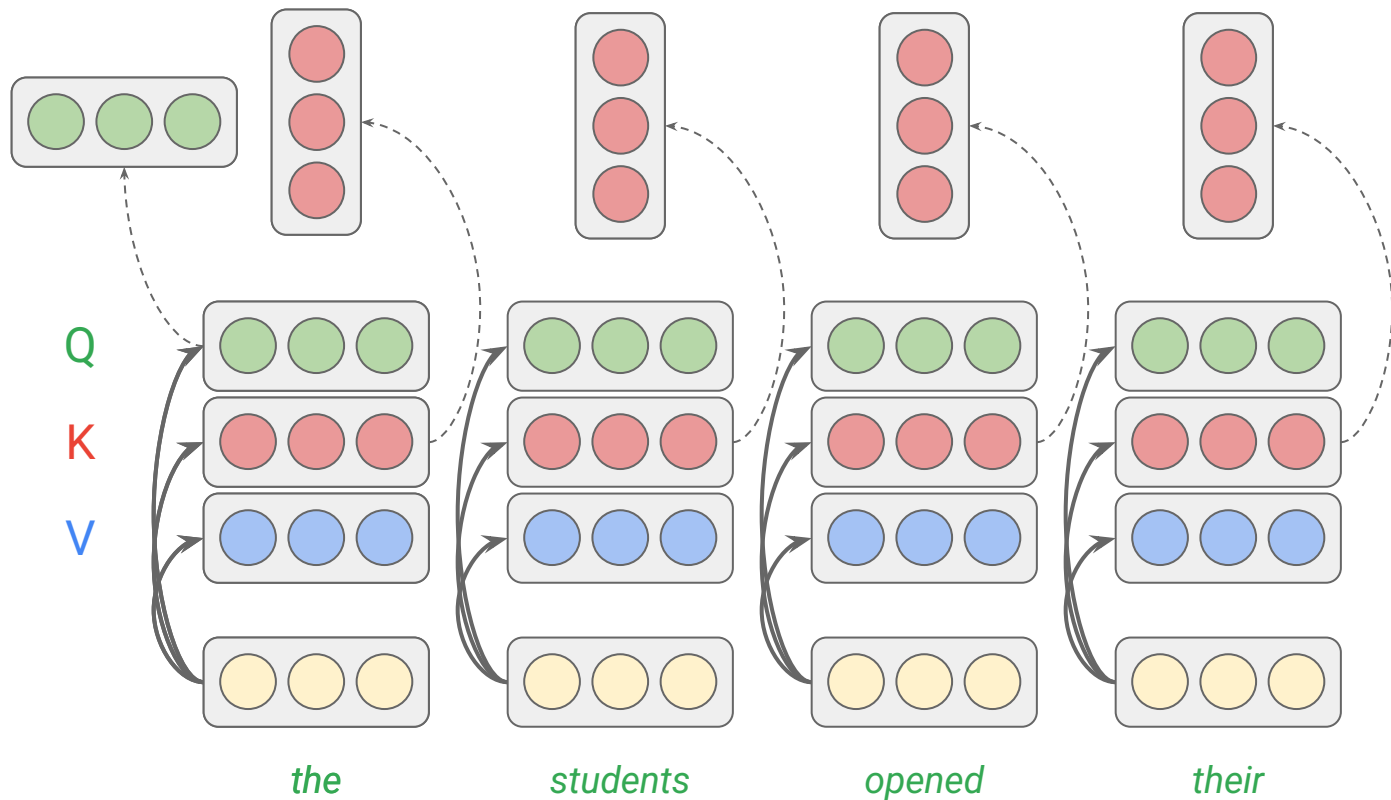
# FlashAttention

- **Tiling** and **recomputation** to reduce GPU memory IOs
  - **Fast** (3x) and **memory efficient** (10-20x) algorithm for **exact** attention
  - **Longer sequences** (up to 16K) yield **higher quality**



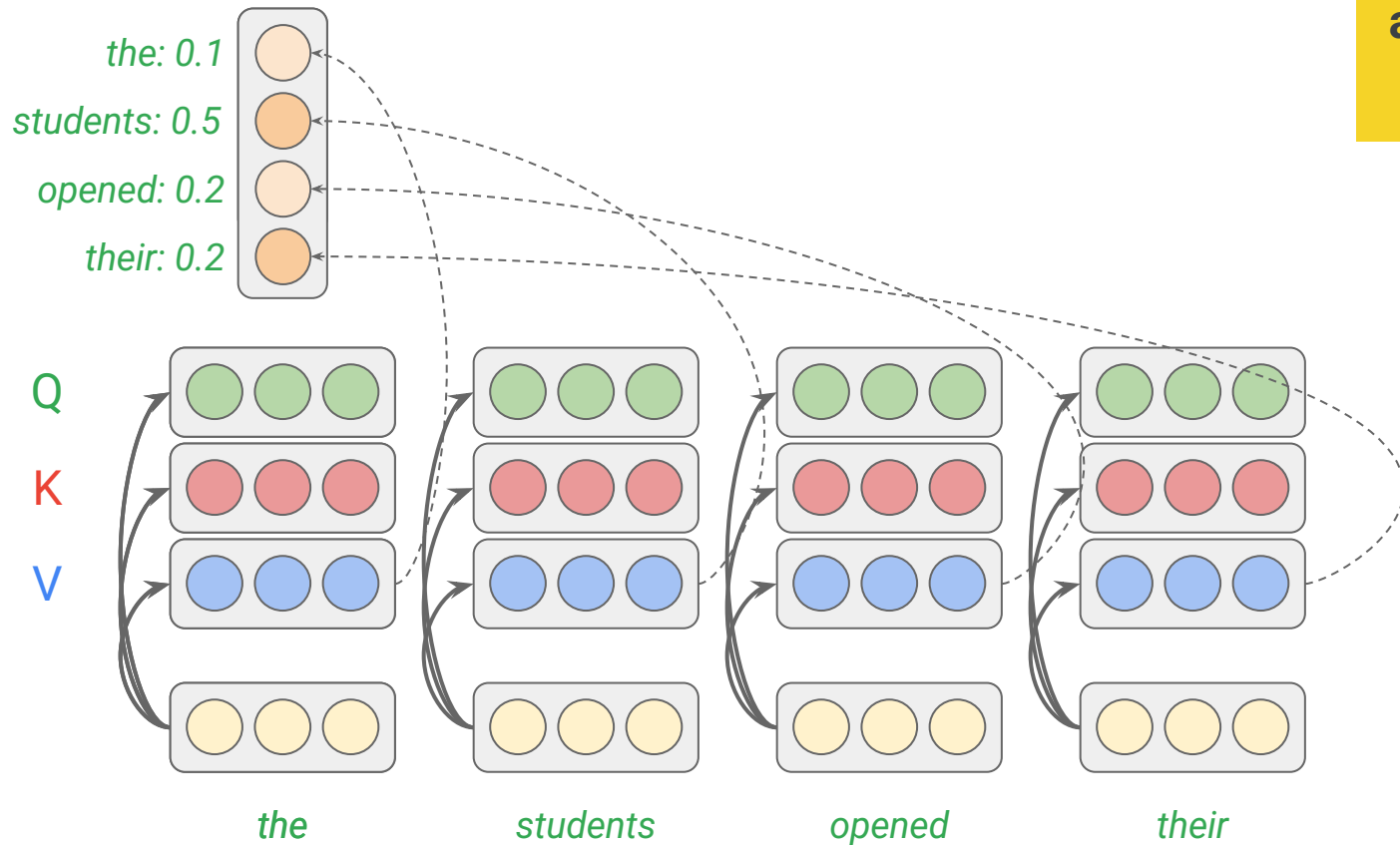
# Attention mechanism review

*all computations  
are parallelized*

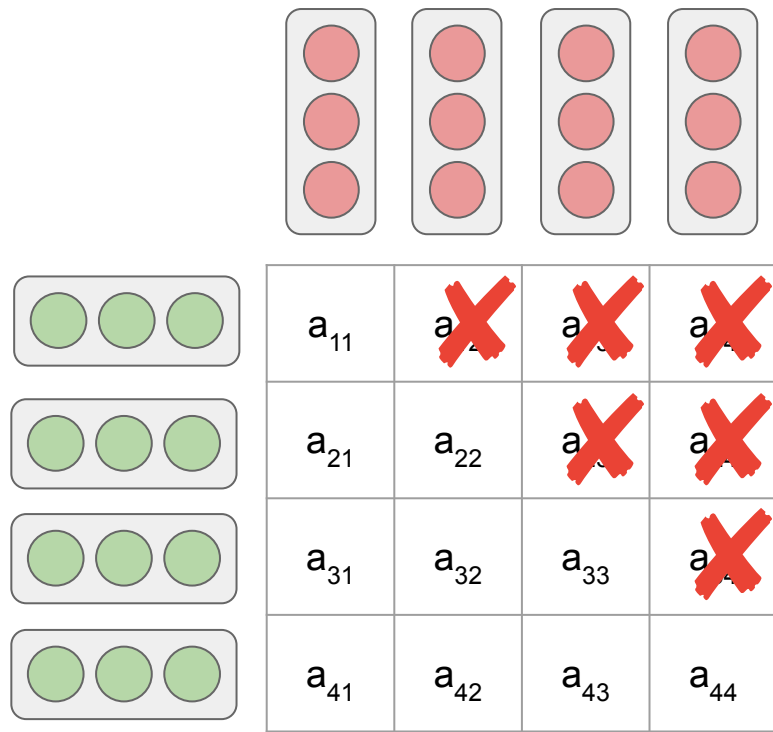


# Attention mechanism review (cont'd)

*all computations  
are parallelized*



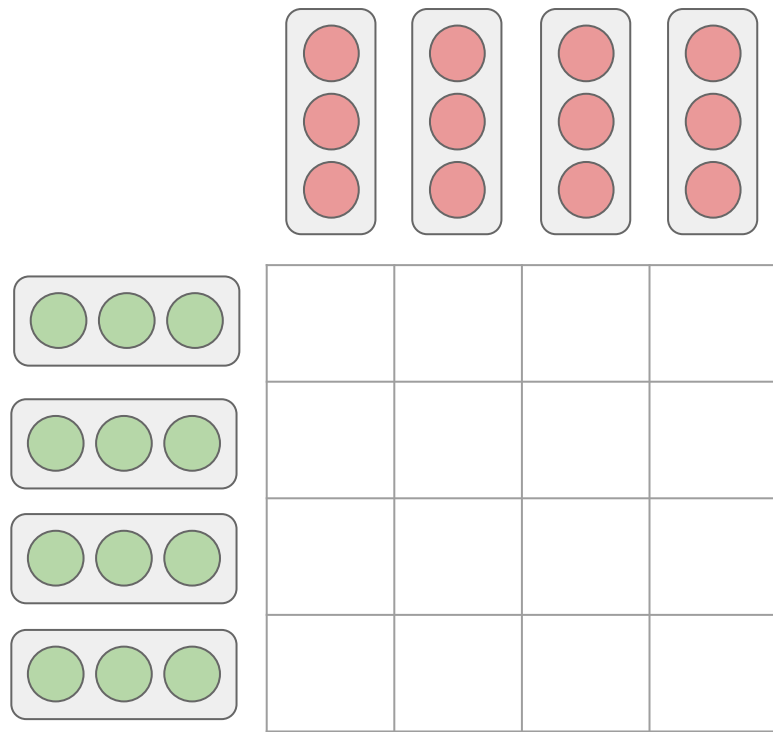
# Attention mechanism review (cont'd)



*masking out all values in the input of the softmax which correspond to illegal connections*

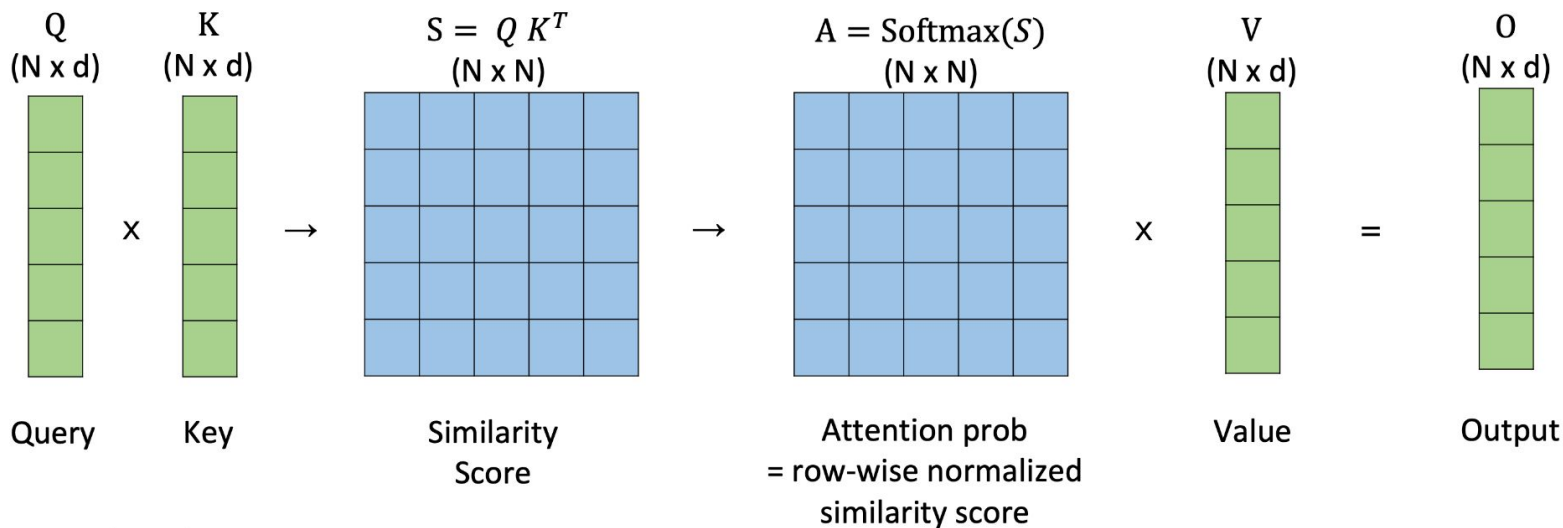


# Quadratic complexity



***The time complexity of self-attention is quadratic in the input length  $O(n^2)$***

# Attention mechanism review (cont'd)

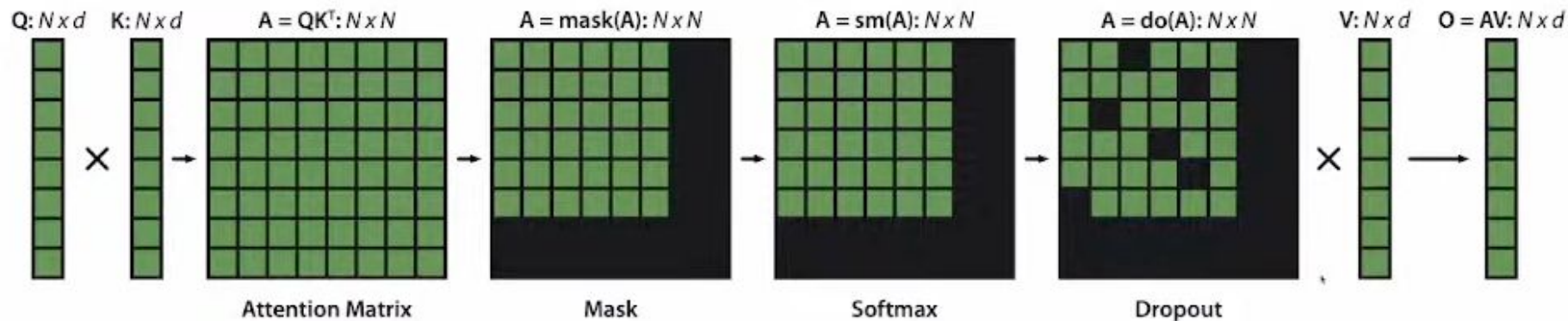


Typical sequence length  $N$ : 1K – 8K  
Head dimension  $d$ : 64 – 128

$$\text{Softmax}([s_1, \dots, s_N]) = \left[ \frac{e^{s_1}}{\sum_i e^{s_i}}, \dots, \frac{e^{s_N}}{\sum_i e^{s_i}} \right]$$

$$O = \text{Softmax}(QK^T)V$$

# Attention mechanism review (cont'd)



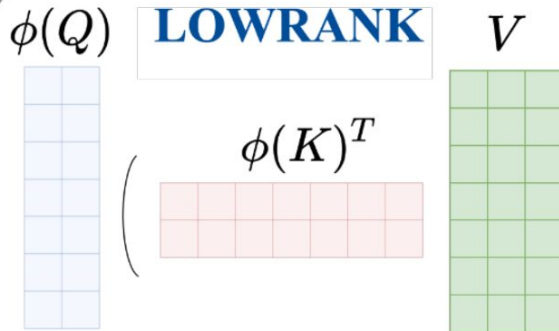
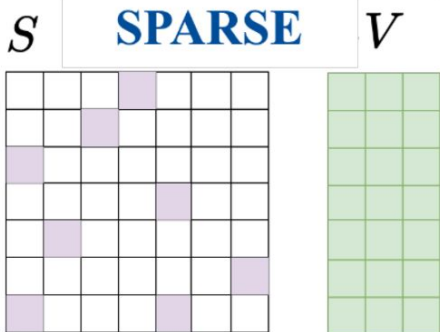
$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T)))\mathbf{V}$$

# Approximate attention

tradeoff **quality** for **speed** fewer FLOPs

*does not result in an actual wall clock speedup*

**Sparse Transformer**  
(Child et al. 19)  
**Reformer**  
(Kitaev et al. 20)  
**Routing Transformer**  
(Roy et al. 20)



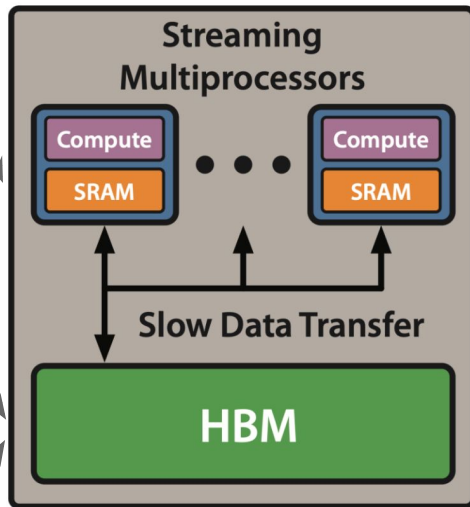
**Linformer**  
(Wang et al. 20)  
**Linear Transformer**  
(Katharopoulos et al. 20)  
**Performer**  
(Choromanski et al. 20)

# GPU compute model & memory hierarchy

**2. Data moved to compute units & SRAM for computation**

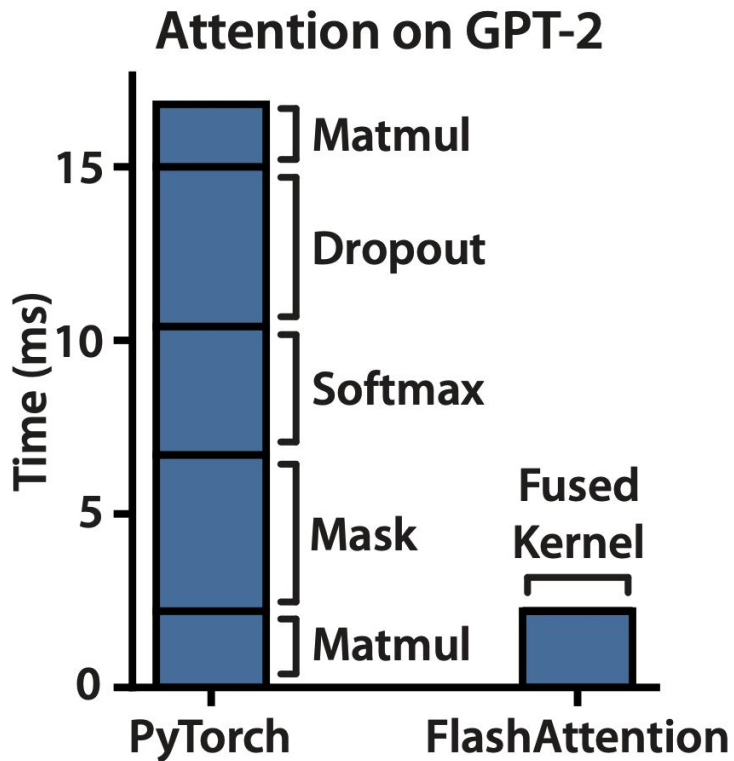
**1. Inputs start out in HBM (GPU memory)**

**3. Output written back to HBM**



**Can we exploit the memory asymmetry to get speed up?**

# Data movement is the key bottleneck



# How to reduce HBM reads/writes: compute by blocks

- **Challenges:**

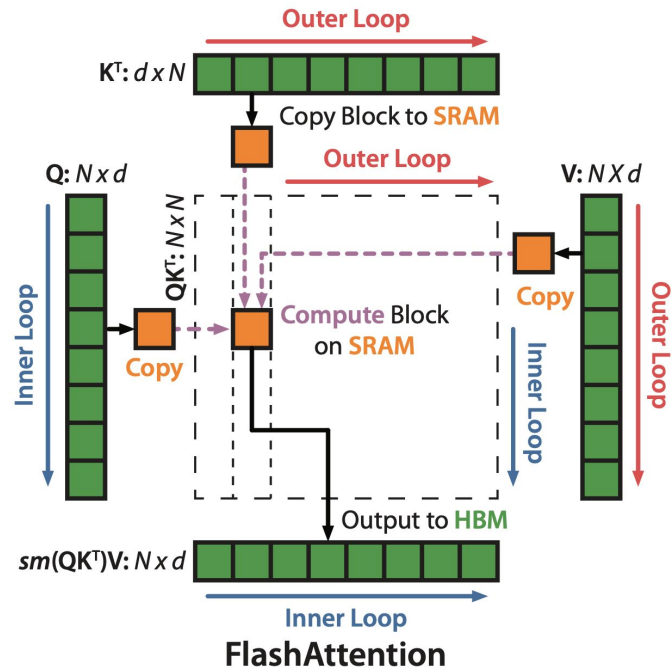
- Compute softmax normalization without access to full input
- Backward without the large attention matrix from forward

- **Approaches:**

- **Tiling:** Restructure algorithm to load block by block from HBM to SRAM to compute attention
- **Recomputation:** Don't store attention matrix from forward, recompute it in the backward

# Tiling

- Decomposing large softmax into smaller ones by scaling

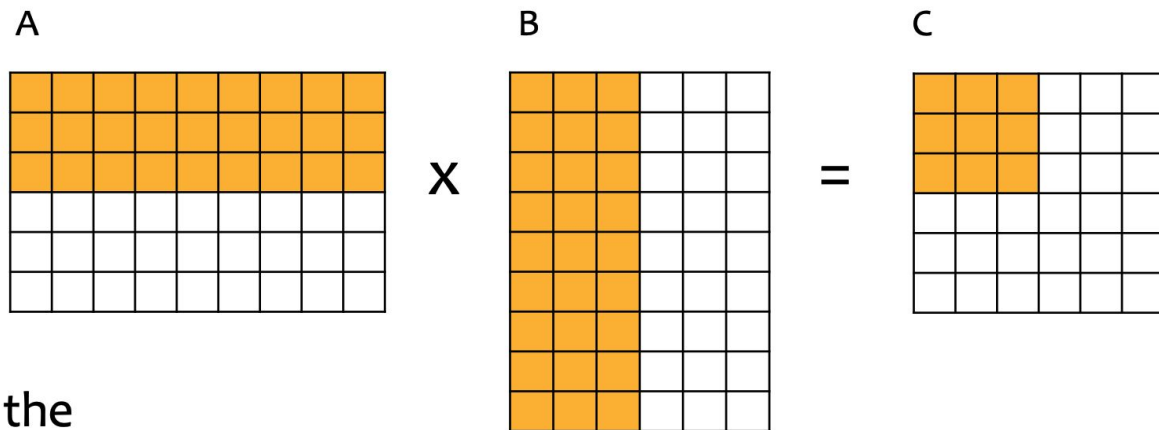


$$\text{softmax}([A_1, A_2]) = [\alpha \times \text{softmax}(A_1), \beta \times \text{softmax}(A_2)]$$

$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha \times \text{softmax}(A_1)V_1 + \beta \times \text{softmax}(A_2)V_2$$



# Tiling for matrix multiplication

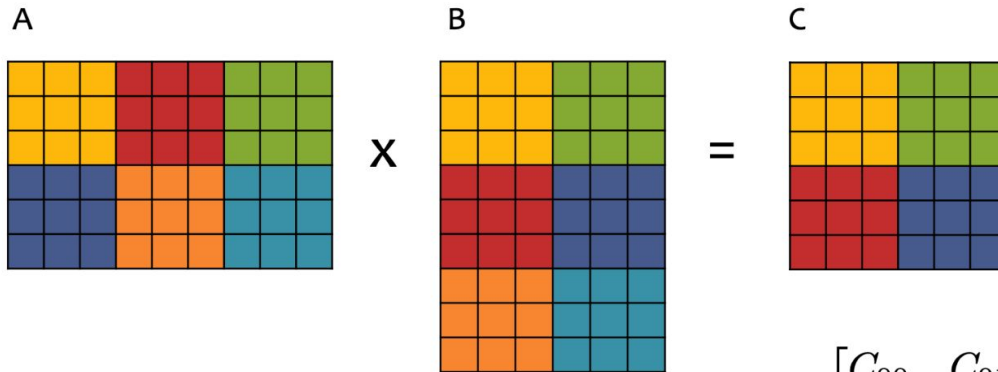


- We can view the computation as decomposing if we consider subsets of rows/columns

$$C_{(1,1):(3,3)} = A_{(1,1):(3,9)} \times B_{(1,1):(9,3)}$$

# Tiling for matrix multiplication (cont'd)

- Tiling capitalizes on this decomposition
- Each output tile is computed by multiplying a pair of input tiles and adding it to the appropriate output tile



$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \end{bmatrix}$$

with each  $A_{ij} \in \mathbb{R}^{3 \times 3}$

$$B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \\ B_{20} & B_{21} \end{bmatrix}$$

with each  $B_{ij} \in \mathbb{R}^{3 \times 3}$

$$C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

with each  $C_{ij} \in \mathbb{R}^{3 \times 3}$

$$C_{00} = A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$$

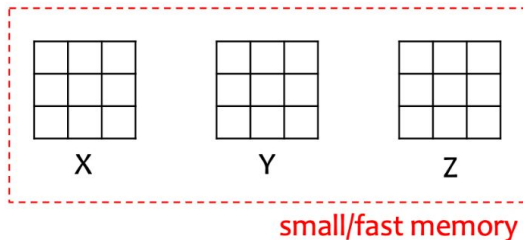
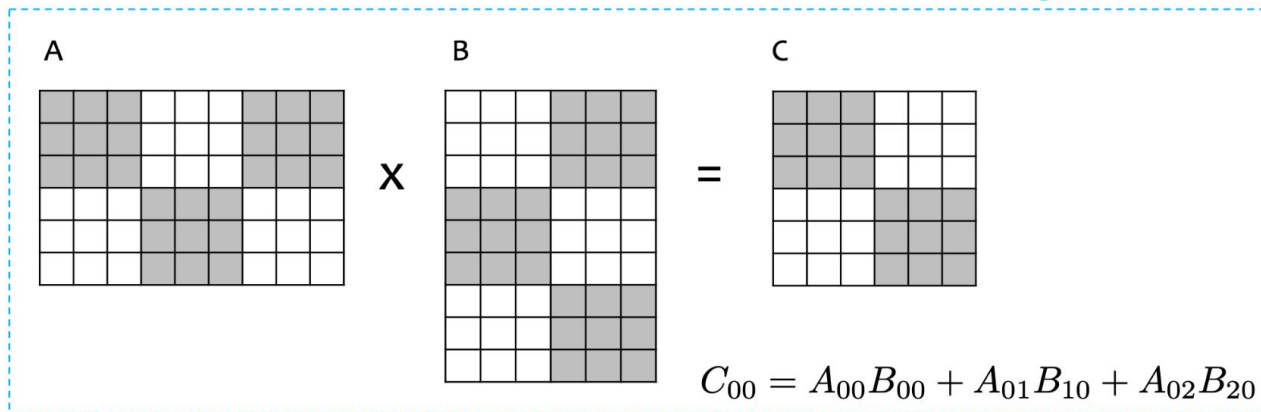
$$C_{01} = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$$

$$C_{10} = A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}$$

$$C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$$

# Tiling for matrix multiplication (cont'd)

- Tiling enables matrix multiplication of two **very** large matrices to capitalize on the small amount of fast memory on a device (e.g. GPU)
- Start by putting the input matrices and storage for the output matrix into large/slow memory
- Do the primary computation in slow/fast memory



$$X = A_{00}$$

$$Y = B_{00}$$

$$Z = XY$$

$$X = A_{01}$$

$$X = A_{02}$$

$$Y = B_{10}$$

$$Y = B_{20}$$

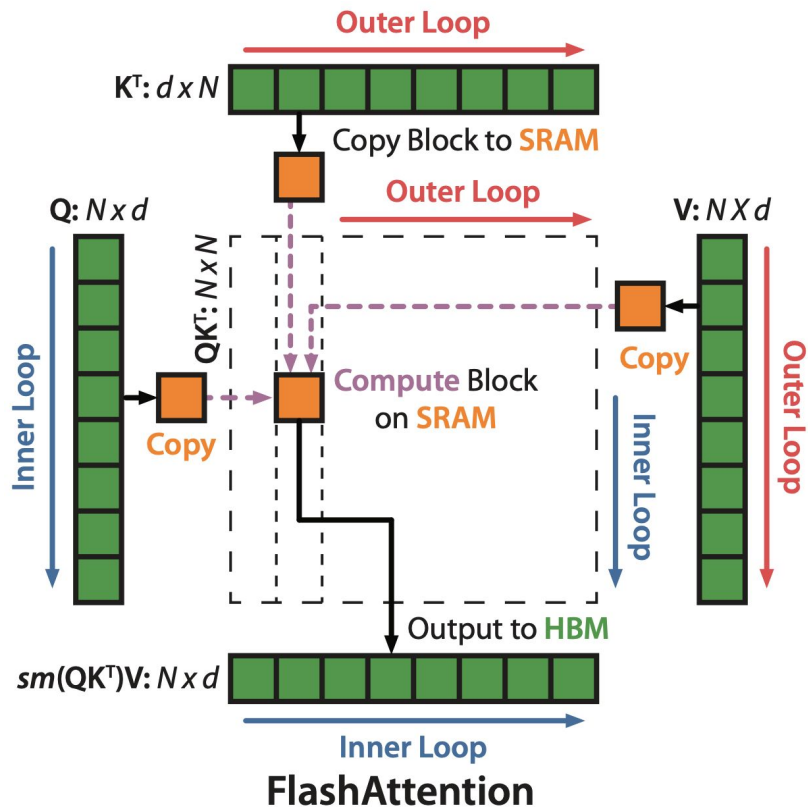
$$Z = Z + XY$$

$$Z = Z + XY$$

$$C_{00} = Z$$

# Tiling (cont'd)

1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output with respect to that block.
3. Update output in HBM by scaling.



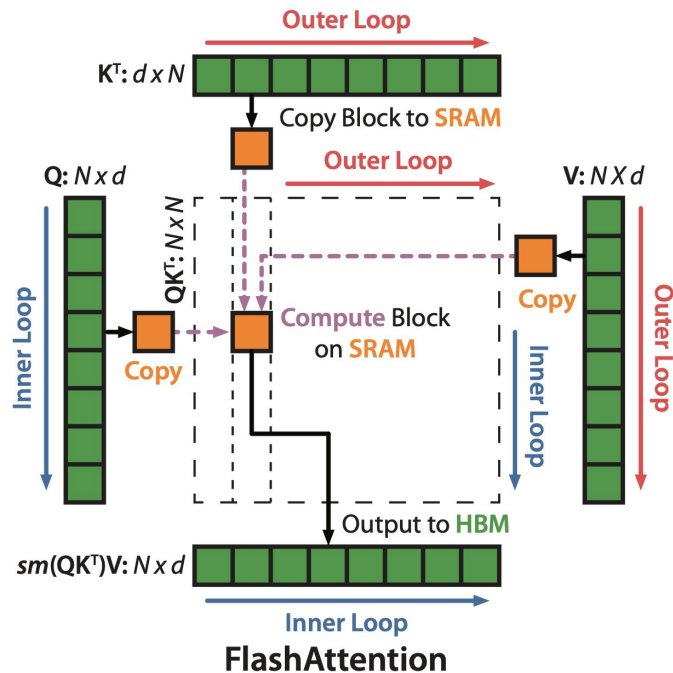
# Demo

- <https://jacksoncakes.com/flashattention-fast-and-memory-efficient-exact-attention/>

# Recomputation (backward pass)

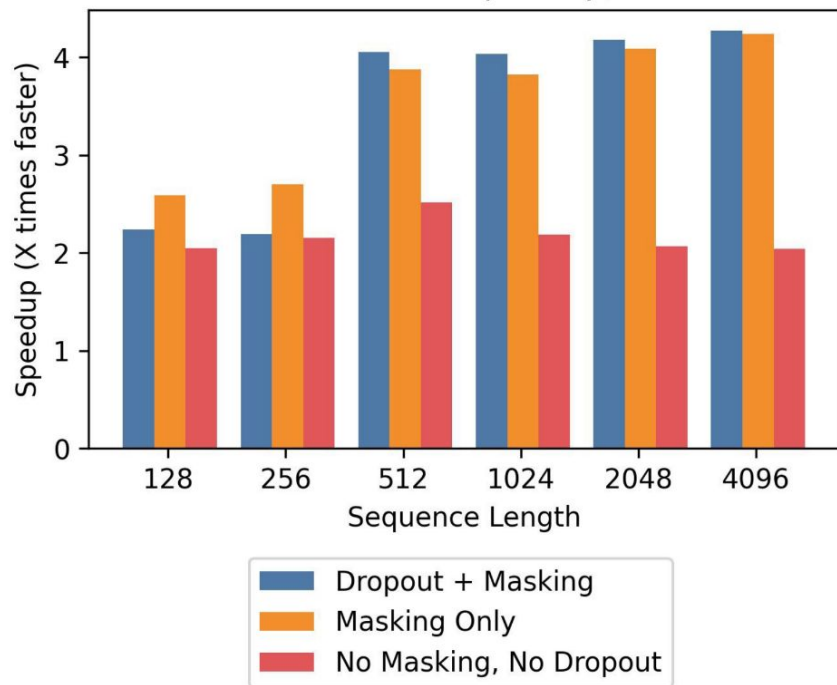
- By storing softmax normalization from forward (size  $N$ ), quickly recompute attention in the backward from inputs in SRAM.

Attention	Standard	FlashAttention
GFLOPs	66.6	75.2 (↑13%)
HBM reads/writes (GB)	40.3	4.4 (↓9x)
Runtime (ms)	41.7	7.3 (↓6x)

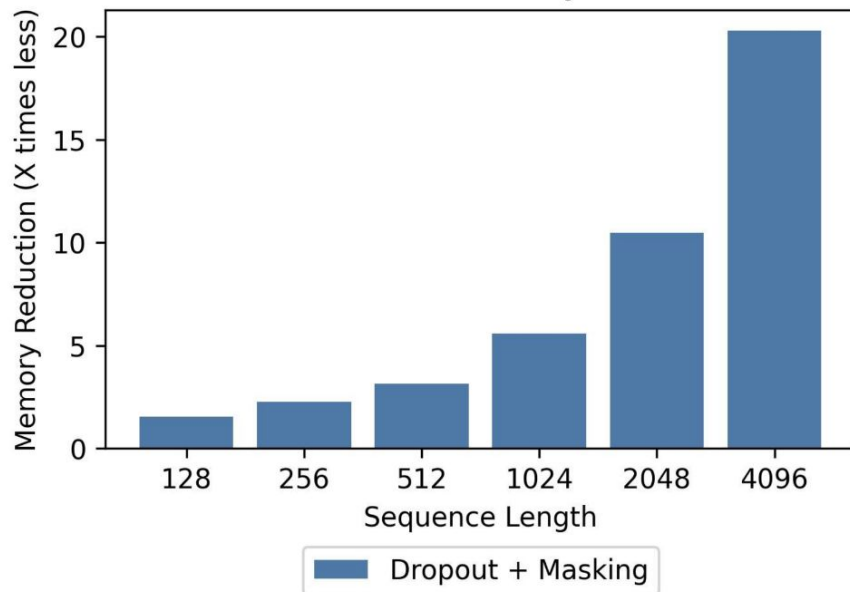


# FlashAttention: 2-4x speedup, 10-20x memory reduction

FlashAttention Speedup, A100



FlashAttention Memory Reduction



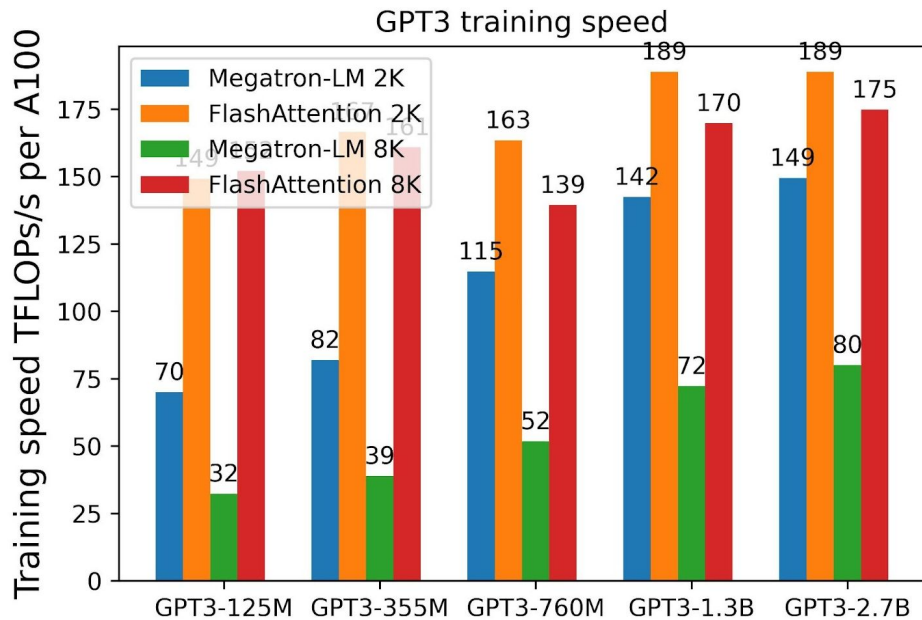
# Faster Training: MLPerf Record for Training BERT-large

- MLPerf: (highly optimized) standard benchmark for training speed
- Time to hit an accuracy of 72.0% on MLM from a fixed checkpoint, averaged across 10 runs on 8 x A100 GPUs

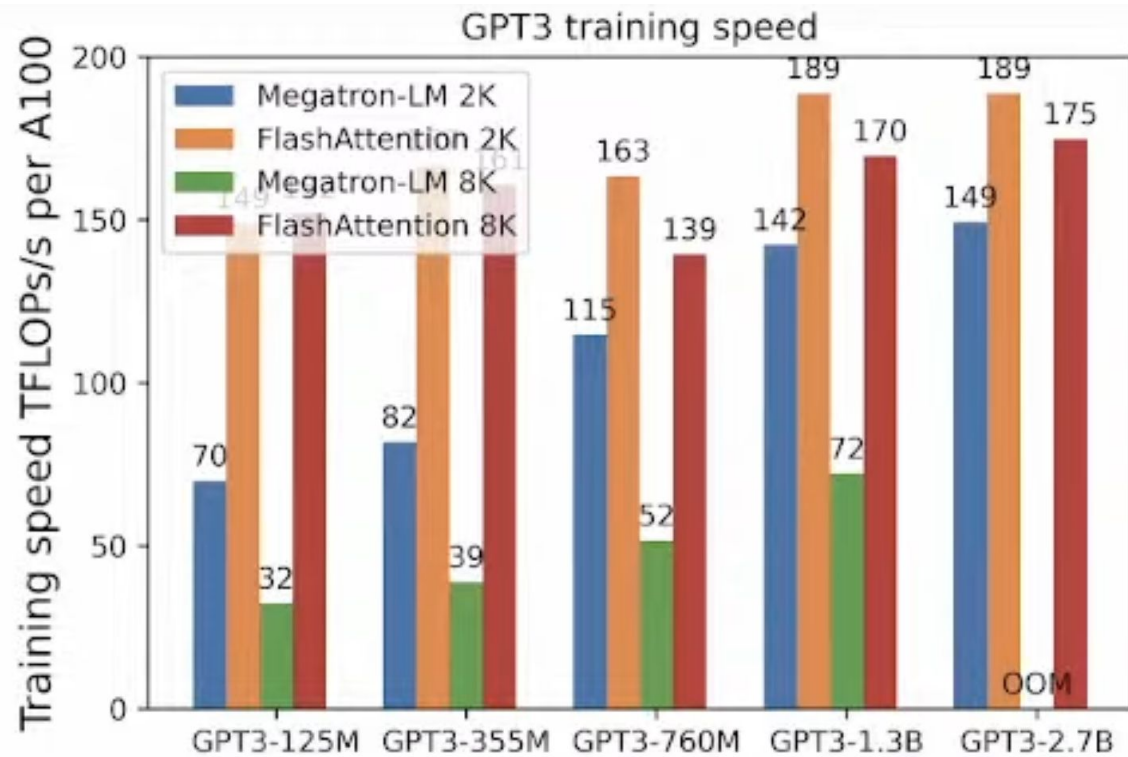
BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [58]	20.0 $\pm$ 1.5
FLASHATTENTION (ours)	<b>17.4 <math>\pm</math> 1.4</b>



# Faster Training, longer context



# Faster Training, longer context



**Thank you!**