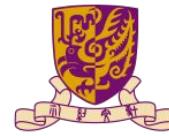


CSC6203: Large Language Model

Lecture 5: Efficiency in LLMs



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Fall 2024
Benyou Wang
School of Data Science

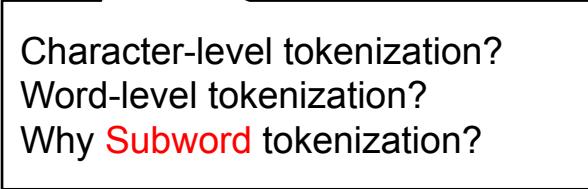
Recap

Overview

- LLM training
 - LLM Pretraining (including Word Tokenization)
 - Instruction Finetuning
 - Reinforcement Learning from Human Feedback
- LLM Evaluation

Tokenization

- Before Tokenization : **This is tokenizing**
- After Tokenization : **This is tokenizing**



Character-level tokenization?
Word-level tokenization?
Why **Subword** tokenization?

Pre-training

Example plain text (**do not need supervised data, e.g. web and books**)

The Large Language Model (LLM) represents a cutting-edge innovation in the field of artificial intelligence, harnessing vast amounts of textual data to provide nuanced responses and generate coherent narratives. As a descendant of OpenAI's renowned GPT series, the LLM showcases the rapid evolution of machine learning capabilities, embodying an unparalleled ability to comprehend, generate, and assist in myriad linguistic tasks. This technological marvel encapsulates the collective knowledge of countless sources and offers a tantalizing glimpse into the future of human-computer symbiosis, where the boundaries between natural and artificial intelligence become increasingly blurred.



Training with purely next word (token) prediction

Instruction fine-tuning (supervised fine-tuning)

Usually a triplet (**instruction**, **input**, **output**) -- **need supervised data**

Instruction:	I am looking for a job and I need to fill out an application form. Can you please help me complete it?
Input:	Application Form: Name: _____ Age: _____ Sex: _____ Phone Number: _____ Email Address: _____ Education: _____ ...
Output:	Name: John Doe Age: 25 Sex: Male Phone Number: ...



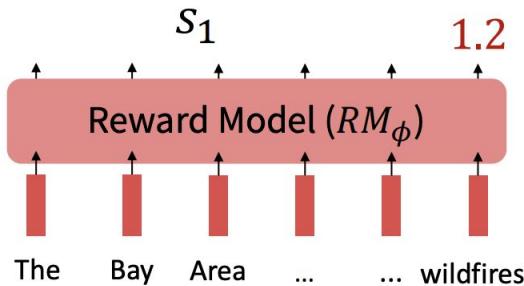
Training with next word (token) prediction but usually only for the **output** part

Reinforcement Learning from Human AI Feedback

An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.

A 4.2 magnitude
earthquake hit
San Francisco,
resulting in
massive damage.

The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.

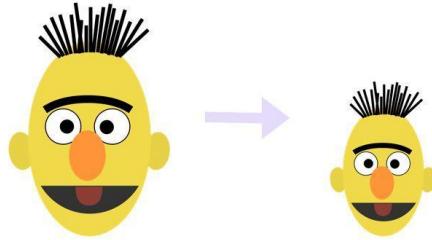


S_3 S_2

Bradley-Terry [1952] paired comparison model

$$J_{RM}(\phi) = -\mathbb{E}_{(s^w, s^l) \sim D} [\log \sigma(RM_{\phi}(s^w) - RM_{\phi}(s^l))]$$

“winning” sample “losing” sample s^w should score higher than s^l

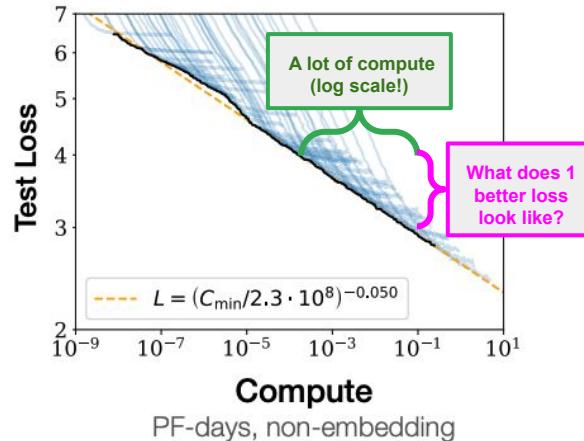
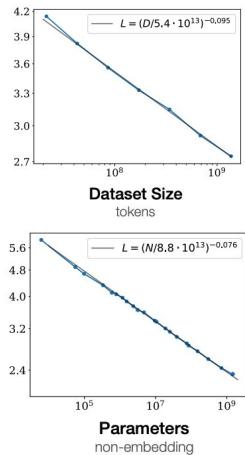


Today's main course: efficiency

又想马儿不吃草，也想马儿跑得快！

Why do we need efficiency?

LLMs follow Scaling Laws



[Scaling laws for neural language models \(2020\).](#)

[Kaplan et al., 2020:](#)
Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute for training.

Jason's rephrase: You should expect to get a better language model if you scale up model size, dataset size, and amount of compute.

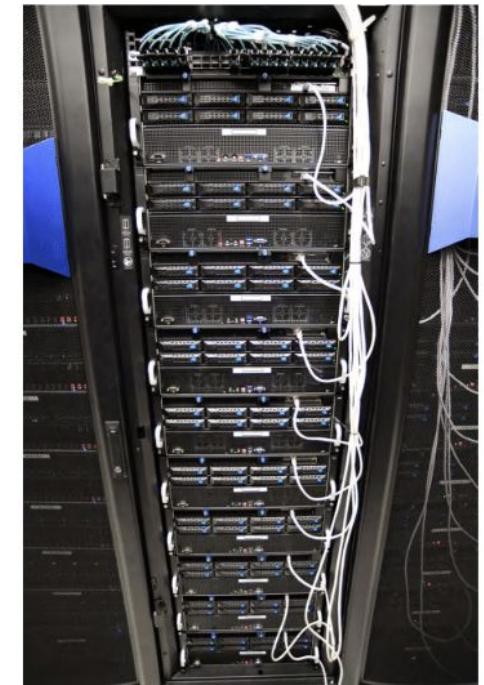
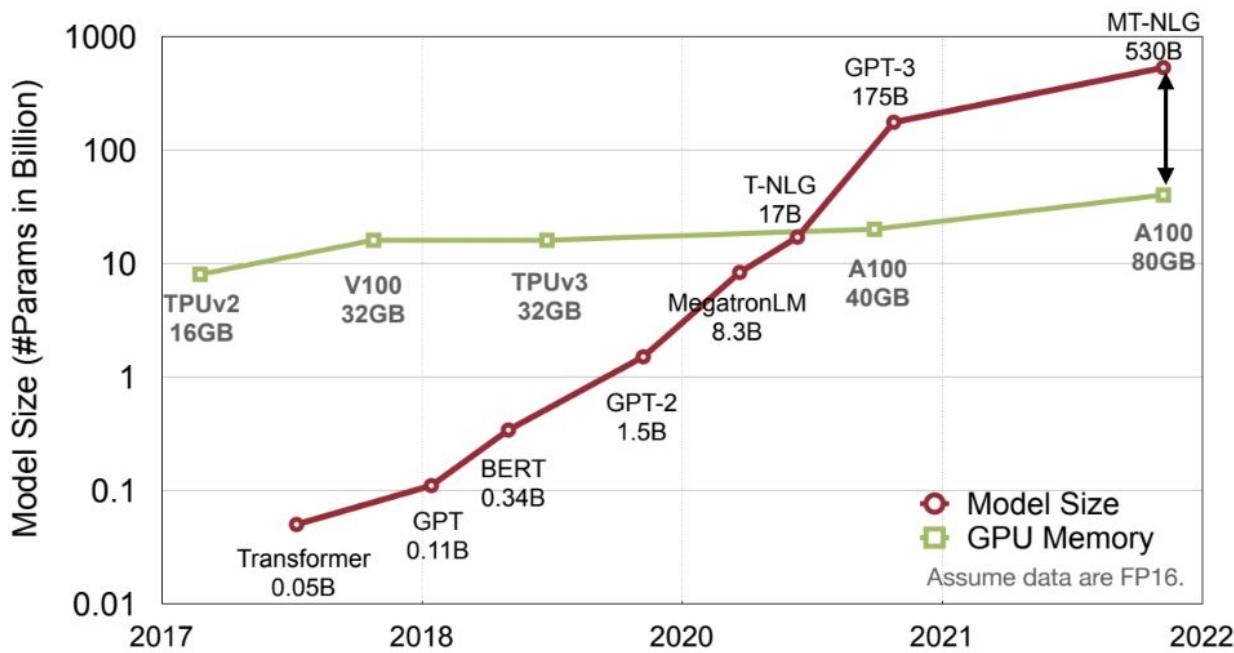
Note 1: "Within reasonable limits, performance depends very weakly on architectural hyperparameters such as depth and width."

Note 2: By the way, data is unlabeled (self-supervised) data from the internet (e.g., common crawl).

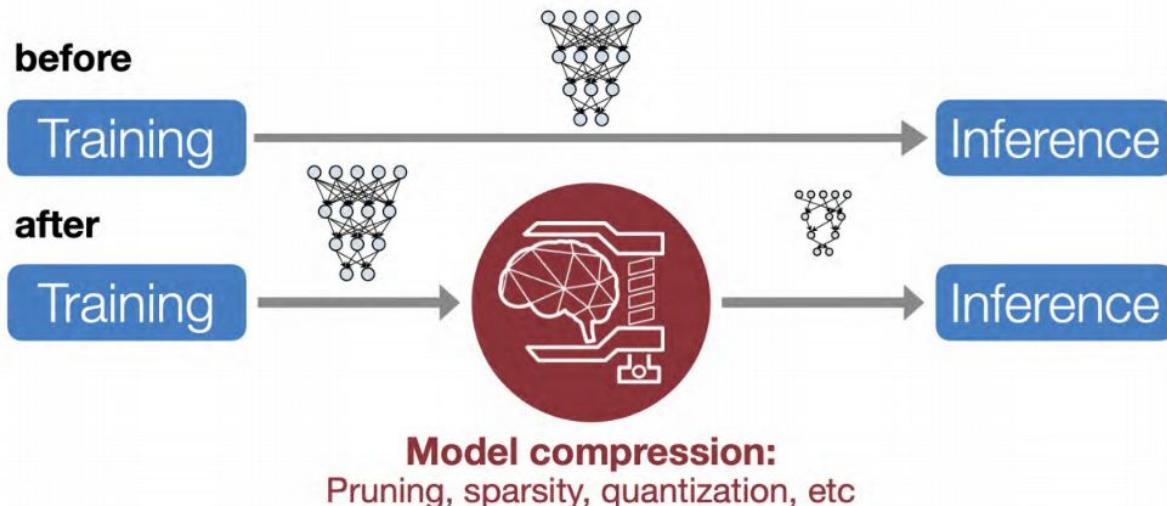
Suggested further reading:

[Scaling laws for neural language models \(2020\).](#)
[Training compute-optimal large language models \(2022\).](#)

Model size of LMs is growing exponentially, yet the hardware...



Bridge the gap between the supply and demand of AI computing



Outline

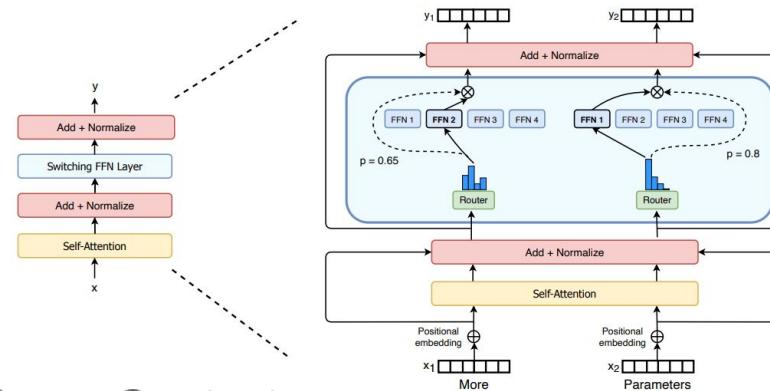
1. Overview of Efficiency in LLMs
2. Efficiency before LLMs
 - o Efficiency **beyond** Transformer (Quantization, Pruning, Knowledge distillation etc.)
 - o Efficiency **within** Transformer (*Sparsity* e.g., Mixture of Expert and efficiency in *long context*)
 - o **Parameter-efficient** finetuning (Lora, adapter, prompt, moe) and modularization
3. Efficiency beyond the LLM
 - a.
 - b. **Memory-efficient** Training: ZeRO, LOMO, Distributed Training, Flash-Attention, Ziniu, Lora/QLORA, Language model Inference: Early existing and Speculative decoding
5. Future direction

Outline

- Efficiency within Transformer
 - Long Attention (e.g., Quadratic Computing → Leaning Computing, w.r.t. the Sequence Length)
 - Sparsity (e.g., Mixture of Expert)
 - Mamba (RNN-style Transformer)
- Efficiency beyond Transformer
 - Quantization
 - Pruning
 - Knowledge Distillation
- Efficiency after LLMs
 - Distributed Training
 - Memory Saving
 - Communication Costs
- ~~Others (e.g., parameter compression and parameter sharing)~~
 - Parameter compression does not necessarily lead to faster inference.

Efficiency within Transformer

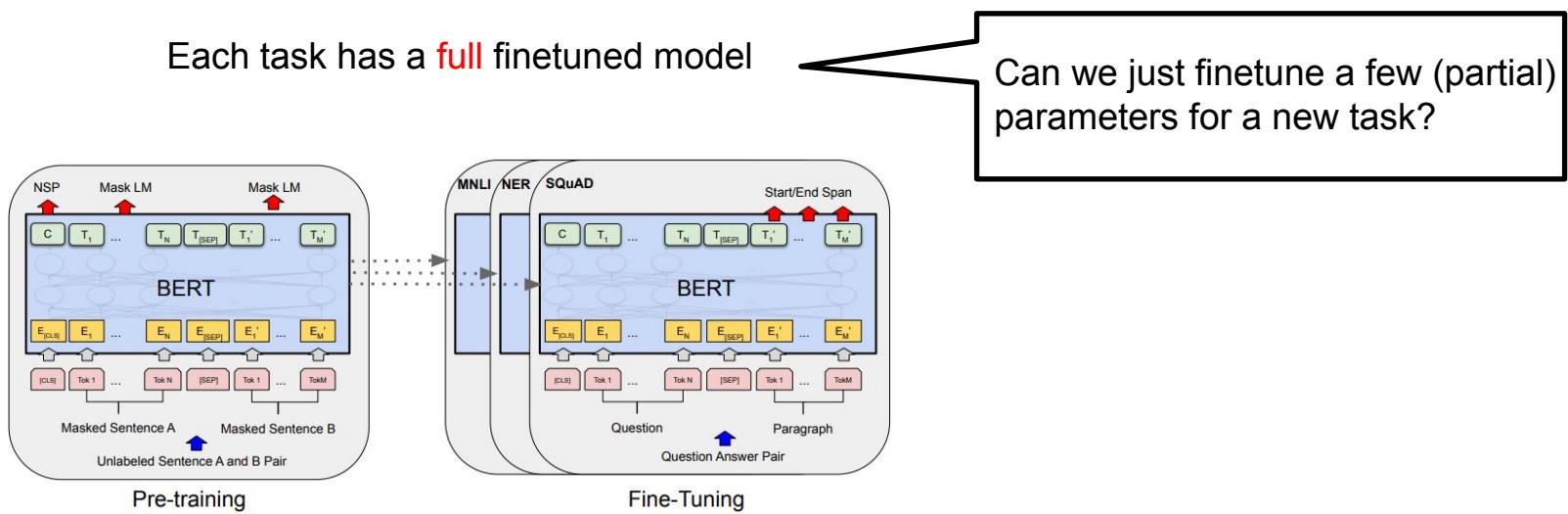
- Sparsity (e.g., Mixture of Expert [1])



- Efficiency for Long Context

- Computing complexity of $O(N^2D)$, which is quadratic to the sequence length

From time-efficiency vs. space-efficiency



Parameter-efficient finetuning

Efficiency beyond Transformer



Quantization
“Low
resolution”



Pruning
Removing weight
connections



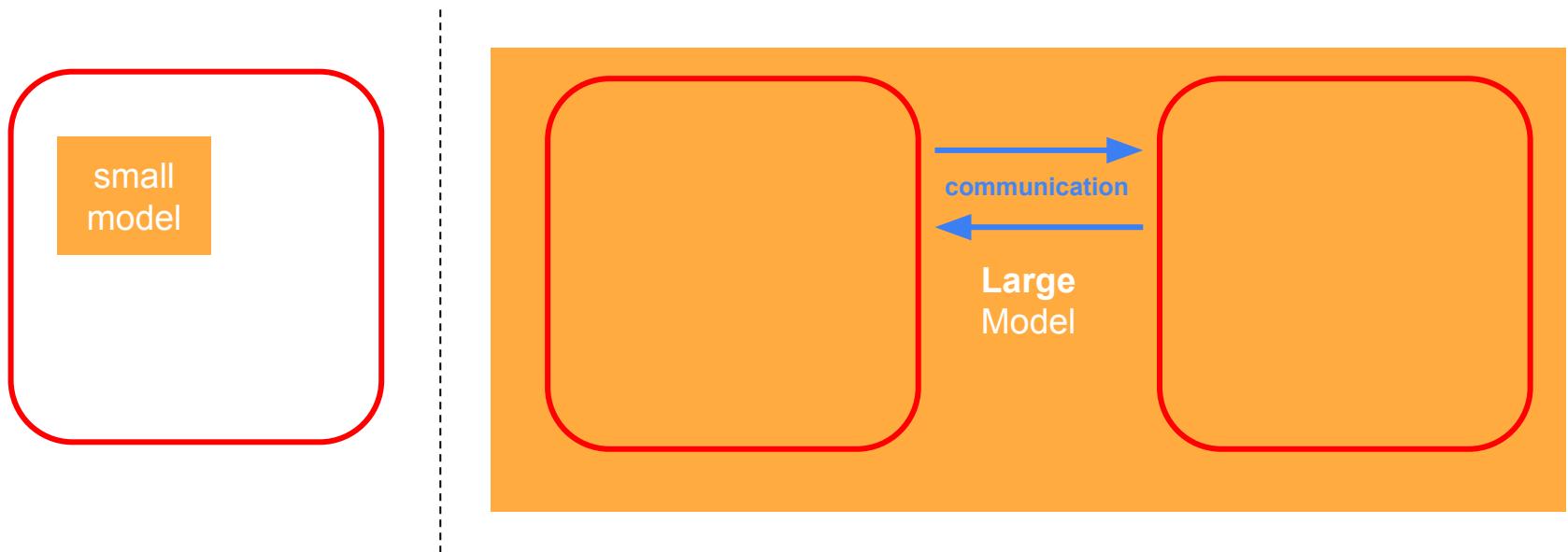
Knowledge distillation
Learning smaller models from big
ones

<https://rasa.com/blog/compressing-bert-for-faster-prediction-2/>

Efficiency after LLMs

- All Efficiency methods before LLMs (within/beyond transformer)
- + **Communication costs**
- + **Memory saving**
- Speculative decoding (only for language model)

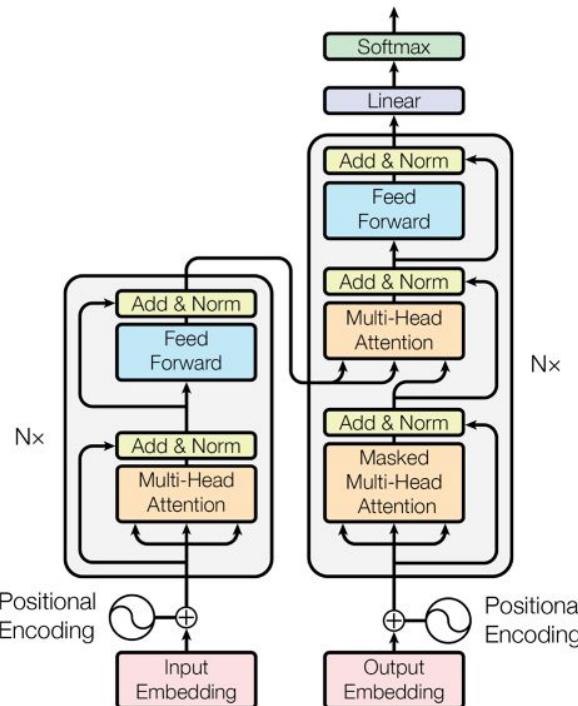
Space-efficient: from parameter to **memory**



In LLM era, a **whole model** might not be stored in a **GPU memory**, **communication costs** may be the bottleneck!

Efficiency within Transformers

Recap: Transformers



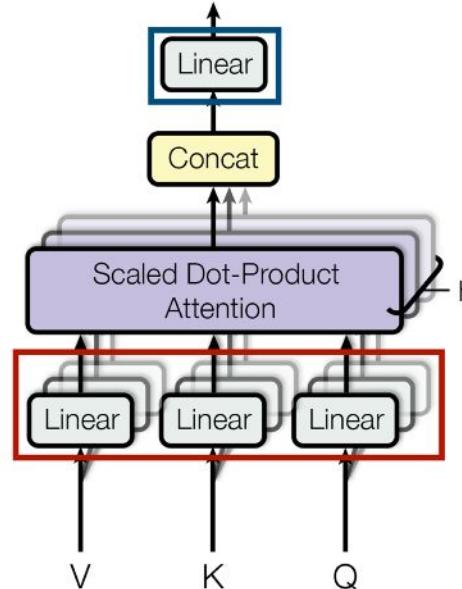
- Each encoder block has two sub-layers:
 - The first is a **multi-head self-attention** mechanism.
 - The second is a position-wise **fully connected feed-forward** network.
- Each decoder block has an additional third sub-layer:
 - The third is a multi-head attention over the output of the encoder stack.
- A residual connection is added around each of the two sub-layers, followed by layer normalization:
$$\text{LayerNorm}(x + \text{Sublayer}(x))$$
- The decoder generates the output sequence of symbols one element at a time in an **auto-regressive** manner.

Recap: Transformers - Multi-head Self-Attention (MHSA)

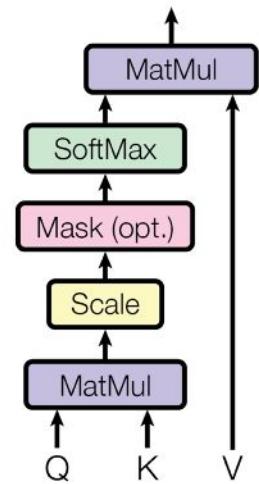
- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$



Scaled Dot-Product Attention



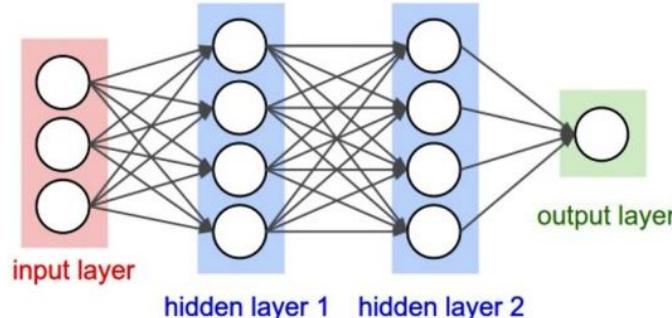
$$\begin{aligned}\text{Attention}(Q, K, V) &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\end{aligned}$$

Recap: Transformers - Feed-Forward Network (FFN)

- Each block in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position **separately** and **identically**.
- This consists of two linear transformations with a ReLU activation in between.

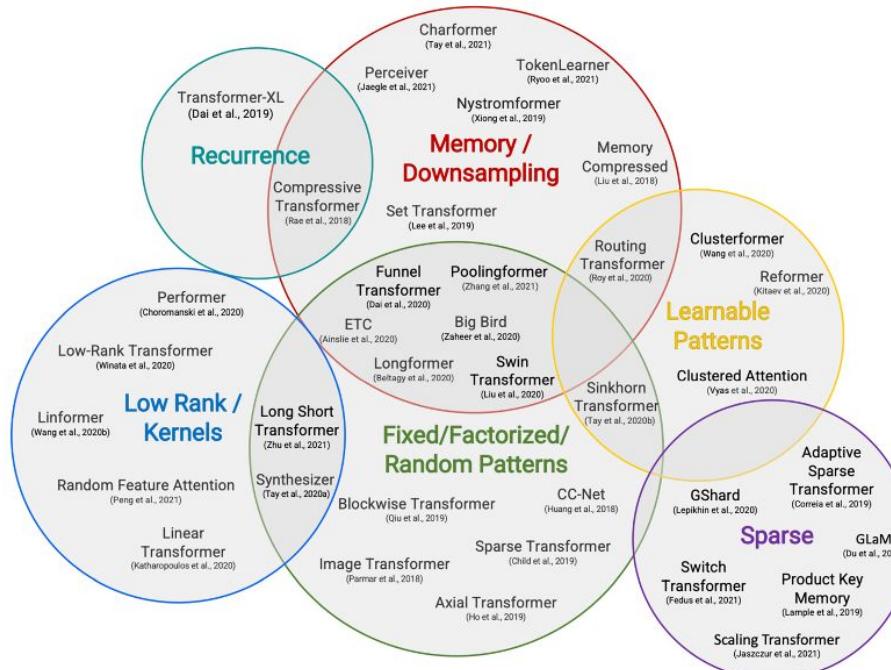
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- The middle hidden size is usually larger than and input and output size (**inverted bottleneck**).



Model	#L	#H	d _{model}	LR	Batch
125M	12	12	768	6.0e-4	0.5M
350M	24	16	1024	3.0e-4	0.5M
1.3B	24	32	2048	2.0e-4	1M
2.7B	32	32	2560	1.6e-4	1M
6.7B	32	32	4096	1.2e-4	2M
13B	40	40	5120	1.0e-4	4M
30B	48	56	7168	1.0e-4	4M
66B	64	72	9216	0.8e-4	2M
175B	96	96	12288	1.2e-4	2M

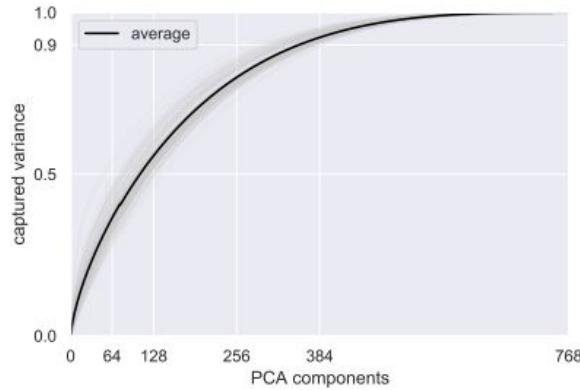
Efficient Transformers



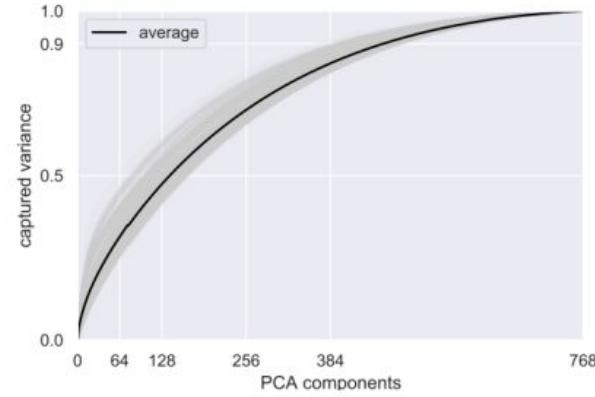
Why could we achieve efficiency within Transformer
but maintaining the performance ?

马儿可以不吃草，也跑得快吗？

Parameter redundancy existed



(a) PCA for each single weight matrix



(b) PCA for a pair of matrices along columns

Figure 1: PCA for existing weight block matrices in BERT-base. We got nearly similar results in Fig. 5 for paired matrices along rows and columns, as shown in App. C.

Decomposability (可分解性)

- A computing module f is **decomposable** if its sub-components $\{g_1, g_2, \dots, g_H\}$ could be independently calculated without interactions: $f(x) = \delta(g_1(x), g_2(x), \dots, g_H(x))$. Usually, δ is a simple operation that has negligible computing cost compared to g

Decomposability might lead to **redundancy**, as it has backup modules.

Self-attention is decomposable

- As there exist multiple heads

$$\text{Att}_h(\mathbf{X}) = \text{Softmax}\left(\frac{1}{\sqrt{d}} \mathbf{X} \mathbf{W}_h^Q \mathbf{W}_h^{K^T} \mathbf{X}^T\right) \mathbf{X} \mathbf{W}_h^V \mathbf{W}_h^{O^T}$$

Feed-forward network is also decomposable

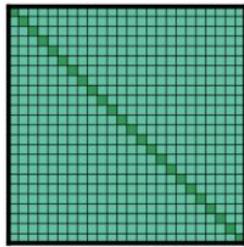
- It performs like a multi-head mechanism

$$\text{FFN}(\mathbf{X}) = \sum_{h=1}^{4D} \text{GeLU}(\mathbf{X} \mathbf{W}_{\cdot,h}^{In} + b_h^{In}) \mathbf{W}_{h,\cdot}^{Out} + b^{Out}$$

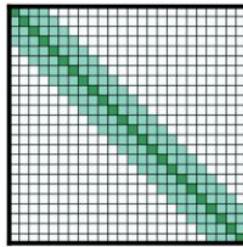
Efficient Transformers - efficient Attention

Sparse Attention - LongFormer

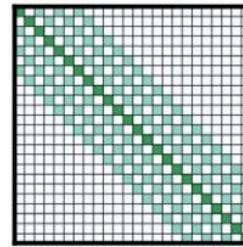
Local Attention + Global Attention



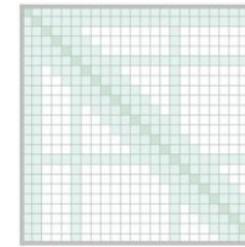
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window

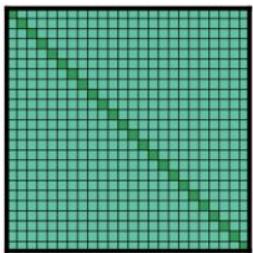


(d) Global+sliding window

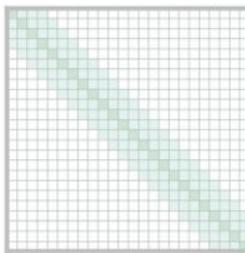
- Attention with **sliding window** (analogous to CNNs):
 - A fixed-size window attention surrounding each token.
 - The complexity is reduced from $O(N^2)$ to **$O(N \times W)$** , where W is the window size.
- Attention with **dilated sliding window** (analogous to dilated CNNs):
 - Dilate the sliding window with gaps of size dilation D .
 - The receptive field is enlarged from W to **$W \times D$** , with the same complexity.

Sparse Attention - LongFormer

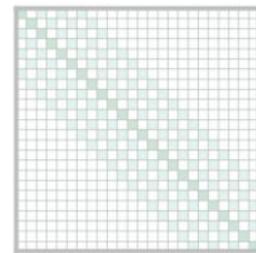
Local Attention + Global Attention



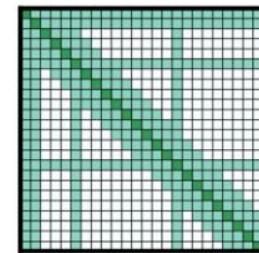
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window

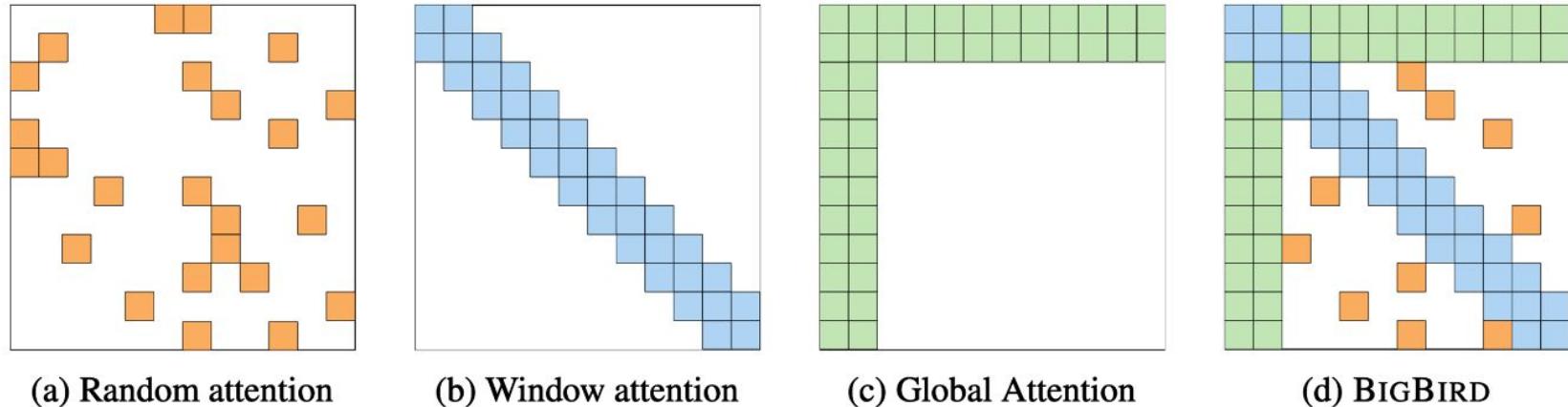


(d) Global+sliding window

- **Global attention** added on a few **pre-selected** input locations:
 - Classification: The special token ([CLS]), aggregating the whole sequence.
 - QA: All question tokens, allowing the model to compare the question with the document.
- Global attention is applied **symmetrically**:
 - A token with a global attention attends to all tokens across the sequence, and all tokens in the sequence attend to it.

Sparse Attention - Big Bird

Random Attention + Local Attention + Global Attention

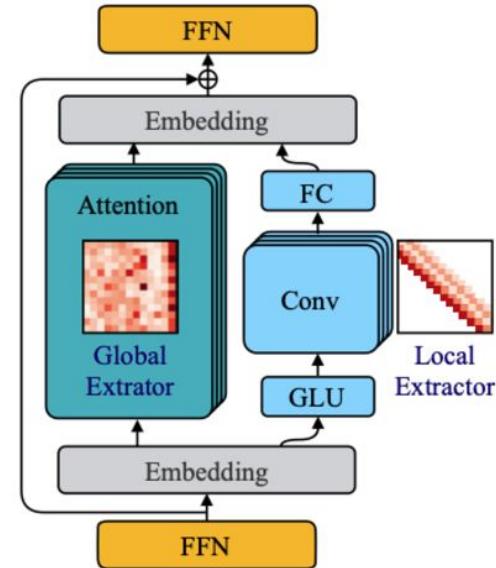
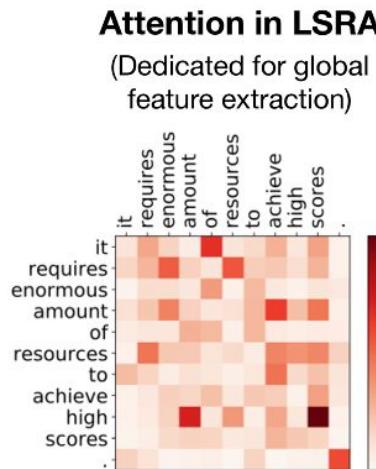
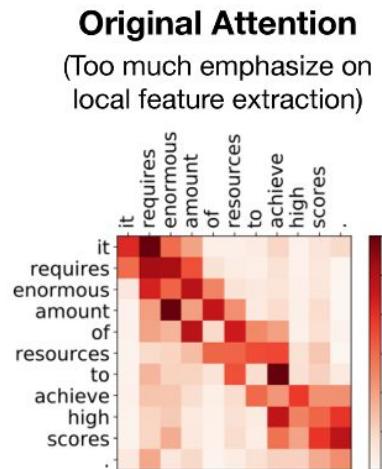


- Random sparse attention:
 - Each query attends over r random number of keys: i.e. $A(i, \cdot) = 1$ for r randomly chosen keys.
 - Information can flow fast between any pair of nodes (rapid mixing time for random walks).

Sparse Attention - Lite Transformer

Local Convolution + Global Attention

- Long-Short Range Attention (LSRA):
 - **Convolution:** Efficiently extract the **local** features.
 - **Attention:** Tailored for **global** feature extraction.

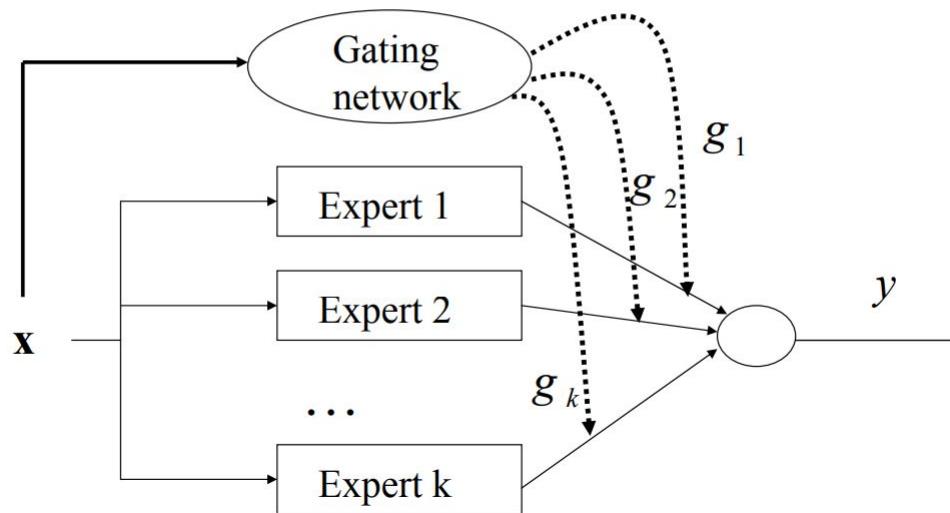


Mixture of Expert (MoE) - efficient FFNs

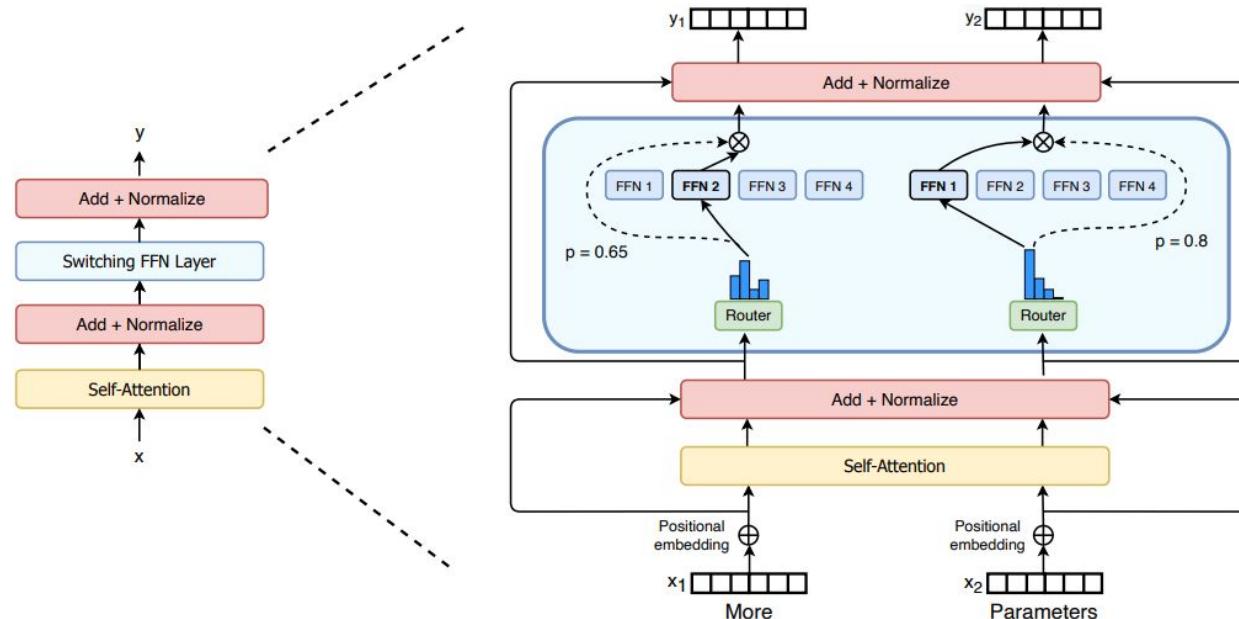
Mixture of Expert

- **Gating network** : decides what expert to use

g_1, g_2, \dots, g_k - gating functions



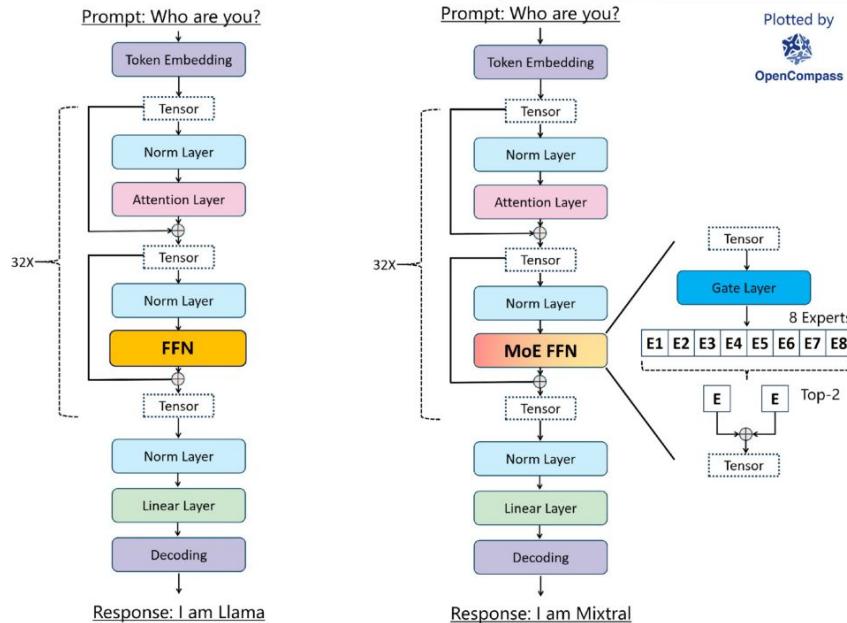
MOE in Transformer



William Fedus, Barret Zoph, Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. <https://arxiv.org/pdf/2101.03961.pdf>

Mixture of Expert

- Model Architectures

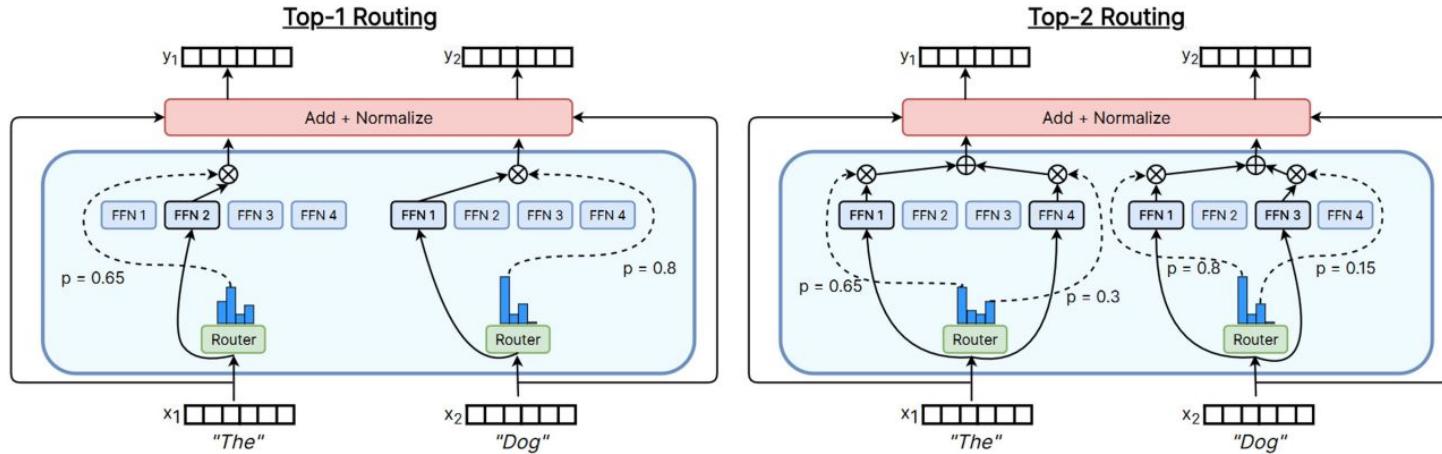


Key points:

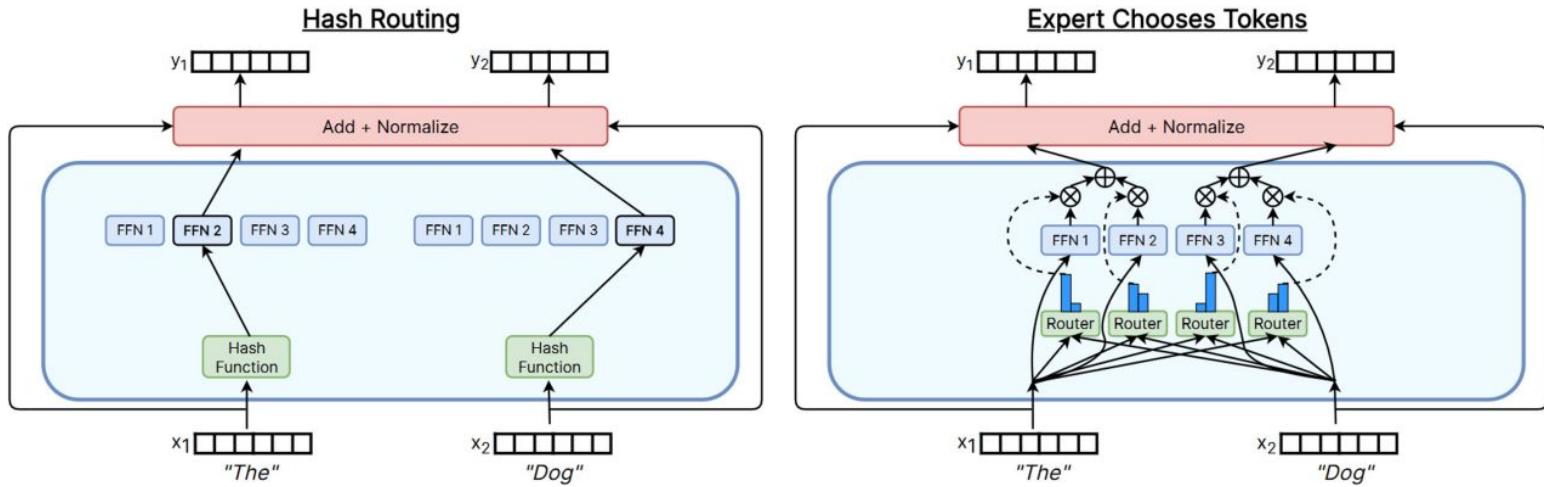
Activate different experts parameters for each input token.

Sparse activation. Not all parameters are activated.

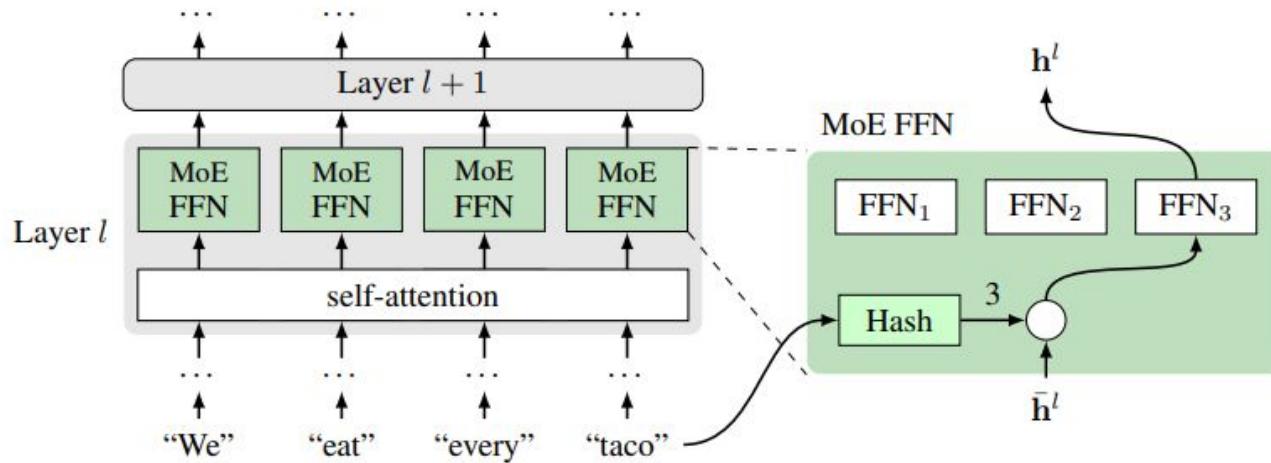
Routing Algorithms



Routing Algorithms



An example of Hash



Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston. Hash Layers For Large Sparse Models.
<https://arxiv.org/pdf/2106.04426.pdf>

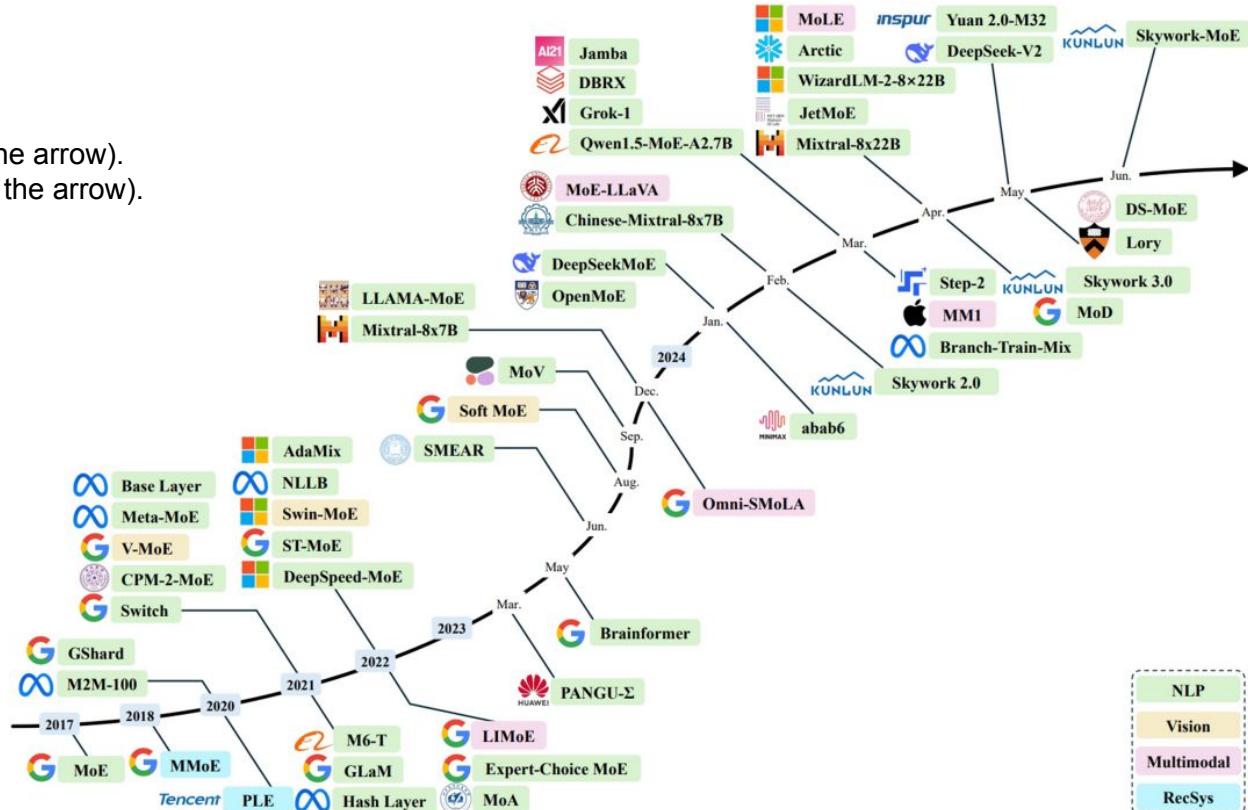
Random hash also works

Table 3: **Different Hash Layering Methods** on pushshift.io Reddit.

Model	Hashing Type	Valid PPL	Test PPL
Baseline Transformer	-	24.90	24.96
Hash Layer 1x64	Balanced assignment	23.16	23.23
Hash Layer 1x64	Fixed random assignment	23.22	23.27
Hash Layer 1x64	Token clustering (using Baseline Transformer)	23.90	23.99
Hash Layer 1x64	Dispersed Hash (within token clusters)	23.17	23.22
Hash Layer 1x64	Hash on position	25.07	25.14
Hash Layer 1x64	Bigrams	24.19	24.28
Hash Layer 1x64	Previous token	24.16	24.22
Hash Layer 1x64	Future token predictions (using Transformer Baseline)	25.02	25.09
Hash Layer 1x64	Future token (Oracle)	1.97	1.97
Hash Layer 5x16	Same hash per layer (balance assignment)	23.74	23.81
Hash Layer 5x16	Different Hash per layer	23.21	23.27

MoE Models

- Open-source (above the arrow).
 - Private models (under the arrow).



MoE Design

What should we care when designing a MoE?

Network types	FFN, Attention
Fine-grained experts	64 experts/128 experts/...
Shared experts	Isolated experts
Activation Function	ReLU/GEGLU/SwiGLU
MoE frequency	Every two layer/Each layer/...
Training auxiliary loss	Auxiliary loss/Z-loss/...

Fine-grained and Shared Experts

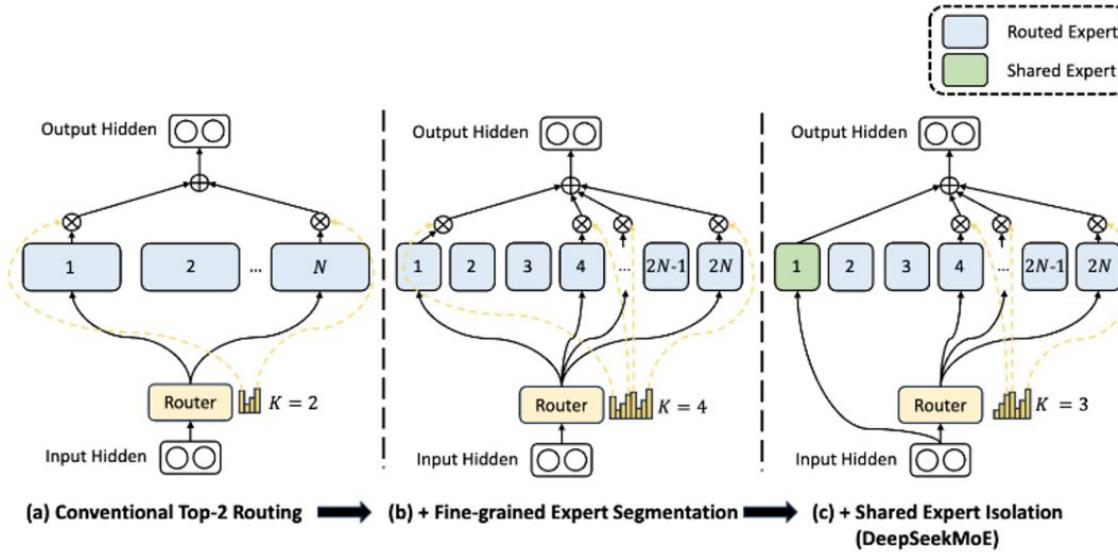


Figure 2 | Illustration of DeepSeekMoE. Subfigure (a) showcases an MoE layer with the conventional top-2 routing strategy. Subfigure (b) illustrates the fine-grained expert segmentation strategy. Subsequently, subfigure (c) demonstrates the integration of the shared expert isolation strategy, constituting the complete DeepSeekMoE architecture. It is noteworthy that across these three architectures, the number of expert parameters and computational costs remain constant.

MoE Experts Design

Reference	Models	Expert Count (Activ.Total)	d_{model}	d_{ffn}	d_{expert}	#L	#H	d_{head}	Placement Frequency	Activation Function	Share Expert Count
GShard [86] (2020)	600B	2/2048	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
	200B	2/2048	1024	8192	d_{ffn}	12	16	128	1/2	ReLU	0
	150B	2/512	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
	37B	2/128	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
Switch [49] (2021)	7B	1/128	768	2048	d_{ffn}	12	12	64	1/2	GEGLU	0
	26B	1/128	1024	2816	d_{ffn}	24	16	64	1/2	GEGLU	0
	395B	1/64	4096	10240	d_{ffn}	24	64	64	1/2	GEGLU	0
	1571B	1/2048	2080	6144	d_{ffn}	15	32	64	1	ReLU	0
GLaM [44] (2021)	0.1B/1.9B	2/64	768	3072	d_{ffn}	12	12	64	1/2	GEGLU	0
	1.7B/27B	2/64	2048	8192	d_{ffn}	24	16	128	1/2	GEGLU	0
	8B/143B	2/64	4096	16384	d_{ffn}	32	32	128	1/2	GEGLU	0
	64B/1.2T	2/64	8192	32768	d_{ffn}	64	128	128	1/2	GEGLU	0
DeepSpeed-MoE [121] (2022)	350M/13B	2/128	1024	$4d_{model}$	d_{ffn}	24	16	64	1/2	GeLU	0
	1.3B/52B	2/128	2048	$4d_{model}$	d_{ffn}	24	16	128	1/2	GeLU	0
	PR-350M/4B	2/32/2/64	1024	$4d_{model}$	d_{ffn}	24	16	64	1/2, 10L-32E, 2L-64E	GeLU	1
	PR-1.3B/31B	2/64/2/128	2048	$4d_{model}$	d_{ffn}	24	16	128	1/2, 10L-64E, 2L-128E	GeLU	1
ST-MoE [197] (2022)	0.8B/4.1B	2/32	1024	2816	d_{ffn}	27	16	64	1/4, add extra FFN	GEGLU	0
	32B/269B	2/64	5120	20480	d_{ffn}	27	64	128	1/4, add extra FFN	GEGLU	0
Mixtral [74] (2023)	13B/47B	2/8	4096	14336	d_{ffn}	32	32	128	1	SwiGLU	0
	39B/141B	2/8	6144	16384	d_{ffn}	56	48	128	1	SwiGLU	0
LLAMA-MoE [149] (2023)	3.0B/6.7B	2/16	4096	11008	688	32	32	128	1	SwiGLU	0
	3.5B/6.7B	4/16	4096	11008	688	32	32	128	1	SwiGLU	0
	3.5B/6.7B	2/8	4096	11008	1376	32	32	128	1	SwiGLU	0
DeepSeekMoE [30] (2024)	0.24B/1.89B	8/64	1280	-	$\frac{1}{2}d_{ffn}$	9	10	128	1	SwiGLU	1
	2.8B/16.4B	8/66	2048	10944	1408	28	16	128	1, except 1st layer	SwiGLU	2
	22B/145B	16/132	4096	-	$\frac{1}{2}d_{ffn}$	62	32	128	1, except 1st layer	SwiGLU	4
OpenMoE [172] (2024)	339M/650M	2/16	768	3072	d_{ffn}	12	12	64	1/4	SwiGLU	1
	2.6B/8.7B	2/32	2048	8192	d_{ffn}	24	24	128	1/6	SwiGLU	1
	6.8B/34B	2/32	3072	12288	d_{ffn}	32	24	128	1/4	SwiGLU	1
Qwen1.5-MoE [151] (2024)	2.7B/14.3B	8/64	2048	5632	1408	24	16	128	1	SwiGLU	4
DBRX [34] (2024)	36B/132B	4/16	6144	10752	d_{ffn}	40	48	128	1	SwiGLU	0
Jamba [94] (2024)	12B/52B	2/16	4096	14336	d_{ffn}	32	32	128	1/2, 1:7 Attention:Mamba	SwiGLU	0
Skywork-MoE [154] (2024)	22B/146B	2/16	4608	12288	d_{ffn}	52	36	128	1	SwiGLU	0
Yuan 2.0-M32 [166] (2024)	3.7B/40B	2/32	2048	8192	d_{ffn}	24	16	256	1	SwiGLU	0

- Most recent models place MoE each layer.
- Some of recent models apply Shared experts.

Auxiliary Loss

Training with different auxiliary loss:

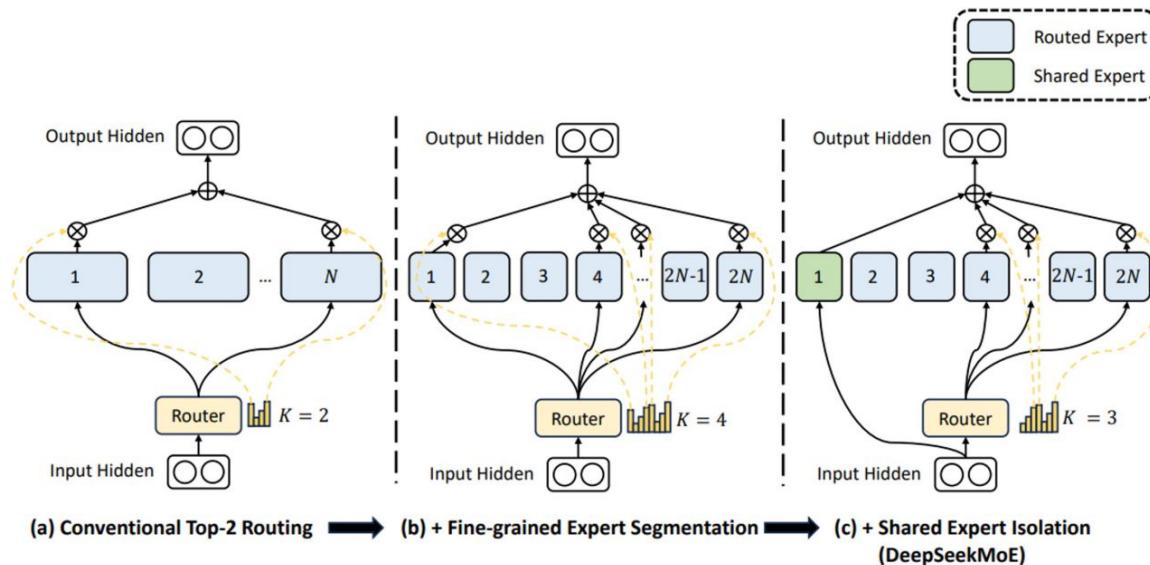
Reference	Auxiliary Loss	Coefficient
Shazeer et al. [135], <u>V-MoE</u> [128] GShard [86], Switch-T [49], GLaM[44], Mixtral-8x7B[74], DBRX[34], Jamba[94], <u>DeepSeekMoE</u> [30], <u>DeepSeek-V2</u> [36], <u>Skywork-MoE</u> [154]	$L_{importance} + L_{load}$ L_{aux}	$w_{importance} = 0.1, w_{load} = 0.1$ $w_{aux} = 0.01$
ST-MoE [197], OpenMoE[172], MoA[182], JetMoE [139]	$L_{aux} + L_z$	$w_{aux} = 0.01, w_z = 0.001$
Mod-Squad [21], <u>Moduleformer</u> [140], <u>DS-MoE</u> [117]	L_{MI}	$w_{MI} = 0.001$

- Importance loss: encourages all experts to have equal importance
- Load loss: ensure balanced loads
- Auxiliary loss: mitigating load balance losses
- Z-loss: improving training stability by penalizing large logits
- MI-loss: mutual information (MI) between experts and tasks to build task-expert alignment

Training MoE - Deepseek (example)

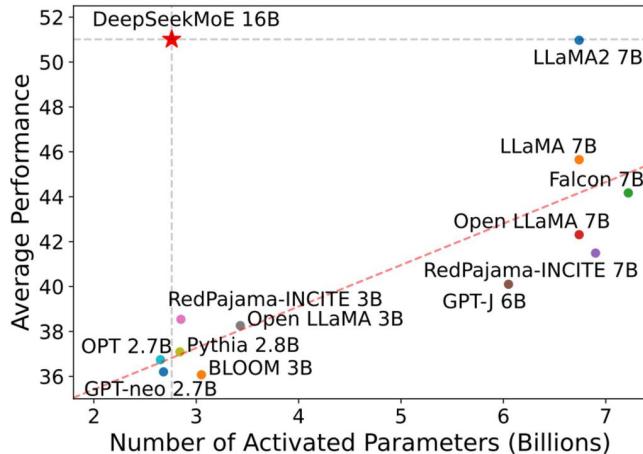
Deepseek-MoE 16B, total 16.4B parameters, 2.8B activate parameters.

Each MoE layer consists of 2 shared experts and 64 routed experts (select 6 experts).



- Most recent models place MoE each layer.
- Some of recent models apply Shared experts.

Training MoE - Deepseek (example)

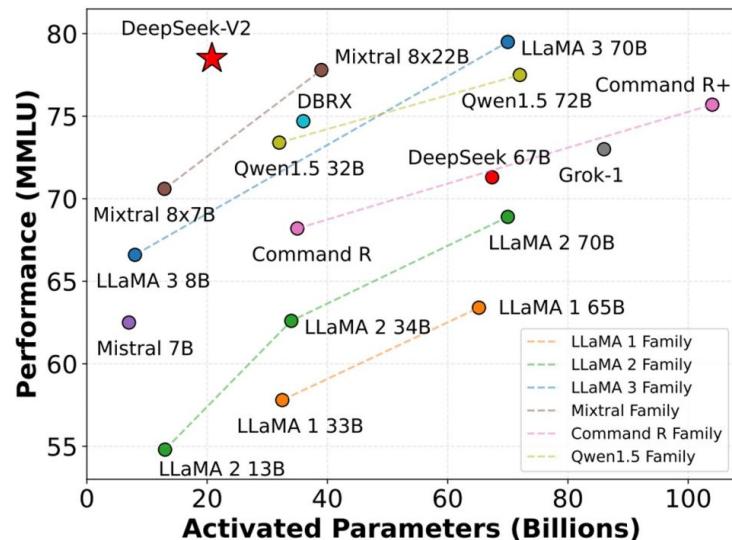


Metric	# Shot	DeepSeek 7B (Dense)	DeepSeekMoE 16B
# Total Params	N/A	6.9B	16.4B
# Activated Params	N/A	6.9B	2.8B
FLOPs per 4K Tokens	N/A	183.5T	74.4T
# Training Tokens	N/A	2T	2T
Pile (BPB)	N/A	0.75	0.74
HellaSwag (Acc.)	0-shot	75.4	77.1
PIQA (Acc.)	0-shot	79.2	80.2
ARC-easy (Acc.)	0-shot	67.9	68.1
ARC-challenge (Acc.)	0-shot	48.1	49.8
RACE-middle (Acc.)	5-shot	63.2	61.9
RACE-high (Acc.)	5-shot	46.5	46.4
DROP (EM)	1-shot	34.9	32.9
GSM8K (EM)	8-shot	17.4	18.8
MATH (EM)	4-shot	3.3	4.3
HumanEval (Pass@1)	0-shot	26.2	26.8
MBPP (Pass@1)	3-shot	39.0	39.2
TriviaQA (EM)	5-shot	59.7	64.8
NaturalQuestions (EM)	5-shot	22.2	25.5
MMLU (Acc.)	5-shot	48.2	45.0
WinoGrande (Acc.)	0-shot	70.5	70.2
CLUEWSC (EM)	5-shot	73.1	72.1
CEval (Acc.)	5-shot	45.0	40.6
CMMLU (Acc.)	5-shot	47.2	42.5
CHID (Acc.)	0-shot	89.3	89.4

Training MoE - Deepseek (example)

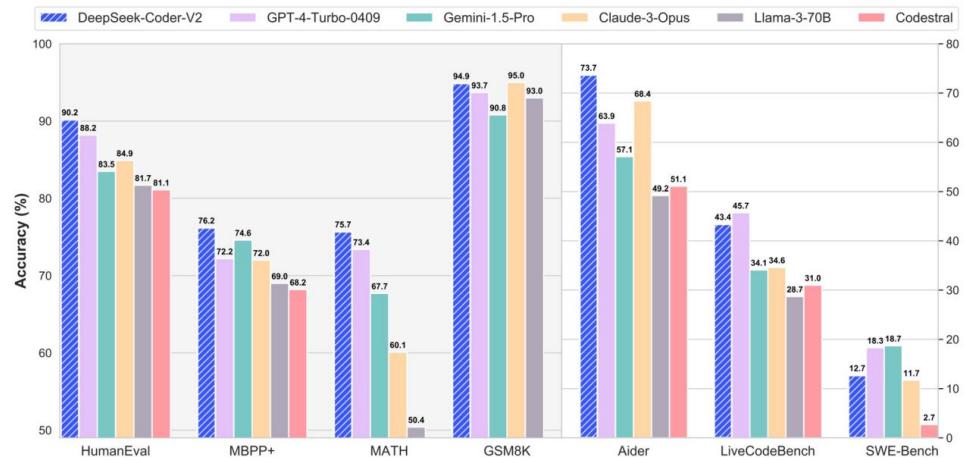
Deepseek-V2

236B total parameters, 21B are activated.
2 shared experts and 160 routed experts (6 select).



Deepseek-Coder-V2

Continue pretraining from an intermediate checkpoint of Deepseek-V2 (4.2T) and further train 6T. Total 10.2T tokens.



Sparse Upcycling - Qwen-MoE

Qwen1.5-MoE-A2.7B (Mar, 2024)

Upcycled from Qwen-1.8B, 14.3B parameters in total and 2.7B activated parameters.

- Fine-grained experts (total 64 experts)
- use shared (4 experts) and routing experts (60 experts, choose 4)

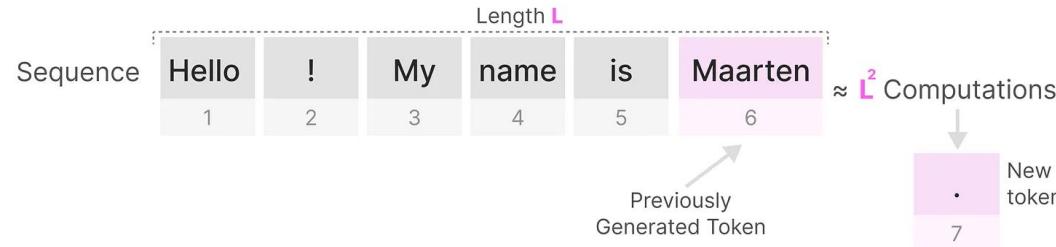
Model	MMLU	GSM8K	HumanEval	Multilingual	MT-Bench
Mistral-7B	64.1	47.5	27.4	40.0	7.60
Gemma-7B	64.6	50.9	32.3	-	-
Qwen1.5-7B	61.0	62.5	36.0	45.2	7.60
DeepSeekMoE 16B	45.0	18.8	26.8	-	6.93
Qwen1.5-MoE-A2.7B	62.5	61.5	34.2	40.8	7.17

A remarkable reduction of 75% in training

Mamba - Efficient Architecture

The Curse with Inference in Transformer!

When generating the next token, we need to re-calculate the attention for the entire sequence, even if we already generated some tokens.

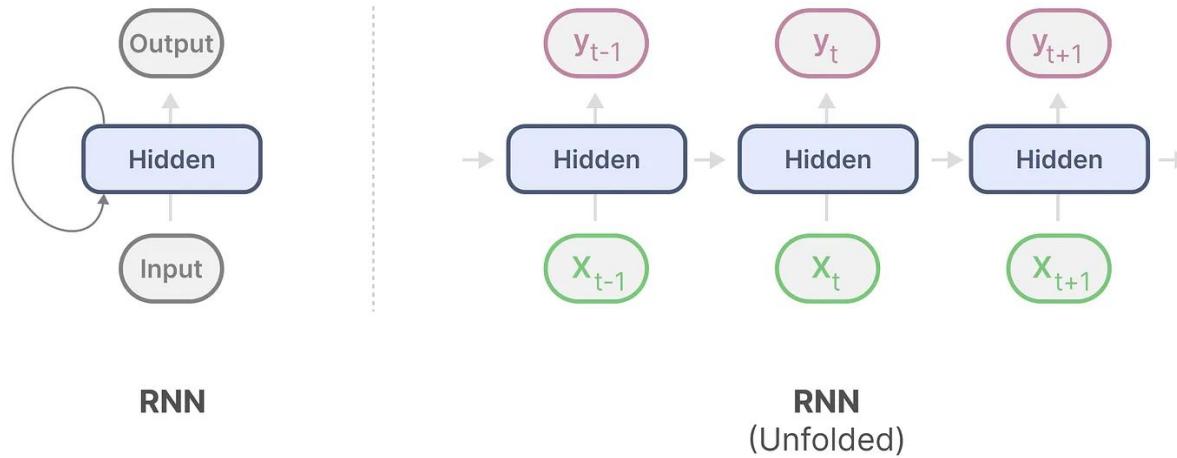


Generating tokens for a sequence of length L needs roughly L^2 computations which can be costly if the sequence length increases.



This need to recalculate **the entire sequence** is a major bottleneck of the Transformer architecture.

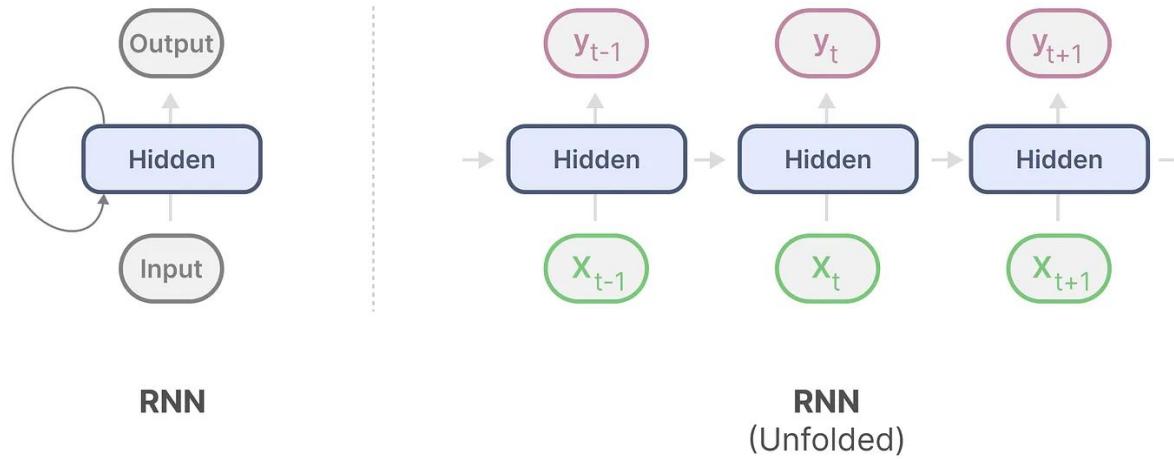
Are RNNs a Solution?



RNNs have a looping mechanism that allows them to pass information from a previous step to the next. We can “unfold” this visualization to make it more explicit.

When generating the output, the RNN only needs to consider the previous hidden state and current input. It prevents recalculating all previous hidden states which is what a Transformer would do.
In other words, it can even have an infinite context length.

Are RNNs a Solution?

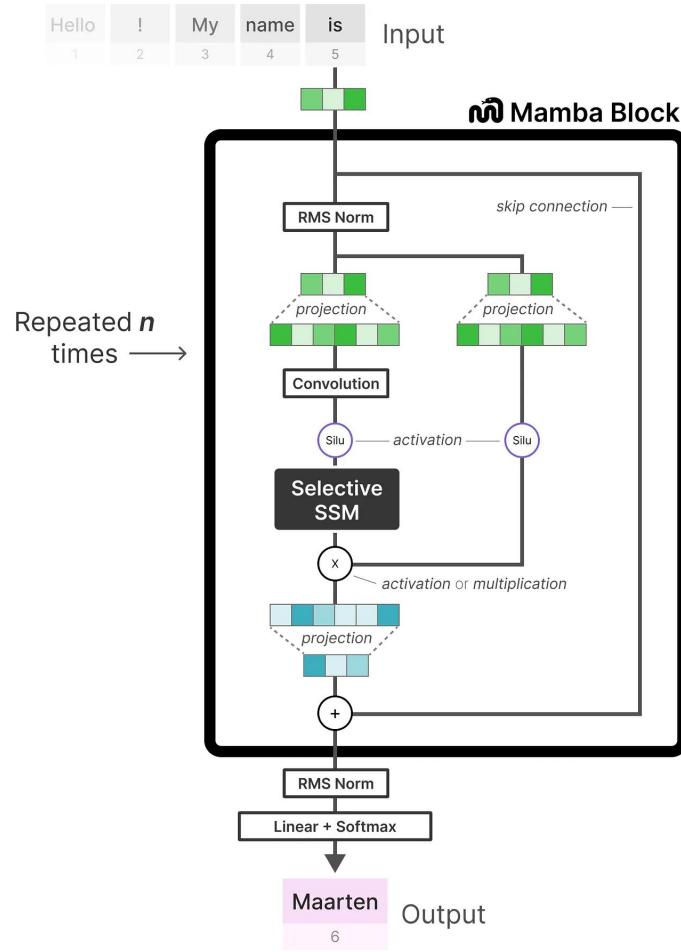


Although RNNs could be fast for both training and inference, they **lacked the accuracy that the Transformer models could offer.**

Mamba

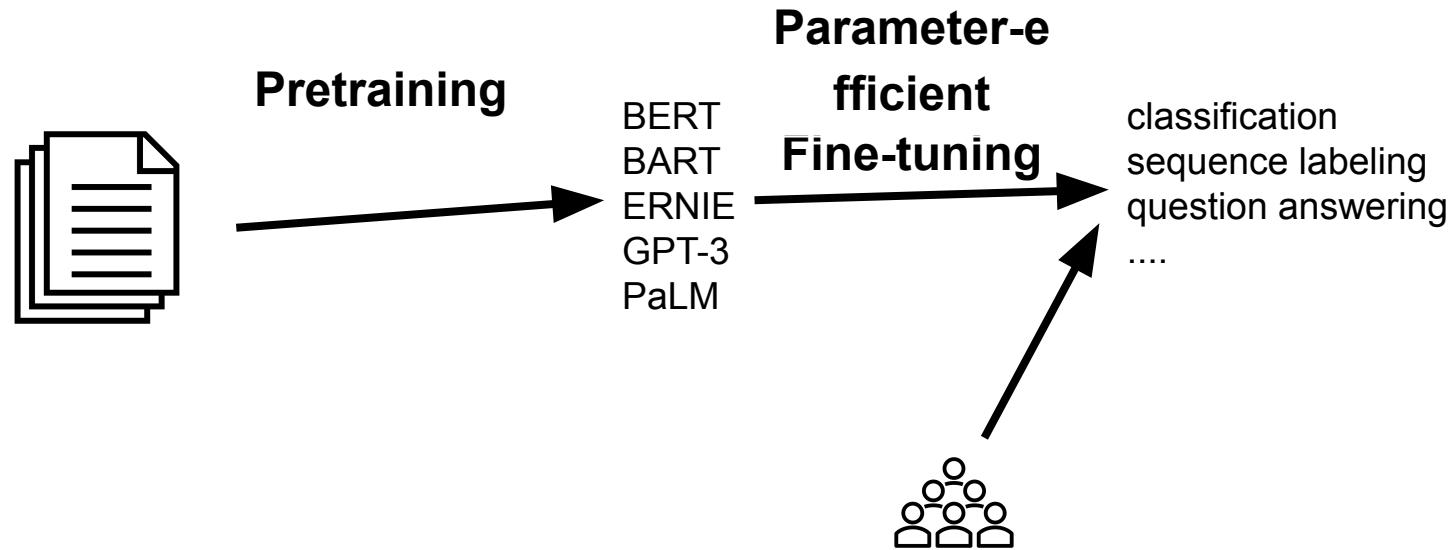
Mamba uses a selective State Space Model. A State Space Model (SSM), like the Transformer and RNN, processes sequences of information, like text but also signals.

Mamba possesses advantages similar to those of RNNs, such as handling sequences of infinite length. More importantly, the authors found that it matches and, in some cases, even surpasses the performance of Transformer models of the same size!



Parameter-efficient Fine-tuning

From Fine-tuning to Parameter-efficient Fine-tuning



Which to implement the efficient finetuning

- **Globally**
 - **LoRA** : Low-rank matrix
- **Locally**
 - **Adapter**: a newly-added layer
 - **Soft Prompt** : “some newly-added fake tokens”
 - **Expert in MOE**
 -

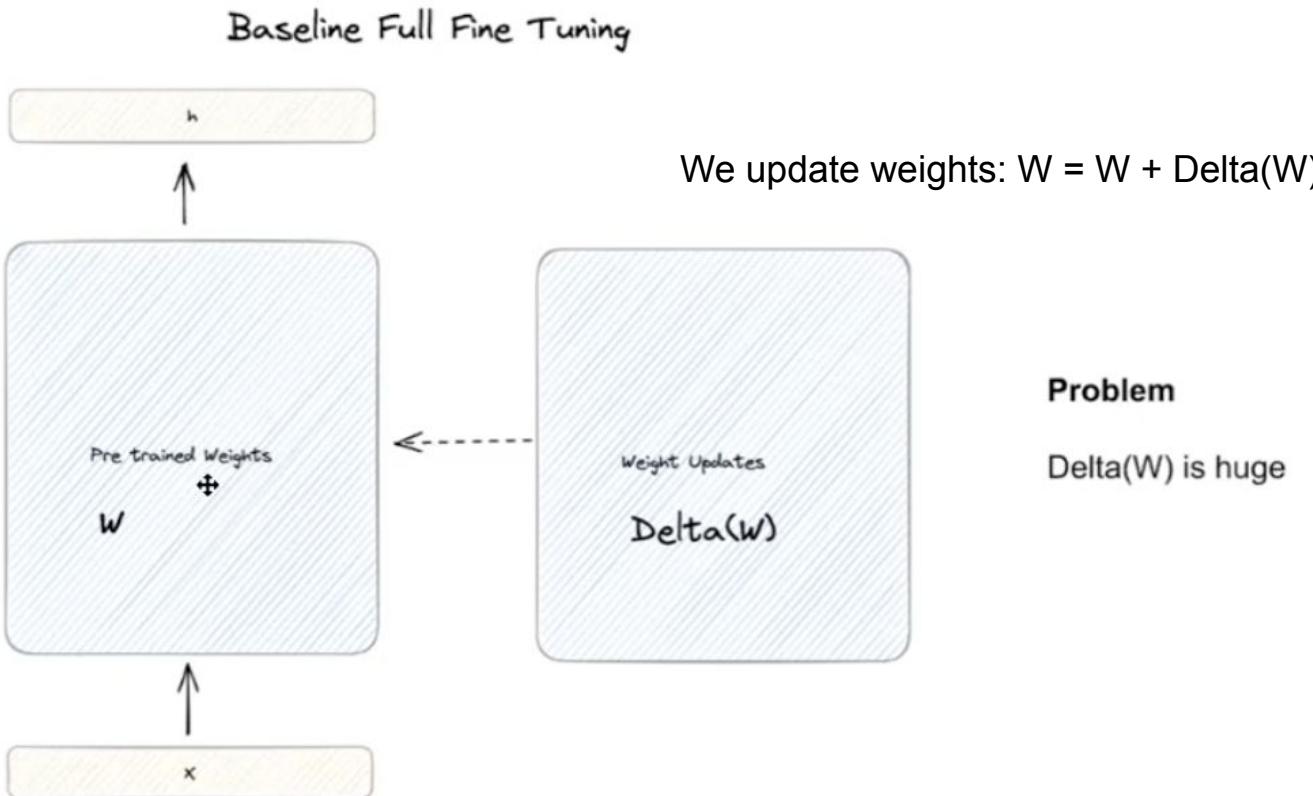
Low-Rank Approximations (LoRA)

- Improve efficiency by leveraging **low-rank** approximations of the self-attention matrix.
- The key idea is to assume **low-rank structure** in the $N \times N$ matrix.

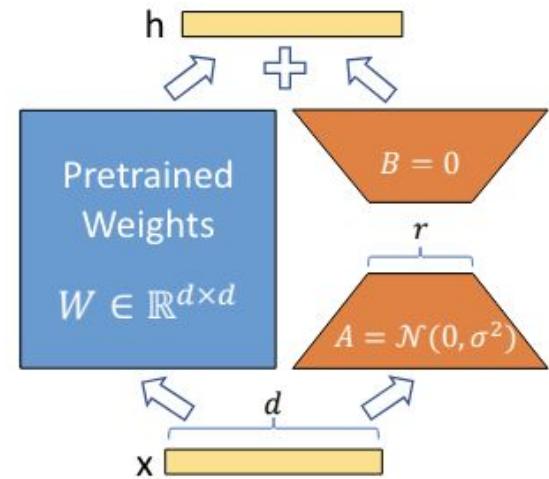
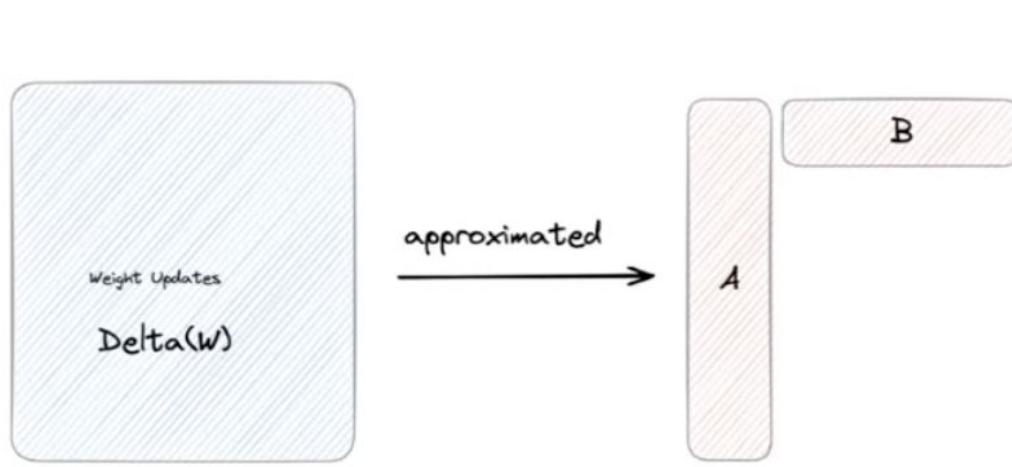
$$M \approx L_k \times R_k^T$$

$m \times n$ $m \times k$ $k \times n$

Baseline Full Fine-tuning



LoRA Tuning



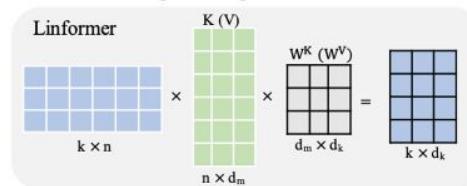
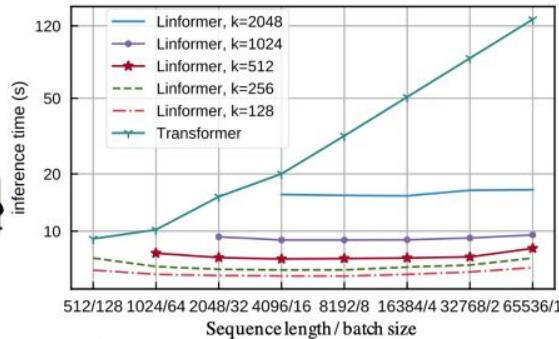
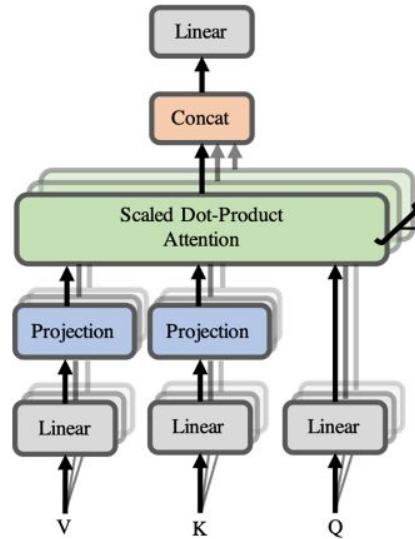
For a pre-trained weights W_0 , we approx $\Delta(w)$ by B and A :

$$h = W_0x + \Delta W x = W_0x + BAx, \text{ where } B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, \text{ and the rank } r \ll \min(d, k)$$

During Training, we only compute gradient w.r.t. $\Delta(w)$

Low-Rank Approximations - Linformer

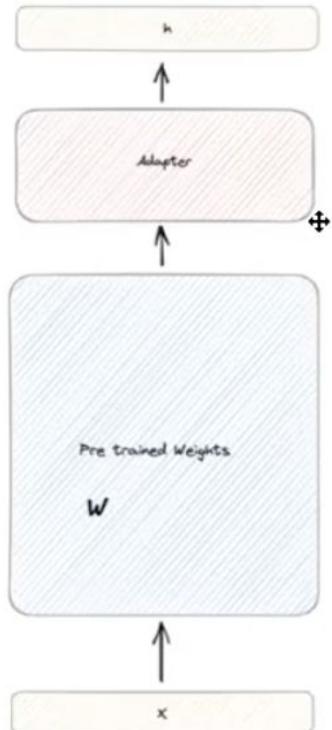
Approximate Self-Attention with Low-Rank Matrix



$$\begin{aligned}\overline{\text{head}_i} &= \text{Attention}(QW_i^Q, \boxed{E_i} KW_i^K, \boxed{F_i} VW_i^V) \\ &= \underbrace{\text{softmax} \left(\frac{QW_i^Q(E_i KW_i^K)^T}{\sqrt{d_k}} \right)}_{\bar{P}:n \times k} \cdot \underbrace{\underbrace{F_i V W_i^V}_{k \times d}}_{},\end{aligned}$$

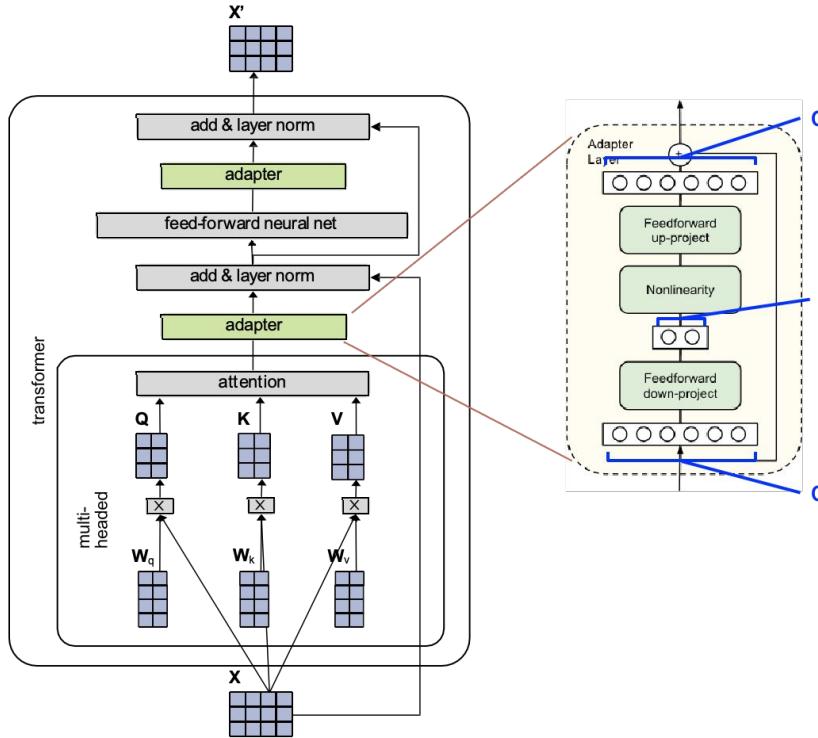
- It projects the **length dimension (not the feature dimension)** of keys and values to a lower-dimensional representation (from N to k).
- Low-rank method reduces the memory complexity problem of self-attention (from $N N$ to $N \times k$).

Other Parameter-efficient Tunings: Adapter Tuning



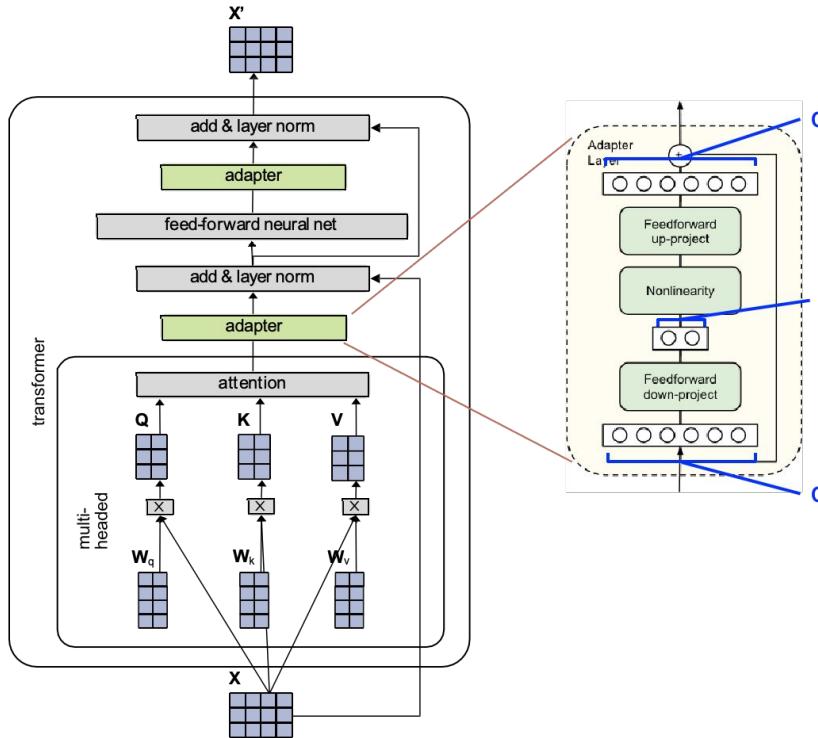
We add an adapter module after pre-trained weights W . And during training, we only compute gradient w.r.t. the adapter

Other Parameter-efficient Tunings: Adapter Tuning



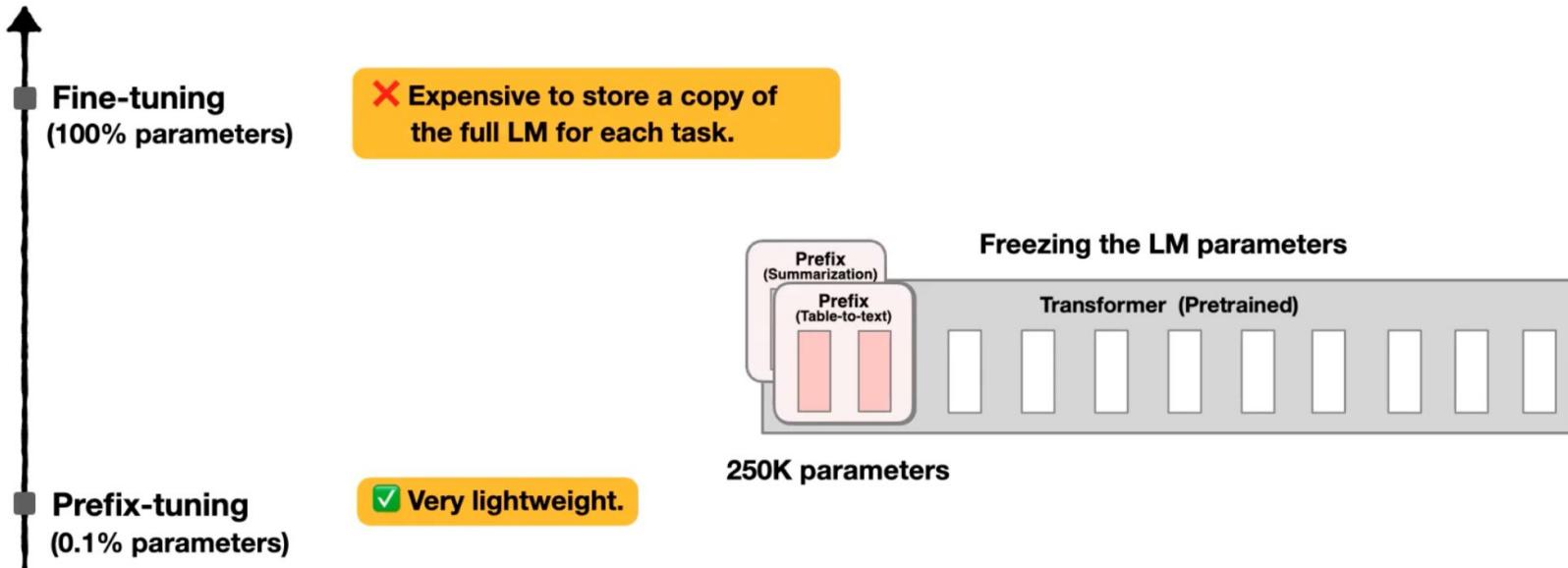
- An adapter layer is simply a feed-forward neural network with one hidden layer, and a residual connection.
- For input dimension, d , the adapter layer also has output dimension d , but bottlenecks to a lower dimension m in the middle.

Other Parameter-efficient Tunings: Adapter Tuning



- In practice, r is chosen s.t. $r \ll d$ and the adapter layers contain **only 0.5% – 8% of the total parameters**.
- When added to a deep neural network (e.g. Transformer) all the other parameters of the pretrained model are kept fixed, and only the adapter layer parameters are fine-tuned.

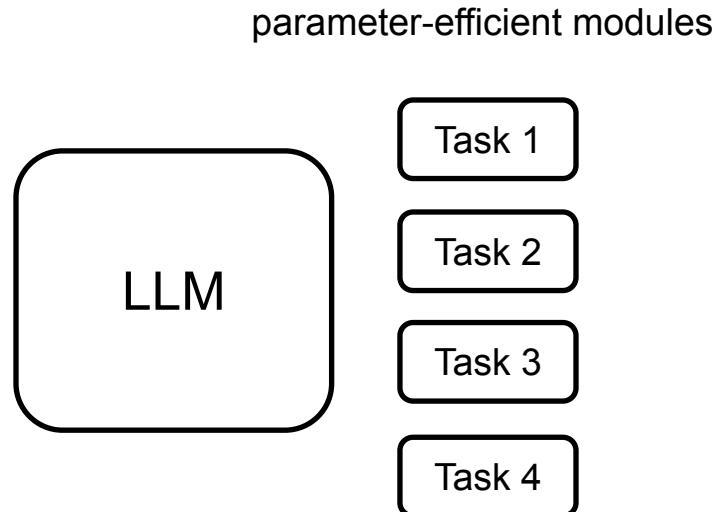
Other Parameter-efficient Tunings: Prefix-Tuning



We prepend a prefix matrix in each transformer layer. And during Training, we only compute gradient w.r.t. Prefix parameters

Parameter-efficient Fine-tuning - **modulization**

Parameter-efficient Fine-tuning

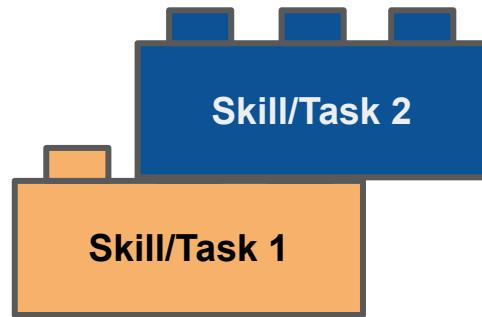


One task, one module

Modularity and Compositionality?



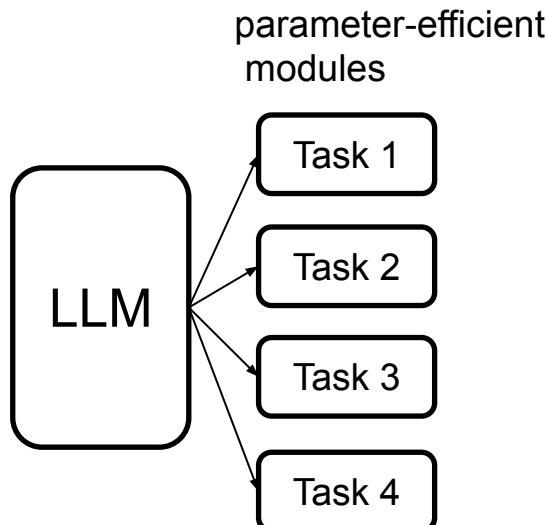
Modularity and Compositionality?



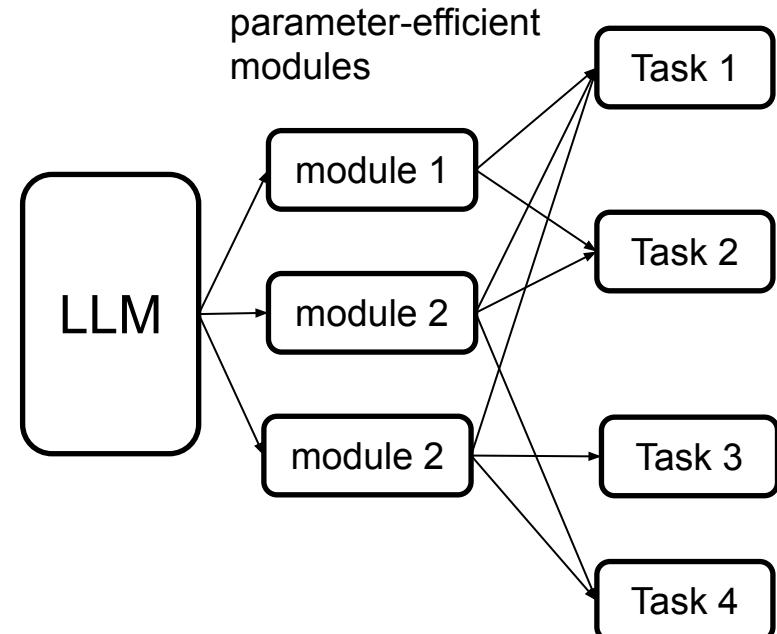
Modularity and Compositionality?



Parameter-efficient Fine-tuning

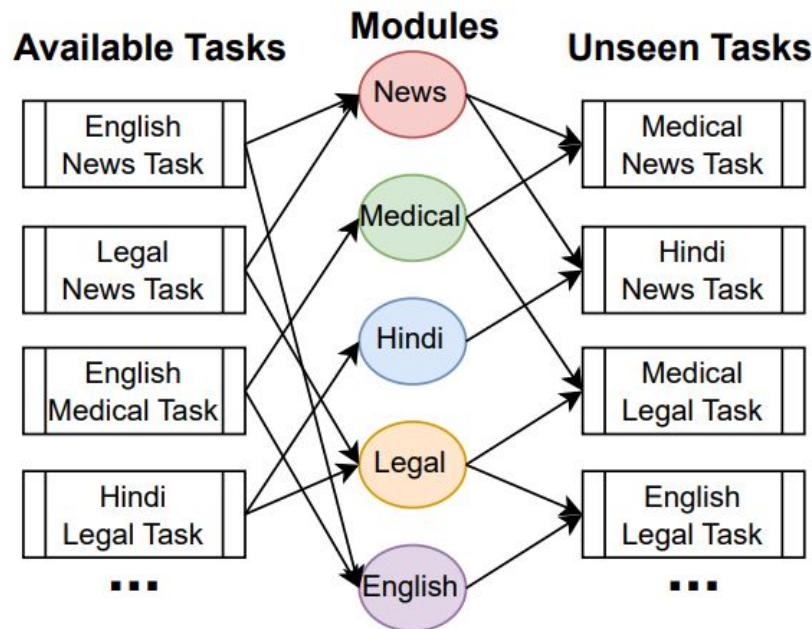


One task, one module



One task, composed modules

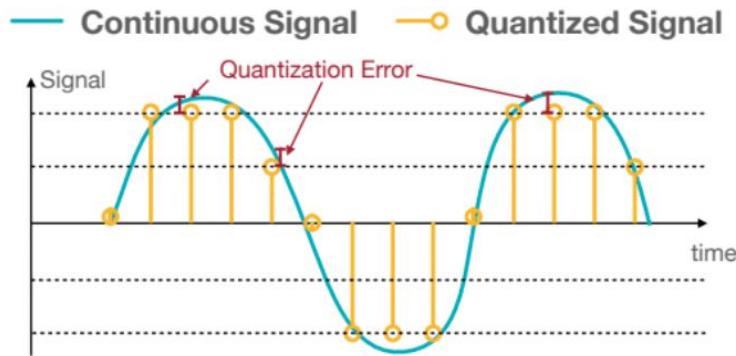
Modular Retrieval



Efficiency Beyond Transformers - Quantization

Quantization - What is Quantization?

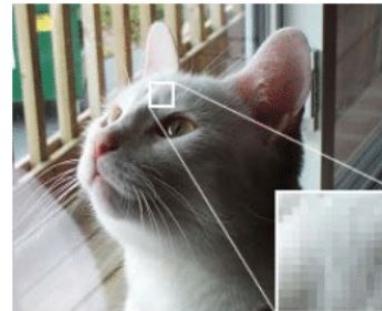
Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



The difference between an input value and its quantized value is referred to as quantization error.

[Quantization \[Wikipedia\]](#)

Original Image



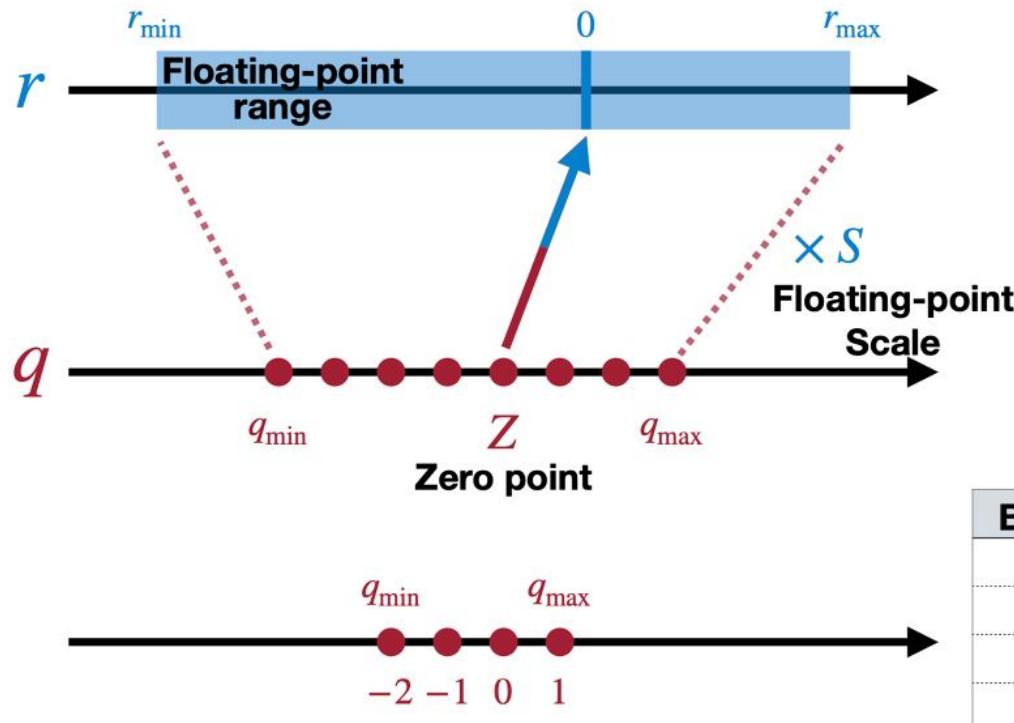
16-Color Image



Images are in the public domain.

"Palettization"

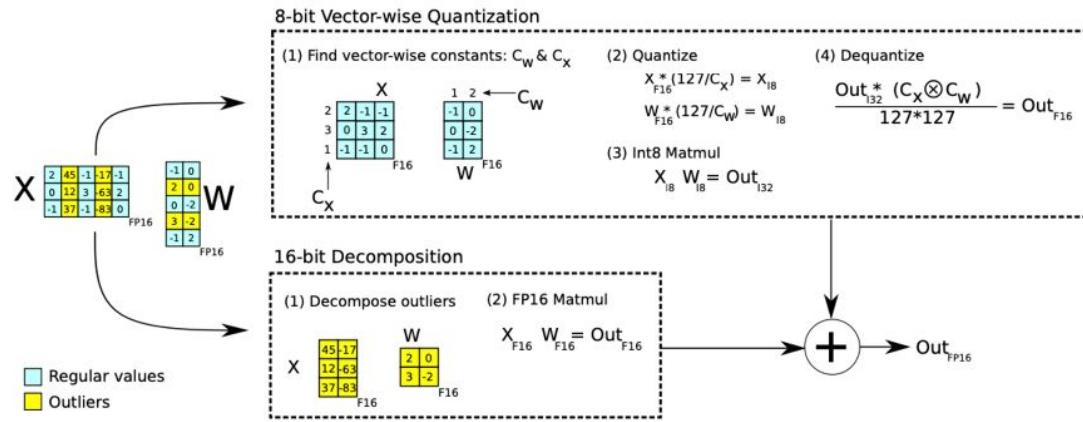
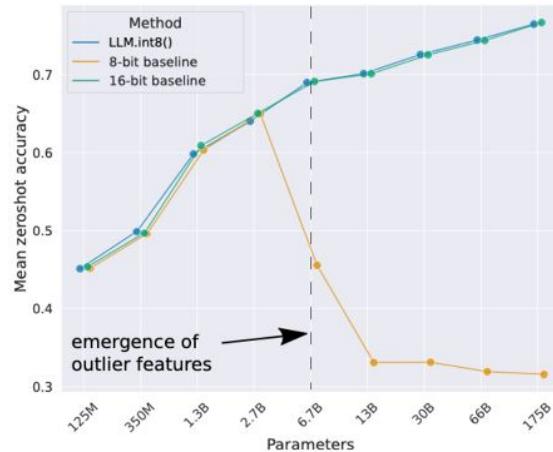
Quantization



Binary	Decimal
01	1
00	0
11	-1
10	-2

Quantization - LLM.int8()

Mixed-Precision Decomposition

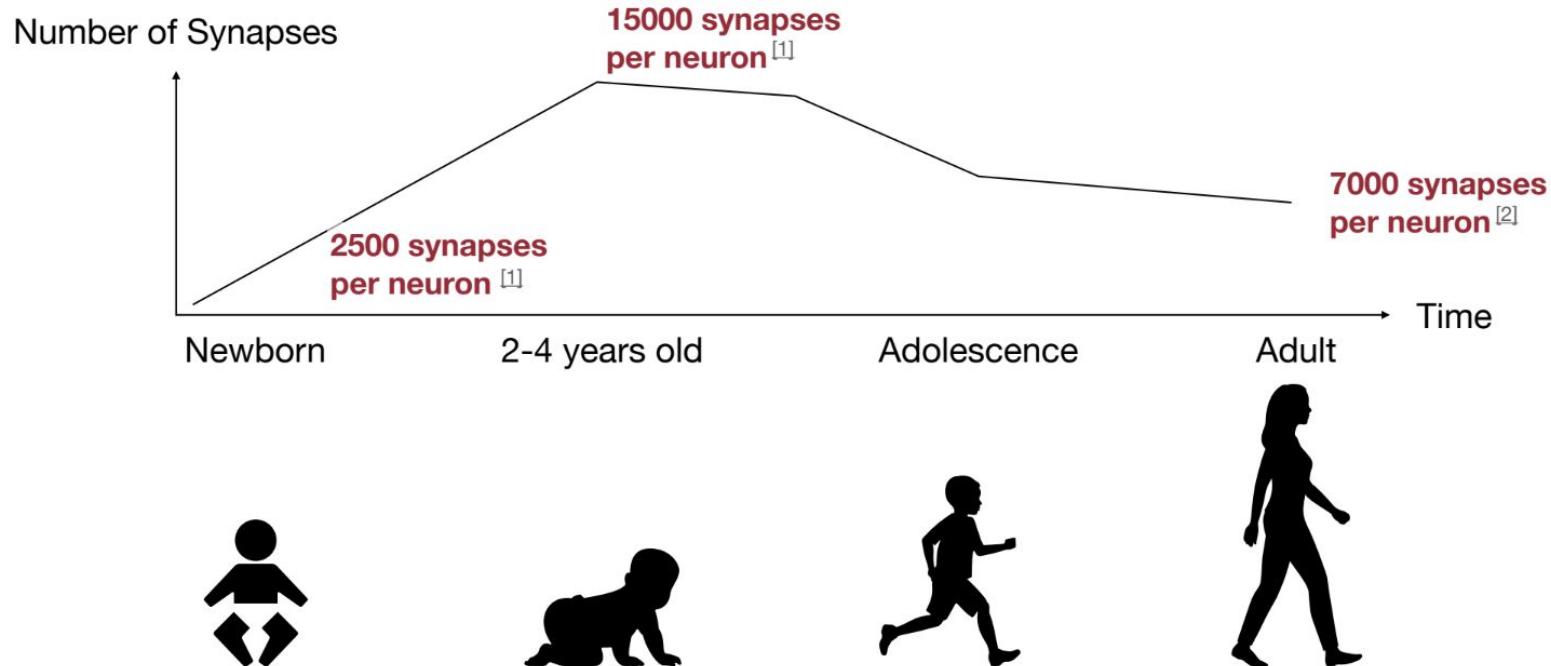


- Motivation:** Transformers have outlier features that have **large values** (especially large models).
 - They occur in particular hidden dimensions, leading to large quantization error.
- Key idea:** Separate outlier features into a **separate FP16 MM**, quantize the other values to Int8.
 - Outlier: At least one feature dimension with a magnitude larger than the threshold (6).
 - Token-wise scale factor (for X) and (output) channel-wise scale factor (for W).

Efficiency Beyond Transformers - Pruning

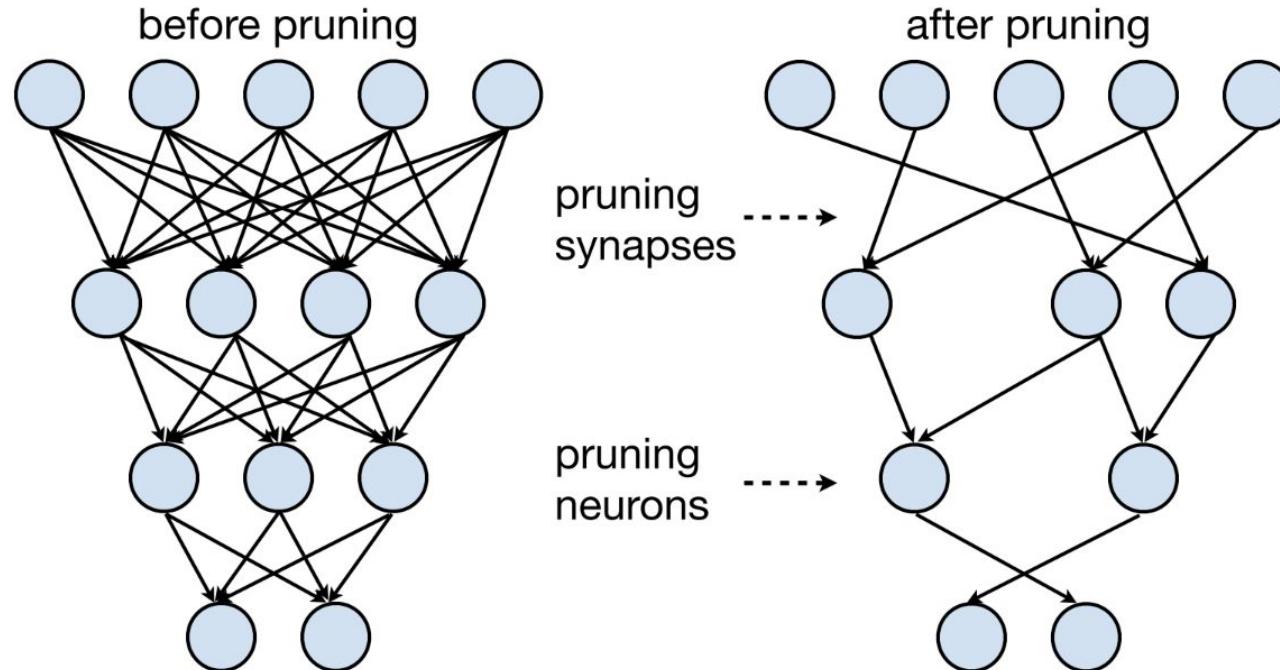
Neural Network Pruning - What is Pruning?

Pruning happens in human brains (synapse 突触 vs. neuron 神经元)



Neural Network Pruning - What is Pruning?

Make neural network smaller by removing synapses and neurons

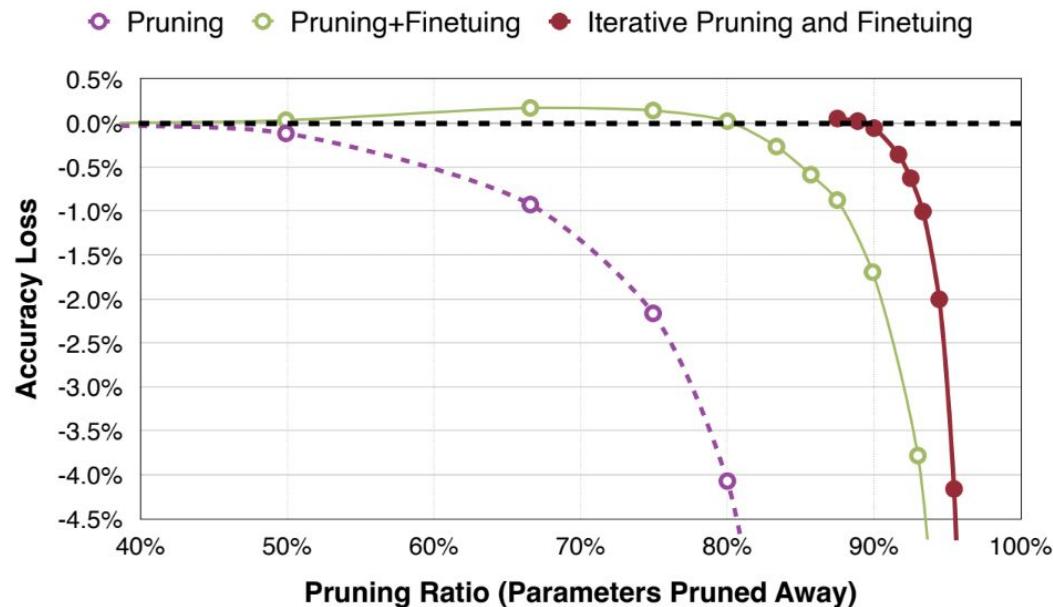
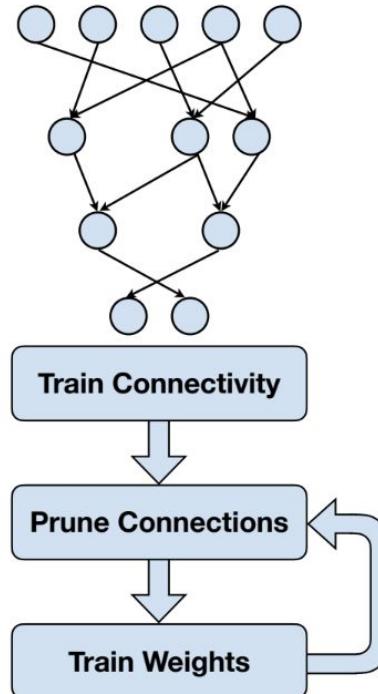


Optimal Brain Damage [LeCun et al., NeurIPS 1989]

Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Neural Network Pruning - What is Pruning?

Make neural network smaller by removing synapses and neurons



Neural Network Pruning - How should we formulate pruning

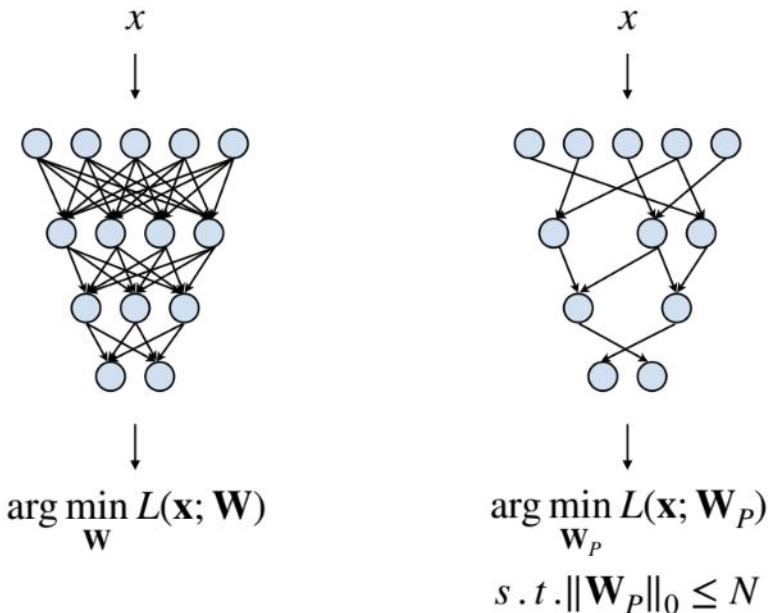
- In general, we could formulate the pruning as follows:

$$\arg \min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

subject to

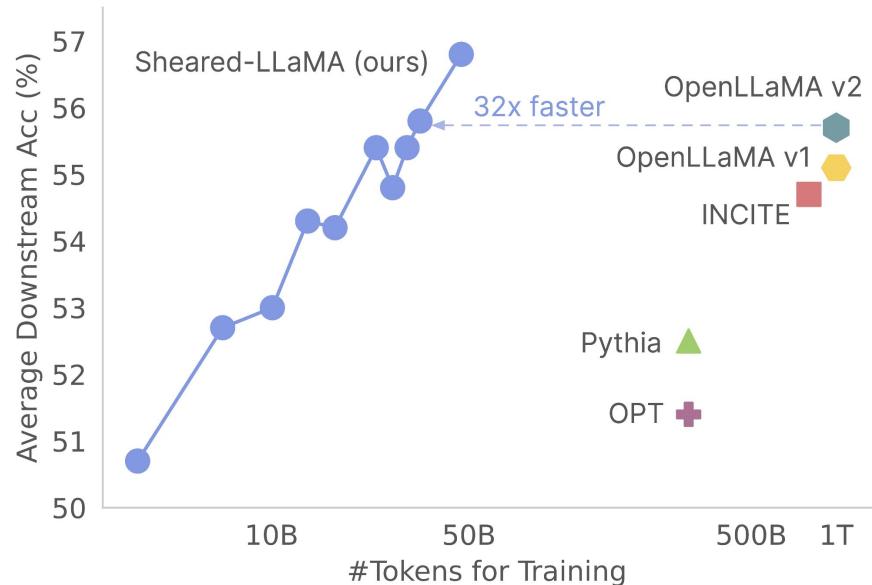
$$\|\mathbf{W}_P\|_0 < N$$

- L represents the objective function for neural network training;
- \mathbf{x} is input, \mathbf{W} is original weights, \mathbf{W}_P is pruned weights;
- $\|\mathbf{W}_P\|_0$ calculates the #nonzeros in W_P , and N is the target #nonzeros.



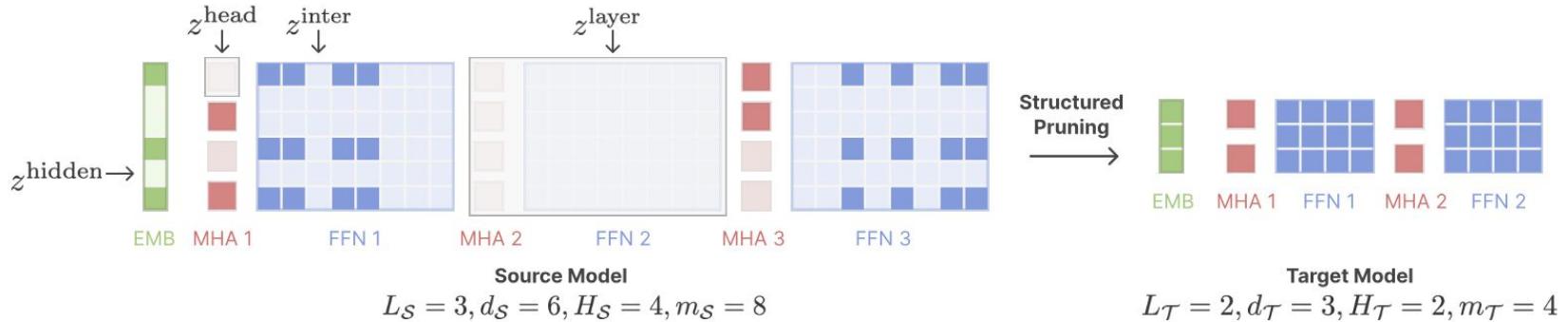
LLM-Shearing: Accelerating via Structured Pruning

An efficient method of constructing LLMs by first pruning a larger existing model and then continually pre-training it.



- Sheared-LLaMA-2.7B achieves better performance than existing open-source models of the same scale with 3% (1/32) of the compute.
- The trajectory shows a compelling case that if we invest more tokens and compute, the capability of Sheared-LLaMA can be further improved.

LLM-Shearing: Accelerating via Structured Pruning



1. Target Structure Pruning: prune a source model to a pre-specified target architecture (e.g., an existing model's config), and meanwhile maximizing the pruned model's performance

LLM-Shearing: Accelerating via Structured Pruning

Algorithm 1: Dynamic Batch Loading

Require: Training data of k domains D_1, D_2, \dots, D_k , validation data $D_1^{\text{val}}, D_2^{\text{val}}, \dots, D_k^{\text{val}}$, initial data loading weights $w_0 \in \mathbb{R}^k$, reference loss $\ell_{\text{ref}} \in \mathbb{R}^k$, LM loss function \mathcal{L} or pruning loss $\mathcal{L}_{\text{prune}}$, training steps T , evaluation interval m , model parameters θ (θ, z, ϕ, λ for pruning)

```
for  $t = 1, \dots, T$  do
    if  $t \bmod m = 0$  then
         $\ell_t[i] \leftarrow \mathcal{L}(\theta, z, D_i^{\text{val}})$  if pruning else  $\mathcal{L}(\theta, D_i^{\text{val}})$ 
         $\Delta_t[i] \leftarrow \max \{\ell_t[i] - \ell_{\text{ref}}[i], 0\}$                                 ▷ Calculate loss difference
         $w_t \leftarrow \text{UpdateWeight}(w_{t-m}, \Delta_t)$                                 ▷ Update data loading proportion
    end
```

Sample a batch of data \mathcal{B} from D_1, D_2, \dots, D_k with proportion w_t ;

```
if pruning then
    | Update  $\theta, z, \phi, \lambda$  with  $\mathcal{L}_{\text{prune}}(\theta, z, \phi, \lambda)$  on  $\mathcal{B}$ 
else
    | Update  $\theta$  with  $\mathcal{L}(\theta, \mathcal{B})$ 
end
```

end

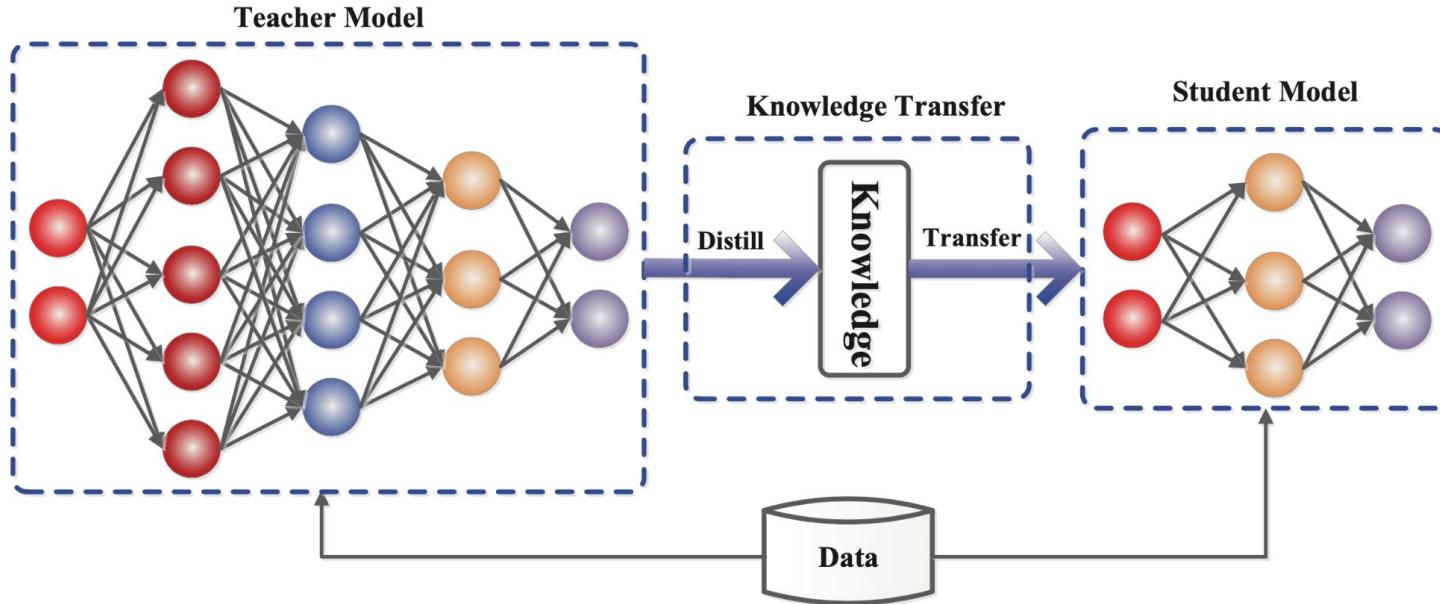
Subroutine $\text{UpdateWeight}(w, \Delta)$

```
     $\alpha \leftarrow w \cdot \exp(\Delta)$                                               ▷ Calculate the unnormalized weights
     $w \leftarrow \frac{\alpha}{\sum_i \alpha[i]}$  return  $w$                                 ▷ Renormalize the data loading proportion
return  $\theta$ 
```

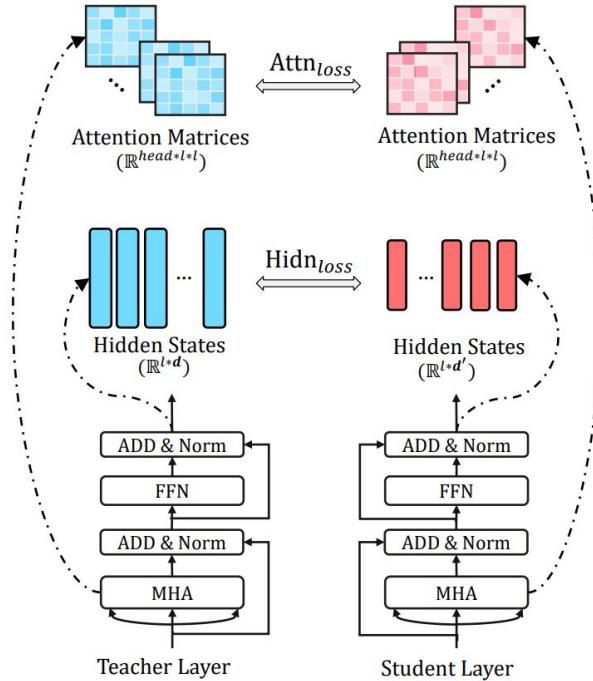
2. Dynamic batch loading:
Pruning results in varying information retainment across domains.
Concretely, they load more data for domains that recover slow, and the loading proportion is dynamically decided on the fly.

Efficiency Beyond Transformers - Distillation

Framework of knowledge distillation



TinyBERT



In transformer, it would be nice to learn attentions from teacher model.

Benefits of KD compared to directly training

- More fine grained supervision (learn on every layers)
- Making use of unannotated data (teacher model provide supervision)
 - Data augmentation is useful.

A few work of KD in LLMs

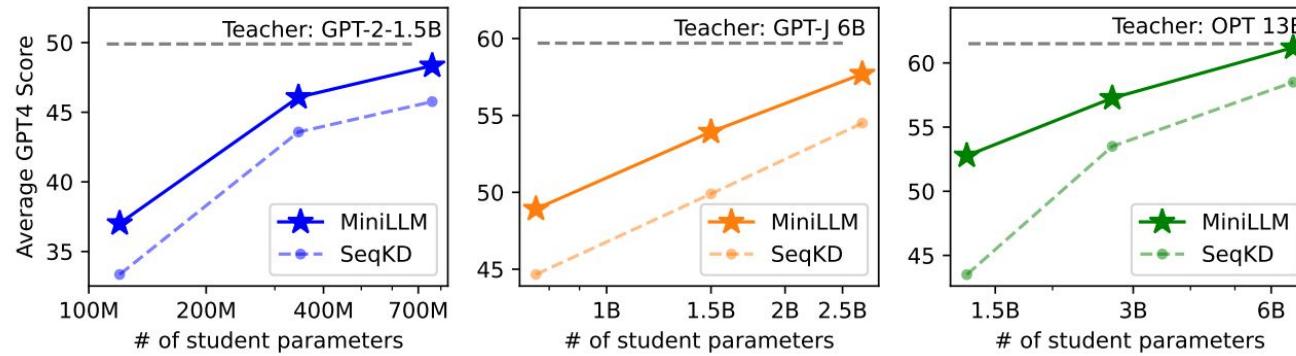


Figure 1: The comparison of MINILM with the sequence-level KD (SeqKD) in terms of the average GPT-4 feedback score on our evaluation sets. **Left:** GPT-2-1.5B as the teacher and GPT-2 125M, 340M, 760M as the students. **Middle:** GPT-J 6B as the teacher and GPT-2 760M, 1.5B, GPT-Neo 2.7B as the students. **Right:** OPT 13B as the teacher and OPT 1.3B, 2.7B, 6.7B as the students.

This seems not that working in LLMs. More investigation is needed

Memory-efficiency training

Models are getting larger and larger

LLMs take much longer time to train!



Boss: What did you do last month?



You: Trained the model for one epoch.



Boss: Umm, fine, what is your plan for next month?



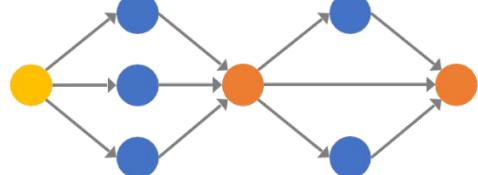
You: Train... train the model for one more epoch?



Distributed Training is almost Necessary for every LLMs!

- Developers / Researchers' time **are more valuable** than hardware .
- If a training takes **10 GPU days**
 - Parallelize with distributed training
 - 1024 GPUs can finish in 14 minutes (ideally)!
- The develop and research cycle will be greatly boosted

Parallelism in Distributed Training - Data Parallelism

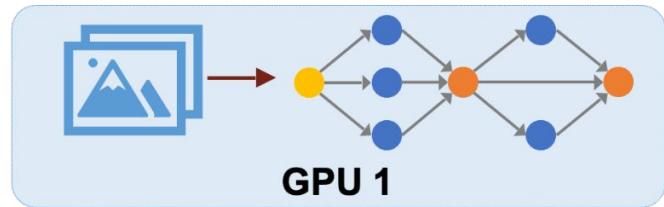


ML Model

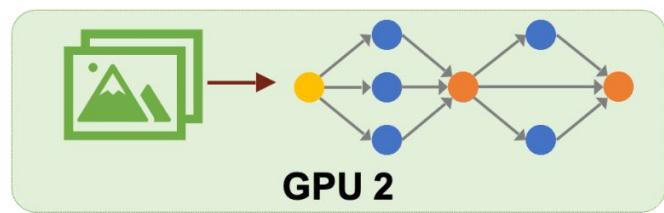


Training Dataset

Data Parallelism

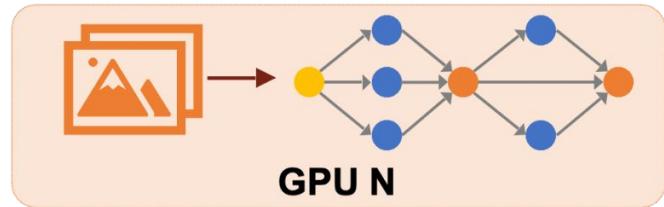


GPU 1



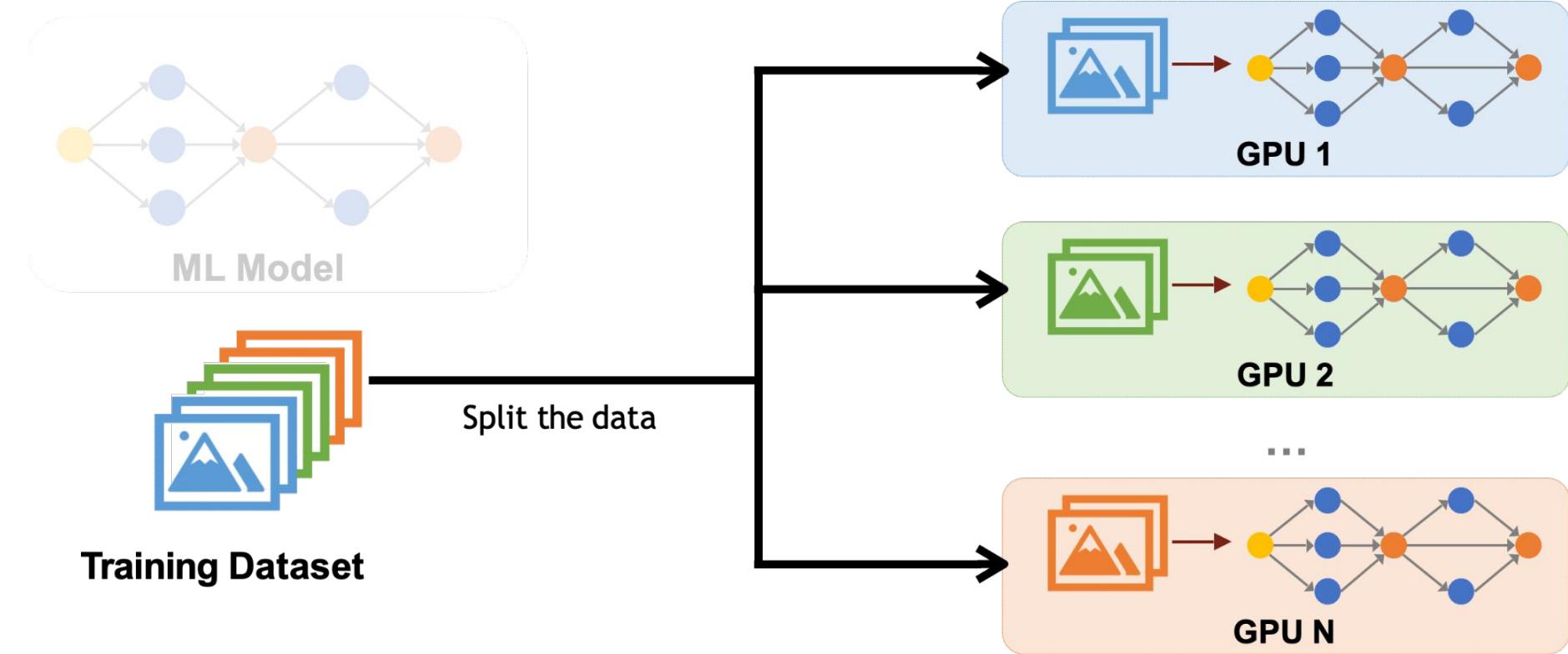
GPU 2

...

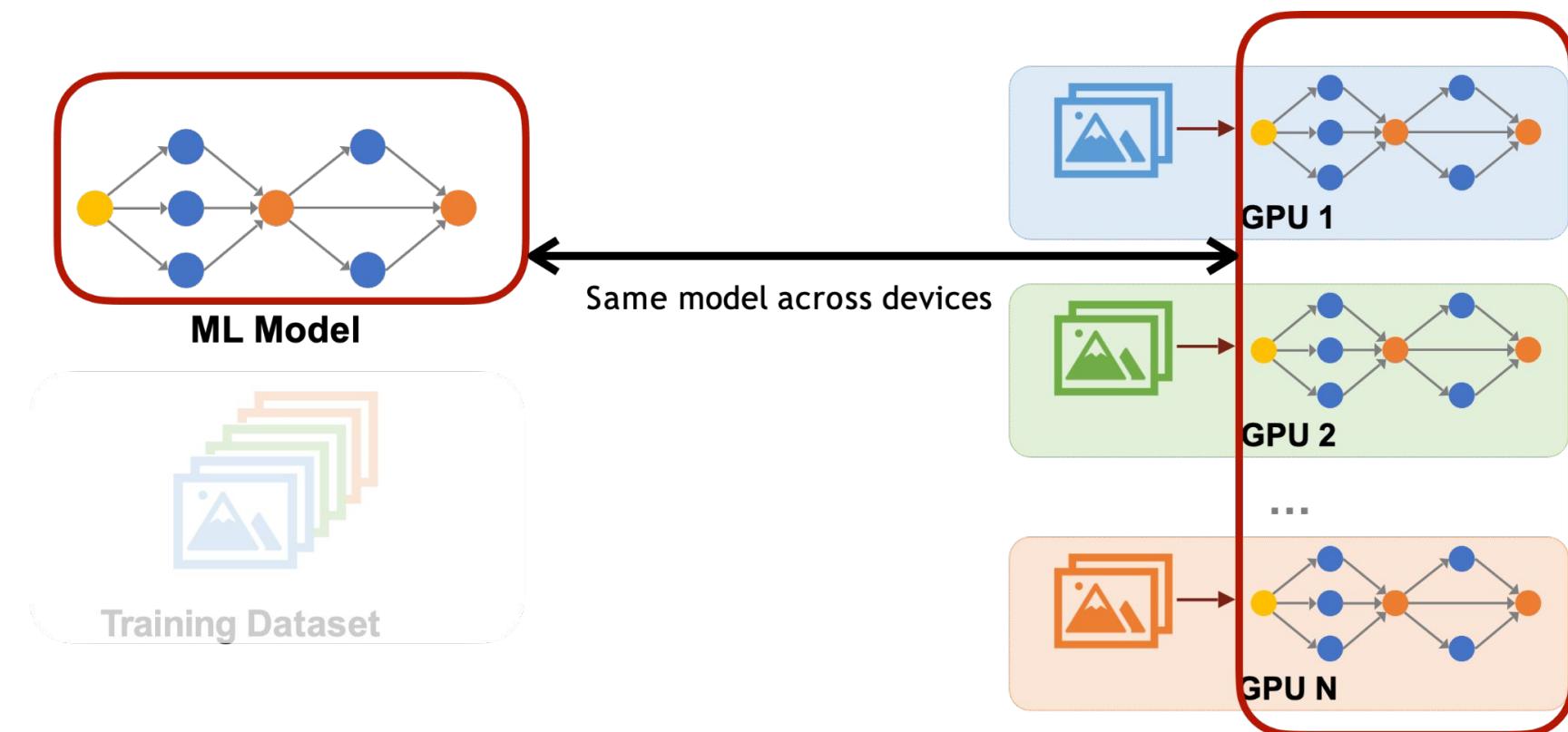


GPU N

Parallelism in Distributed Training - Data Parallelism



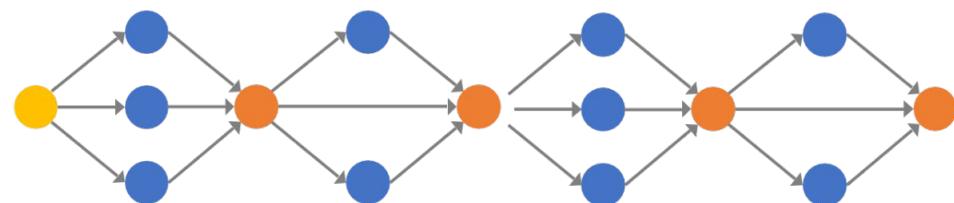
Parallelism in Distributed Training - Data Parallelism



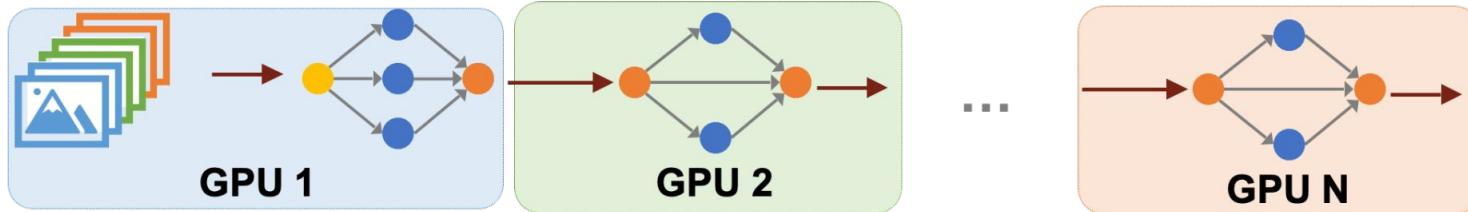
Parallelism in Distributed Training - Model Parallelism



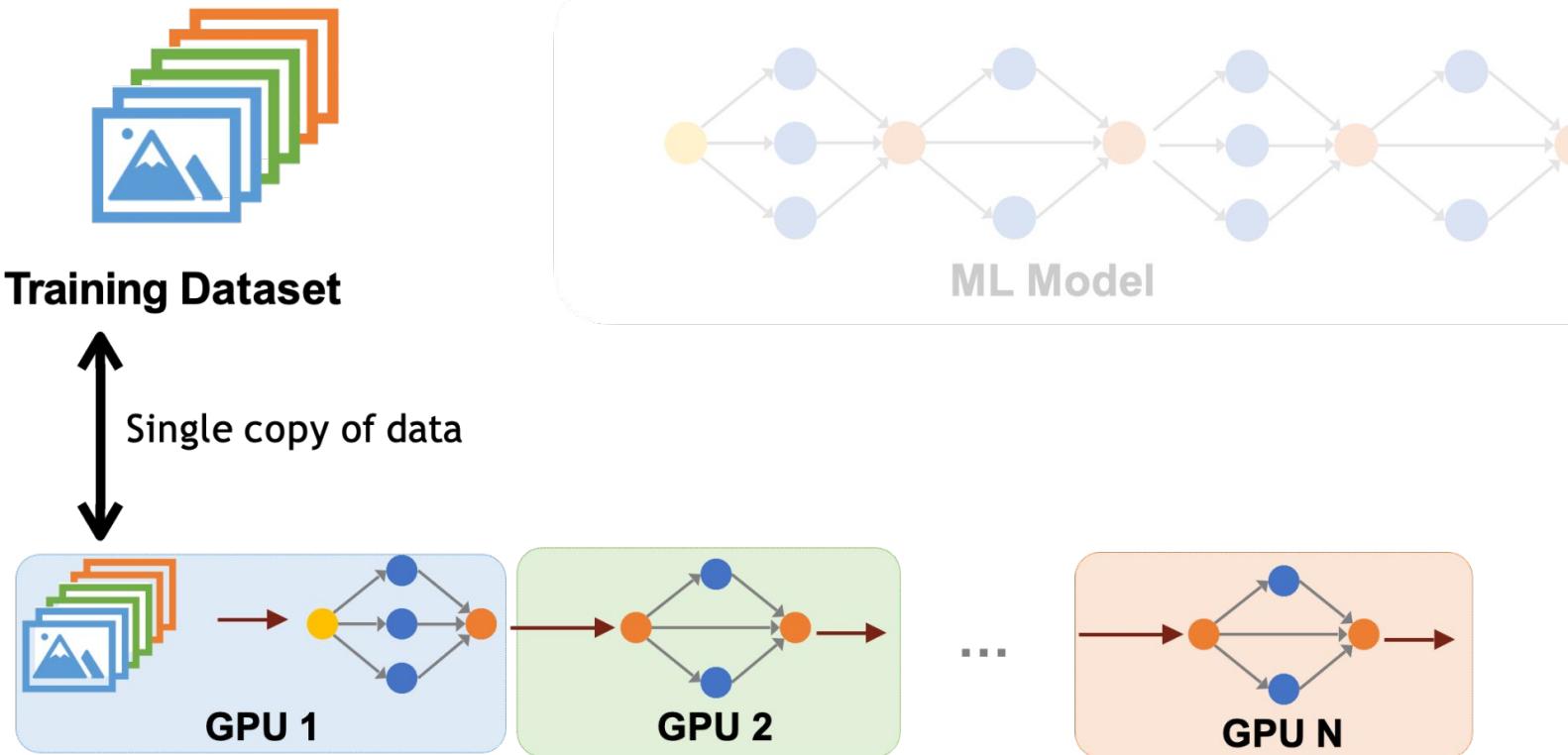
Training Dataset



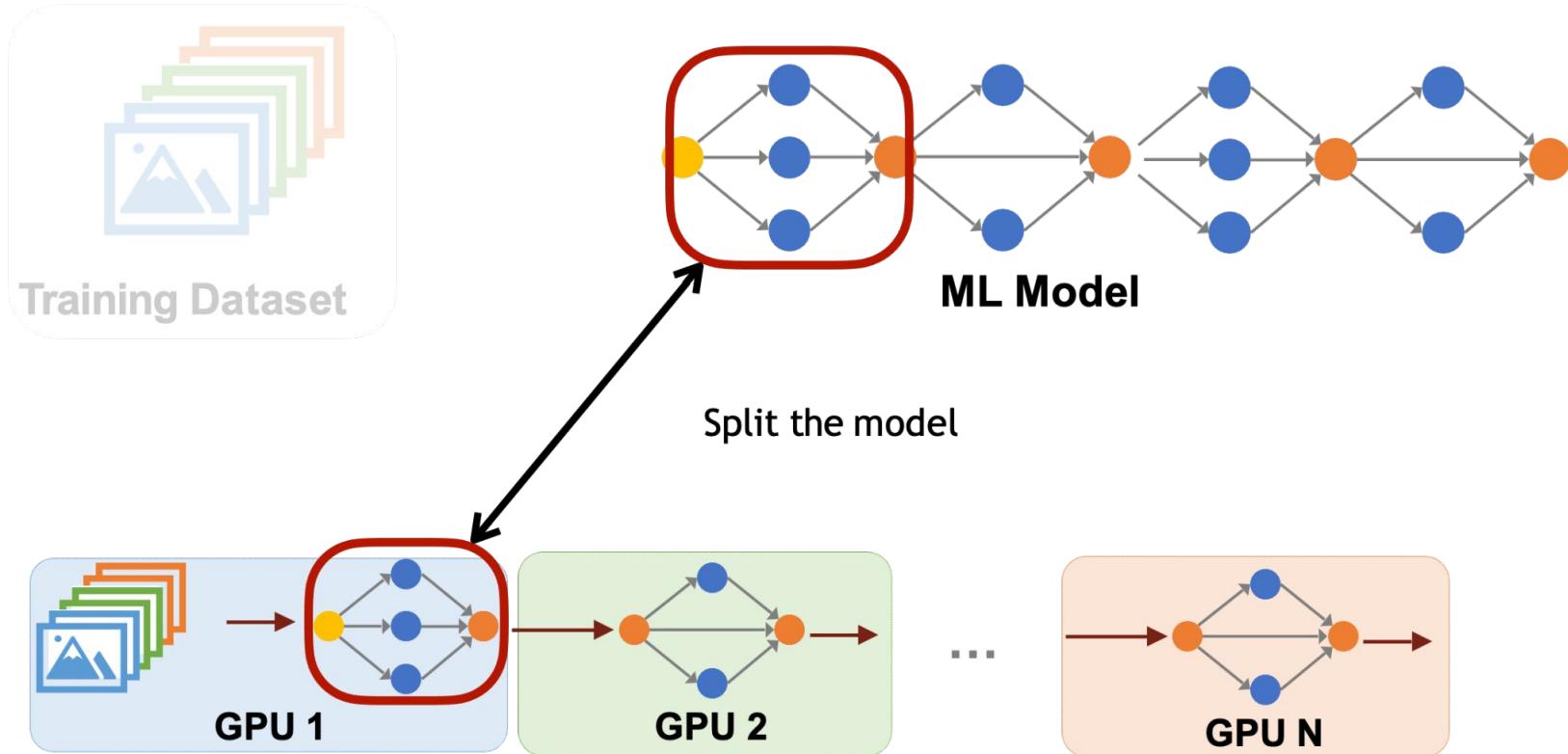
ML Model



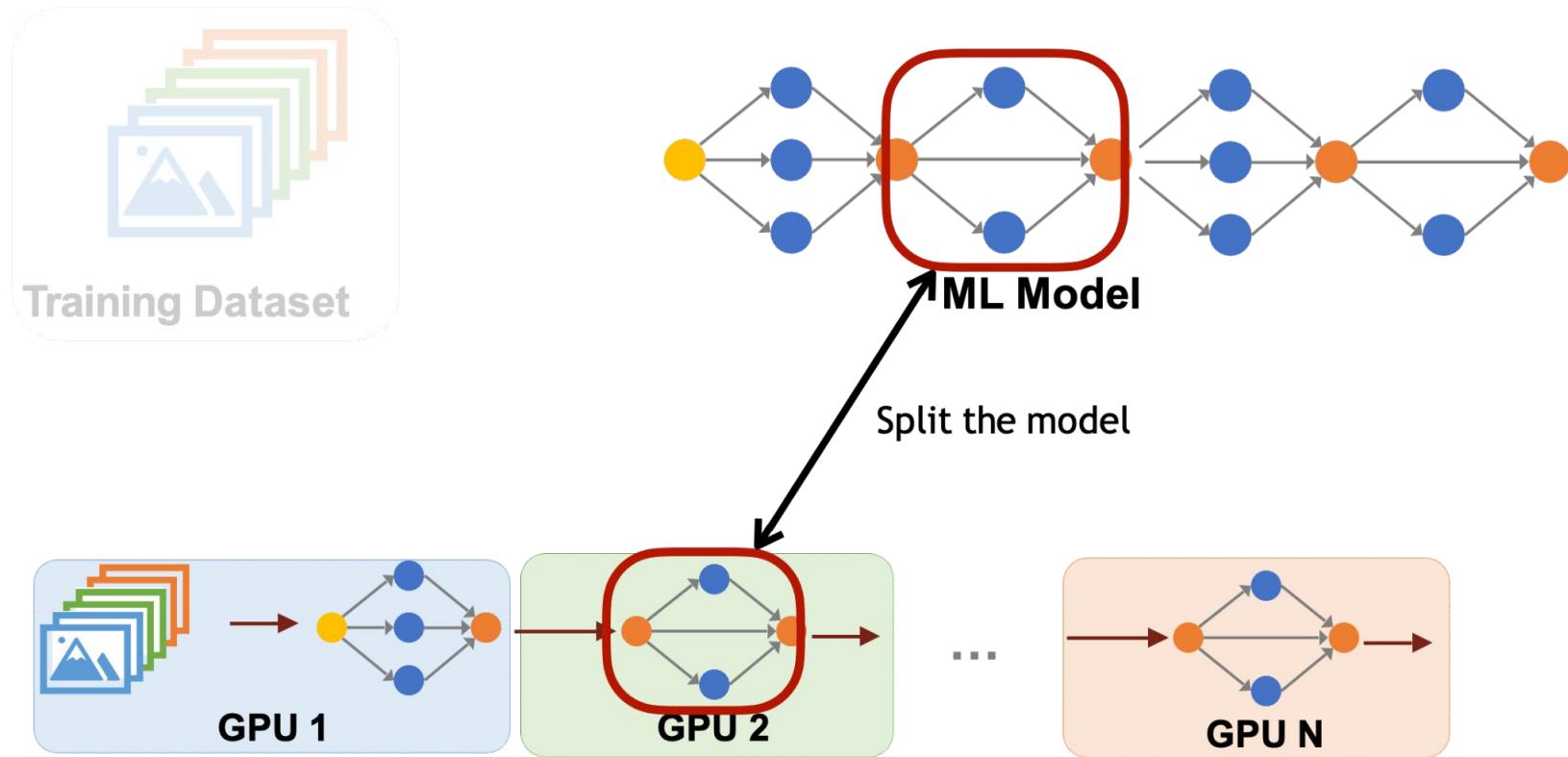
Parallelism in Distributed Training - Model Parallelism



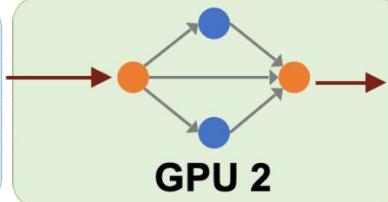
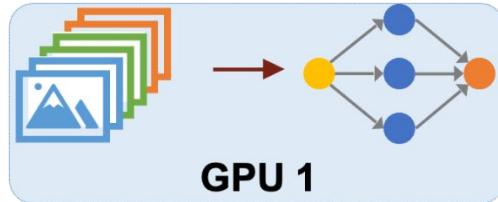
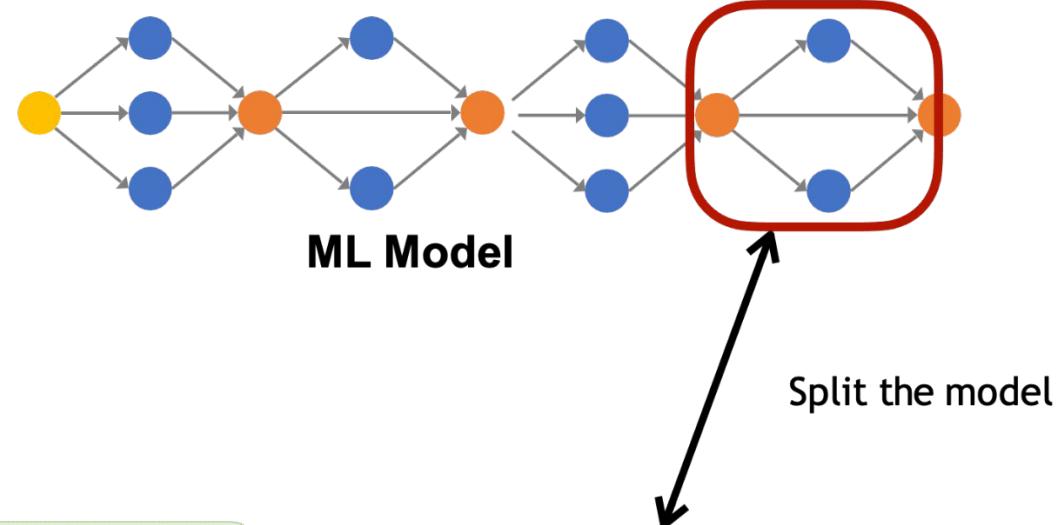
Parallelism in Distributed Training - Model Parallelism



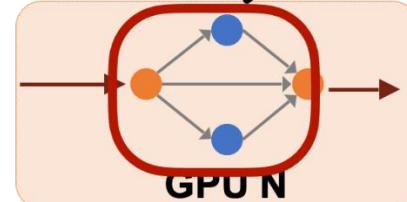
Parallelism in Distributed Training - Model Parallelism



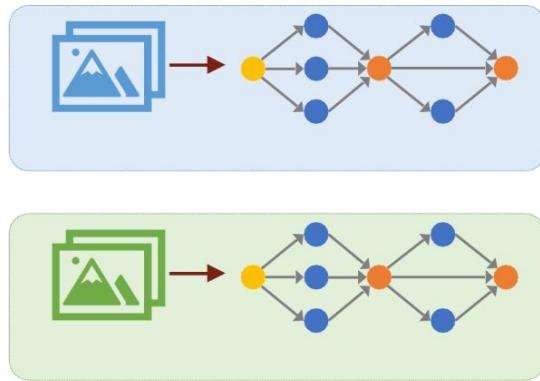
Parallelism in Distributed Training - Model Parallelism



...

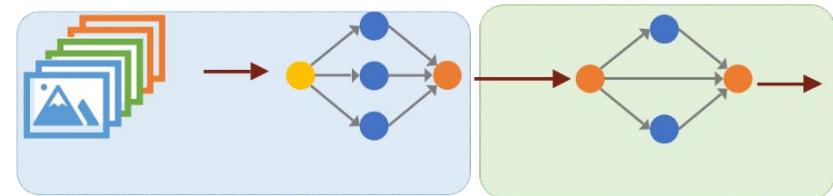


Parallelism in Distributed Training - DP vs MP



Data Parallelism:

- Split the data
- Same model across devices
- Easy to parallelize, high utilization
- N copies of model



Model Parallelism:

- Split the model
- Move activations through devices
- Hard to parallelize, load balancing issue
- Single copy of model

Distributed Training and Memory Optimizations - ZeRO: Train Trillion-scale models

Let's take a step back for training a singler layer in practice

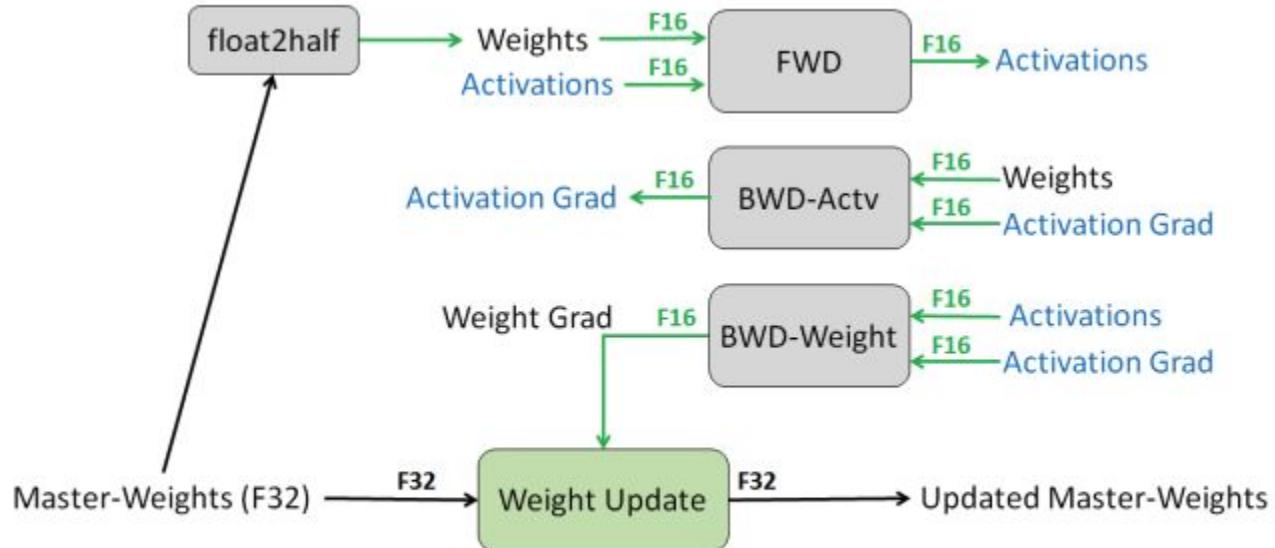


Figure 1: Mixed precision training iteration for a layer.

Memory Consumptions for this example:

Suppose the layer (or model) is trained using Adam Optimizer.

The number of parameters are Φ .

Then in a single training iteration, we have to save (corresponding memory consumption):

- Model parameters (fp16): 2Φ
- Model gradients (fp16): 2Φ
- Adam Optimizer states - copy of Parameters, Momentum and Variance (fp32): $4\Phi + 4\Phi + 4\Phi = 12\Phi$
- Residual states, including activations, buffer, fragmentations

For a GPT-2 model, even it has only 1.5B model parameters (3GB memory is enough to hold it), training it would cost at least 24GB memory!

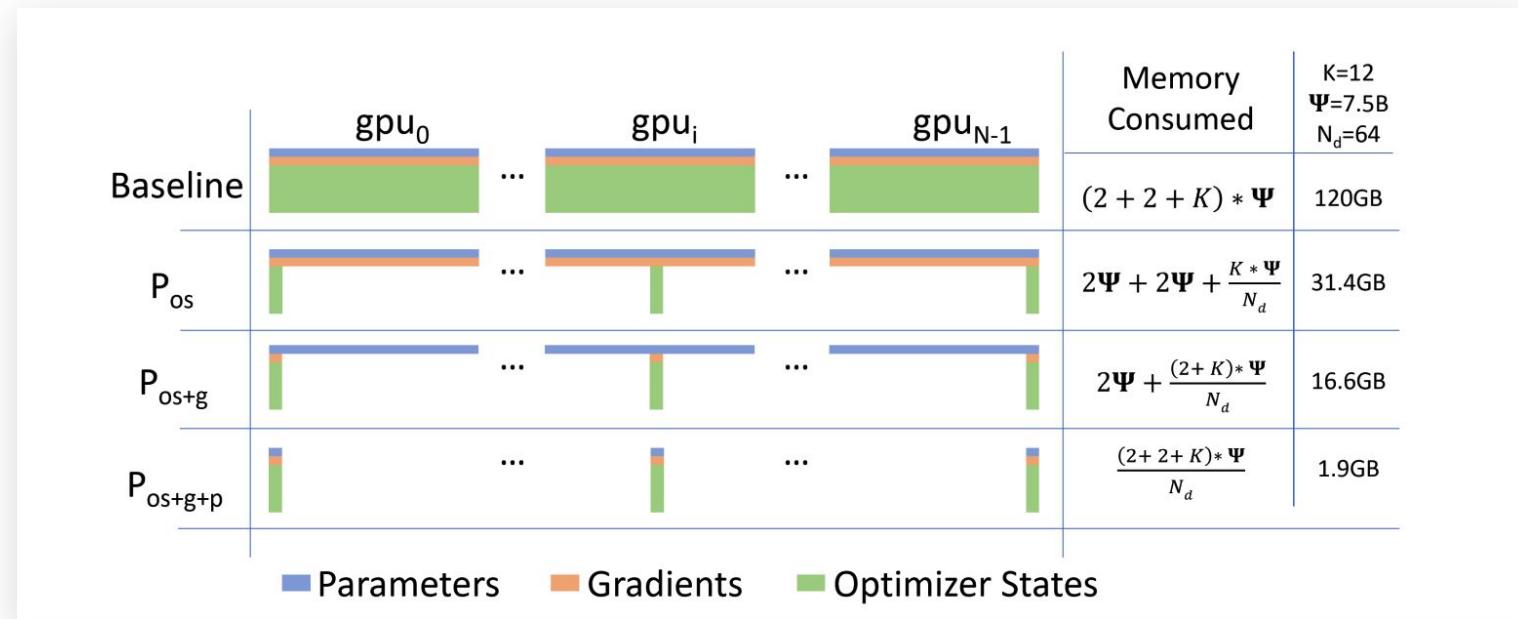
VRam Estimation

Model: HuatuoGPT-7B

1. Model
 - a. Param(fp16): $7\text{B} \times 2 = 14\text{GB}$
 - b. Grad(fp16): $7\text{B} \times 2 = 14\text{GB}$
2. Optimizer(AdamW)
 - a. Master Weights(fp32): $7\text{B} \times 4 = 28\text{GB}$
 - b. Adam m(fp32): $7\text{B} \times 4 = 28\text{GB}$
 - c. Adam v(fp32): $7\text{B} \times 4 = 28\text{GB}$
3. Activation
4. Buffer&Fragmentation

Parallel Strategy: ZeRO

1. ZeRO-DP: Shard the optimizer state
2. ZeRO-1&2: Same communication volume as DP
3. ZeRO-3: 1.5 communication volume as DP



ZeRO: the More GPUs, the Less Memory Consumption!

DP	7.5B Model (GB)			128B Model (GB)			1T Model (GB)		
	P _{os}	P _{os+g}	P _{os+g+p}	P _{os}	P _{os+g}	P _{os+g+p}	P _{os}	P _{os+g}	P _{os+g+p}
1	120	120	120	2048	2048	2048	16000	16000	16000
4	52.5	41.3	30	896	704	512	7000	5500	4000
16	35.6	21.6	7.5	608	368	128	4750	2875	1000
64	31.4	16.6	1.88	536	284	32	4187	2218	250
256	30.4	15.4	0.47	518	263	8	4046	2054	62.5
1024	30.1	15.1	0.12	513	257	2	4011	2013	15.6

- We can train a 7.5B model (like Llama2) using only 4 V100-32GB GPUs
- We can even train a 128B model using 64 V100-32GB GPUs

Future

- Efficiency Long context for LLMs
- Hybrid efficiency
 - QLoRA
 - LongLoRA
 - QMOE
- MOE and modularization

Distributed Training and Memory Optimizations - LOMO: Fuse gradient computation and gradient update!

LOMO: Compute Gradient and Update Parameter in SGD

Algorithm 1 Fusion Update in LOMO

Require: model $f(\cdot)$ with L layers and p parameters, parameter $\theta \in \mathbb{R}^p$, learning rate α , max step T , training dataset \mathcal{D} , loss function \mathcal{L}

```
1: for  $t = 1, \dots, T$  do
2:   Sample batch  $\mathcal{B} = (\mathbf{x}, \mathbf{y}) \subset \mathcal{D}$ 
3:    $\hat{\mathbf{y}} \leftarrow f(\mathbf{x}, \theta)$                                      ▷ Forward pass
4:    $\ell \leftarrow \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 
5:   for  $l = L, \dots, 1$  do                                         ▷ Backward propagation
6:      $\theta_l \leftarrow [\theta_i \text{ for } \theta_i \in \text{layer } l]$ 
7:      $\mathbf{g}_l \leftarrow \frac{\partial \ell}{\partial \theta_l}$ 
8:      $\theta_l \leftarrow \theta_l - \alpha * \mathbf{g}_l$ 
9:      $\mathbf{g}_l \leftarrow \text{None}$                                          ▷ Clear gradients
10:    end for
11: end for
```

In detail, we can express the vanilla gradient descent as $\text{grad} = \partial \mathcal{L} / \partial p$, $p = p - lr * \text{grad}$, which is a two-step process, computing the gradients first and updating it to the parameters. The fusion version is $p = p - lr * \partial \mathcal{L} / \partial p$

LOMO: Computation Graph and Space Complexity

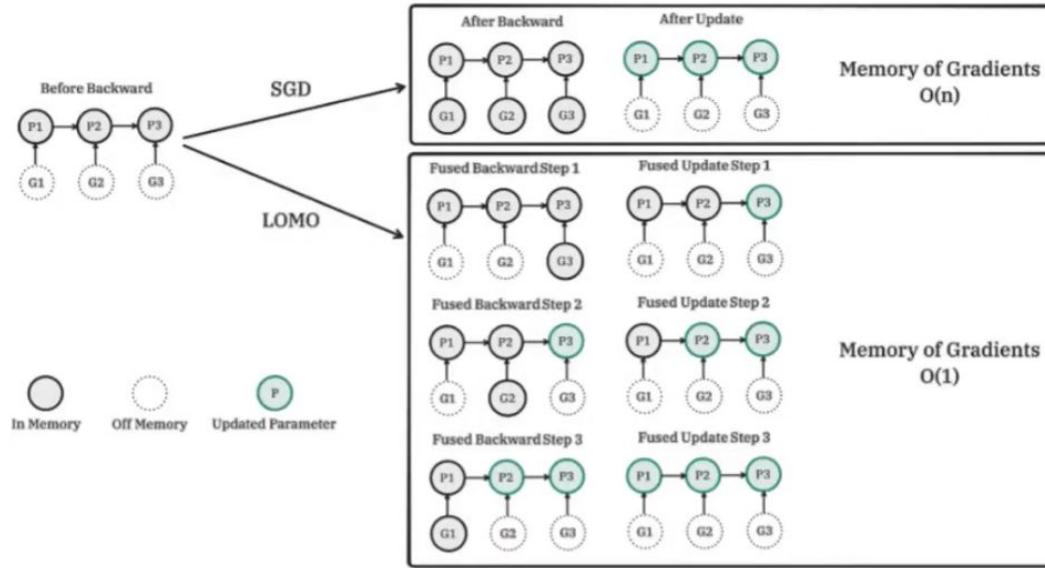


Figure : Comparison of SGD and LOMO in backpropagation and parameter update stages.

P_i refers to the parameter of the model and **G_i** refers to the gradient corresponding to **P_i**.

LOMO fused gradient computation and parameter update in one step to minimize the size of gradient tensors.

LOMO: Main results

- **Memory Profile**

	AC	Params	Gradients	Optim States	Activations	Total Memory
AdamW	X ✓	12.55	12.55	75.31	45.61 1.79	147.02 102.20
SGD	X ✓	12.55	12.55	25.10	45.61 1.79	96.81 51.99
LOMO	X ✓	12.55	0.24	0.00 ↴	45.61 1.79	59.40 14.58

Table : Memory usage (GB) when training LLaMA-7B under different settings.

AC refers to Activation Checkpointing.

- **Downstream performance**

Method	Params	RTE	BoolQ	WSC	WIC	MultiRC	COPA	Avg.
Zero-shot	7B	57.0	66.5	36.5	49.7	42.3	85.0	56.2
LoRA	7B	85.9	85.2	64.4	65.5	84.8	87.0	78.8
LOMO	7B	86.6	87.5	66.4	71.2	84.0	89.0	80.8
Zero-shot	13B	60.6	65.0	36.5	49.5	43.4	88.0	57.2
LoRA	13B	89.9	87.1	63.5	69.9	86.1	92.0	81.4
LOMO	13B	89.9	87.3	75.0	74.3	85.7	93.0	84.2
Zero-shot	30B	53.4	74.6	36.5	50.0	46.9	89.0	58.4
LoRA	30B	91.0	89.7	83.7	74.0	87.0	93.0	86.4
LOMO	30B	92.8	89.3	85.6	74.1	87.9	93.0	87.1
Zero-shot	65B	59.6	73.6	44.2	51.3	48.3	91.0	61.3
LoRA	65B	93.1	90.9	88.5	74.5	90.0	97.0	89.0
LOMO	65B	93.9	90.7	92.3	75.4	89.9	97.0	89.9

Table 3: Main results on SuperGLUE using LLaMA at all sizes (with 1,000 training examples).

QLoRA: Combine Quantization and Low-Rank Approximations

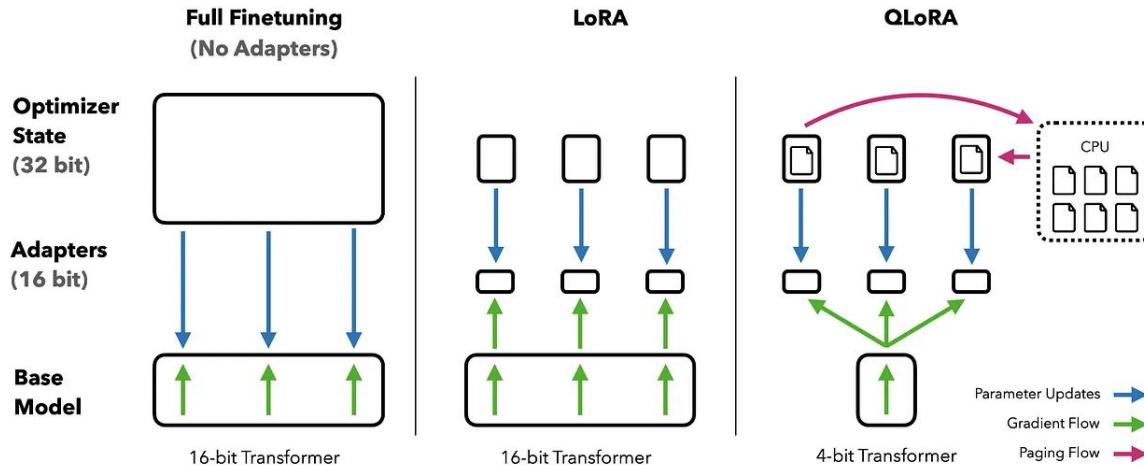
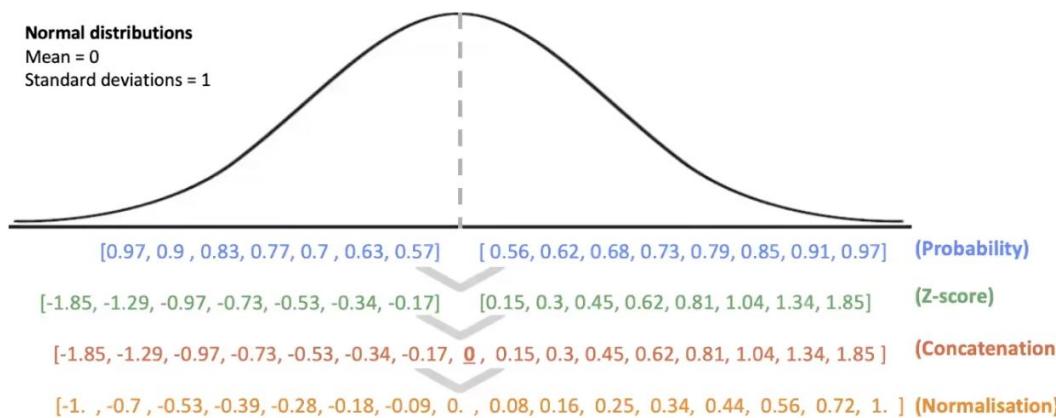


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Key Idea: The LLM is loaded with 4bit. During training, the values are de-quantized to bf16 for processing. Utilizing the features of LoRA, the original model parameters can be locked and excluded from training. Only a small number of LoRA parameters are trained, which significantly reduces the required GPU memory.

QLoRA: 4-bit NormalFloat Quantization

4-bit NormalFloat (NF4) an information-theoretically optimal data type for normal distributions



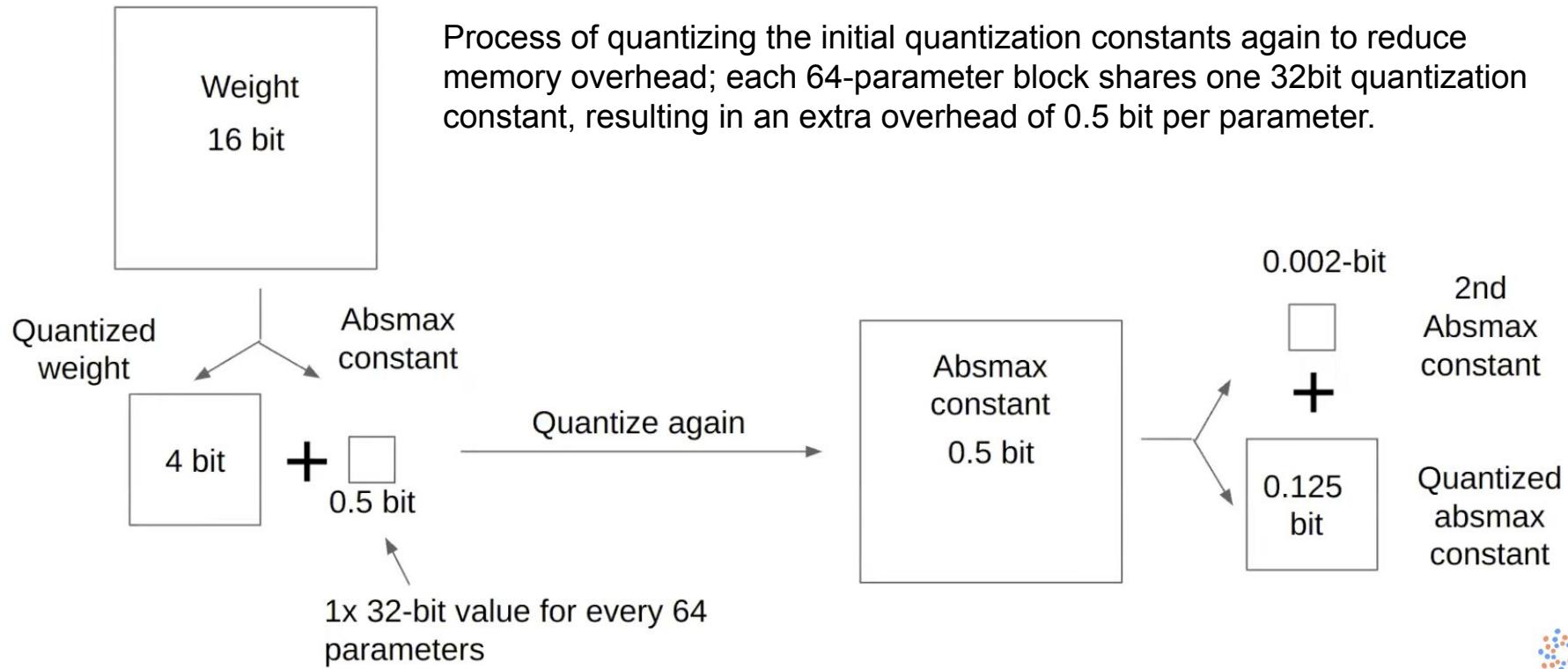
Steps for generating the NF4 data type values:

1. Generate 8 evenly spaced values from 0.56 to 0.97 (Set I).
2. Generate 7 evenly spaced values from 0.57 to 0.97 (Set II).
3. Calculate the z-score values for the probabilities generated in Step 1 and Step 2. For Set II, calculate the negative inverse of the z-scores.
4. Concatenate Set I, a zero value, and Set II together.
5. Normalize the values by dividing them by the absolute maximum value.

- Based on quantile quantization techniques, making data conform to a $N(0,1)$ distribution.

- Adjusts weights to match the data type range by using scaling factors and maintaining zero points during quantization.

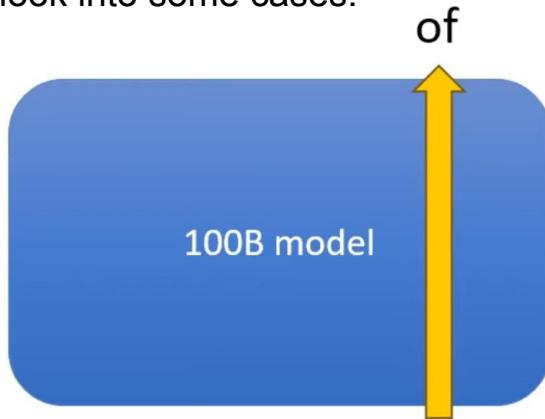
QLoRA: Double Quantization



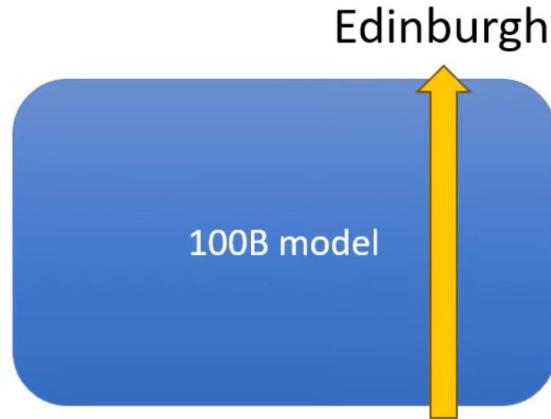
Efficiency Beyond Transformers - Speculative Sampling

Speculative Sampling - Single Token Prediction

Let's look into some cases:



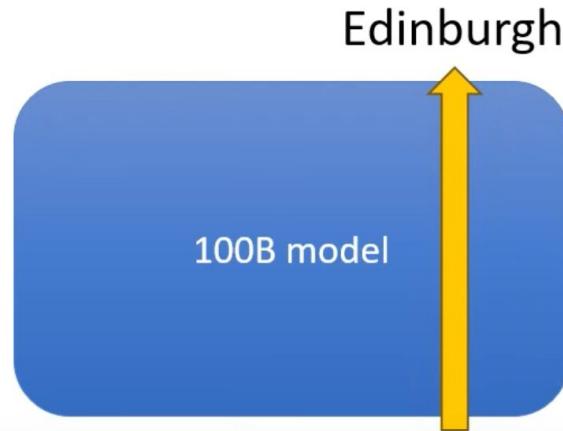
Geoffrey Hinton did his PhD
at the University...



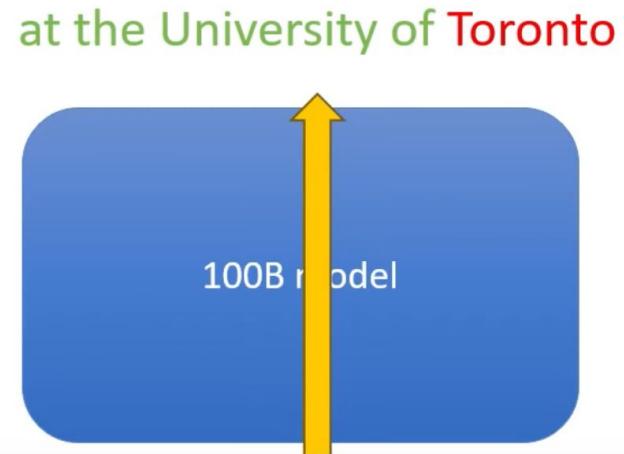
Geoffrey Hinton did his PhD
at the University of...

- Case 1: Predicting “of” is very easy, maybe we should use a 1B model which is enough
- Case 2: Predicting “Edinburgh” requires knowledge, which can be difficult, maybe we should use a 100B model
- This is key idea 1 behind: let small model deal with easy tokens, while large model deals with difficult tokens

Speculative Sampling - Utilize Transformer Structure



Geoffrey Hinton did his PhD
at the University of...



Geoffrey Hinton did his PhD
at the University of Toronto

- We can give a transformer model multiple tokens at once, and let a large transformer model check them in parallel, while it does not increase compute time at well
- In this case, the probability for “Toronto” is low, cause the 100B model recognize it.
- This is key idea 2: let large transformer models check error tokens!

Speculative Sampling - Algorithm

M_p = draft model

≈ meta-llama/Llama-2-7b-chat-hf

M_q = target model

≈ meta-llama/Llama-2-70b-chat-hf

pf = prefix, $K = 5$ tokens

$$p_1(x) = M_p(pf) \xrightarrow{\hspace{10cm}} x_1$$

$$p_2(x) = M_p(pf, x_1) \xrightarrow{\hspace{10cm}} x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \xrightarrow{\hspace{10cm}} x_5$$

Speculative Sampling - Algorithm

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Run draft model
for K steps

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Speculative Sampling - Algorithm

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Speculative Sampling - Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

In this case, we accept “dogs”, “love”, what about “chasing”? - we accept it with probability 0.8/0.9!

Speculative Sampling - Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

In this case, we accept “dogs”, “love”, what about “chasing”? - we accept it with probability $0.8/0.9$, maybe we should accept it!

Speculative Sampling - Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

In this case, we accept “dogs”, “love”, what about “chasing”? - we accept it with probability $0.8/0.9$, maybe we should accept it!

Speculative Sampling - Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

If we accept “chasing”, then what about “after”? The probability = 0.3/0.8, so maybe it should be rejected.

Speculative Sampling - Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

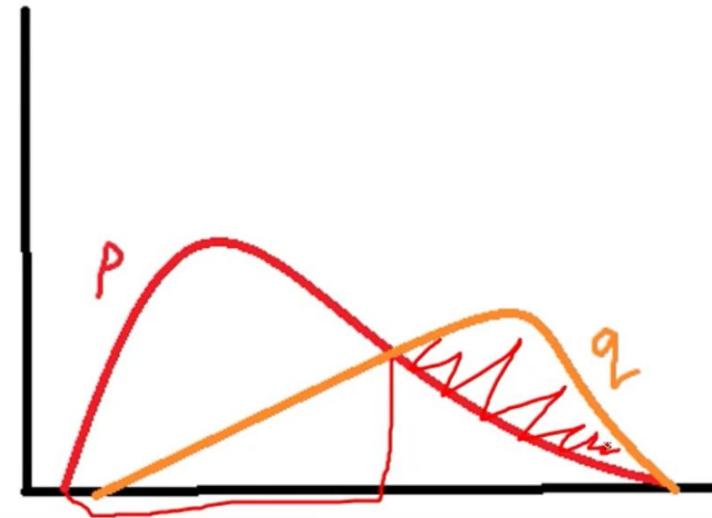
Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

If we reject “after”, then we can sample a token from q(4)!

Speculative Sampling - Rejection Sampling

Actually, don't sample $q(x)$

Adjusted distribution: $(q(x) - p(x))_+$



We sample the 4th token by $(q(4) - p(4))_+!$

Theoretically, we can ensure the token distribution is exactly $q(x)$, so no loss in accuracy!

Speculative Sampling - #tokens generated in one pass

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Worst case: first token is rejected -> 1 token

Best case: all tokens accepted -> K+1 tokens

Speculative Sampling - Wall Time



$K = 7$



$K = 3$



Wall time →

Speculative Sampling - Wall Time

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1x
SpS (Nucleus)		0.114	7.52ms/Token	1.92x
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1x
SpS (Greedy)		0.156	7.00ms/Token	2.01x
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1x
SpS (Nucleus)		47.0%	5.73ms/Token	2.46x



Recommends K = 3-4
Finds 2-2.5x speedup



Recommends K = 3-7
Finds 2-3.4x speedup

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	3.4X
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	2.6X
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	3.1X
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	2.3X
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

System Support for Efficiency - Flash-Attention

Flash-Attention Overview

Background Knowledge: GPU Structure

- SRAM: High-speed Cache Memory
 - High-speed, volatile, limited capacity
- HBM: High Bandwidth Memory
 - High speed, volatile, large capacity

Key Idea: Utilize Characteristics of Attention

- Improve flops, optimize for SRAM storage
- Reduce IO, optimize the data bandwidth and efficiency

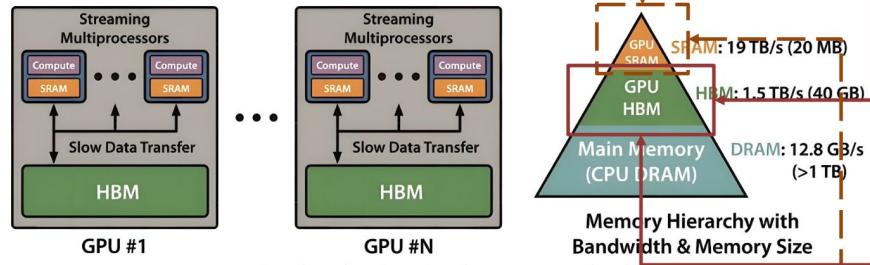
Implementation:

- Softmax: online softmax
 - Online softmax optimization, increasing computational efficiency
- Tiling: On-the-fly tiling (reducing computation)
 - Reduce recomputation (save time and resources)

Algorithm 0 Standard Attention Implementation

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

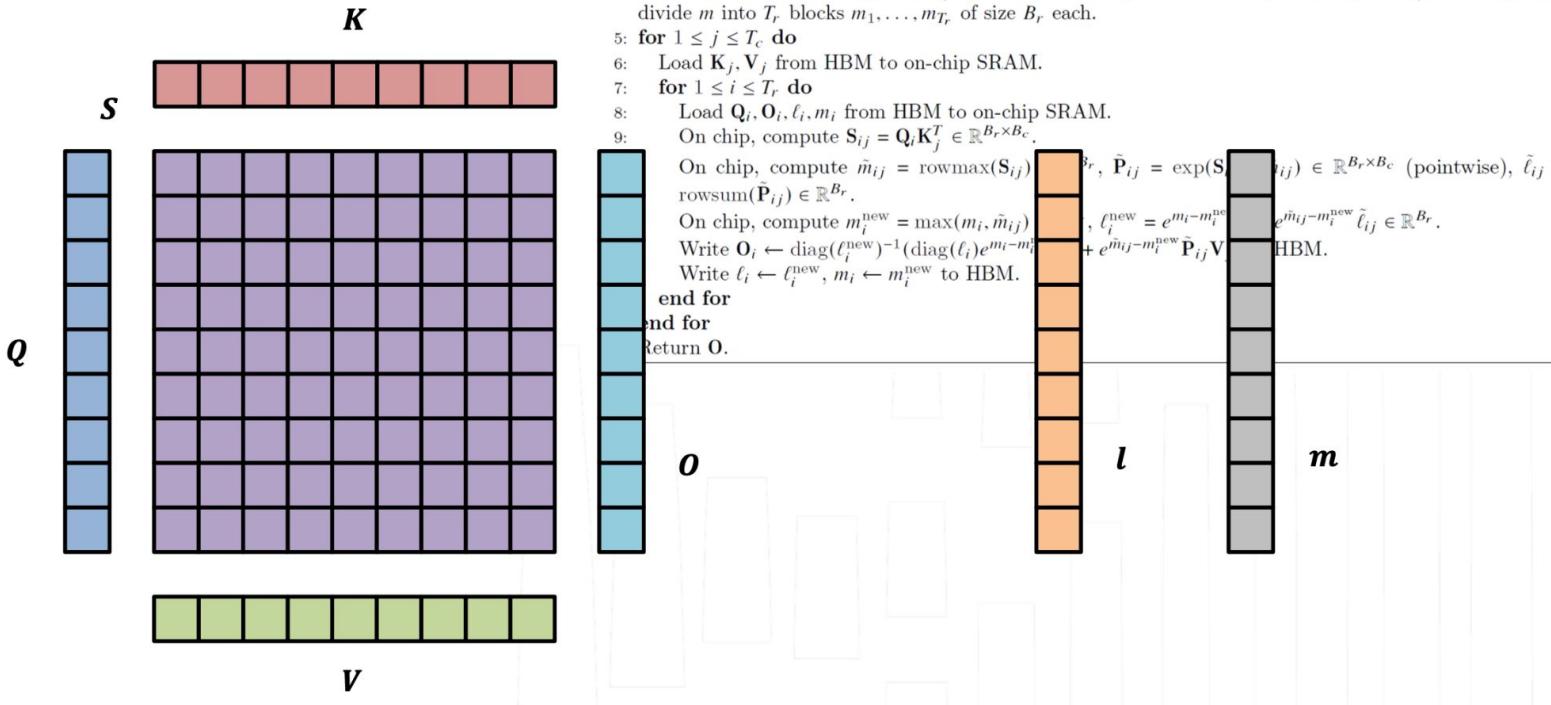
- 1: Load Q, K by blocks from HBM, compute $S = QK^T$, write S to HBM.
 - 2: Read S from HBM, compute $P = \text{softmax}(S)$, write P to HBM.
 - 3: Load P and V by blocks from HBM, compute $O = PV$, write O to HBM.
 - 4: Return O .
-



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

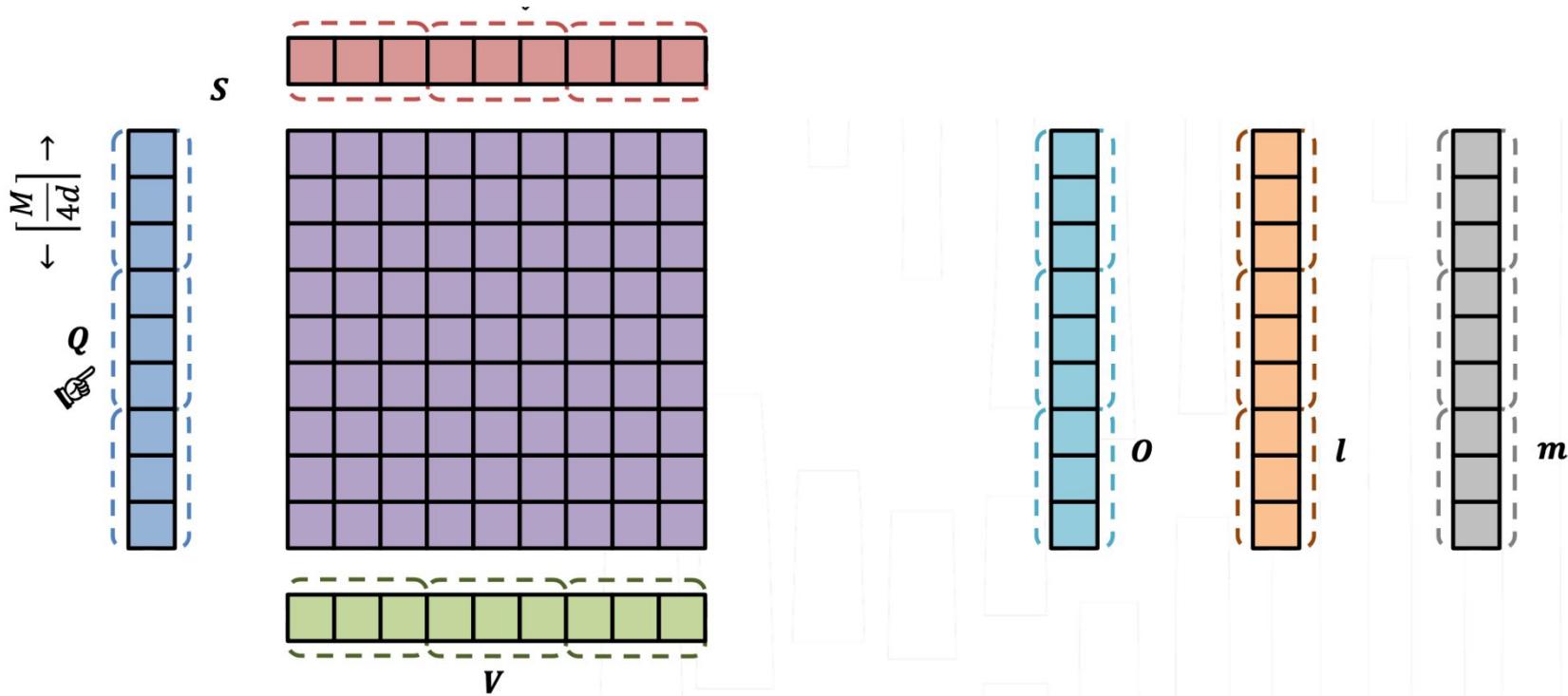
Flash-Attention

Notations



Flash-Attention

Split Blocks



Algorithm 1 FLASHATTENTION

Require: Matrices $Q, K, V \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.

2: Initialize $O = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

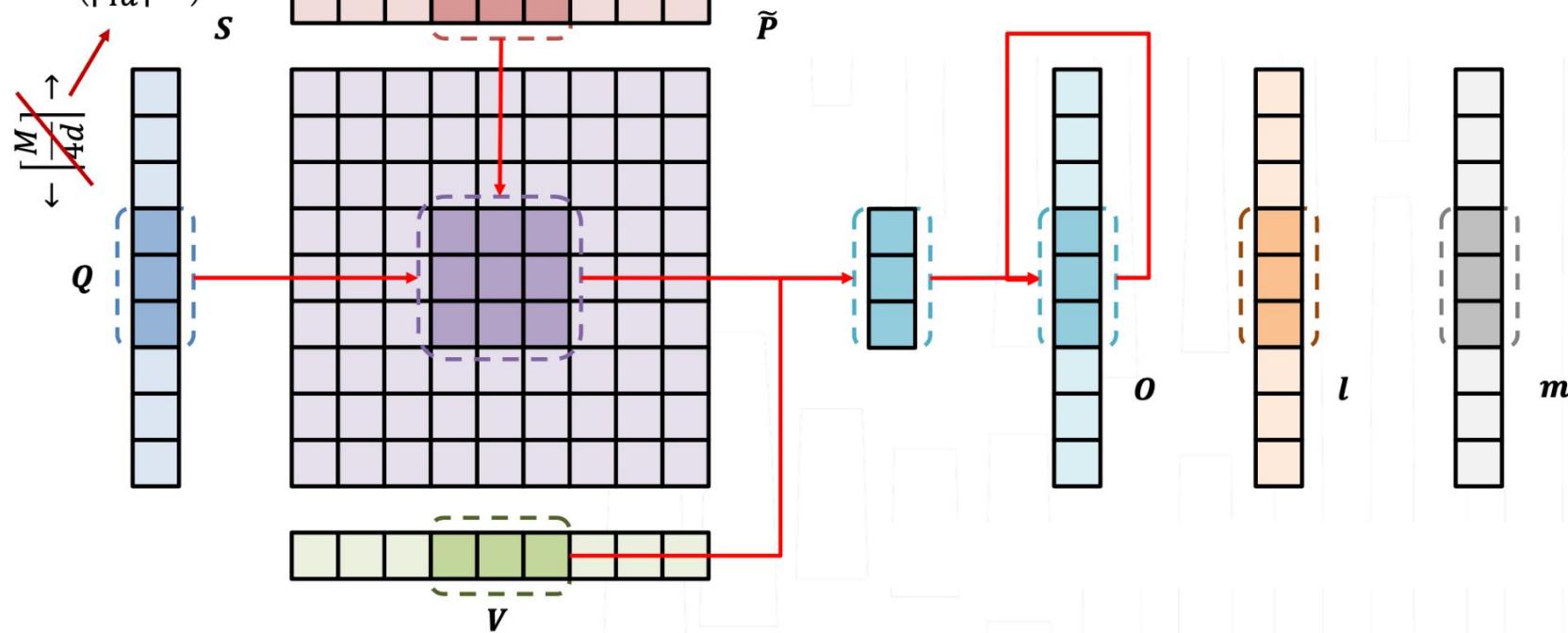
3: Divide Q into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks Q_1, \dots, Q_{T_r} of size $B_r \times d$ each, and divide K, V in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks K_1, \dots, K_{T_c} and V_1, \dots, V_{T_c} , of size $B_c \times d$ each.

4: Divide O into T_r blocks O_1, \dots, O_{T_r} of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.

Flash-Attention

Split Blocks

$$\min(|\overline{4d}|, d)$$



Algorithm 1 FLASHATTENTION

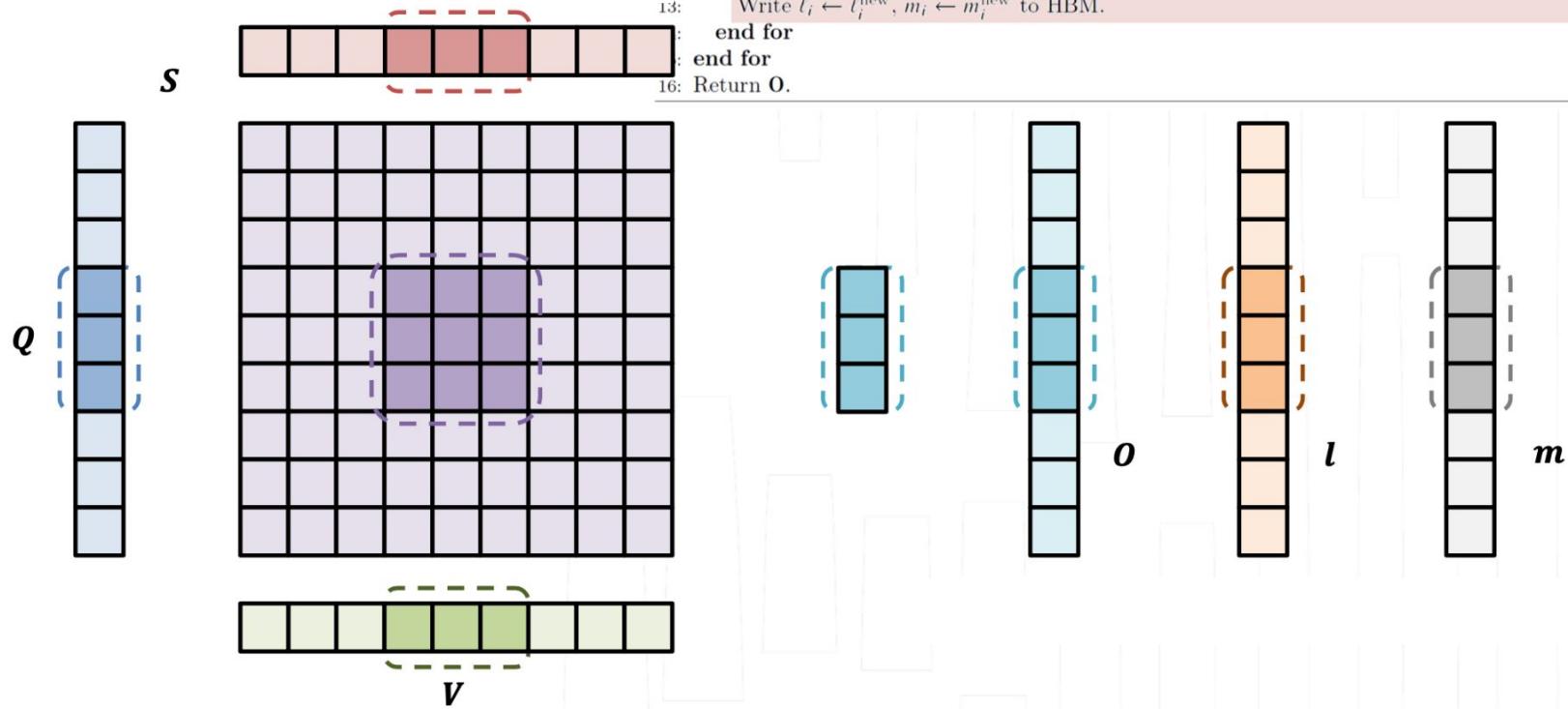
Require: Matrices $Q, K, V \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

- 1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
- 2: Initialize $O = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
- 3: Divide Q into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks Q_1, \dots, Q_{T_r} of size $B_r \times d$ each, and divide K, V in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks K_1, \dots, K_{T_c} and V_1, \dots, V_{T_c} , of size $B_c \times d$ each.
- 4: Divide O into T_r blocks O_1, \dots, O_{T_r} of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.

Flash-Attention

Softmax Reduction

```
1: for  $1 \leq j \leq T_c$  do
2:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
3:   for  $1 \leq i \leq T_r$  do
4:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
5:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
6:     On chip, compute  $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
7:     On chip, compute  $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
8:     Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
9:     Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
10:    end for
11:  end for
12:  Return  $\mathbf{O}$ .
```



```

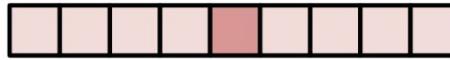
5: for  $1 \leq j \leq T_c$  do
6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
7:   for  $1 \leq i \leq T_r$  do
8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
10:    On chip, compute  $\hat{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \hat{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
11:    On chip, compute  $m_i^{\text{new}} = \max(m_i, \hat{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
14:   end for
15:  end for
16: Return  $\mathbf{O}$ .

```

Flash-Attention

Softmax Reduction

S



: end for
: end for
16: Return \mathbf{O} .

$$O_i = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_N^T} \cdot V_N}{\sum_{j'}^N e^{Q_i K_{j'}^T}}$$



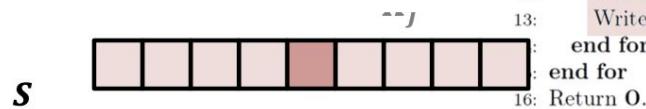
online softmax $O_1 = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^1 e^{Q_i K_{j'}^T}}$ $O_j = O_{j-1} \cdot \frac{\sum_{j'}^{j-1} e^{Q_i K_{j'}^T}}{\sum_{j'}^j e^{Q_i K_{j'}^T}} + \frac{e^{Q_i K_j^T} \cdot V_j}{\sum_{j'}^j e^{Q_i K_{j'}^T}}$



V

Flash-Attention

Softmax Reduction



```

5: for  $1 \leq j \leq T_c$  do
6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
7:   for  $1 \leq i \leq T_r$  do
8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
10:    On chip, compute  $\hat{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \hat{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
11:    On chip, compute  $m_i^{\text{new}} = \max(m_i, \hat{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
14:   end for
15:  end for
16: Return  $\mathbf{O}$ .

```

$$O_i = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_N^T} \cdot V_N}{\sum_{j'}^N e^{Q_i K_{j'}^T}} = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^1 e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^1 e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_j^T} \cdot V_j}{\sum_{j'}^j e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^j e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_N^T} \cdot V_N}{\sum_{j'}^N e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^N e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}}$$



online softmax

$$O_1 = V_1 \quad O_j = O_{j-1} \cdot \frac{l_{ij-1}}{l_{ij}} + \frac{e^{S_{ij}} \cdot V_j}{l_{ij}}$$

$$l_{i1} = e^{S_{i1}} \quad l_{ij} = l_{ij-1} + e^{S_{ij}}$$



V

Flash-Attention

Softmax Reduction
(Numerical Stable)

S



```

5: for  $1 \leq j \leq T_c$  do
6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
7:   for  $1 \leq i \leq T_r$  do
8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
10:    On chip, compute  $\hat{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \hat{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
11:    On chip, compute  $m_i^{\text{new}} = \max(m_i, \hat{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\hat{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
14:  end for
15: end for
16: Return  $\mathbf{O}$ .

```

$$O_i = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_N^T} \cdot V_N}{\sum_{j'}^N e^{Q_i K_{j'}^T}} = \frac{e^{Q_i K_1^T} \cdot V_1}{\sum_{j'}^1 e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^1 e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_j^T} \cdot V_j}{\sum_{j'}^j e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^j e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}} + \dots + \frac{e^{Q_i K_N^T} \cdot V_N}{\sum_{j'}^N e^{Q_i K_{j'}^T}} \cdot \frac{\sum_{j'}^N e^{Q_i K_{j'}^T}}{\sum_{j'}^N e^{Q_i K_{j'}^T}}$$

Q

Q_i



o

o

l

m

online softmax

$$\begin{aligned} O_1 &= V_1 \\ m_{i1} &= S_{i1} \\ l_{i1} &= e^{S_{i1} - m_{i1}} \end{aligned}$$

$$O_i = O_i^{\text{old}} \cdot \frac{l_i^{\text{old}}}{l_i} \cdot \frac{e^{m_i^{\text{old}}}}{e^{m_i}} + \frac{e^{S_i - m_i} \cdot V_j}{l_i}$$

$$m_i = \max(m_i^{\text{old}}, S_i)$$

$$l_i = l_i^{\text{old}} + e^{S_i - m_i}$$

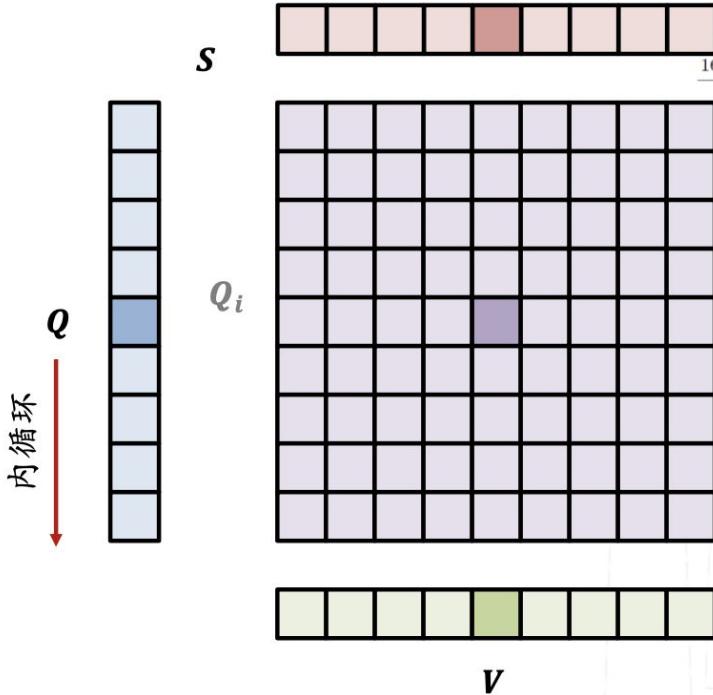


V

$i K_{j'}^T$

Flash-Attention

Softmax Reduction



```

5: for  $1 \leq j \leq T_c$  do
6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
7:   for  $1 \leq i \leq T_f$  do
8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i^\top \mathbf{K}_j$ .
10:    On chip, compute  $\hat{m}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
11:    On chip, compute  $m_i^{\text{new}} = \max(m_i^{\text{old}}, \hat{m}_{ij})$ .
12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\mathbf{O}_i)$ .
13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$ .
14:   end for
15: end for
16: Return  $\mathbf{O}$ .

```

$$\begin{aligned}
\mathbf{O}_i &\Rightarrow \mathbf{O}_i^{\text{old}} \cdot \frac{l_i^{\text{old}}}{l_i} \cdot \frac{e^{m_i^{\text{old}}}}{e^{m_i}} + \frac{|e^{S_{i-1}-m_i} \cdot \mathbf{V}_j|}{l_i} \\
m_i &= \max(m_i^{\text{old}}, S_i) \\
l_i &= l_i^{\text{old}} + |e^{S_{i-1}-m_i}|
\end{aligned}$$

O的增量

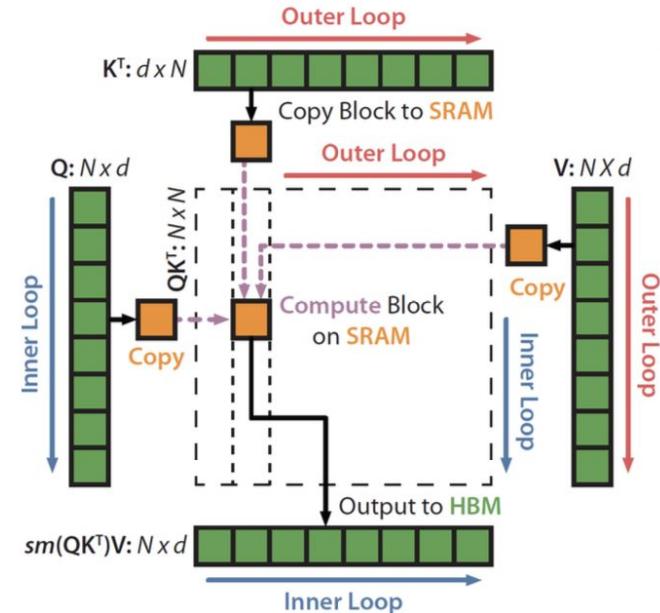
Flash-Attention

Summary: Split blocks, Update Softmax, Complexity= $\mathcal{O}(Nd \cdot Nd/M) = \mathcal{O}(N^2d^2/M)$

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

- 1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
 - 2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
 - 3: Divide \mathbf{Q} into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide \mathbf{K}, \mathbf{V} in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
 - 4: Divide \mathbf{O} into T_r blocks $\mathbf{O}_1, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.
 - 5: **for** $1 \leq j \leq T_c$ **do**
 - 6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
 - 7: **for** $1 \leq i \leq T_r$ **do**
 - 8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
 - 9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
 - 10: On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
 - 11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
 - 12: Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
 - 13: Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.
 - 14: **end for**
 - 15: **end for**
 - 16: Return \mathbf{O} .
-



Acknowledgement

- <https://hanlab.mit.edu/courses/2023-fall-65940>
- <https://hanlab.mit.edu/courses/2022-fall-6s965>
- https://docs.google.com/presentation/d/1EUV7W7X_w0BDrscDhPg7lMGzJCkeaPkGCJ3bN8dluXc/edit?resourcekey=0-7Nz5A7y8JozyVrnDtcEKJA
- https://www.youtube.com/watch?v=bQrdd3BI_fM
- <https://github.com/princeton-nlp/LLM-Shearing>
- https://www.youtube.com/watch?v=S-8yr_RibJ4
- <https://www.youtube.com/watch?v=y9PHWGOa8HA>
- <https://www.youtube.com/watch?v=ZsompoMeIcI>
- <https://www.youtube.com/watch?v=D2DdEstvS30>
- Shanghai AI Lab: 大型语言模型的技术原理