Efficient Training, Adaptation and Models

Bailin Wang

bailinw@mit.edu

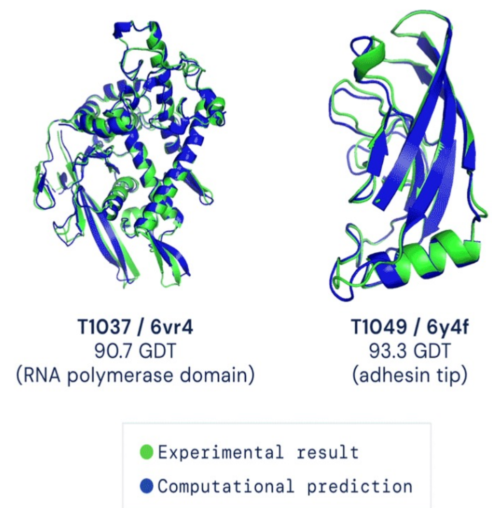
Slides adapted from [Tri Dao's lecture]

Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

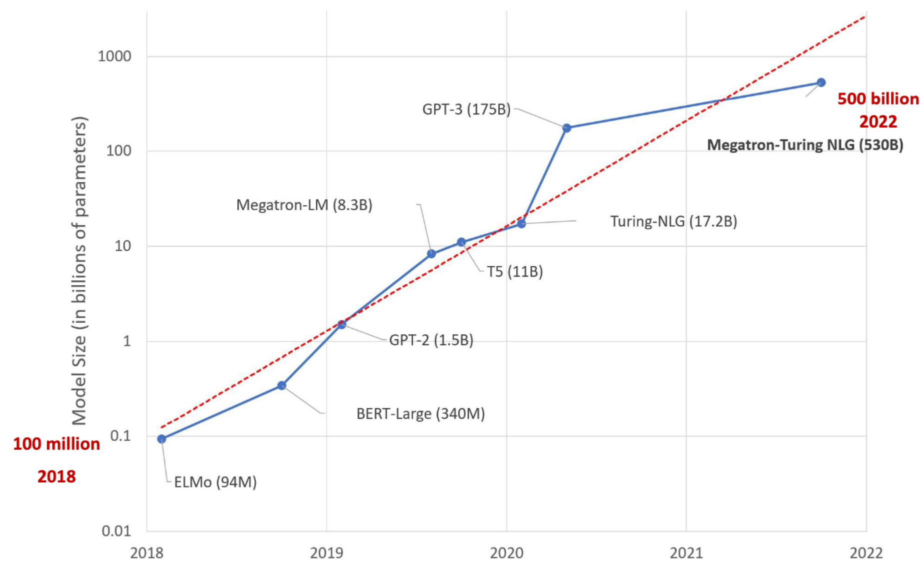
Generate Art
(Stable Diffusion – Stability.AI)

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

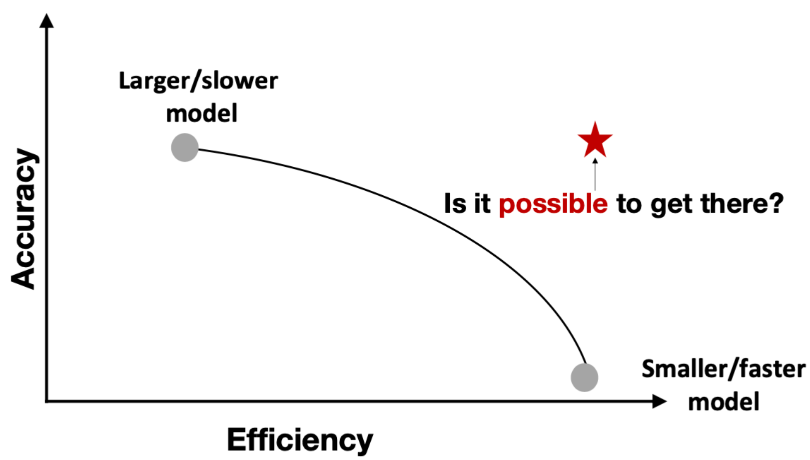
Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

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.

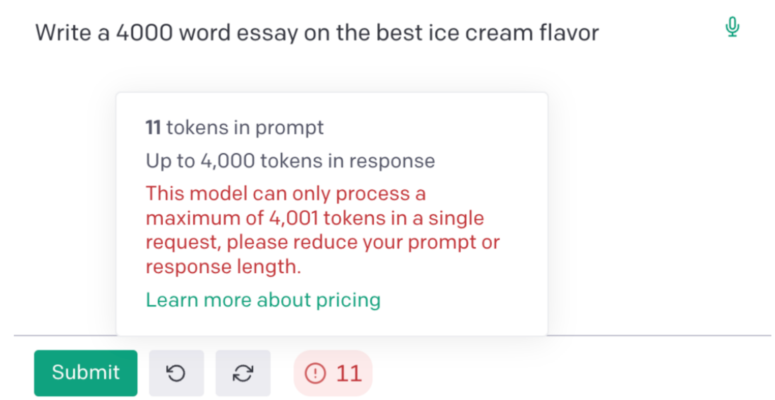
175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm.
The punchline, "I guess **no good seed goes unpunished**," is a play on the phrase "**no good deed goes unpunished**." In this case, "good seed" refers to a starting point for the random restarts, 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)

1. Efficient Training

Question: how to ***train*** *Transformers efficiently?*

2. Efficient Adaptation

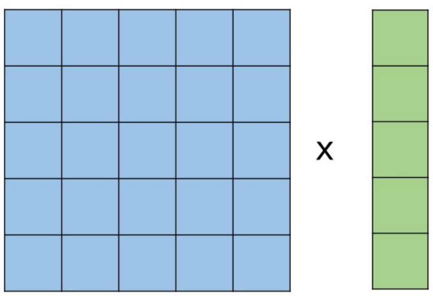
Question: how to ***finetune*** *Transformers efficiently?*

3. Efficient Model Architectures

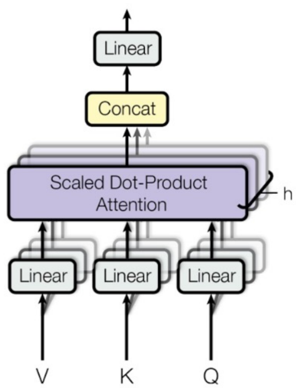
Question: how to design a new model with ***low inference cost***

Approach to Efficiency: Understanding Algorithms & Systems

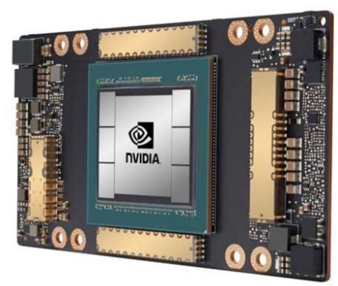
Fundamental algorithms	Hardware accelerators & distributed systems



Fast matrix-vector multiply



Attention mechanism

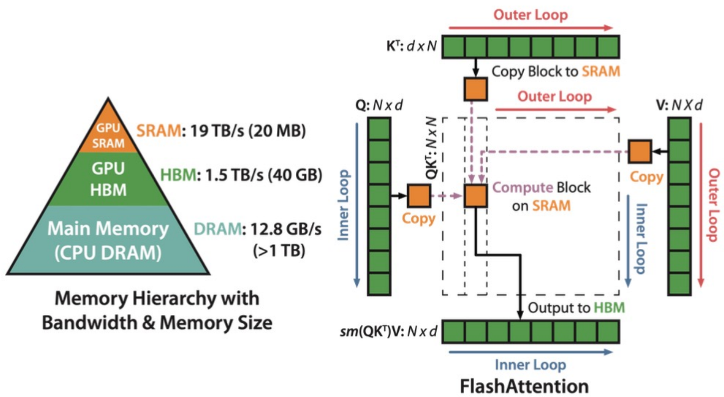


Block-oriented device

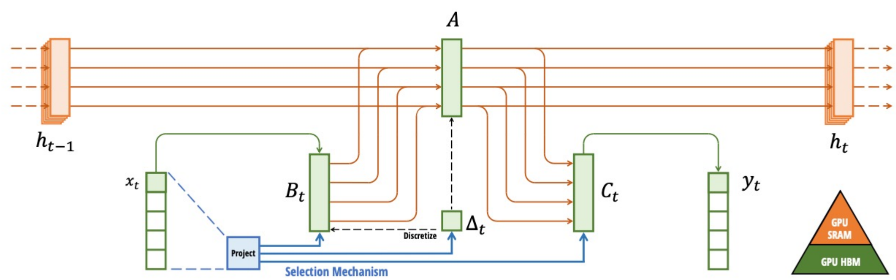
Asymmetric memory hierarchy

Main Idea: Hardware-aware Algorithms

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



State-space expansion:
expand recurrent states in SRAM only to avoid memory cost



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

D., Fu, Ermon, Rudra, Ré, NeurIPS 2022
D., 2023

Mamba: selective state-space model that matches Transformers on language model, with fast inference and up to 1M context

Gu*, D.*, 2023.

1. Flash-Attention

Motivation: Modeling Long Sequences

Enable New Capabilities

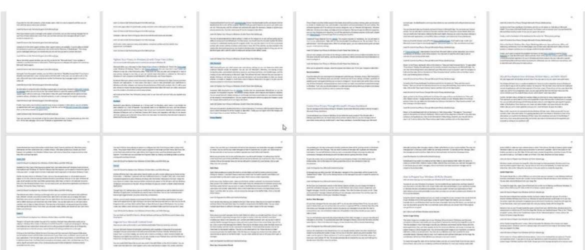
NLP: Large context required to understand books, plays, codebases.

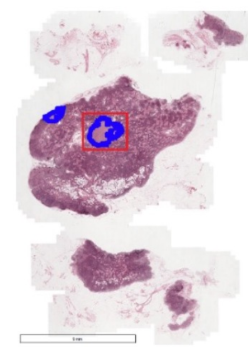
Close Reality Gap

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.

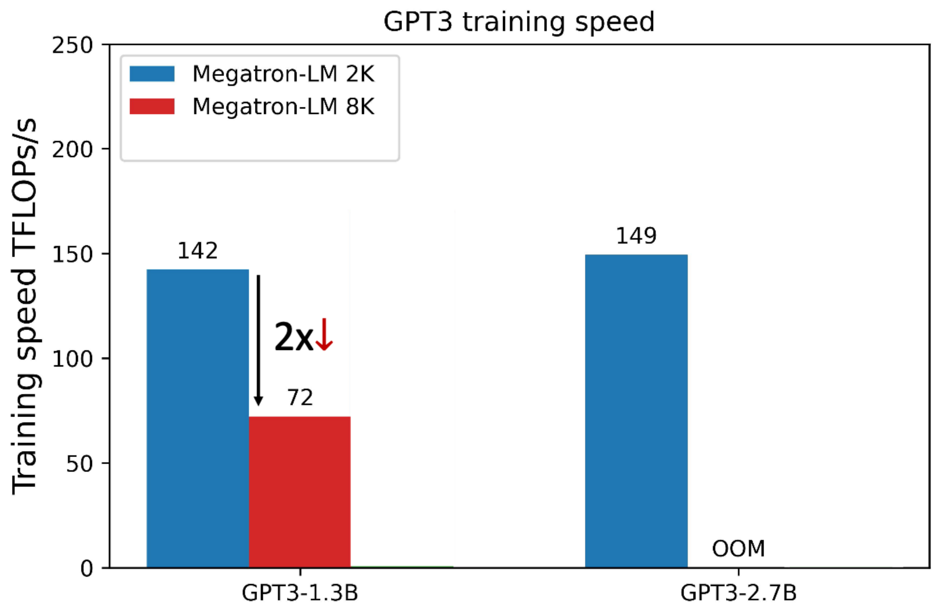




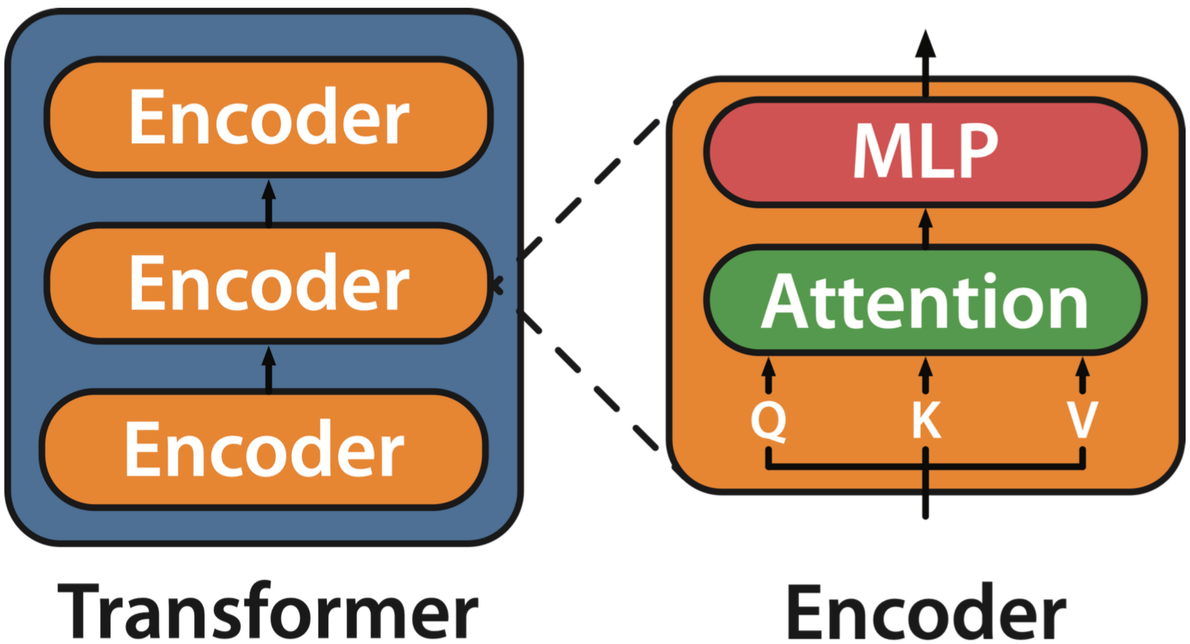
Efficiency is the Bottleneck for Modeling Long Sequences with Attention

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

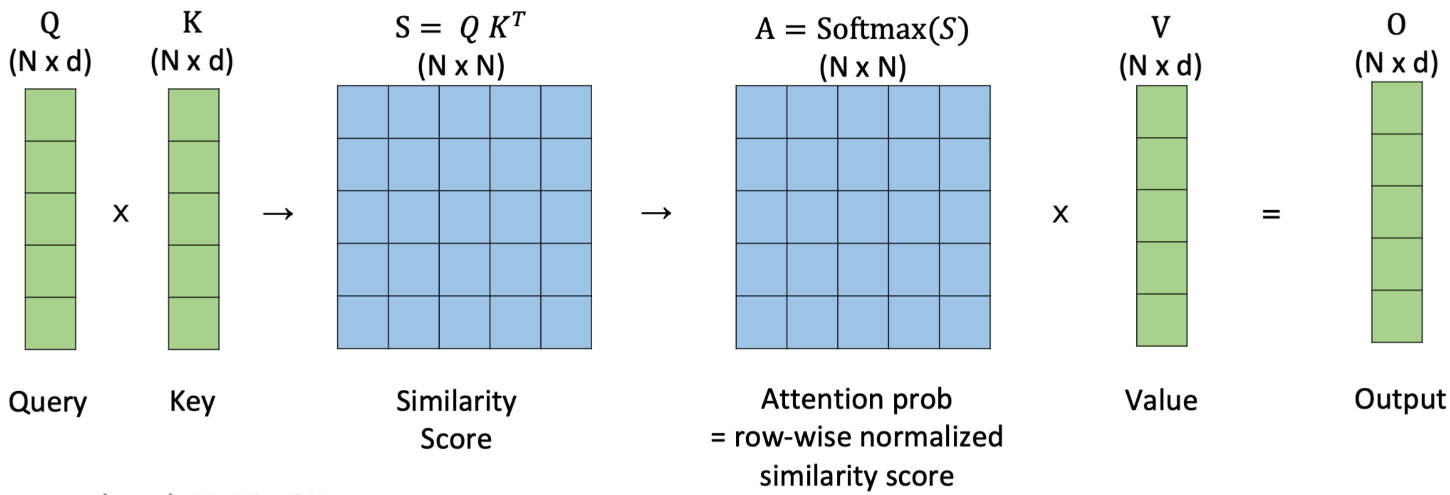
Increasing context length slows down (or stops) training



Background: Attention is the Heart of Transformers



Background: Attention Mechanism



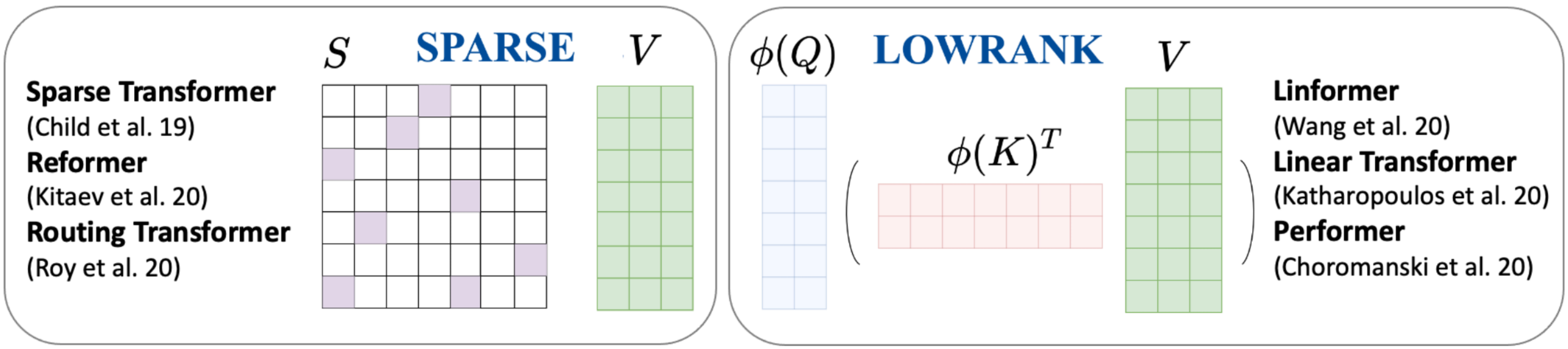
Typical sequence length N: 1K – 8K
Head dimension d: 64 – 128

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

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

Attention scales quadratically in sequence length N

Background: Approximate Attention

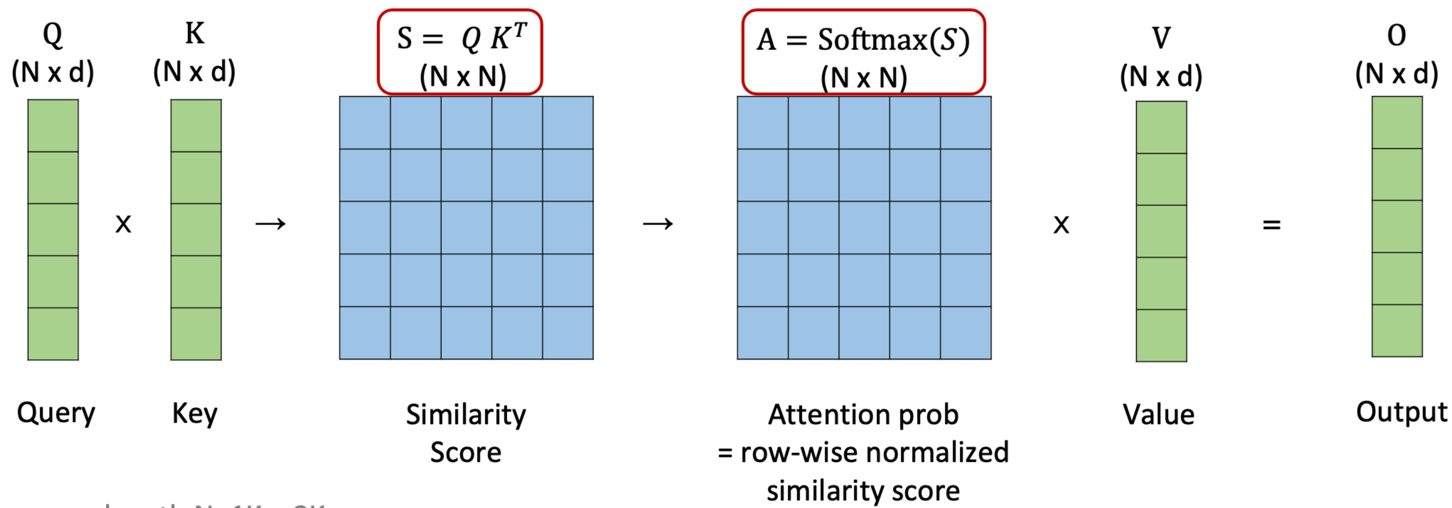


Approximate attention: tradeoff quality for ~~speed~~ fewer FLOPs

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

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

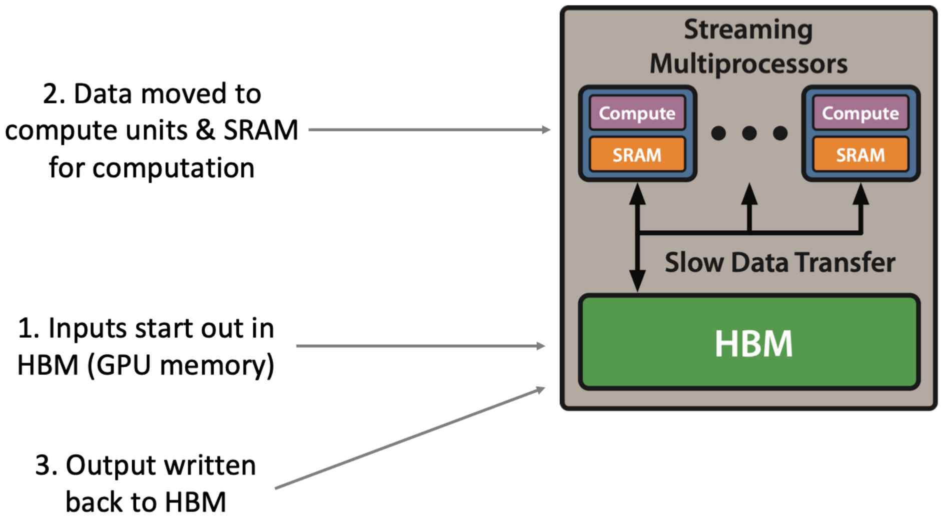
Our Observation: Attention is Bottlenecked by Memory Reads/Writes

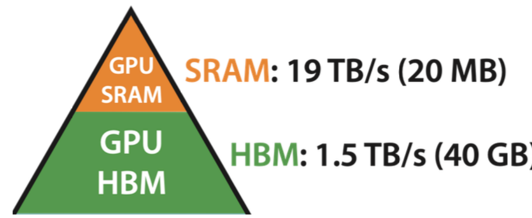


Typical sequence length N: 1K – 8K
Head dimension d: 64-128

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

Background: GPU Compute Model & Memory Hierarchy





[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.

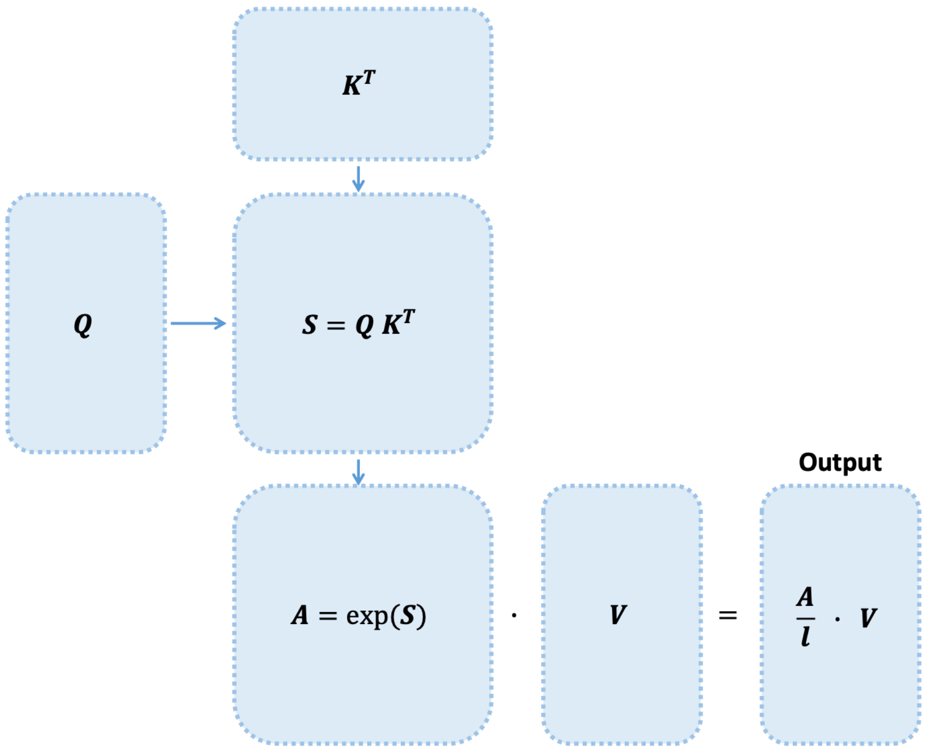
(2) Backward without the large attention matrix from forward.

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

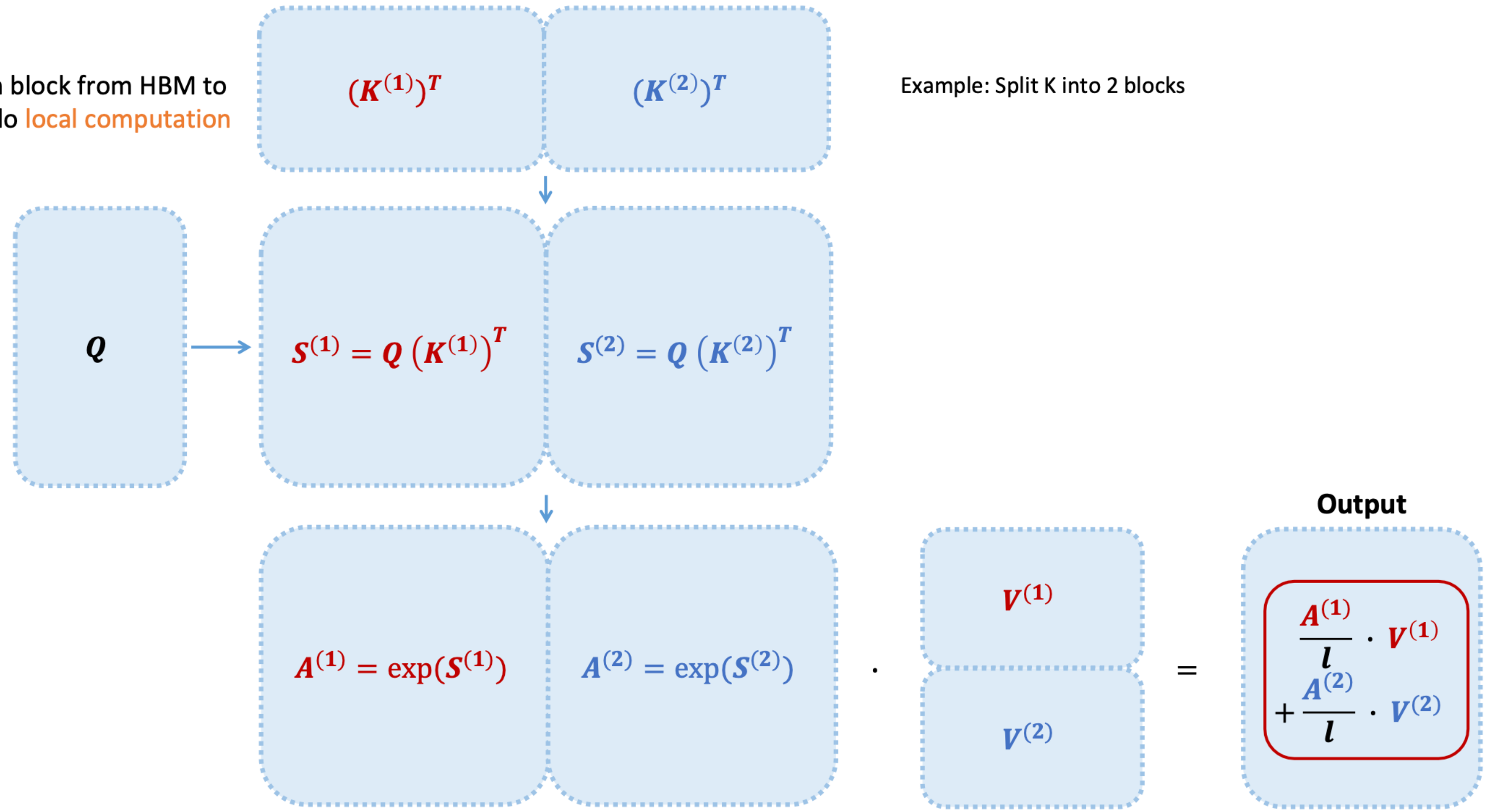
$$l = \sum_i \exp(S)_i$$

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

Example: Split K into 2 blocks



Softmax row-wise normalization constant

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

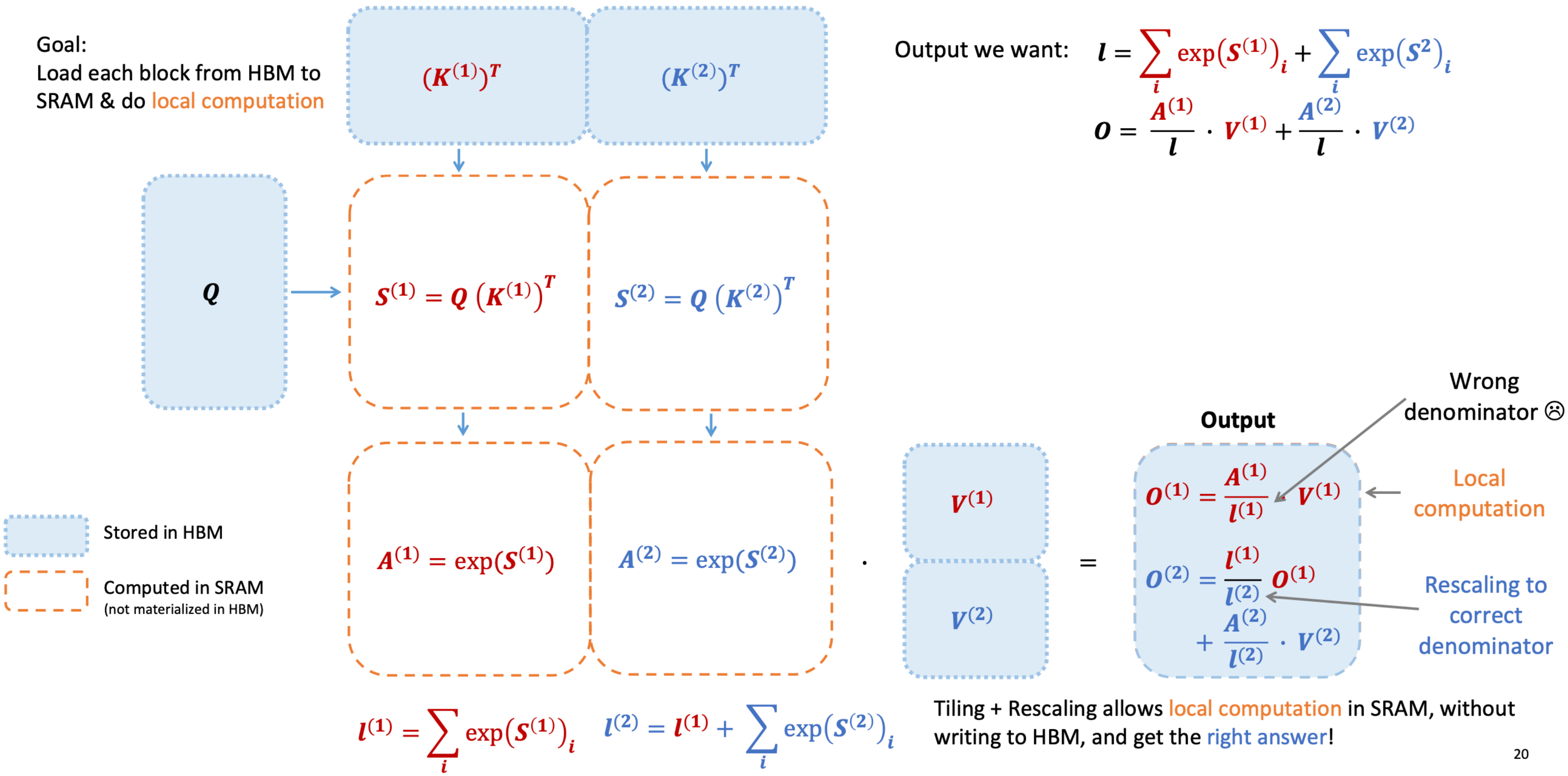
Challenge: How to compute softmax normalization with just local results?

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

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

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

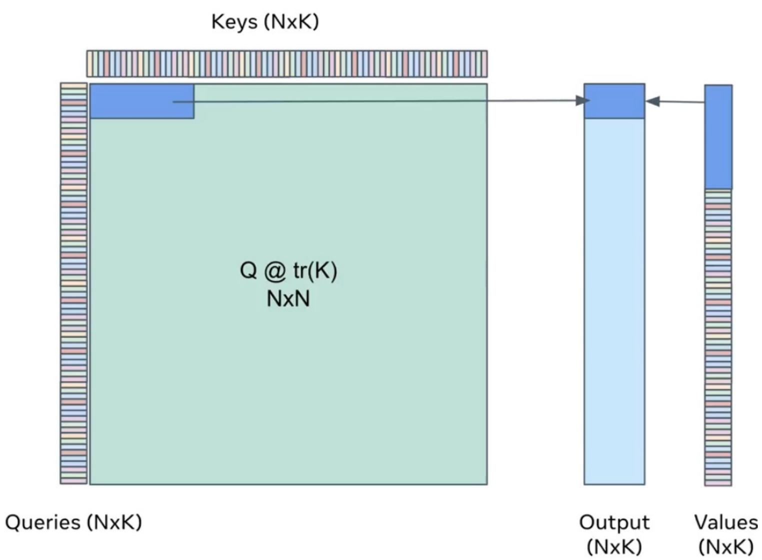
$$\boldsymbol{O} = \frac{\boldsymbol{A}^{(1)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(1)} + \frac{\boldsymbol{A}^{(2)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(2)}$$



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

Tiling

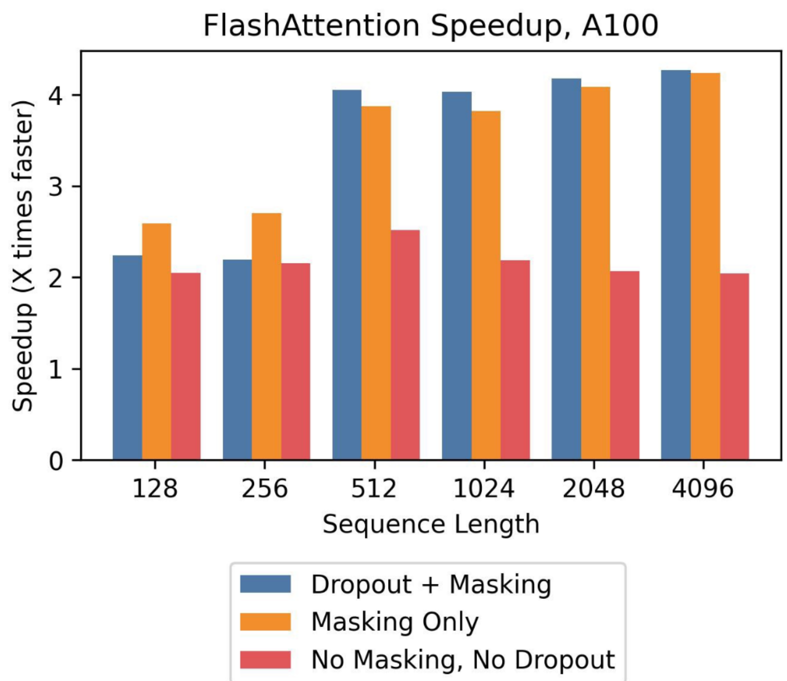
Decomposing large softmax into smaller ones by scaling.

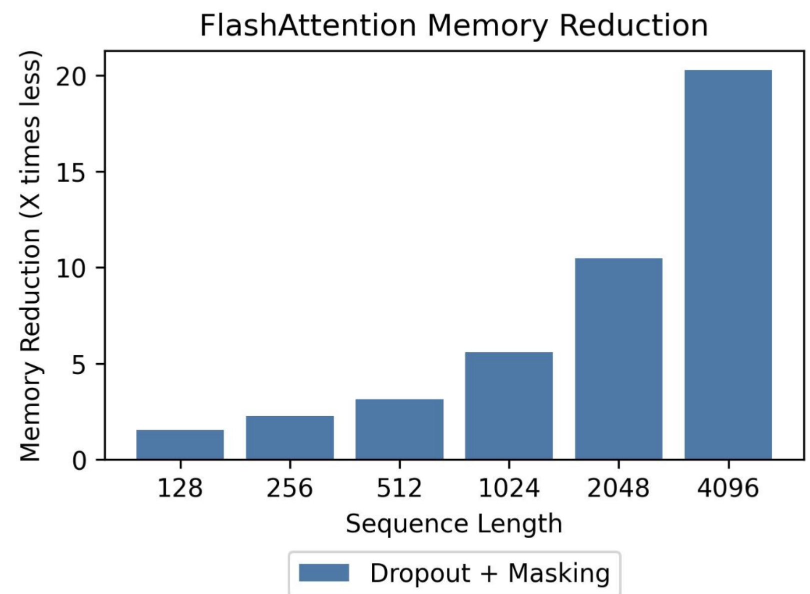


1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output with respect to that block.
3. Update output in HBM by scaling.

Animation credit: Francisco Massa

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

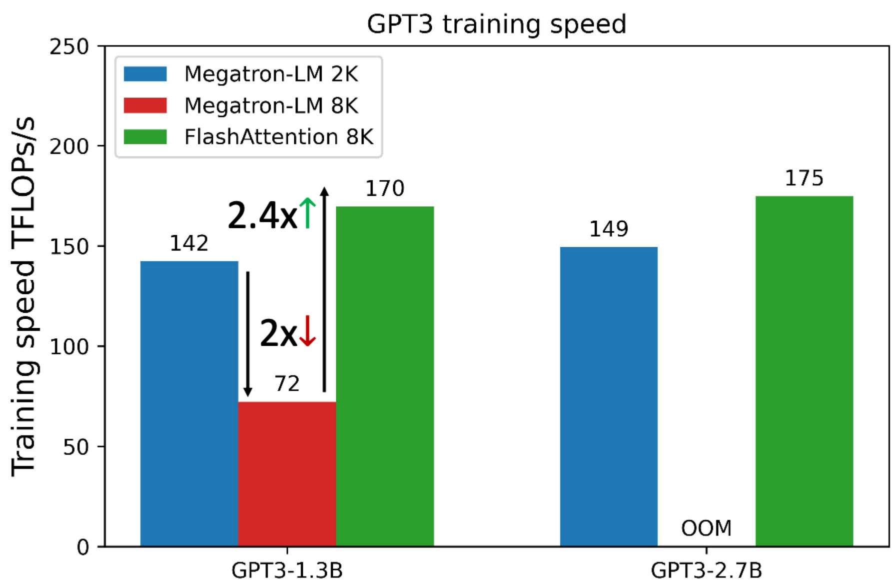




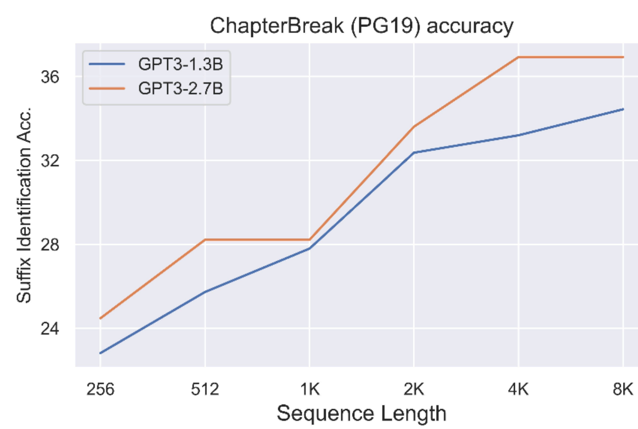
2-4x speedup — with no approximation!

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

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

2. Low-Rank Adaptation

Revisit the full fine-tuning

- Assume we have a pre-trained autoregressive language model $P_\phi(y|x)$
  - E.g., GPT based on Transformer

- Adapt this pretrained model to downstream tasks (e.g., summarization, NL2SQL, reading comprehension)
  - Training dataset of context-target pairs $\{(x_i, y_i)\}_{i=1,\ldots,N}$

- During full fine-tuning, we update ϕ_o to $\phi_o + \Delta\phi$ by following the gradient to maximize the conditional language modeling objective

$$\max_{\phi} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_\phi(y_t|x, y_{<t}))$$

LoRA: low rank adaptation (Hu et al., 2021)

- For each downstream task, we learn a different set of parameters $\Delta\phi$
  - $|\Delta\phi|$ = $|\phi_o|$
  - GPT-3 has a $|\ \phi_o\ |$ of 175 billion
  - Expensive and challenging for storing and deploying many independent instances

- Key idea: encode the task-specific parameter increment $\Delta\phi = \Delta\phi(\Theta)$ by a smaller-sized set of parameters Θ, $|\Theta| \ll |\phi_o|$

- The task of finding $\Delta\phi$ becomes optimizing over Θ

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\phi_o+\Delta\phi(\Theta)}(y_t|x, y_{<t}))$$

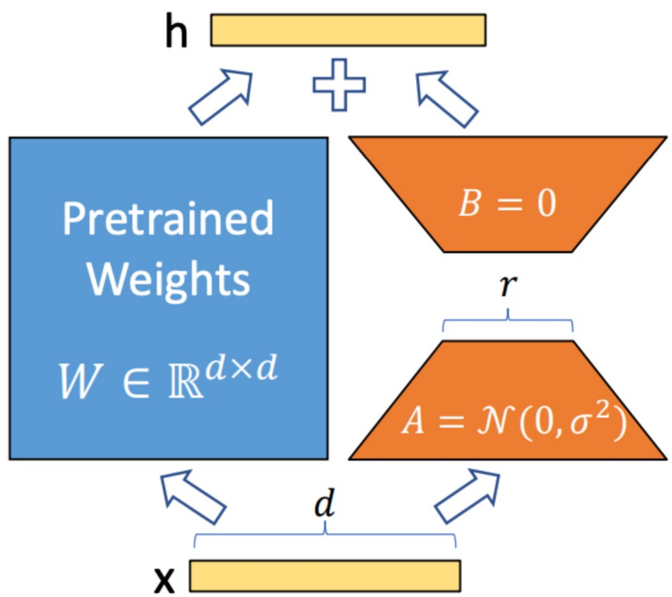
Low-rank-parameterized update matrices

- Updates to the weights have a low "intrinsic rank" during adaptation (Aghajanyan et al. 2020)
- $W_0 \in \mathbb{R}^{d \times k}$: a pretrained weight matrix
- Constrain its update with a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA$$

where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, r \ll \min(d, k)$

- Only A and B contain **trainable** parameters

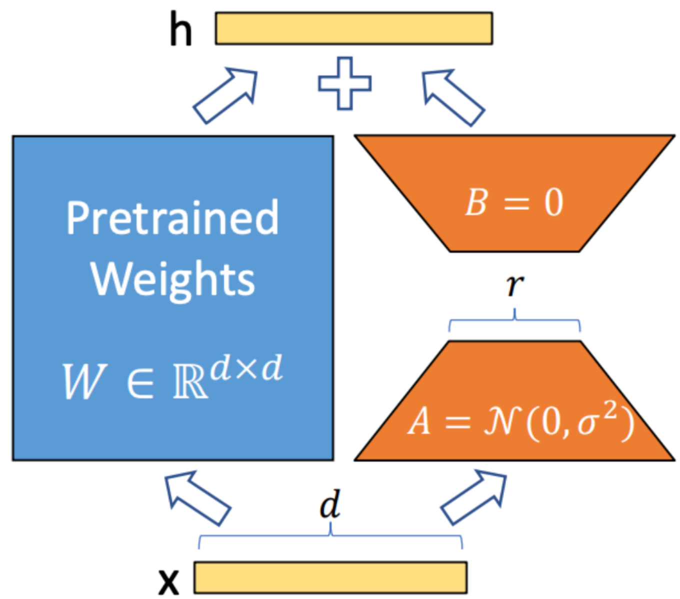


Low-rank-parameterized update matrices

- As one increase the number of trainable parameters, training LoRA converges to training the original model

- **No additional inference latency:** when switching to a different task, recover W_0 by subtracting BA and adding a different $B'A'$

- Often LoRA is applied to the weight matrices in the self-attention module



Applying LoRA to Transformer

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen.
"Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

Understanding low-rank adaptation

Which weight matrices in Transformers should we apply LoRA to?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Adapting both Wq and Wv gives the best performance overall.

What is the optimal rank r for LoRA?

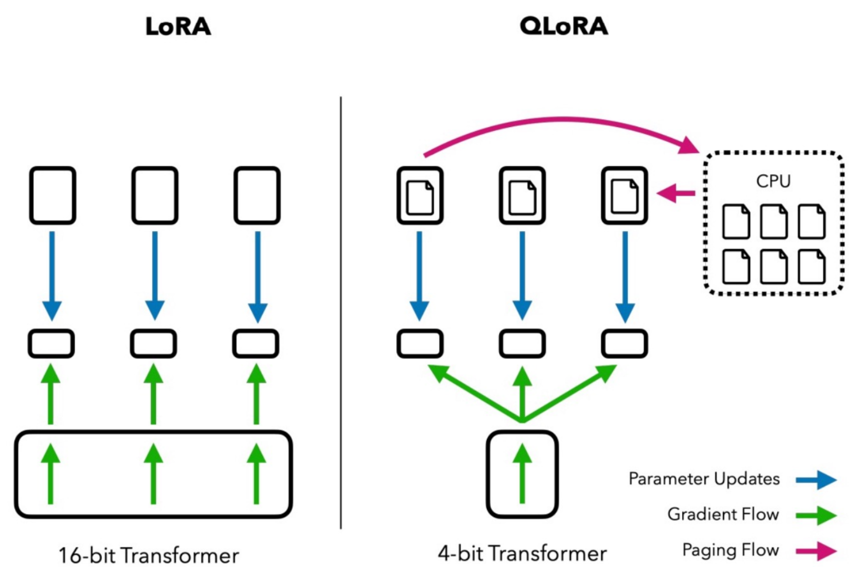
	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

LoRA already performs competitively with a very small r

From LoRA to QLoRA

- QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizer to handle memory spikes

- 4-bit NormalFloat (NF4)
  - A new data type that is information theoretically optimal for normally distributed weights



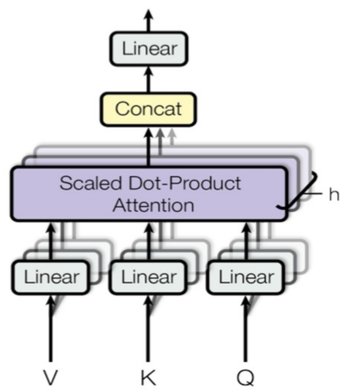
Dettmers, Tim, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. "Qlora: Efficient finetuning of quantized llms." arXiv preprint arXiv:2305.14314 (2023).

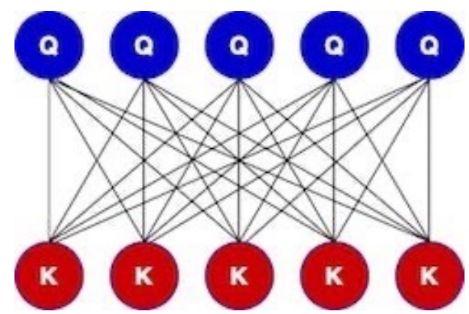
Summary- LoRA and QLoRA

- Low-rank adaptation for efficient finetuning
- QLoRA for finetuning quantized Transformers

3. Mamba

Attention (Transformers)



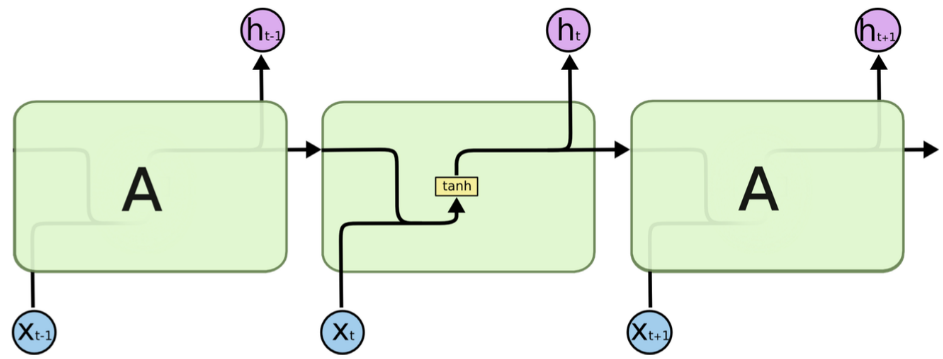


Dense interactions

✓ **Strong performance, parallelizable**

✗ **Quadratic-time training, linear-time inference (in the length of the sequence)**

Recurrent Neural Networks (RNN)

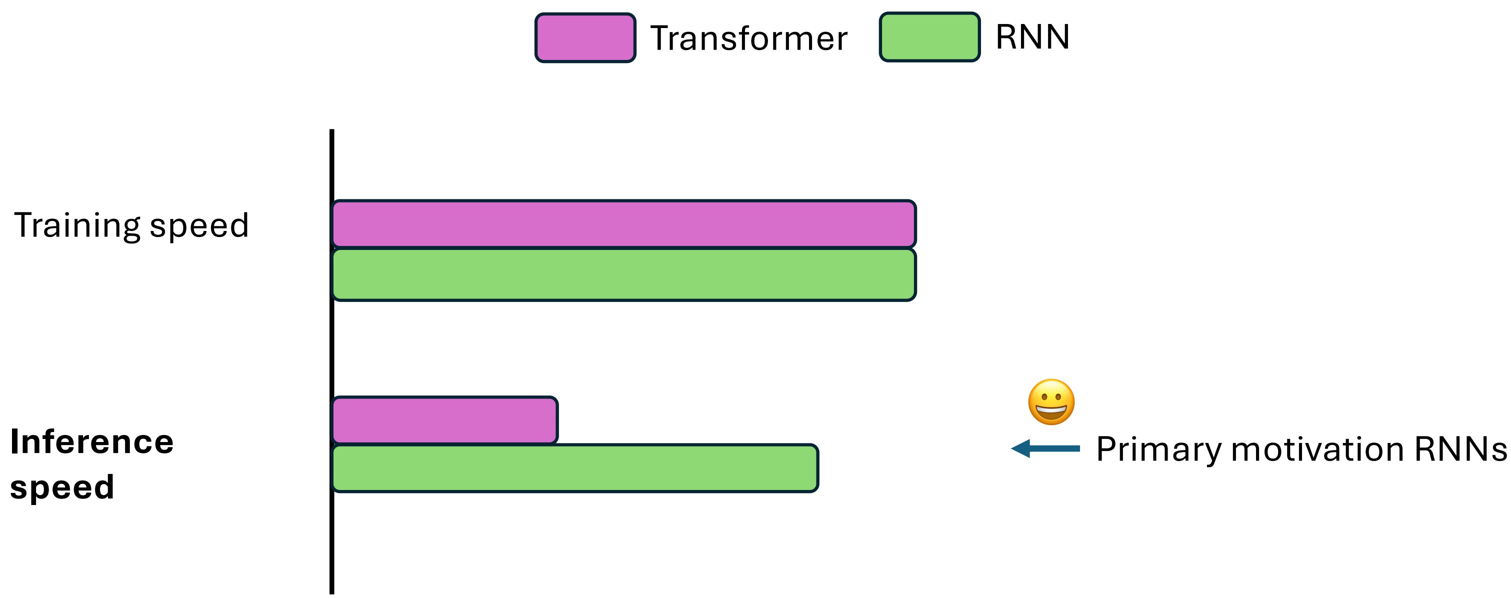


Sequential

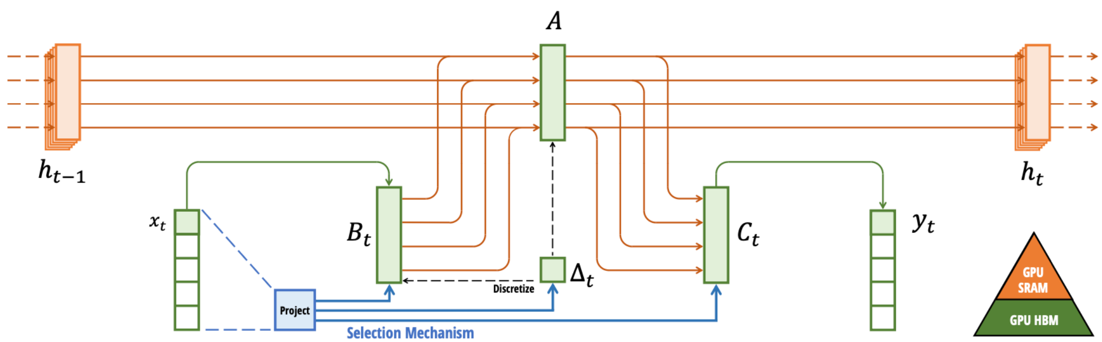
✓ **Natural autoregressive (causal) model**

✗ **Slow training on accelerators and poor optimization (vanishing gradients)**

RNN for Inference Efficiency



Selective State Spaces

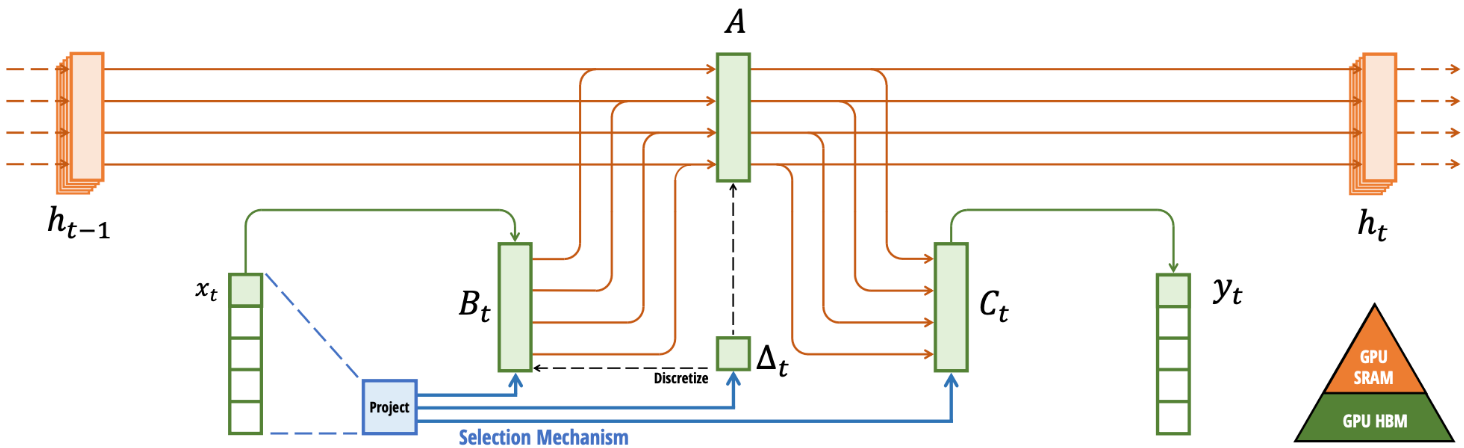


✓ **Efficiency: parallelizable training + fast inference**

✓ **Performance: matches Transformers on LM**

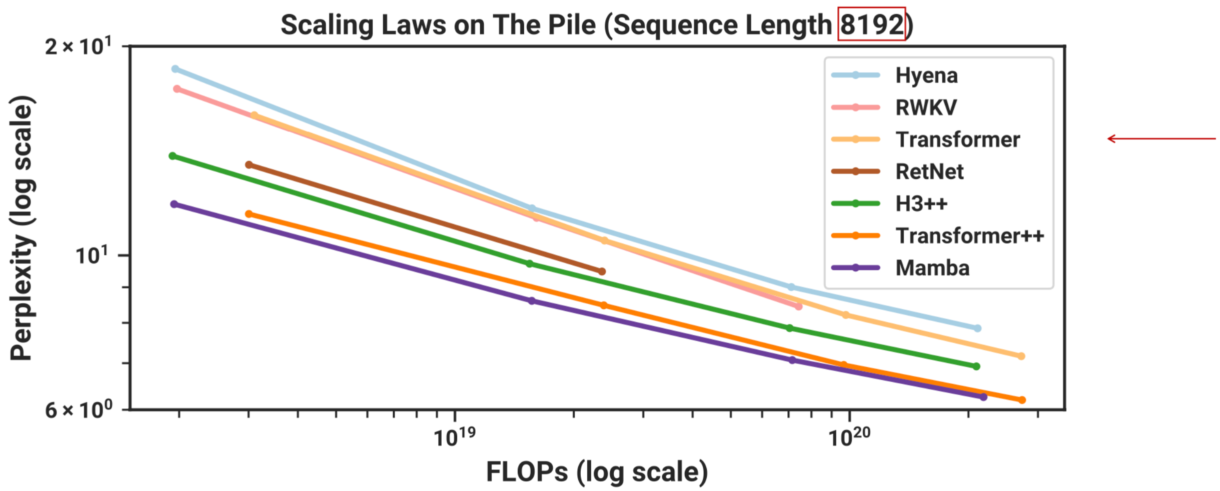
✓ **Long Context: improves up to million-length sequences**

Hardware-aware State Expansion



Idea: Only materialize the expanded state in more efficient levels of the memory hierarchy

Language Modeling – Scaling Laws



Transformer: GPT-3 model + training recipe

Language Modeling – Zero-shot Evals

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

Mamba matches/beats Transformers of similar size

Summary – Mamba

Match or beat strongest Transformer architecture on language

Key algorithmic ideas: **selection mechanism, hardware-aware state expansion**

Upshot: **better** models with **linear (instead of quadratic)** scaling in sequence length