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

Tri Dao†, Daniel Y. Fu , Stefano Ermon
Atri Rudra , Christopher Ré

# Contribution

- Fast Transformer training

  Train BERT-large (seq. length 512) 15% faster than the training speed record in MLPerf 1.1, GPT2 (seq. length 1K) 3x faster than baseline implementations from HuggingFace and Megatron-LM, and long-range arena (seq. length 1K-4K) 2.4x faster than baselines.

- Memory efficient

  compared to vanilla attention, which is quadratic in sequence length, $O(N^2)$, this method is sub-quadratic/linear in N ($O(N)$).

- Exact but also extended to block-sparse (Approximate) attention
- IO Aware

# Background: Language Model

Sequence to Sequence Model

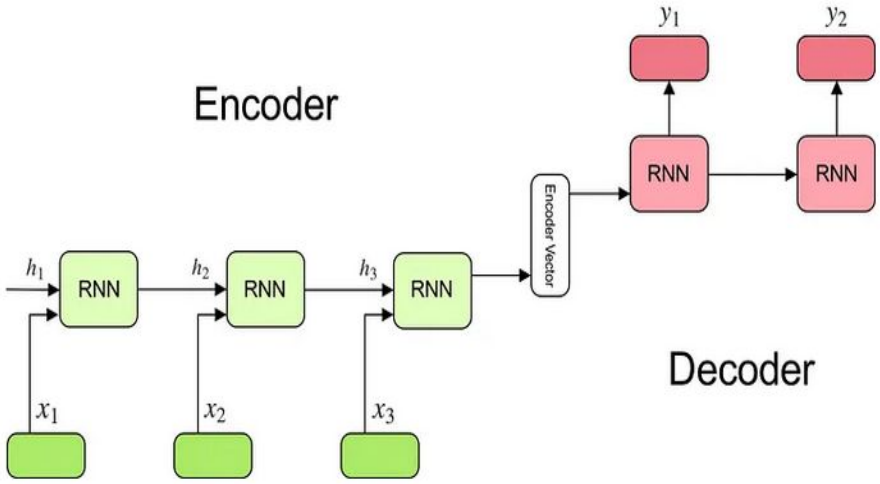
– Encoder Decoder Architecture

– RNN, LSTM

- Problem

  - Short term memory

  - Sequential Execution




Encoder-decoder sequence to sequence model

**Don't** eat the delicious looking and smelling pizza.

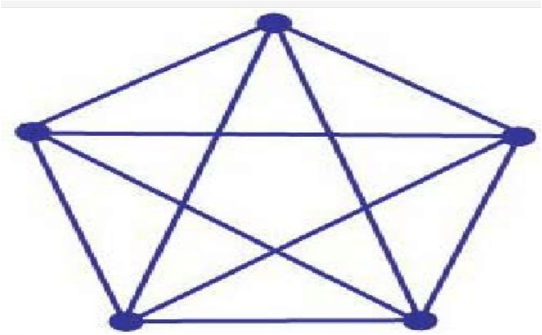
Eat the delicious looking and smelling pizza.

# Background: Transformer

- Attention Mechanism

**– Keeps track how similar each word is to all of the words in the sentences including itself**

**– Calculate similarity score between each pair of words**

# Background: Transformer

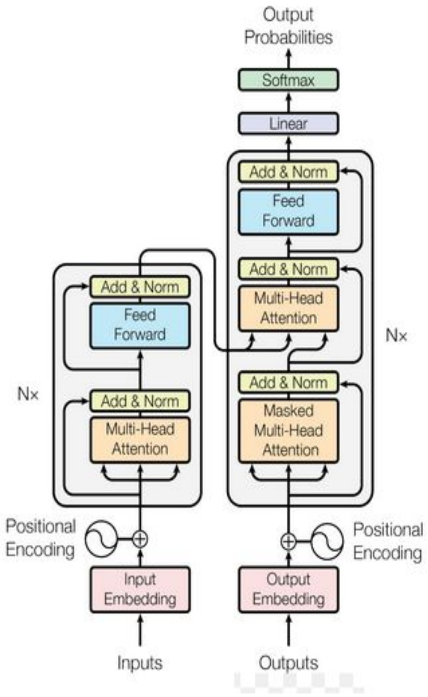
```
inputs:
tensor([[47, 58,  1, 51, 59, 57, 58,  1],
        [ 0, 24, 43, 58,  1, 46, 47, 51],
        [41, 53, 51, 43, 11,  0, 13, 52],
        [59,  1, 50, 53, 53, 49,  5, 57]])
torch.Size([4, 8])
targets:
tensor([[58,  1, 51, 59, 57, 58,  1, 40],
        [24, 43, 58,  1, 46, 47, 51,  1],
        [53, 51, 43, 11,  0, 13, 52, 42],
        [ 1, 50, 53, 53, 49,  5, 57, 58]])
torch.Size([4, 8])
```

```
when input is [47] the target: 58
when input is [47, 58] the target: 1
when input is [47, 58, 1] the target: 51
when input is [47, 58, 1, 51] the target: 59
when input is [47, 58, 1, 51, 59] the target: 57
when input is [47, 58, 1, 51, 59, 57] the target: 58
when input is [47, 58, 1, 51, 59, 57, 58] the target: 1
when input is [47, 58, 1, 51, 59, 57, 58, 1] the target: 40
when input is [0] the target: 24
when input is [0, 24] the target: 43
when input is [0, 24, 43] the target: 58
when input is [0, 24, 43, 58] the target: 1
when input is [0, 24, 43, 58, 1] the target: 46
when input is [0, 24, 43, 58, 1, 46] the target: 47
when input is [0, 24, 43, 58, 1, 46, 47] the target: 51
when input is [0, 24, 43, 58, 1, 46, 47, 51] the target: 1
```

# Background: Transformer

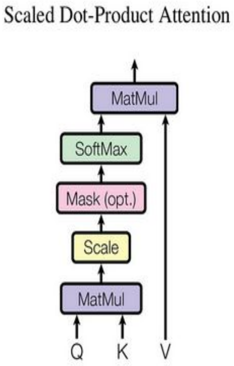
**BERT**

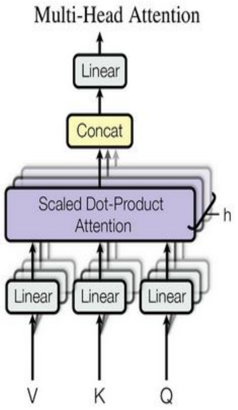
Encoder




**GPT**

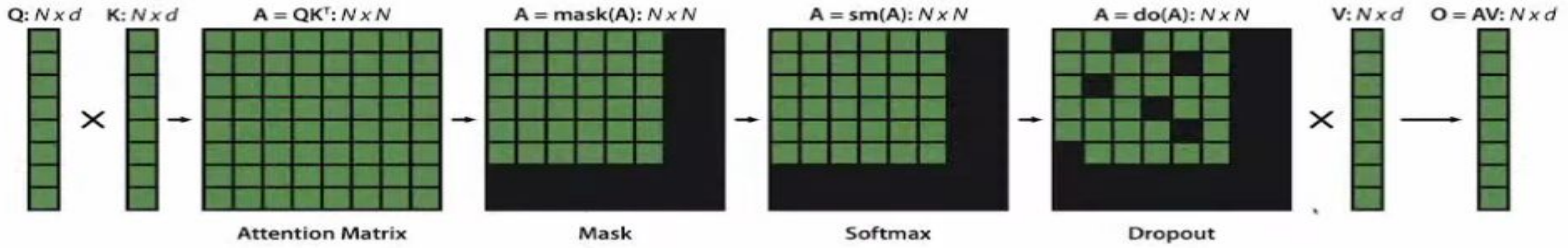
Decoder



# Background: Attention mechanism

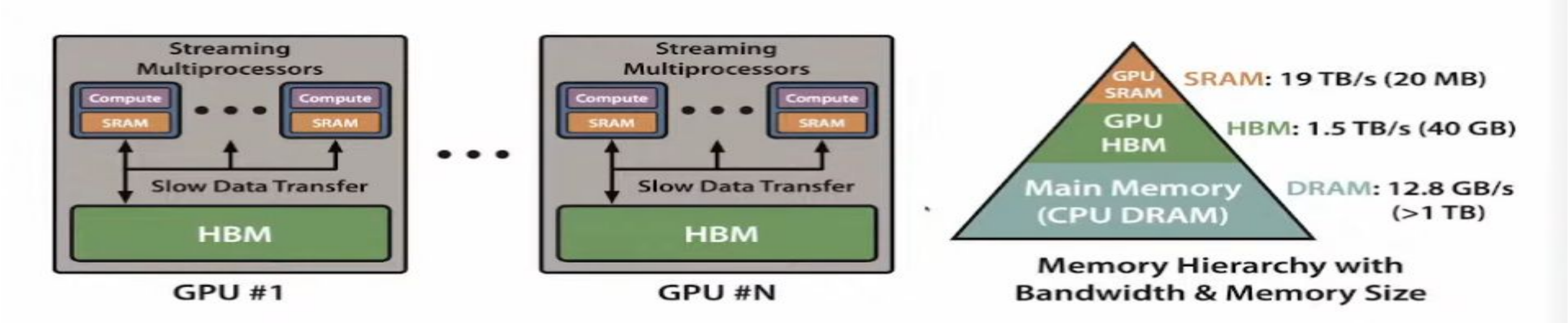
- **Quadratic** in Memory and Time
- **Challenge**:  how to scale transformer to **longer sequence**
- Attention is bottlenecked by **memory read and write**

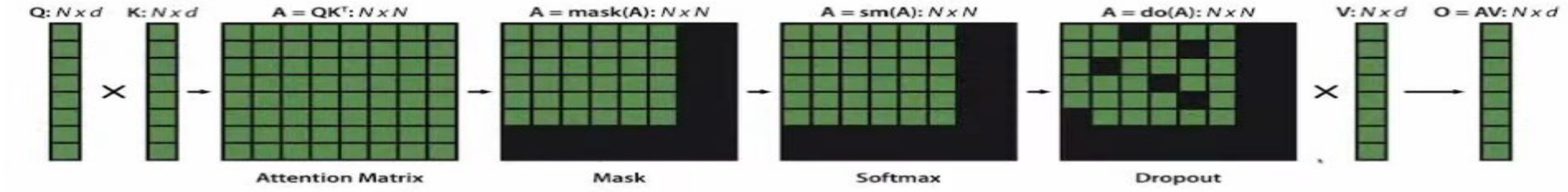



$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK^T})))\mathbf{V}$$

# Background: Attention mechanism

- Naive implementation requires repeated read and write from slow GPU HBM memory. Hard to scale for longer sequence.

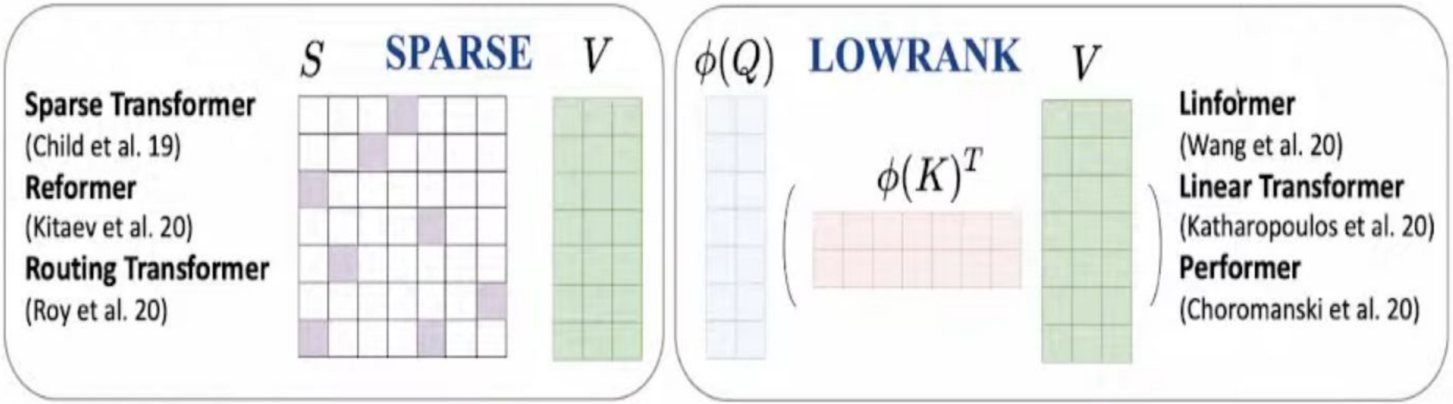



$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK^T})))\mathbf{V}$$

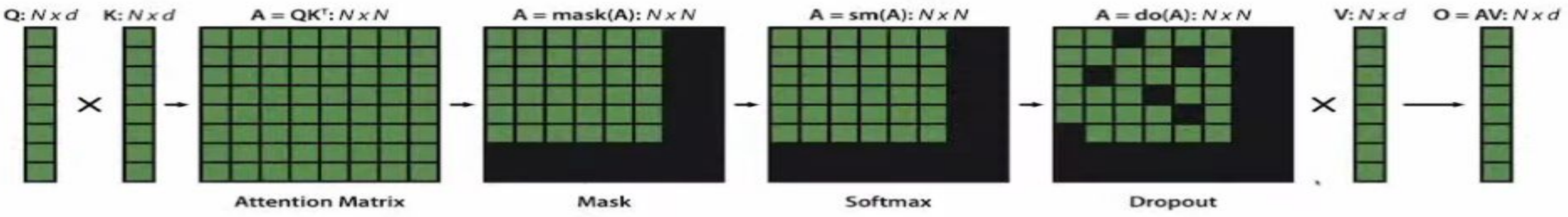
# Background: Approximate Attention

- **No training time improvement in Approximate Attention mechanism**



# FlashAttention

- Exploits Memory Asymmetry with IO awareness
- It reduces HBM read/write by using **Tiling** mechanism, **block by block** loading from HBM memory to SRAM for computing attention
- It also does not store Attention Matrix for backpropagation rather **recompute**
- It does not **materialize** the whole Attention Matrix into **SRAM**
- But **Softmax** computation needs the **whole Attention Matrix**




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

# SoftMax: Online Normalizer Calculation

- Online normalizer calculation for softmax – paper by Nvidia
- Naive Softmax and safe version

$$y_i = \frac{e^{x_i}}{\sum\limits_{j=1}^{V} e^{x_j}} \qquad\qquad y_i = \frac{e^{x_i - \max\limits_{k=1}^{V} x_k}}{\sum\limits_{j=1}^{V} e^{x_j - \max\limits_{k=1}^{V} x_k}}$$

- Safe version — Online Version (reduce HBM R/W)

**Algorithm 2** Safe softmax

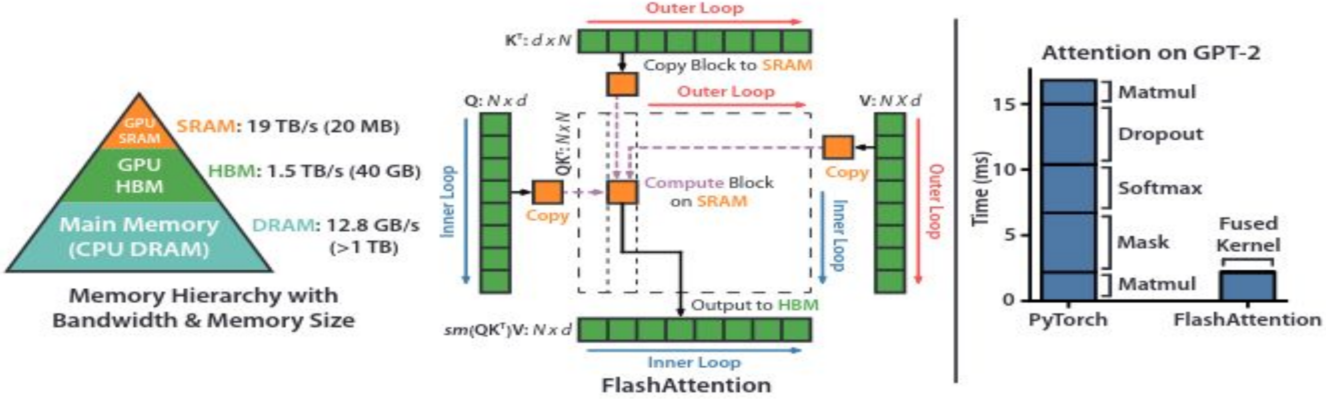
$m_0 \leftarrow -\infty$
**for** $k \leftarrow 1, V$ **do**
  $m_k \leftarrow \max(m_{k-1}, x_k)$
**end for**
$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
  $d_j \leftarrow d_{j-1} + e^{x_j - m_V}$
**end for**
**for** $i \leftarrow 1, V$ **do**
  $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
**end for**

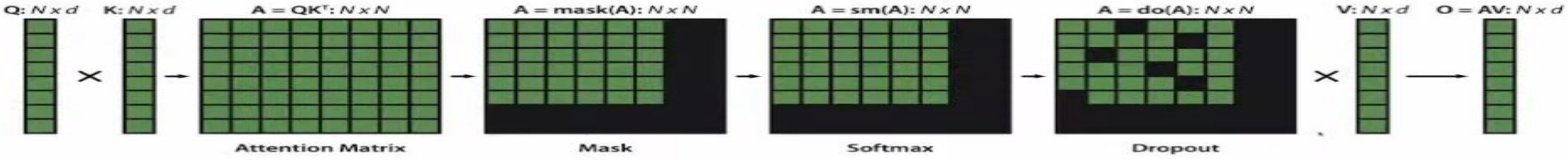
**Algorithm 3** Safe softmax with online normalizer calculation

$m_0 \leftarrow -\infty$
$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
  $m_j \leftarrow \max(m_{j-1}, x_j)$
  $d_j \leftarrow d_{j-1} \times e^{m_{j-1} - m_j} + e^{x_j - m_j}$
**end for**
**for** $i \leftarrow 1, V$ **do**
  $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
**end for**

# FlashAttention

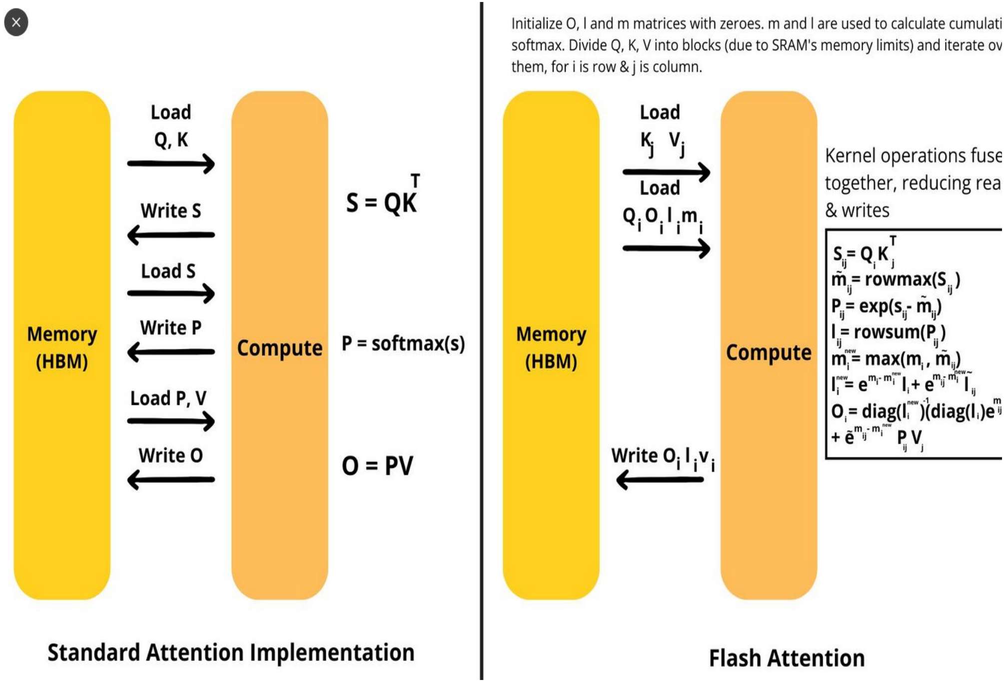
- Used Fused Kernel with Tiling

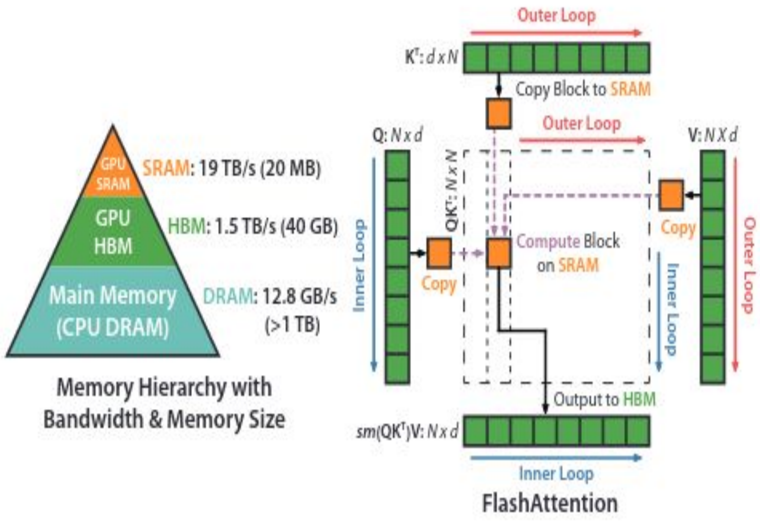



$$\mathbf{O} = \mathrm{Dropout}(\mathrm{Softmax}(\mathrm{Mask}(\mathbf{QK^T})))\mathbf{V}$$

# FlashAttention



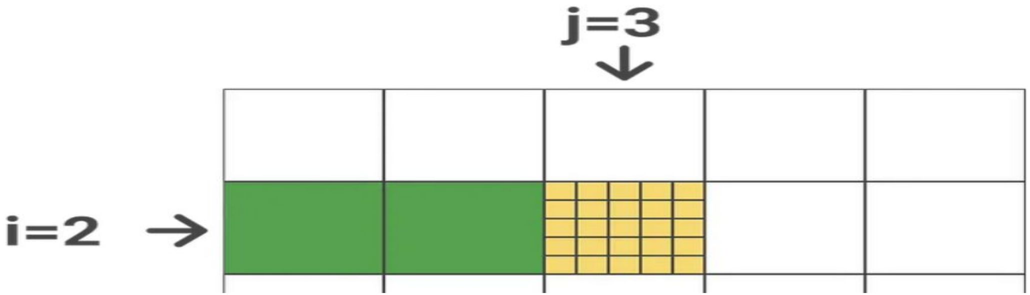
# FlashAttention

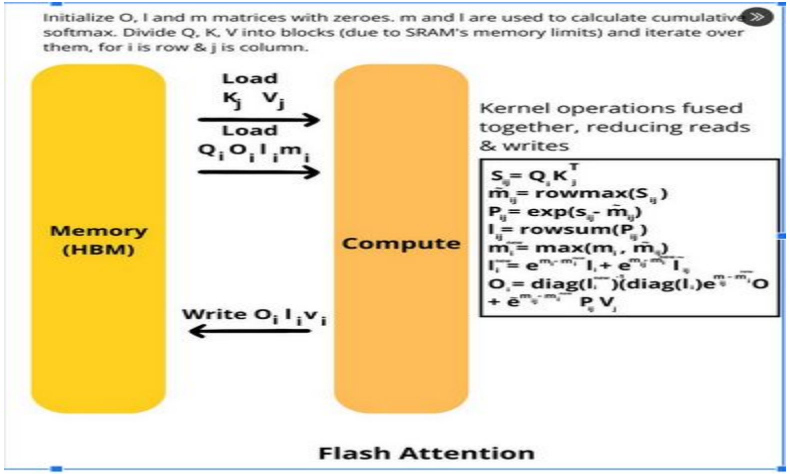
## Softmax calculation

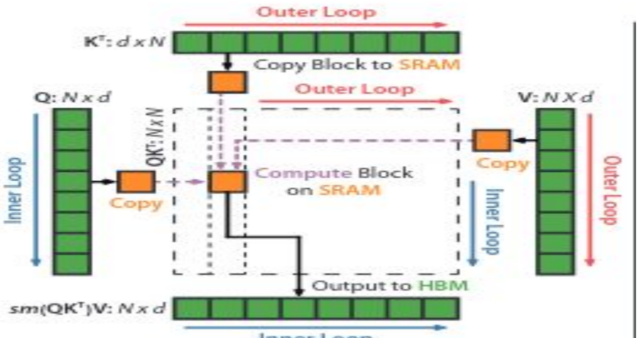
On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$

$$\text{Write } \mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}\left(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j\right)$$

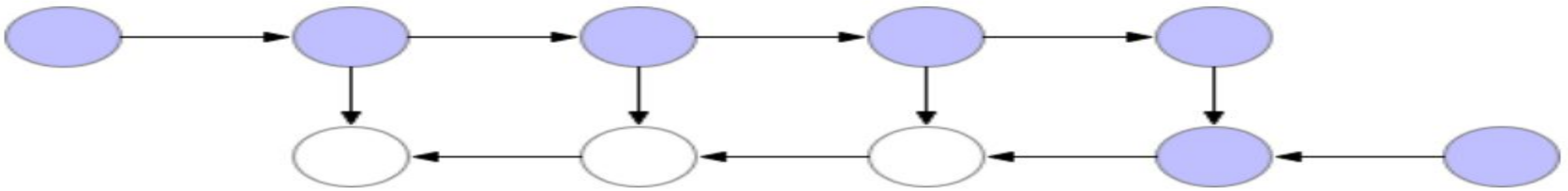


# FlashAttention

- It uses recomputation during backpropagation



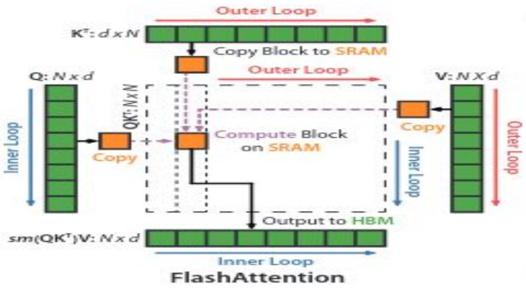
Graph 1. *Vanilla backprop*

**Algorithm 3** Standard Attention Backward Pass

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

1: Load $\mathbf{P}, \mathbf{dO}$ by blocks from HBM, compute $\mathbf{dV} = \mathbf{P}^\top \mathbf{dO} \in \mathbb{R}^{N \times d}$, write $\mathbf{dV}$ to HBM.

2: Load $\mathbf{dO}, \mathbf{V}$ by blocks from HBM, compute $\mathbf{dP} = \mathbf{dO}\mathbf{V}^\top \in \mathbb{R}^{N \times N}$, write $\mathbf{dP}$ to HBM.




**Algorithm 4** FlashAttention Backward Pass

**Require:** Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$ in HBM, vectors $\ell, m \in \mathbb{R}^N$ in HBM, on-chip SRAM of size $M$, softmax scaling constant $\tau \in \mathbb{R}$, masking function MASK, dropout probability $p_{\text{drop}}$, pseudo-random number generator state $\mathcal{R}$ from the forward pass.

# FlashAttention: Memory Efficiency and IO Awareness

- Similar work from Google 1 year ago but that was not IO Aware. so Flash Attention has better performance gain

SELF-ATTENTION DOES NOT NEED $O(n^2)$ MEMORY

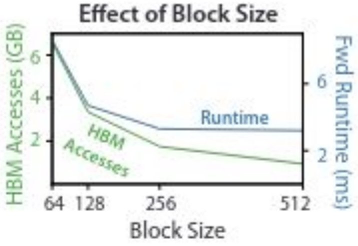
A PREPRINT

**Markus N. Rabe and Charles Staats**
Google Research
{mrabe,cstaats}@google.com

- Uses online softmax, tilling, gradient checkpoint during backpropagation
- Perspective was showing Attention does not need quadratic memory, so their Implementation was not IO aware

# Analysis: IO Complexity of FlashAttention

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



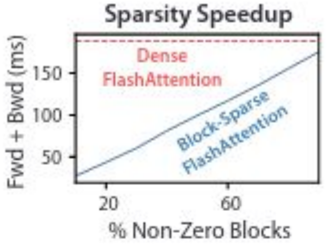




Figure 2: **Left**: Forward + backward runtime of standard attention and FlashAttention for GPT-2 medium (seq. length 1024, head dim. 64, 16 heads, batch size 64) on A100 GPU. HBM access is the primary factor affecting runtime. **Middle**: Forward runtime of FlashAttention (seq. length 1024, head dim. 64, 16 heads, batch size 64) on A100 GPU. Fewer HBM accesses result in faster runtime, up to a point. **Right**: The runtime (for seq. length 4K) of block-sparse FlashAttention is faster than FlashAttention by a factor proportional to the sparsity.

# Faster Models with FlashAttention

Table 1: Training time of BERT-large, starting from the same initialization provided by the MLPerf benchmark, to reach the target accuracy of 72.0% on masked language modeling. Averaged over 10 runs on 8×A100 GPUs.

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

Table 2: GPT-2 small and medium using FlashAttention achieve up to 3× speed up compared to Huggingface implementation and up to 1.7× compared to Megatron-LM. Training time reported on 8×A100s GPUs.

| 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×)** |

# Faster Models with FlashAttention

Table 3: The performance of standard attention, FlashAttention, block-sparse FlashAttention, and approximate attention baselines on the Long-Range-Arena benchmarks.

| 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**× |
| Linformer [84] | 35.6 | 55.9 | 77.7 | 37.8 | 67.6 | 54.9 | 2.5× |
| Linear Attention [50] | 38.8 | 63.2 | 80.7 | 42.6 | 72.5 | 59.6 | 2.3× |
| Performer [12] | 36.8 | 63.6 | 82.2 | 42.1 | 69.9 | 58.9 | 1.8× |
| Local Attention [80] | 36.1 | 60.2 | 76.7 | 40.6 | 66.6 | 56.0 | 1.7× |
| Reformer [51] | 36.5 | 63.8 | 78.5 | 39.6 | 69.4 | 57.6 | 1.3× |
| Smyrf [19] | 36.1 | 64.1 | 79.0 | 39.6 | 70.5 | 57.9 | 1.7× |

# Better Models with Longer Sequences

Table 4: GPT-2 small with FlashAttention, with 4× larger context length compared to Megatron-LM, is still 30% faster while achieving 0.7 better perplexity. Training time on 8×A100 GPUs is reported.

| 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×) |

# Runtime and Memory Usage

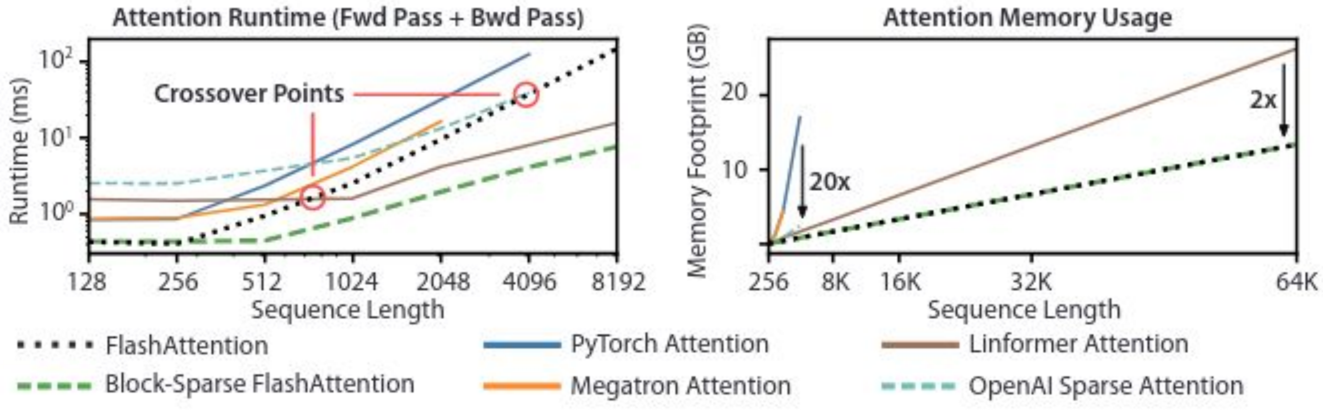



Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

Thanks