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

Tri Dao
https://tridao.me

Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

Find the bug with this code:

```
for (var i = 0; i < 5; i++) {
  setTimeout(() => console.log(i), 1000)
}
```

The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

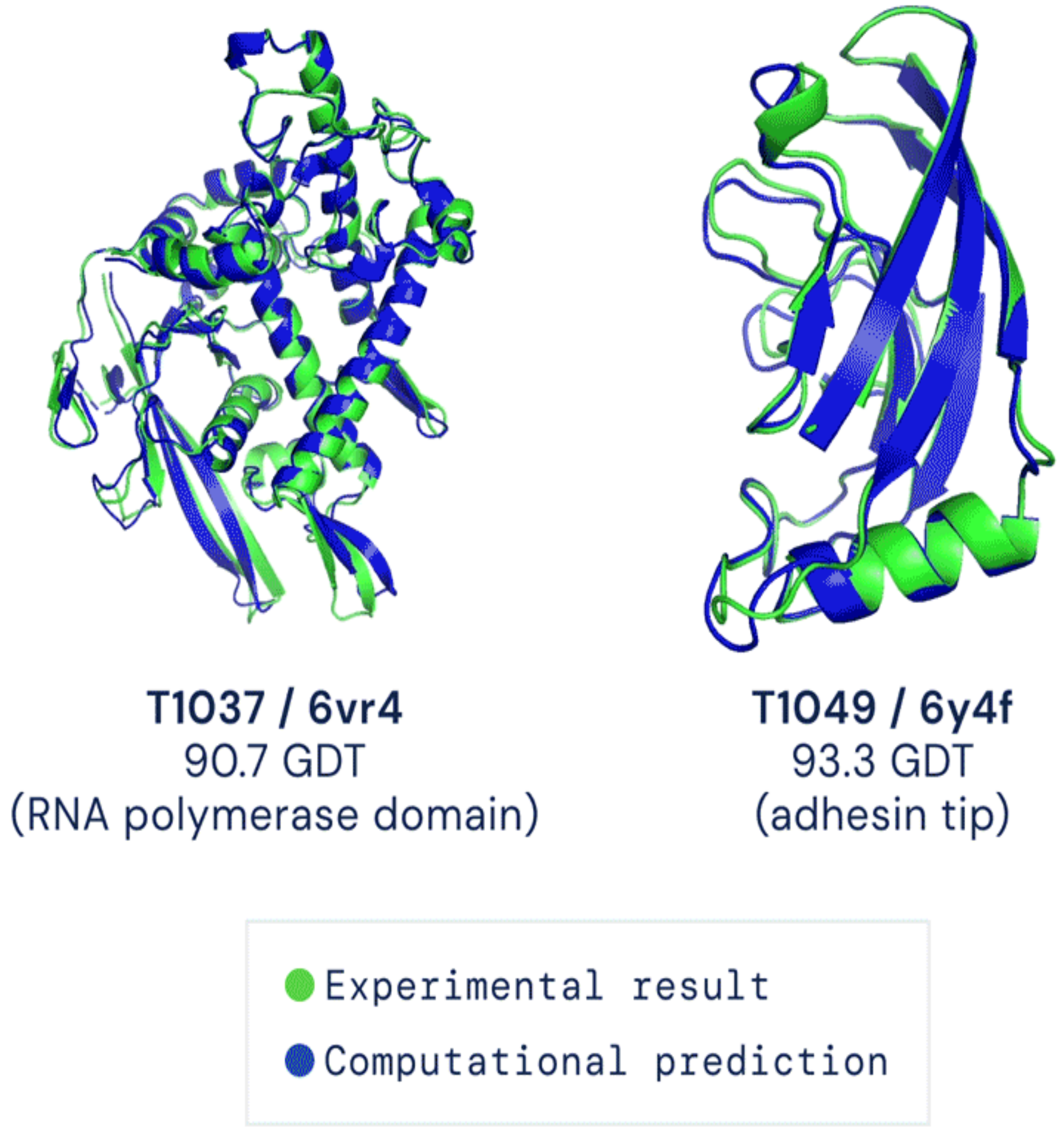
To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

Copy code

```
for (var i = 0; i < 5; i++) {
  // Create a new variable and pass its value to setTimeout
  const value = i;
  setTimeout(() => console.log(value), 1000)
}
```

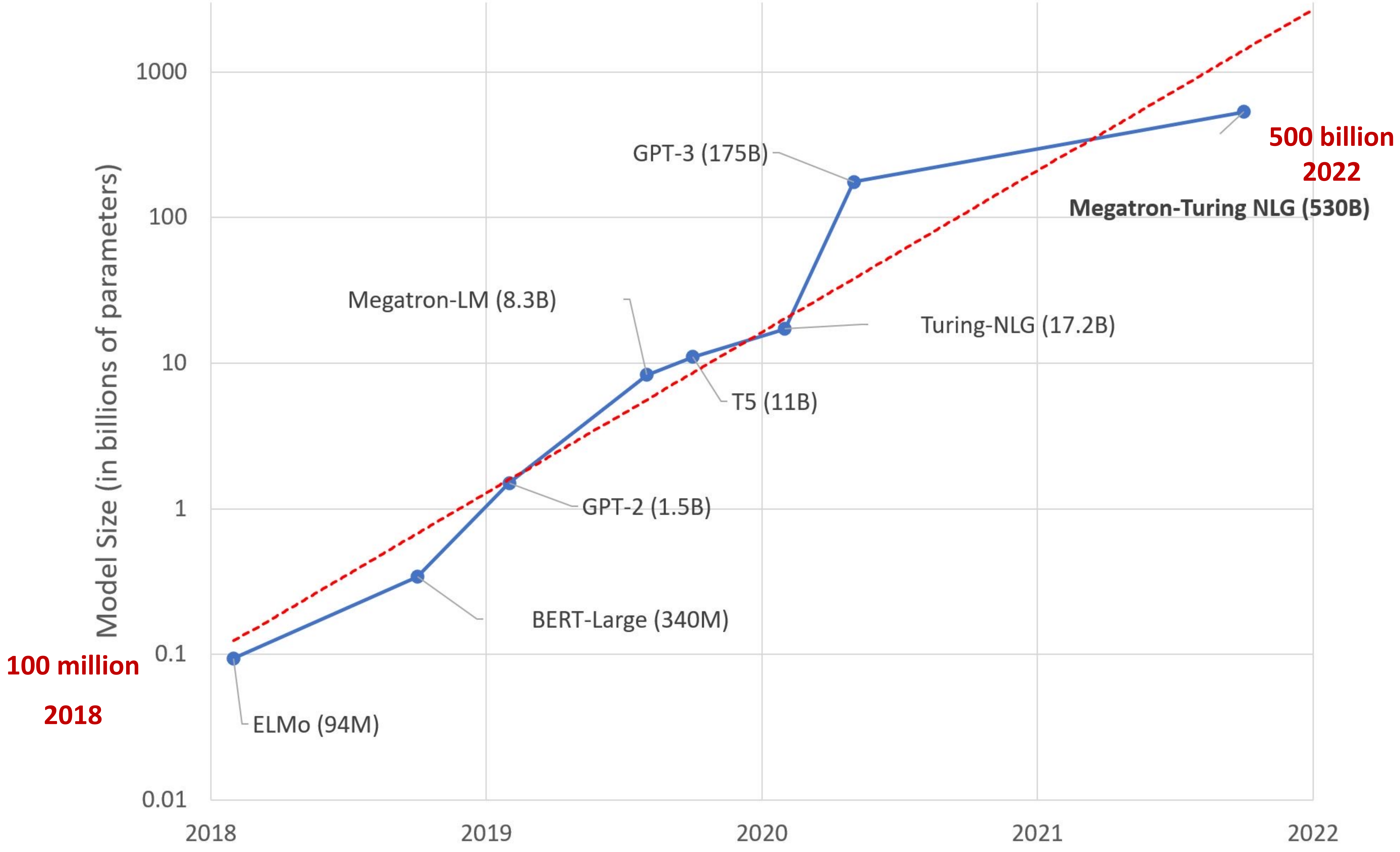
Generate Art
(Stable Diffusion – Stability.AI)

Design Drugs
(AlphaFold – DeepMind)



What enabled these advances? What are outstanding problems? How do we approach them?

Scale Brings Quality and Capabilities



Language models explaining jokes

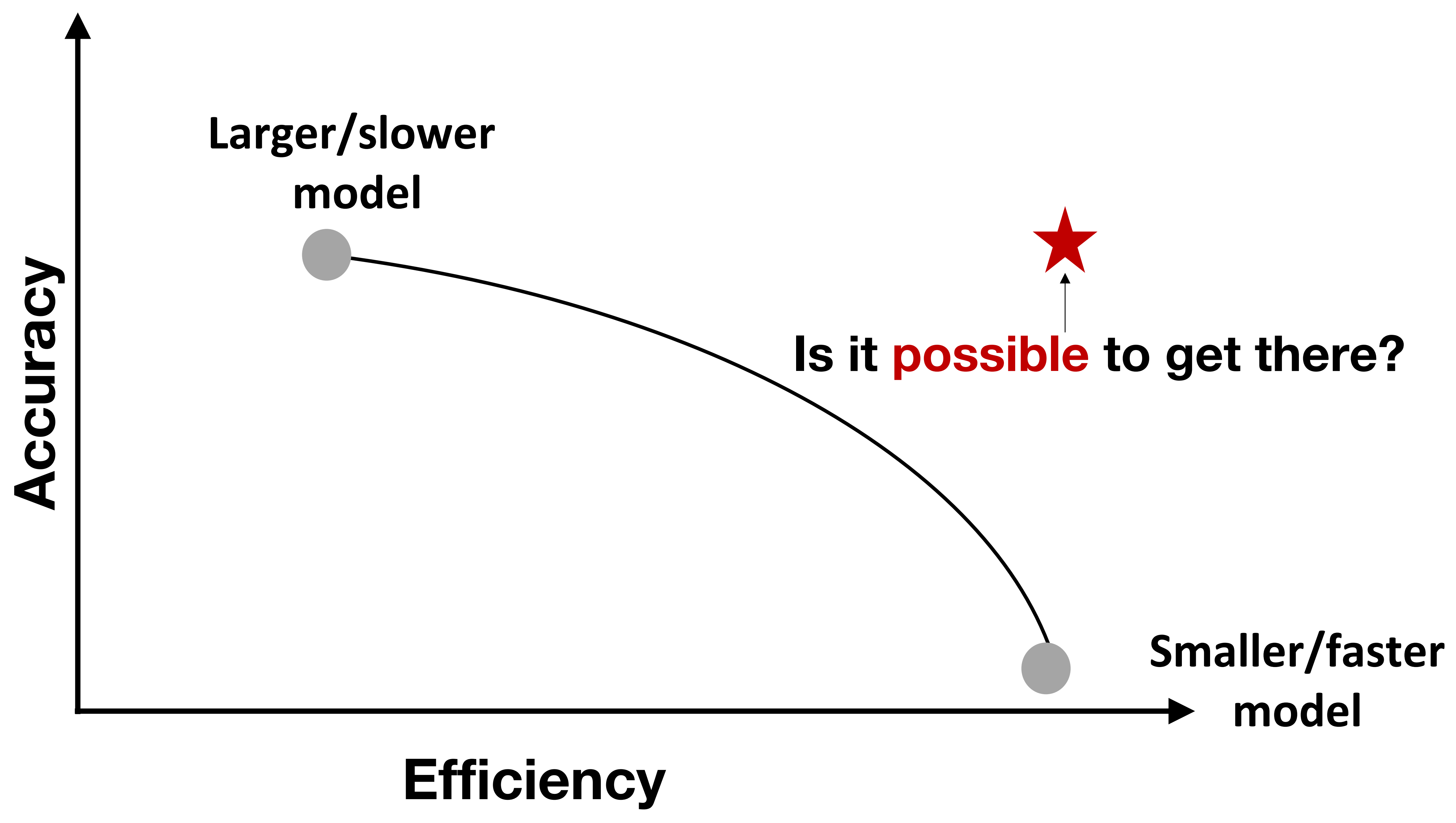
Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

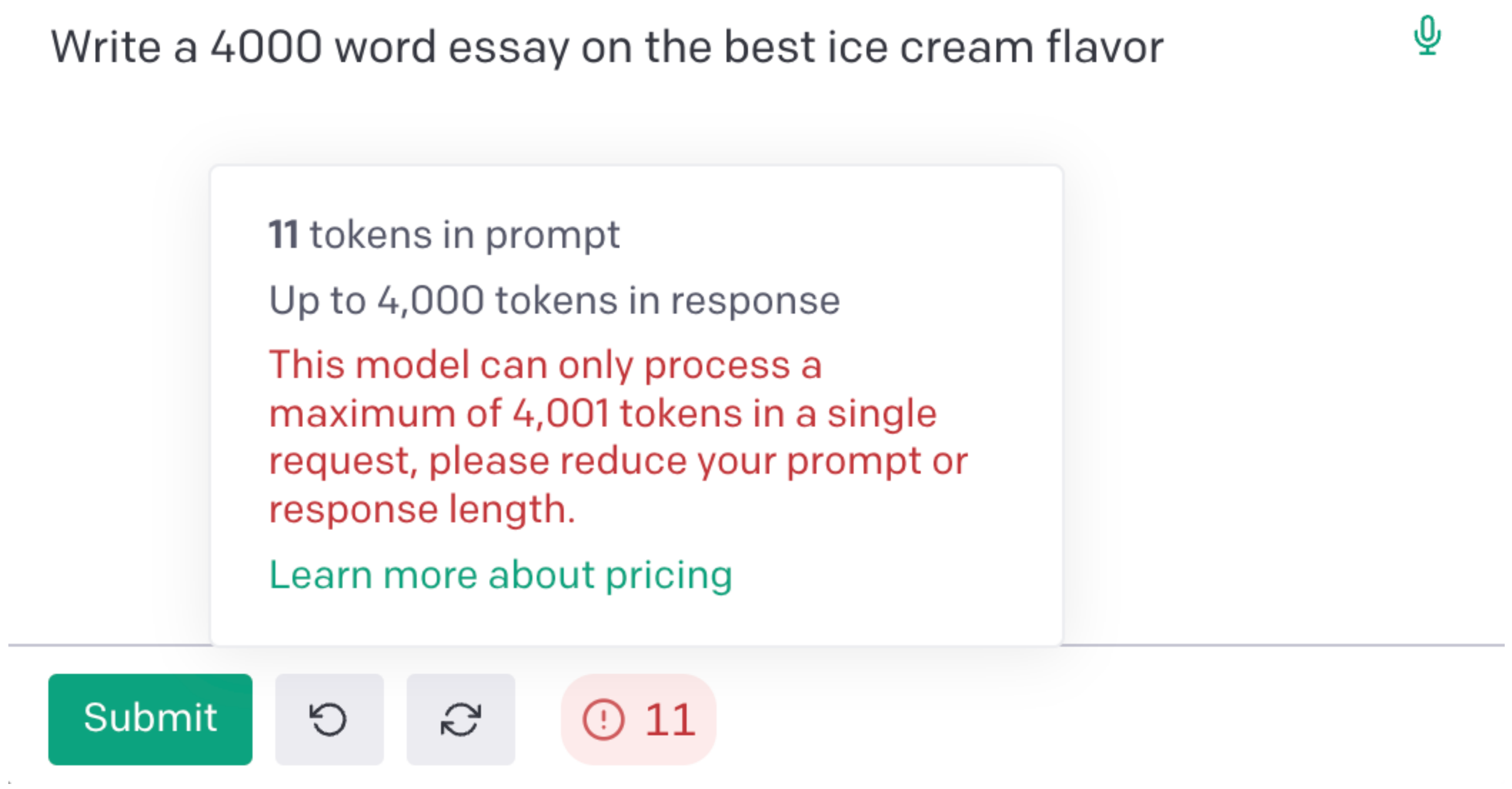
1.3B model: The joke is that if you try 10000 different seed choices, you'll eventually find one that works, but you'll be accused of overfitting.

175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm.
The punchline, "I guess **no good seed goes unpunished**," is a play on the phrase "**no good deed goes unpunished**." In this case, "good seed" refers to a starting point for the random restarts, and the joke implies that even when trying to improve the neural network's performance, the person is still accused of overfitting.

Scale is more closely tied to advances in ML than ever before

Core Challenge with Scale: Efficiency



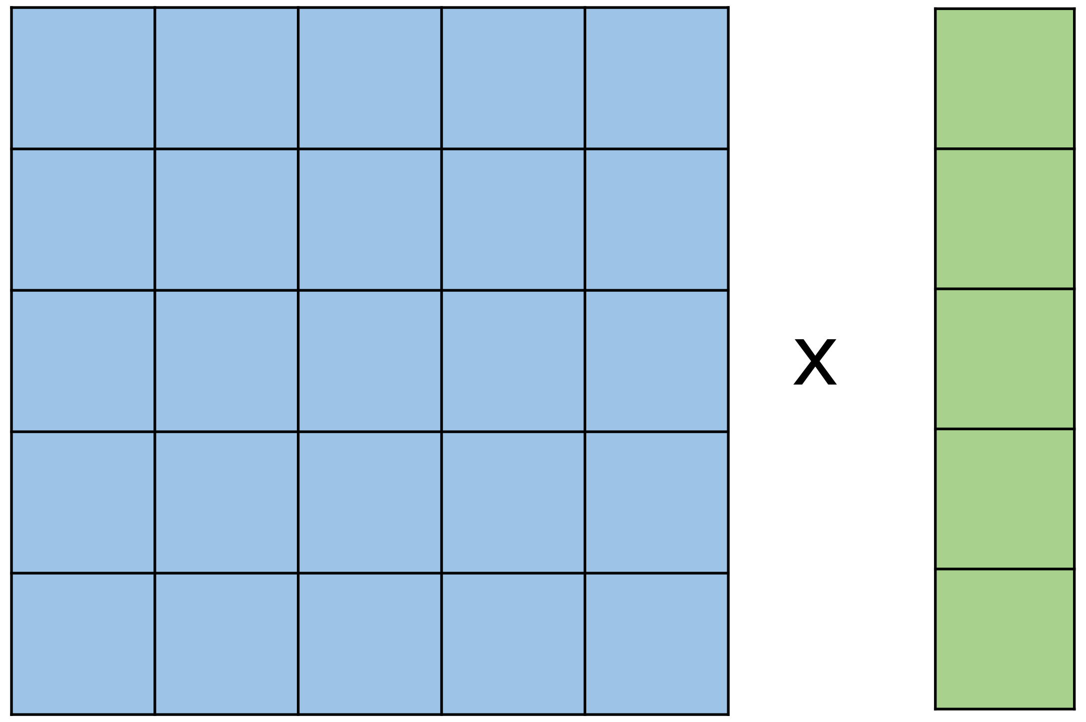


Efficiency eases training, deployment, and facilitates research

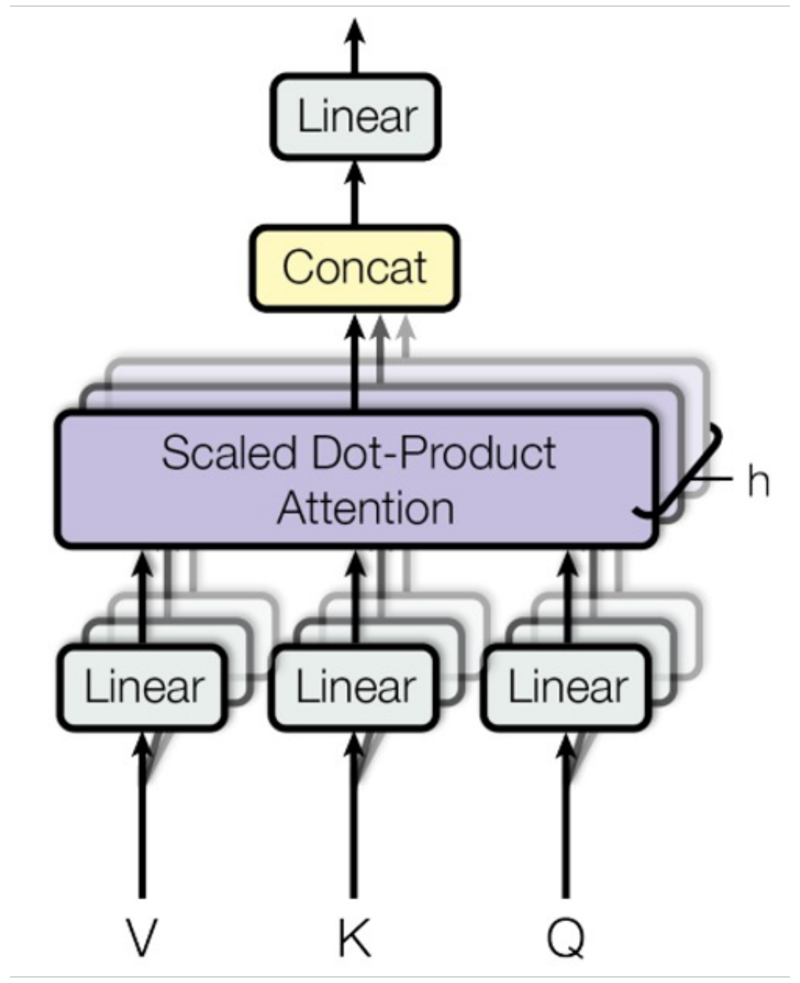
Efficiency unlocks new capabilities (e.g., long context)

My Approach to Efficiency: Understanding Algorithms & Systems

Fundamental algorithms



Fast matrix-vector multiply



Attention mechanism

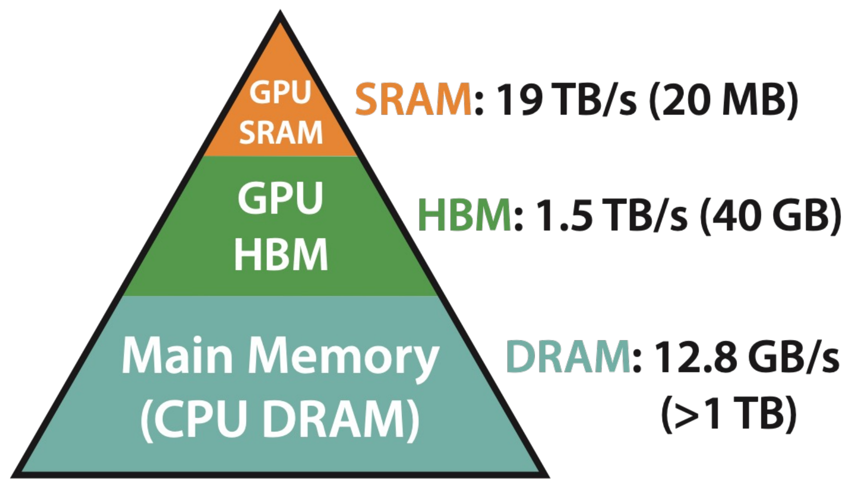
Hardware accelerators & distributed systems

Block-oriented device

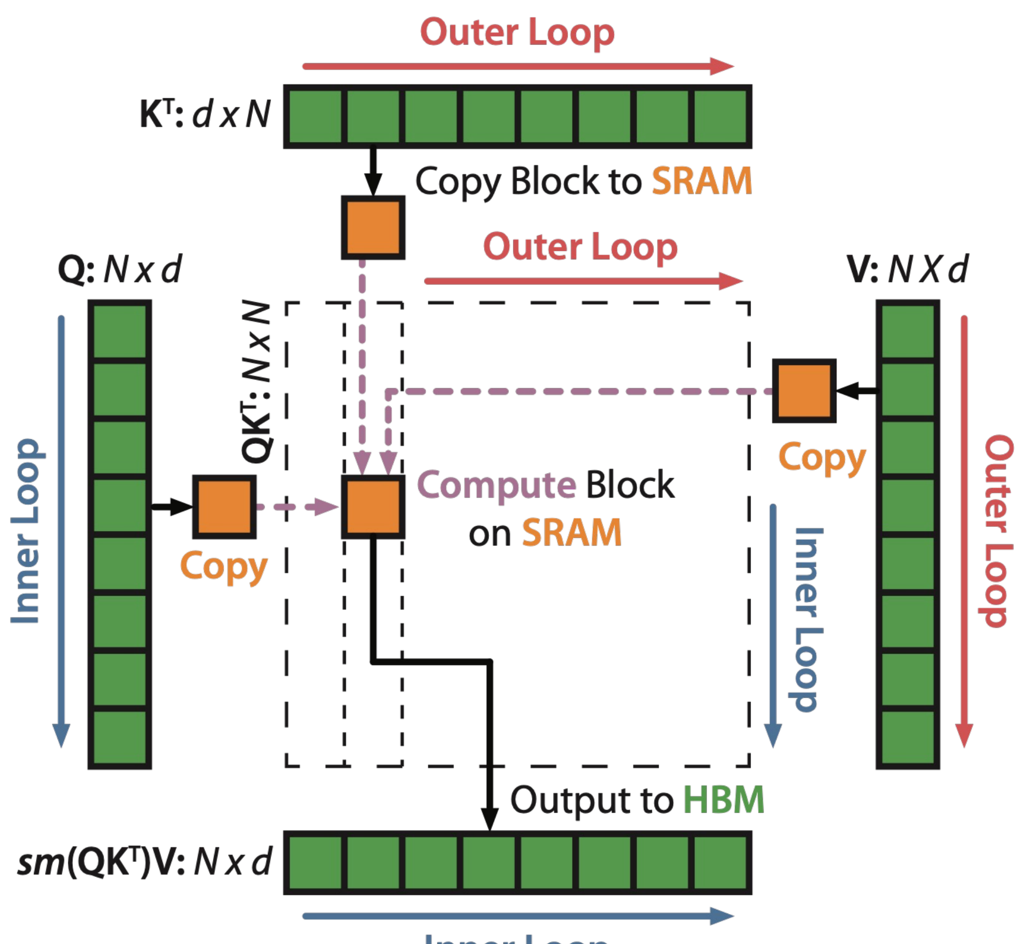
Asymmetric memory hierarchy

Main Idea: Hardware-aware Algorithms

IO-awareness:
reducing reads/writes to GPU memory yields significant speedup



Memory Hierarchy with Bandwidth & Memory Size

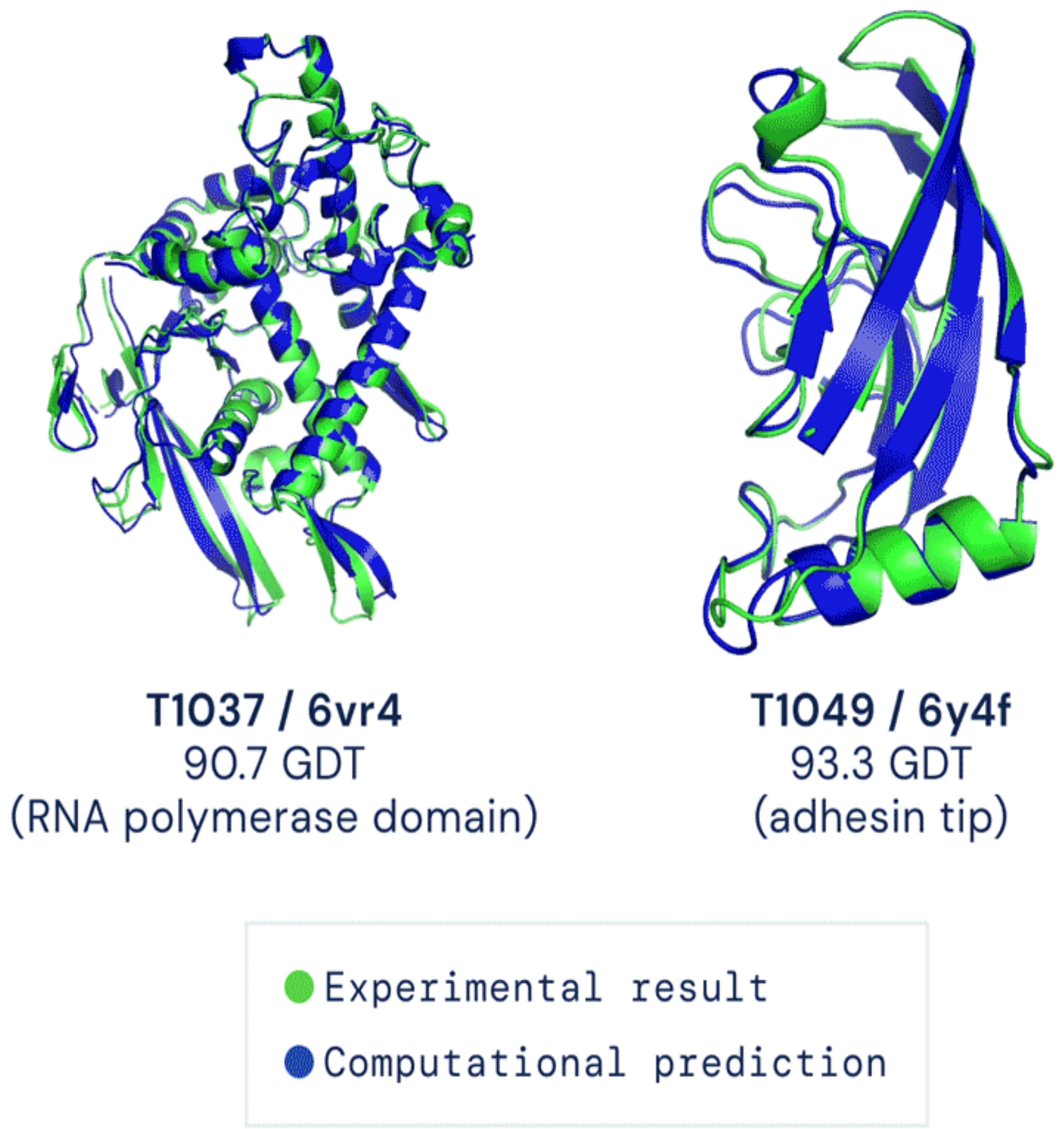


FlashAttention

FlashAttention: fast and memory-efficient attention algorithm, with no approximation

PyTorch OpenAI Meta NVIDIA Microsoft ANTHROP\C stability.ai mosaicML SambaNova SYSTEMS HUGGING FACE

FlashAttention Adoption Areas



Text Generation

(Llama - Meta, Falcon - TII UAE, MPT - Mosaic)

Image Generation

(Stable Diffusion - Stability.AI)

Drug Discovery

(OpenFold, UniFold)

Outlines

FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Future Directions

Software-hardware co-design, Long context for new workflow

Outlines

FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Future Directions

Software-hardware co-design, Long context for new workflow

Motivation: Modeling Long Sequences

Enable New Capabilities

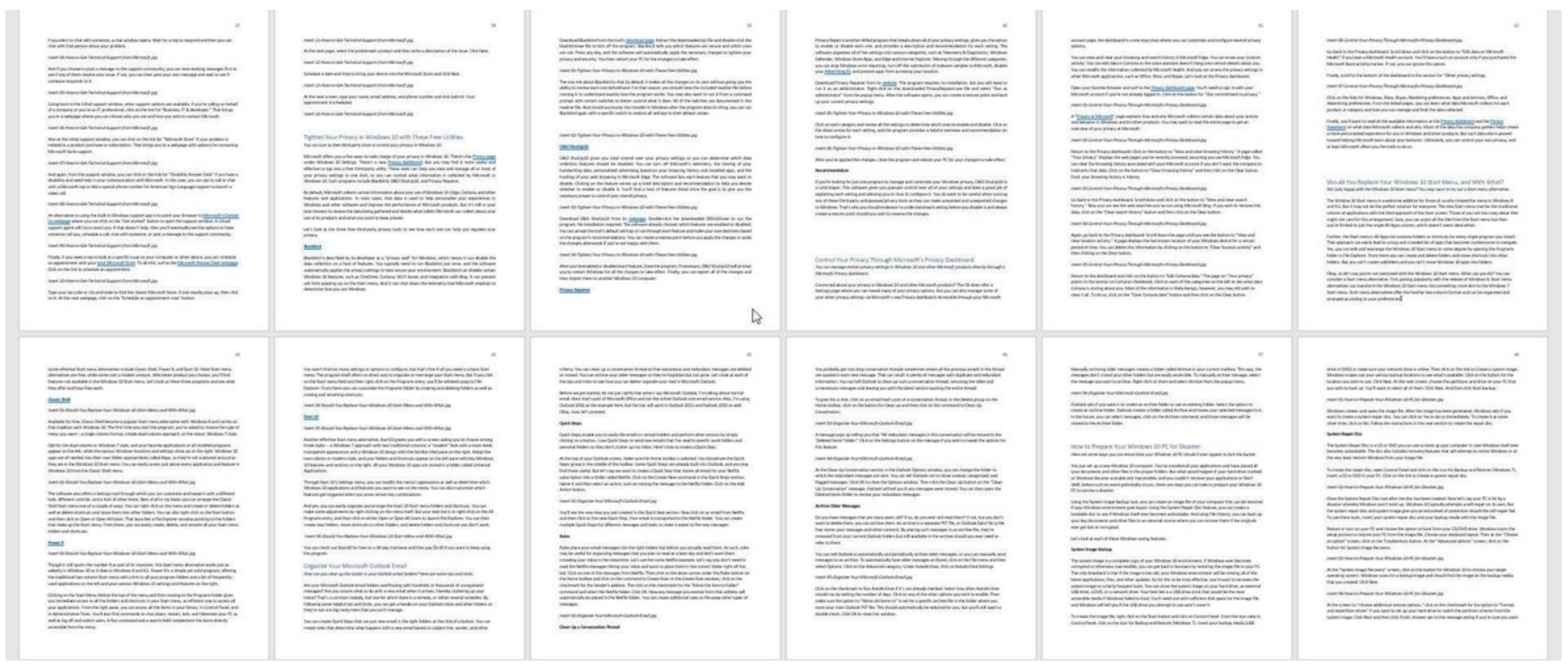
NLP: Large context required to understand books, plays, codebases.

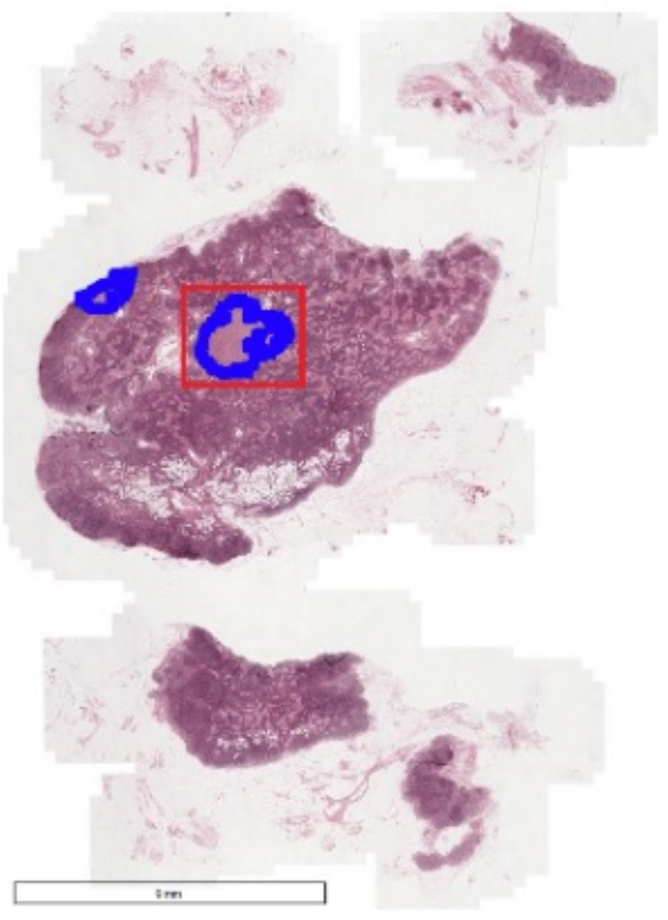
Close Reality Gap

Computer vision: higher resolution can lead to better, more robust insight.

Open New Areas

Time series, audio, video, medical imaging data naturally modeled as sequences of millions of steps.

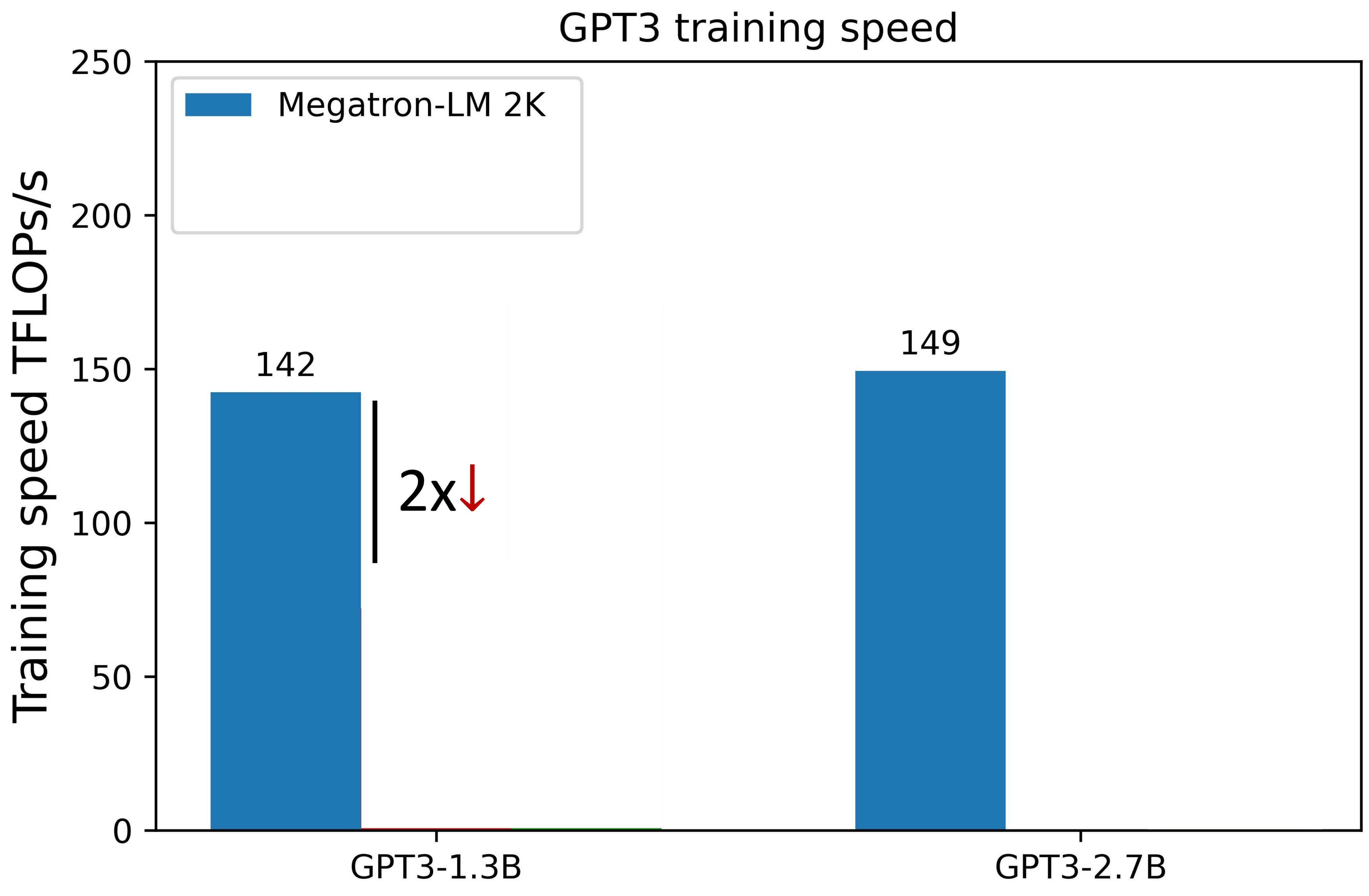




Efficiency is the Bottleneck for Modeling Long Sequences with Attention

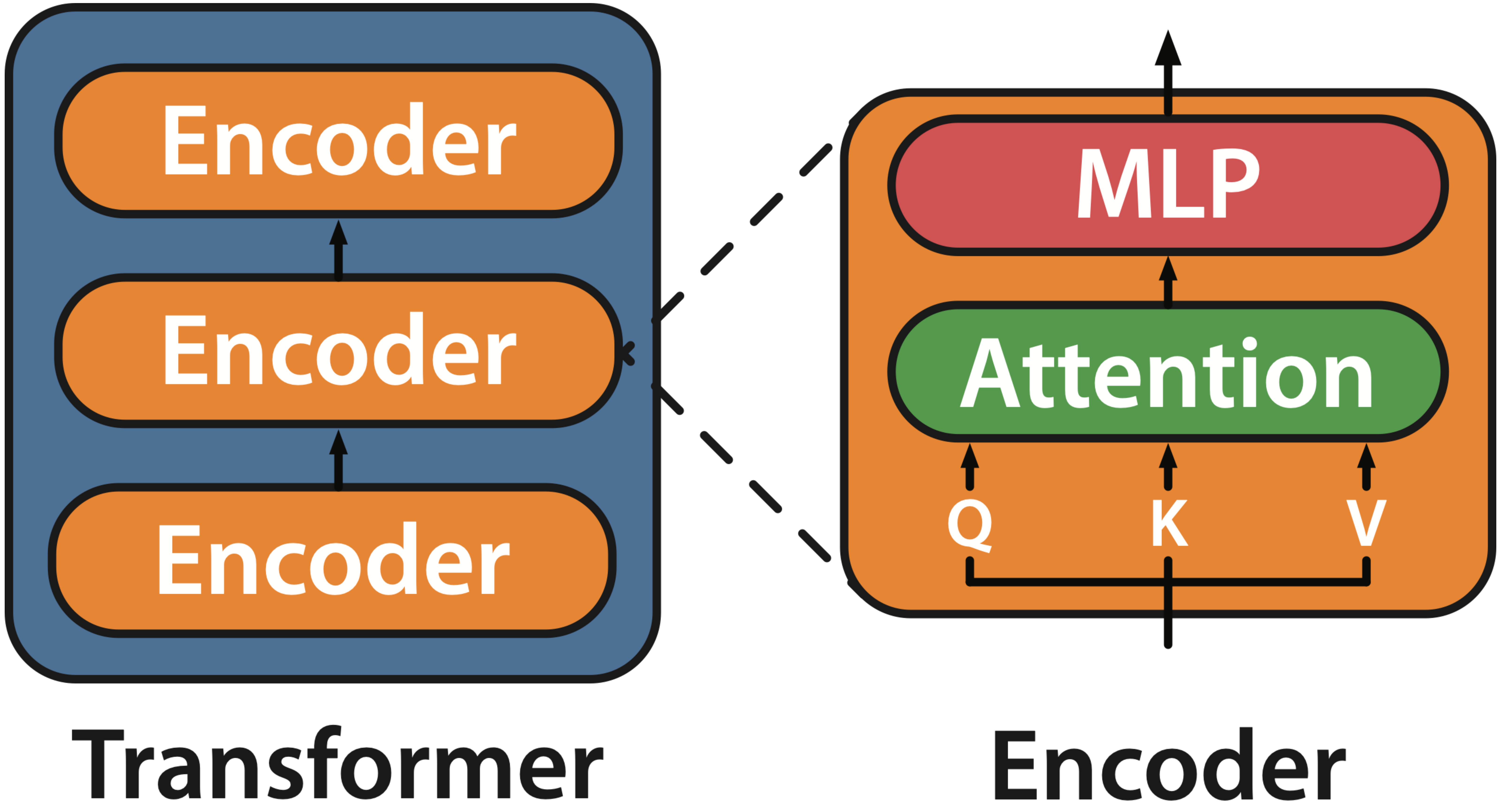
Context length: how many other elements in the sequence does the current element interact with.

Increasing context length slows down (or stops) training

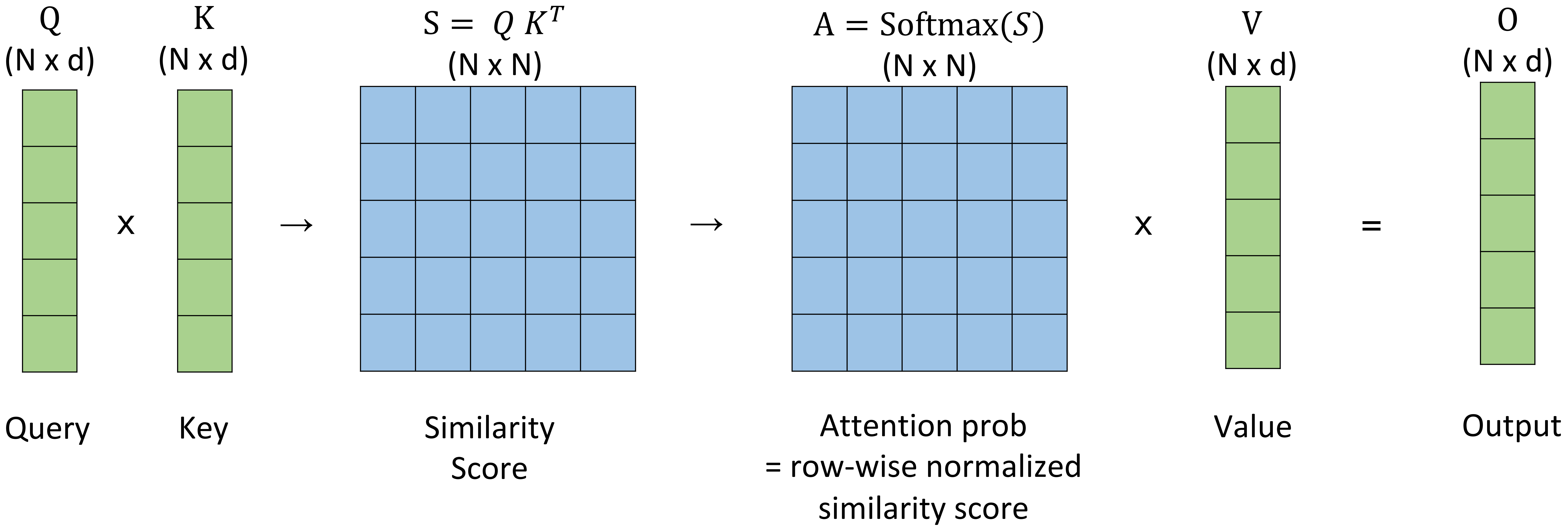


How to efficiently scale models to longer sequences?

Background: Attention is the Heart of Transformers



Background: Attention Mechanism



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

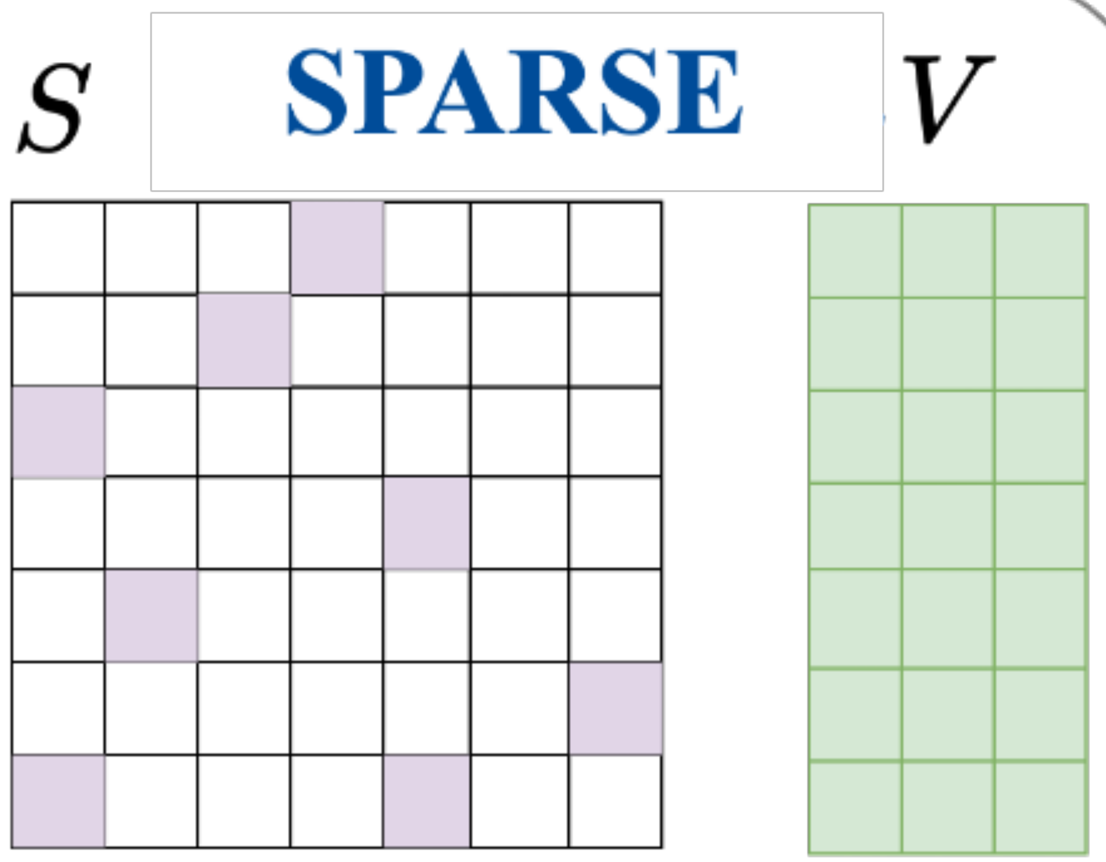
$$\mathrm{Softmax}([s_1, \cdots, s_N]) = \left[\frac{e^{s_1}}{\sum_i e^{s_i}}, \cdots, \frac{e^{s_N}}{\sum_i e^{s_i}}\right]$$

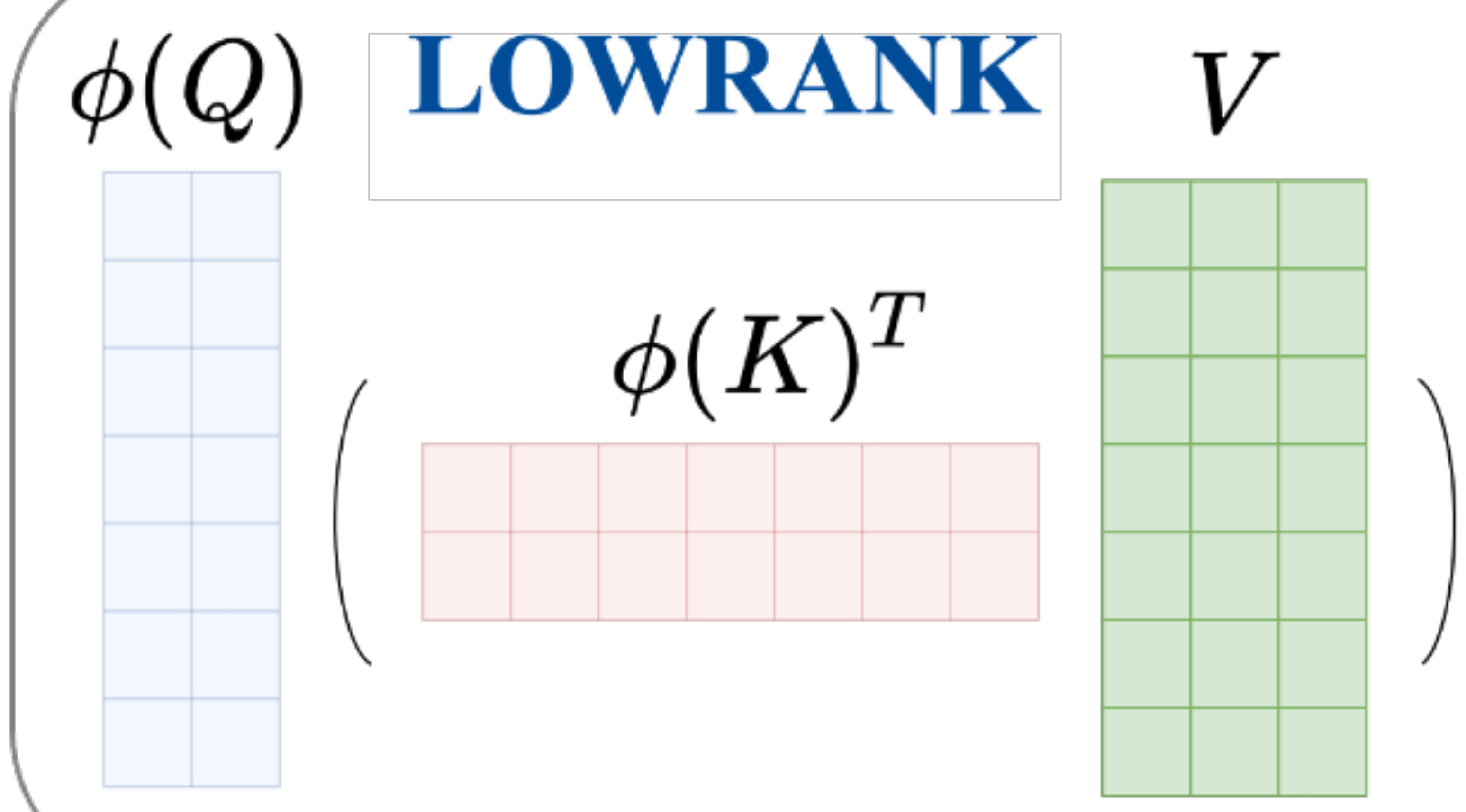
$$O = \mathrm{Softmax}(QK^T)V$$

Attention scales quadratically in sequence length N

Background: Approximate Attention

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





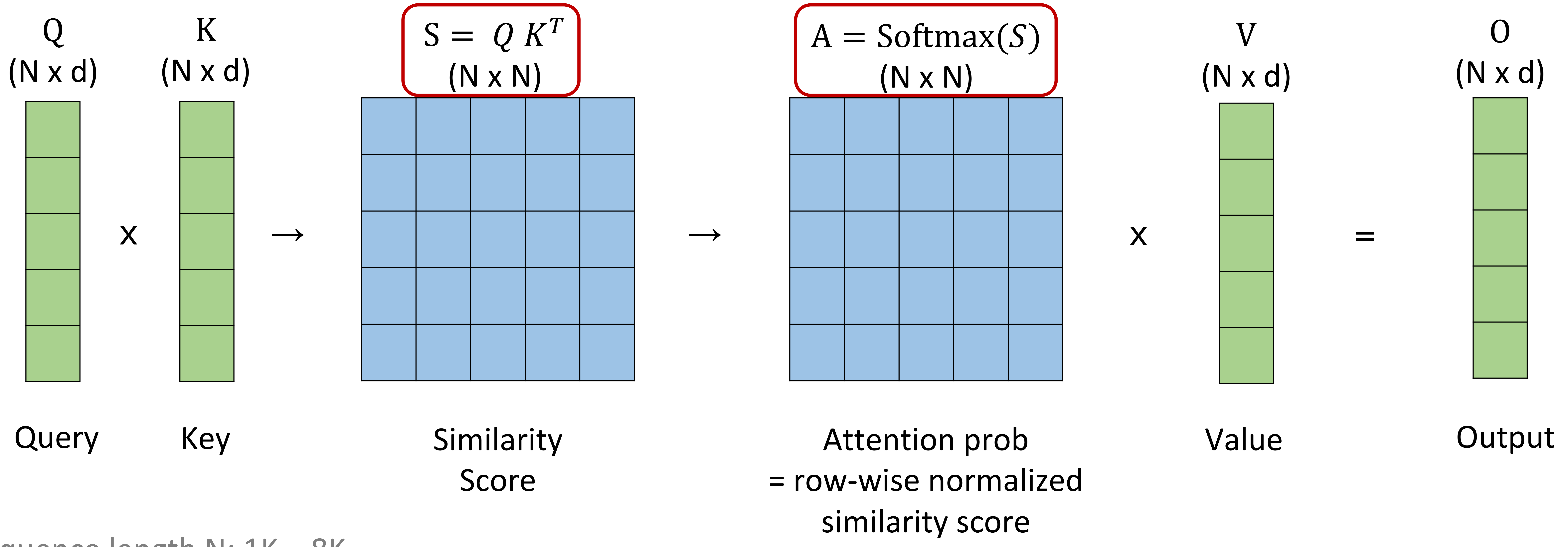
Linformer
(Wang et al. 20)
Linear Transformer
(Katharopoulos et al. 20)
Performer
(Choromanski et al. 20)

Approximate attention: tradeoff quality for ~~speed~~ fewer FLOPs

Survey: Tay et al. Long Range Arena : A Benchmark for Efficient Transformers. ***ICLR 2020.***

Is there a fast, memory-efficient, and exact attention algorithm?

Our Observation: Attention is Bottlenecked by Memory Reads/Writes



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

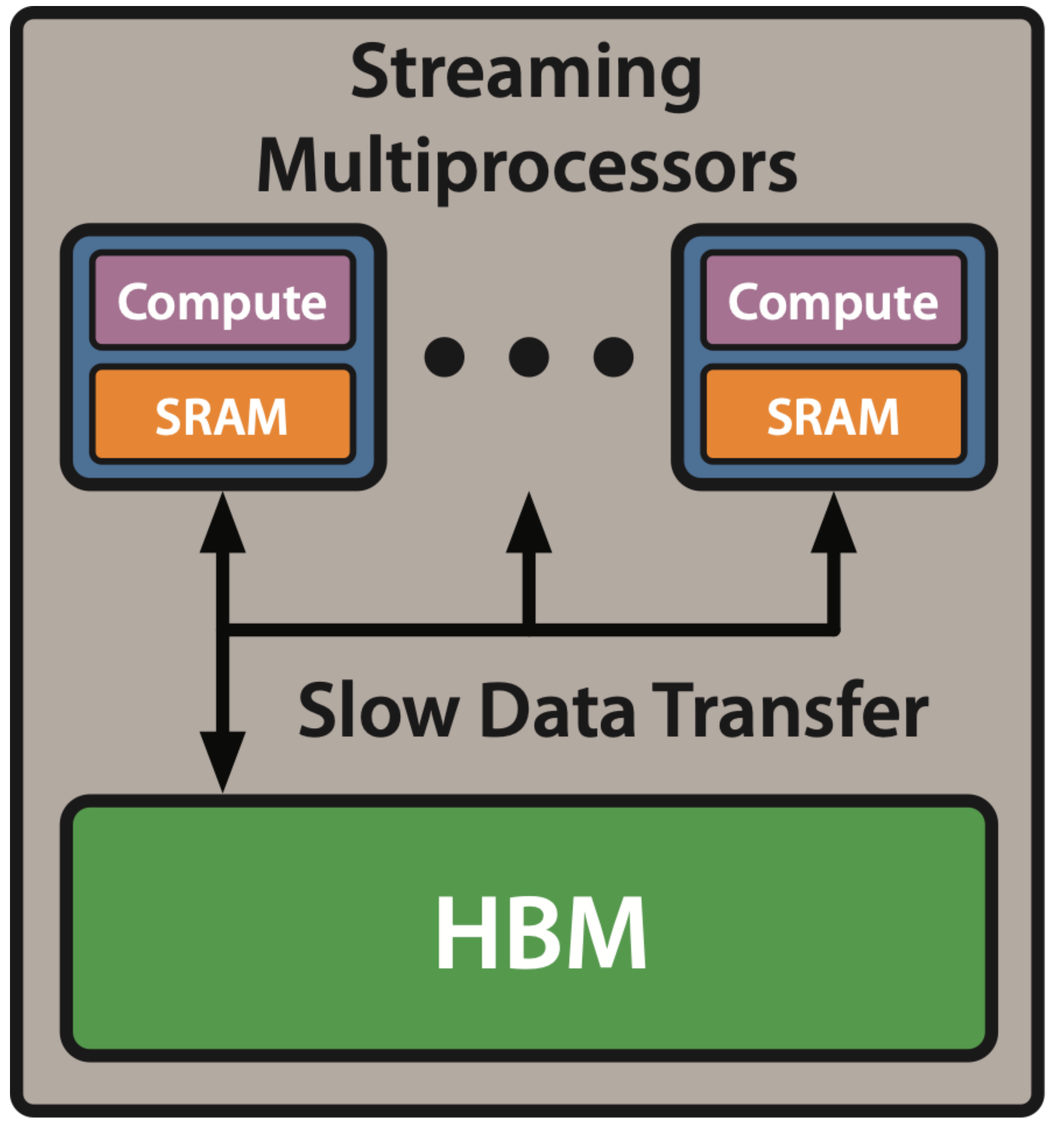
The biggest cost is in moving the bits!
Standard implementation requires repeated R/W from slow GPU memory

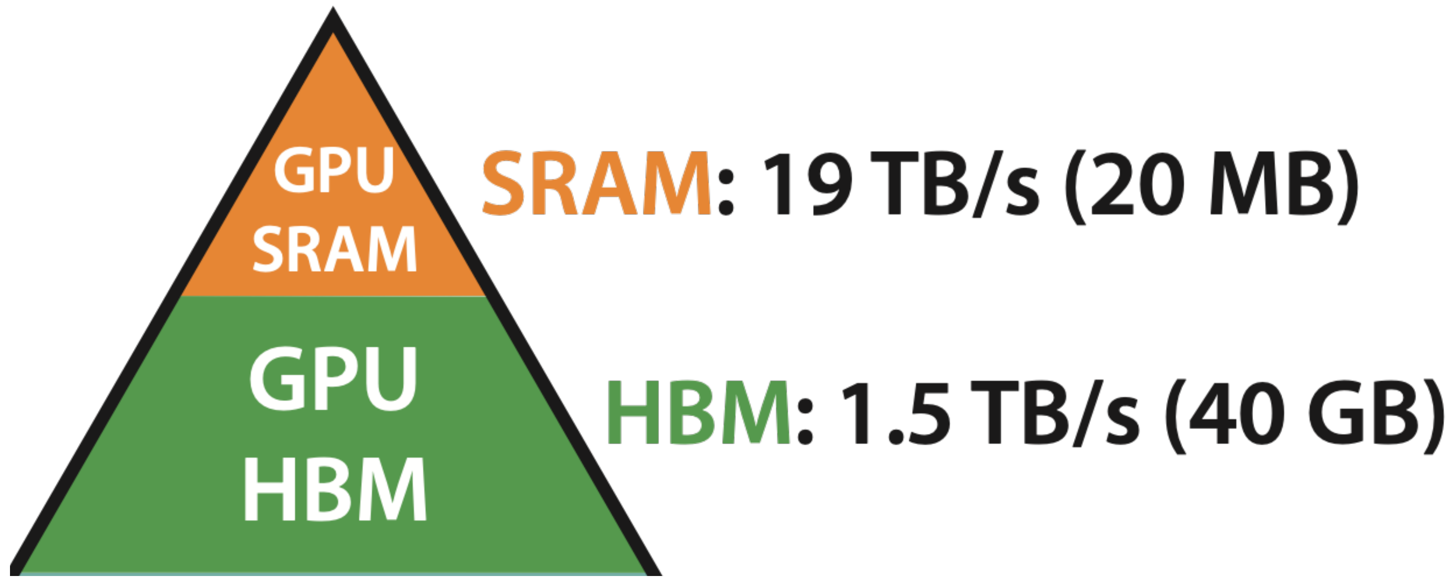
Background: GPU Compute Model & Memory Hierarchy

2. Data moved to compute units & SRAM for computation

1. Inputs start out in HBM (GPU memory)

3. Output written back to HBM





Blogpost: Horace He, Making Deep Learning Go Brrrr From First Principles.

Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges:

(1) Compute softmax normalization without access to full input.

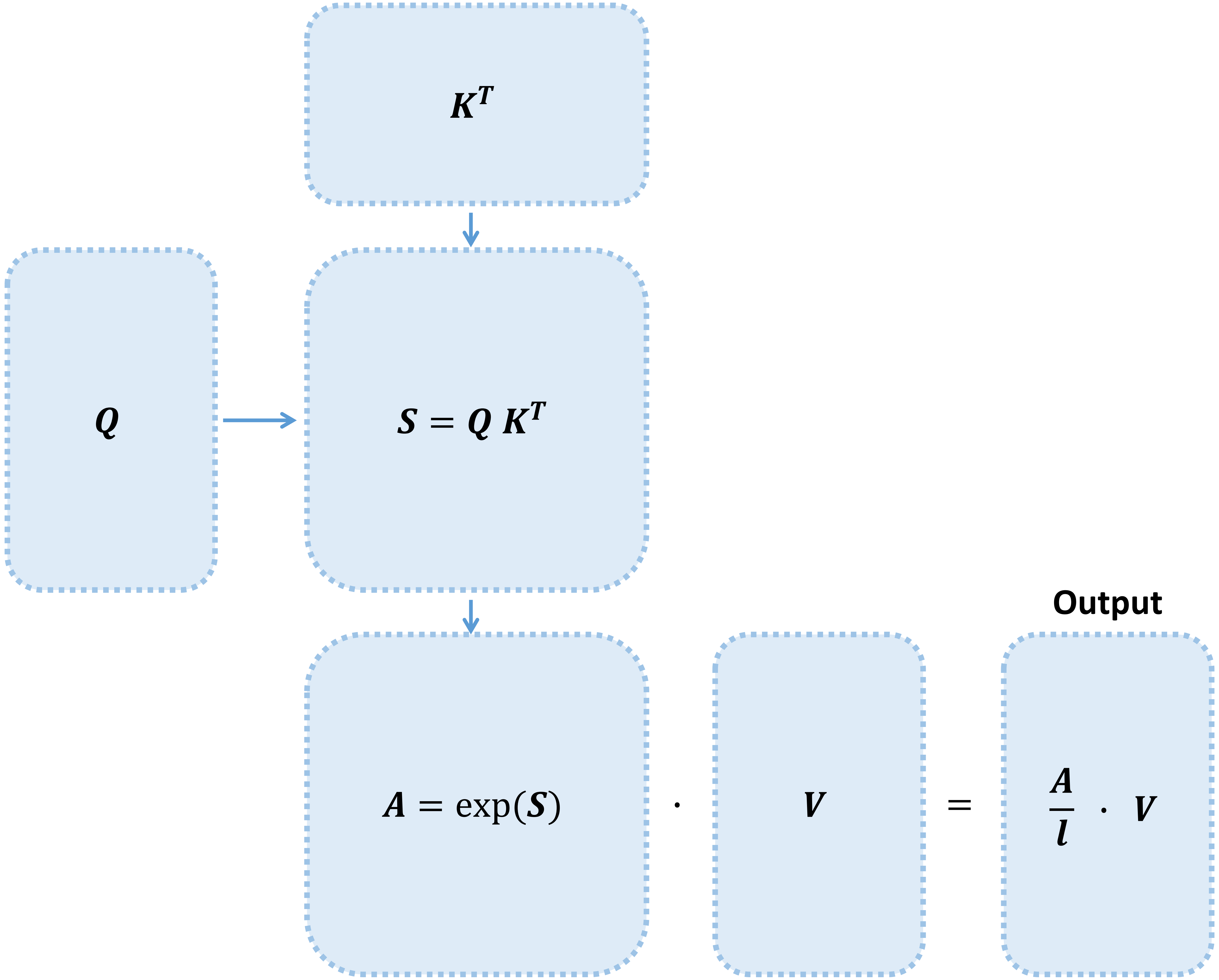
(2) Backward without the large attention matrix from forward.

Approaches:

(1) Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

(2) Recomputation: Don’t store attn. matrix from forward, recompute it in the backward.

Attention Computation Overview



Softmax row-wise
normalization constant

$$l = \sum_i \exp(S)_i$$

Compute by blocks: easy to split Q, but how do we split K & V?

Tiling – 1st Attempt: Computing Attention by Blocks

Goal:
Load each block from HBM to SRAM & do local computation

$(\boldsymbol{K}^{(1)})^T$ $(\boldsymbol{K}^{(2)})^T$

Example: Split K into 2 blocks

$\boldsymbol{Q}$

$\boldsymbol{S}^{(1)} = \boldsymbol{Q}\left(\boldsymbol{K}^{(1)}\right)^T$ $\boldsymbol{S}^{(2)} = \boldsymbol{Q}\left(\boldsymbol{K}^{(2)}\right)^T$

$\boldsymbol{A}^{(1)} = \exp(\boldsymbol{S}^{(1)})$ $\boldsymbol{A}^{(2)} = \exp(\boldsymbol{S}^{(2)})$ $\cdot$ $\boldsymbol{V}^{(1)}$ $\boldsymbol{V}^{(2)}$ $=$

Output

$$\boxed{\frac{\boldsymbol{A}^{(1)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(1)} + \frac{\boldsymbol{A}^{(2)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(2)}}$$

Softmax row-wise normalization constant

$$\boxed{\boldsymbol{l} = \sum_i \exp\left(\boldsymbol{S}^{(1)}\right)_i + \sum_i \exp\left(\boldsymbol{S}^{2}\right)_i}$$

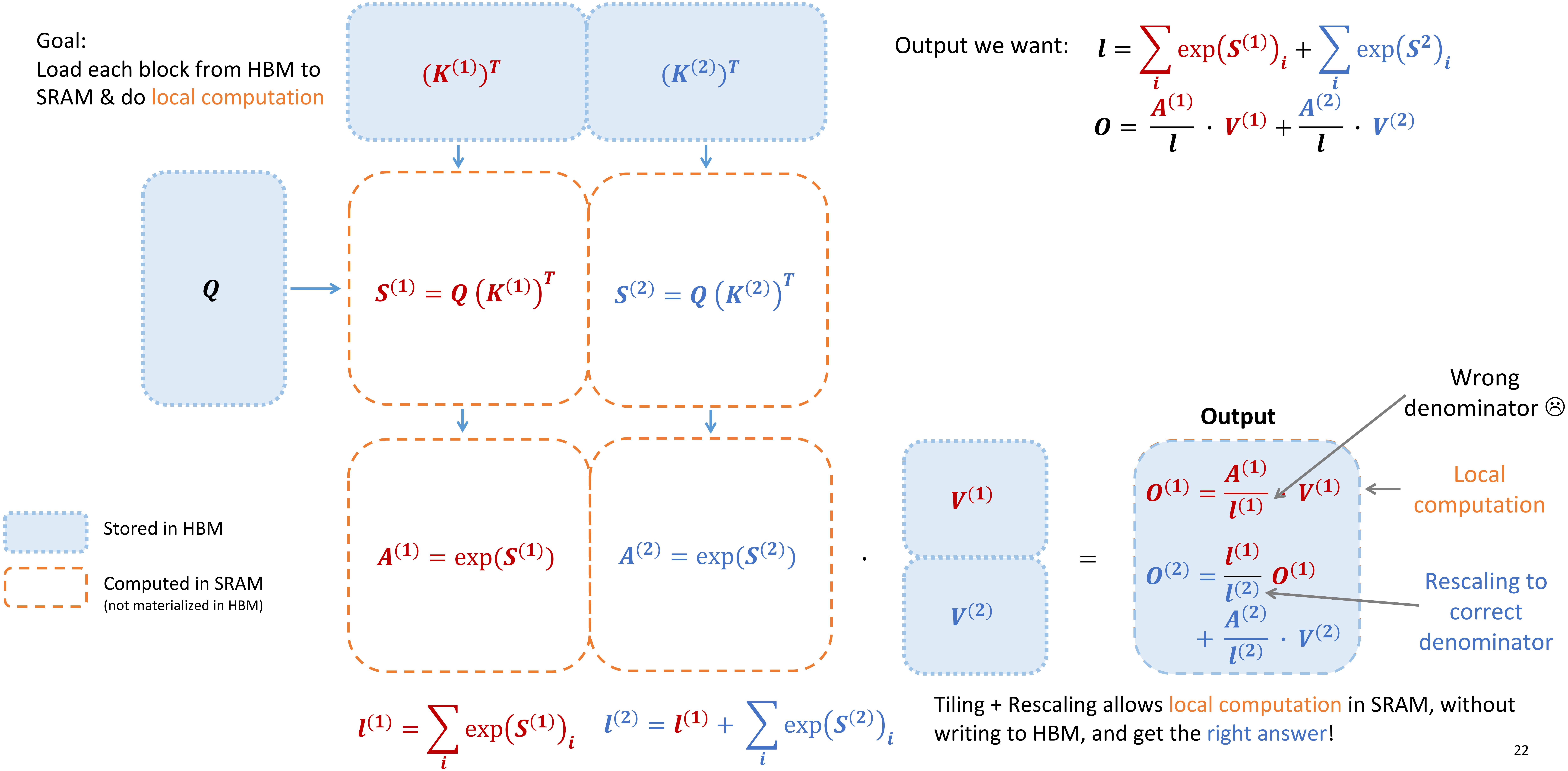
Challenge: How to compute softmax normalization with just local results?

Tiling – 2nd Attempt: Computing Attention by Blocks, with Softmax Rescaling

Goal:
Load each block from HBM to SRAM & do local computation

Output we want: $l = \sum_i \exp(S^{(1)})_i + \sum_i \exp(S^2)_i$

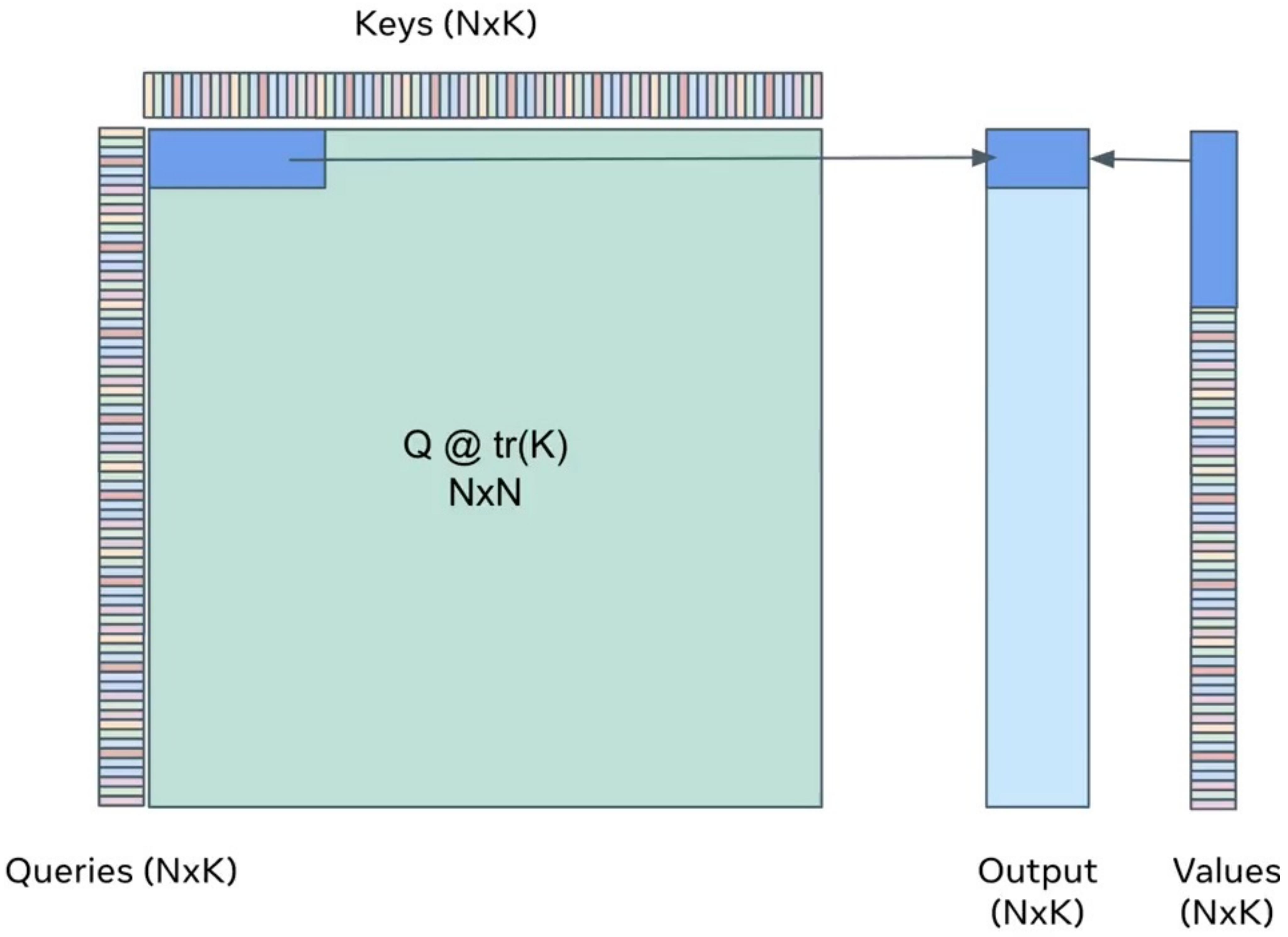
$$O = \frac{A^{(1)}}{l} \cdot V^{(1)} + \frac{A^{(2)}}{l} \cdot V^{(2)}$$



Tiling + Rescaling allows local computation in SRAM, without writing to HBM, and get the right answer!

Tiling

Decomposing large softmax into smaller ones by scaling.



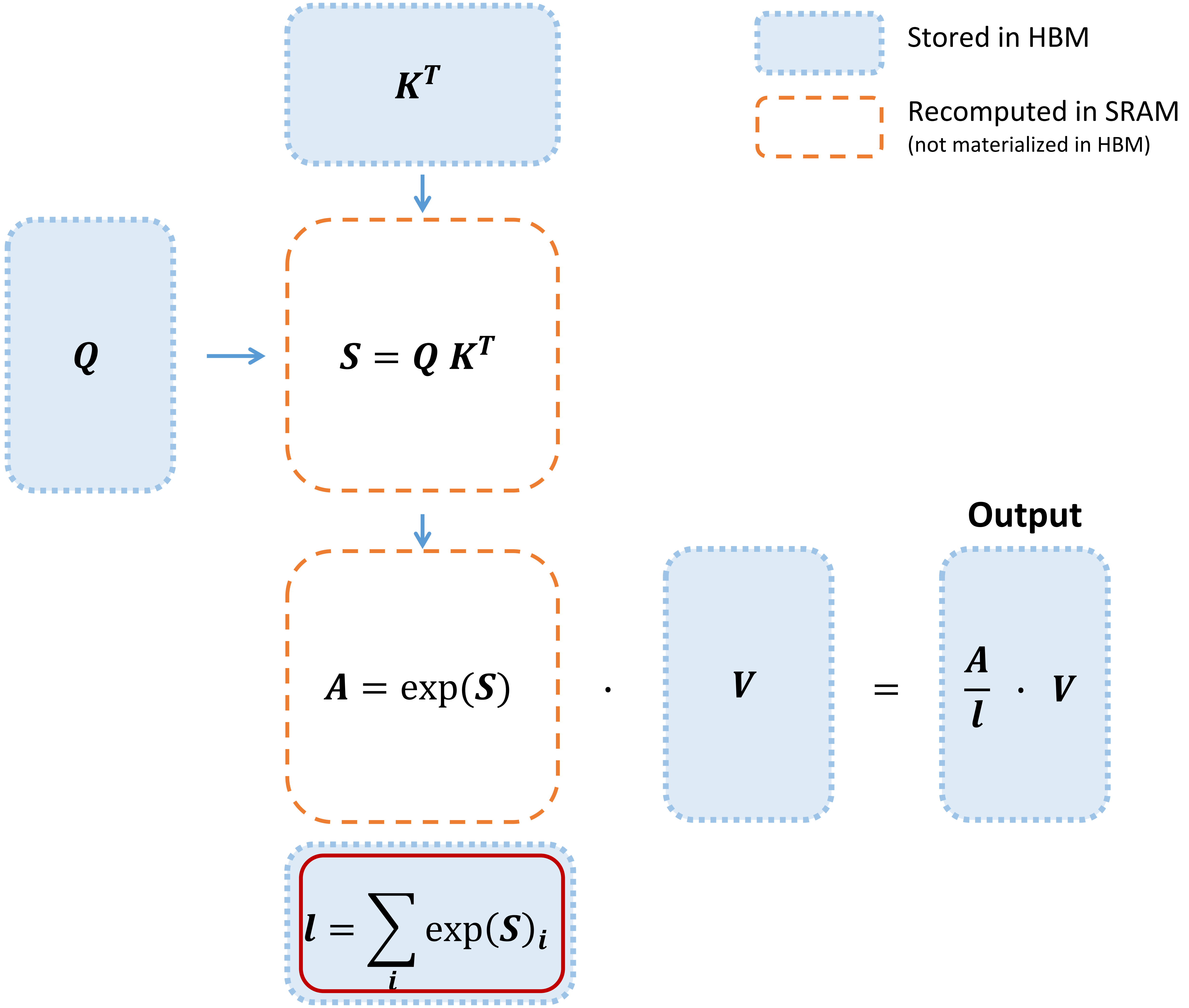
1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output with respect to that block.
3. Update output in HBM by scaling.

Animation credit: Francisco Massa

Recomputation (Backward Pass)

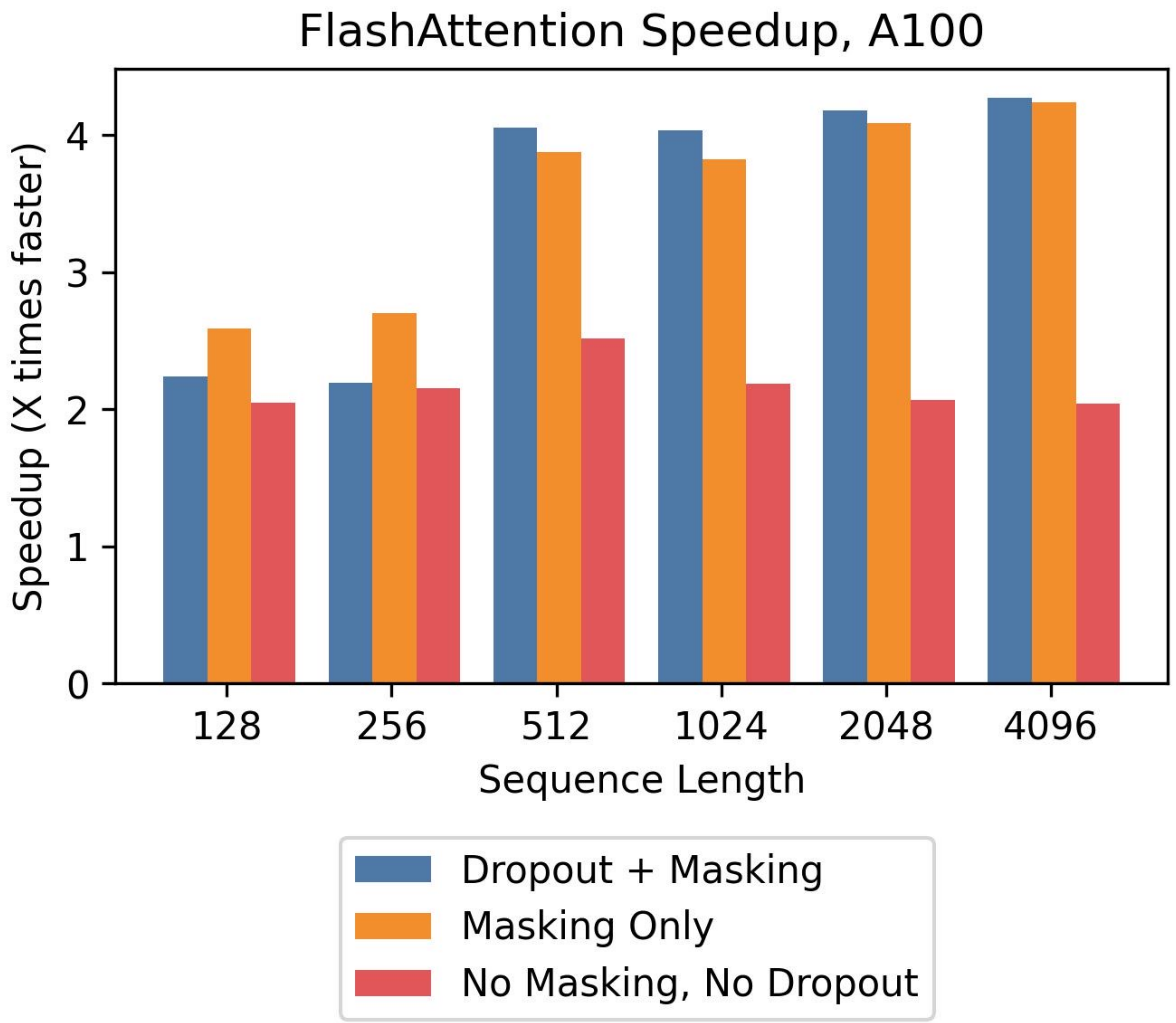
By storing softmax normalization from forward (size N), quickly recompute attention in the backward from inputs in SRAM.

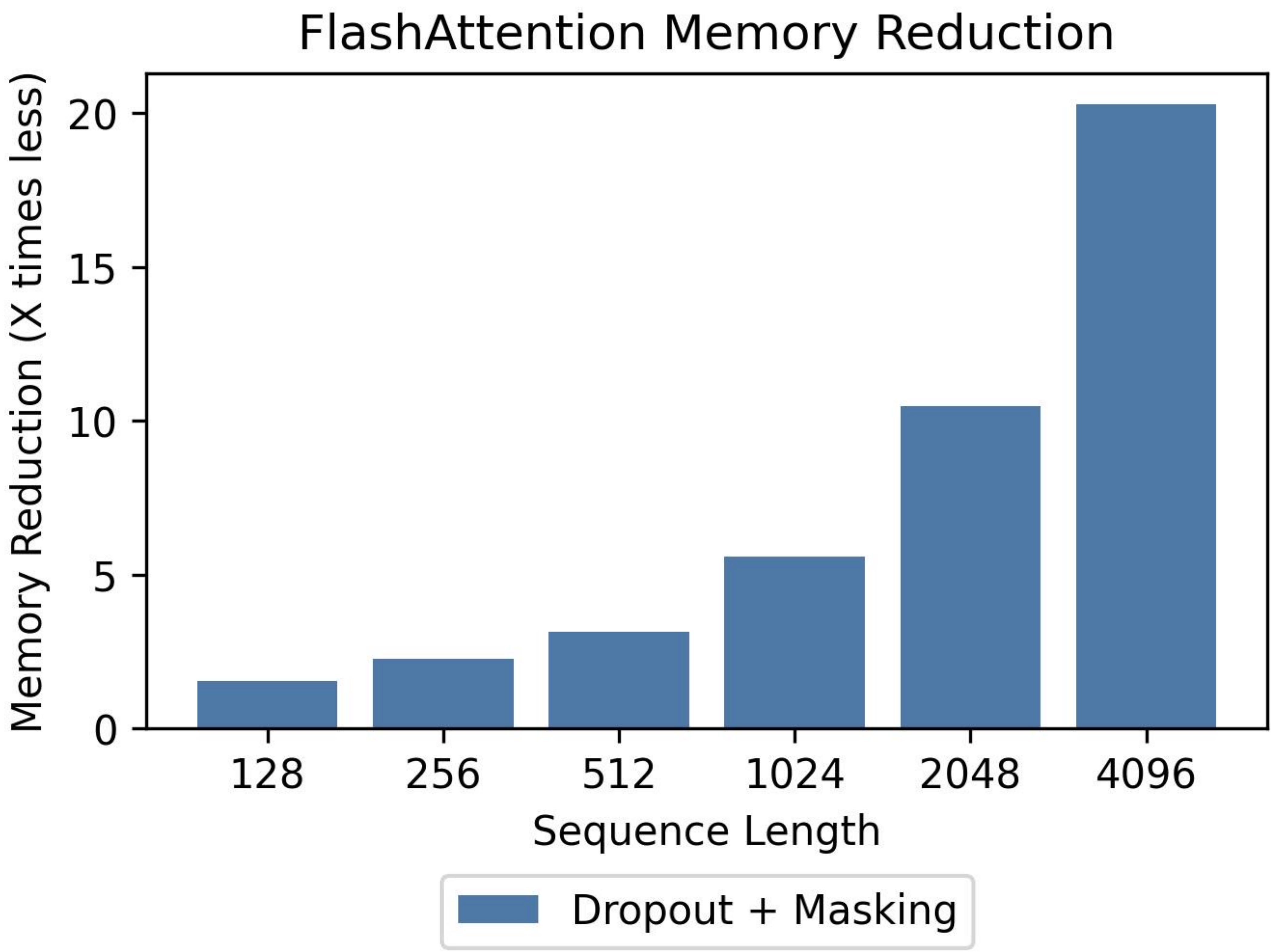
Attention	Standard	FlashAttention
GFLOPs	66.6	75.2 (↑13%)
HBM reads/writes (GB)	40.3	4.4 (↓9x)
Runtime (ms)	41.7	7.3 (↓6x)



FlashAttention speeds up backward pass even with increased FLOPs.

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





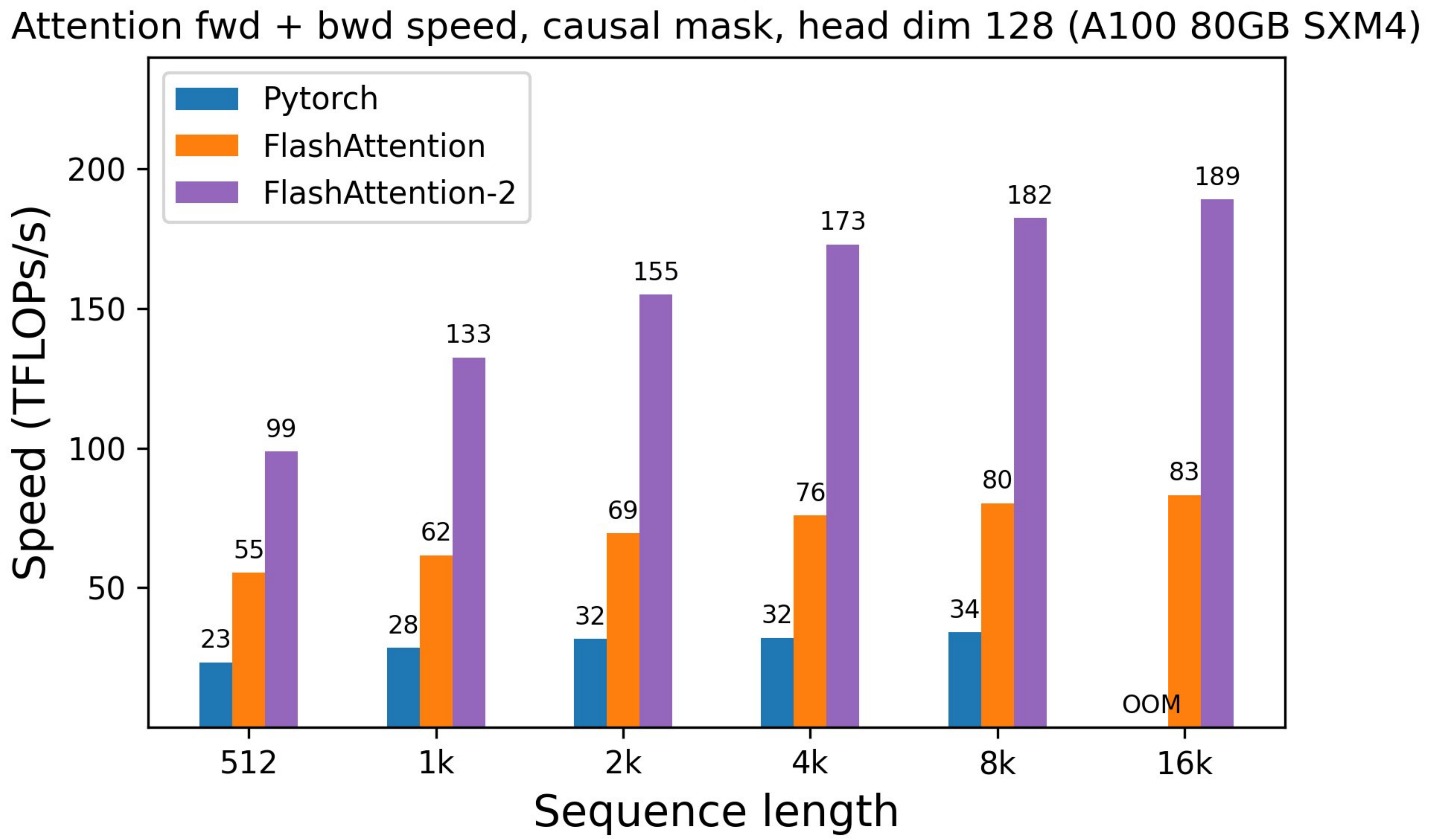
2-4x speedup — with no approximation!

10-20x memory reduction — memory linear in sequence length

FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning

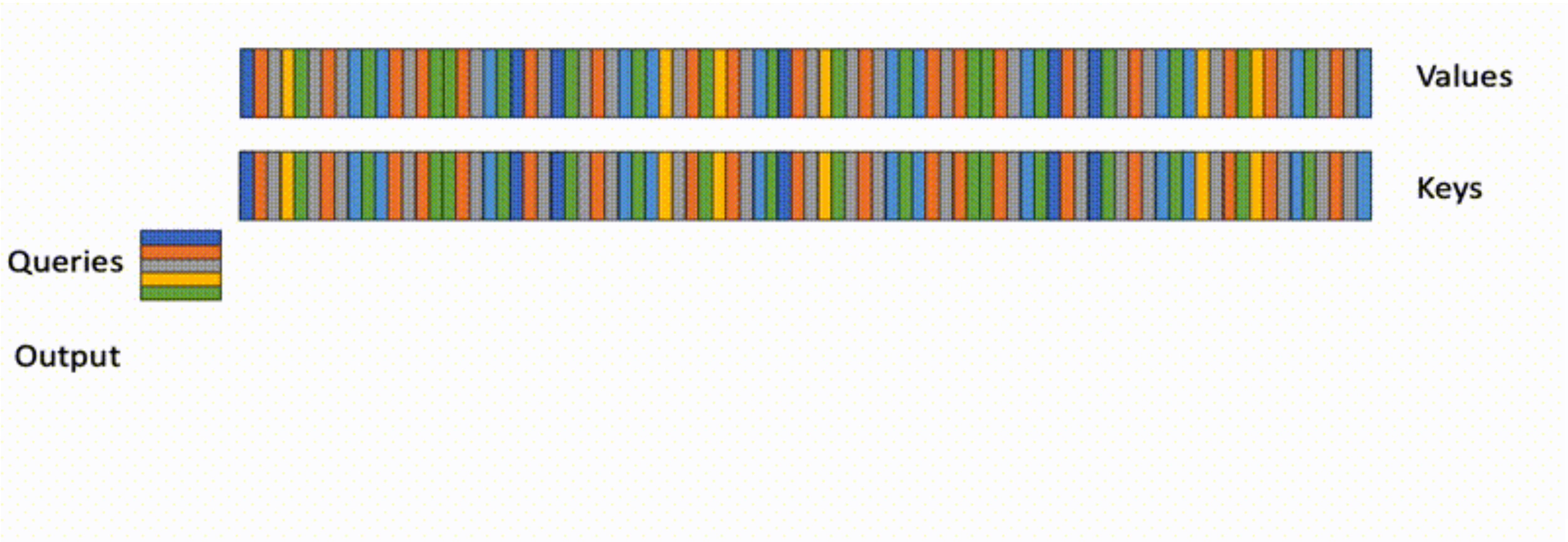
Key ideas:
- Reduce non-matmul FLOPs
- Parallelize over seqlen dimension to improve occupancy
- Better work partitioning between warps to reduce communication

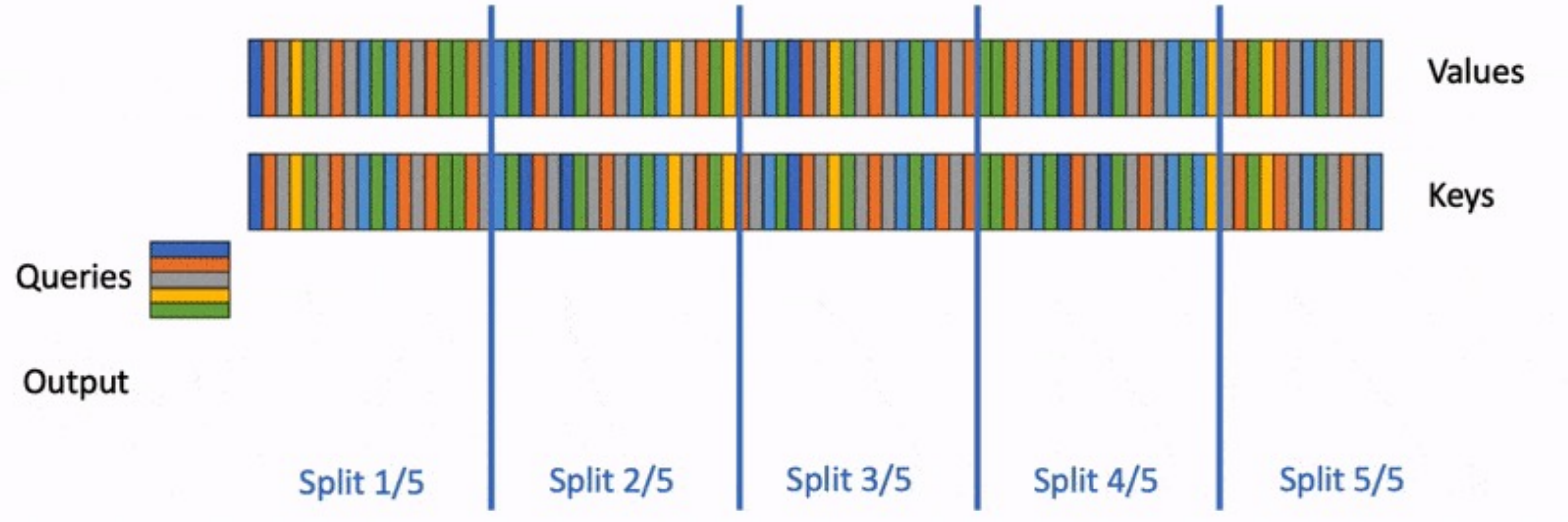
Upshot: **2x** faster wallclock, can train models with 2x context length for the same cost



Flash-Decoding: Faster Decoding for Long Context Inference

Tri Dao, Daniel Haziza, Francisco Massa, Grigory Sizov





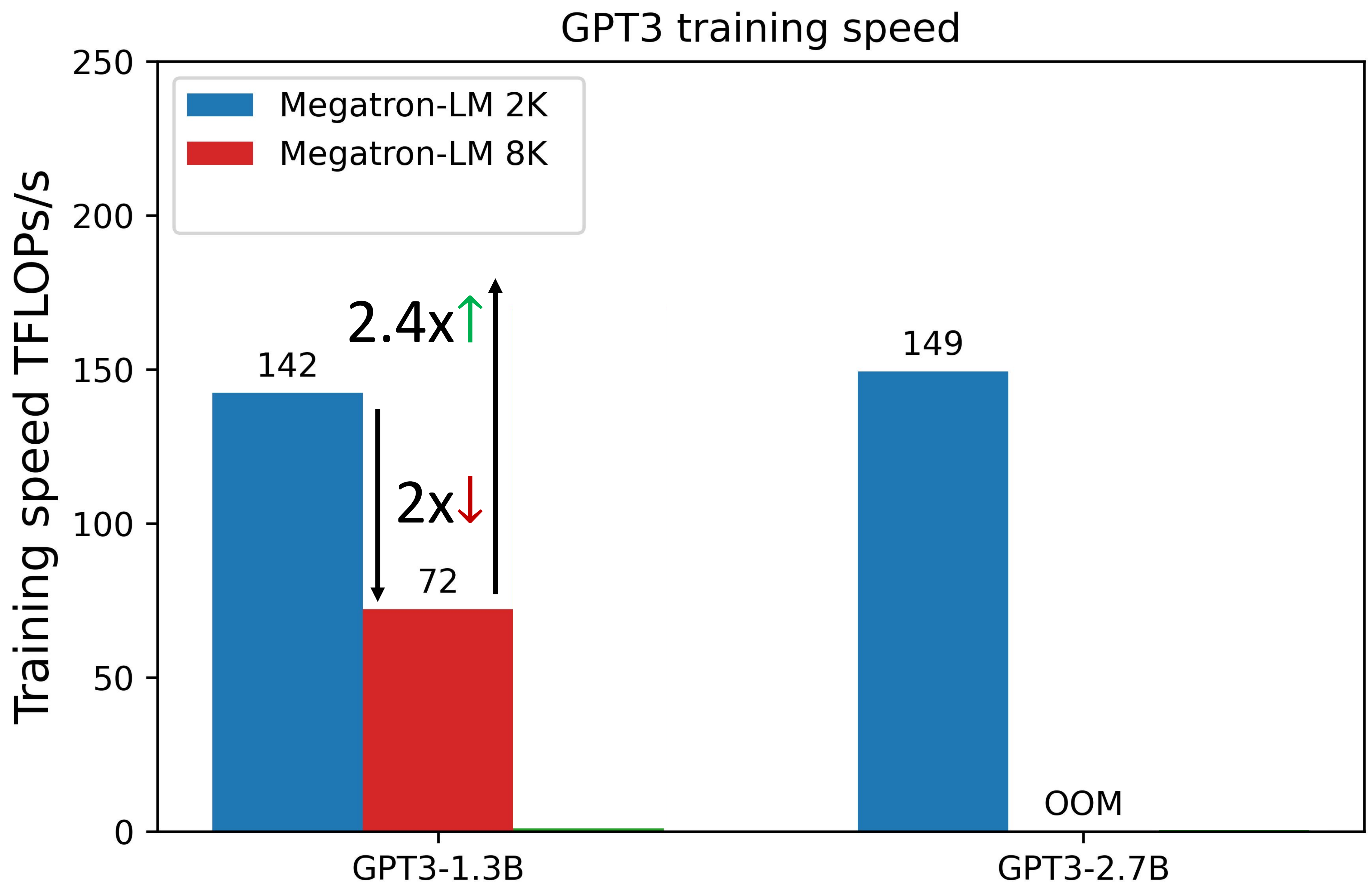
FlashAttention:
- Parallelizes across blocks of queries, batch size, and heads only
- Does not to occupy the entire GPU during decoding.

Flash-Decoding:
- Parallelize KV cache loading over seqlen dim
- Separate reduction step to combine results

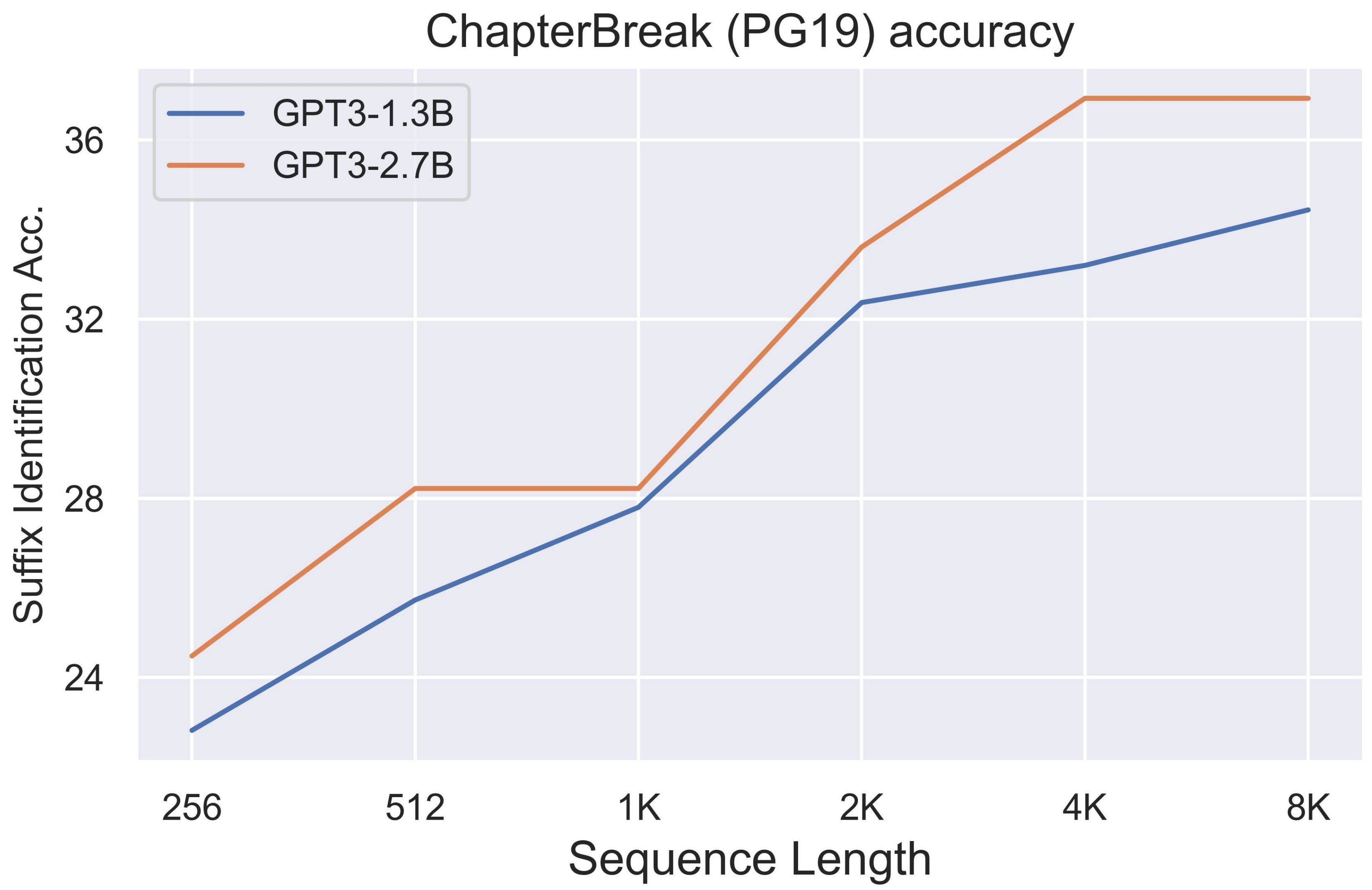
Upshot: **2-8x** faster end-to-end generation on CodeLlama 34B with context 32k-100k.

Animation credit: Daniel Haziza

GPT3: Faster Training, Longer Context, Better Model



Model	Val perplexity on the Pile (lower better)
GPT-1.3B, 2K context	5.45
GPT-1.3B, 8K context	**5.24**
GPT-2.7B, 2K context	5.02
GPT-2.7B, 8K context	**4.87**



FlashAttention speeds up GPT-3 training by 2x, increase context length by 4x, improving model quality

Shoeybi et al. arXiv:1909.08053 2019.

Summary – FlashAttention

FlashAttention: **fast** and **memory-efficient** algorithm for **exact** attention

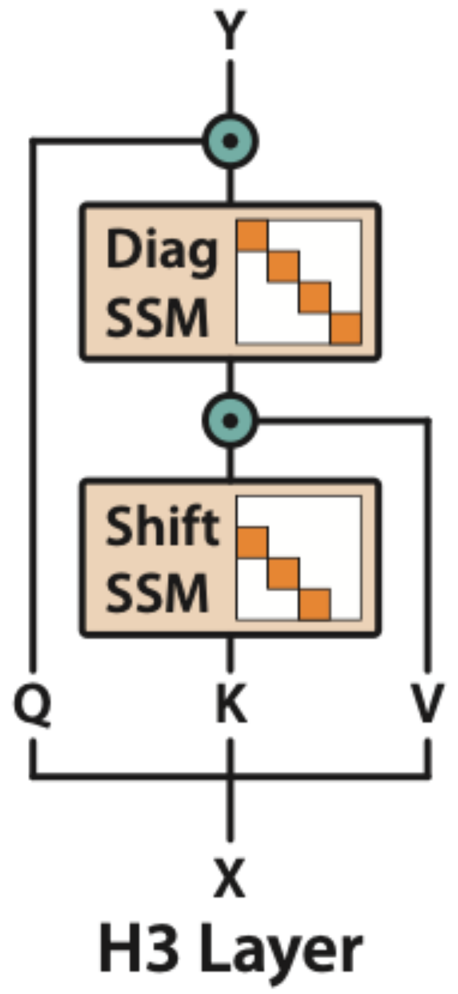
Key algorithmic ideas: **tiling**, **recomputation**

Upshot: **faster** training, **better** models with **longer** sequences

Code: https://github.com/Dao-AILab/flash-attention

How These Hardware-efficiency Ideas Generalize – Hungry Hungry Hippos: Language Modeling with State-space Models

Daniel Y. Fu*, Tri Dao*, Khaled K. Saab, Armin W. Thomas, Atri Rudra, Christopher Ré. Spotlight, ICLR 2023



Challenges:

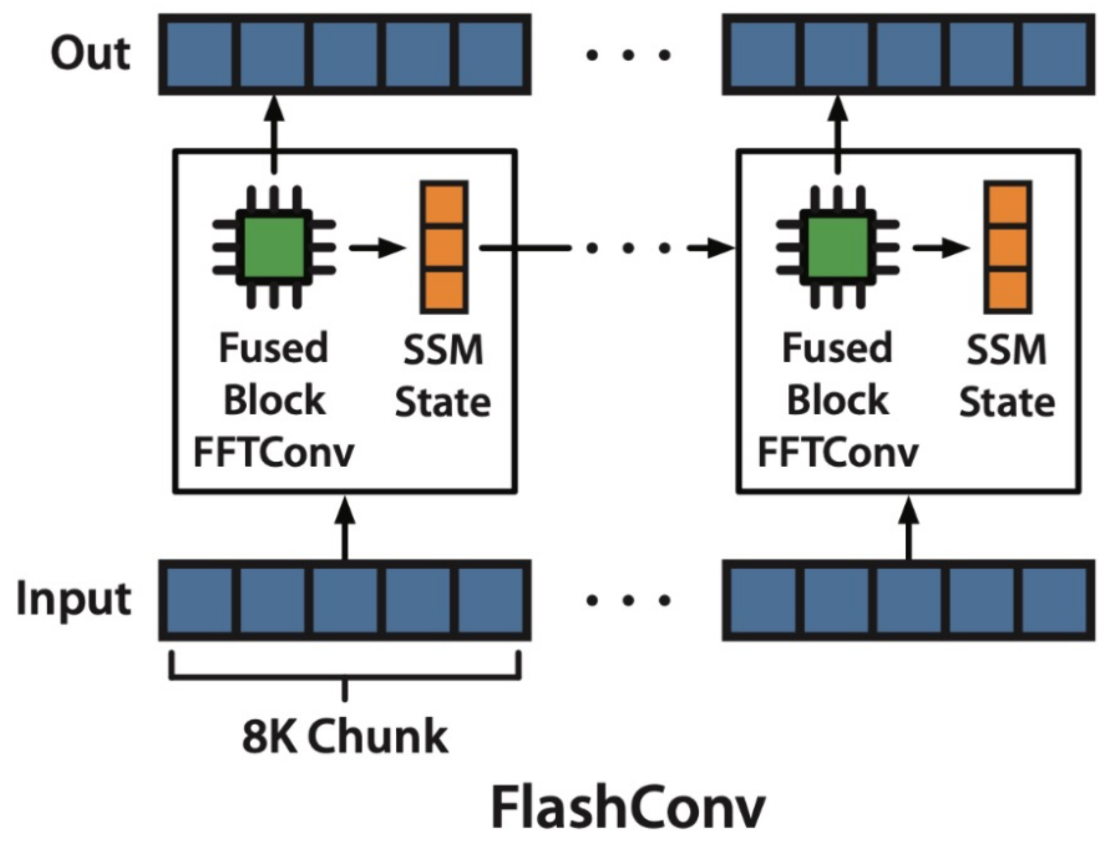
(1) Expressiveness: State-space models (e.g., S4) underperforms on discrete domains (text)

(2) Efficiency: SSMs scale as $O(N \log N)$ in theory, but is slower than attention ($O(N^2)$) in practice.

Approaches:

(1) Design SSMs with multiplicative interaction and discrete recurrence.

(2) Hardware-efficient (long) convolution



Model	Val perplexity on the Pile (lower better)
GPT Neo 1.3B	6.2
H3 + 2 attn (1.3B)	**6.0**
GPT Neo 2.7B	5.7
H3 + 2 attn (2.7B)	**5.4**

Fundamental algorithm and hardware-efficiency unlock promising approach to long context.

Outlines

FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

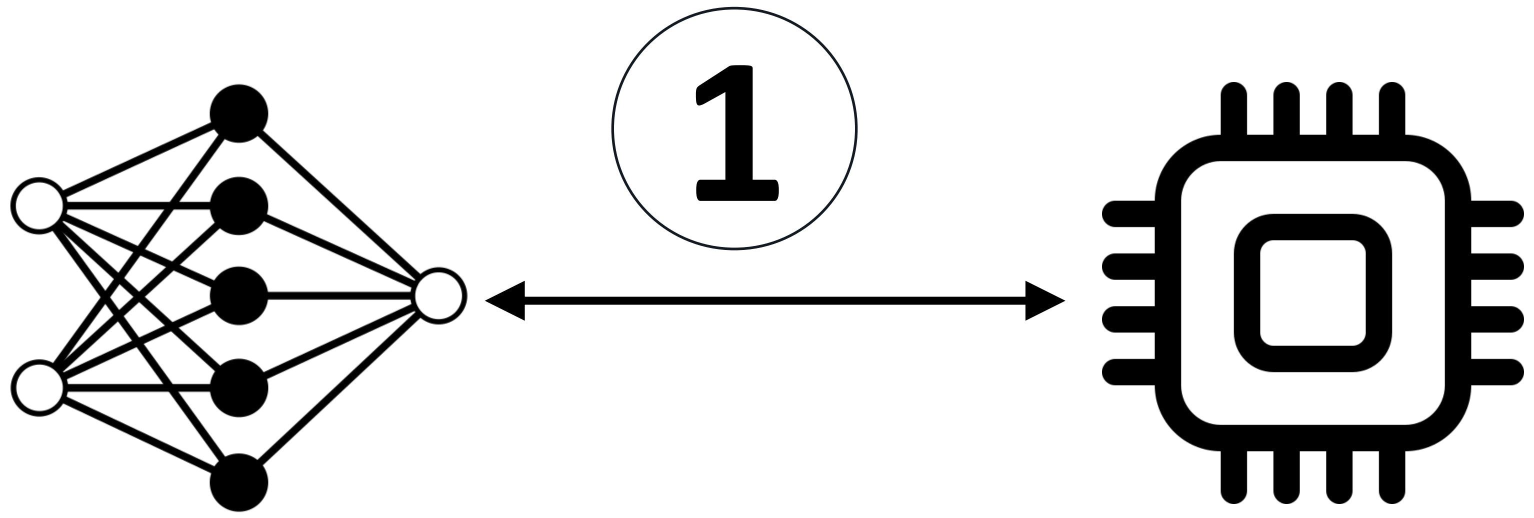
Future Directions

Software-hardware co-design, Long context for new workflow

Future Directions: Accelerate AI in the real world

1 Software-hardware Co-design:

Model

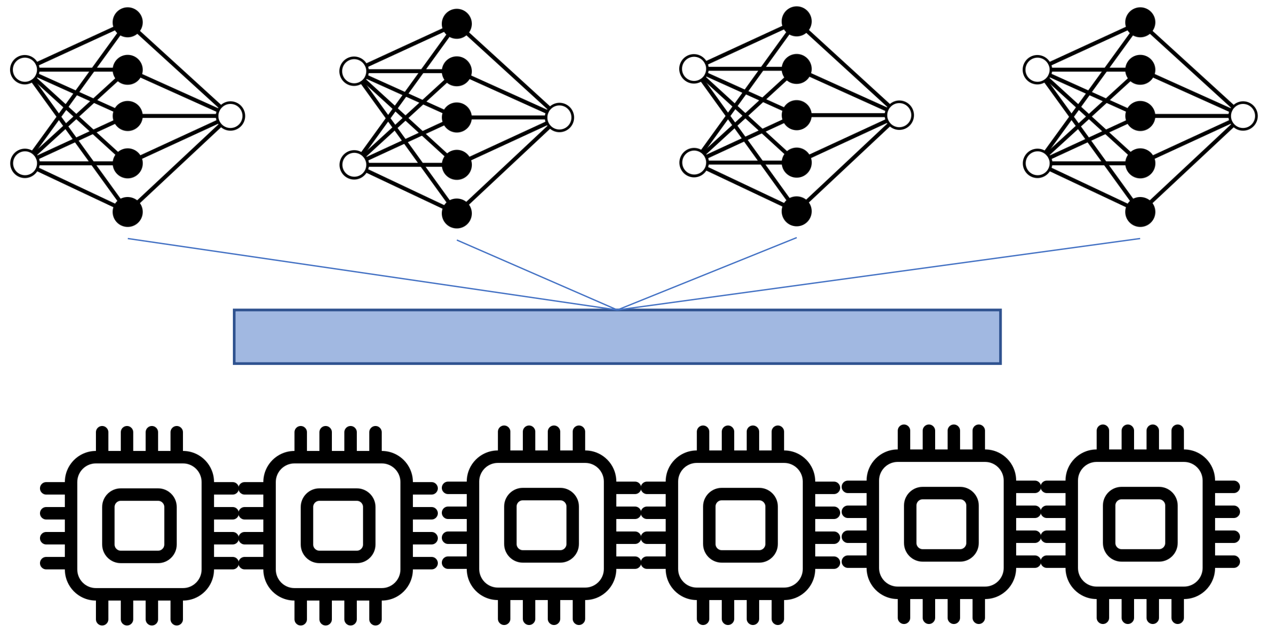


Hardware

PyTorch OpenAI NVIDIA

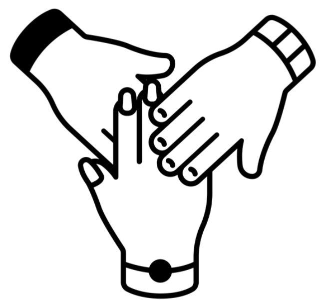
Future Directions: Accelerate AI in the real world

1 Software-hardware Co-design:



Smart compiler + runtime

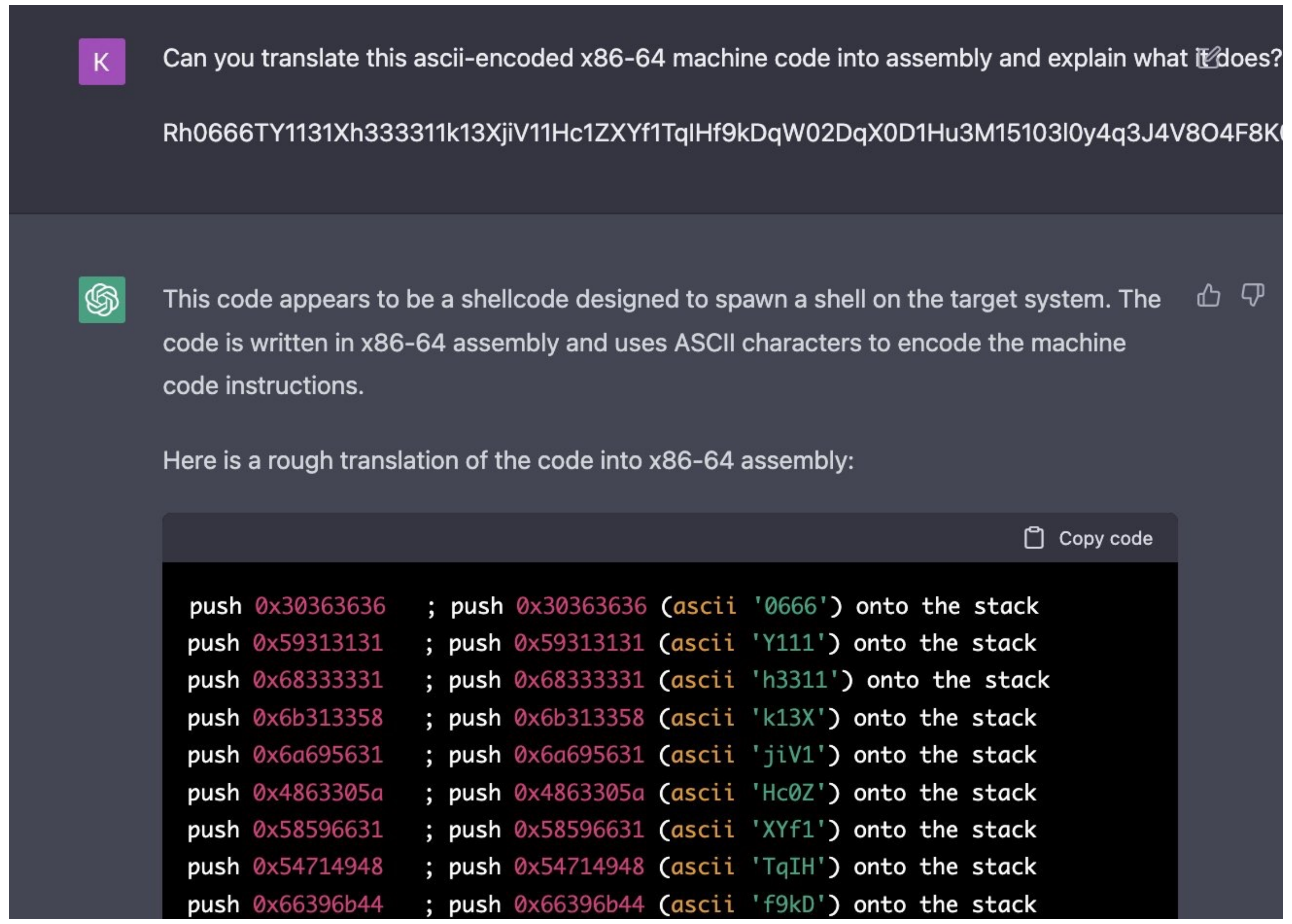
Sparse matrix multiplication
Data-flow architecture

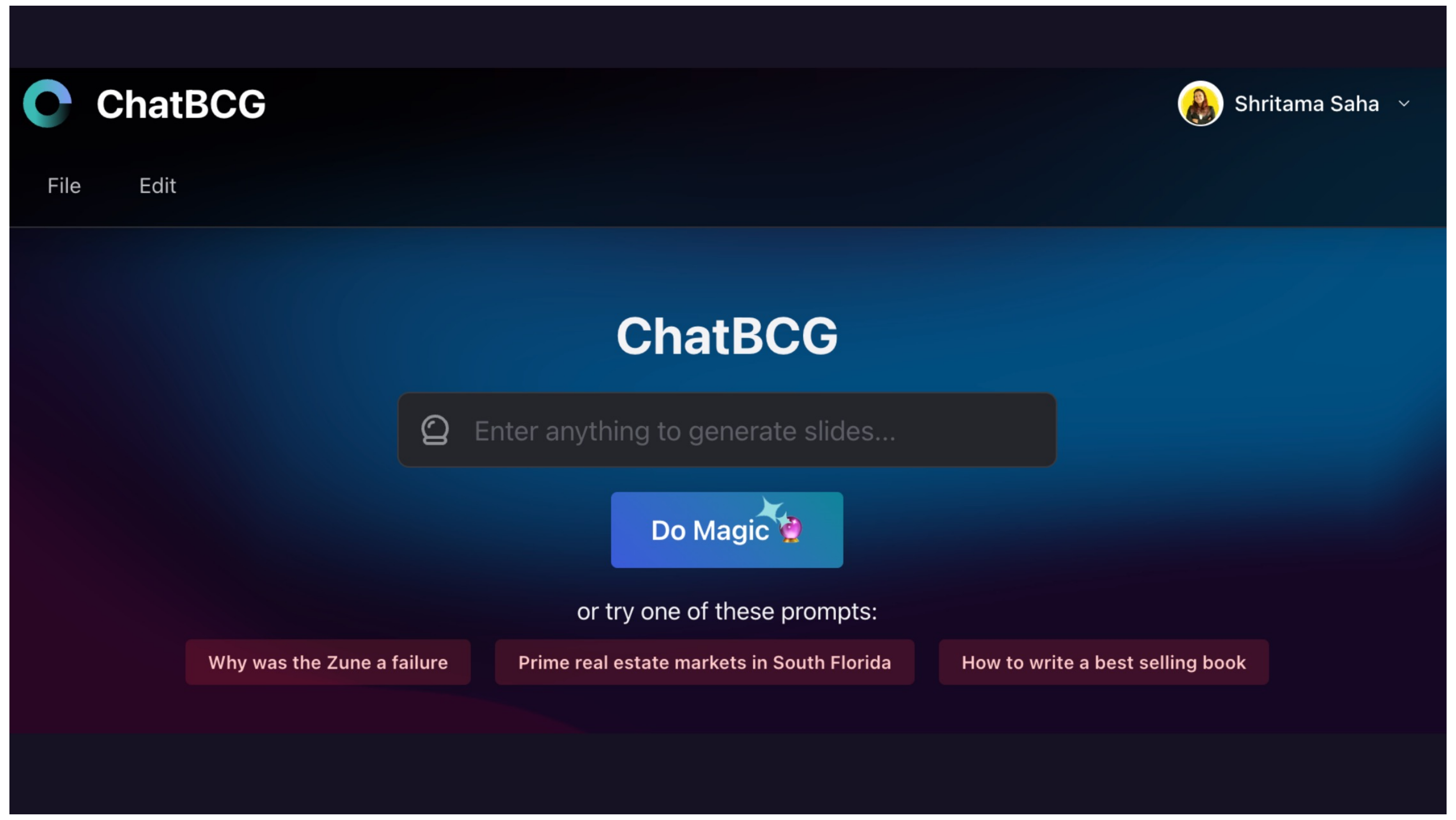


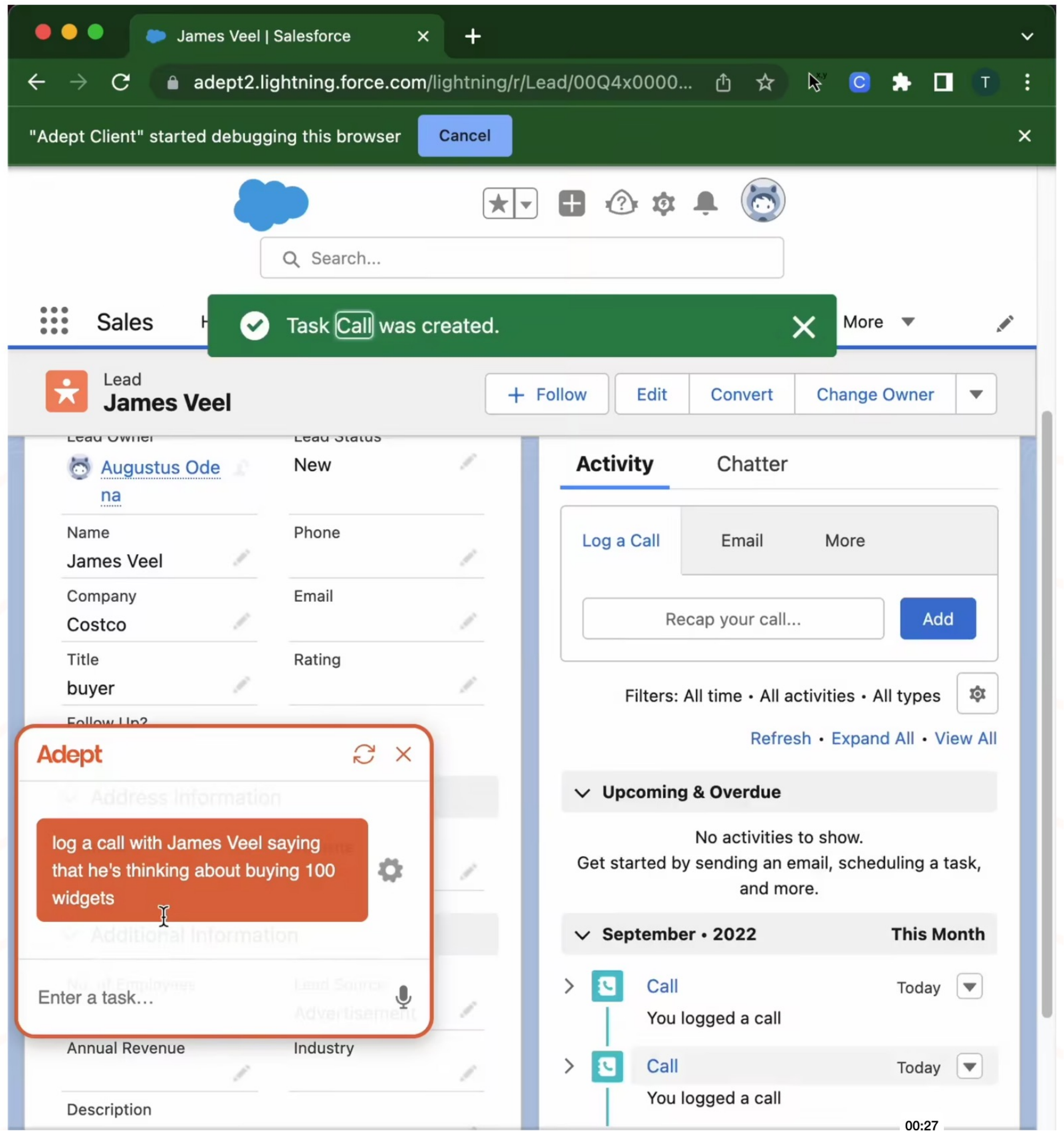
ML, Architecture, PL, Compiler, HPC, Networks …

Future Directions: Accelerate AI in the real world

2 Long context for new interactive AI workflows







Models	Context
GPT-3	2k
GPT-3.5 (ChatGPT)	4k
GPT-4	8k
GPT-4	32k