CSCI 5541: Natural Language Processing

Lecture 15: LLM Compute efficiency and engineering

James Mooney

With slides borrowed from Song Han (MIT)

University of Minnesota
Driven to Discover®

Announcements

- Data & Annotation Lecture to be folded into ‘Modern Evaluations
- HW4 due today
- HW5 out tomorrow morning
- Midterm Office Hours Meeting

What Is Efficiency and Why Does It Matter?

- Efficiency for NLP is concerned with delivering faster, cheaper, smaller, less energy intensive solutions to problems involving natural language
- Faster models means LLM model services (GPT3.5, Claude 2.0, etc.) can meet the demands of many clients more quickly
- Cheaper models reduce costs for LLM model service providers
- Smaller model sizes allow for service providers to use fewer resources and can allow for individuals to deploy LLMs to their own (smaller) devices
- Less energy intensive means lower cost and easier to deploy at the edge, where energy is harder to come by

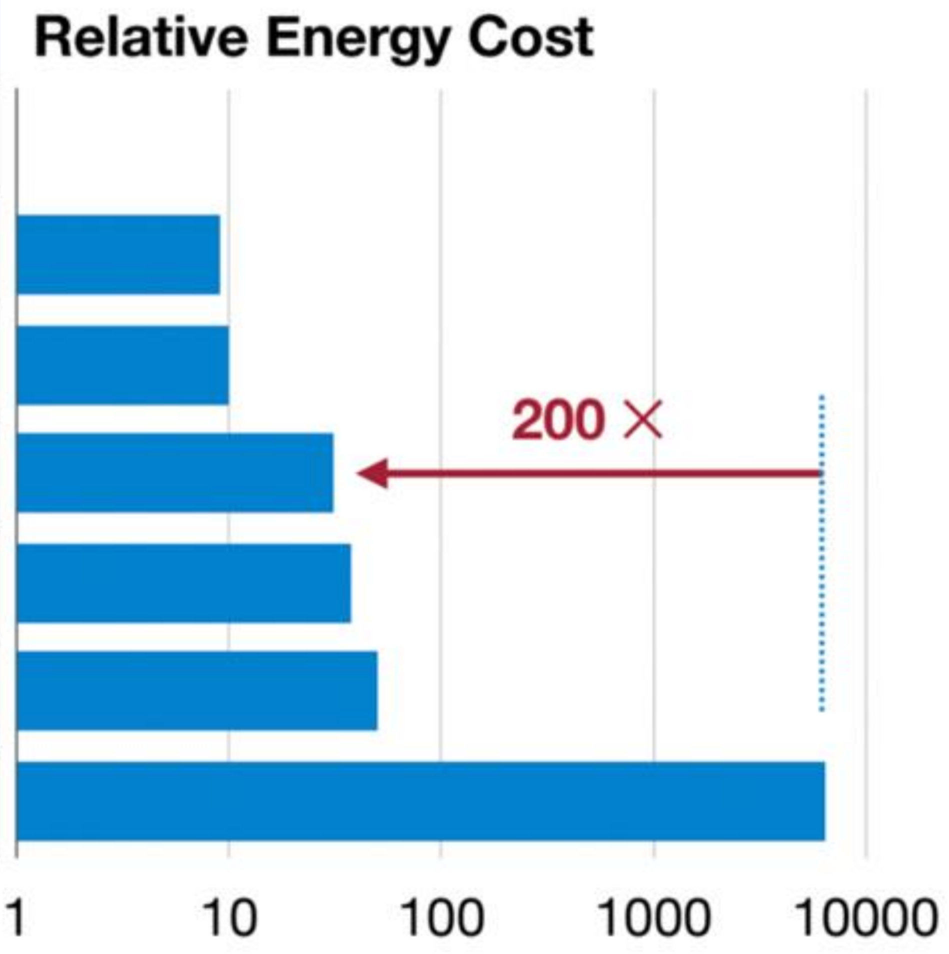
What Is Efficiency and Why Does It Matter?

- Efficiency for NLP is concerned with delivering **faster**, **cheaper**, **smaller**, **less energy** intensive solutions to problems involving natural language
- **Faster** models means LLM model services (GPT3.5, Claude 2.0, etc.) can meet the demands of many clients more quickly
- **Cheaper** models reduce costs for LLM model service providers
- **Smaller** model sizes allow for service providers to use fewer resources and can allow for individuals to deploy LLMs to their own (smaller) devices
- **Less energy** intensive means lower cost and easier to deploy at the edge, where energy is harder to come by

Model Energy Use

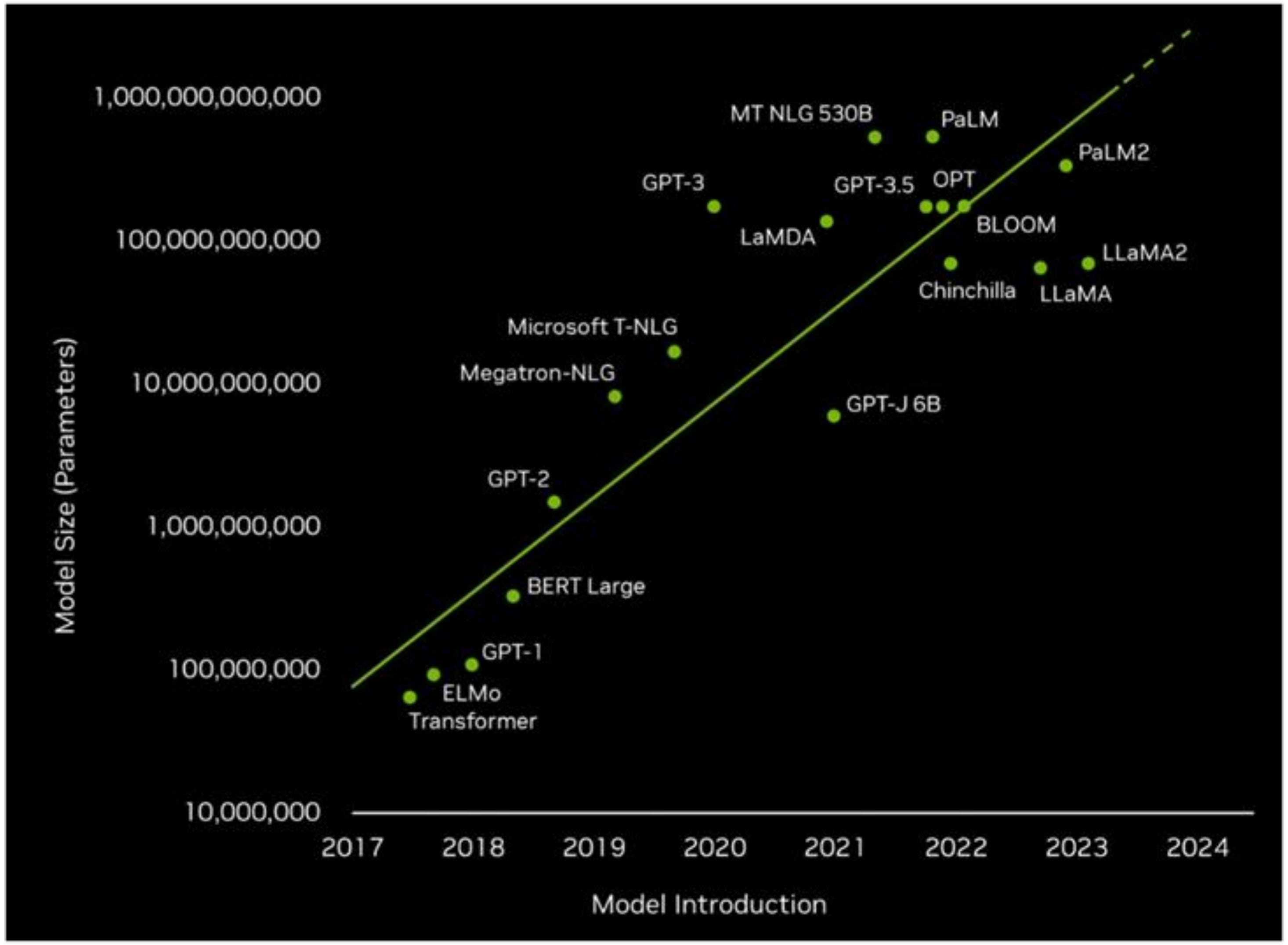
Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V

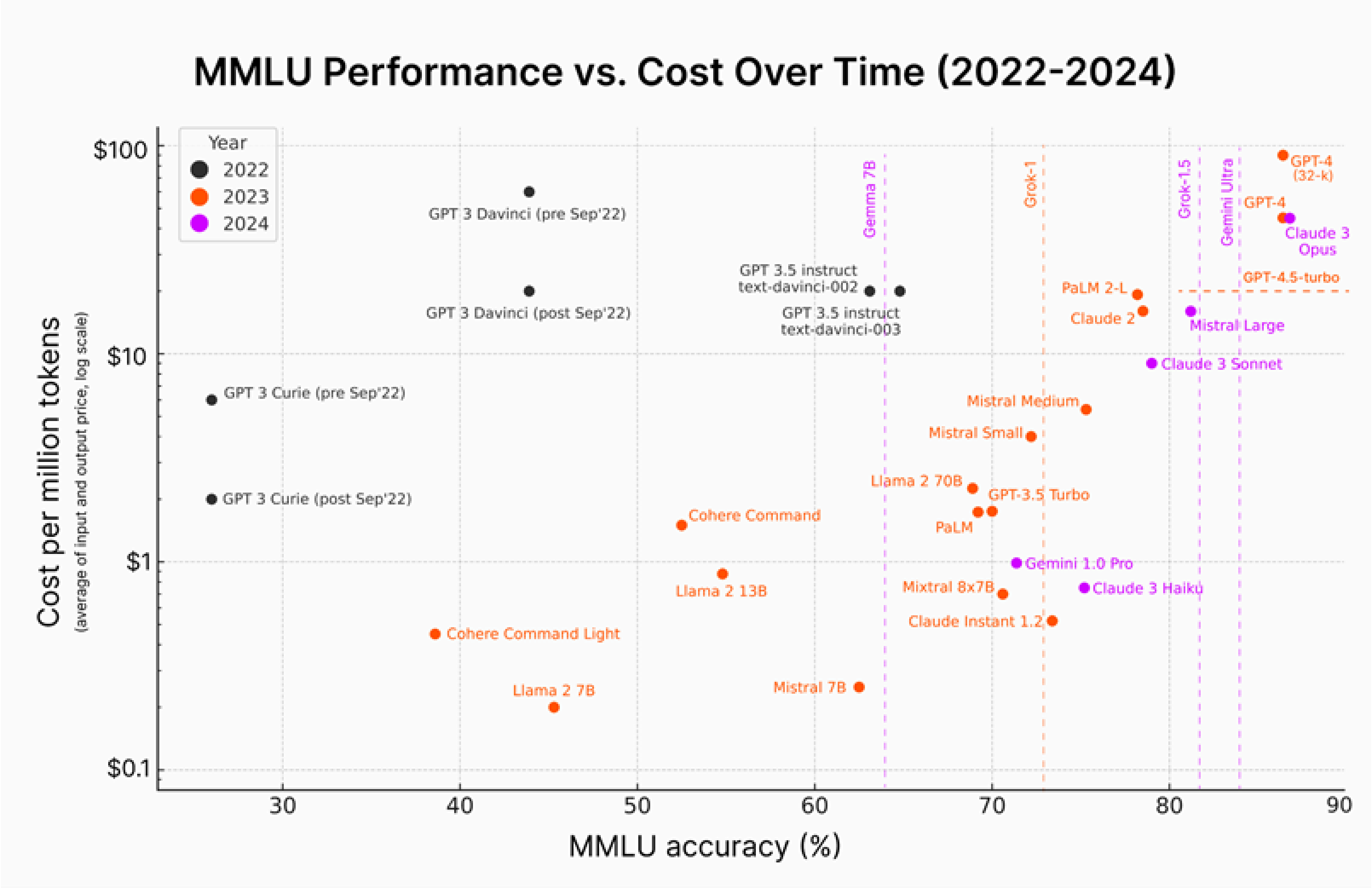


Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014

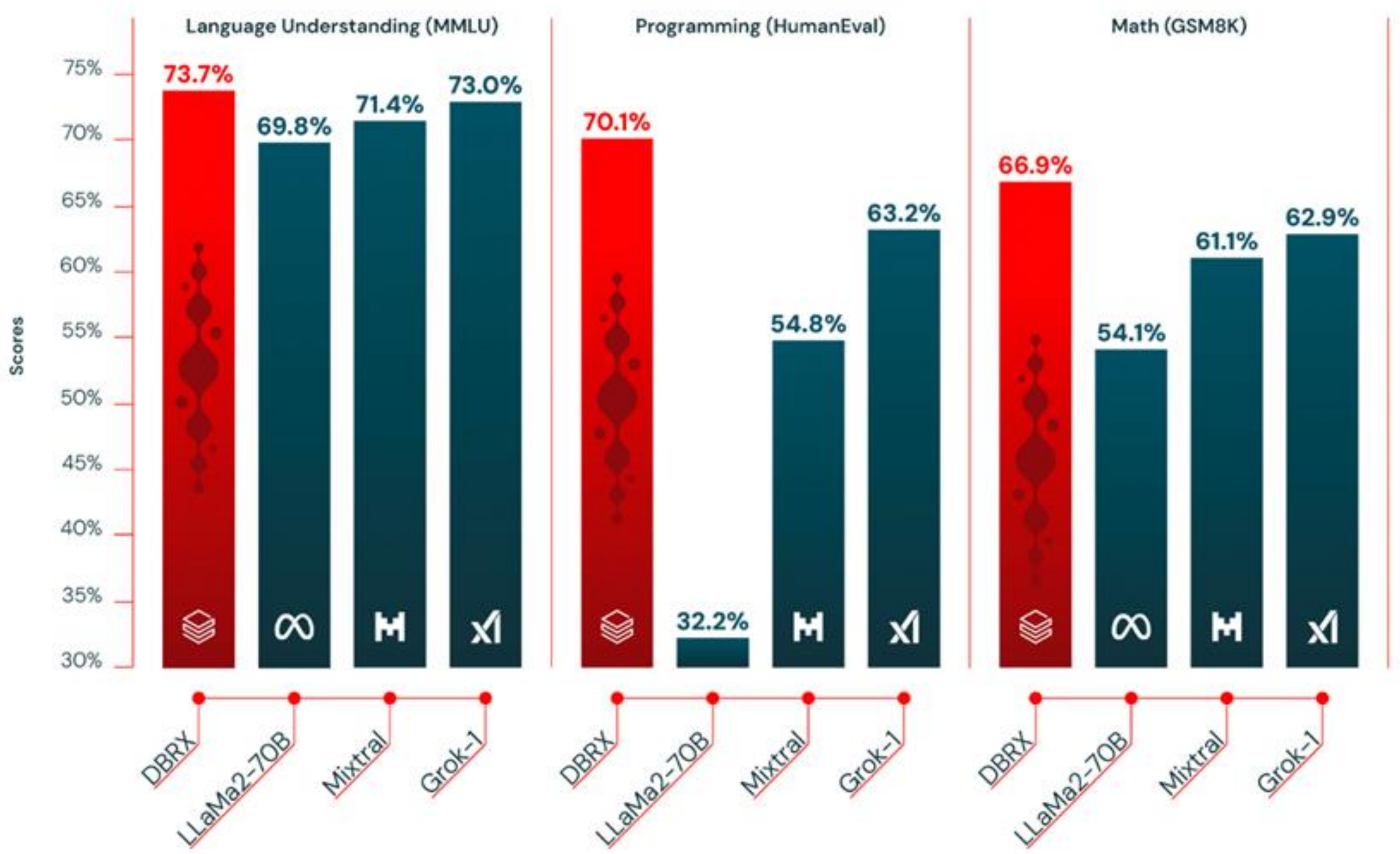
Model Size


1,000,000,000,000
100,000,000,000
10,000,000,000
1,000,000,000
100,000,000
10,000,000
Model Size (Parameters)
2017
2018
2019
2020
2021
2022
2023
2024
Model Introduction
MT NLG 530B
PaLM
PaLM2
GPT-3
GPT-3.5
OPT
LaMDA
BLOOM
LLaMA2
Chinchilla
LLaMA
Microsoft T-NLG
Megatron-NLG
GPT-J 6B
GPT-2
BERT Large
GPT-1
ELMo
Transformer

Model Cost


MMLU Performance vs. Cost Over Time (2022-2024)
Year
2022
2023
2024
$100
$10
$1
$0.1
Cost per million tokens
(average of input and output price, log scale)
30
40
50
60
70
80
90
MMLU accuracy (%)
GPT 3 Davinci (pre Sep'22)
GPT 3 Davinci (post Sep'22)
GPT 3 Curie (pre Sep'22)
GPT 3 Curie (post Sep'22)
GPT 3.5 instruct text-davinci-002
GPT 3.5 instruct text-davinci-003
Gemma 7B
Grok-1
Grok-1.5
Gemini Ultra
GPT-4 (32-k)
GPT-4
Claude 3 Opus
GPT-4.5-turbo
PaLM 2-L
Claude 2
Mistral Large
Claude 3 Sonnet
Mistral Medium
Mistral Small
Llama 2 70B
GPT-3.5 Turbo
PaLM
Cohere Command
Gemini 1.0 Pro
Mixtral 8x7B
Claude 3 Haiku
Llama 2 13B
Claude Instant 1.2
Cohere Command Light
Llama 2 7B
Mistral 7B

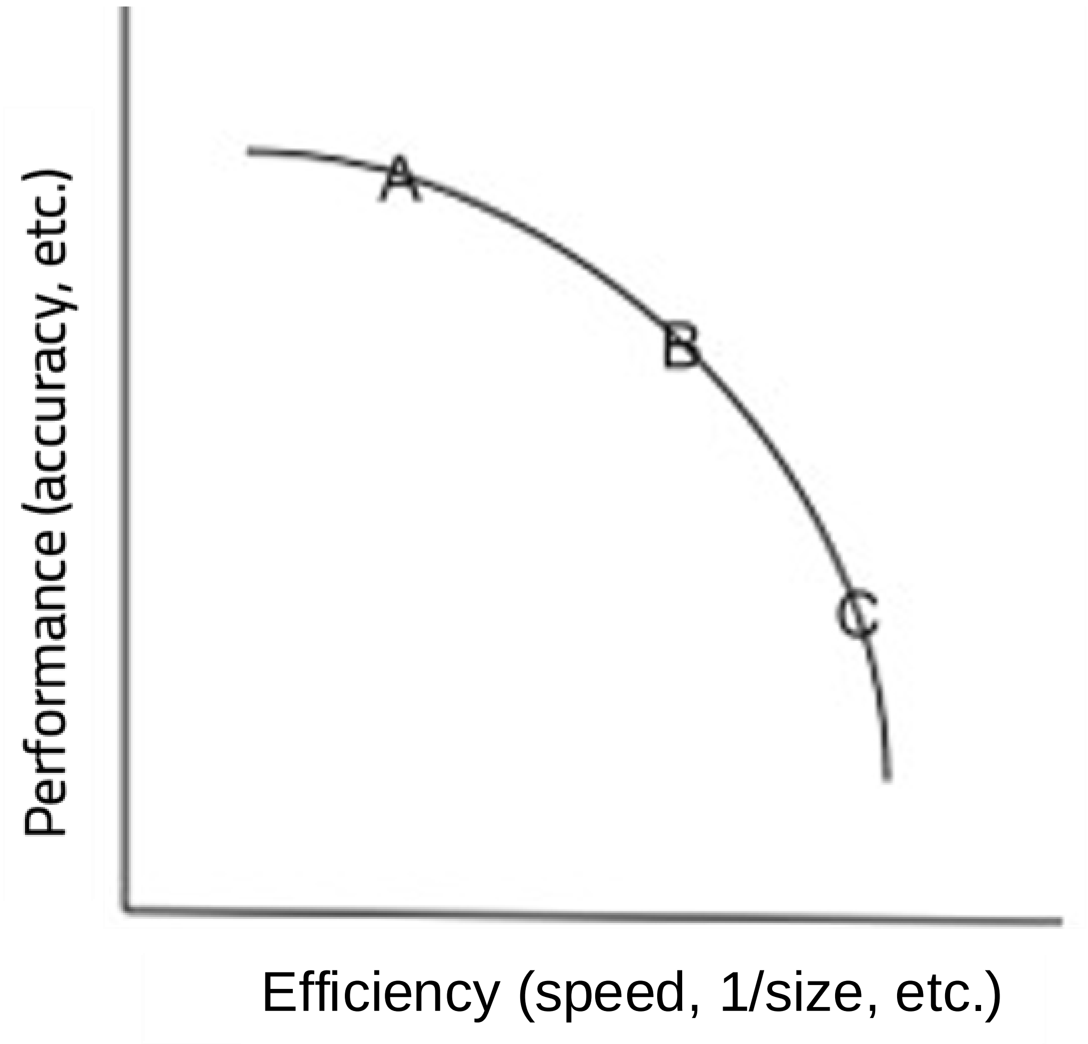
Development Speed



https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm

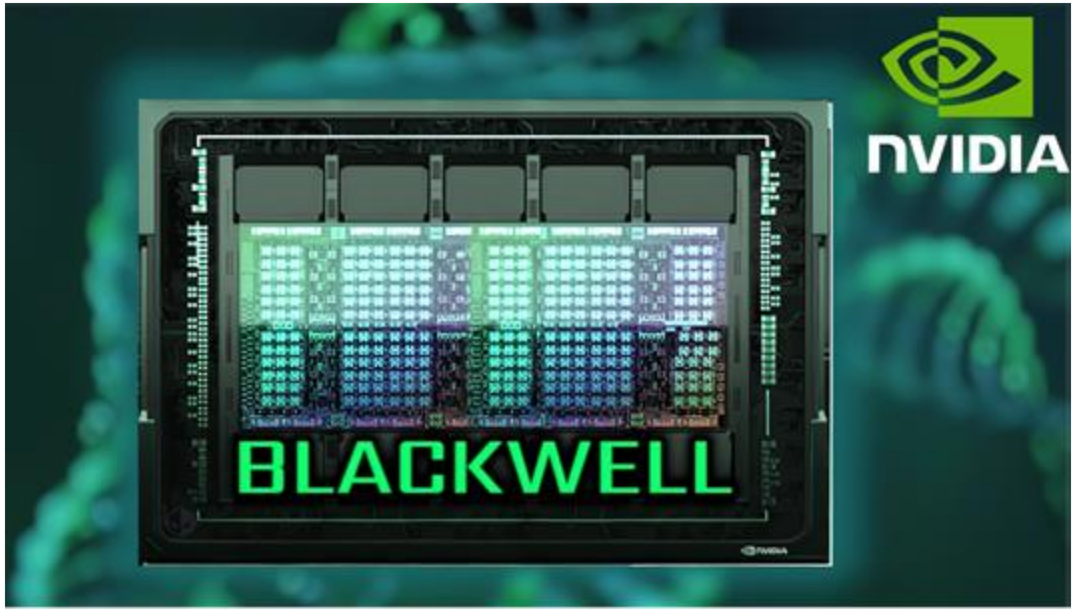
Efficiency Tradeoff

- More efficient models (smaller, faster) typically come at a cost of some performance of the model itself
- In the other direction, getting more performance from a model architecture likely means it will be larger, and require more computation

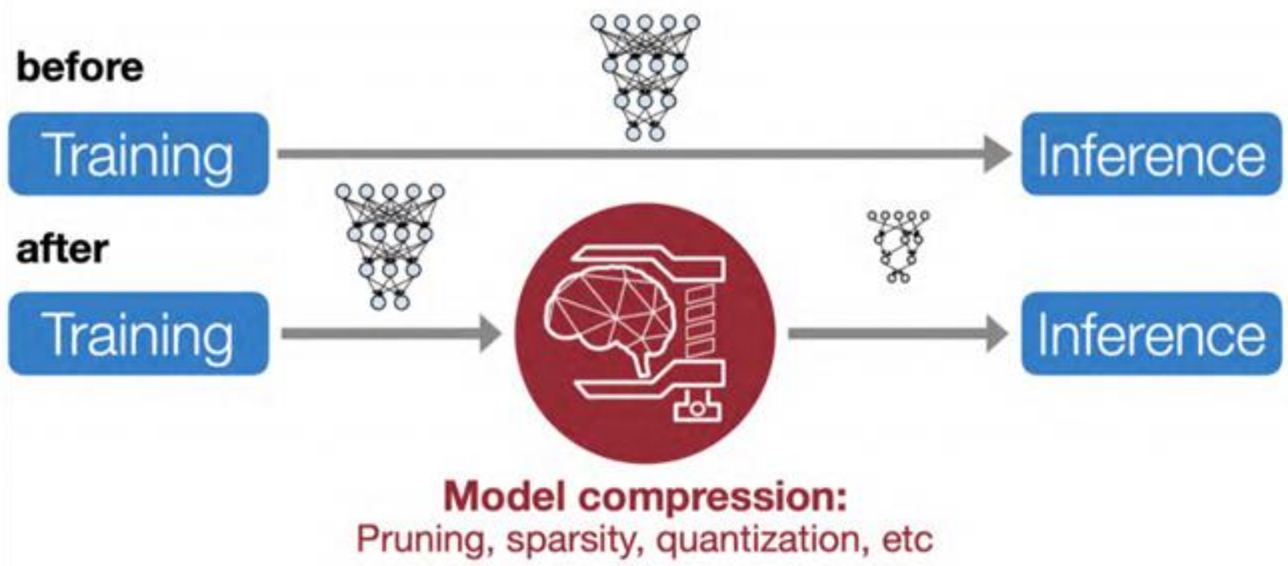


How to Improve Model Efficiency?

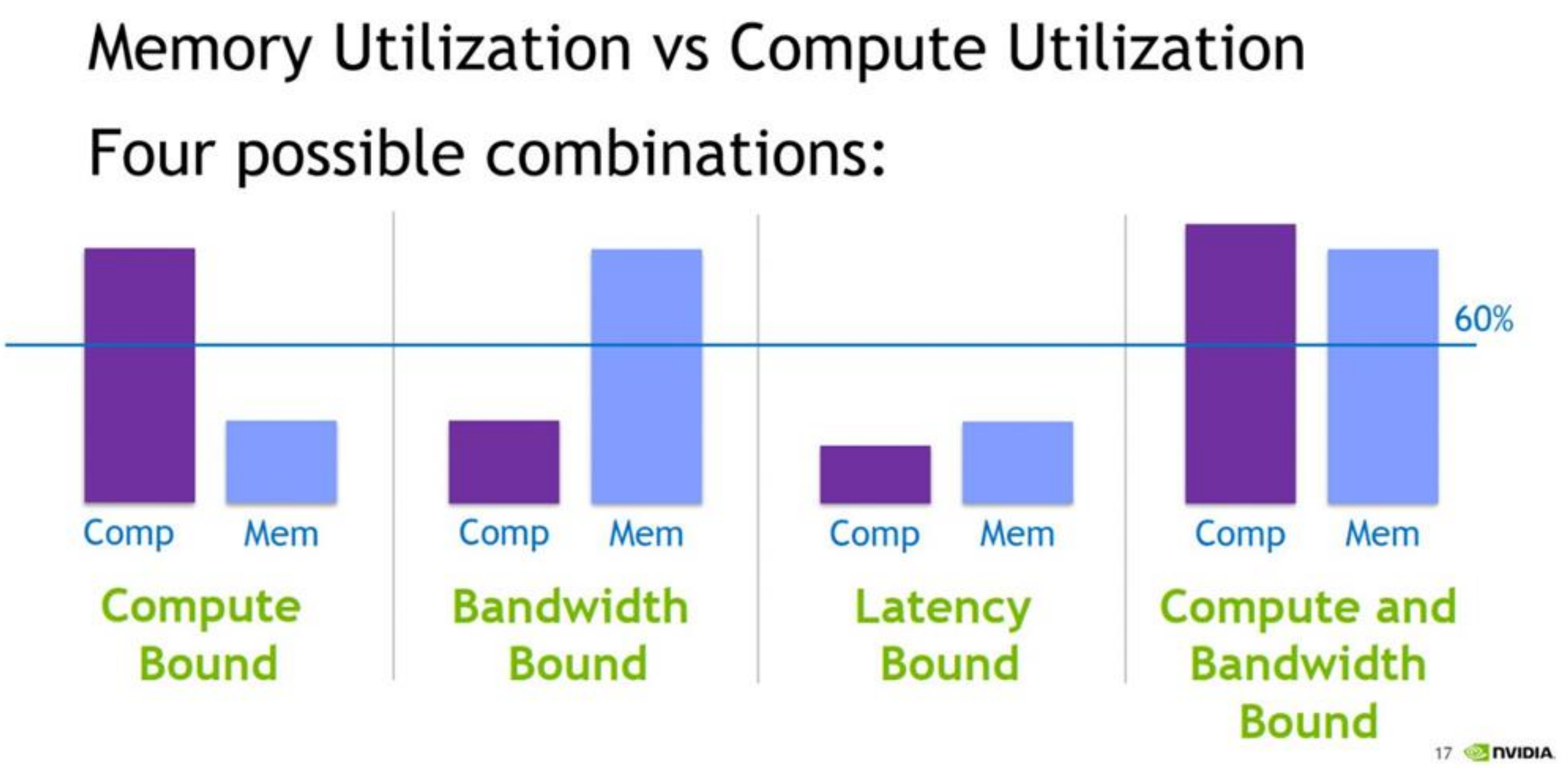
Hardware



Software



What Makes a Language Model Slow



Efficient LLMs

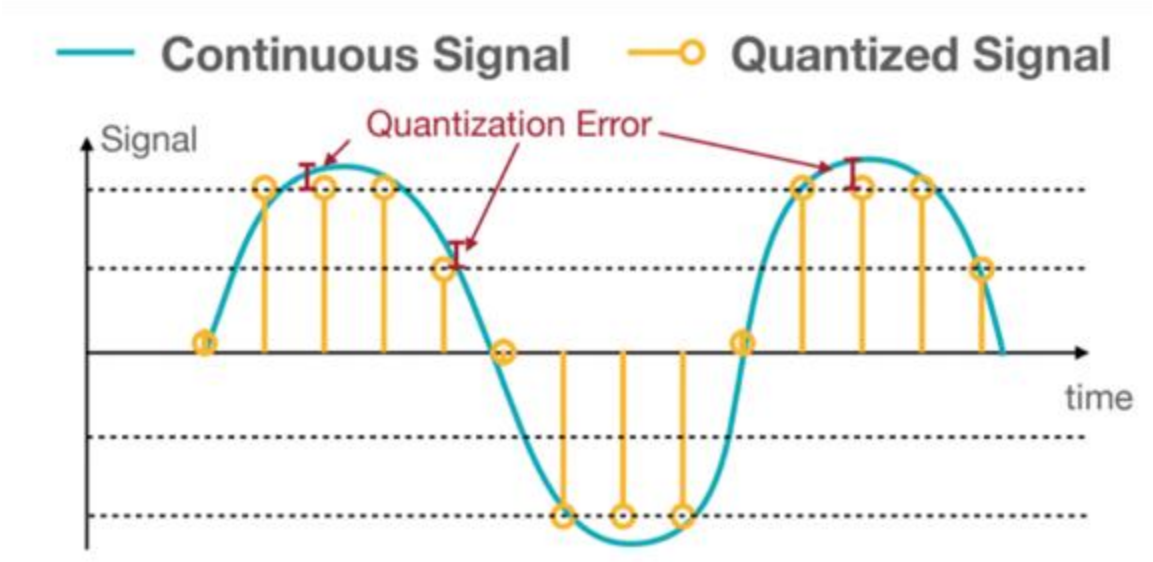
- Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

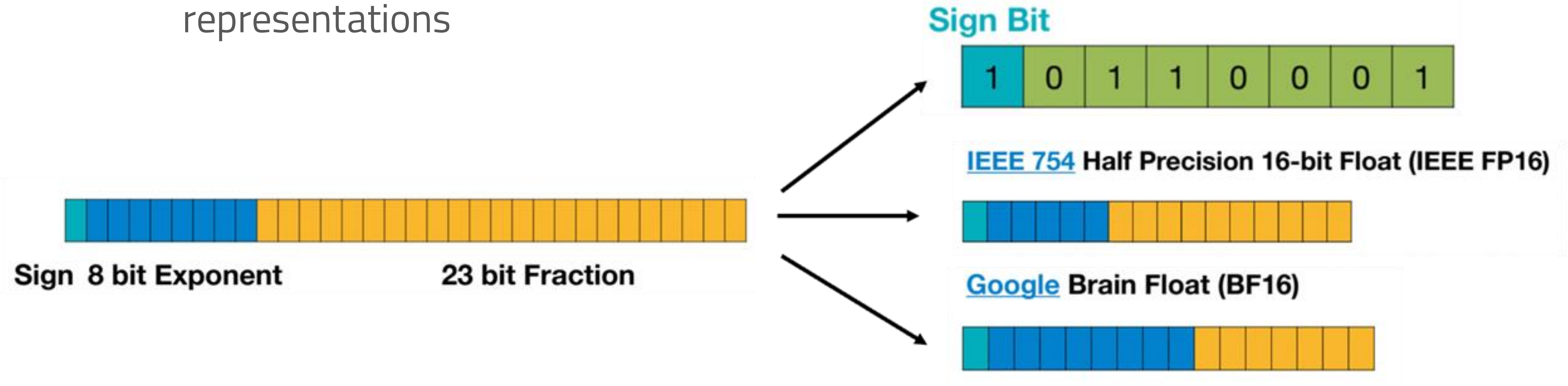
Efficient LLMs

- **Quantization**
  - **Background**
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

Quantization

Reduce model size by replacing high bit-width representations with low bit-width representations

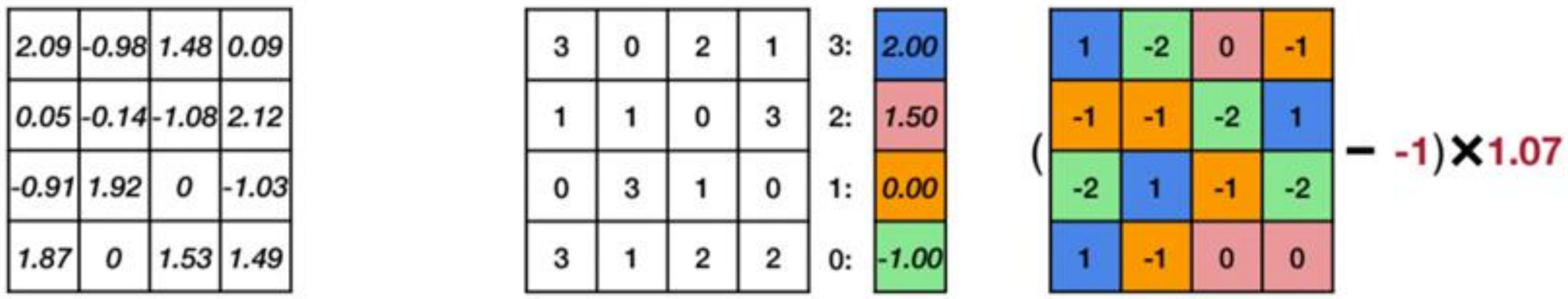




Efficient LLMs

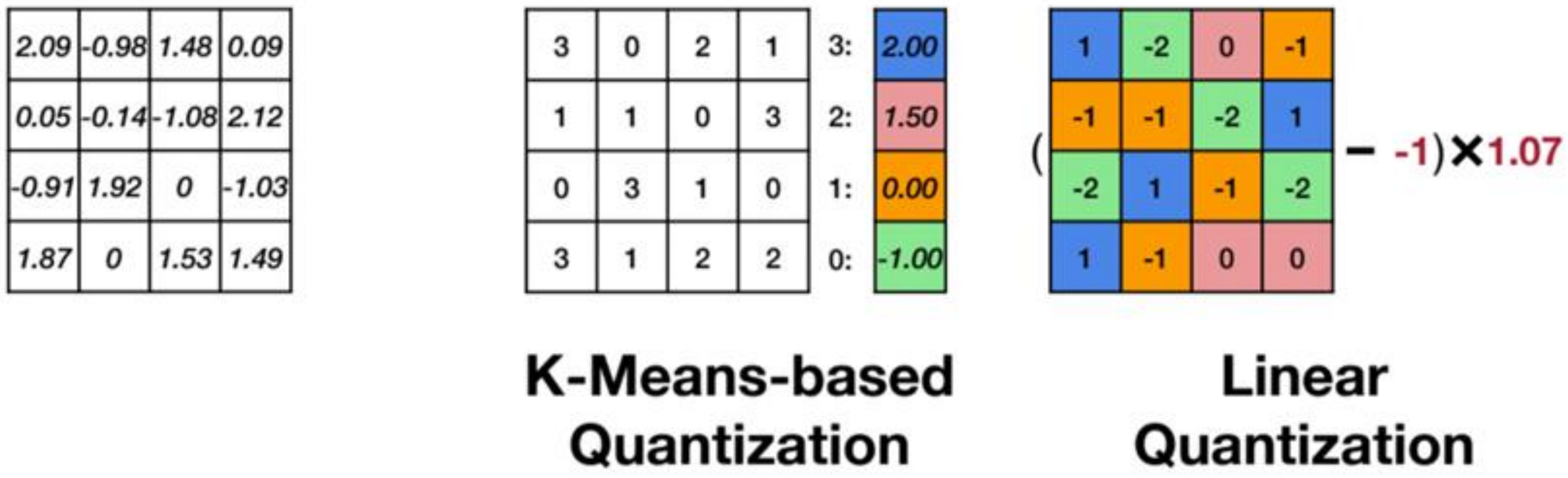
- **Quantization**
  - Background
  - **K-Means vs. Linear Quantization**
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

K-Means Quantization vs Linear Quantization



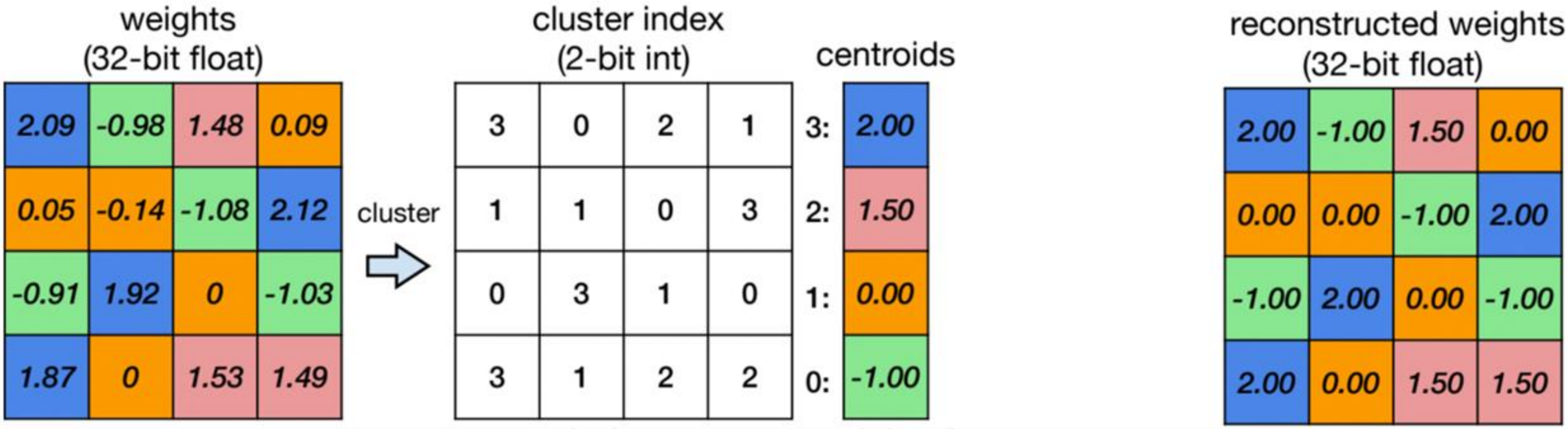
		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

K-Means Quantization vs Linear Quantization



		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

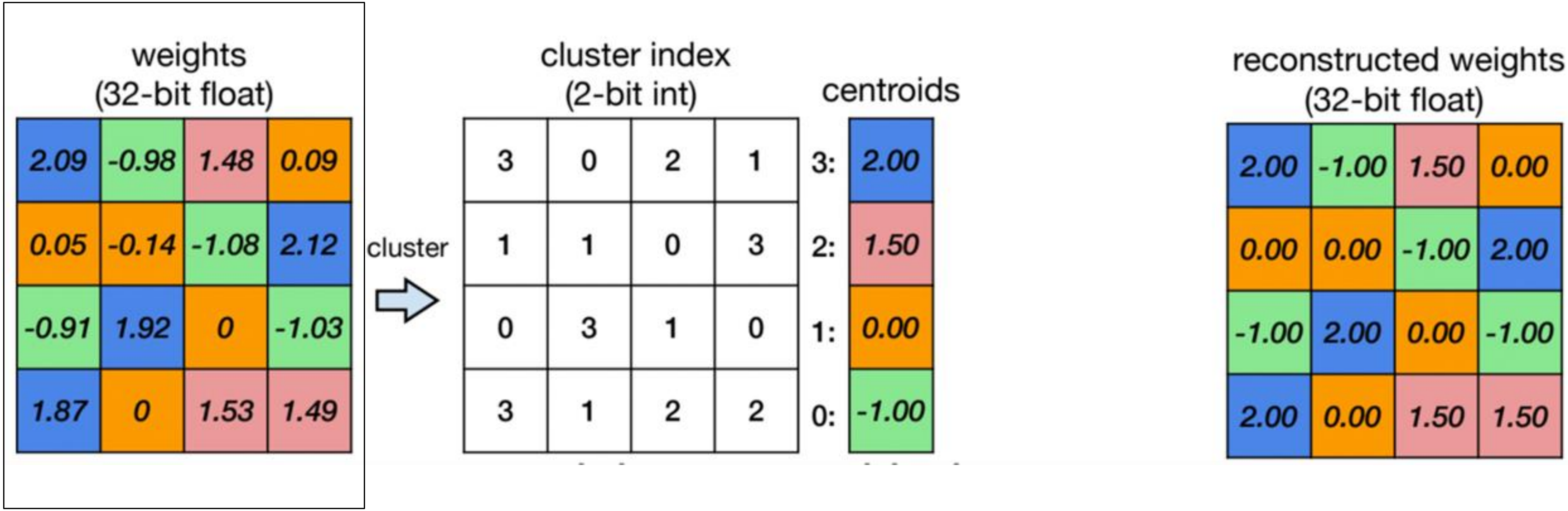
K-Means Quantization



Deep Compression [Han et al., ICLR 2016]

K-Means Quantization

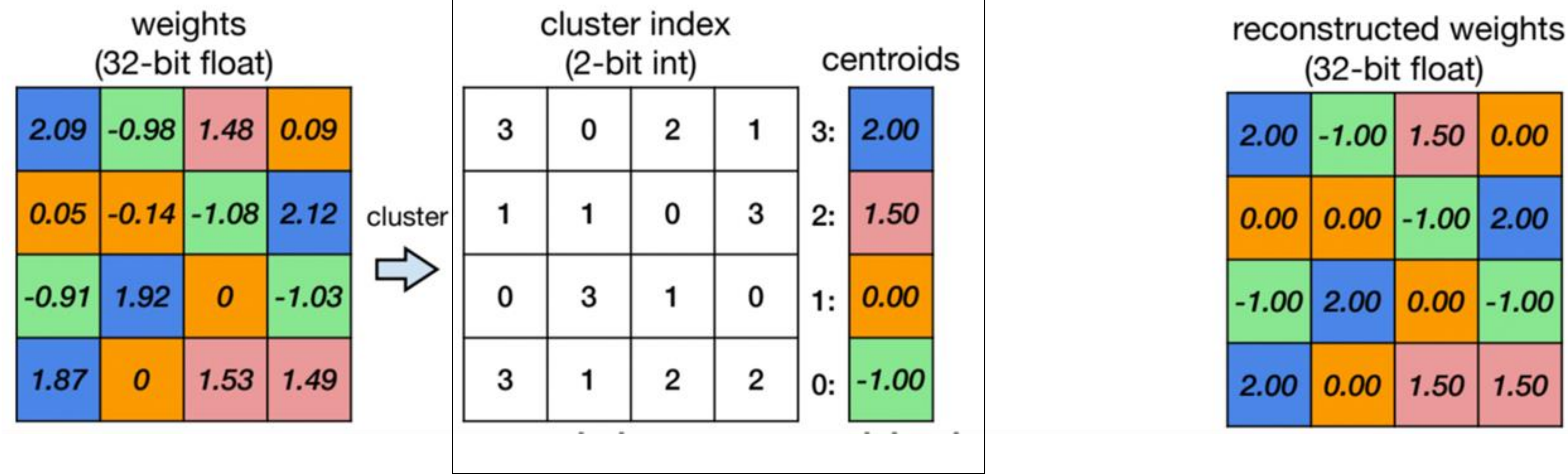
Original weights



Deep Compression [Han et al., ICLR 2016]

K-Means Quantization

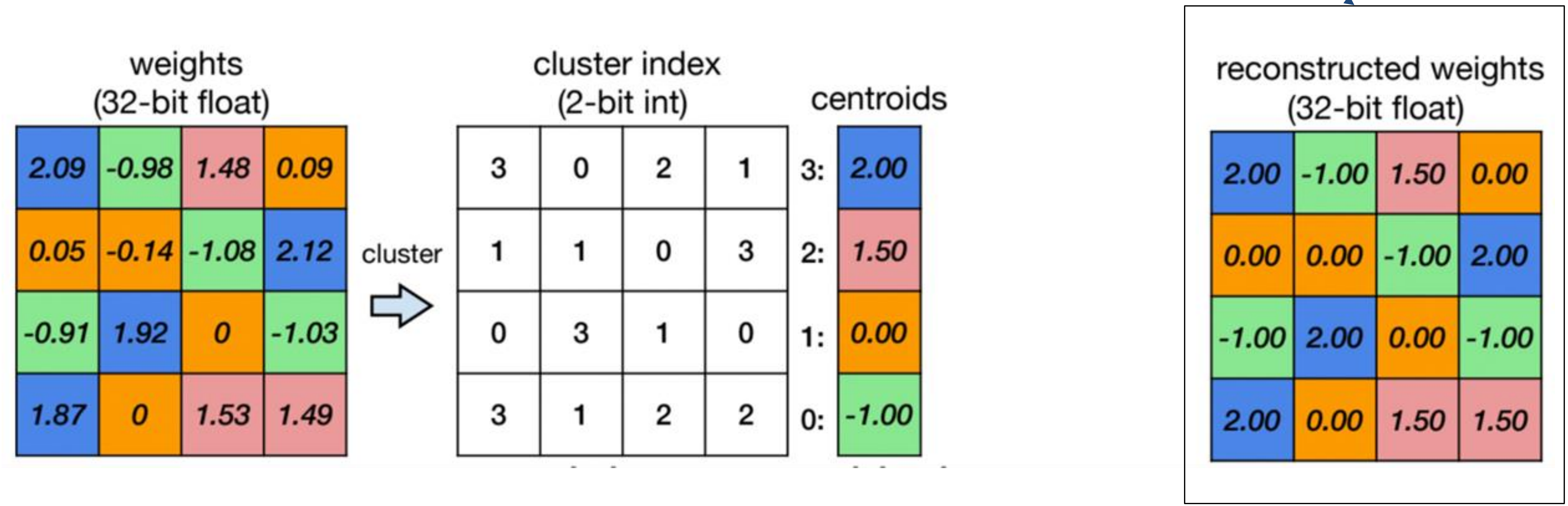
Stored weights after clustering



Deep Compression [Han et al., ICLR 2016]

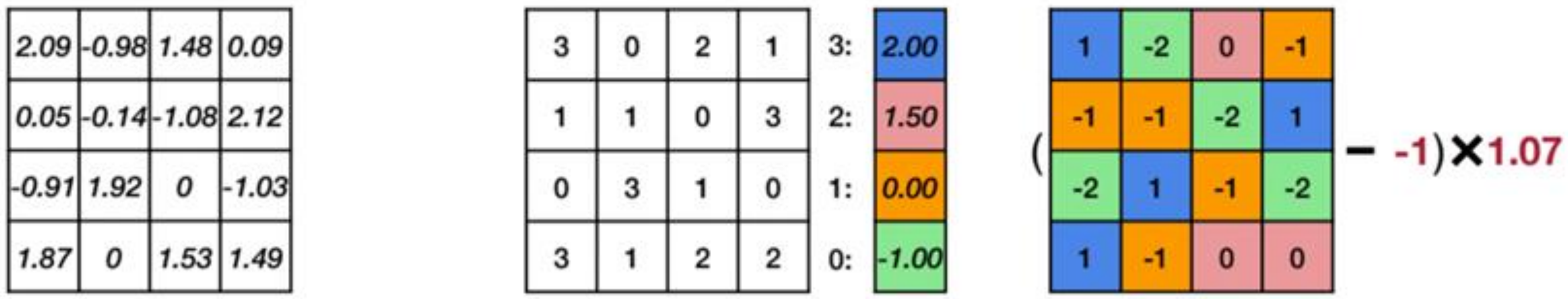
K-Means Quantization

Retrieved weights to be used at inference time



Deep Compression [Han et al., ICLR 2016]

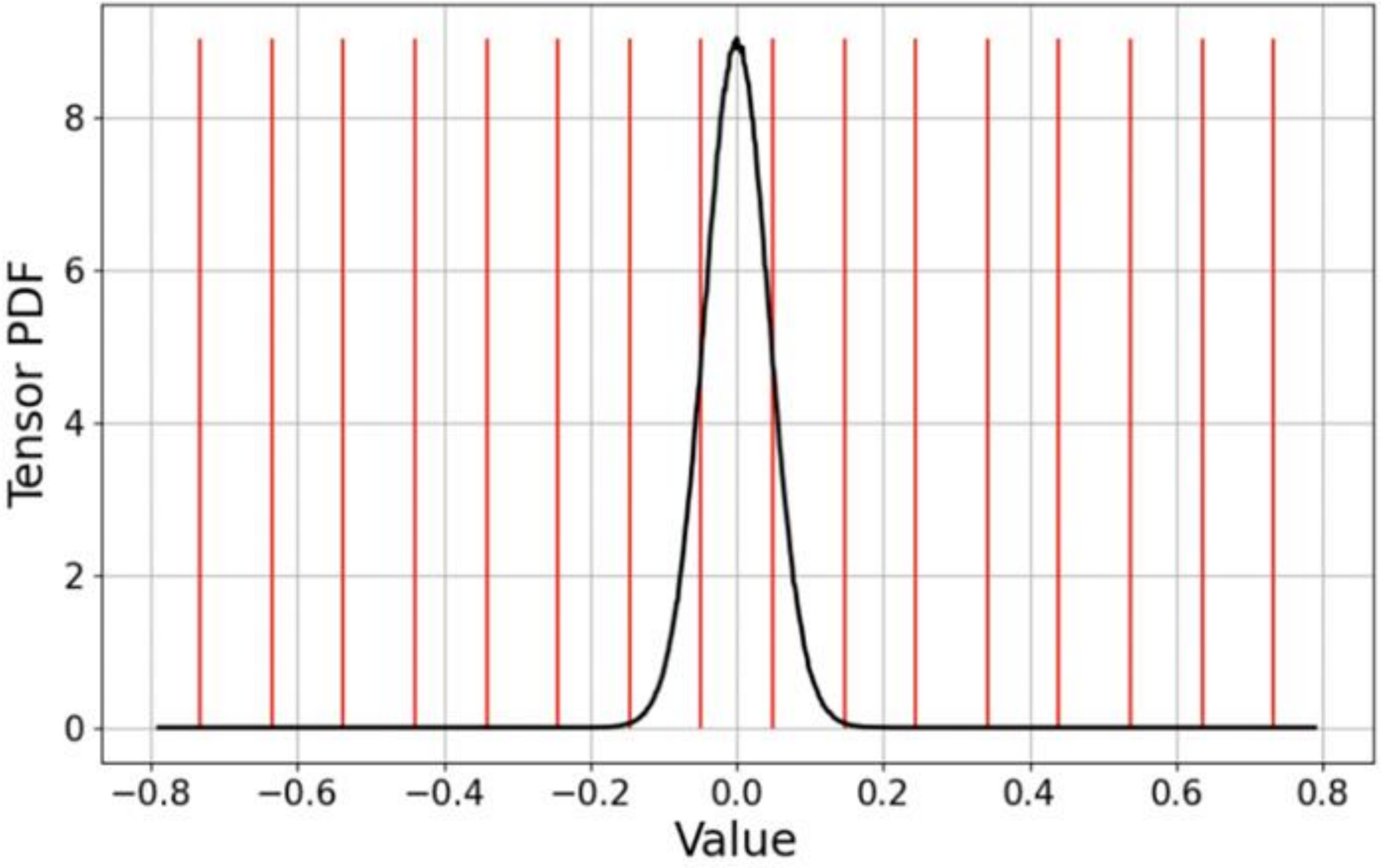
K-Means Quantization vs Linear Quantization



		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

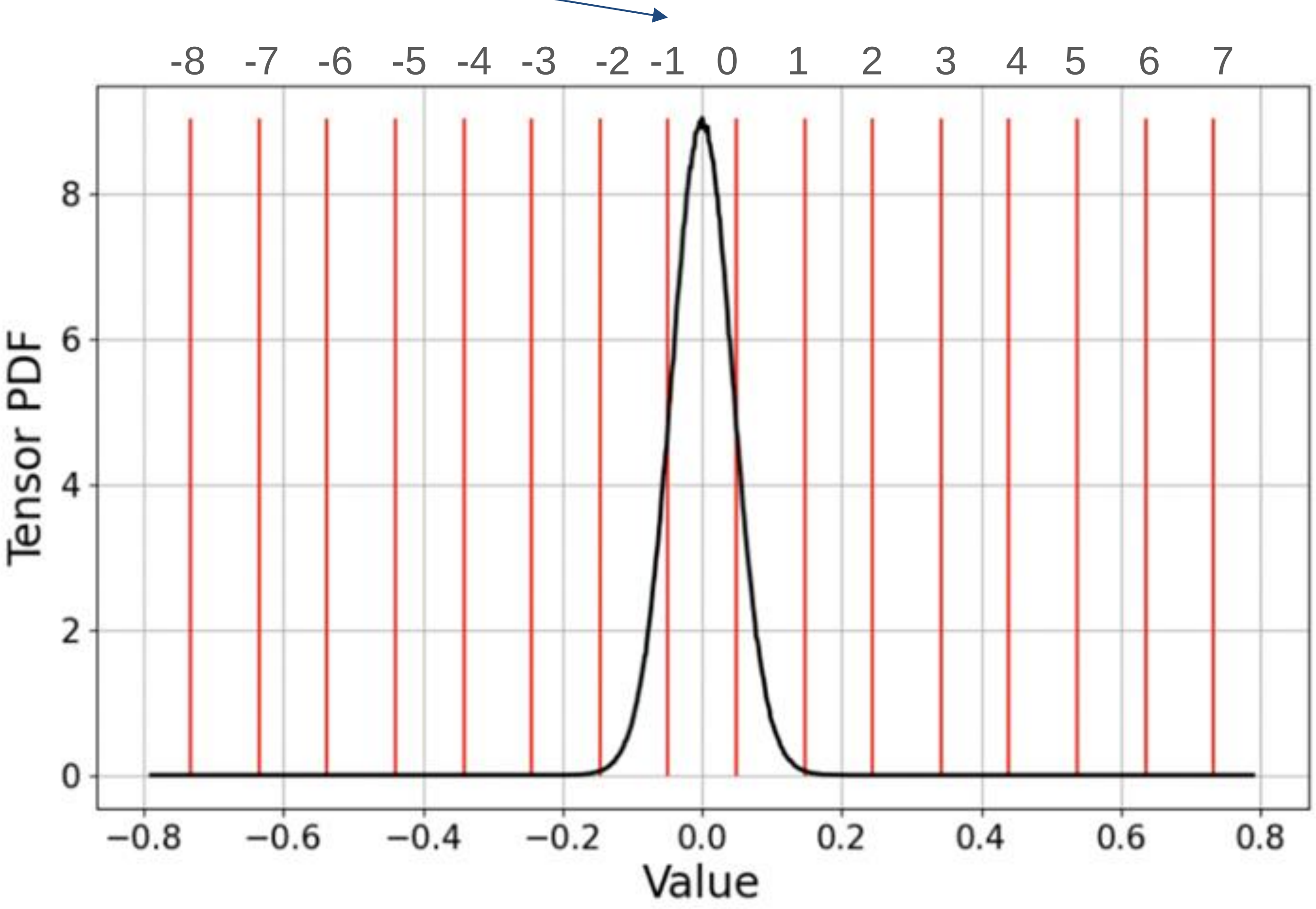
Linear Quantization

- Apply linear function on weights and hidden state activations from floating point values (r) to integer values (q)
- Original weights (black), Quantized bins (red)
- Black weights are mapped to one of the vertical red lines



Linear Quantization

32-bit float to 4-bit int

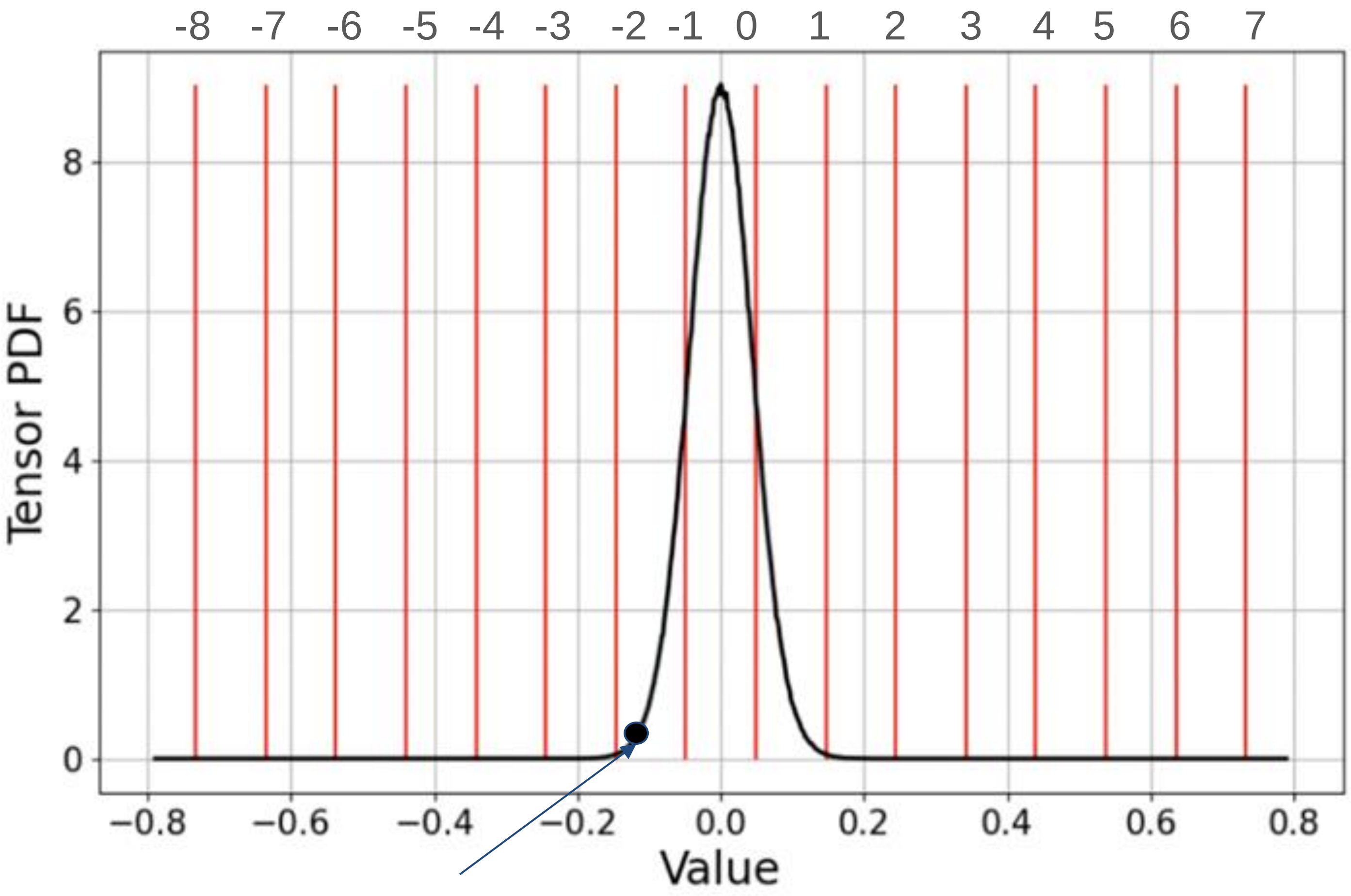


- ❑ Apply linear function on weights and hidden state activations from floating point values (r) to integer values (q)
- ❑ Original weights (black), Quantized bins (red)
- ❑ Black weights are mapped to one of the vertical red lines

Linear Quantization

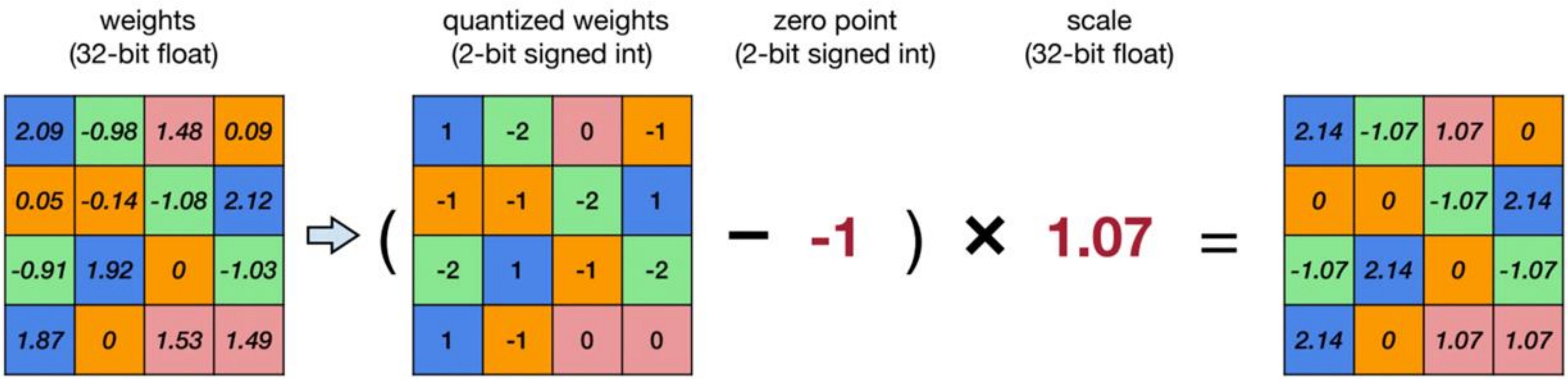
- Apply linear function on weights and hidden state activations from floating point values (r) to integer values (q)
- Original weights (black), Quantized bins (red)
- Black weights are mapped to one of the vertical red lines

32-bit float to 4-bit int



Before Quantization: -.14
After Quantization: -2

Linear Quantization



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

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacobet al., CVPR 2018]

Linear Quantization

weights (32-bit float) | quantized weights (2-bit signed int) | zero point (2-bit signed int) | scale (32-bit float)

$$
\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}
\Rightarrow
\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - (-1) \right) \times 1.07 =
\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}
$$

Original Weights

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

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacobet al., CVPR 2018]

Linear Quantization

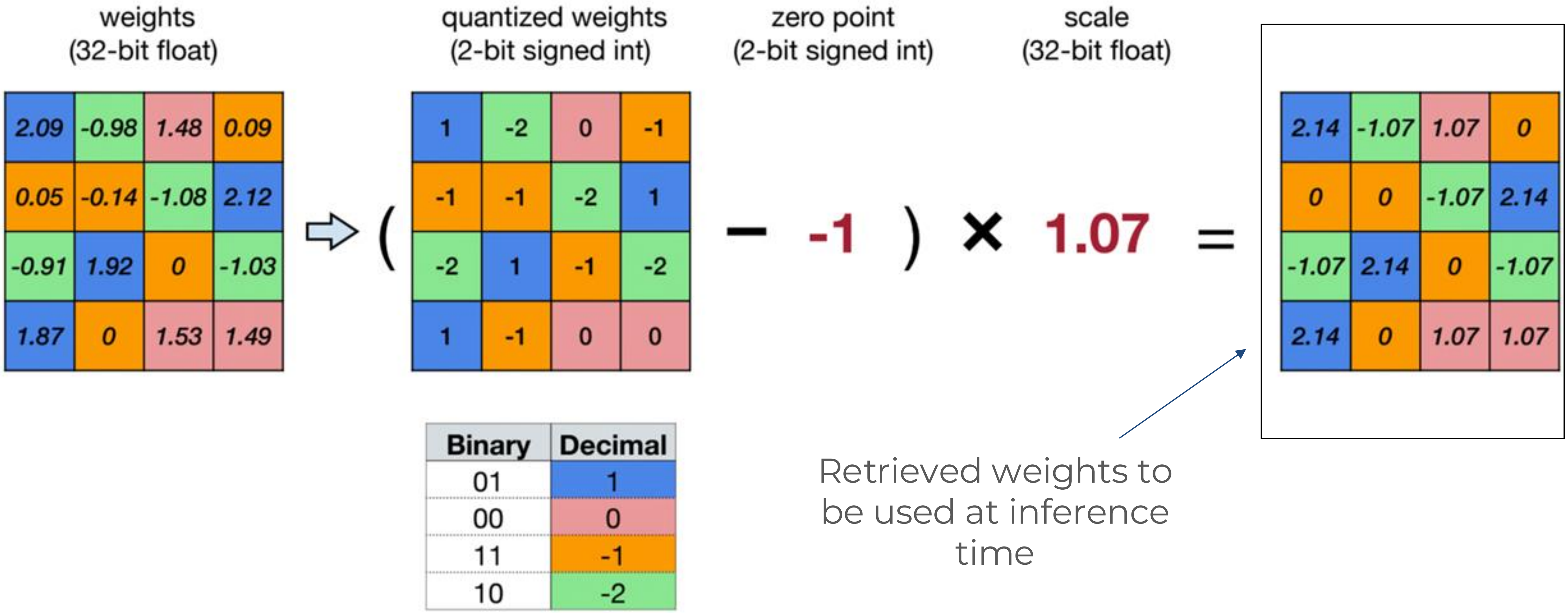
$$\overset{\text{weights (32-bit float)}}{\begin{bmatrix}2.09&-0.98&1.48&0.09\\0.05&-0.14&-1.08&2.12\\-0.91&1.92&0&-1.03\\1.87&0&1.53&1.49\end{bmatrix}} \Rightarrow \left(\overset{\text{quantized weights (2-bit signed int)}}{\boxed{\begin{bmatrix}1&-2&0&-1\\-1&-1&-2&1\\-2&1&-1&-2\\1&-1&0&0\end{bmatrix}}} - \overset{\text{zero point (2-bit signed int)}}{\boxed{-1}}\right)\times\overset{\text{scale (32-bit float)}}{\boxed{1.07}} = \begin{bmatrix}2.14&-1.07&1.07&0\\0&0&-1.07&2.14\\-1.07&2.14&0&-1.07\\2.14&0&1.07&1.07\end{bmatrix}$$

Stored Values

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

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacobet al., CVPR 2018]

Linear Quantization



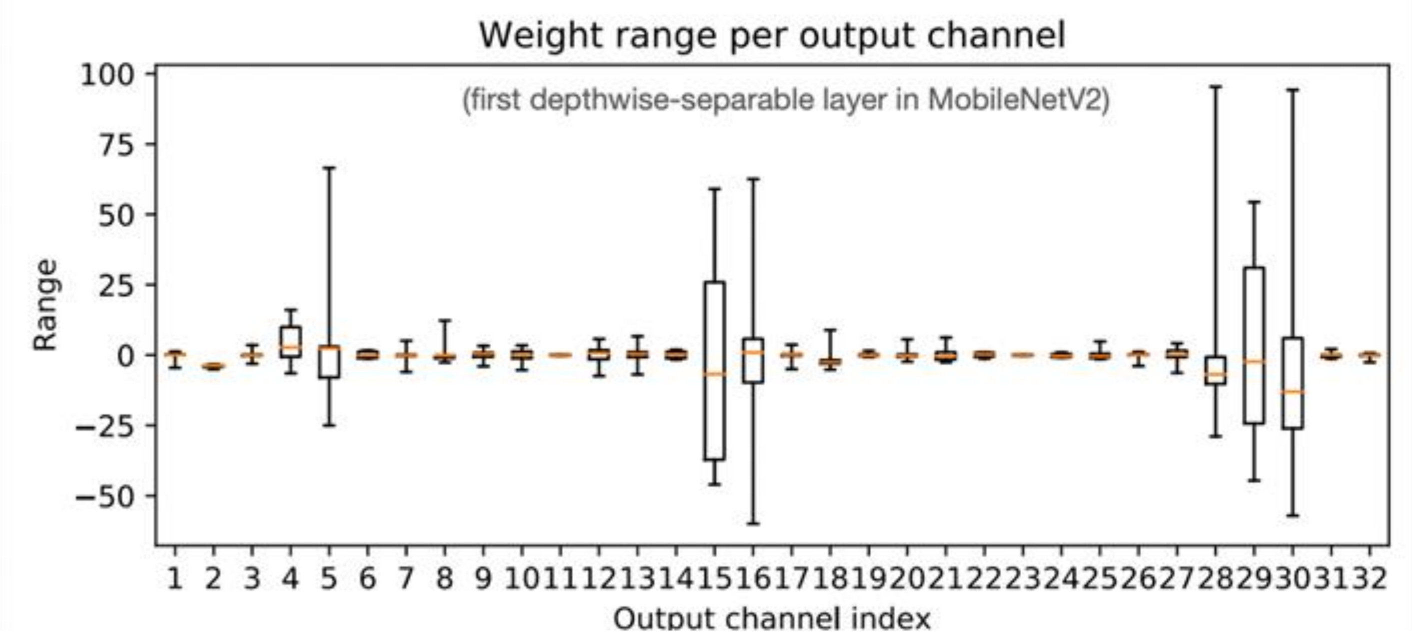
Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacobet al., CVPR 2018]

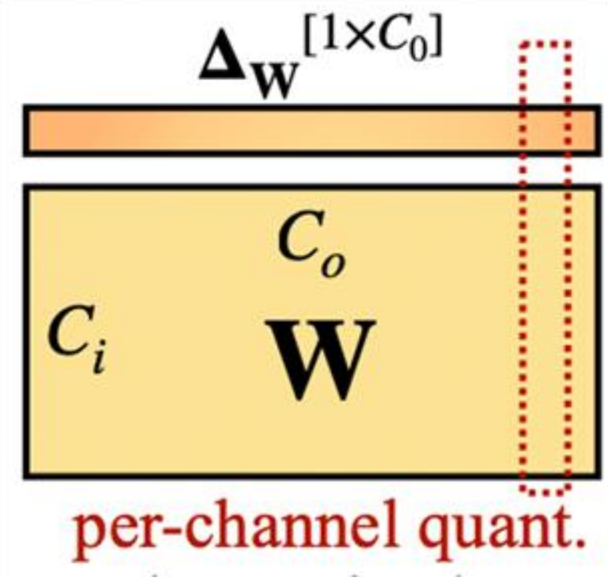
Efficient LLMs

- **Quantization**
  - Background
  - K-Means vs. Linear Quantization
  - **Quantization Granularity**
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

Weight Granularity

- ❑ Weight matrices will often have different variances along each output channel
- ❑ High variance in weights means that applying linear quantization will result in large performance degradation
- ❑ To fix this, we can perform linear quantization along each channel of the weight tensor separately

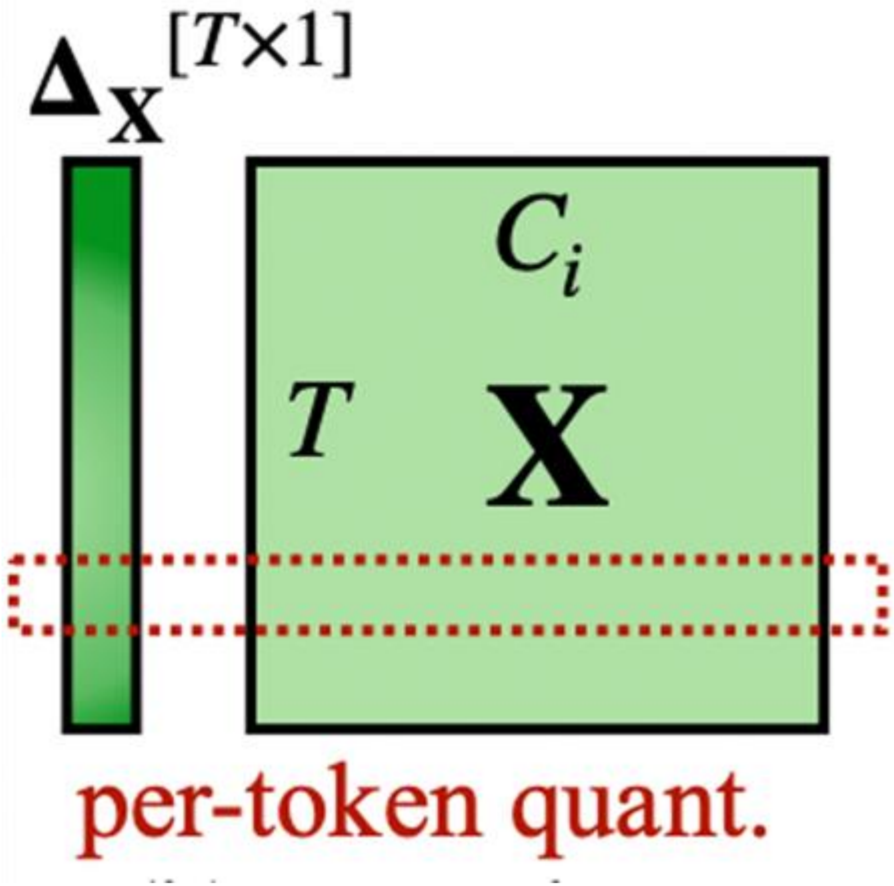




SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

Activation Granularity

- Activations can have a similar problem whereby the variance by channel can be quite different
- The variance by token can also differ dramatically
- When applying quantization, we should split up channels, tokens to take this into account



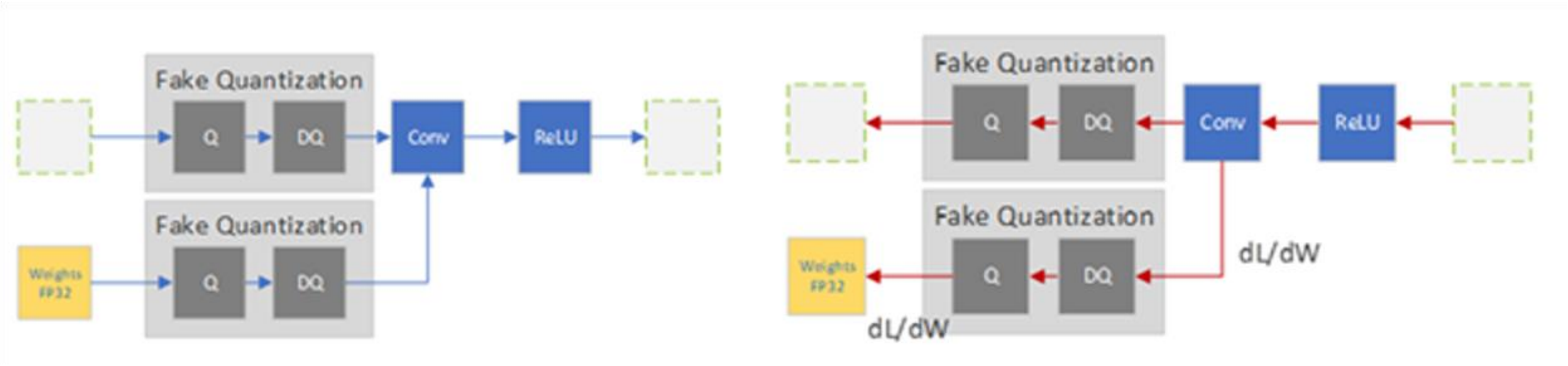
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

Efficient LLMs

- **Quantization**
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - **Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)**
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

Quantization Aware Training (QAT)

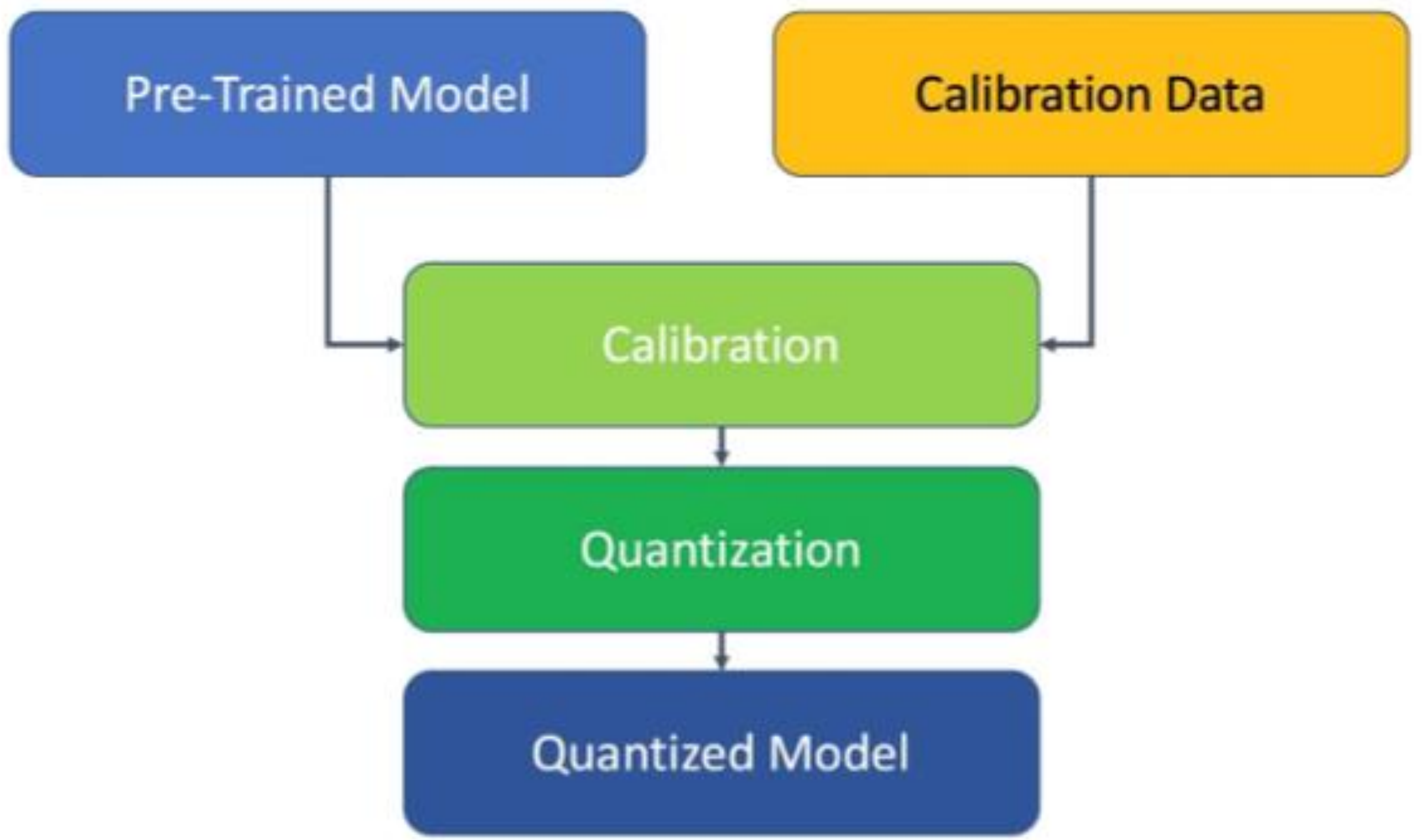
Quantize while training



https://pytorch.org/blog/quantization-in-practice/

Post Training Quantization (PTQ)

Quantize after training

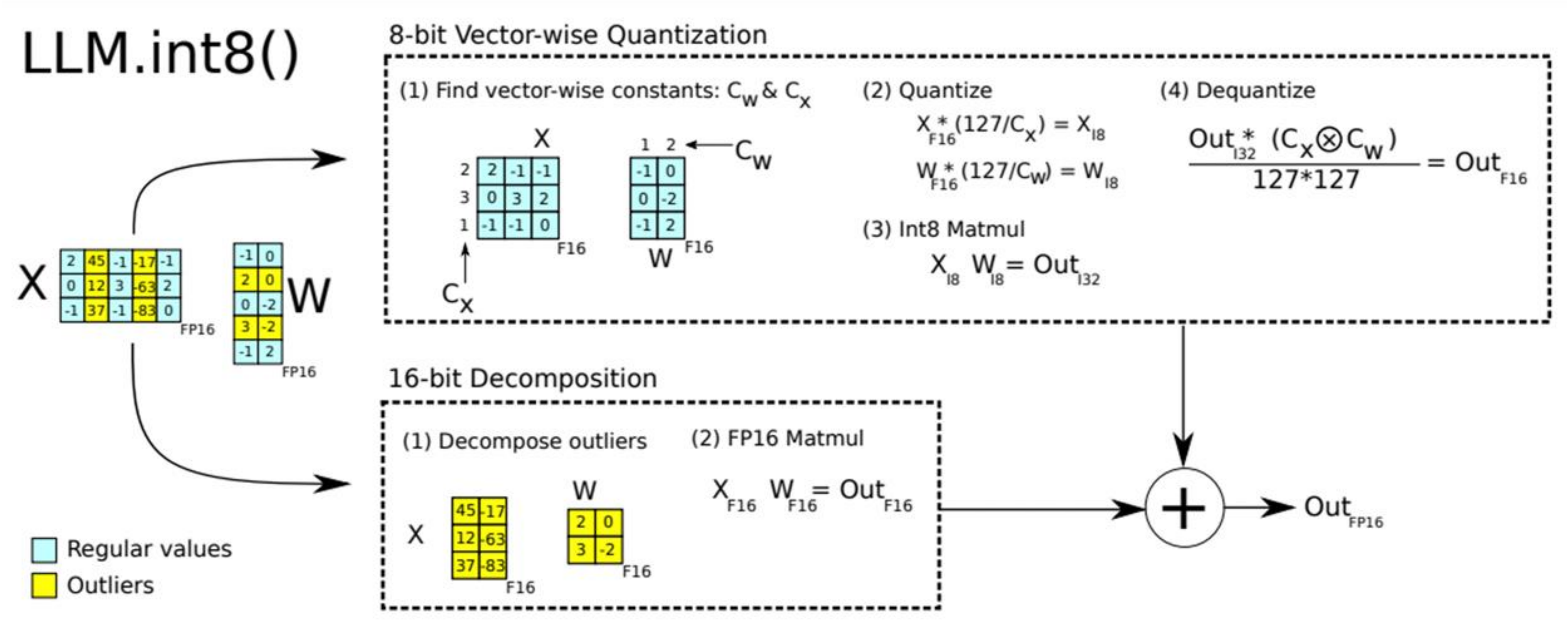


https://pytorch.org/blog/quantization-in-practice/

Efficient LLMs

- ❑ **Quantization**
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - **LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)**
- ❑ Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- ❑ Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- ❑ Speculative Decoding
- ❑ Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- ❑ Distillation

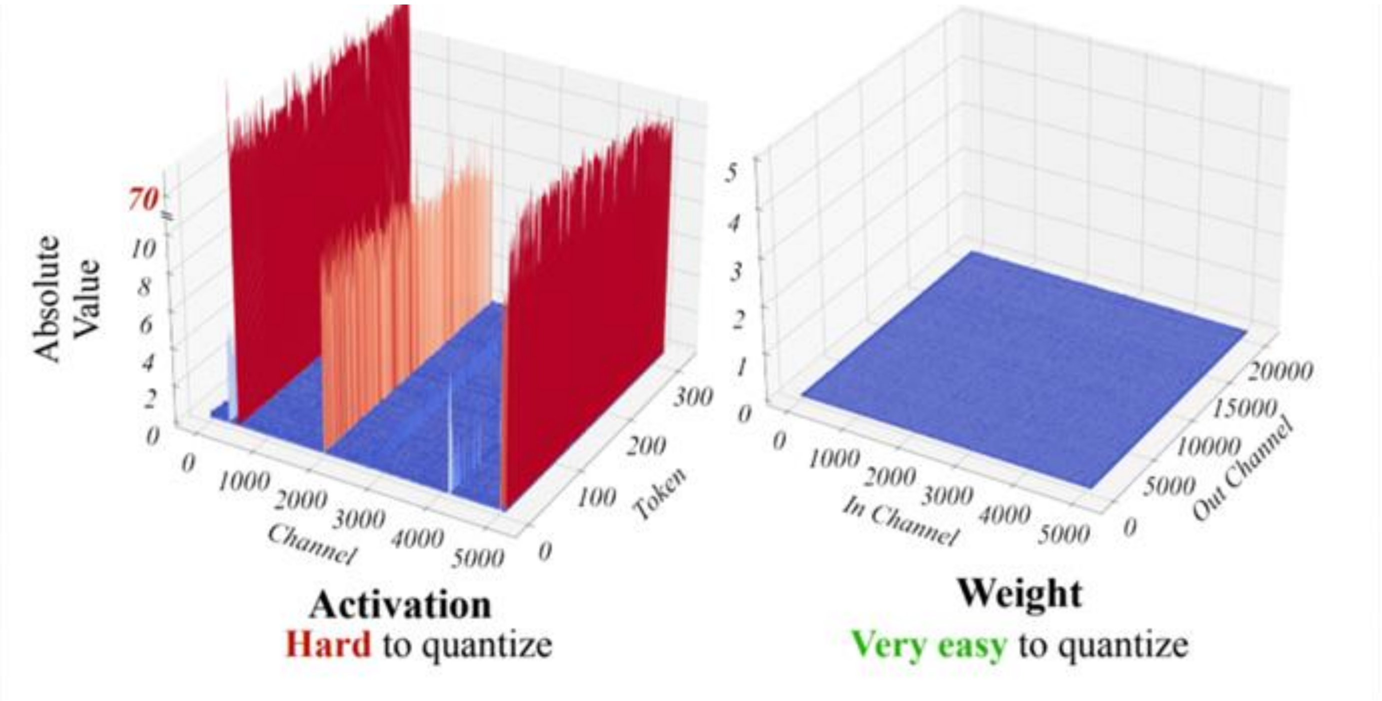
LLM.int8() (W8A8)



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale [Dettmers et. al., NeurIPS 2022]

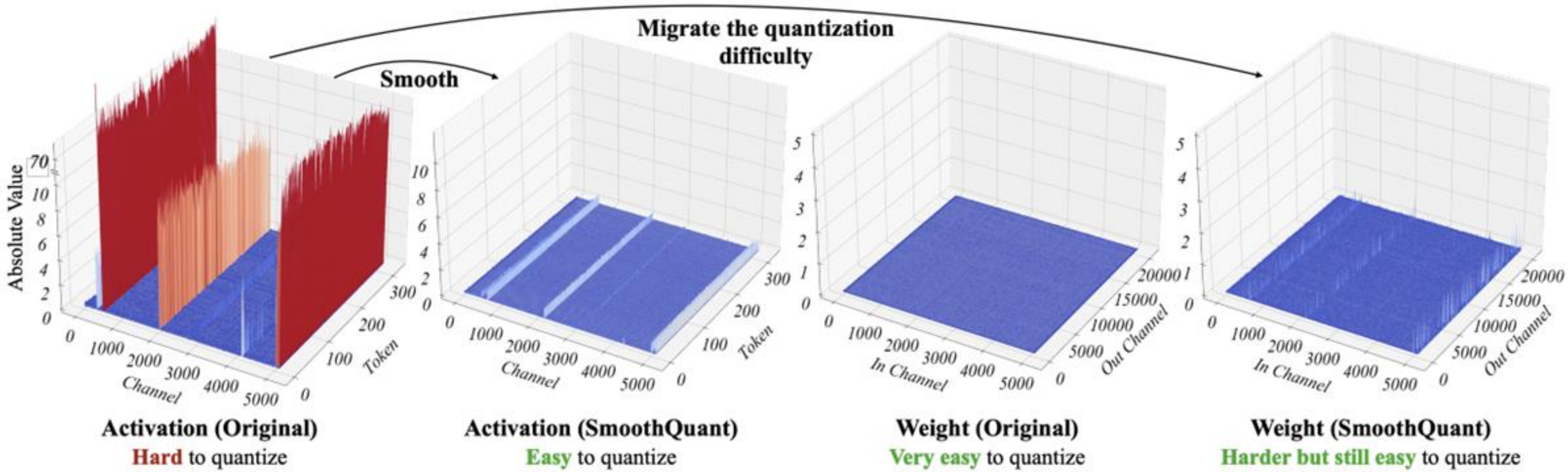
SmoothQuant (W8A8)

Observation: High variance channels are fixed in activations in LLM FFN layers-weights have relatively little difference in variance



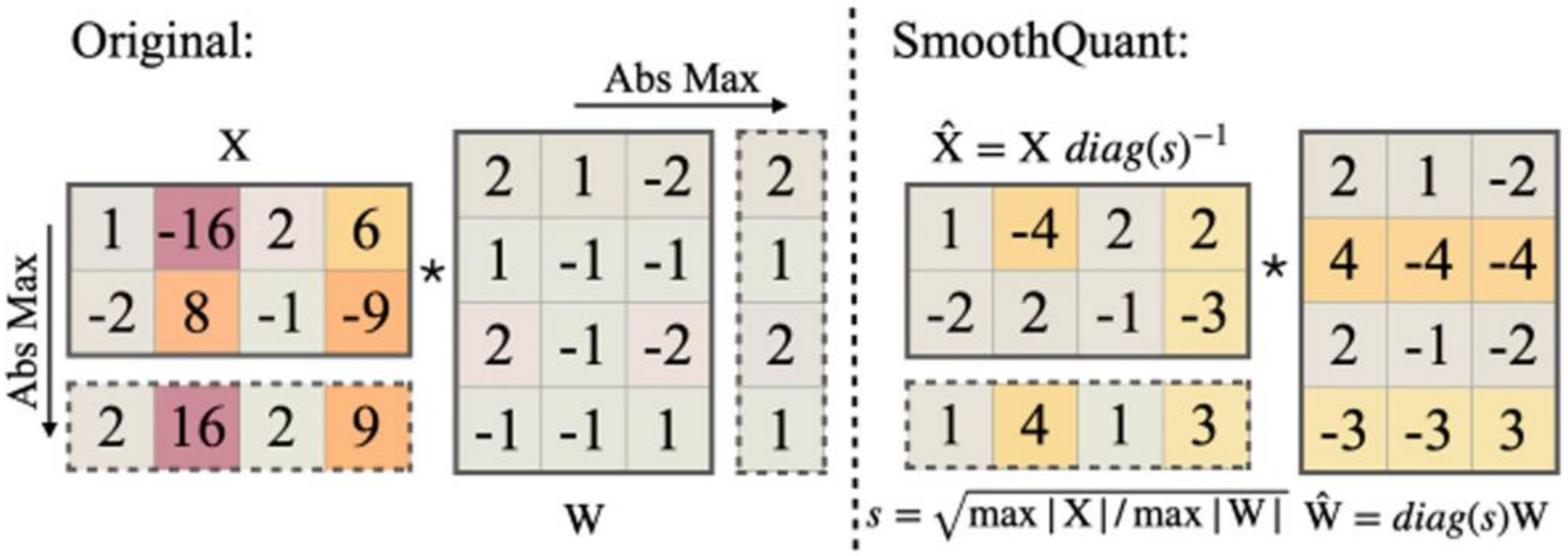
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

SmoothQuant (W8A8)



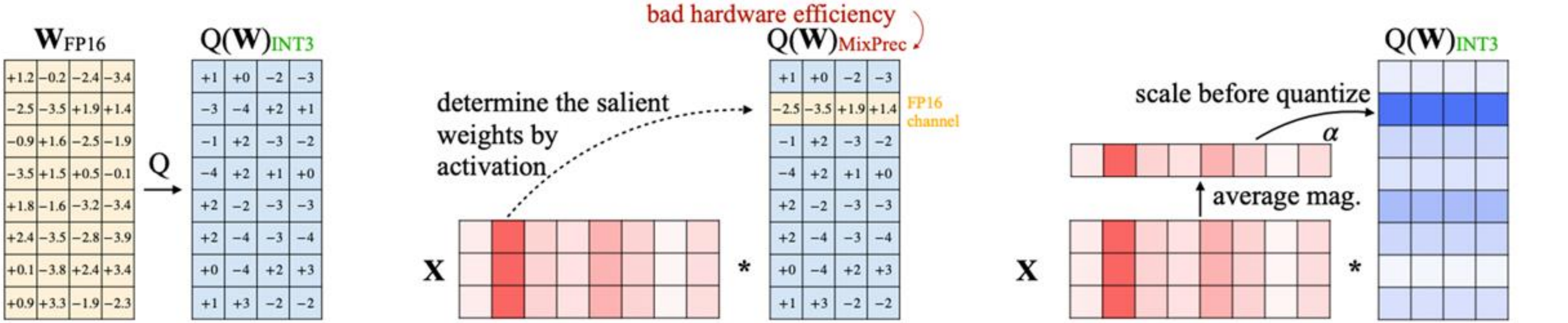
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

SmoothQuant (W8A8)



SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

AWQ (W4A16)

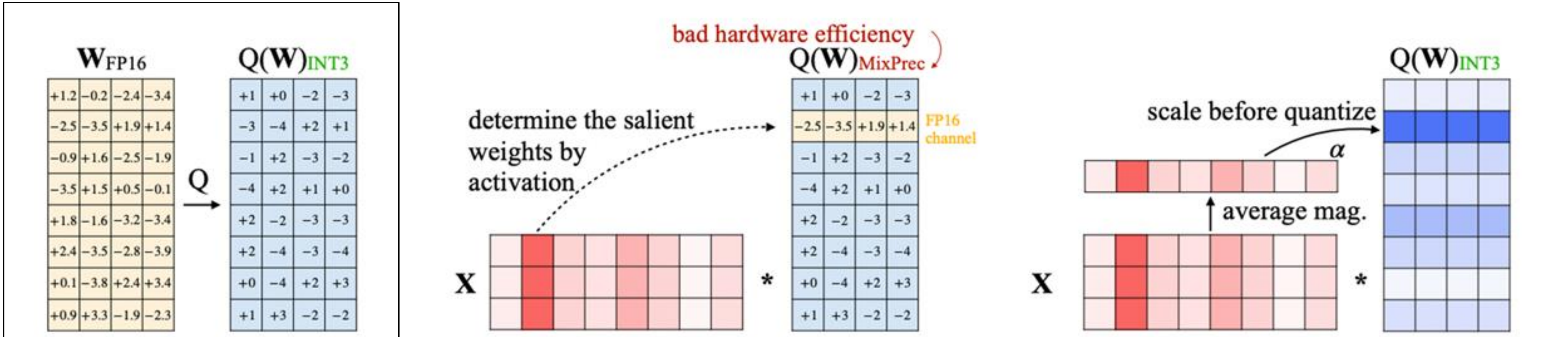


(a) RTN quantization (**PPL** 43.2) (b) Keep 1% salient weights in FP16 (**PPL** 13.0) (c) Scale the weights before quantization (**PPL** 13.0)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

AWQ (W4A16)

Normal quantization on LLMs performs poorly due to outliers in the model's hidden state



(a) RTN quantization (**PPL** 43.2)

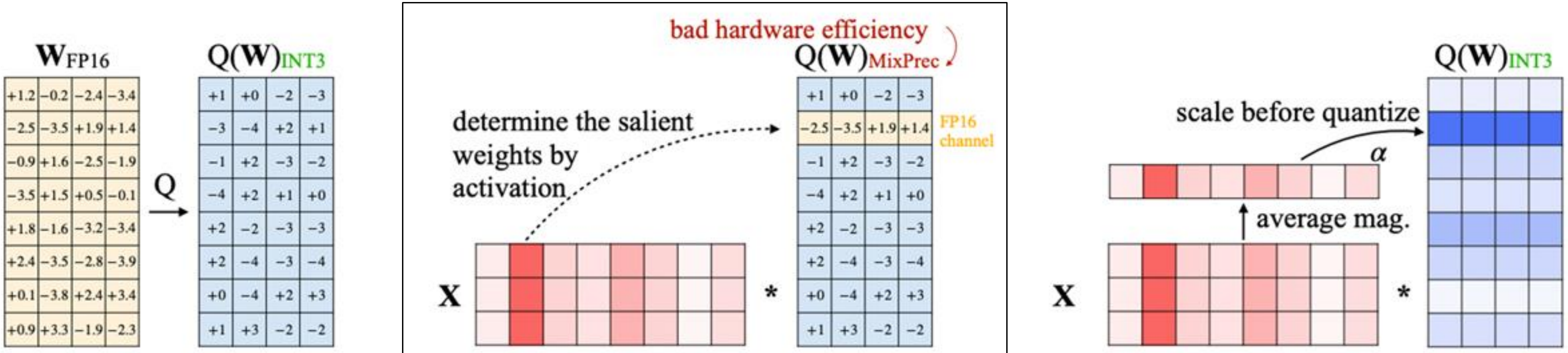
(b) Keep 1% salient weights in FP16 (**PPL** 13.0)

(c) Scale the weights before quantization (**PPL** 13.0)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

AWQ (W4A16)

LLM.int8() can resolve these issues, but mixed precision matrix multiplication is hardware inefficient

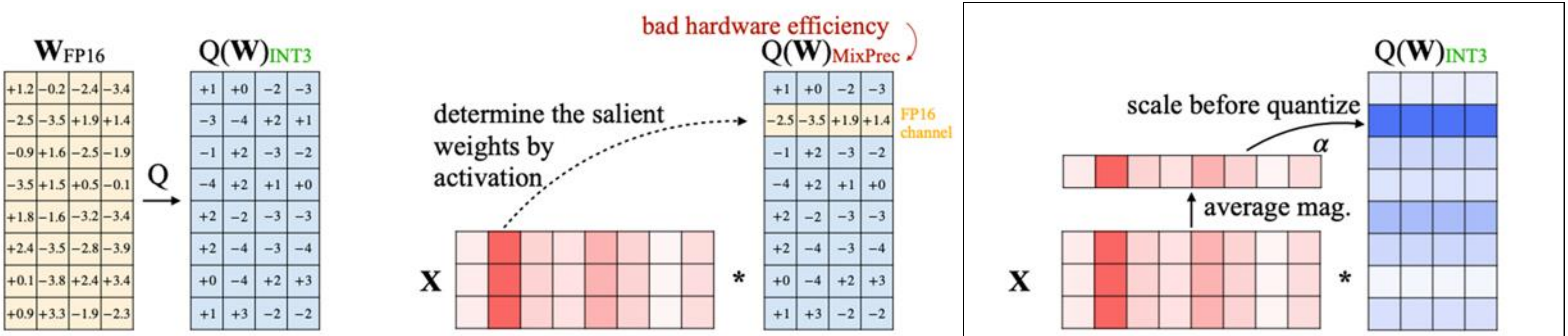


(a) RTN quantization (**PPL** 43.2) (b) Keep 1% salient weights in FP16 (**PPL** 13.0) (c) Scale the weights before quantization (**PPL** 13.0)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

AWQ (W4A16)

As in SmoothQuant, we can resolve this issue by shifting the difficulty to the weights using a scaling factor.

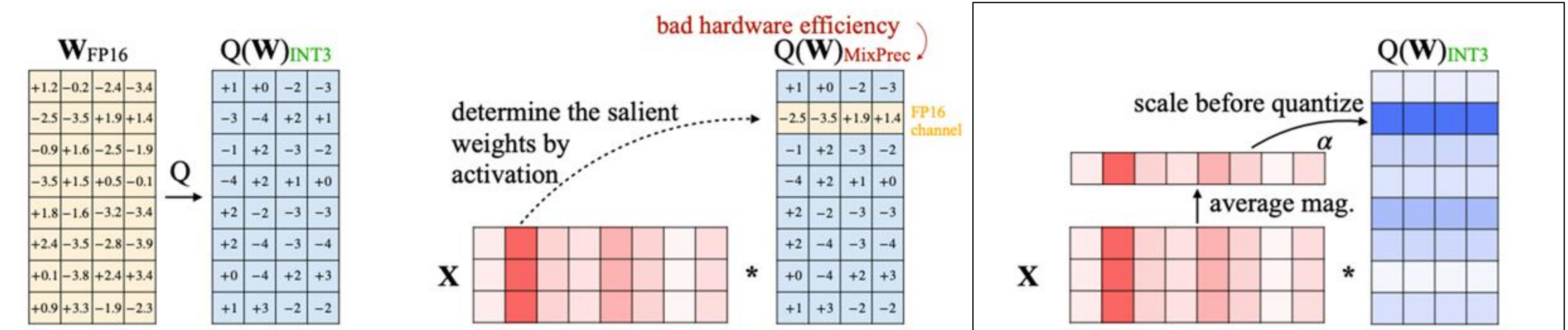


(a) RTN quantization (PPL 43.2) (b) Keep 1% salient weights in FP16 (PPL 13.0) (c) Scale the weights before quantization (PPL 13.0)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

AWQ (W4A16)

Where Smoothquant quantizes both activations and weights, AWQ only quantizes the weights

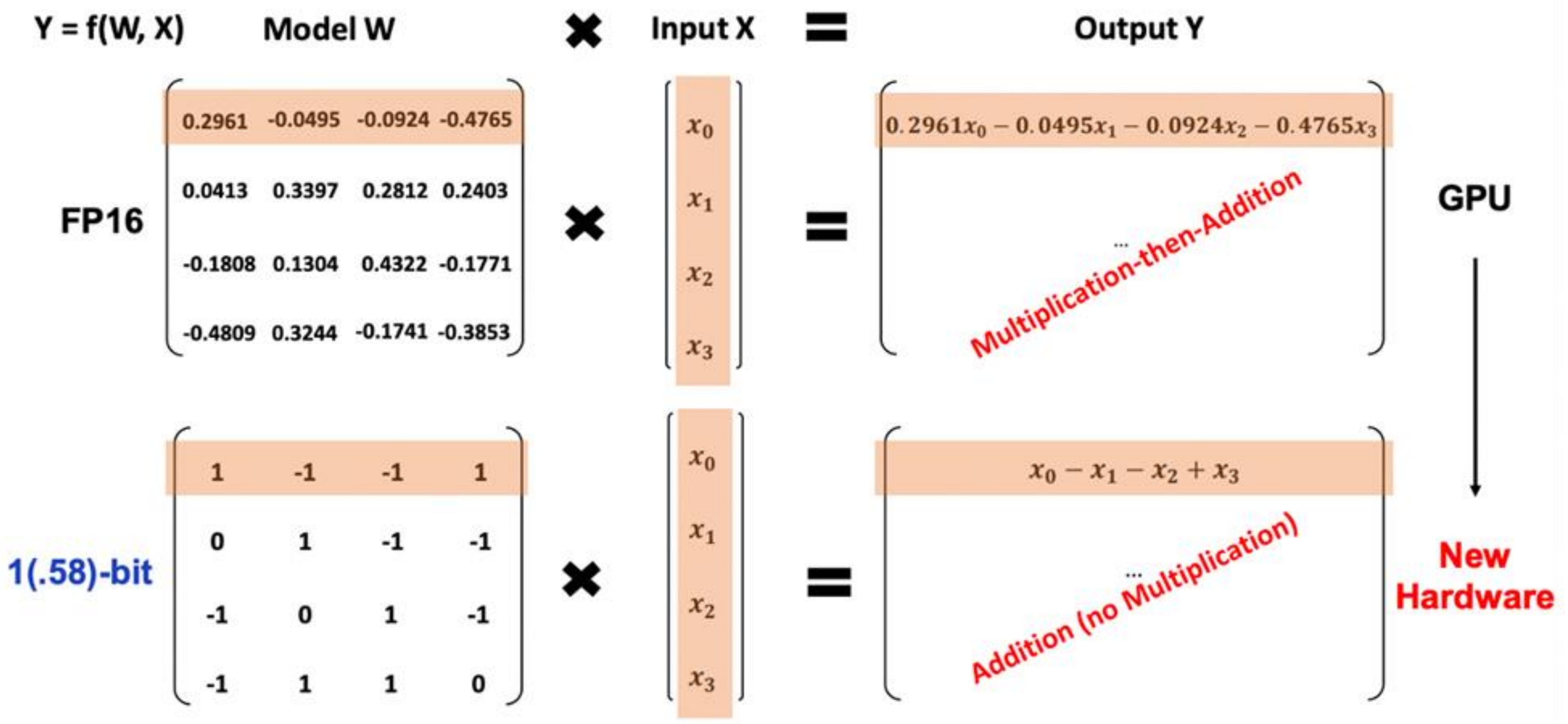


(a) RTN quantization (**PPL** 43.2) (b) Keep 1% salient weights in FP16 (**PPL** 13.0) (c) Scale the weights before quantization (**PPL** 13.0)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

Era of 1-bit LLMs (W1.58A8)

Weight-only QAT algorithm that uses only weights in {-1, 0, 1}



The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits [Ma et al., 2024]

Era of 1-bit LLMs (W1.58A8)

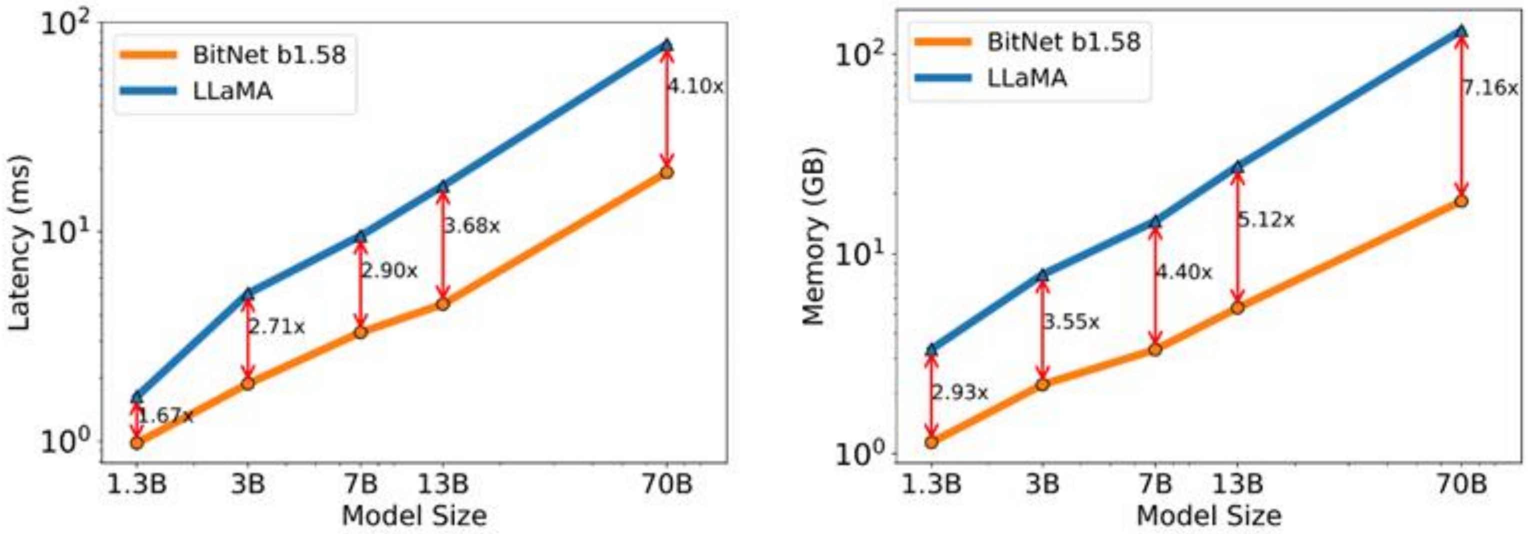


Figure 2: Decoding latency (Left) and memory consumption (Right) of BitNet b1.58 varying the model size.

Models	Size	Max Batch Size	Throughput (tokens/s)
LLaMA LLM	70B	16 (1.0x)	333 (1.0x)
BitNet b1.58	70B	**176 (11.0x)**	**2977 (8.9x)**

Table 3: Comparison of the throughput between BitNet b1.58 70B and LLaMA LLM 70B.

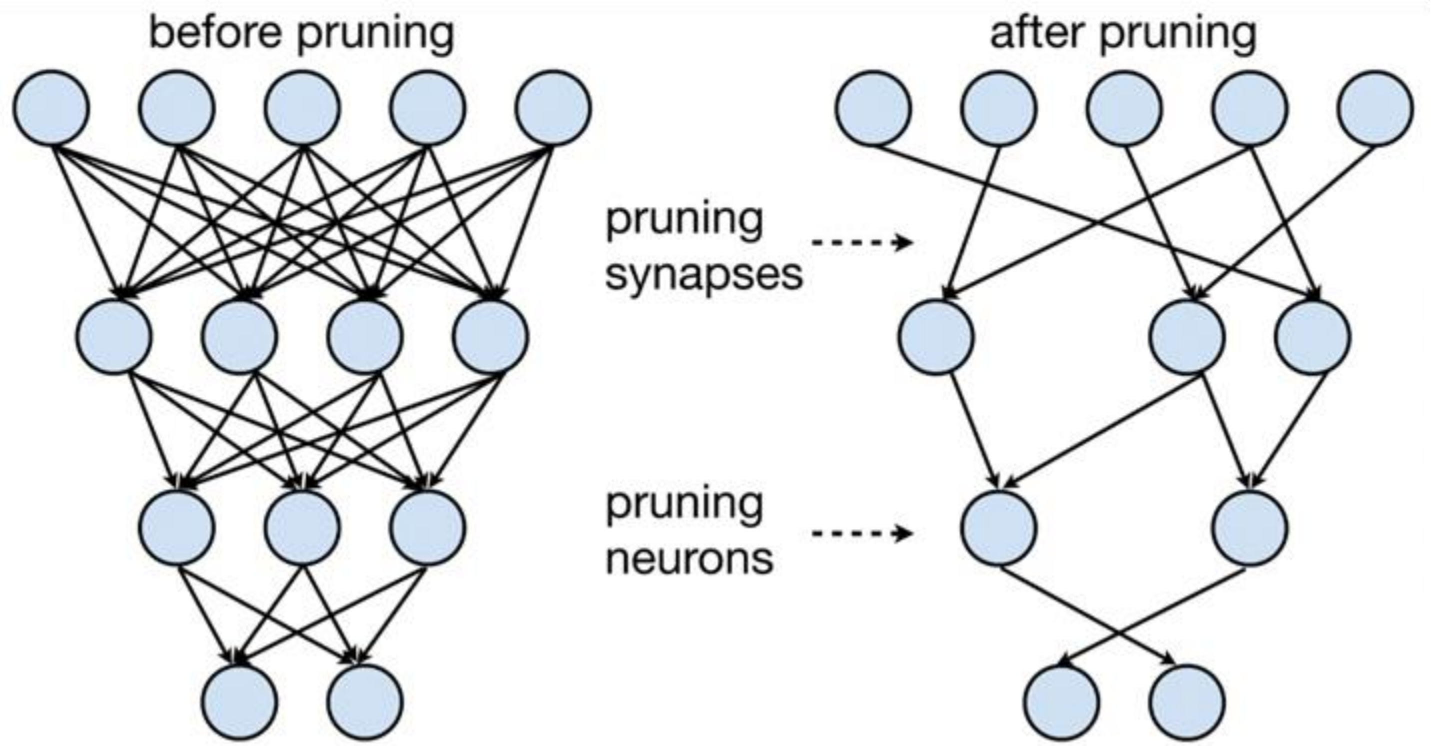
The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits [Ma et al., 2024]

Efficient LLMs

- Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- **Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)**
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)

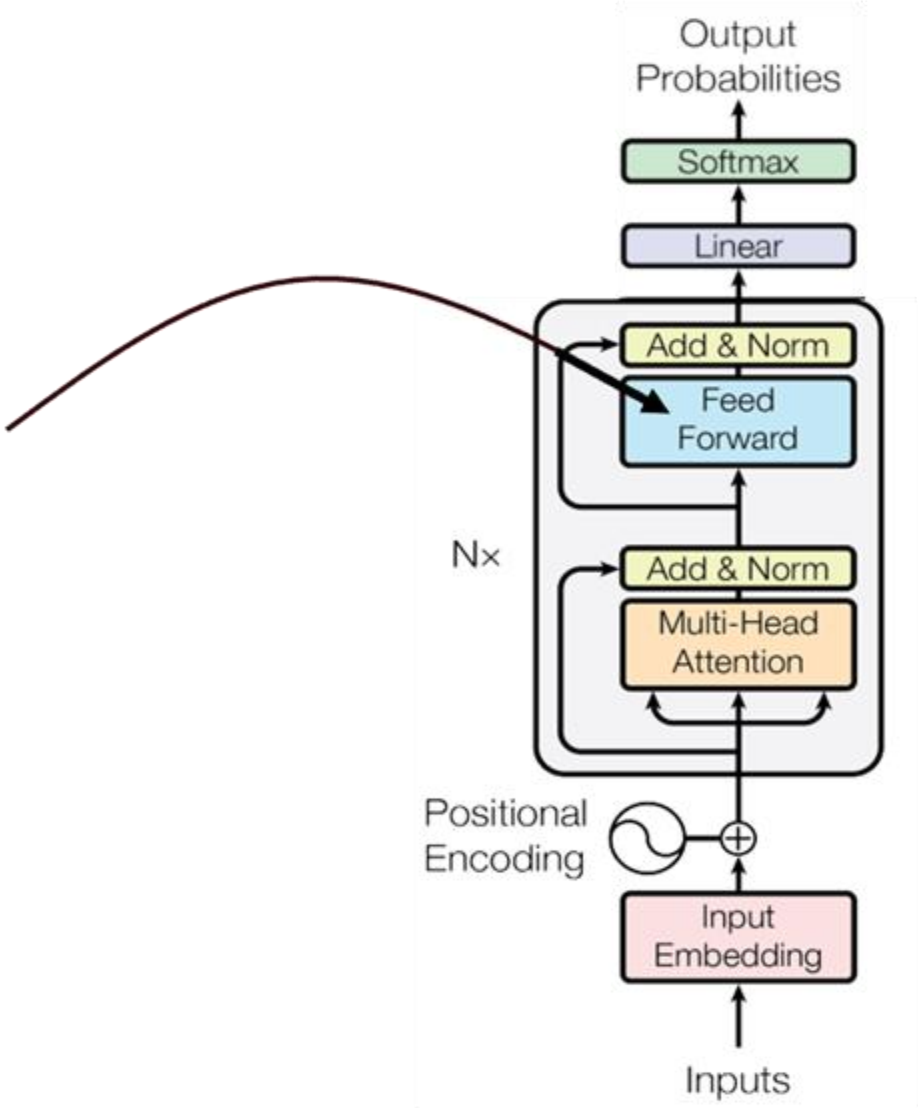
Sparsity

Even though our model may have many parameters, we can get speedups by only using a much smaller number of those parameters for a given instance

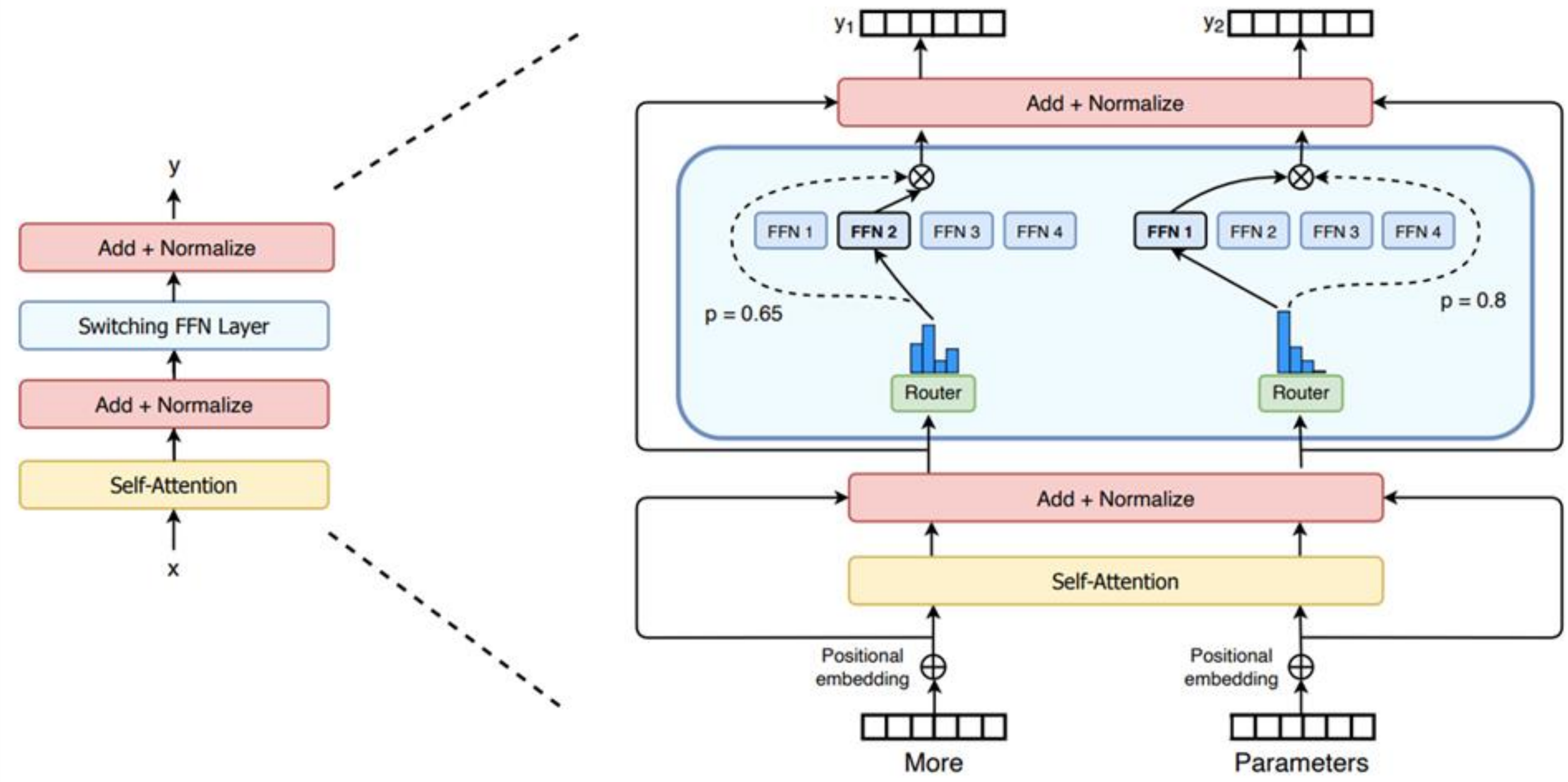


Mixture of Experts (MoE)

Replace FFN layers in traditional transformers with a switching FFN layer (more generally called an MoE layer)



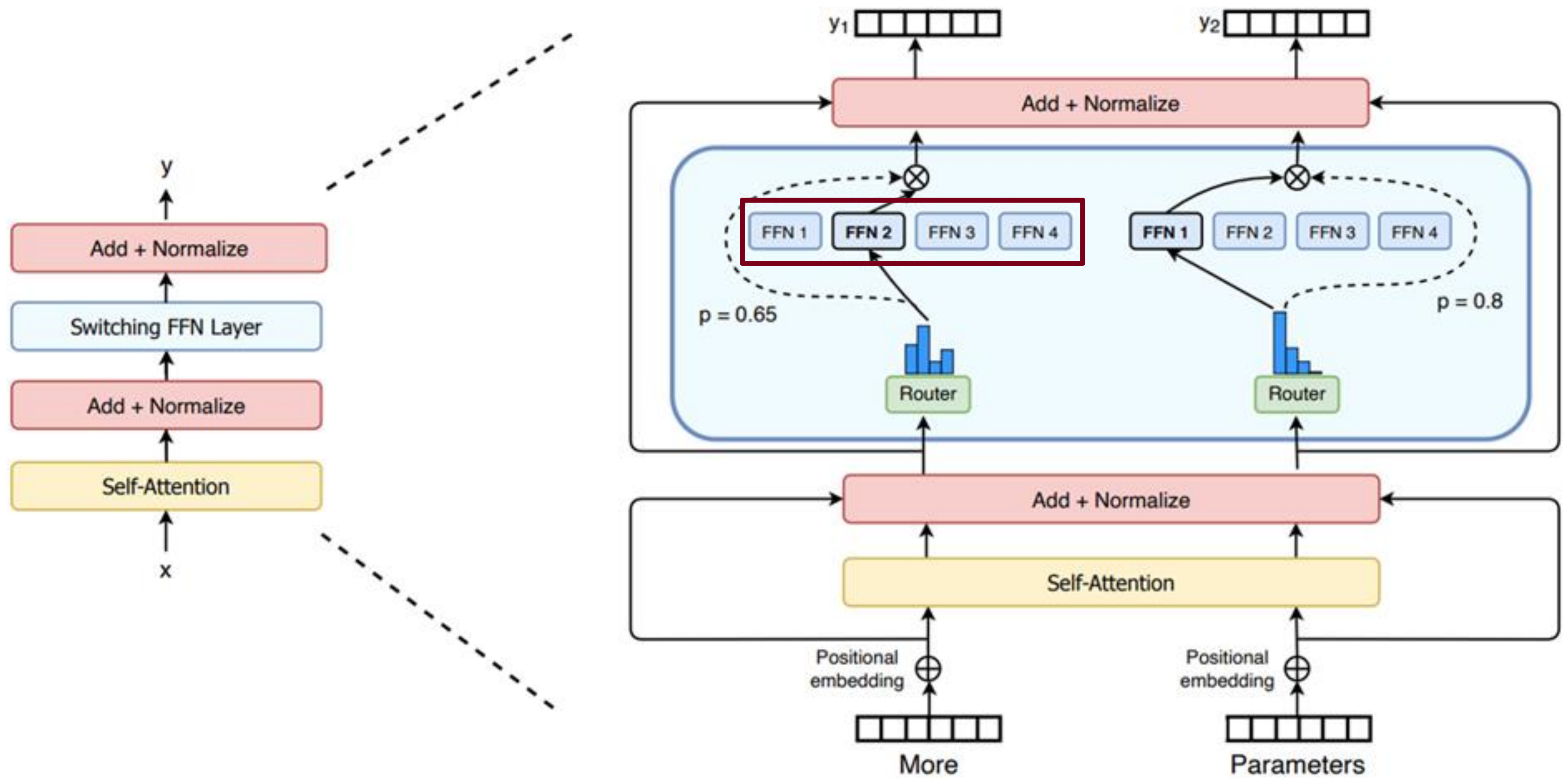
Mixture of Experts (MoE)



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., CoRR 2021]

Mixture of Experts (MoE)

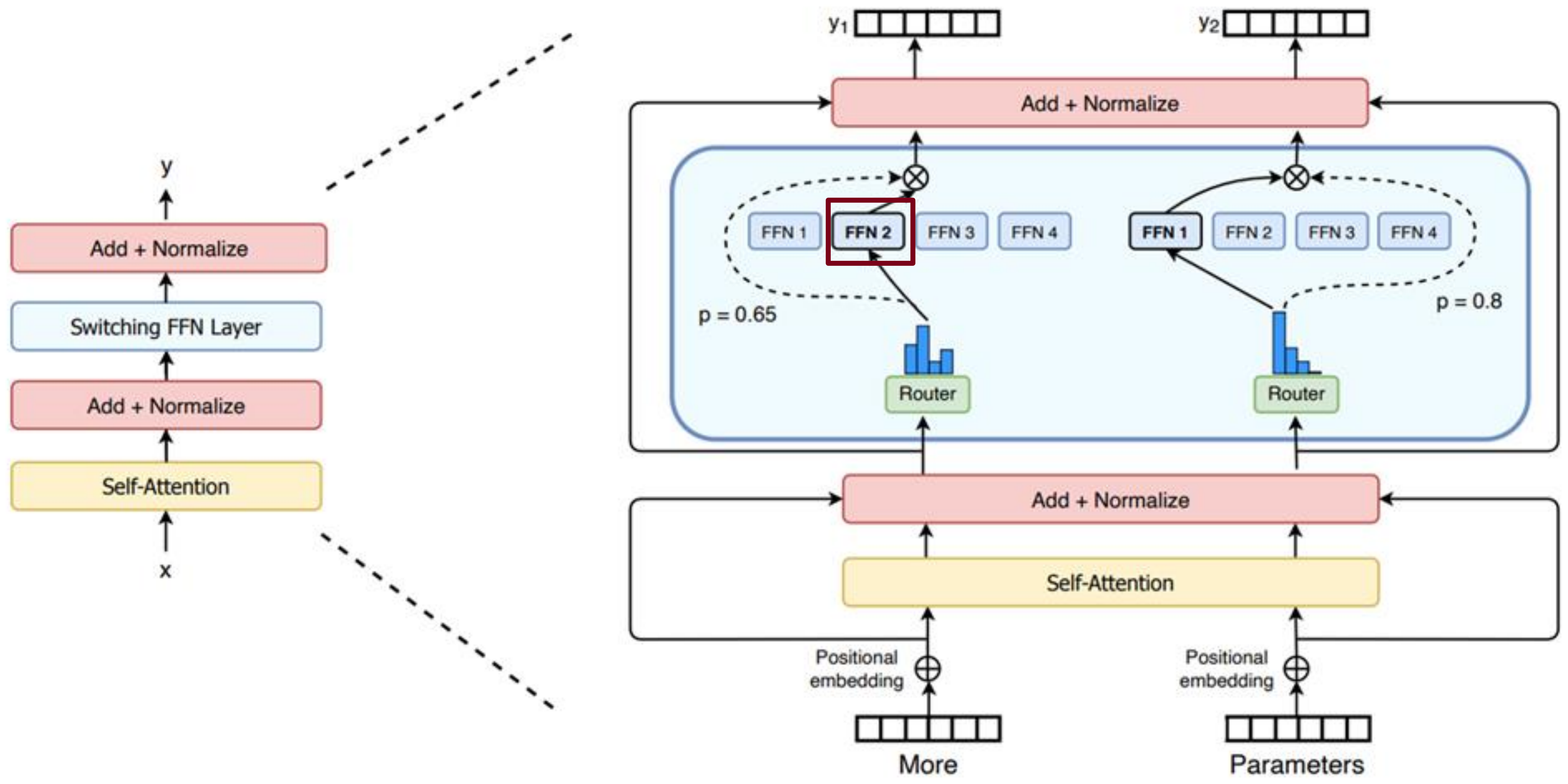
Four FFN layers



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., CoRR 2021]

Mixture of Experts (MoE)

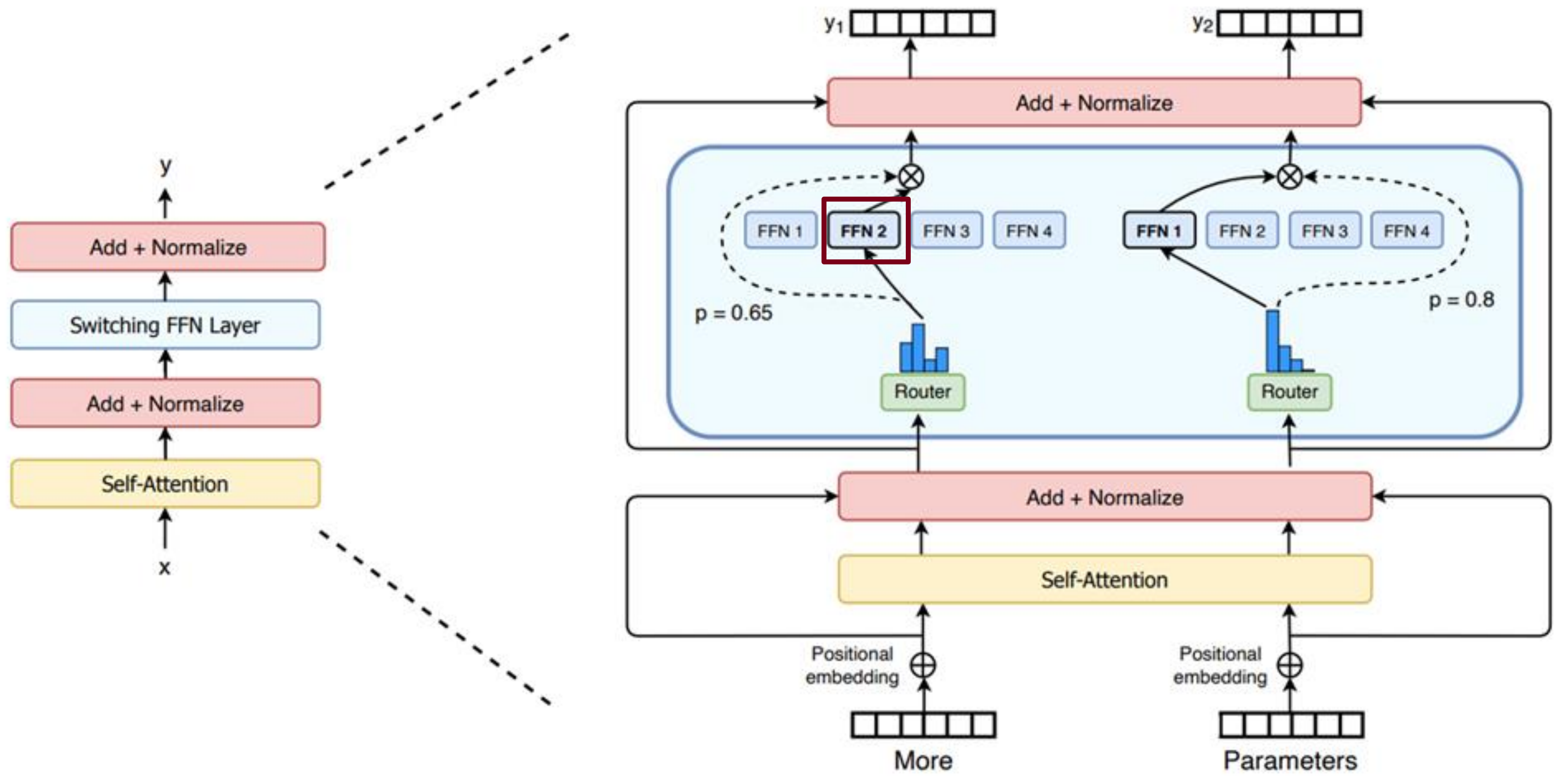
Only one is used per token



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., CoRR 2021]

Mixture of Experts (MoE)

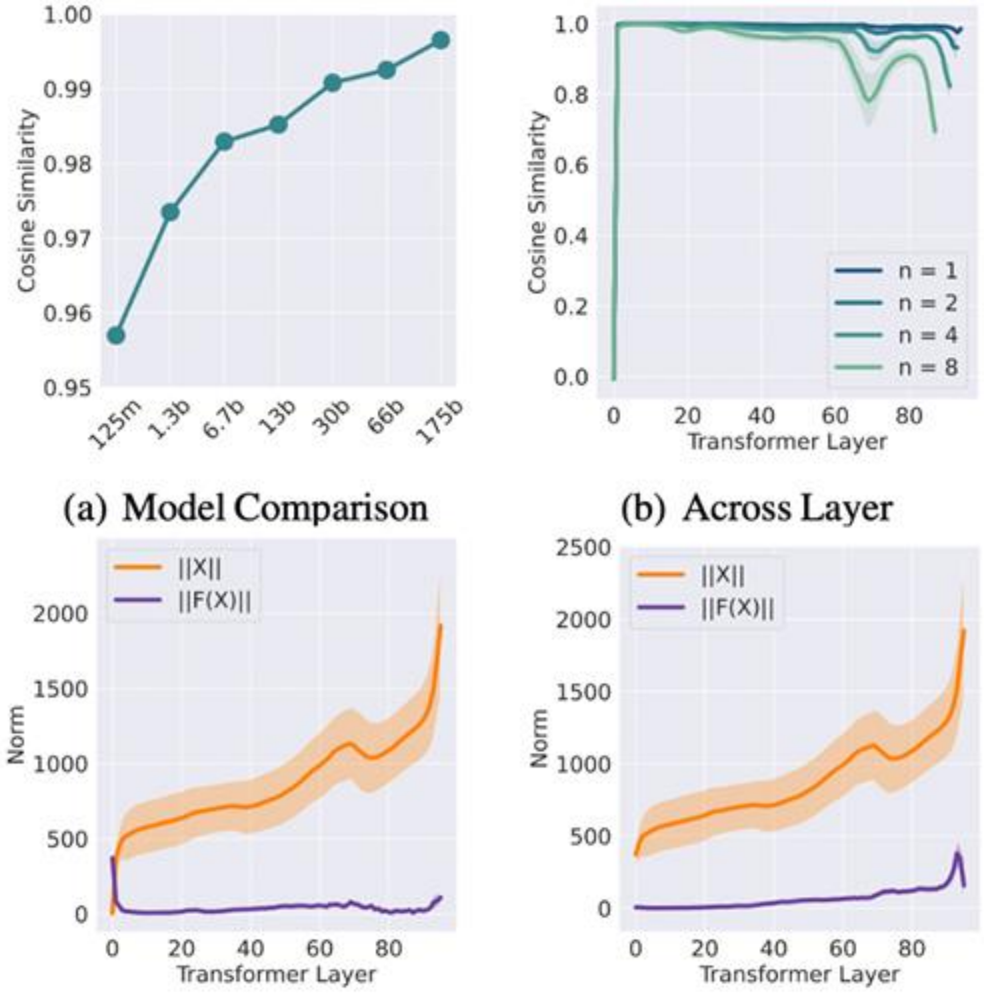
Only 25% of the FFN parameters are used for a single token



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., CoRR 2021]

Deja Vu: Contextual Sparsity

Observation 1: Model activations change very little between consecutive layers of a network



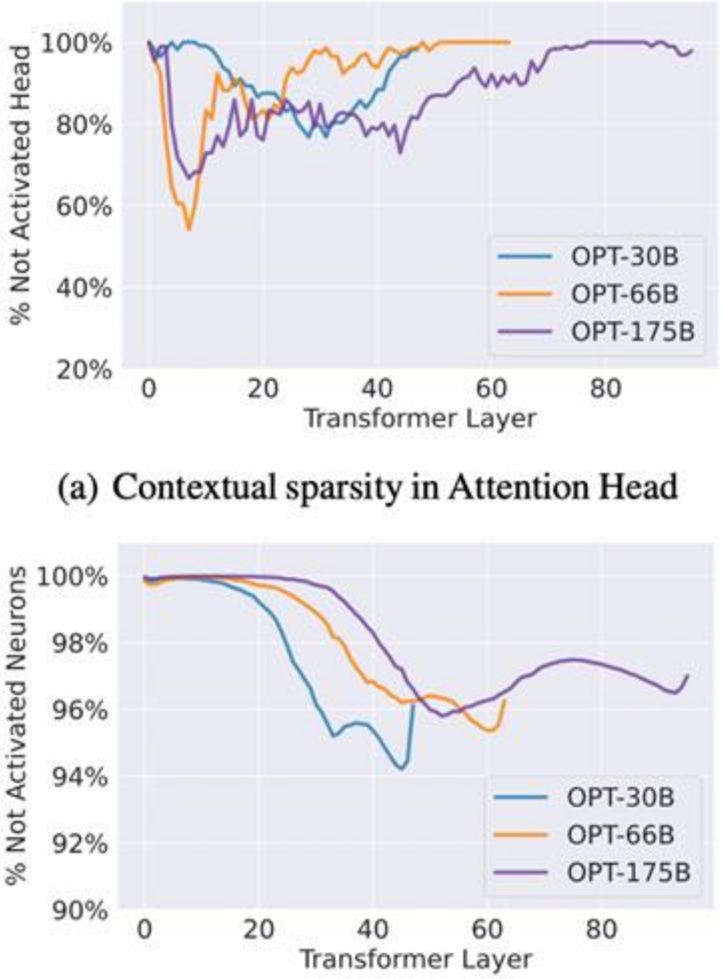
(a) Model Comparison (b) Across Layer

(c) Residual Around Attention (d) Residual Around MLP

Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time [Liu et al., 2023]

Deja Vu: Contextual Sparsity

Observation 2: Most attention heads and most neurons are not used



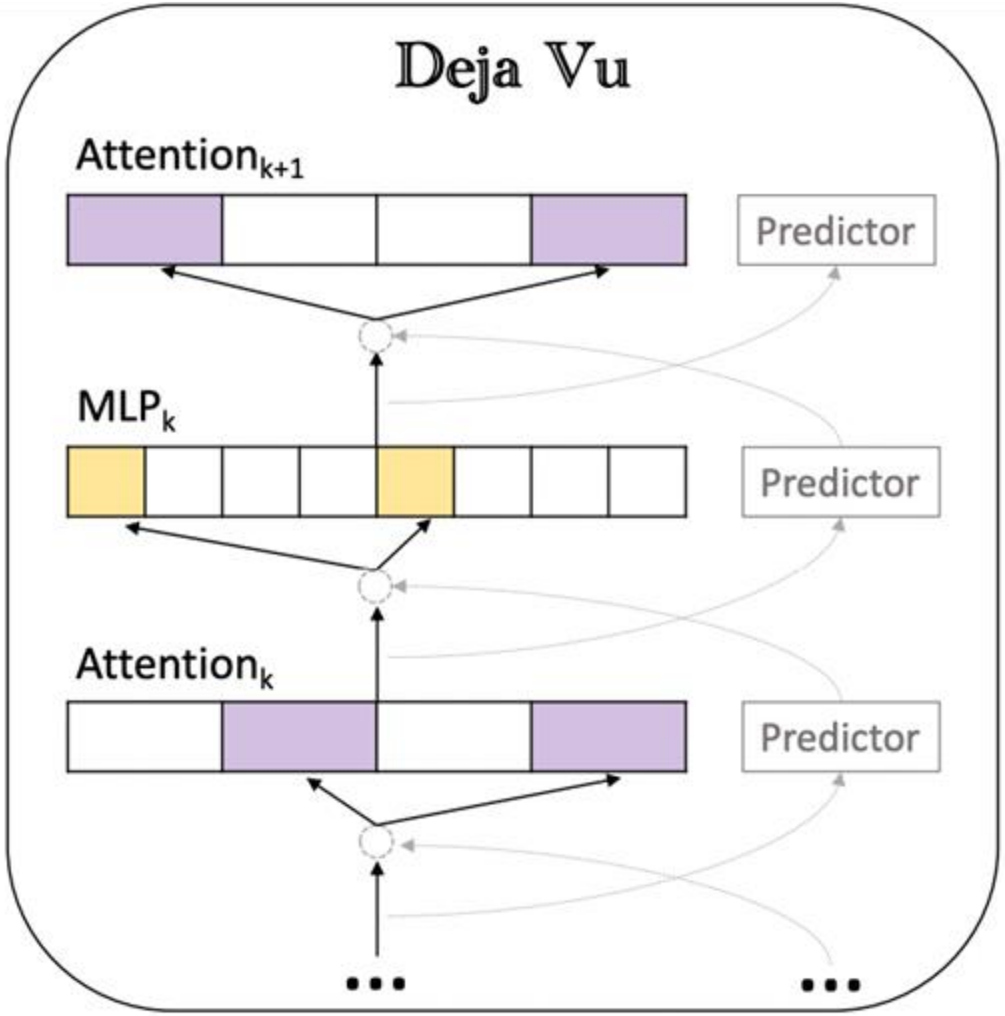
(a) Contextual sparsity in Attention Head

(b) Contextual sparsity in MLP Block

Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time [Liu et al., 2023]

Deja Vu: Contextual Sparsity

Sparsification: Use predictors in each layer to determine which neurons to activate and which attention heads to use – ignore all unpredicted heads/neurons



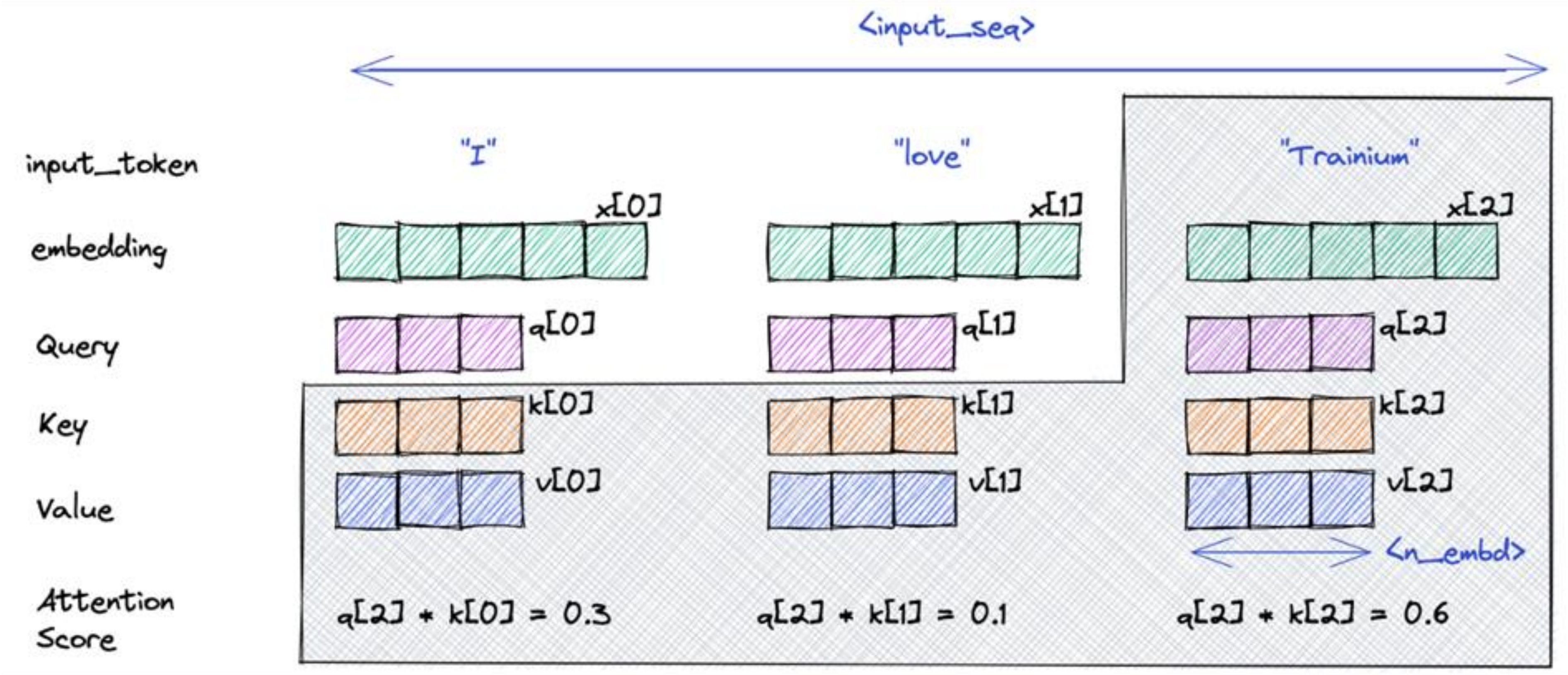
Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time [Liu et al., 2023]

Efficient LLMs

- Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- **Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)**
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- Distillation

The KV-Cache

The transformer needs to have access to the keys and values for all previous tokens in all layers for all heads when



https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/appnotes/transformers-neuronx/generative-llm-inference-with-neuron.html

The KV-Cache

In total, we must store

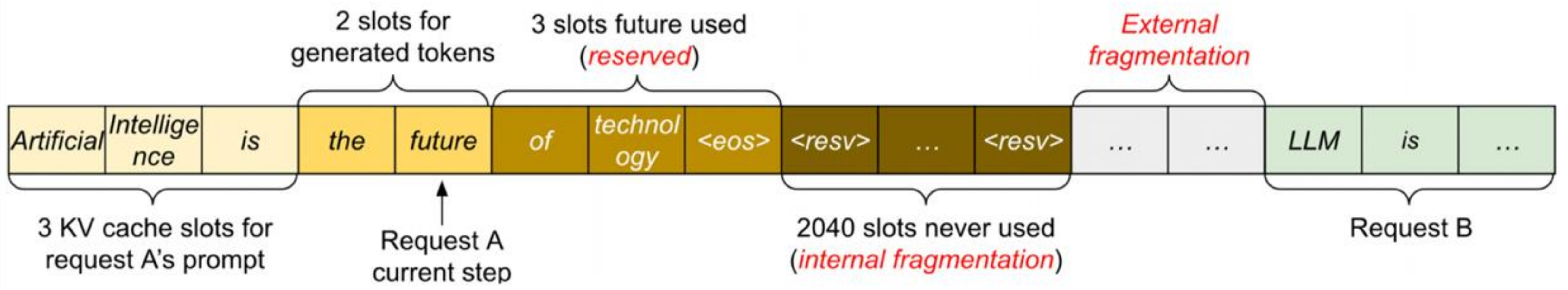
Batch_size * seq_len * num_heads * num_layers * emb_dim * 2

separate values in the kv cache

PagedAttention

How does a large LLM service (large ChatGPT) handle multiple incoming requests with respect to the KV-cache?

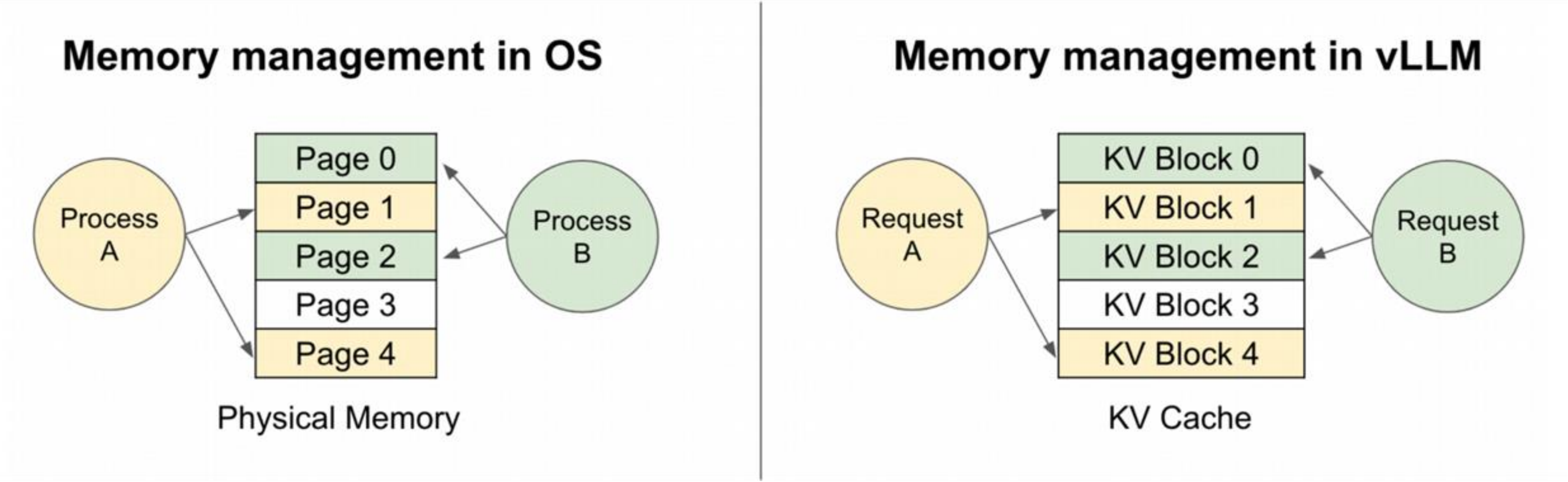
-Originally, most systems just assign fixed sized blocks of memory to each incoming request. How to improve?



Efficient Memory Management for Large Language Model Serving with PagedAttention (Kwon et al., 2023)

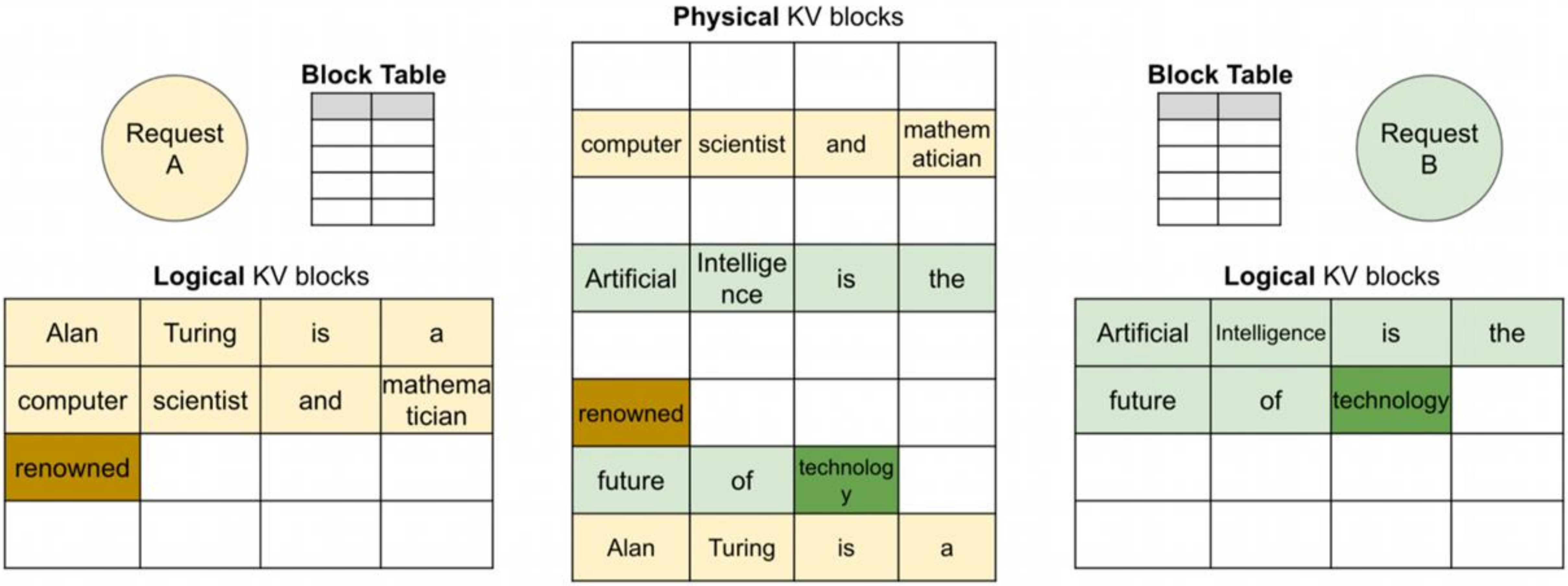
PagedAttention

Let's adopt a similar approach to that found in virtual memory!



Efficient Memory Management for Large Language Model Serving with PagedAttention (Kwon et al., 2023)

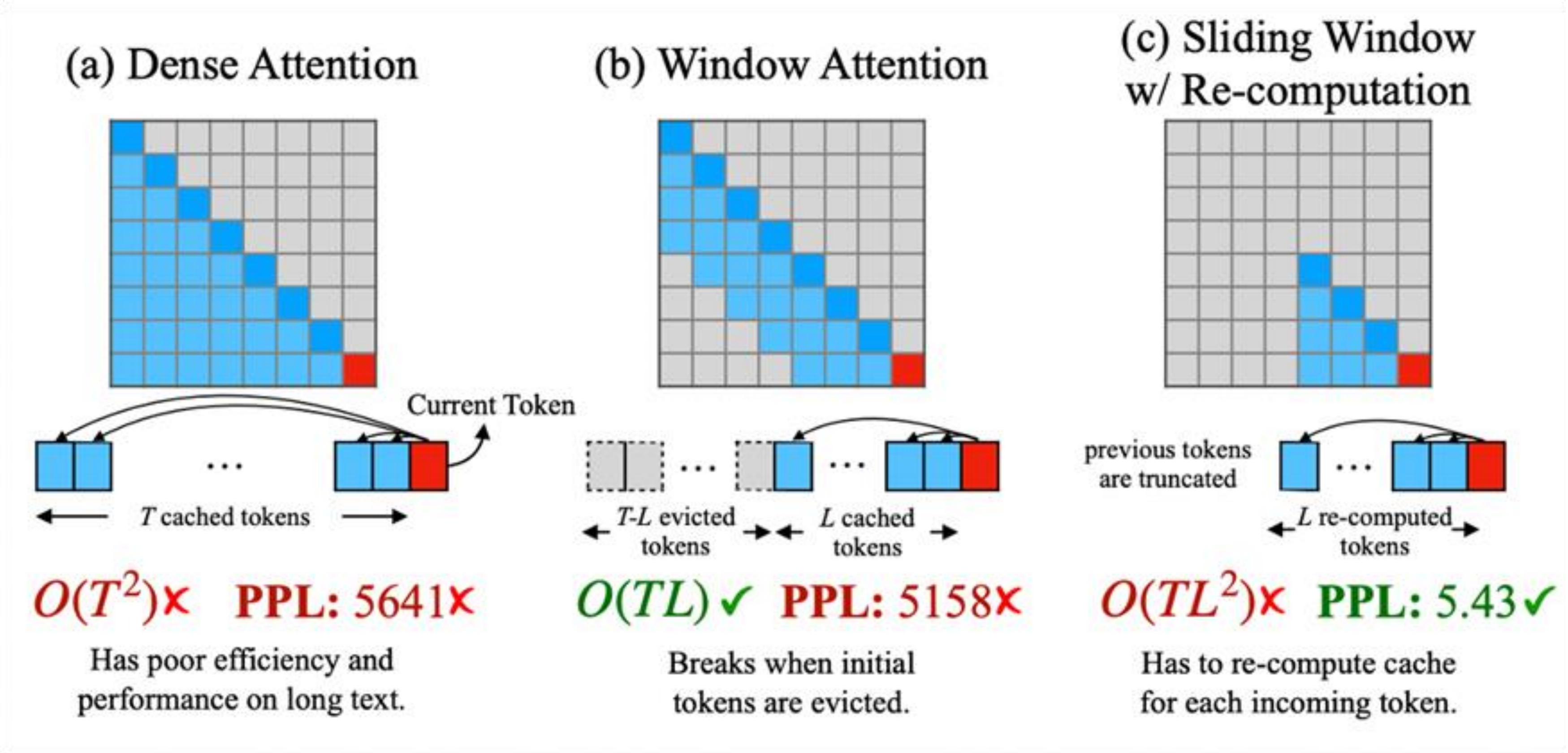
PagedAttention



Efficient Memory Management for Large Language Model Serving with PagedAttention (Kwon et al., 2023)

StreamingLLM

How can we extend models to have much longer context length at minimal cost?

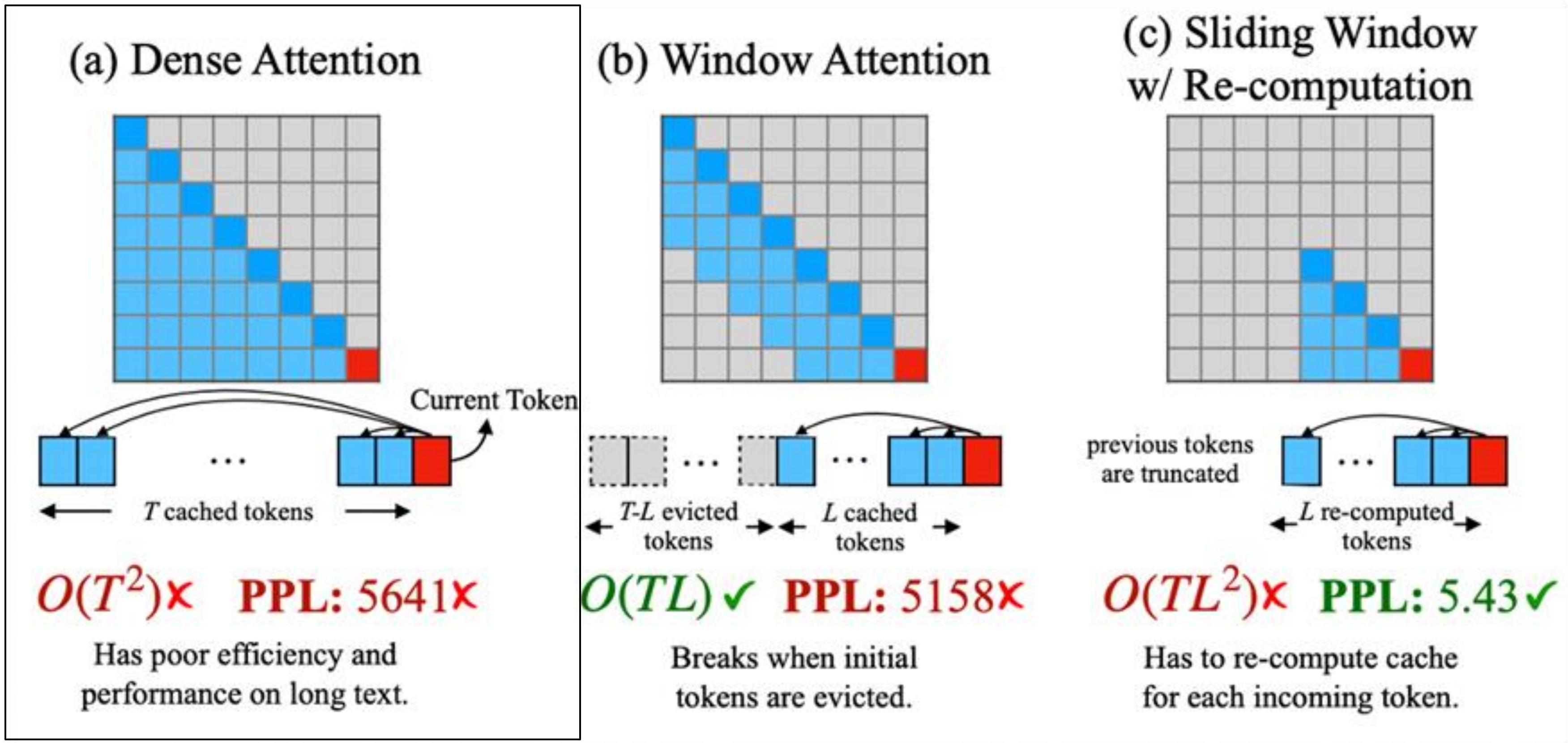


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

How can we extend models to have much longer context length at minimal cost?

Too much storage

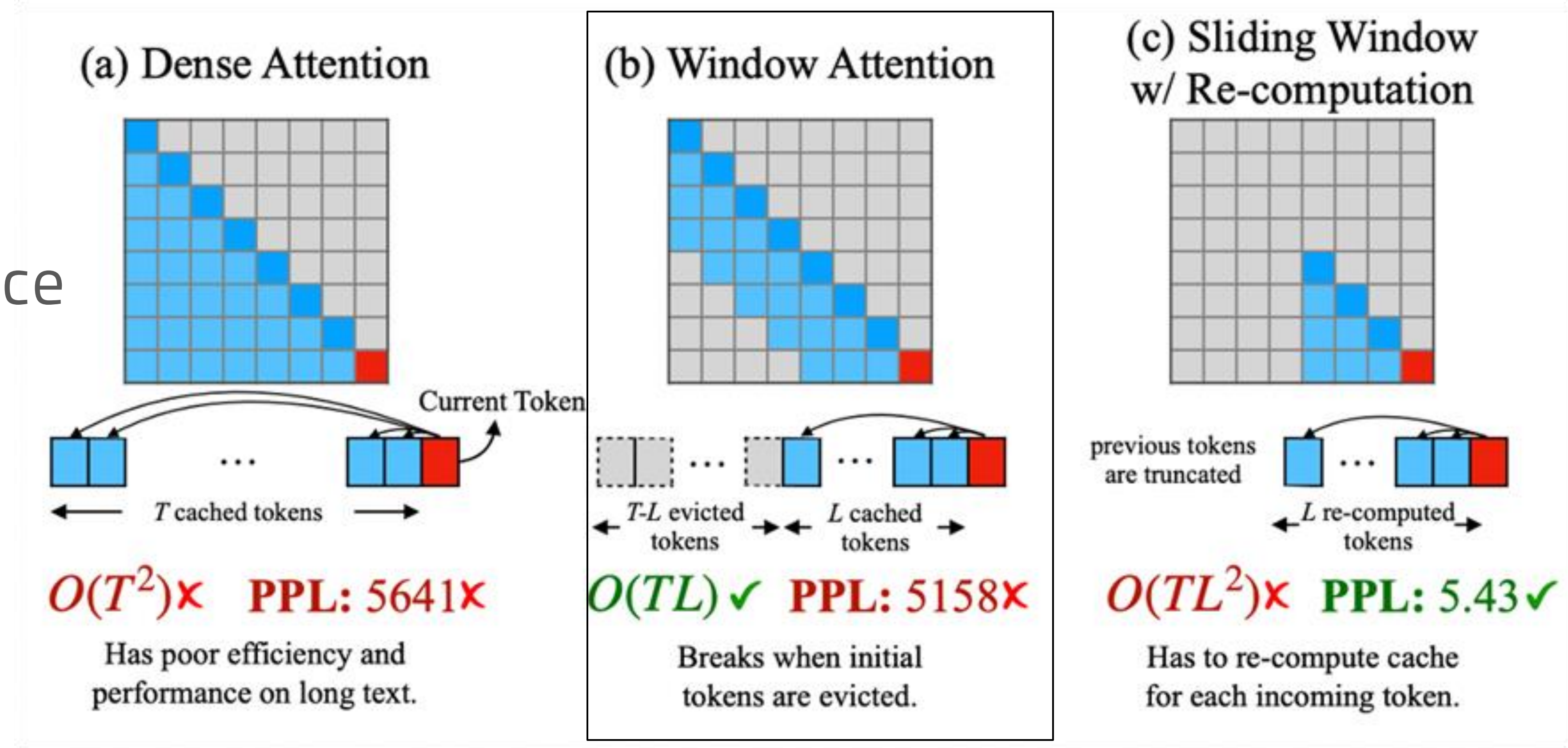


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

How can we extend models to have much longer context length at minimal cost?

Bad performance

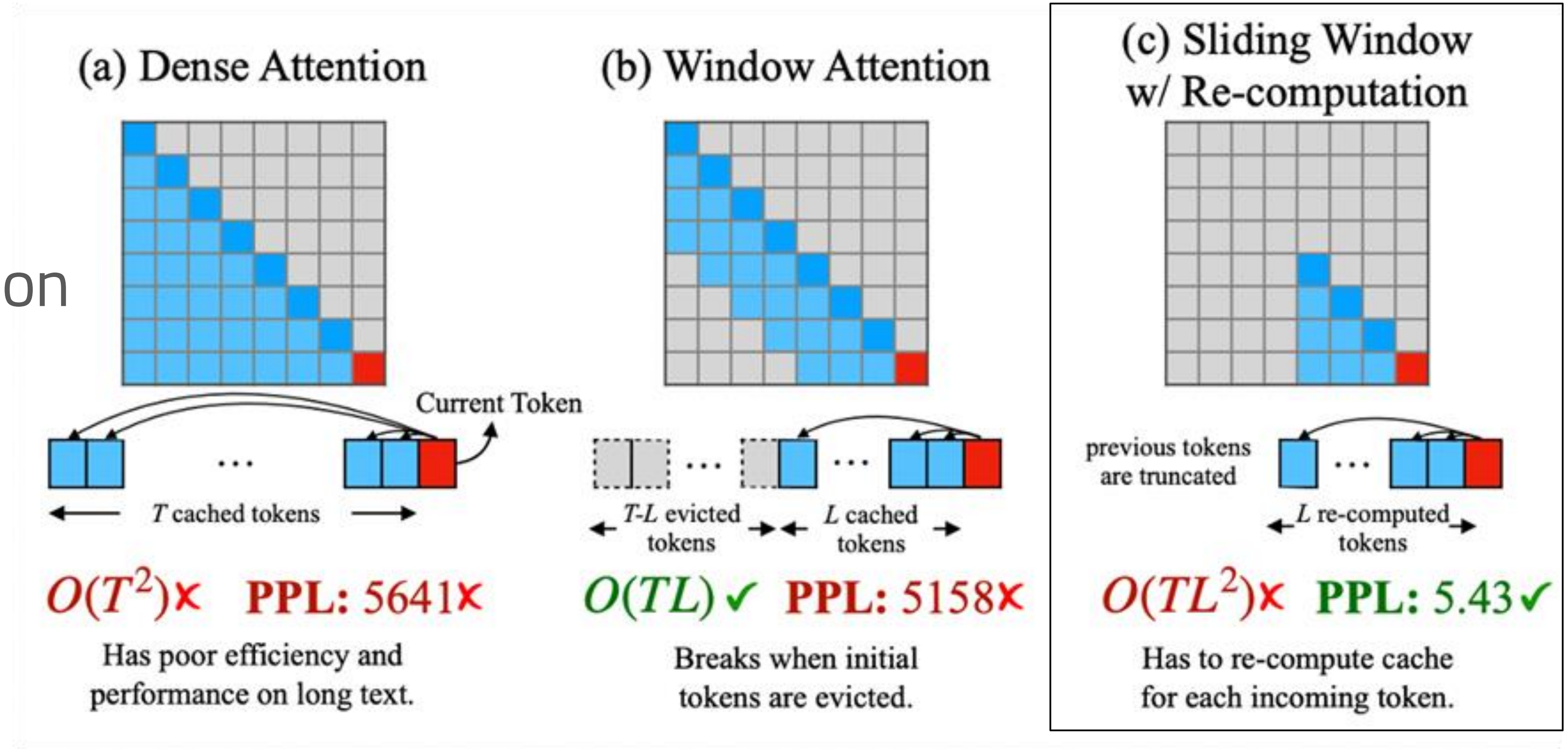


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

How can we extend models to have much longer context length at minimal cost?

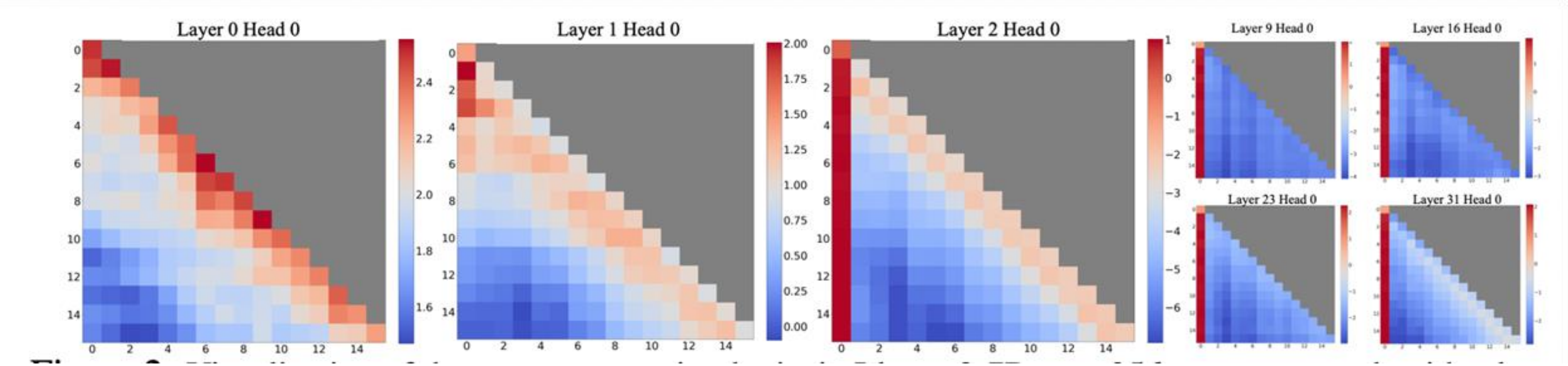
Too much recomputation



Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

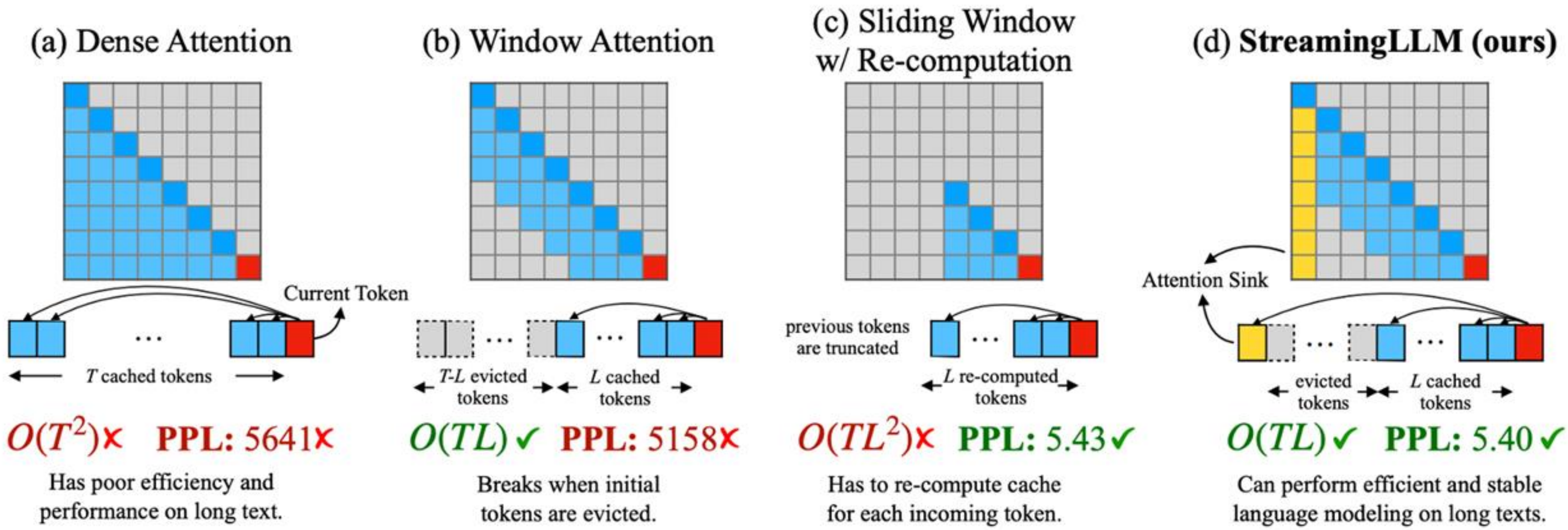
StreamingLLM

Observation: Most attention is either placed on the first token or to tokens that the model has recently seen.



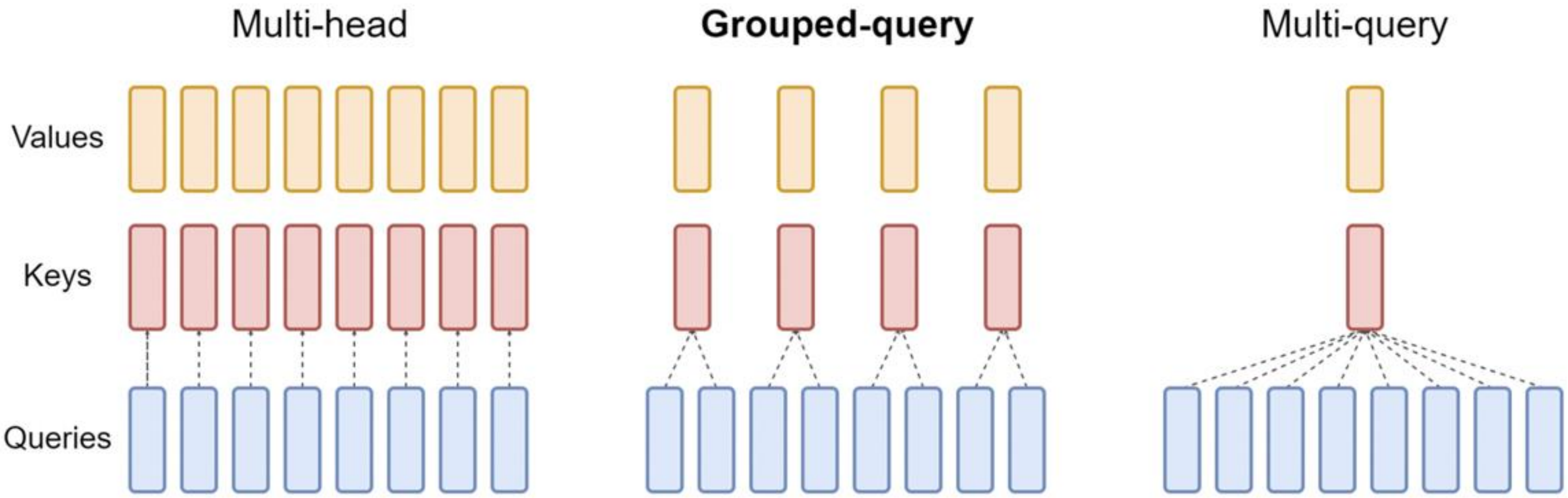
Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM



Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

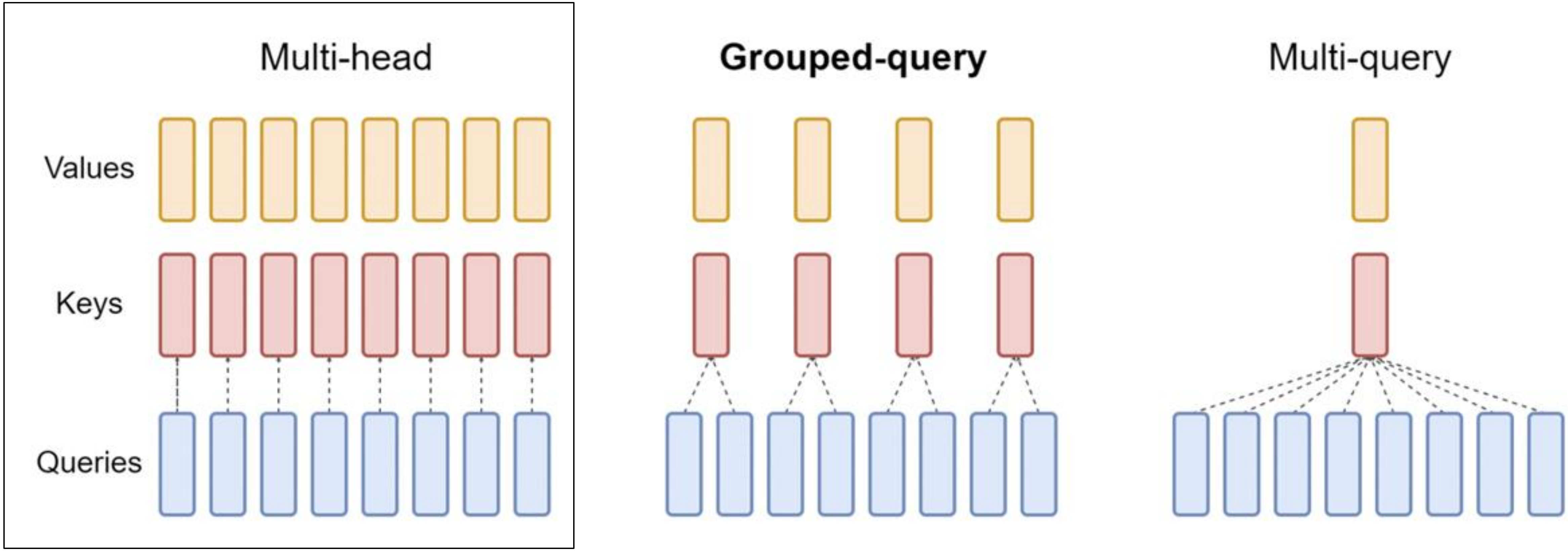
MHA/GQA/MQA



GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

MHA/GQA/MQA

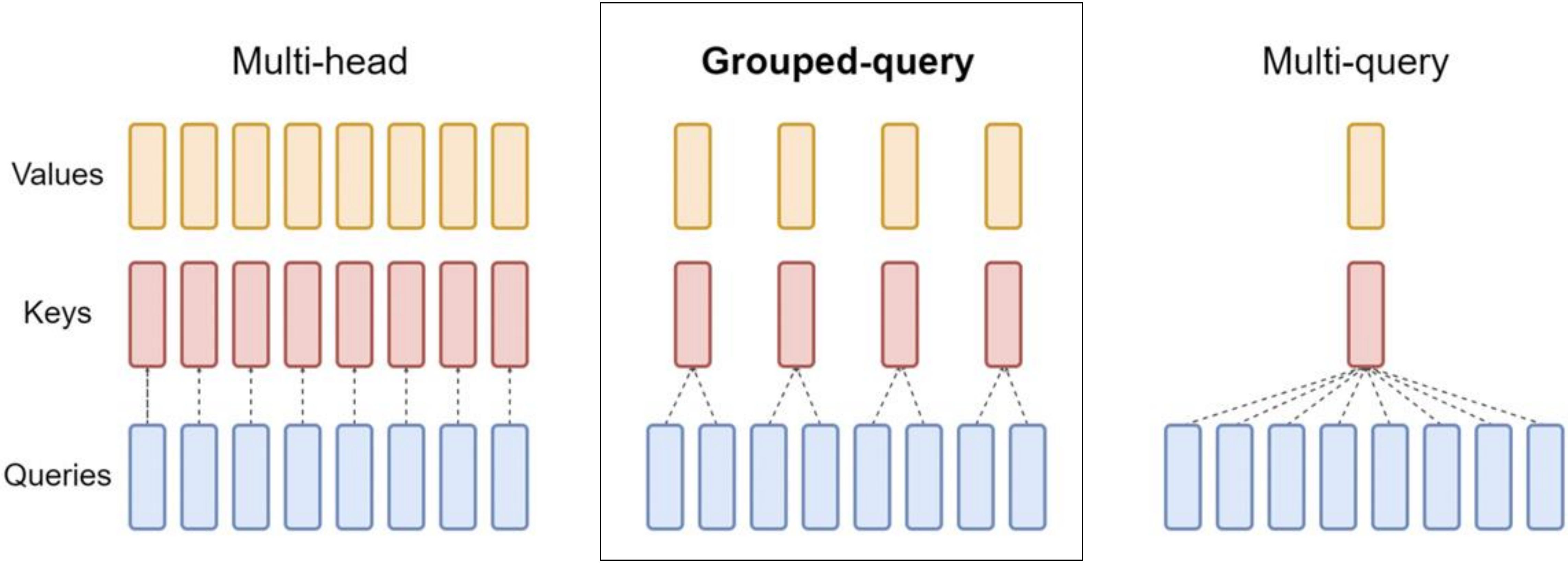
Each attention head calculates separate keys and values for each token



GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

MHA/GQA/MQA

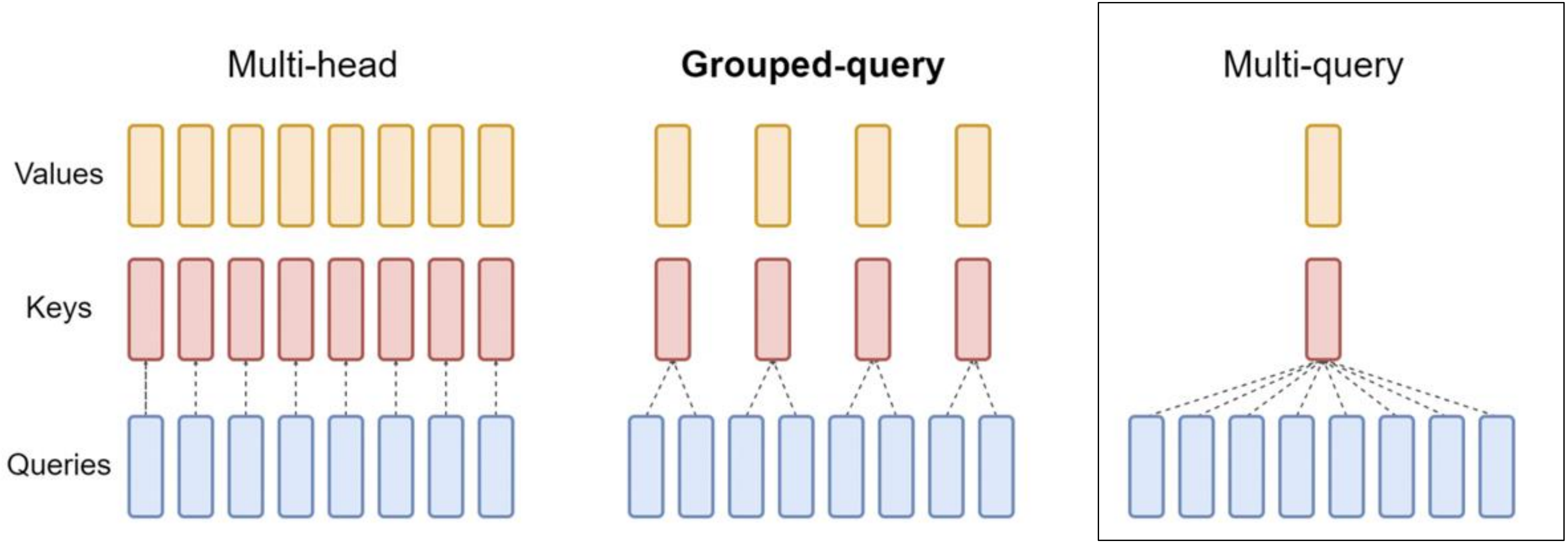
Attention heads are split into groups.
Each group has one key/value per token.



GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

MHA/GQA/MQA

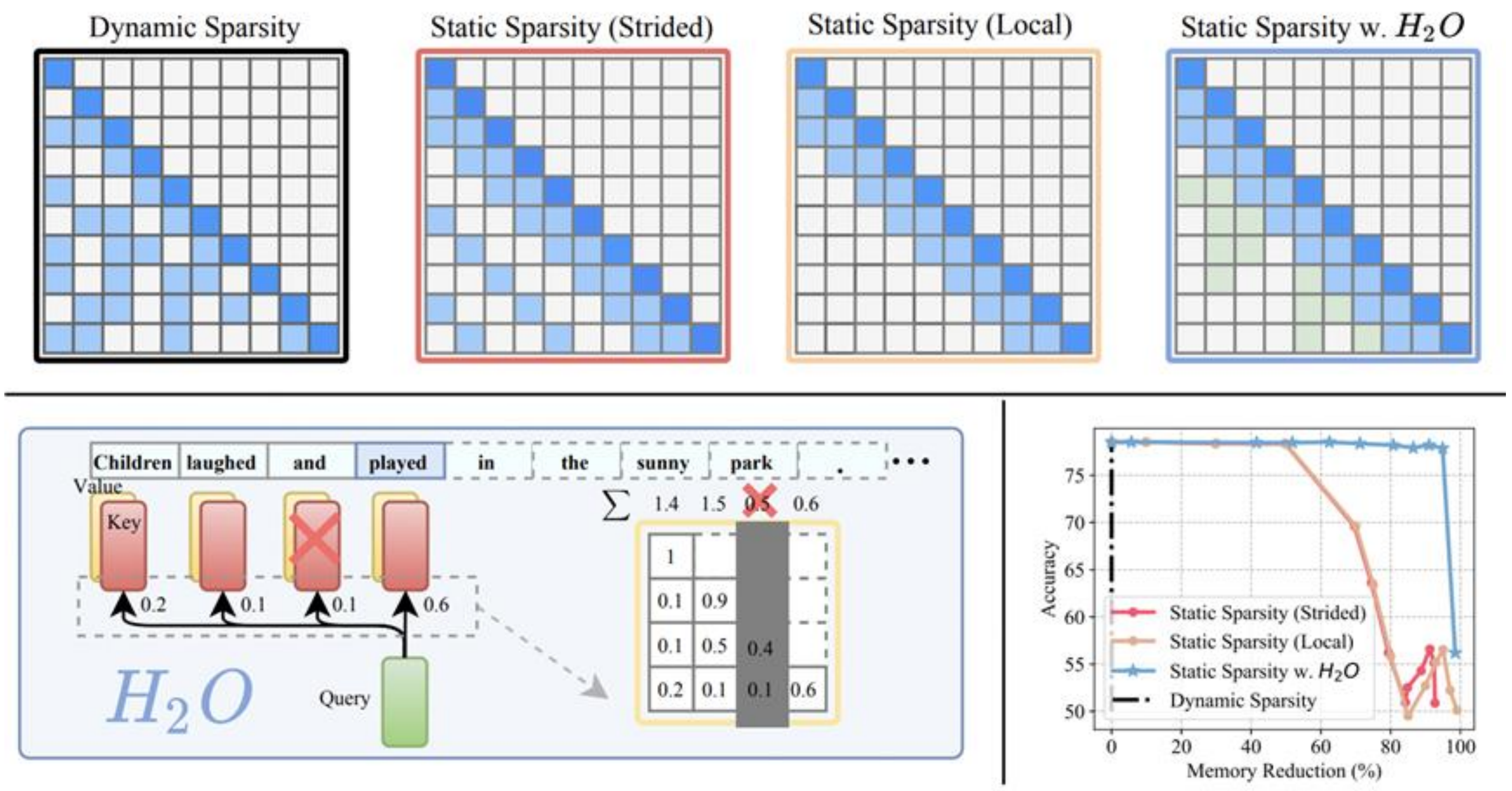
Attention heads share the same keys and values for each token



GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

H_2O: Heavy Hitter Oracle

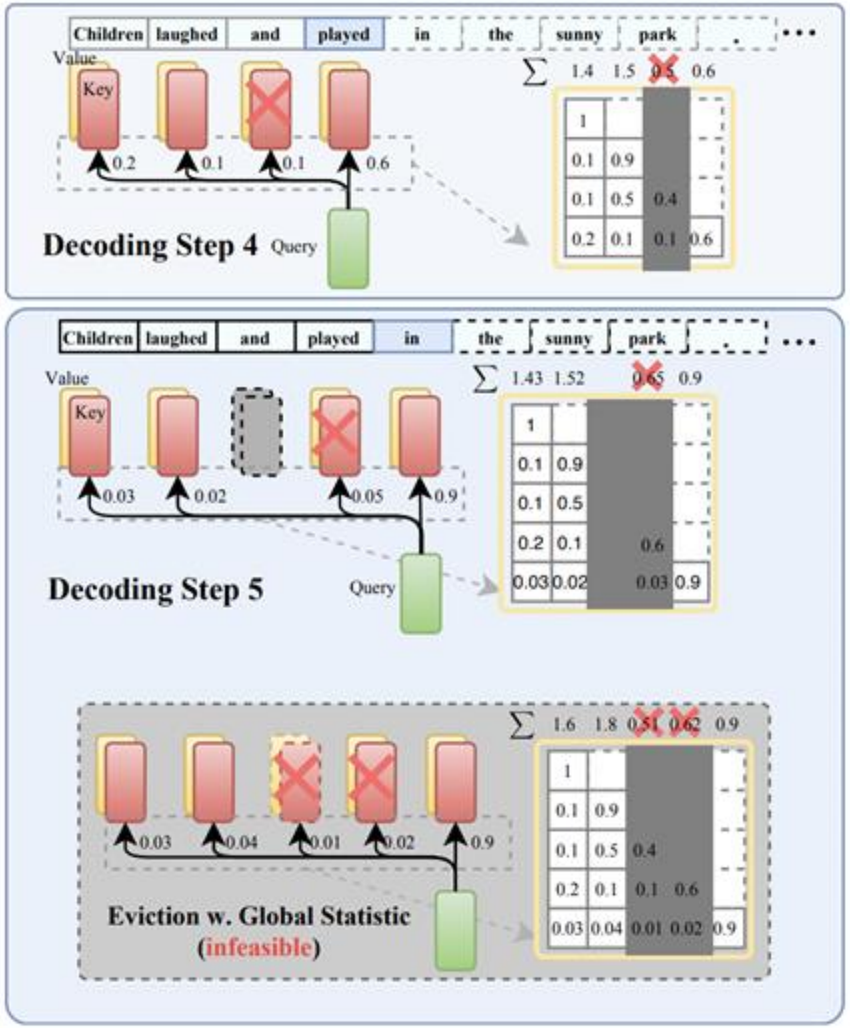
Evict all but k-highest cumulative attention tokens from cache



H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models [Zhang et. al, 2023]

H_2O: Heavy Hitter Oracle

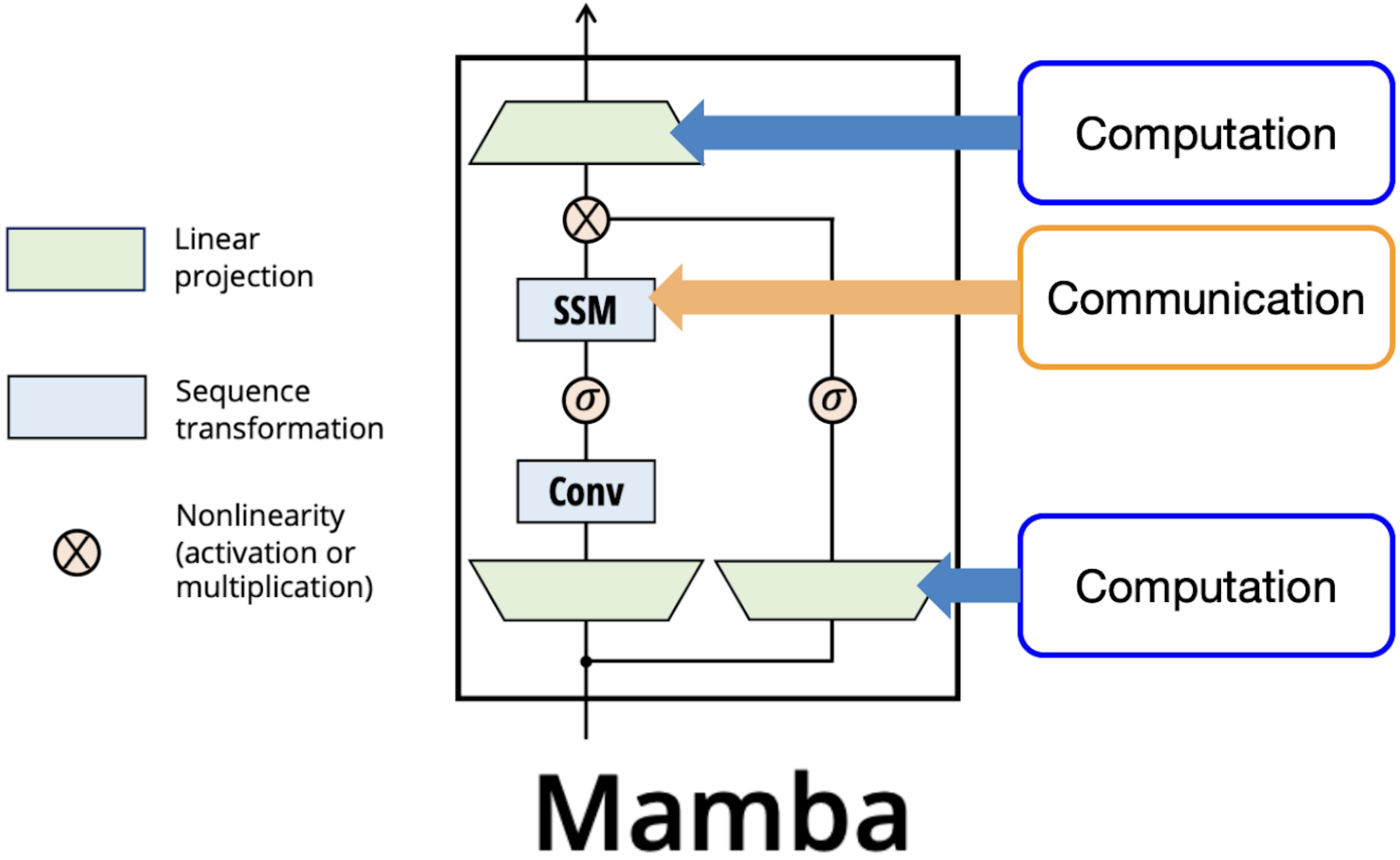
Evict all but k-highest cumulative attention tokens from cache



H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models [Zhang et. al, 2023]

State Space Model (SSM)

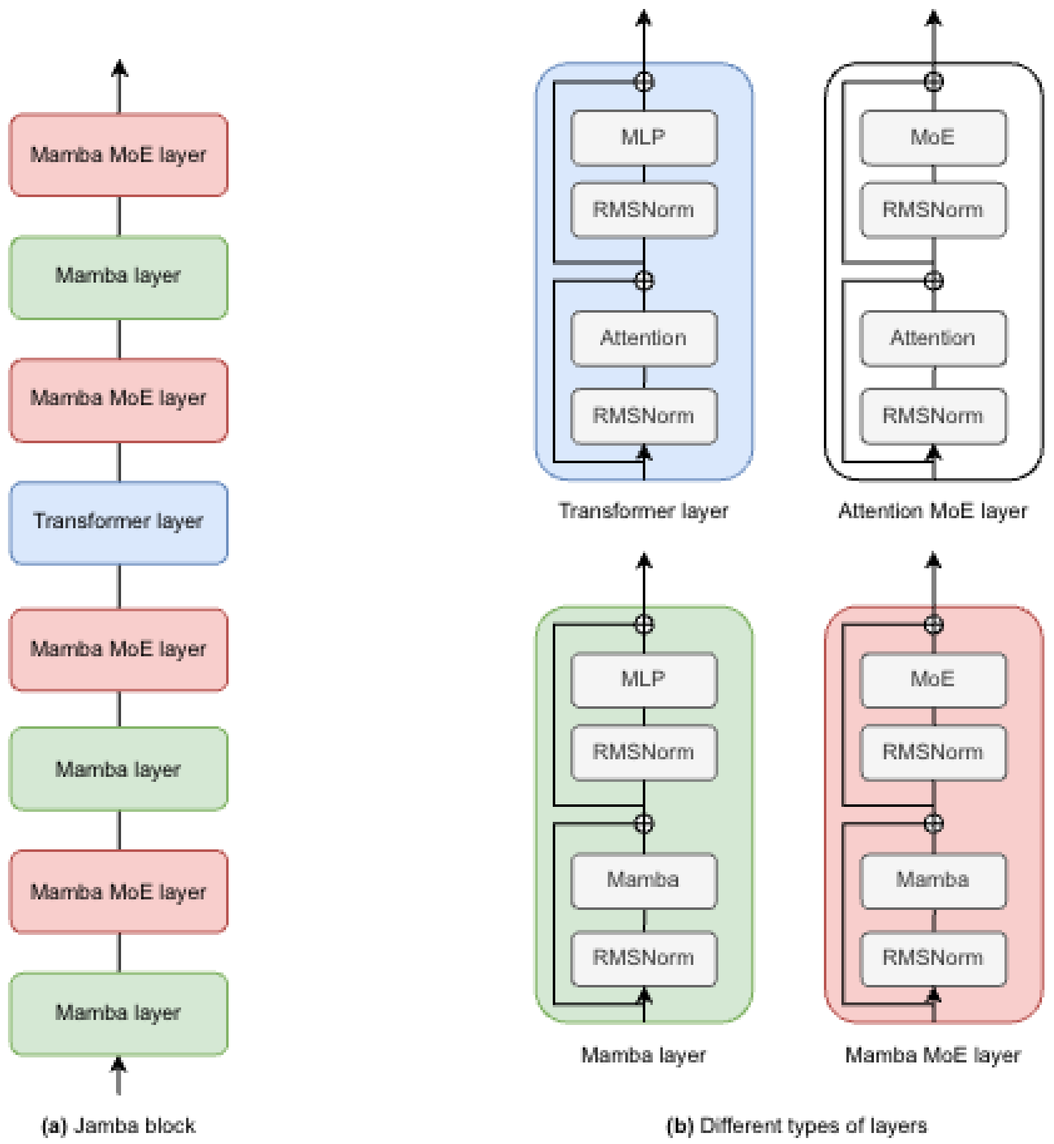
Linearity Strikes back



Mamba: Linear-Time Sequence Modeling with Selective State Spaces (Gu et. Al)

SSM

- SSMs are growing in popularity
- Linear complexity makes them excellent candidates for very long tasks
- If they can begin outperforming transformers in practical settings, these could displace the transformer architecture



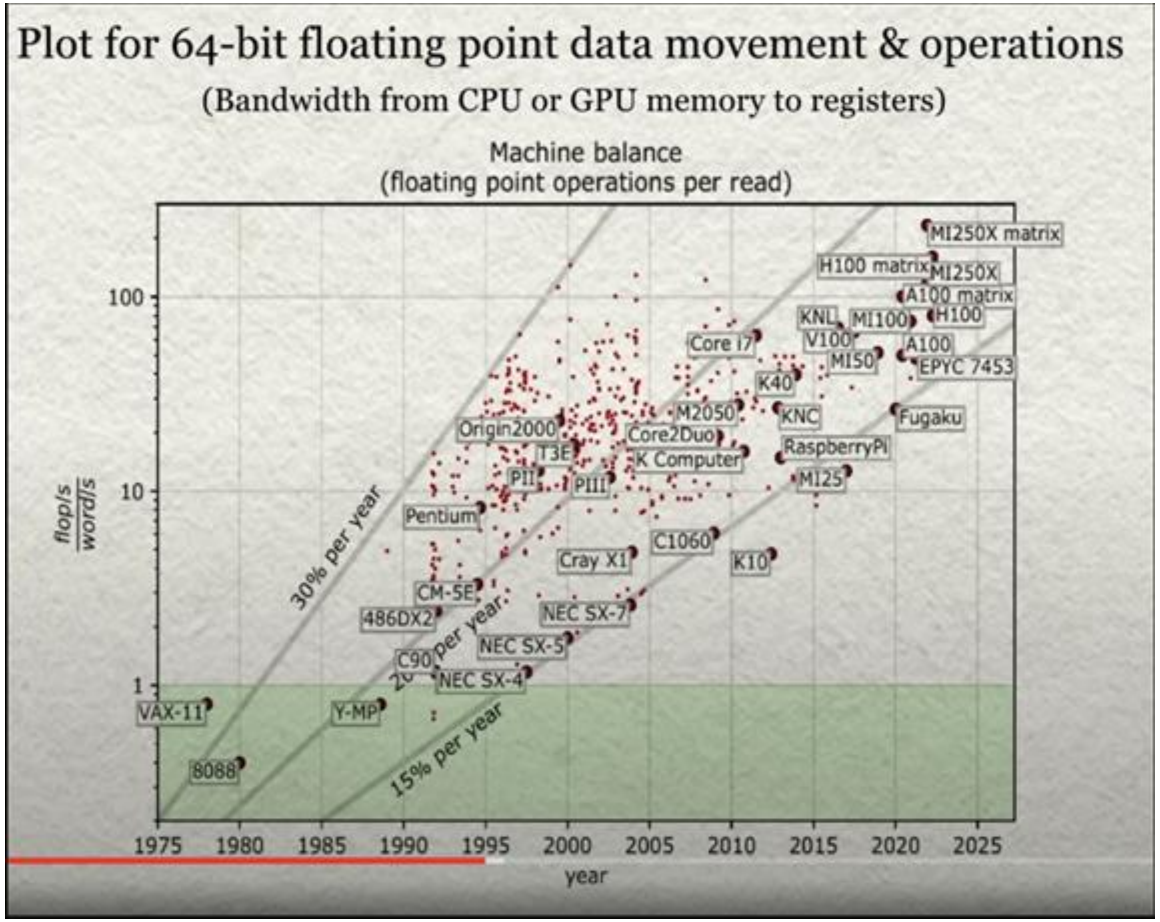
Jamba: A Hybrid Transformer-Mamba Language Model

Efficient LLMs

- ❑ Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- ❑ Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- ❑ Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- ❑ **Speculative Decoding**
- ❑ Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- ❑ Distillation

The Memory Bandwidth Bottleneck

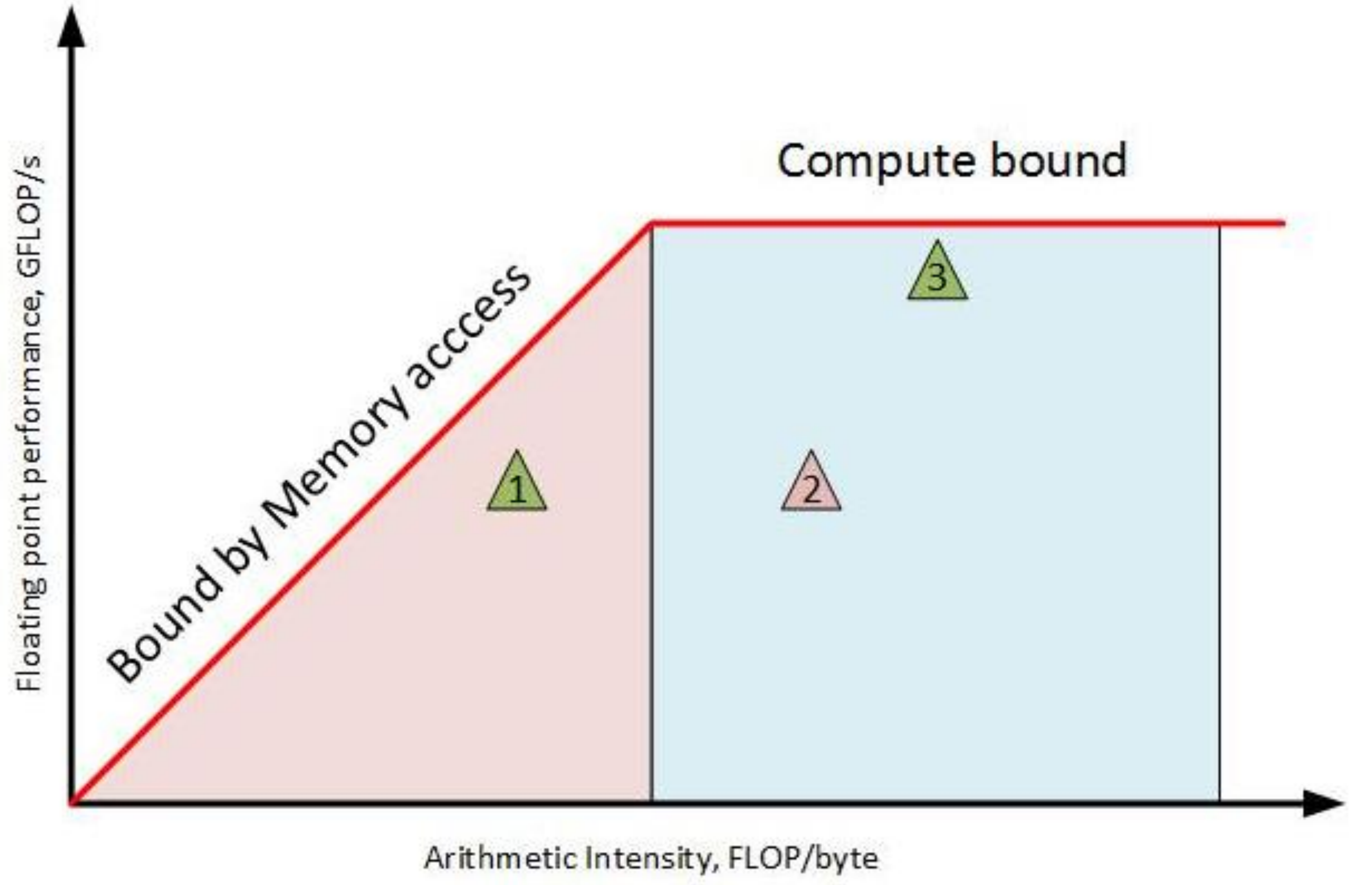
- Two parts of computing each add time to any given task →
  - Memory loading (Gb/s)
  - Computation (FLOPS)
- Over time, memory loading has gotten slower relative to computation
- This means memory loading can be more of a bottleneck if we are only using things we load from memory one time



[Turing Award Presentation, 2021](#) [Dongarra]

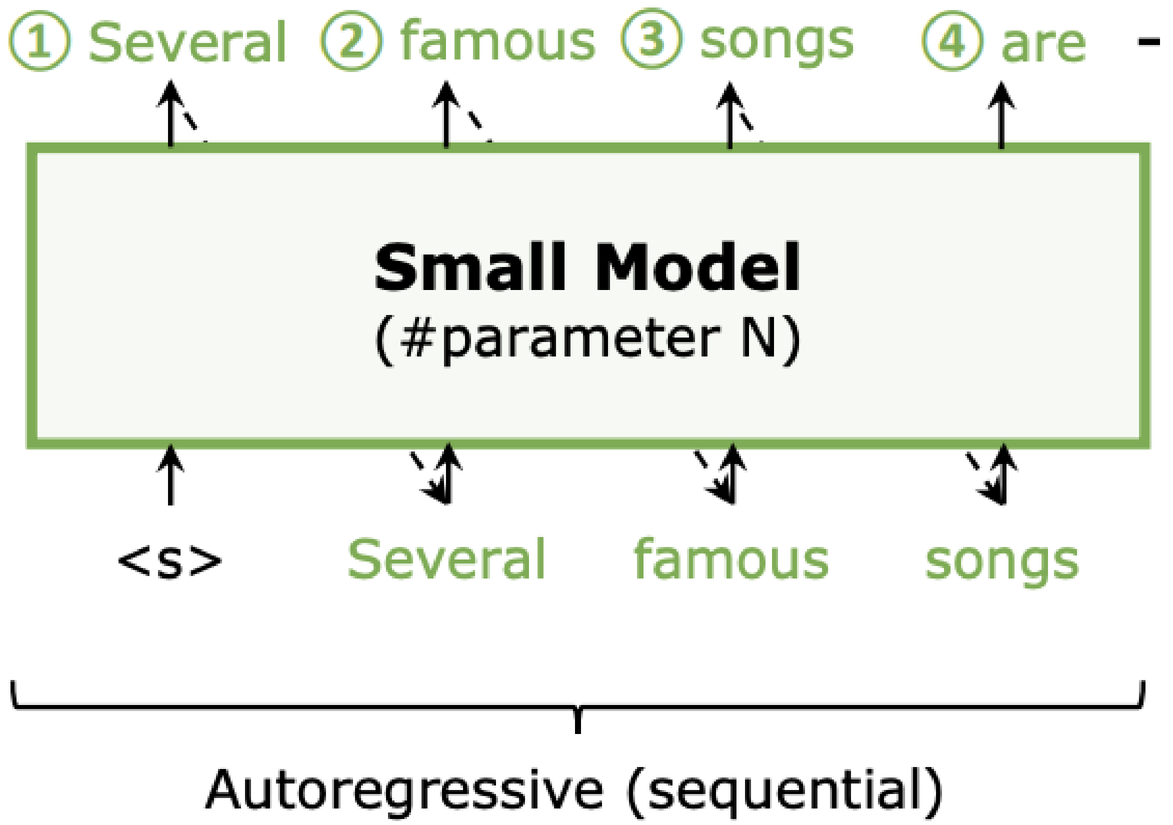
Compute vs. IO

- One way to alleviate this is to increase the amount of computation we perform for each byte we load from memory
- This is called the ***arithmetic intensity*** of a given program/function
- Generally, we would prefer to be in the compute bound region more often as this is where work is being done



Speculative Decoding

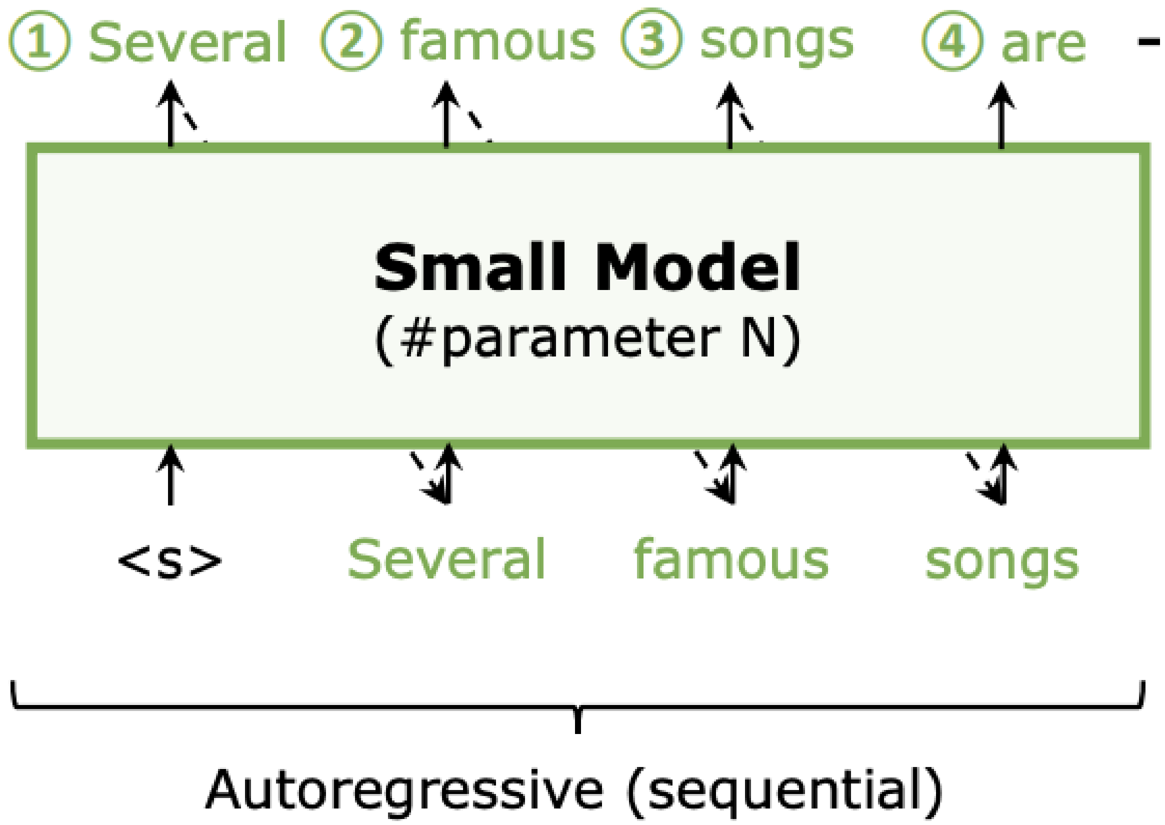
- In standard autoregressive decoding, we are only using each parameter ***one time*** when the batch size is 1
- This means standard decoding has a ***low*** arithmetic intensity and is memory bound
- We have a bunch more compute we could be getting for free given how massively parallel GPUs are



Big Little Decoder [Kim, et. al]

Speculative Decoding

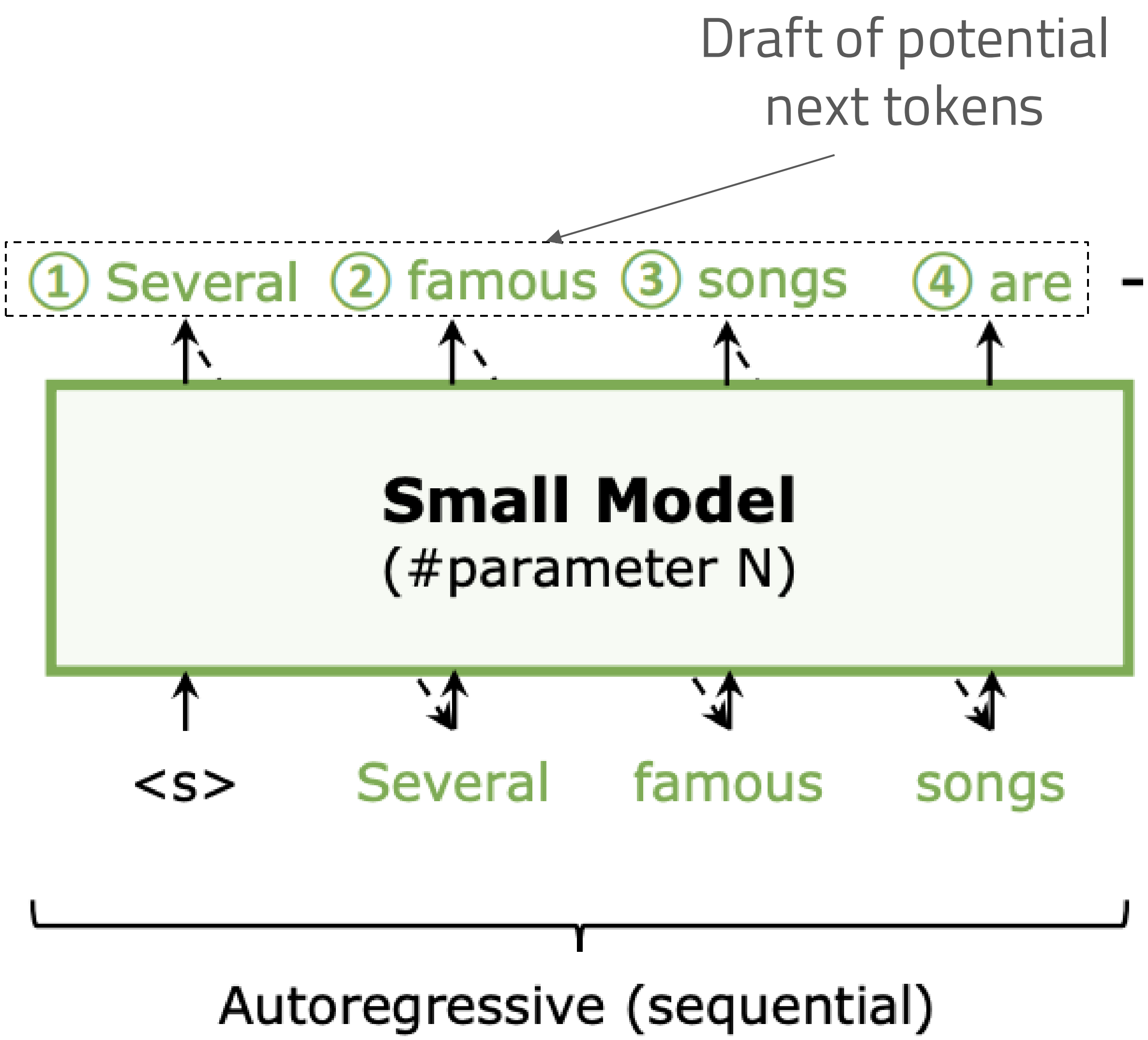
- Speculative decoding resolves this by 'speculating' multiple tokens into the future with a smaller, cheaper model



Big Little Decoder [Kim, et. al]

Speculative Decoding

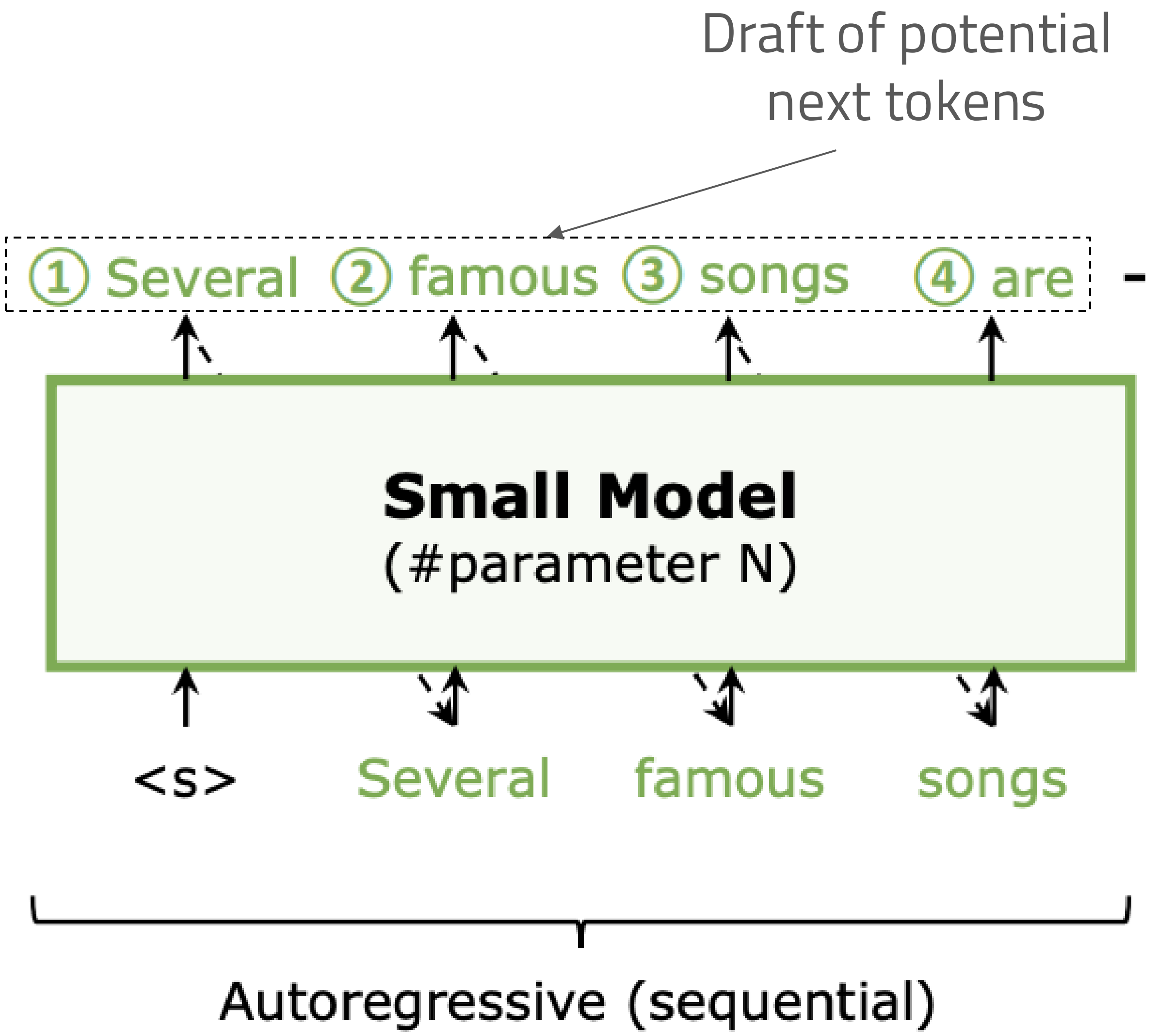
- Speculative decoding resolves this by 'speculating' multiple tokens into the future with a smaller, cheaper model



Big Little Decoder [Kim, et. al]

Speculative Decoding

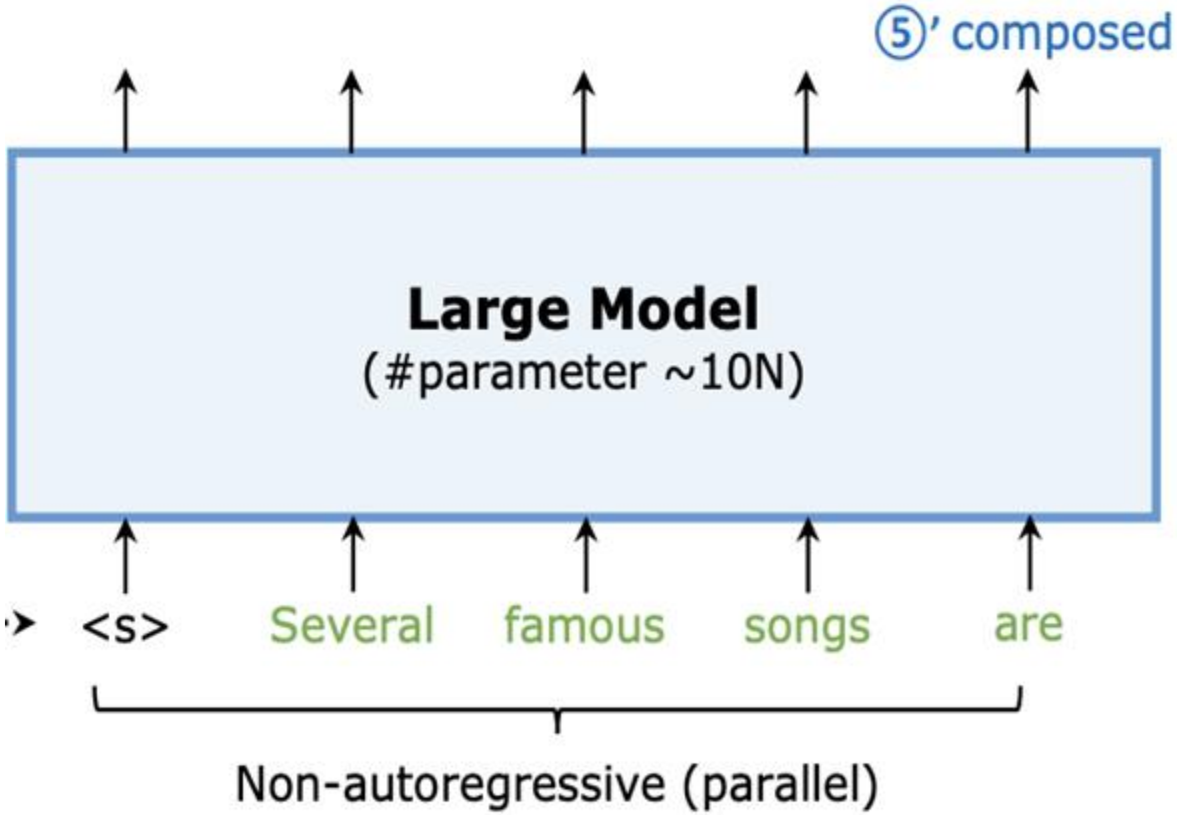
- Speculative decoding resolves this by 'speculating' multiple tokens into the future with a smaller, cheaper model
- We can now send this set of tokens on to a much larger model to verify the sequence



Big Little Decoder [Kim, et. al]

Speculative Decoding

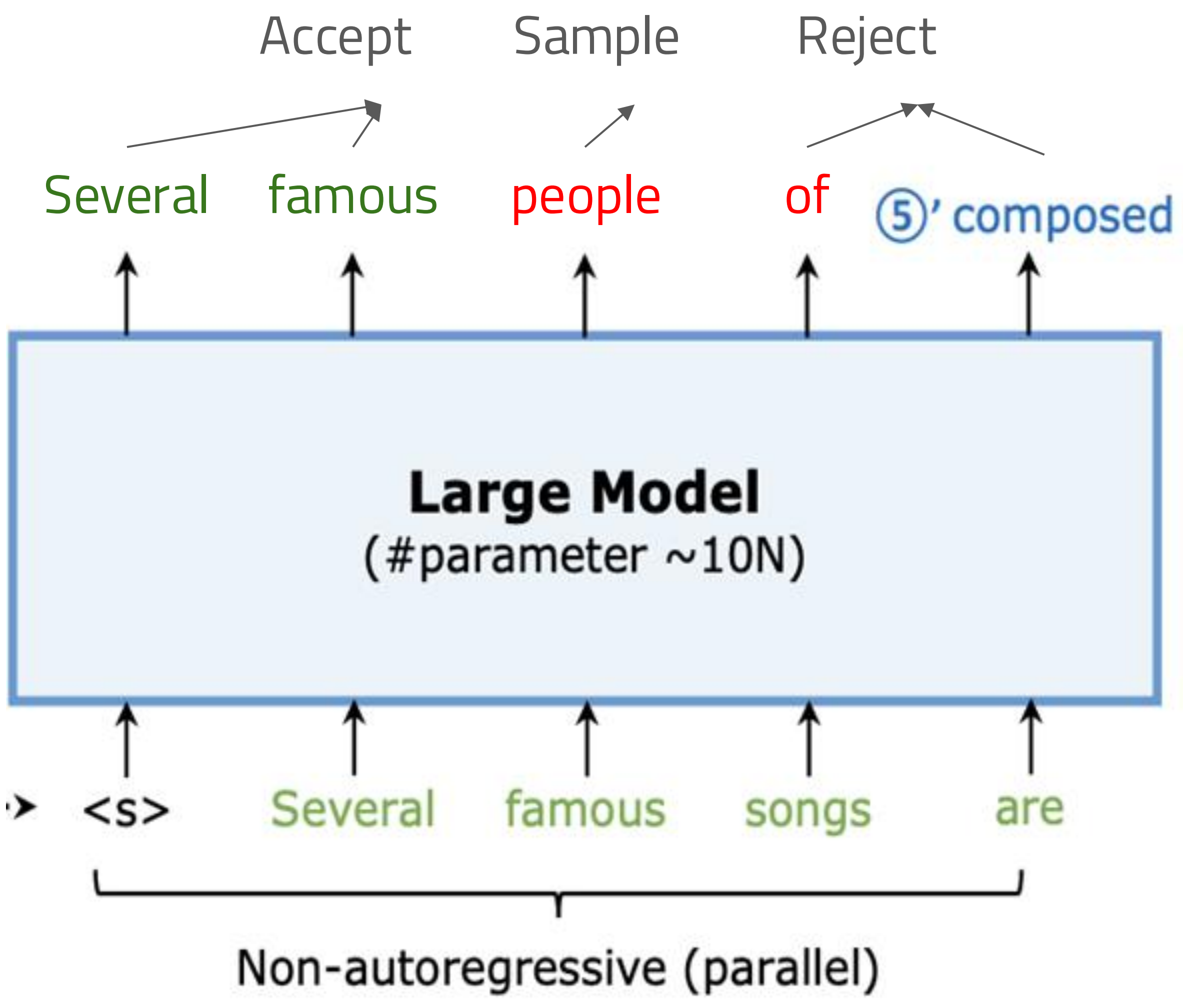
- Because the sequence will be run in parallel the arithmetic intensity will be proportional to the number of draft tokens
- We run each token through and see if the output of the large model matches that of the smaller, draft model



Big Little Decoder [Kim, et. al]

Speculative Decoding

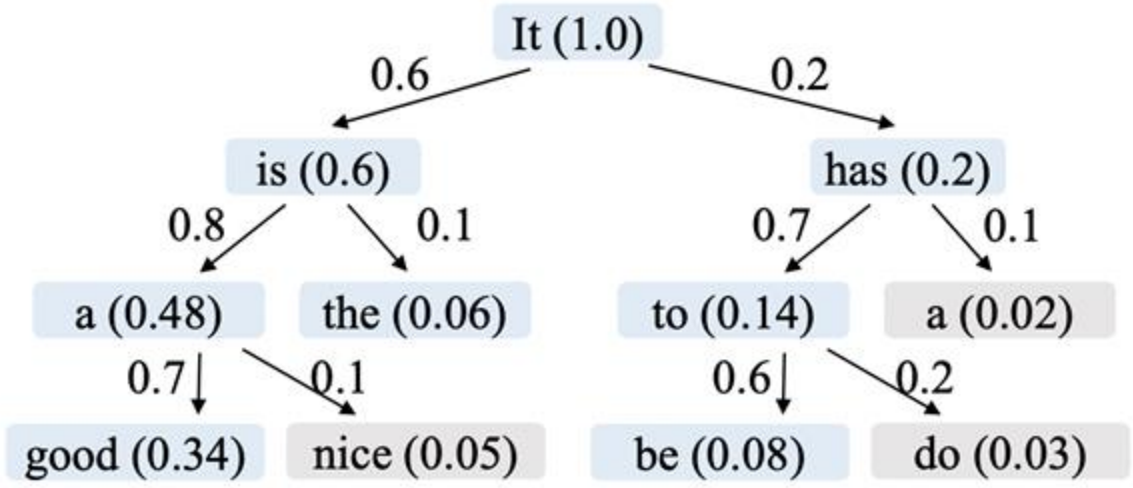
- Because the sequence will be run in parallel the arithmetic intensity will be proportional to the number of draft tokens
- We run each token through and see if the output of the large model matches that of the smaller, draft model
- We accept the matching tokens



Big Little Decoder [Kim, et. al]

Advanced Approaches

More advanced approaches will use draft trees, rather than draft sequences



Attention mask

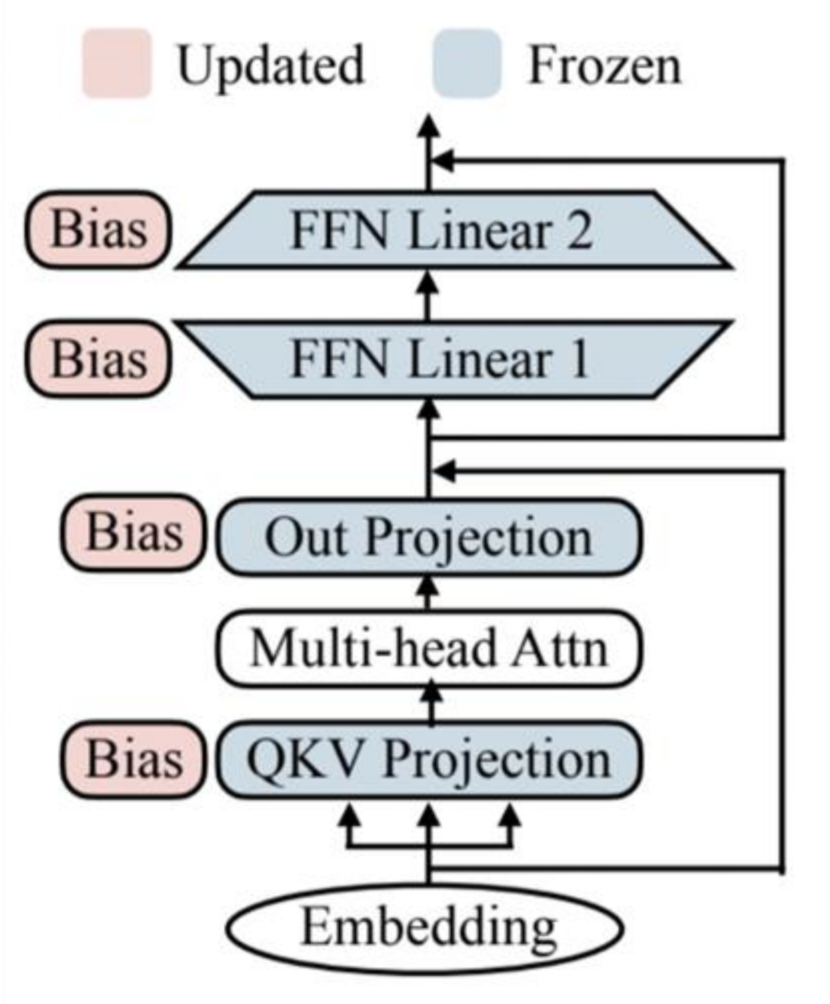
	It	is	has	a	the	to	good	be
It	✓							
is	✓	✓						
has	✓		✓					
a	✓	✓		✓				
the	✓	✓			✓			
to	✓		✓			✓		
good	✓	✓		✓			✓	
be	✓		✓			✓		✓

Big Little Decoder [Kim, et. al]

Efficient LLMs

- Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- **Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)**
- Distillation

BitFit



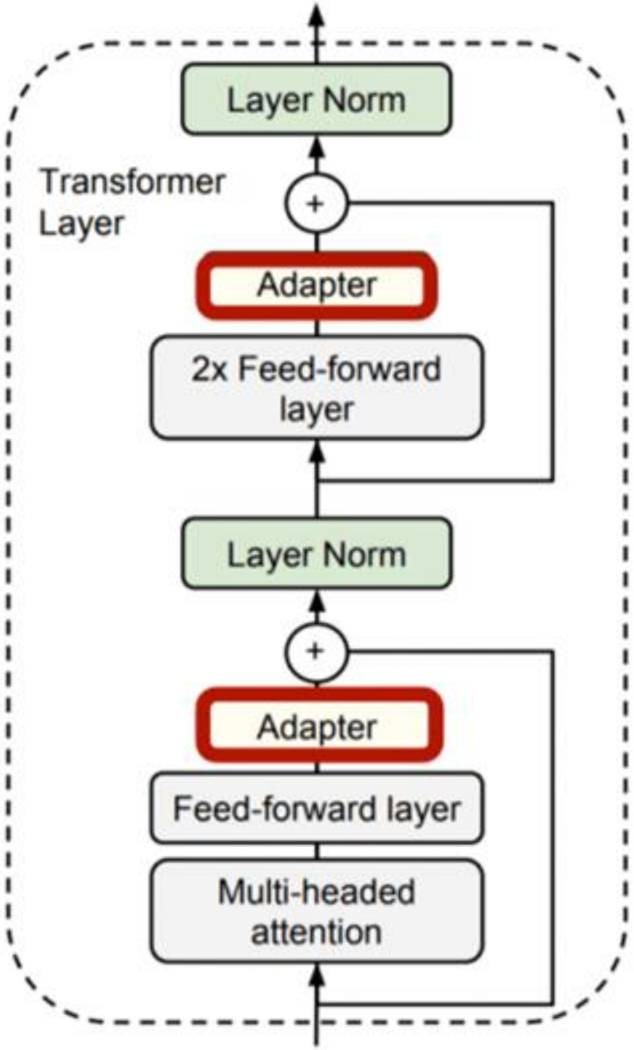
Update only the bias parameters

		%Param	QNLI	SST-2	MNLI$_m$	MNLI$_{mm}$	CoLA	MRPC	STS-B	RTE	QQP	Avg.
	Train size		105k	67k	393k	393k	8.5k	3.7k	7k	2.5k	364k	
(V)	Full-FT†	100%	**93.5**	**94.1**	**86.5**	**87.1**	**62.8**	**91.9**	89.8	71.8	**87.6**	**84.8**
(V)	Full-FT	100%	91.7±0.1	93.4±0.2	85.5±0.4	85.7±0.4	62.2±1.2	90.7±0.3	**90.0±0.4**	**71.9±1.3**	87.5±0.4	84.1
(V)	Diff-Prune†	0.5%	**93.4**	**94.2**	**86.4**	**86.9**	63.5	91.3	89.5	71.5	**86.6**	**84.6**
(V)	BitFit	0.08%	91.4±2.4	93.2±0.4	84.4±0.2	84.8±0.1	**63.6±0.7**	**91.7±0.5**	**90.3±0.1**	**73.2±3.7**	85.4±0.1	84.2
(T)	Full-FT‡	100%	91.1	**94.9**	86.7	85.9	**60.5**	**89.3**	**87.6**	70.1	**72.1**	**81.8**
(T)	Full-FT†	100%	**93.4**	94.1	86.7	**86.0**	59.6	88.9	86.6	**71.2**	71.7	81.5
(T)	Adapters‡	3.6%	90.7	94.0	84.9	85.1	59.5	89.5	**86.9**	71.5	**71.8**	81.1
(T)	Diff-Prune†	0.5%	**93.3**	94.1	**86.4**	**86.0**	**61.1**	**89.7**	86.0	70.6	71.1	**81.5**
(T)	BitFit	0.08%	92.0	**94.2**	84.5	84.8	59.7	88.9	85.5	**72.0**	70.5	80.9

Table 1: BERT$_{\text{LARGE}}$ model performance on the GLUE benchmark validation set (V) and test set (T). Lines with † and ‡ indicate results taken from Guo et al. (2020) and Houlsby et al. (2019) (respectively).

BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models [Zeken et al, ACL 2021]

Adapter



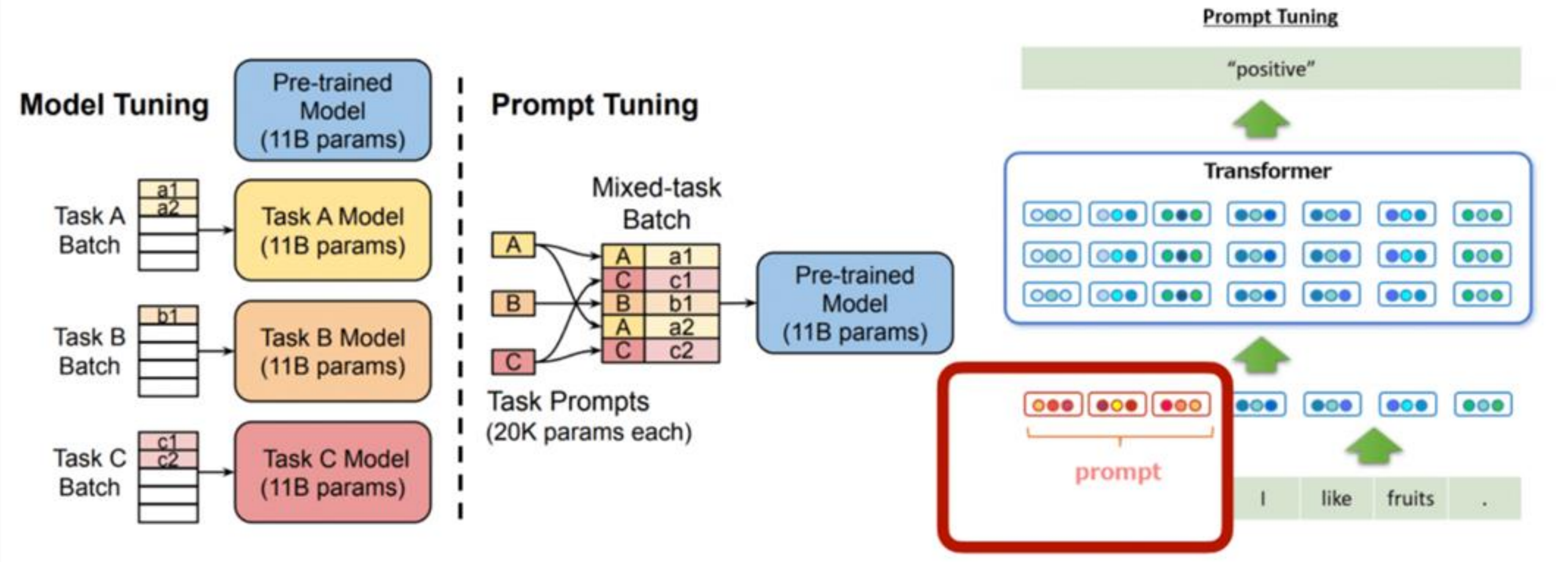
Add trainable layers after each feedforward layer

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_{m}	MNLI_{mm}	QNLI	RTE	Total
$\text{BERT}_{\text{LARGE}}$	9.0×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	1.3×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	1.2×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

Parameter-Efficient Transfer Learning for NLP [Houlsby et al, ICML 2019]

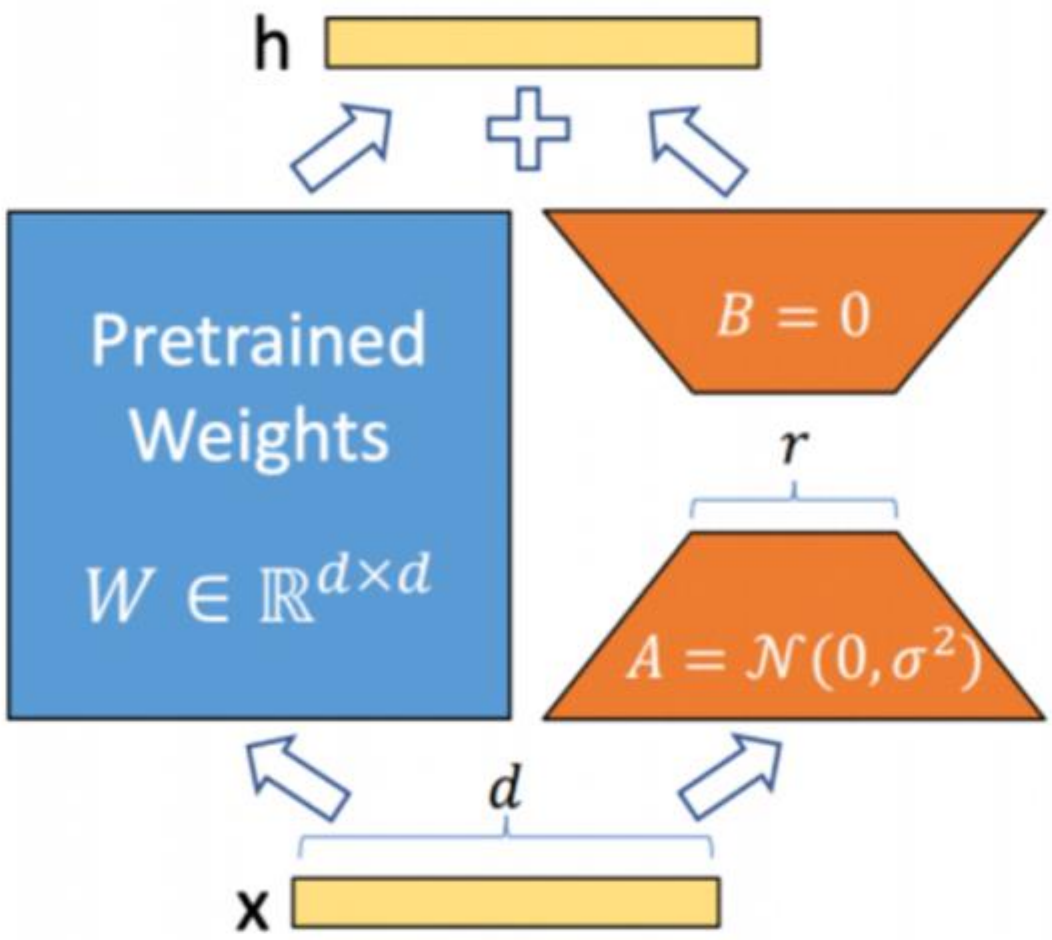
Prompt Tuning (Soft Prompting)

Train a continuous, learnable prompt in embedding space for each task we are training on



The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

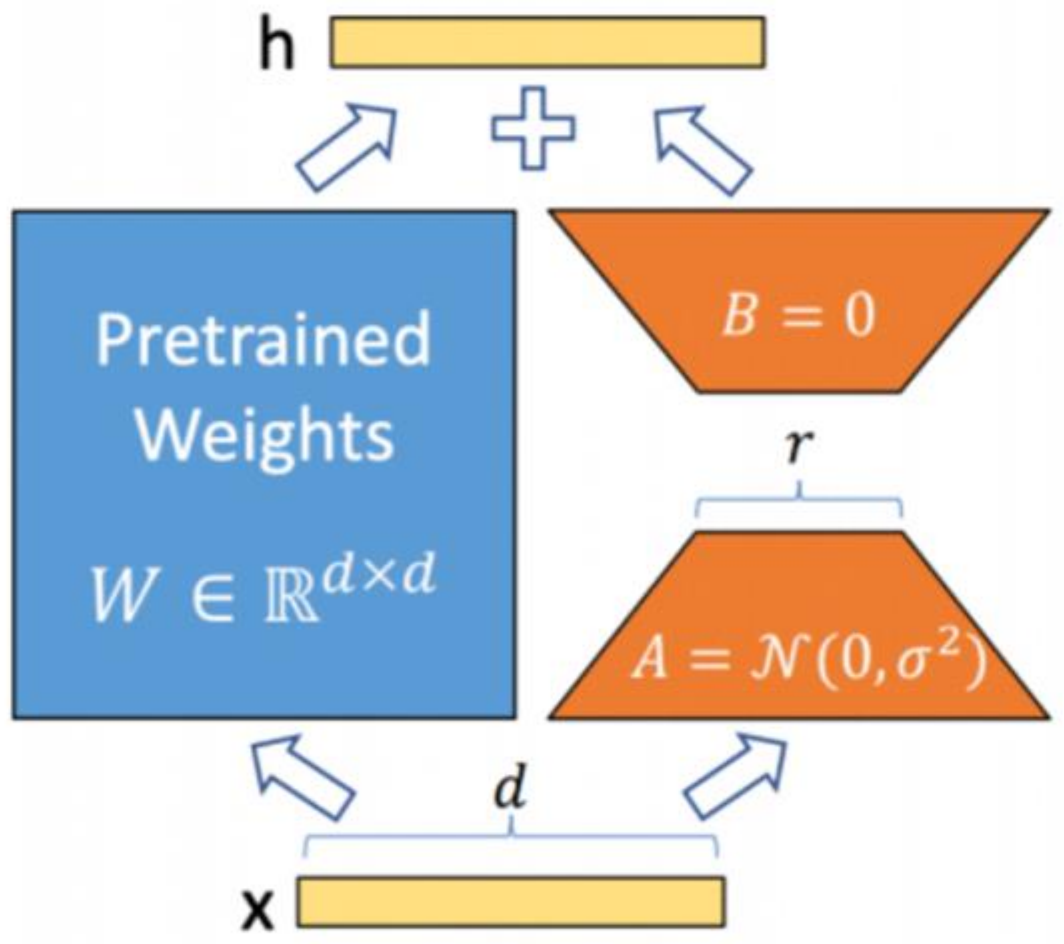
LoRA



- Hypothesizes that fine-tuning results in only low rank updates
- Thus, we may approximate the updates themselves as low-rank and train on this low-rank approximation directly

The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

LoRA



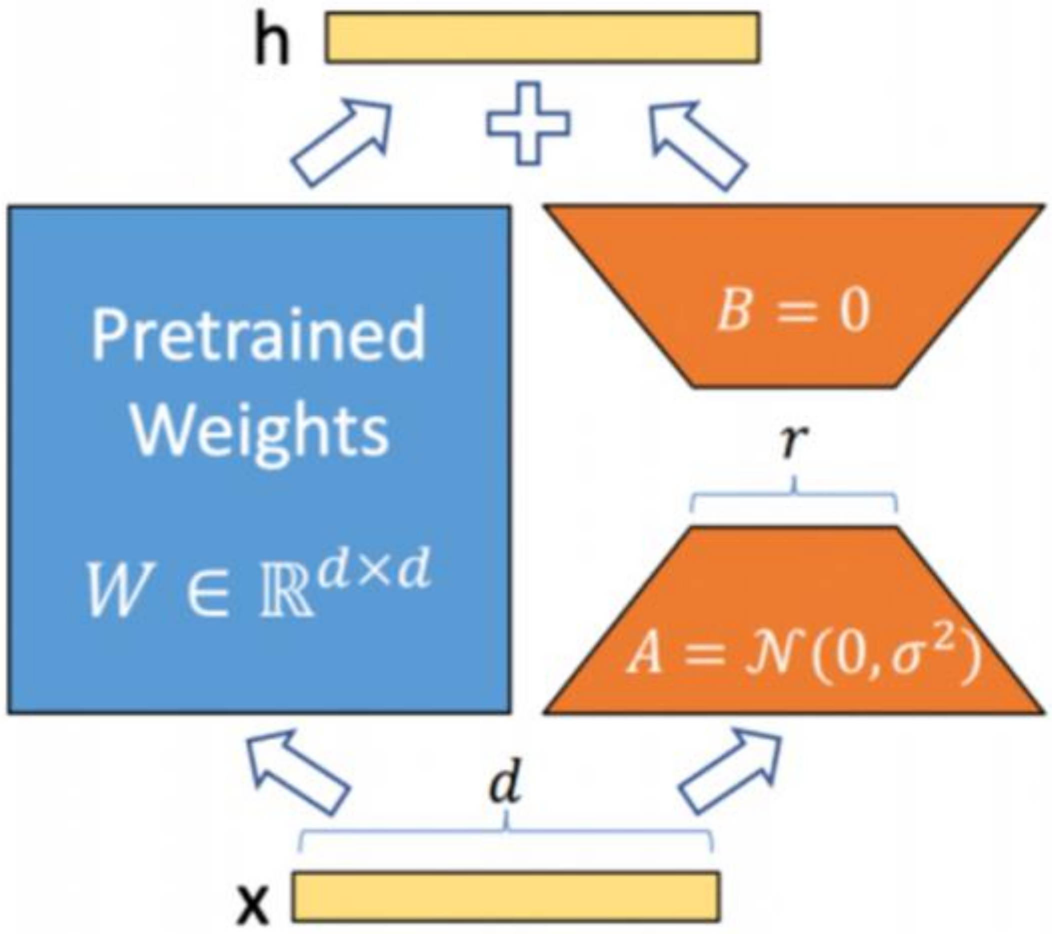
Normal Finetuning:

$\mathbf{h} = \boxed{\mathbf{W}}\mathbf{x}$

Update W

The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

LoRA



The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

Normal Finetuning:

h = $\boxed{\textbf{W}}$**x**

Update W

LoRA Finetuning:

h = **Wx** + $\boxed{\textbf{BA}}$**x**

Update B,A
Leave W unchanged

LoRA

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	87.1$_{\pm.0}$	94.2$_{\pm.1}$	88.5$_{\pm1.1}$	60.8$_{\pm.4}$	93.1$_{\pm.1}$	90.2$_{\pm.0}$	71.5$_{\pm2.7}$	89.7$_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	87.3$_{\pm.1}$	94.7$_{\pm.3}$	88.4$_{\pm.1}$	62.6$_{\pm.9}$	93.0$_{\pm.2}$	90.6$_{\pm.0}$	75.9$_{\pm2.2}$	90.3$_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	87.5$_{\pm.3}$	**95.1**$_{\pm.2}$	89.7$_{\pm.7}$	63.4$_{\pm1.2}$	**93.3**$_{\pm.3}$	90.8$_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	91.6$_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	90.2$_{\pm.3}$	96.1$_{\pm.3}$	90.2$_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	83.8$_{\pm2.9}$	92.1$_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	89.7$_{\pm1.2}$	67.8$_{\pm2.5}$	**94.8**$_{\pm.3}$	91.7$_{\pm.2}$	80.1$_{\pm2.9}$	91.9$_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	89.9$_{\pm.5}$	96.2$_{\pm.3}$	88.7$_{\pm2.9}$	66.5$_{\pm4.4}$	94.7$_{\pm.2}$	92.1$_{\pm.1}$	83.4$_{\pm1.1}$	91.0$_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	90.3$_{\pm.3}$	96.3$_{\pm.5}$	87.7$_{\pm1.7}$	66.3$_{\pm2.0}$	94.7$_{\pm.2}$	91.5$_{\pm.1}$	72.9$_{\pm2.9}$	91.5$_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.2**$_{\pm1.0}$	68.2$_{\pm1.9}$	**94.8**$_{\pm.3}$	91.6$_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	**91.9**$_{\pm.2}$	96.9$_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

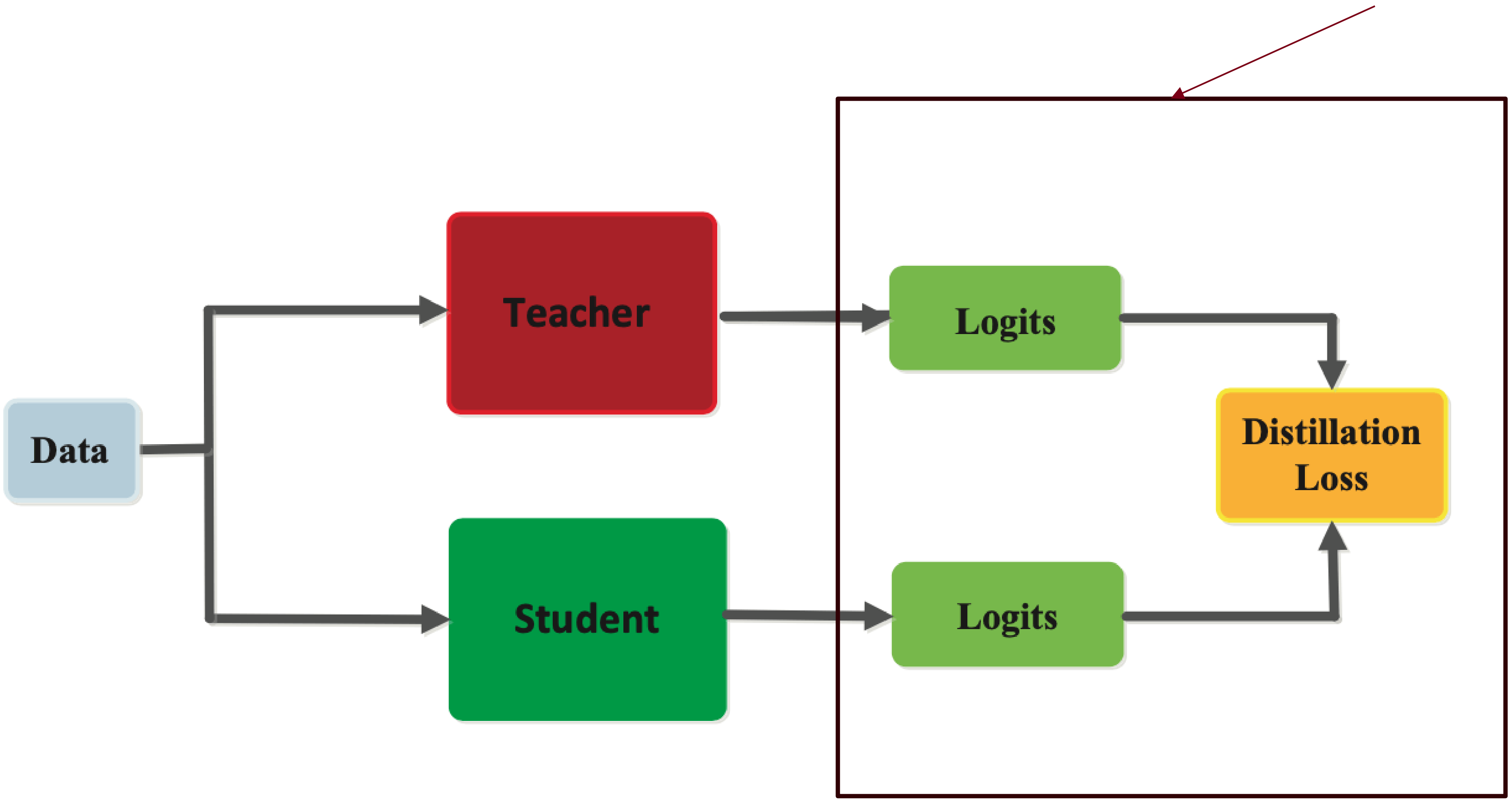
The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

Efficient LLMs

- Quantization
  - Background
  - K-Means vs. Linear Quantization
  - Quantization Granularity
  - Quantization Aware Training (QAT) vs Post-Training Quantization (PTQ)
  - LLM Quantization (LLM.int8(), SmoothQuant, AWQ, 1-bit LLMs)
- Sparsity (Mixture of Experts, Deja Vu: Contextual Sparsity)
- Long Context (PagedAttention, StreamingLLM, MHA/GQA/MQA, H_2O, SSM)
- Speculative Decoding
- Parameter Efficient Fine-Tuning (BitFit, Adapter, Prompt Tuning, LoRA)
- **Distillation**

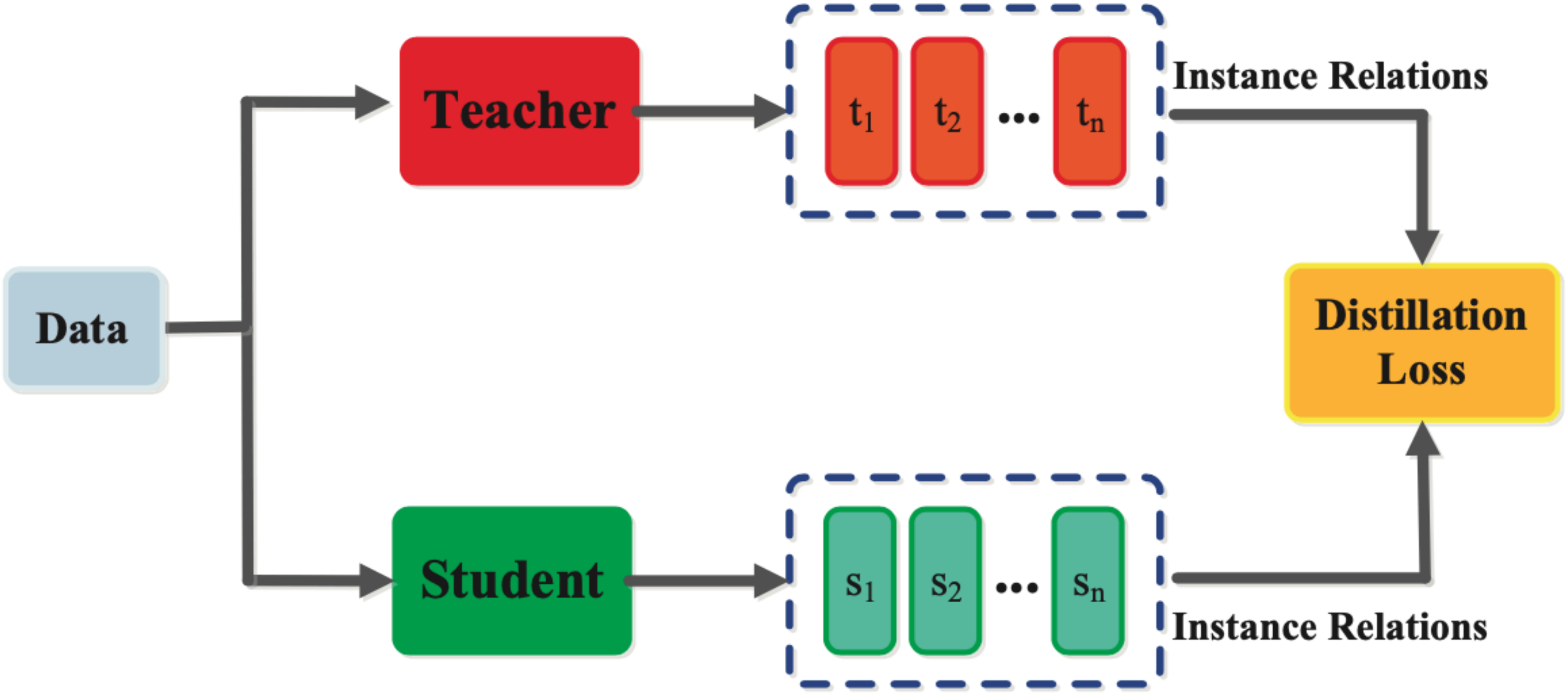
Logit Distillation

Don't use labels, use the modeling distribution of a better 'Teacher' model to train a smaller, 'Student' model

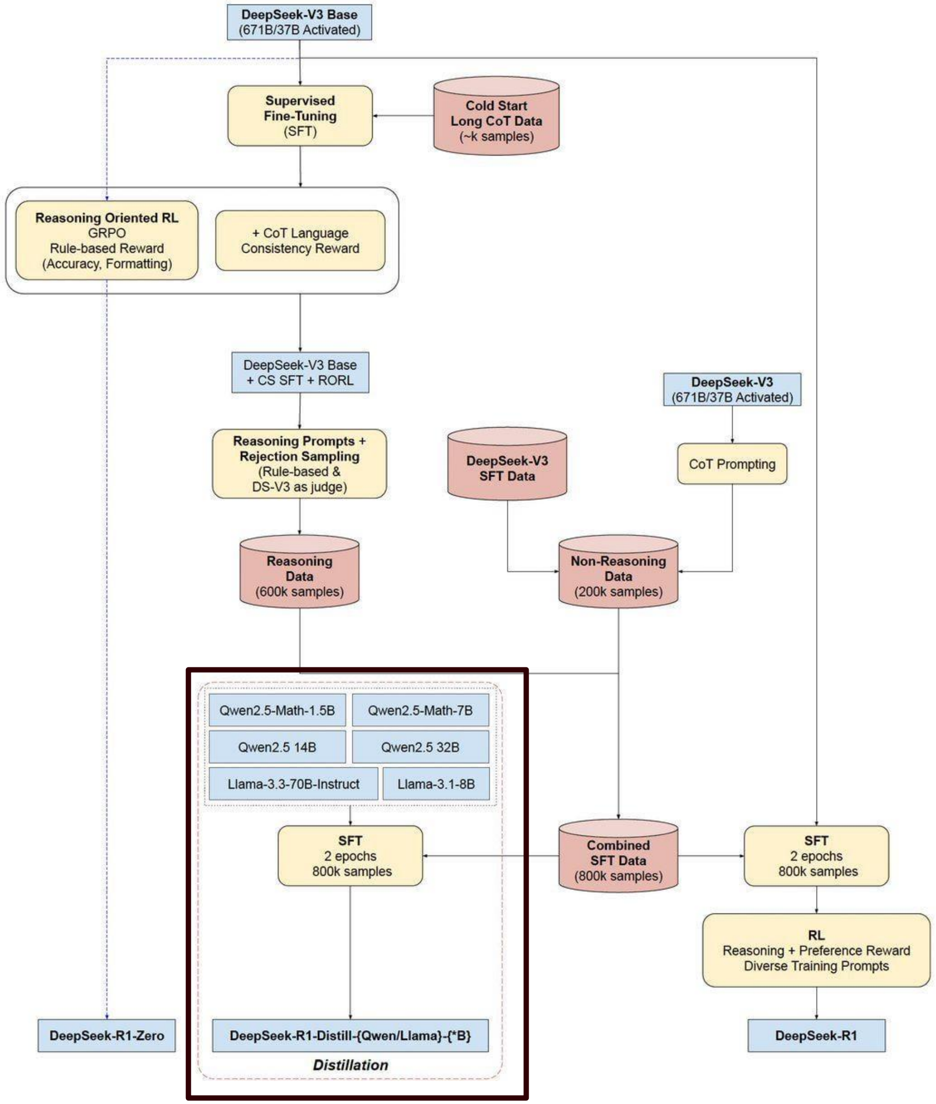


Layer Wise Distillation (LWD)

This can be expanded to trying to match the hidden states of the student and teacher model as well



Output Tokens as Distillation (Synthetic Data)



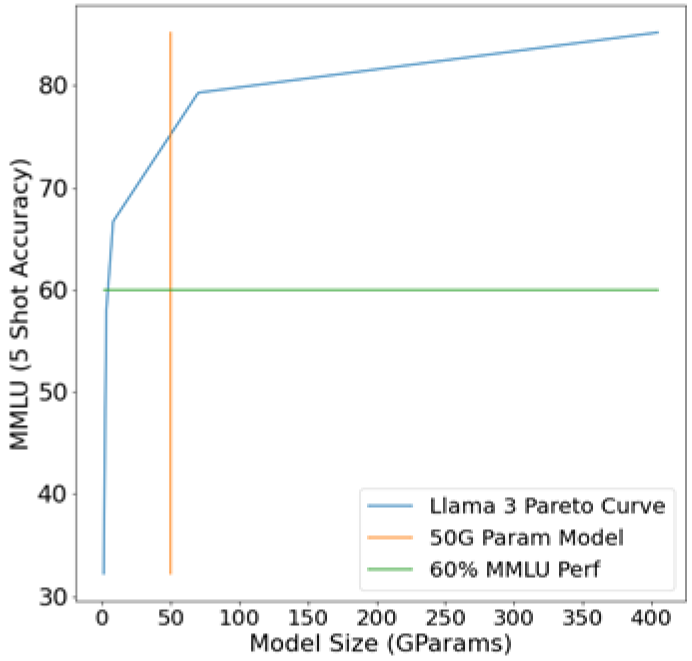
DeepSeekR1 outputs tokens for a given set of problems. In this case 'distillation', just means performing Supervised fine tuning on the tokens create by the larger V3 Models

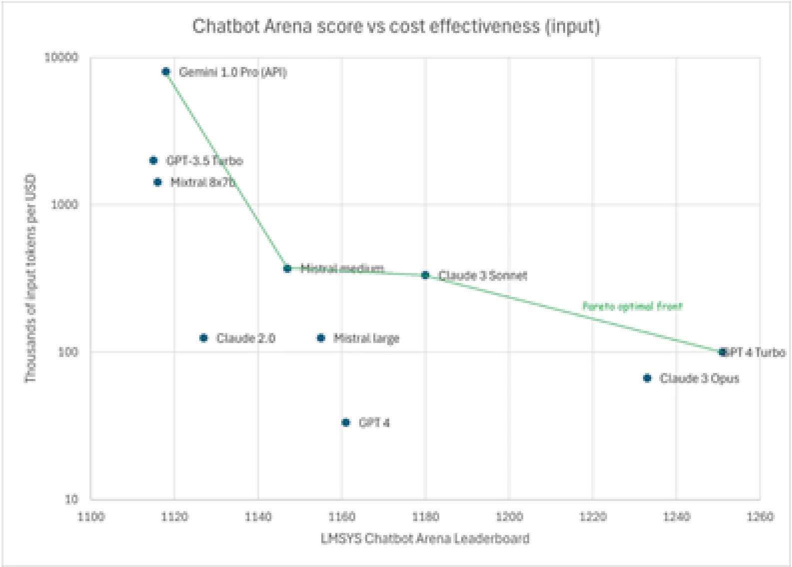
Summary

- Efficient inference algorithms in LLMs lead to lower cost, faster inference, and smaller models
- Quantization and sparsity are the primary techniques for realizing these efficiencies
- PEFT techniques allow for faster fine-tuning with smaller storage requirements

Future Directions

- ❑ Better, more adaptive inference systems
  - ○ Adaptive speculative decoding
  - ○ Variable Model Serving
- ❑ Improved efficiency benchmarking
- ❑ More efficient architectures





Open Source Models/Inference Systems

- Models
  - Llama3.2
  - Qwen2.5
  - Mixtral
- Quantization
  - AWQ
  - LLM.int8()
  - QLoRA
  - GGUF
- Inference Systems
  - vLLM
  - SGLang
  - Tensor-RT LLM
  - Llama.cpp
  - oLLama
  - Huggingface TGI