FlashAttention

Александр Крапухин, АИРИ.

Outline

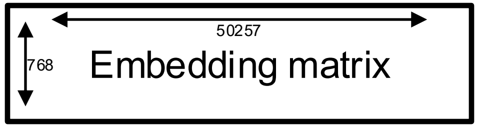
- Background: Transformer, Attention, Softmax, GPUs.
- FlashAttention.
- Results.

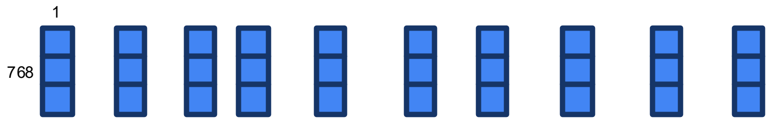
Transformer Architecture (GPT-2)

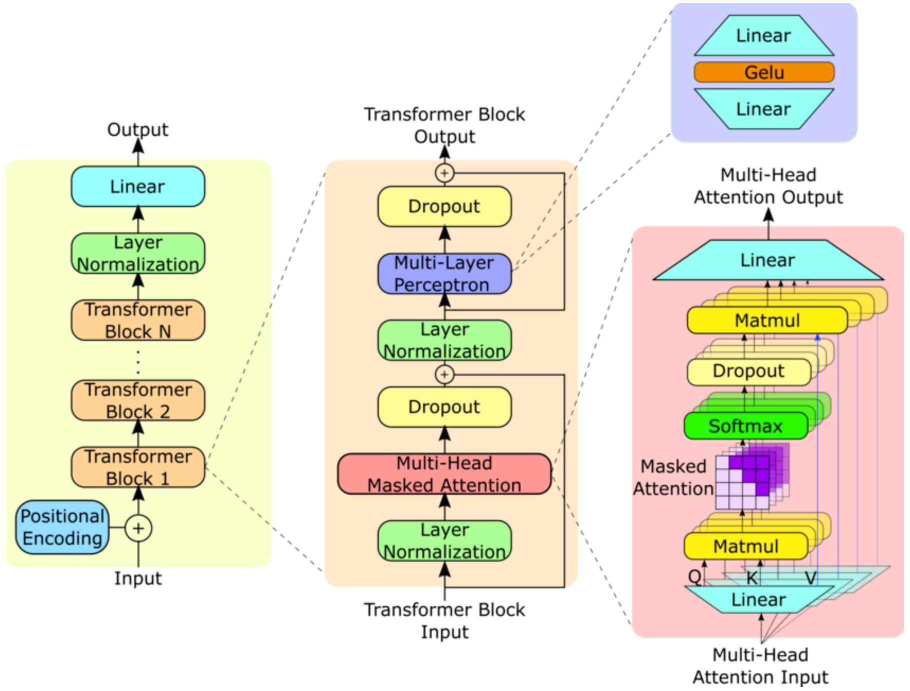
Today is a beautiful day outside

[To, day, is, a, beaut, iful, day, out, side, .]

[98, 1452, 43, 15, 2932, 1709, 740, 1452, 3112, 3823]





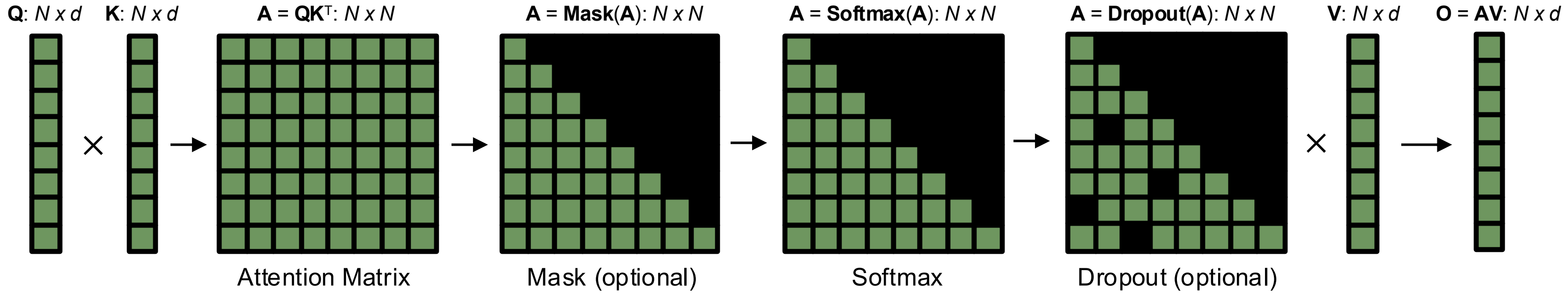


Transformer block: communication + computation

Attention

Inputs: **Q, K, V**
O = Dropout(Softmax(Mask($\mathbf{QK}^T$)))**V**

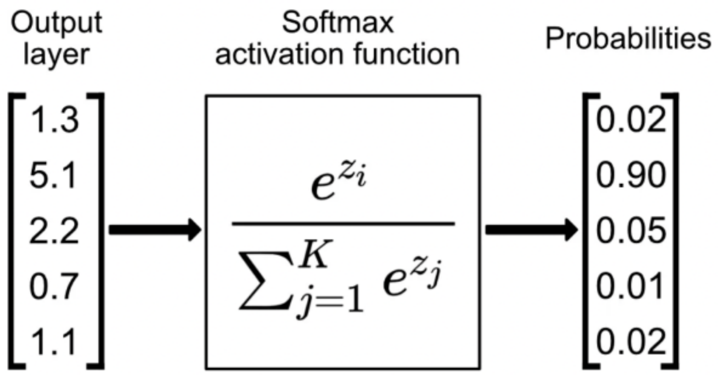
Q (queries) – what am I looking for?
K (keys) – what do I contain?
V (values) – if you find me interesting, here is what I will communicate to you



Q: *N x d*
K: *N x d*
A = **QK**^{T}: *N x N*
A = **Mask**(**A**): *N x N*
A = **Softmax**(**A**): *N x N*
A = **Dropout**(**A**): *N x N*
V: *N x d*
O = **AV**: *N x d*
Attention Matrix
Mask (optional)
Softmax
Dropout (optional)

- Naively, attention has compute & memory quadratic in sequence length N
- N: 1K, 2K
- d: 64, 128

SoftMax


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

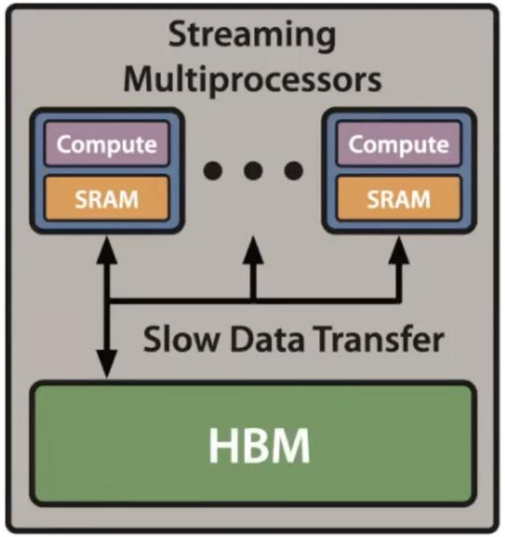
- On real hardware the range of numbers is limited.
- The sum can overflow or underflow.

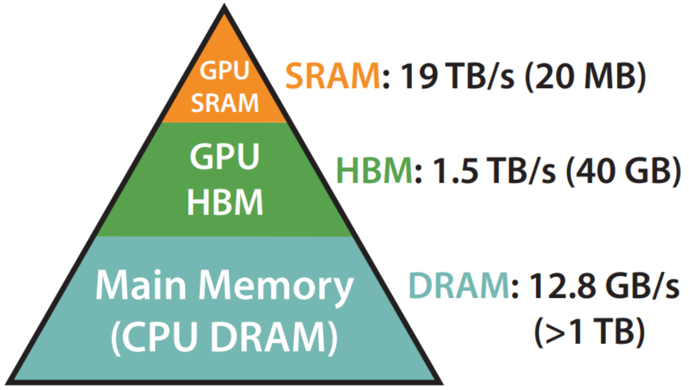
$$\text{softmax}(z)_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_k}}$$

$$\big\downarrow$$

$$\text{softmax}(z)_i = \frac{e^{(z_i - z_{max})}}{e^{(z_1 - z_{max})} + e^{(z_2 - z_{max})} + \cdots + e^{(z_k - z_{max})}}$$

GPU Compute Model & Memory Hierarchy

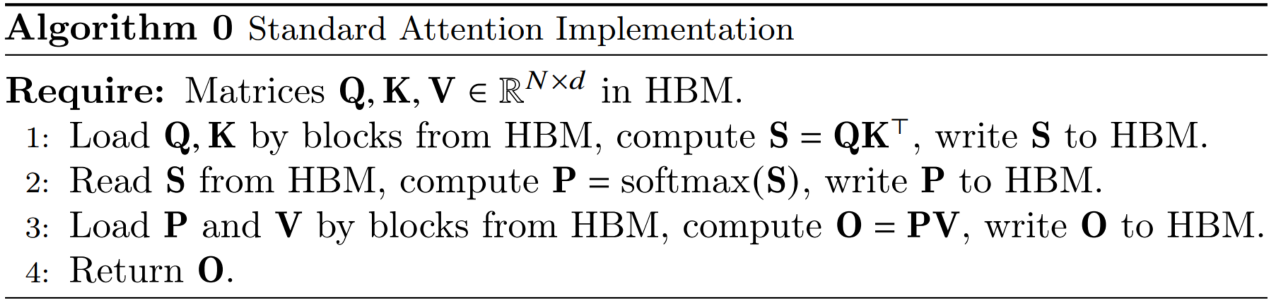




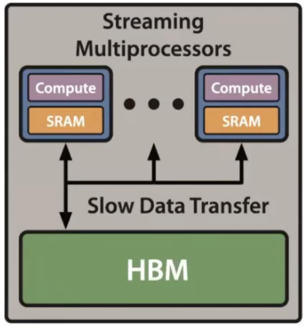
Memory Hierarchy with Bandwidth & Memory Size

SRAM – static random-access memory
HBM – high bandwidth memory

Standard Attention implementation



```
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 $\mathbf{O}$.
```



- Slow: most of time is spent on HBM reads/writes
- Quadratic memory complexity

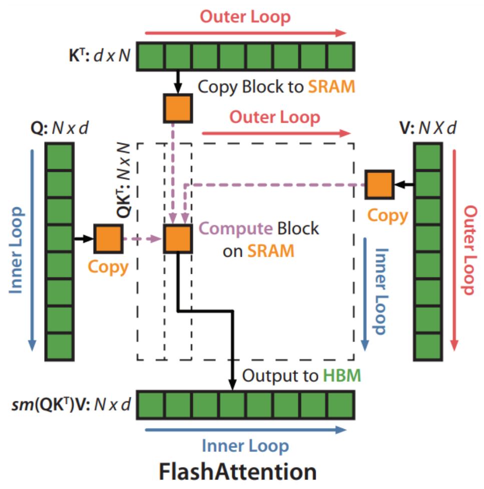
FlashAttention

- Reduce HBM reads/writes.
- Compute attention by blocks (tiling).
- Problem: Softmax need the whole row. Solution: rescaling.

$$y_i = \frac{e^{x_i - \max_{k=1}^{V} x_k}}{\sum_{j=1}^{V} e^{x_j - \max_{k=1}^{V} x_k}}$$

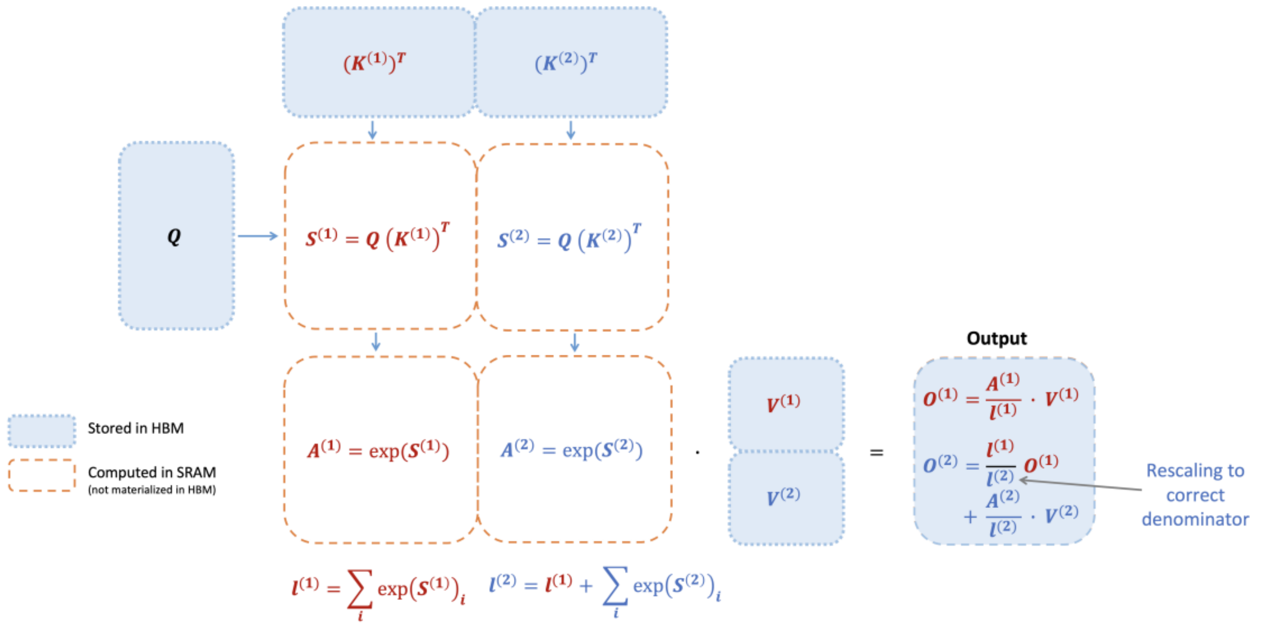
Algorithm:

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



Dao et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. NeurIPS 2022.

Online softmax in FlashAttention



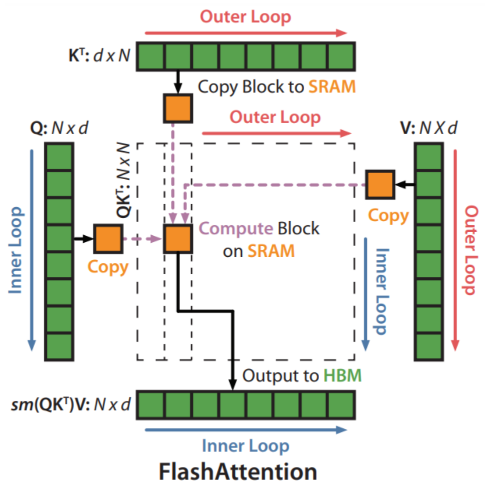
$(K^{(1)})^T$
$(K^{(2)})^T$
Q
$S^{(1)} = Q\,(K^{(1)})^T$
$S^{(2)} = Q\,(K^{(2)})^T$
$A^{(1)} = \exp(S^{(1)})$
$A^{(2)} = \exp(S^{(2)})$
$V^{(1)}$
$V^{(2)}$
Output
$O^{(1)} = \frac{A^{(1)}}{l^{(1)}} \cdot V^{(1)}$
$O^{(2)} = \frac{l^{(1)}}{l^{(2)}} O^{(1)} + \frac{A^{(2)}}{l^{(2)}} \cdot V^{(2)}$
Rescaling to correct denominator
Stored in HBM
Computed in SRAM (not materialized in HBM)
$l^{(1)} = \sum_i \exp(S^{(1)})_i$
$l^{(2)} = l^{(1)} + \sum_i \exp(S^{(2)})_i$

Figure 1: Diagram of how FlashAttention forward pass is performed, when the key **K** is partitioned into two blocks and the value **V** is also partitioned into two blocks. By computing attention with respect to each block and rescaling the output, we get the right answer at the end, while avoiding expensive memory reads/writes of the intermediate matrices **S** and **P**. We simplify the diagram, omitting the step in softmax that subtracts each element by the row-wise max.

FlashAttention Backward Pass

- Problem: backward pass needs Attention matrix (N by N).
- Solution: recompute it.
- Less memory and faster.

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



Attention module: 2-4x speedup

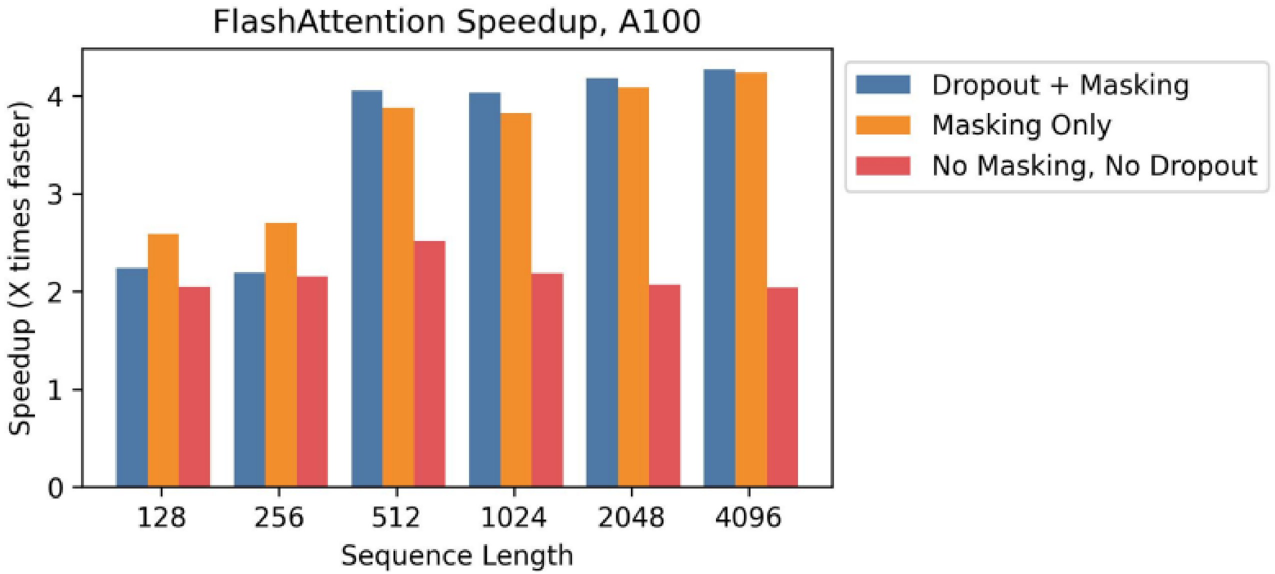
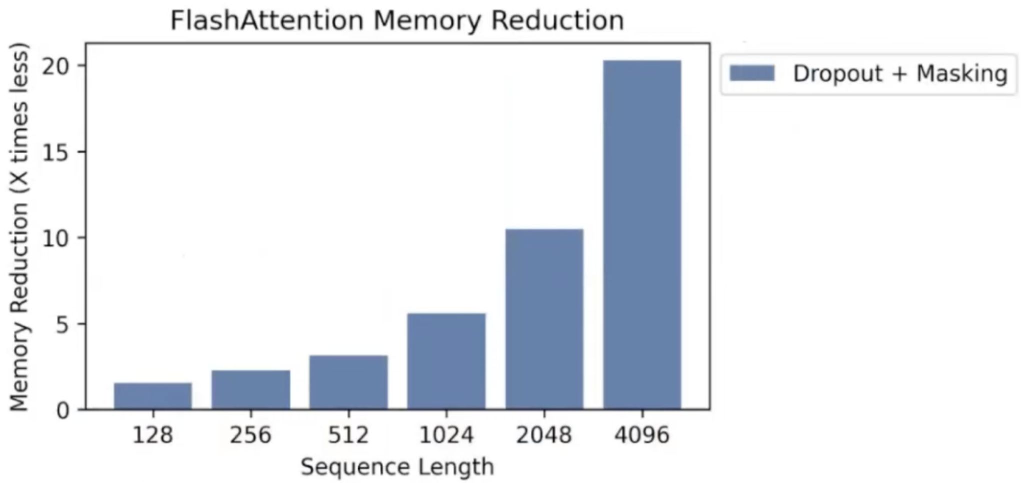


Figure 5: Speedup over standard PyTorch attention at different sequence lengths, on A100.

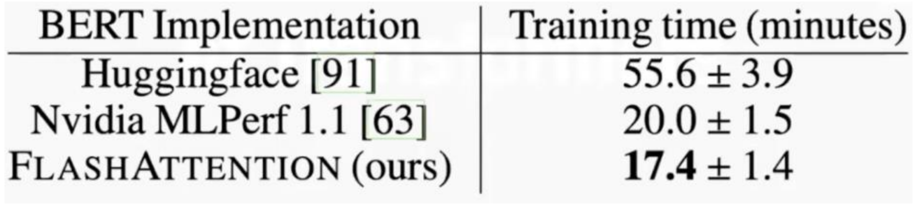
Attention module: 10-20x memory reduction



- Memory is linear in sequence length, not quadratic.

Faster Training (BERT Large)

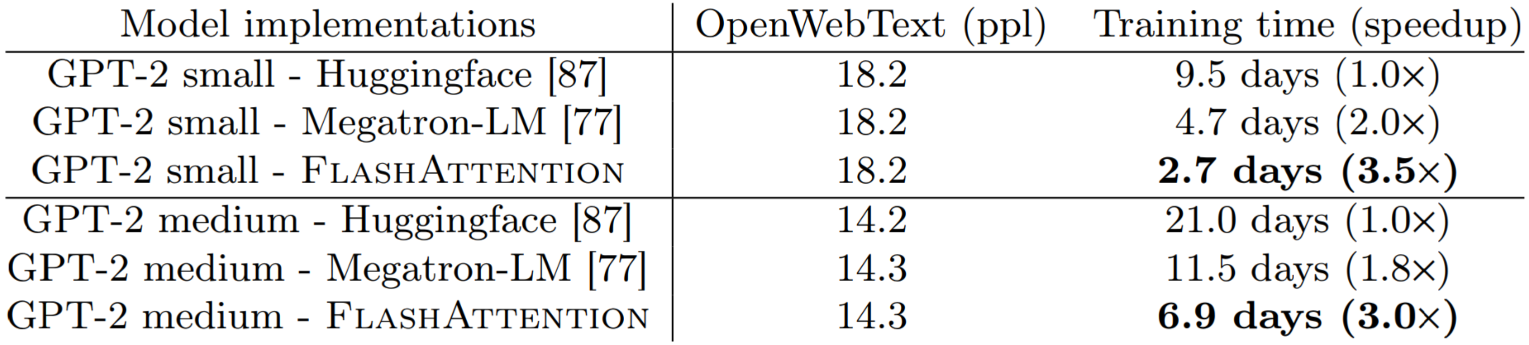
Training time to hit an accuracy of 72% on Masked Language Modelling, averaged across 10 runs on 8xA100 GPUs.



BERT Implementation	Training time (minutes)
Huggingface [91]	55.6 ± 3.9
Nvidia MLPerf 1.1 [63]	20.0 ± 1.5
FLASHATTENTION (ours)	$\mathbf{17.4} \pm 1.4$

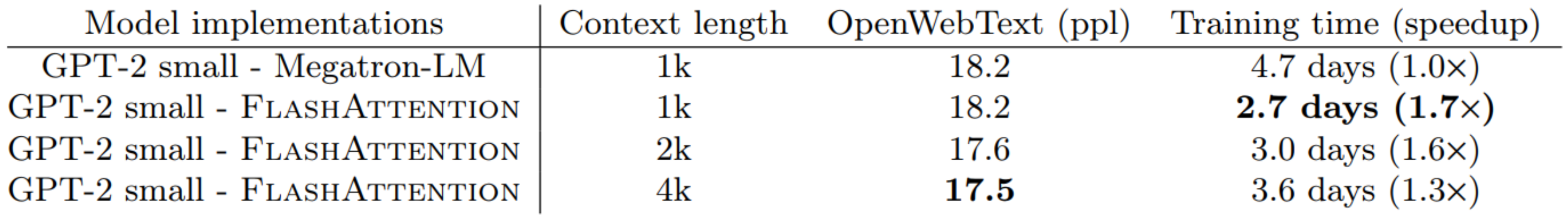
Faster Training (GPT-2)

Training GPT-2 from scratch on OpenWebText using 8xA100 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×)**

Better Models With Longer Sequences



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×)

Conclusion

FlashAttention:

- Key idea: reduce HBM reads/writes - tiling, recomputation.
- Faster model training.
- Better models with longer sequences.