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®

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 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

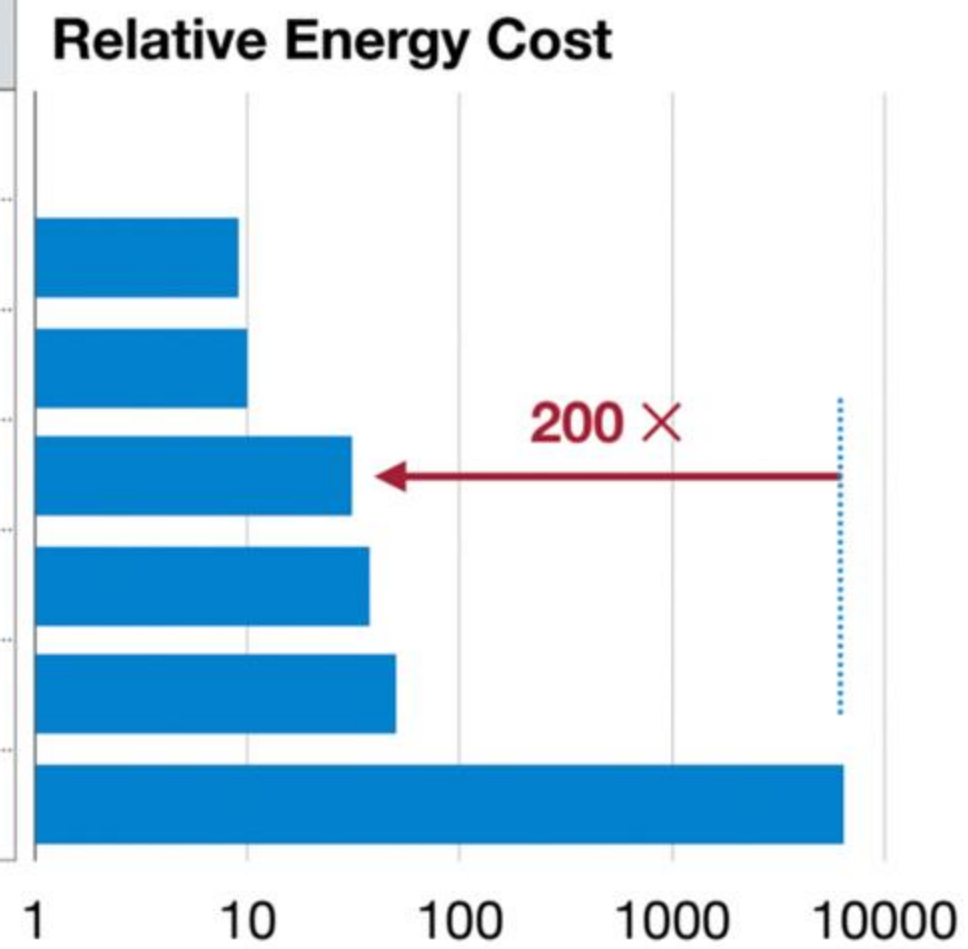
What Is Efficiency and Why Does It Matter?

- 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 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

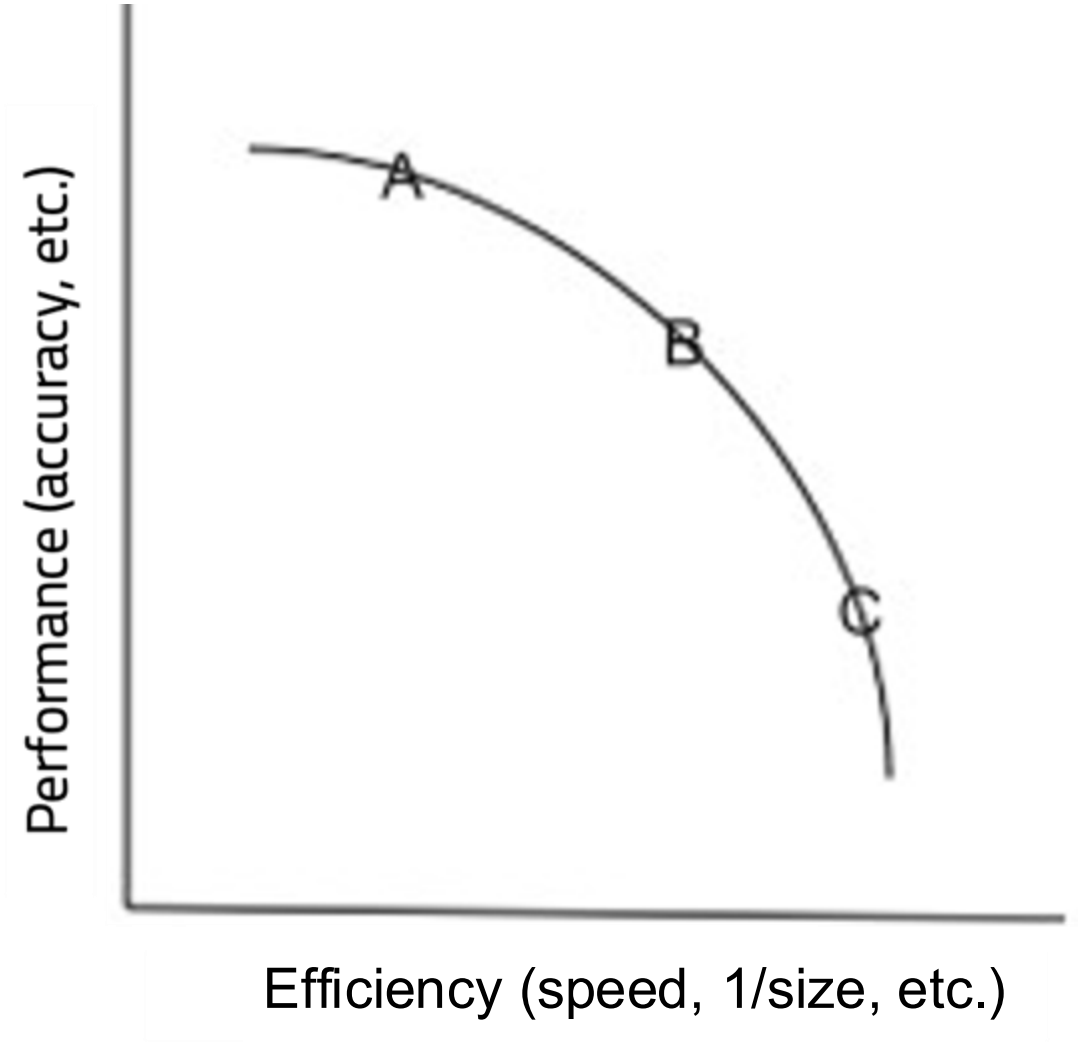
Rough Energy Cost For Various Operations in 45nm 0.9V



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

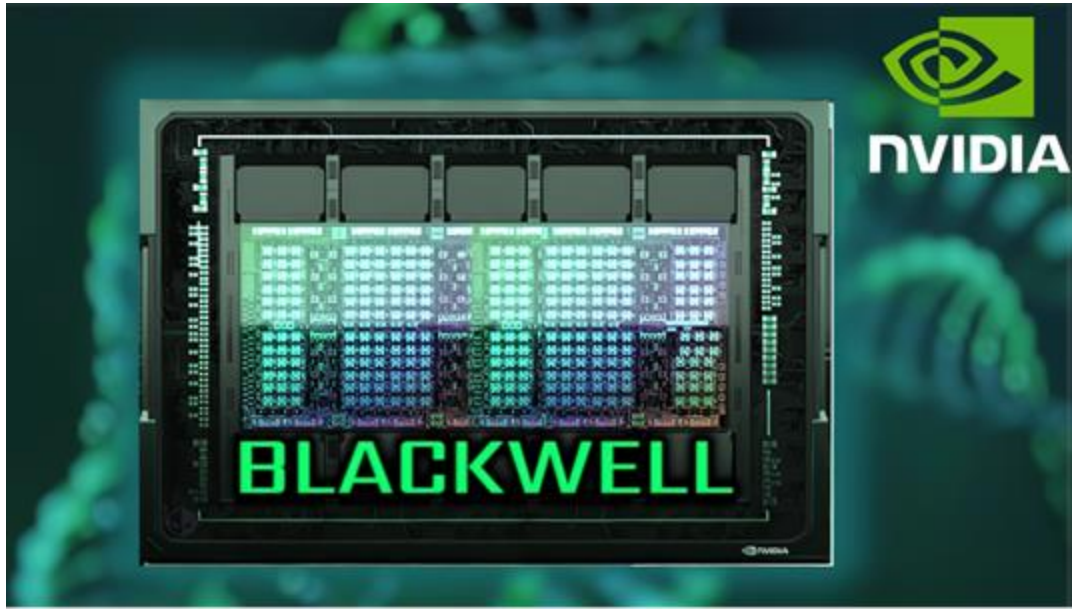
Efficiency Tradeoff

- 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

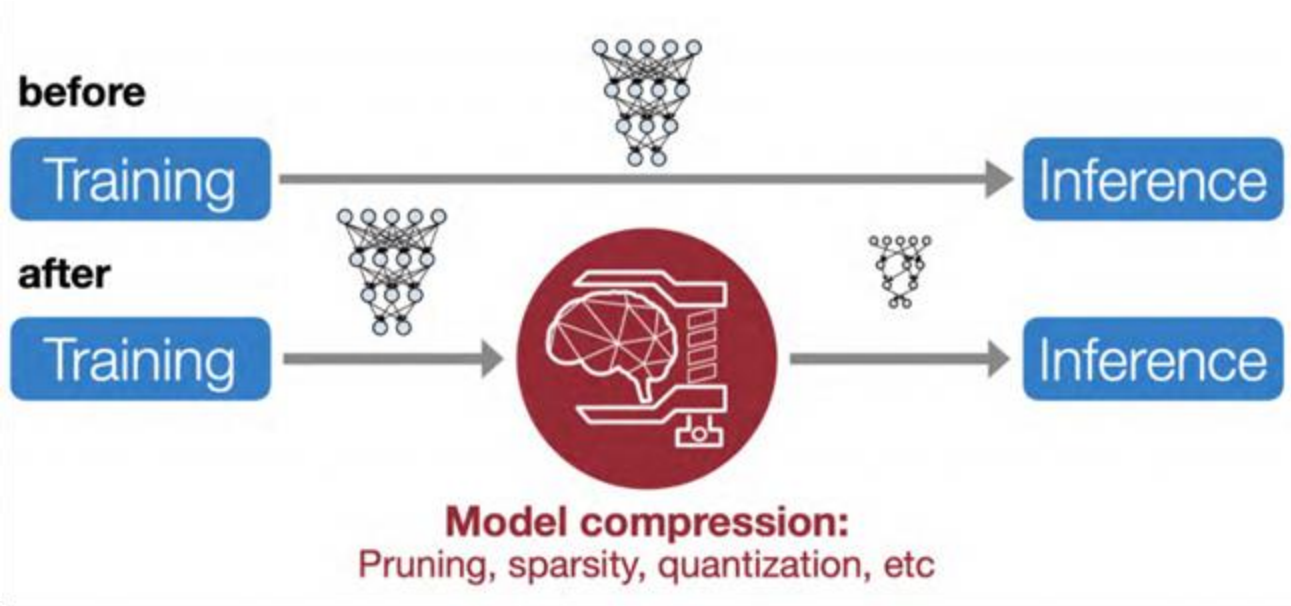


How to Improve Model Efficiency?

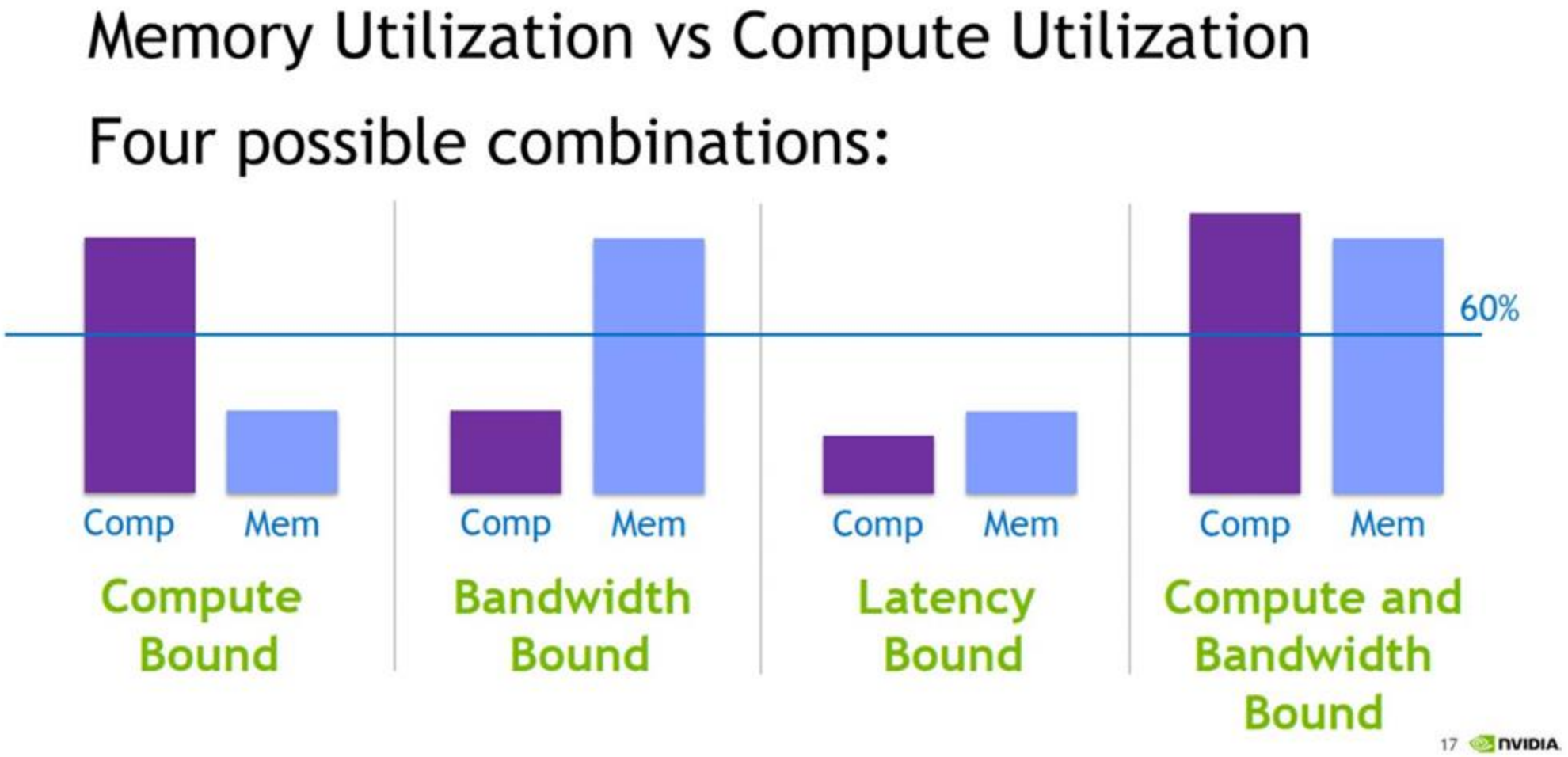
Hardware



Software



What Makes a Language Model Slow



Efficient LLMs

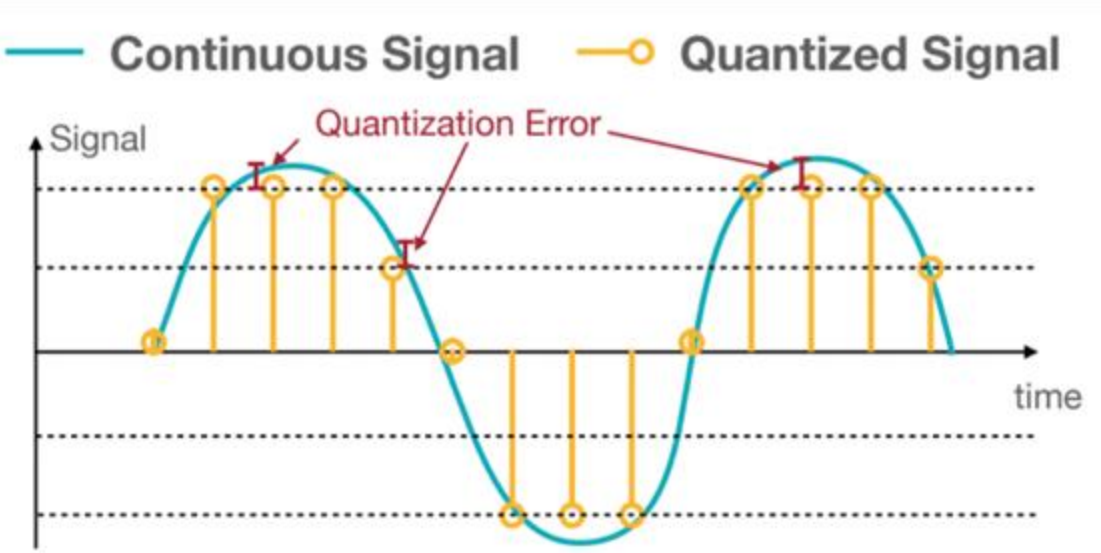
- ❑ Quantization
- ❑ Sparsity
- ❑ Long Context
- ❑ Serving & Systems
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

Efficient LLMs

- ❑ **Quantization**
- ❑ Sparsity
- ❑ Long Context
- ❑ Serving & Systems
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

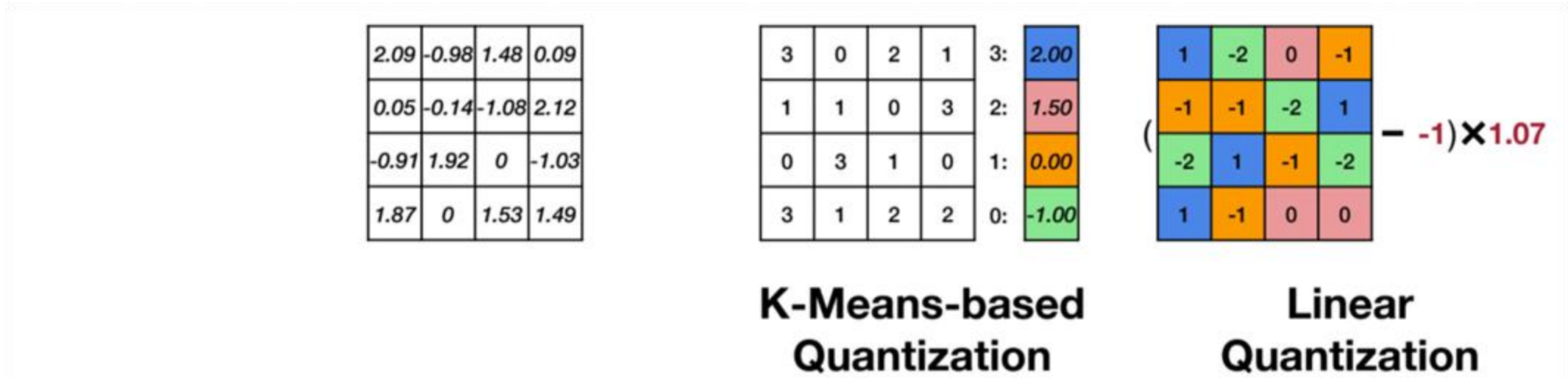
Quantization

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



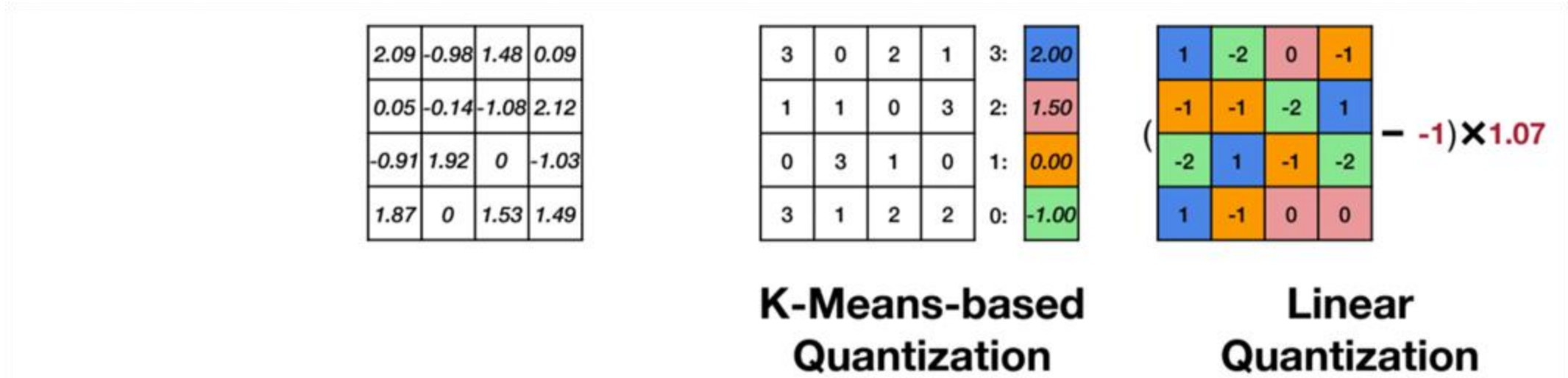
How do we go from a high-bit width data type to a low-bit width data type?

K-Means Quantization vs Linear Quantization



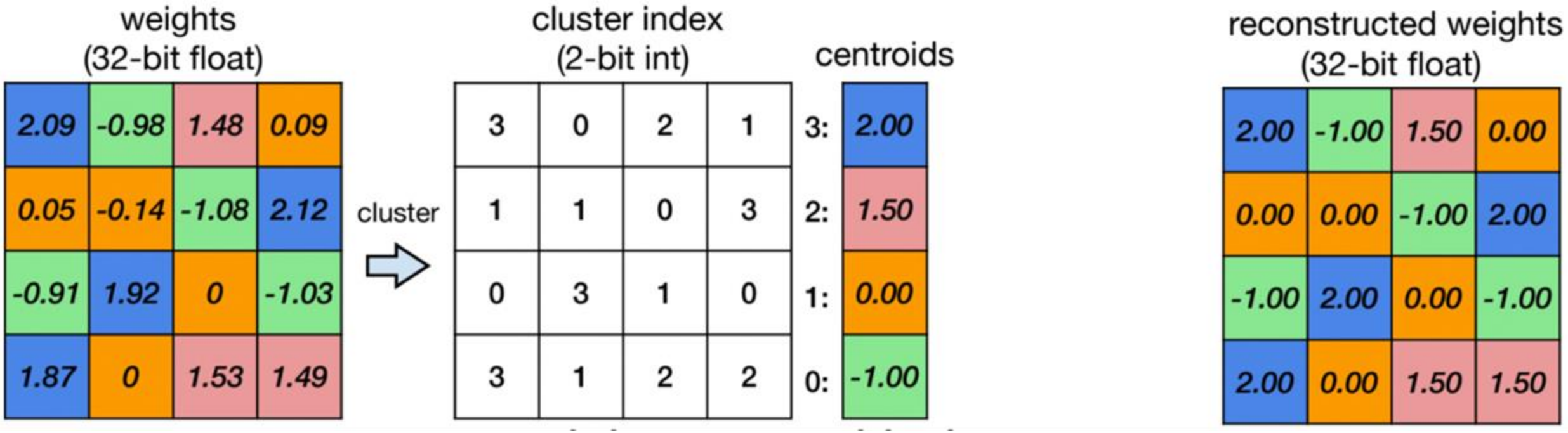
	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Storage			
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

K-Means Quantization vs Linear Quantization



	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Storage			
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

K-Means Quantization



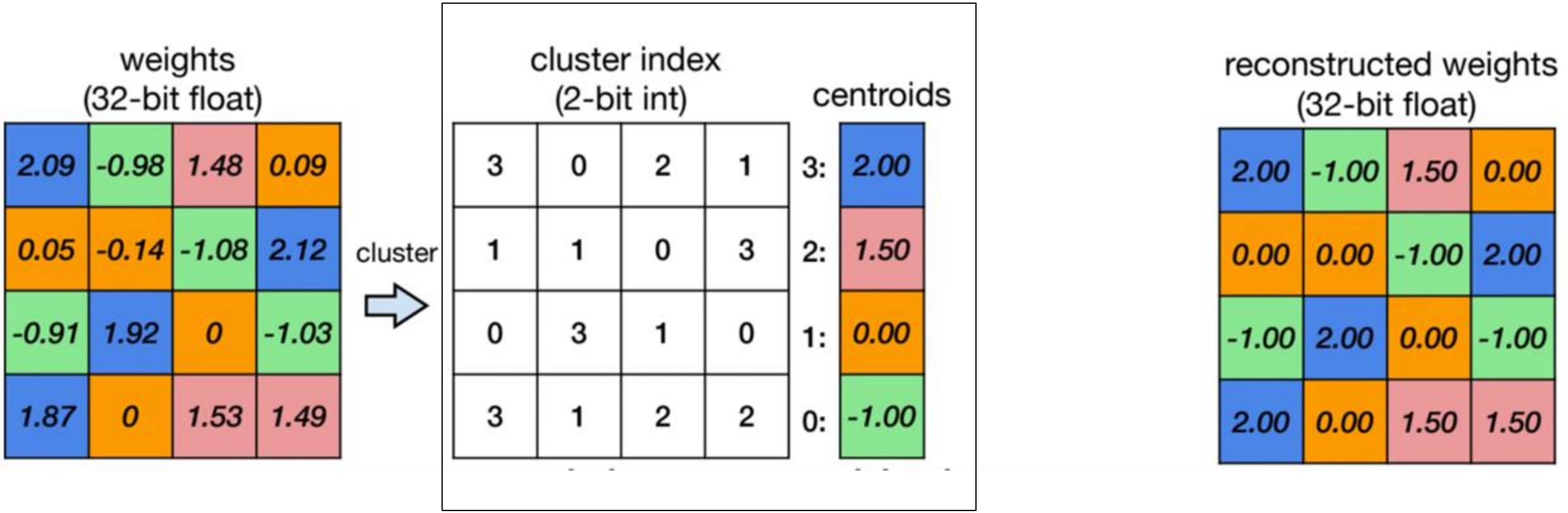
Deep Compression [Han et al., ICLR 2016]

K-Means Quantization

Deep Compression [Han et al., ICLR 2016]

K-Means Quantization

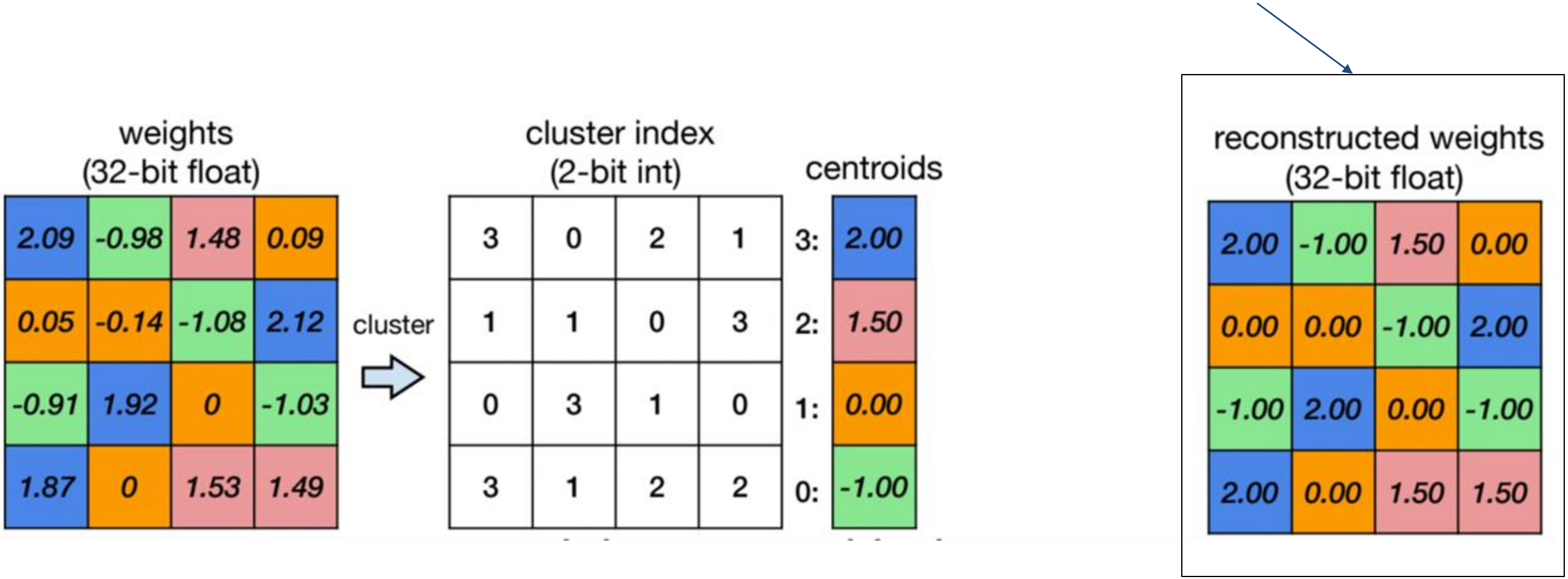
Stored weights after clustering



Deep Compression [Han et al., ICLR 2016]

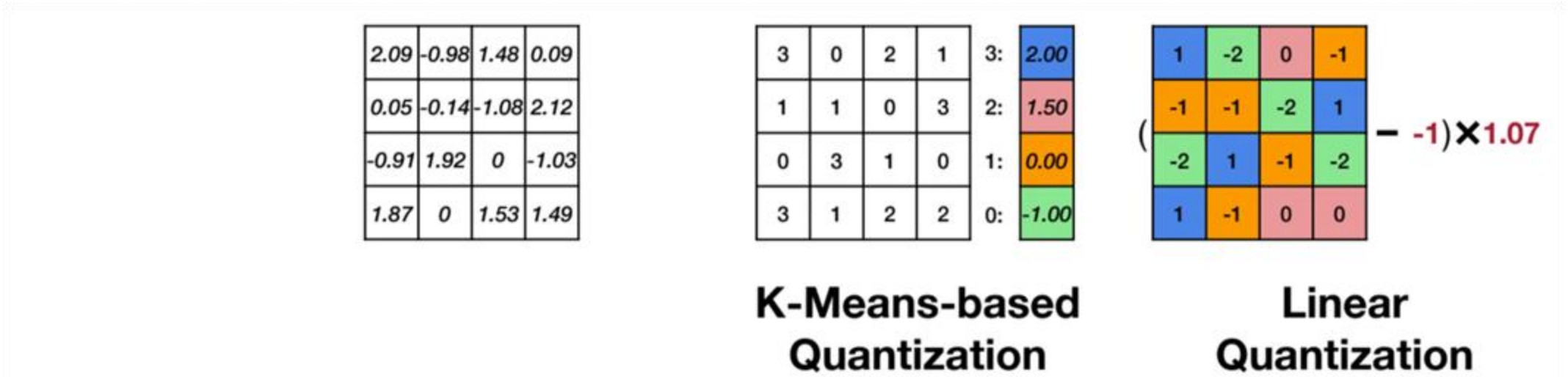
K-Means Quantization

Retrieved weights to be used at inference time



Deep Compression [Han et al., ICLR 2016]

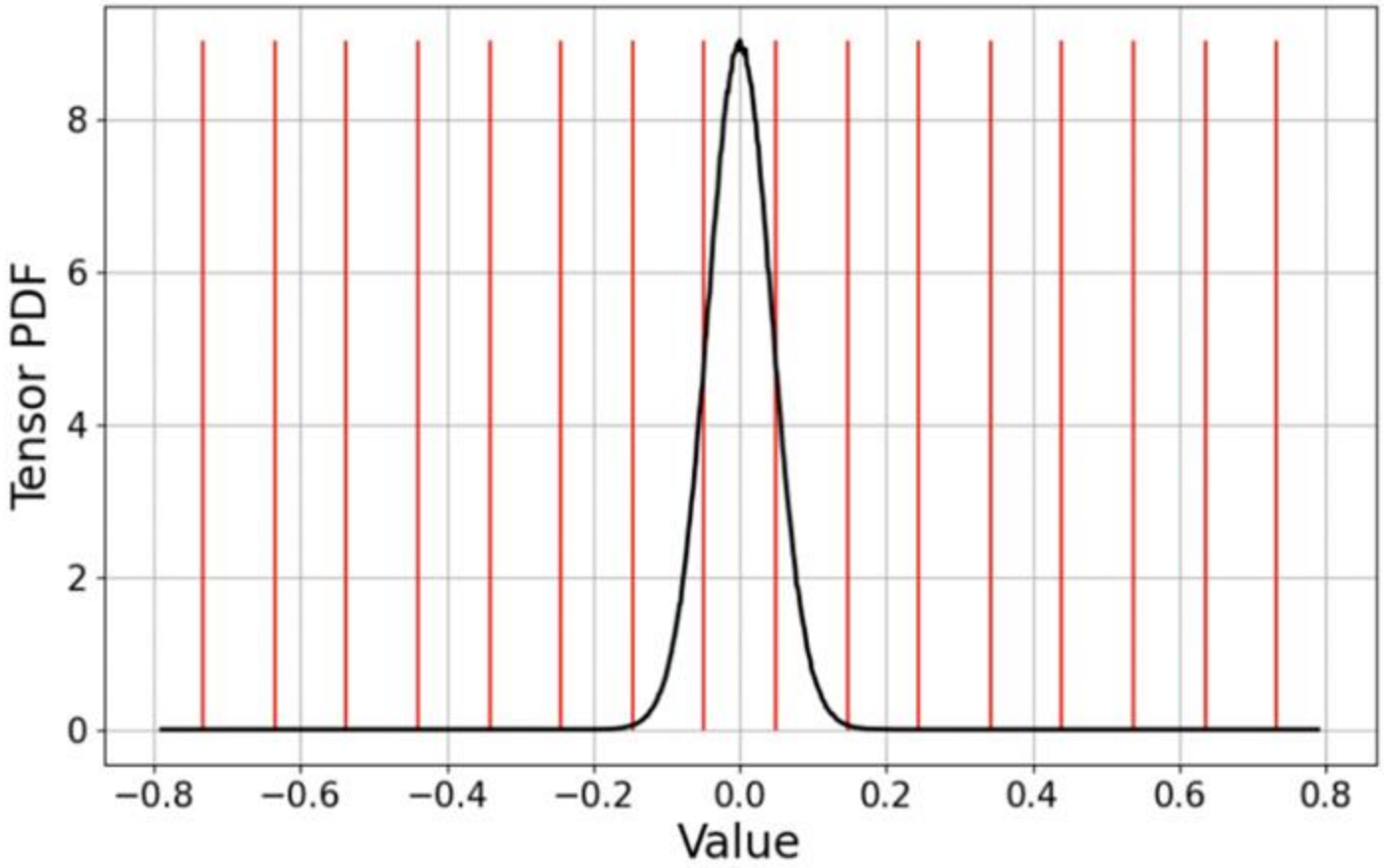
K-Means Quantization vs Linear Quantization



	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Storage			
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

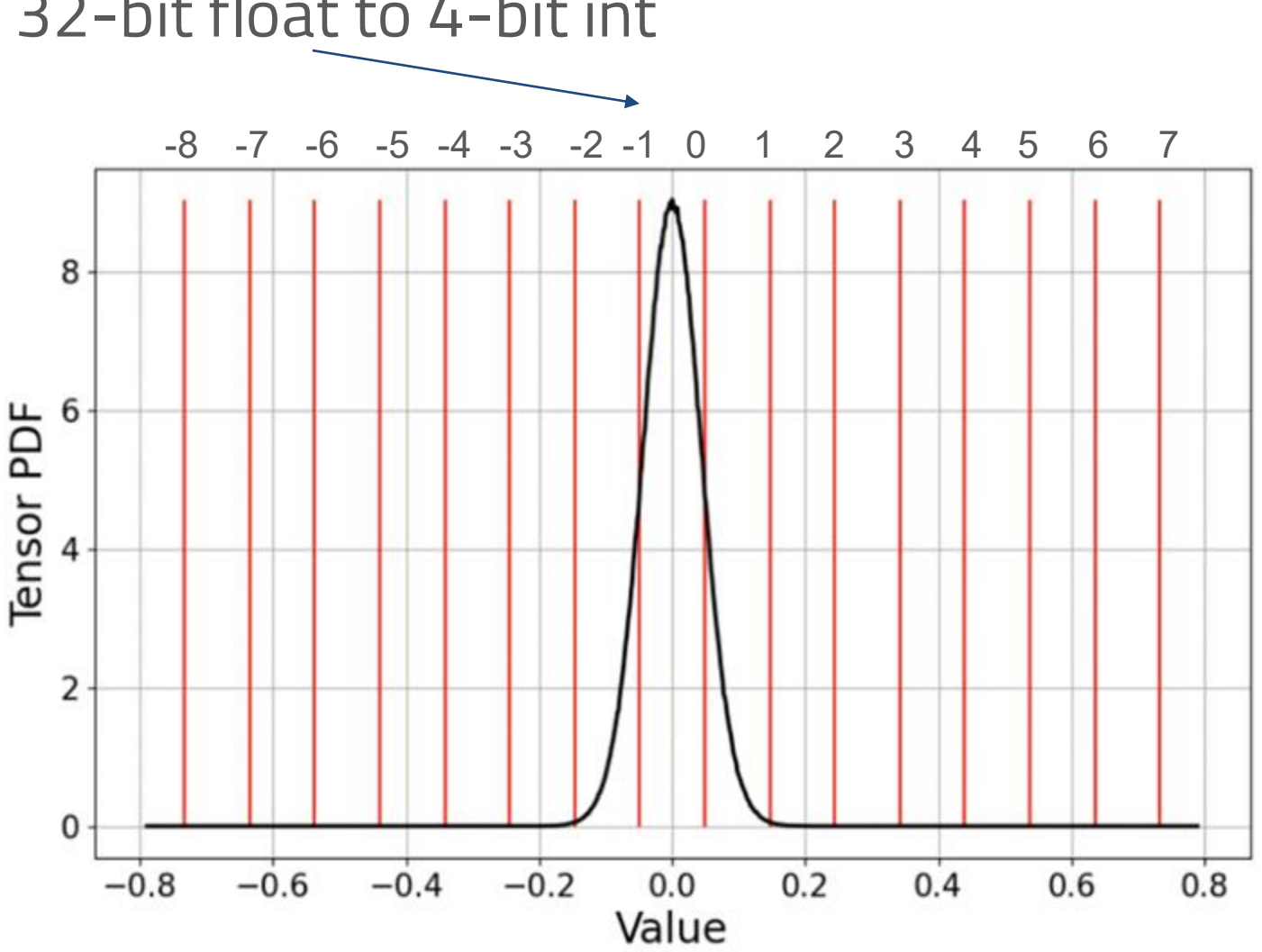
Linear Quantization

- 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



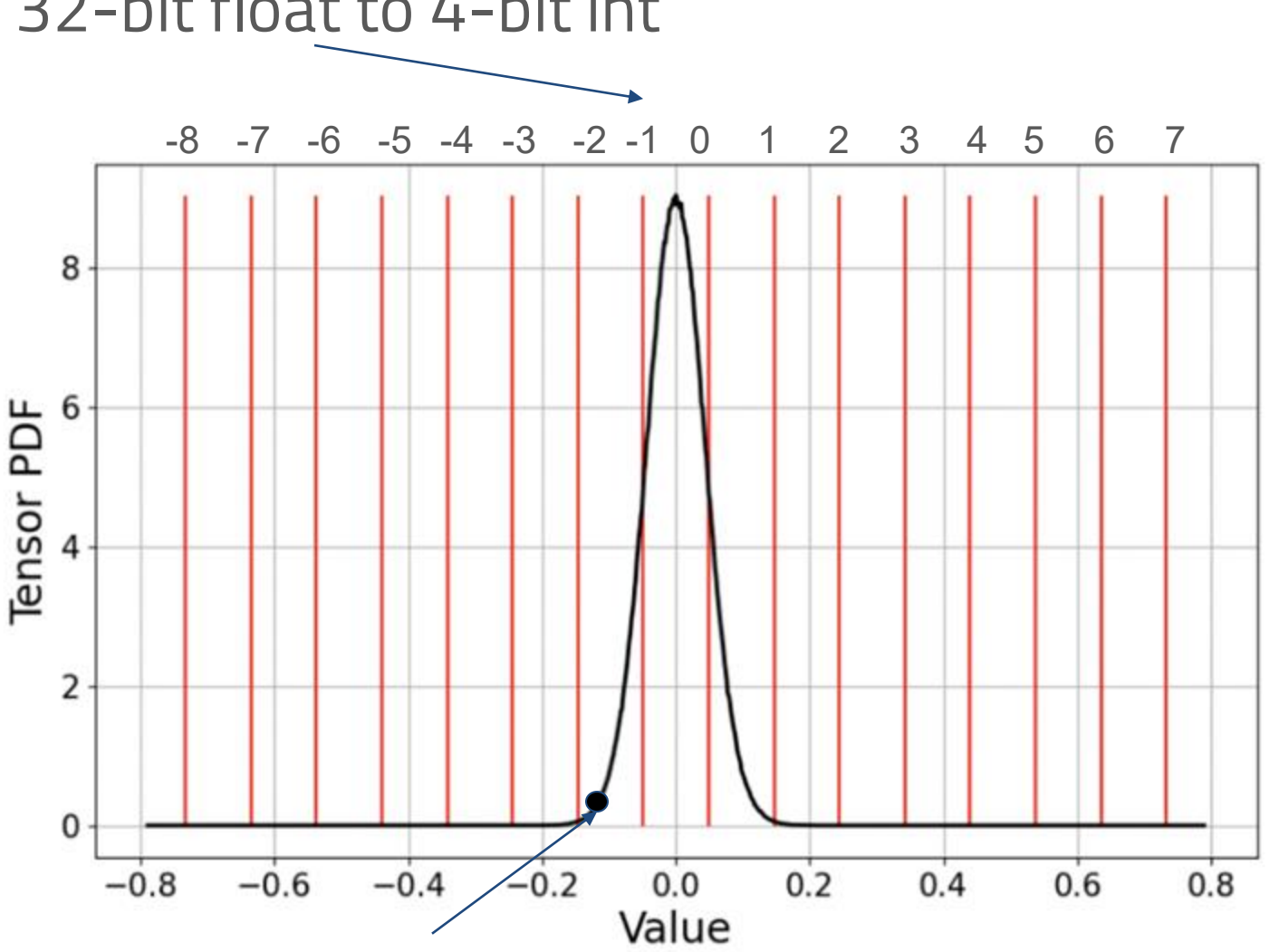
Linear Quantization

- 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



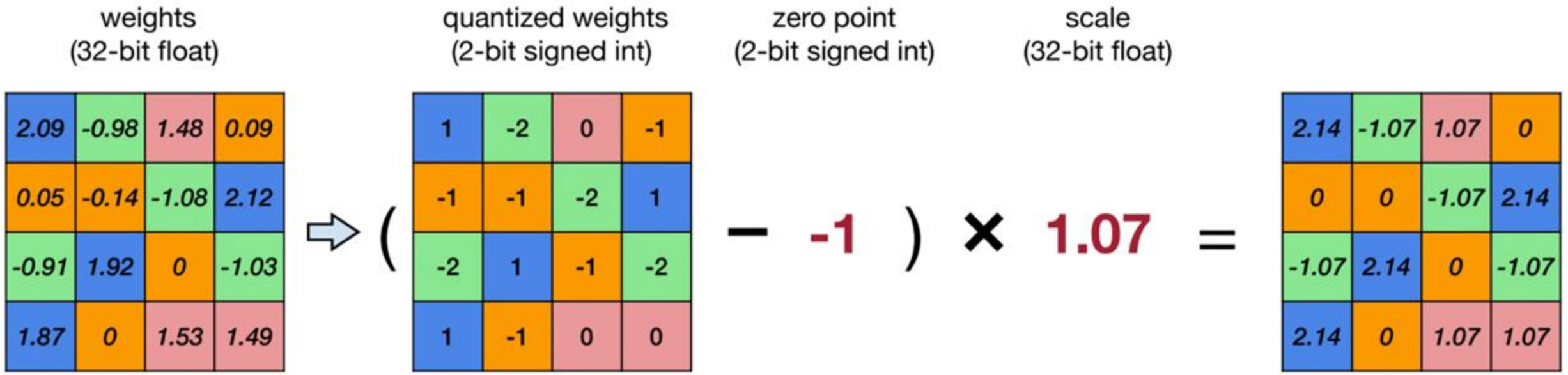
Linear Quantization

- 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



Before Quantization: -.14
After Quantization: -2

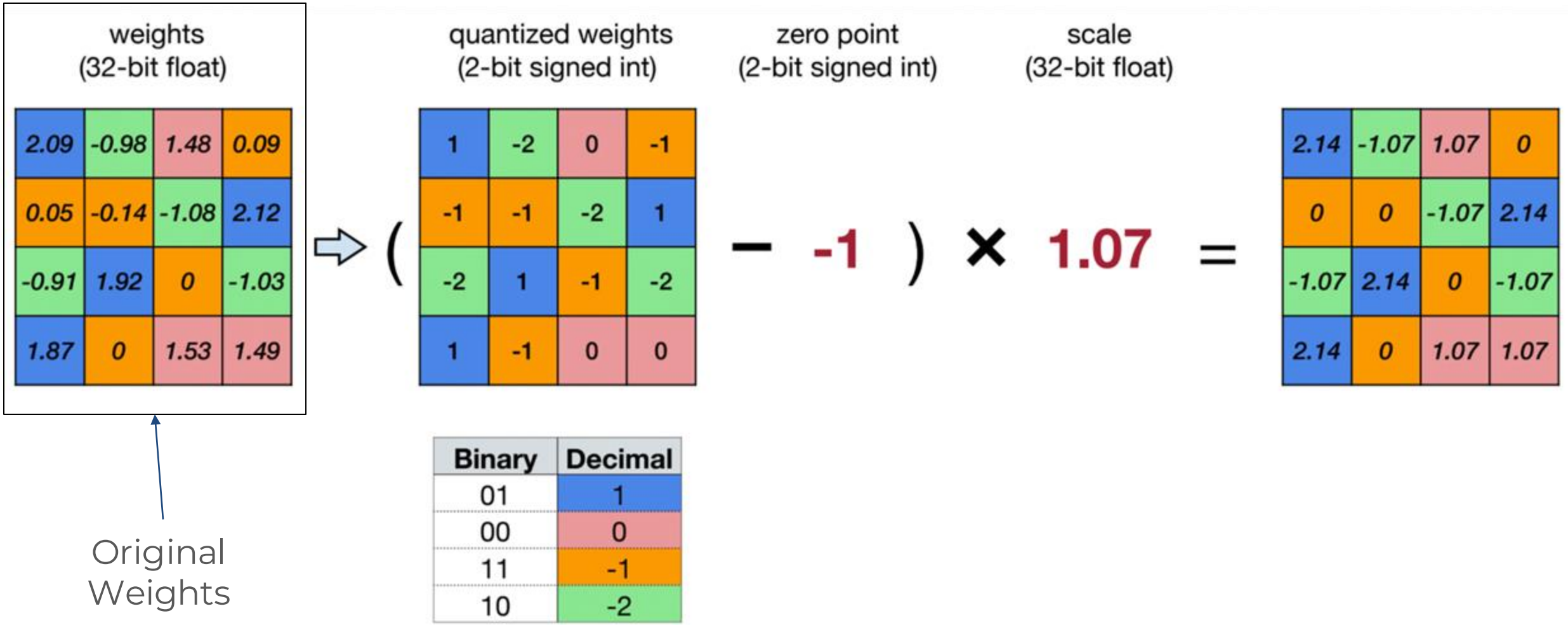
Linear Quantization



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]

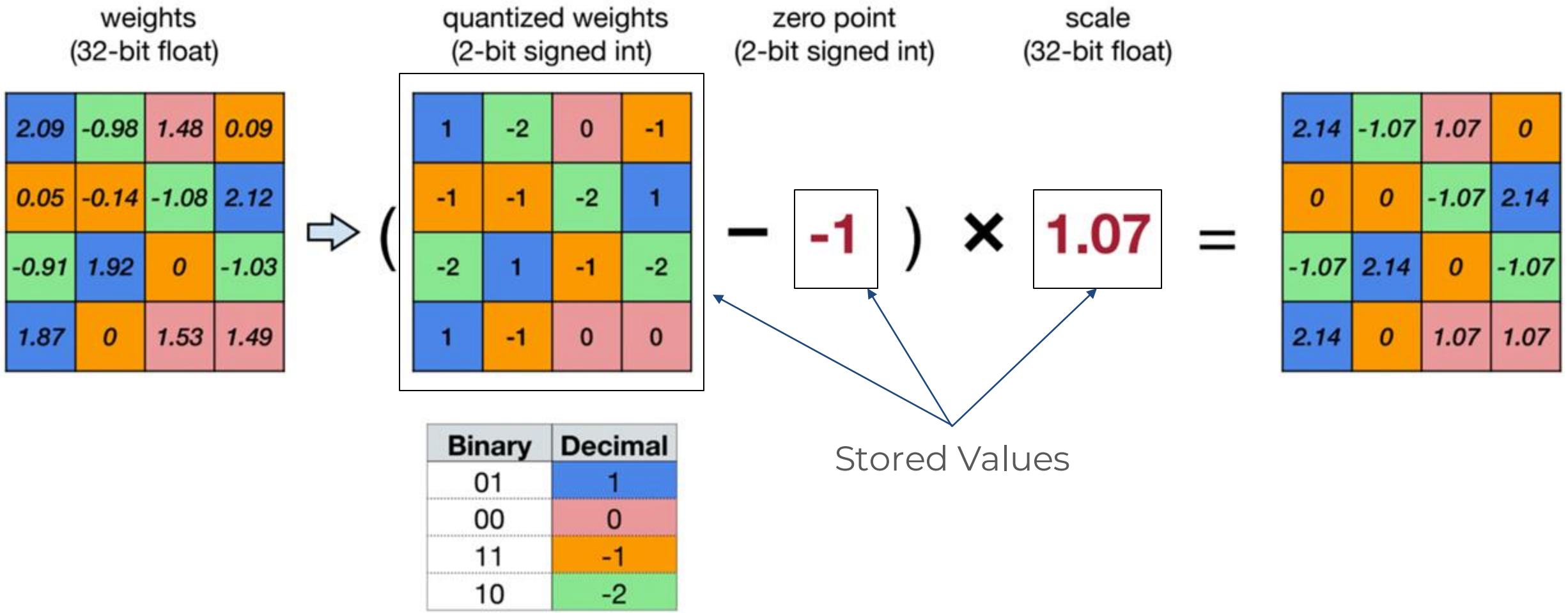
Linear Quantization



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

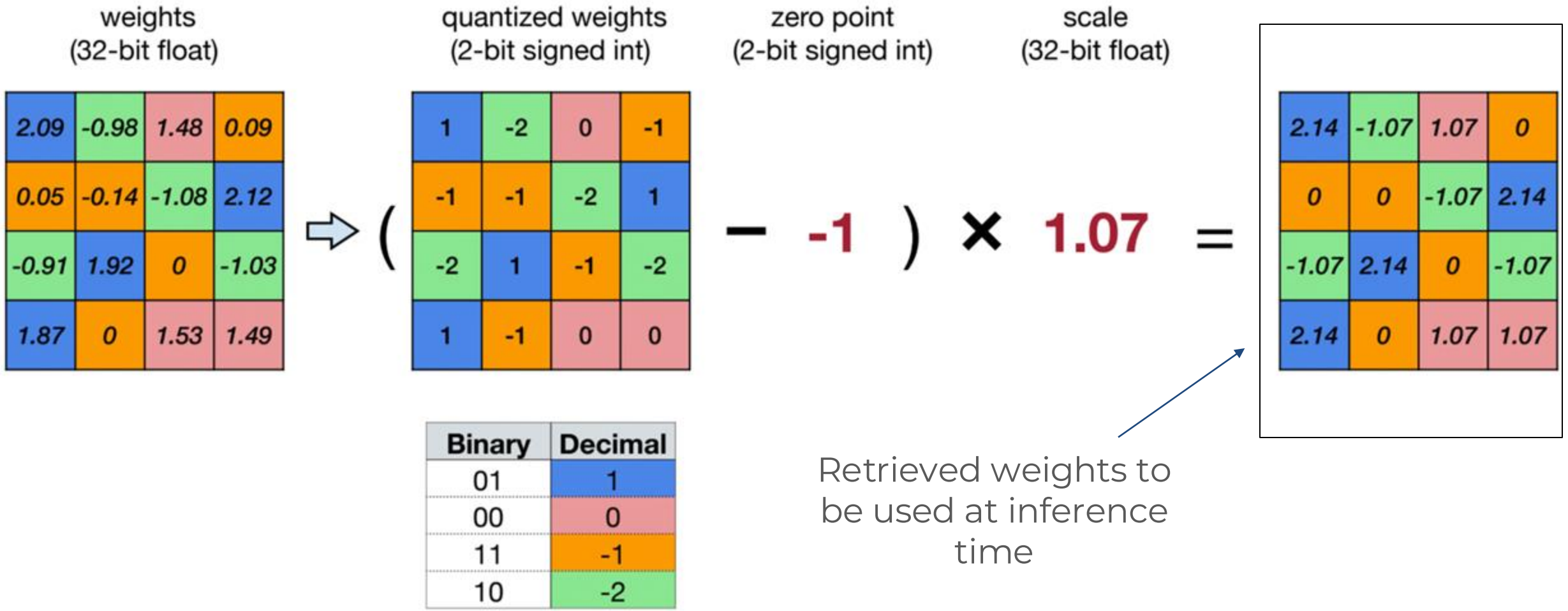
Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacobet al., CVPR 2018]

Linear Quantization



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

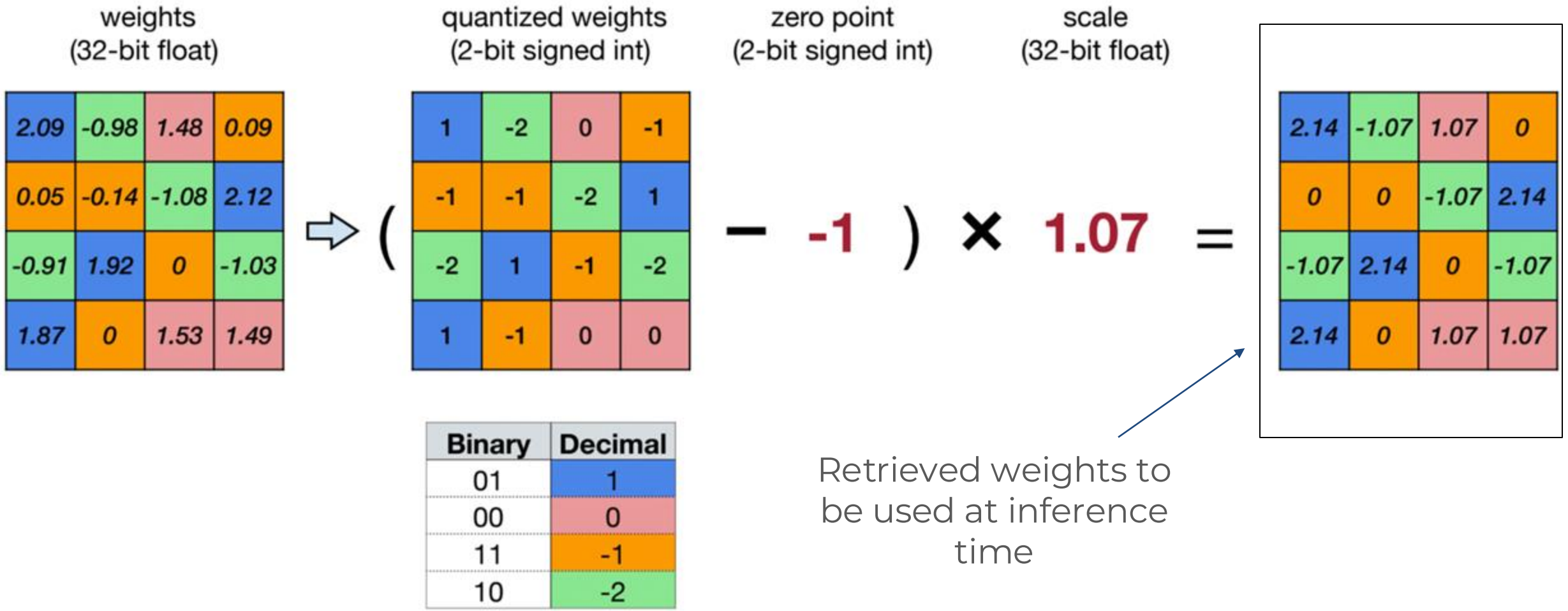
Linear Quantization



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

Linear Quantization

Virtually all modern methods use this (as opposed to KMeans Quantization)



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

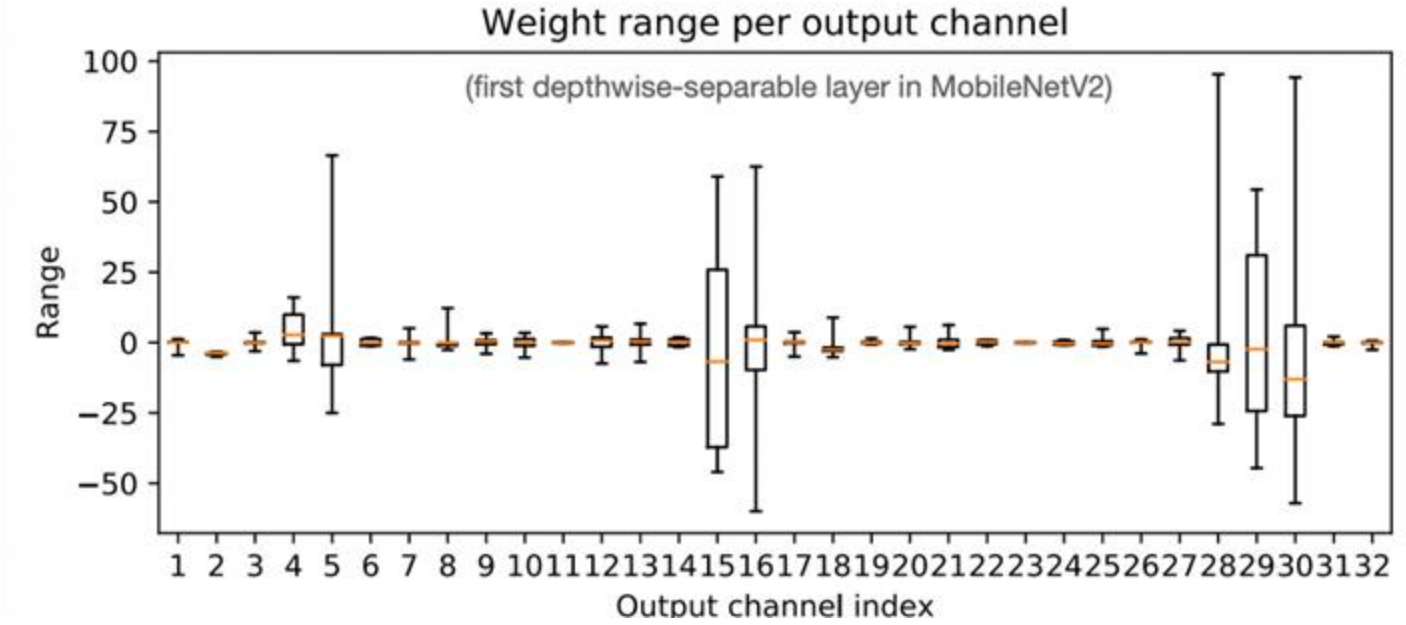
Should we use KMeans/Linear Quantization on all weights at the same time?

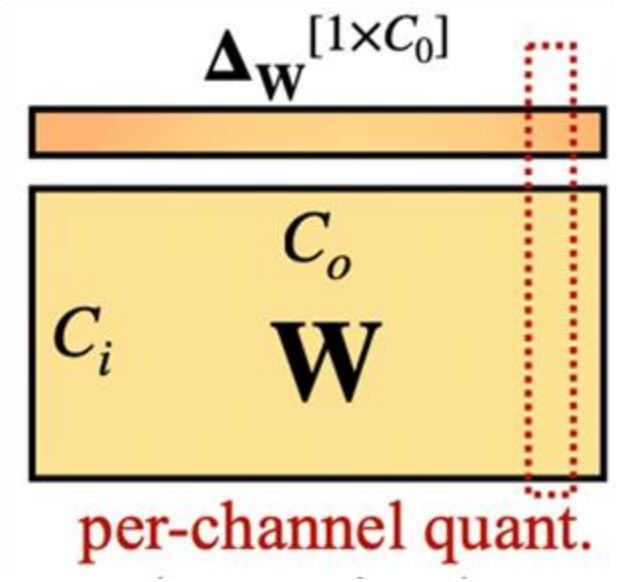
Should we use KMeans/Linear Quantization on all weights at the same time?

No. We should group them.
The size of the grouping we choose is denoted as the granularity.

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

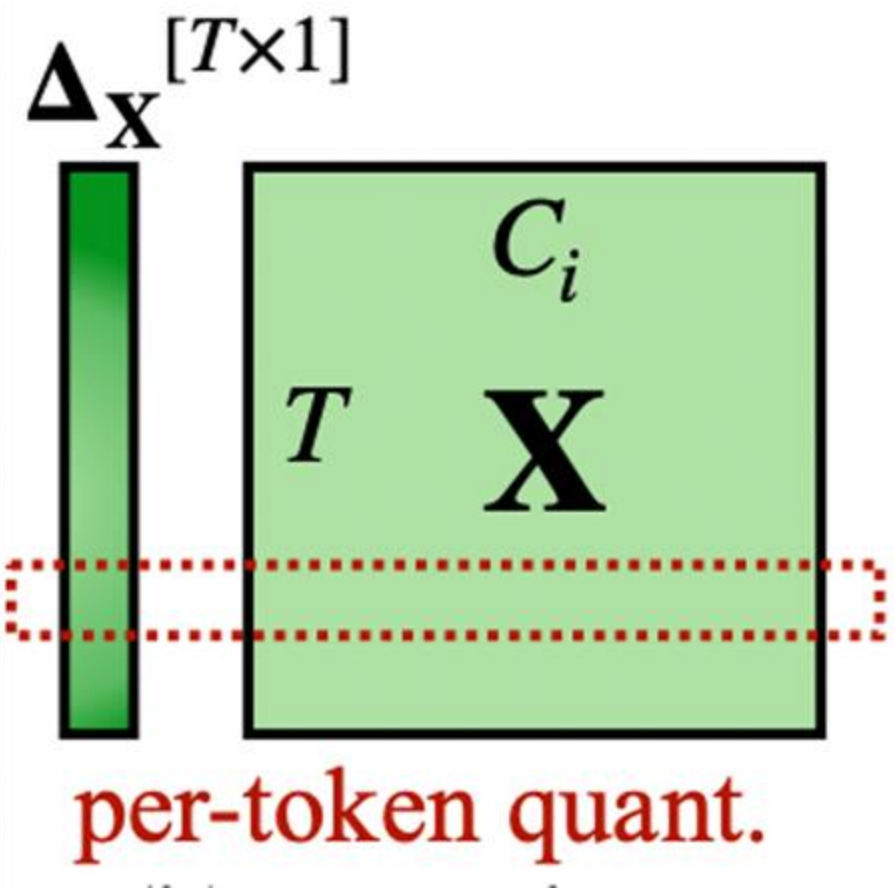




SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models[Xiao et. al., ICML 2023]

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]

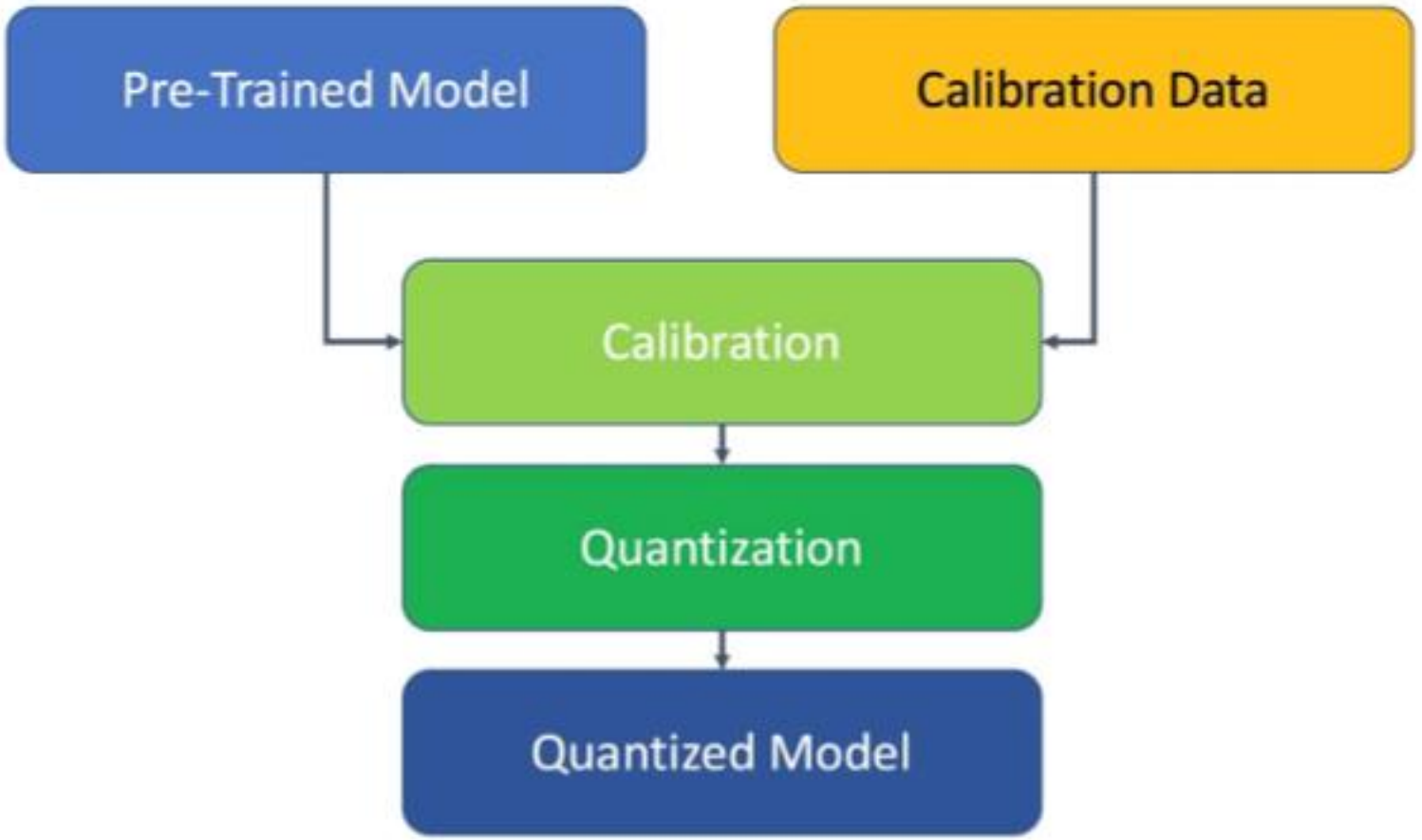
When do we perform this quantization?

When do we perform this quantization?

Typically, this is done after first completely training a model (pretraining → SFT → post-training)

Post Training Quantization (PTQ)

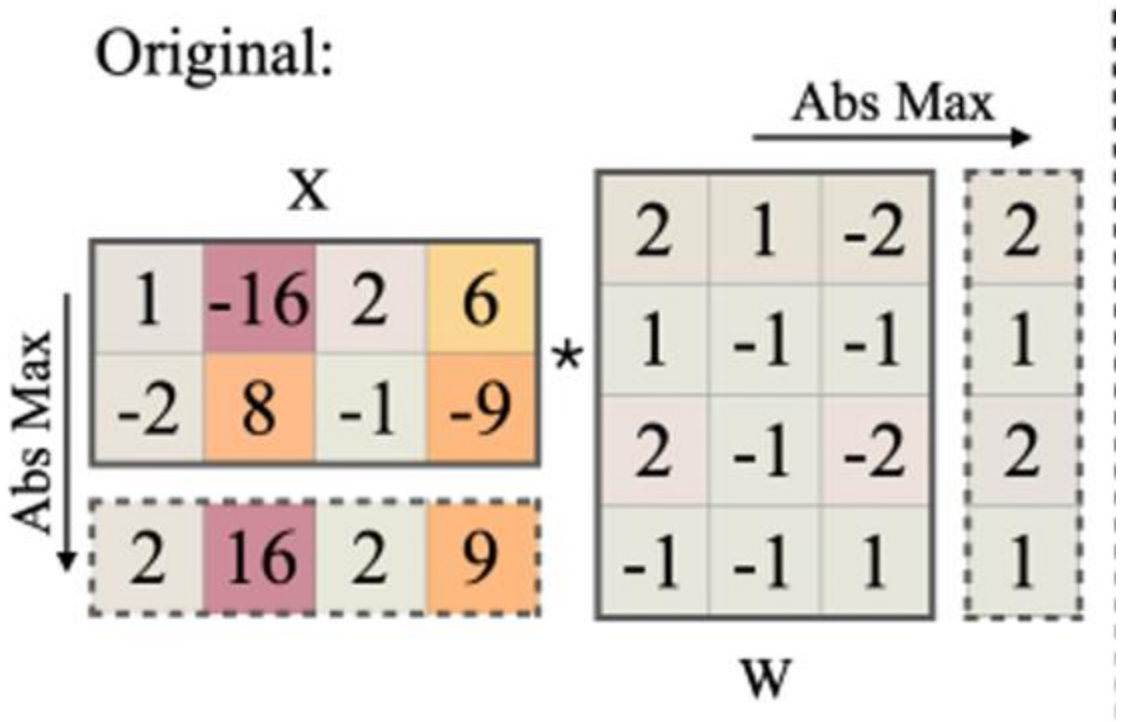
Quantize after training



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

What are some of the modern methods for performing quantization?

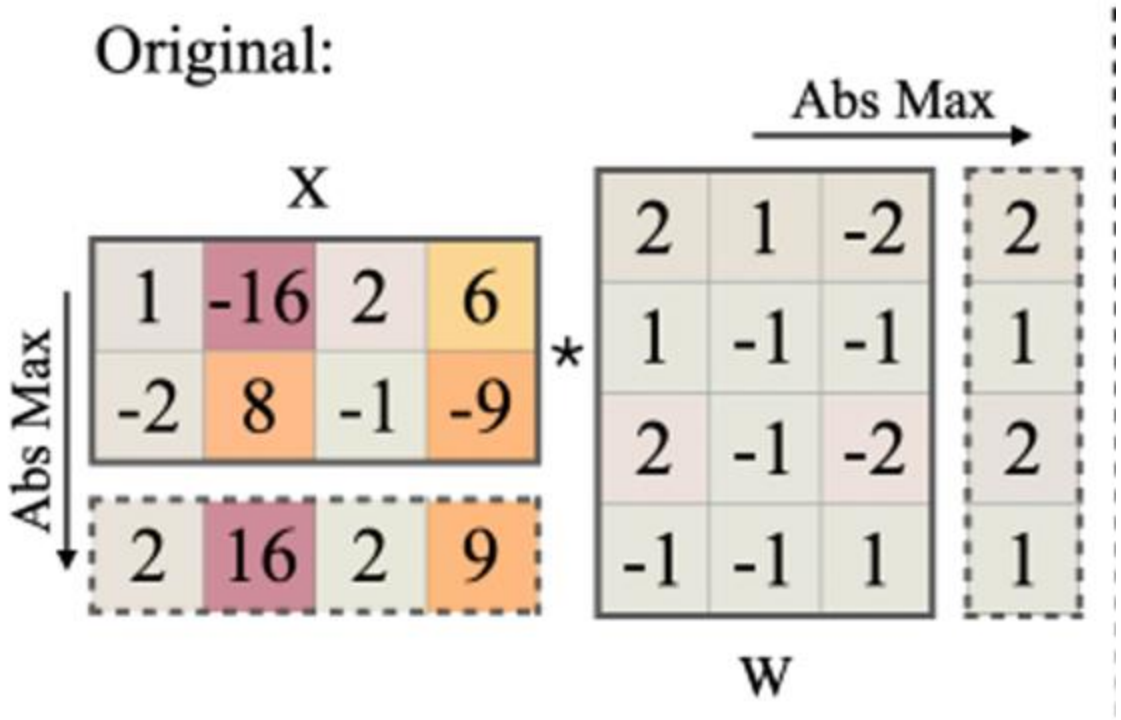
AWQ (W4A16)



Observation: ?

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

AWQ (W4A16)

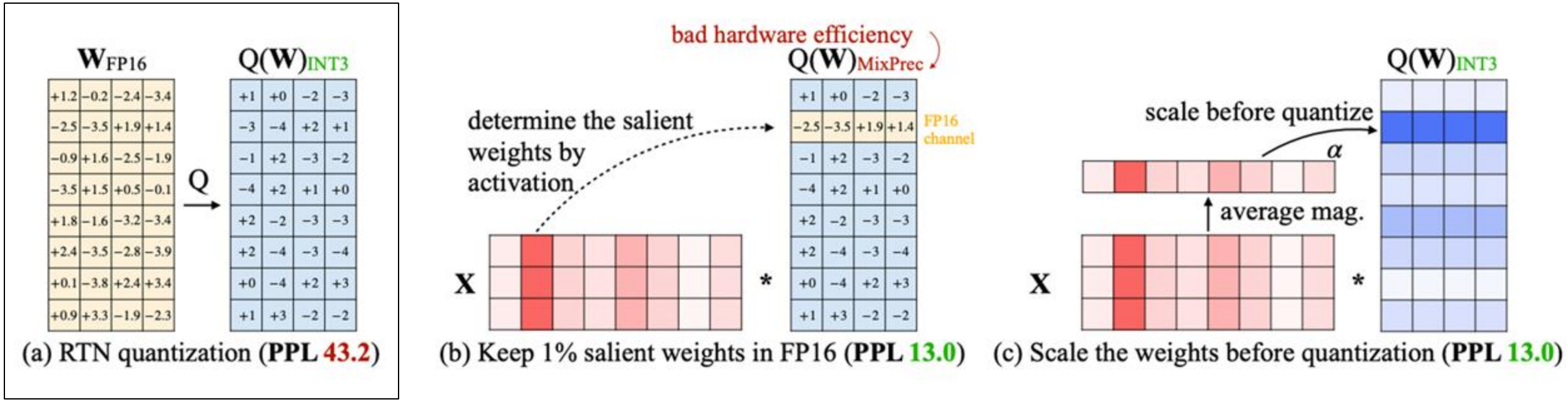


Observation: High variance channels are fixed in activations in LLM FFN layers- weights have relatively little difference in variance

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

AWQ (W4A16)

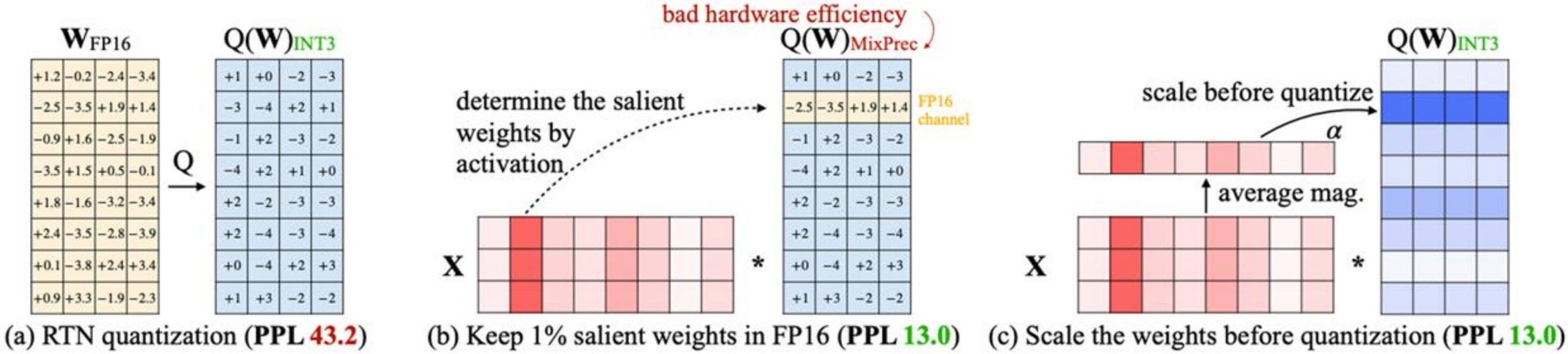
Normal quantization on LLMs performs poorly due to outliers in the model's hidden state



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

AWQ (W4A16)

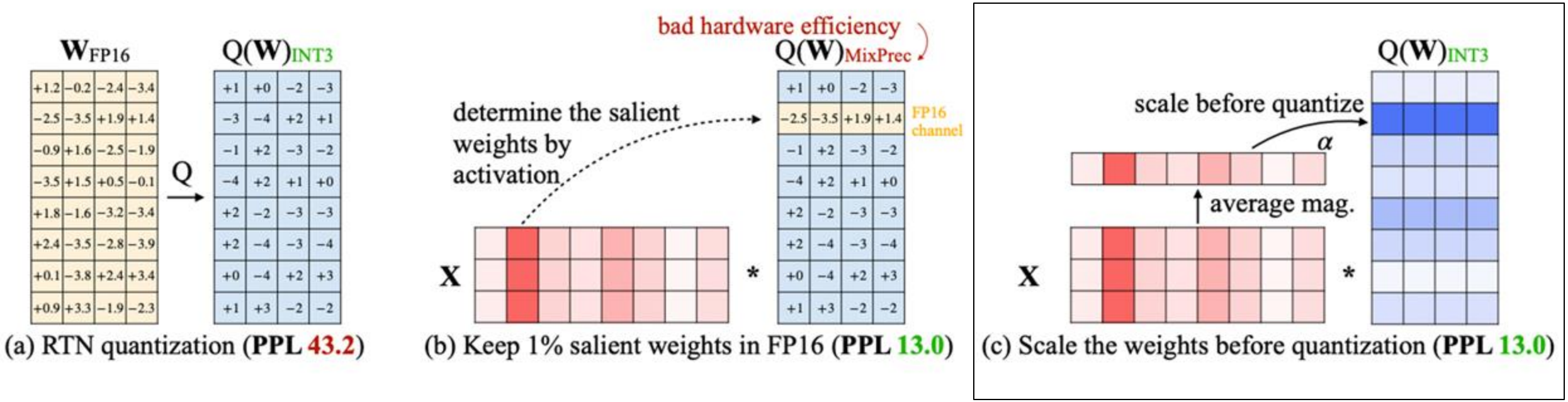
AWQ (c below) scales weights to handle the channels separately



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

AWQ (W4A16)

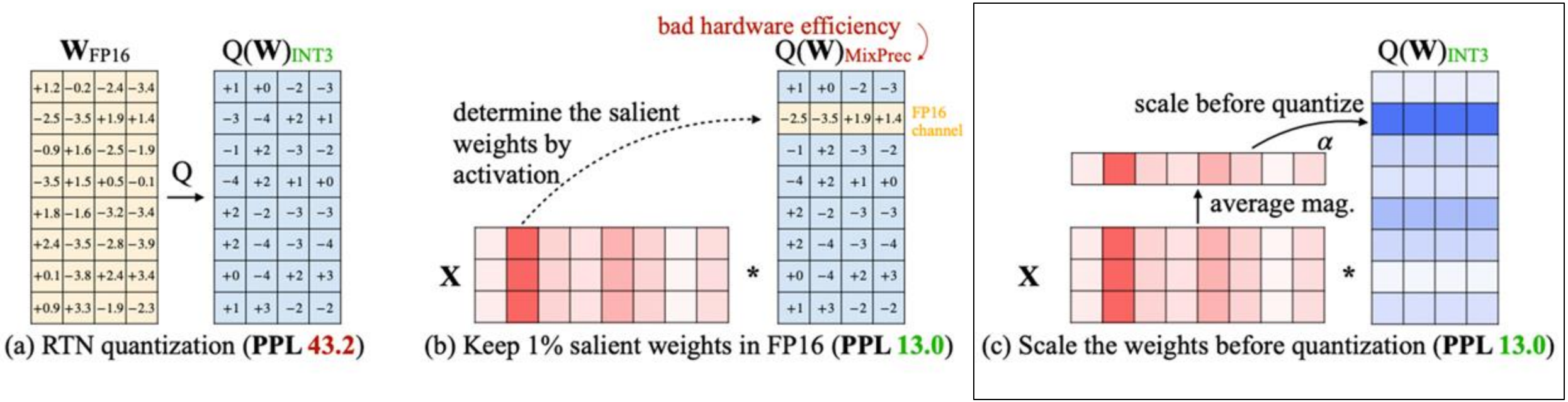
Uses a calibration batch of data to determine what the outlier channels are along with a per-channel scaling factor for these channels



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

AWQ (W4A16)

AWQ is weight-only → multiplication is still performed in BF16/FP16.

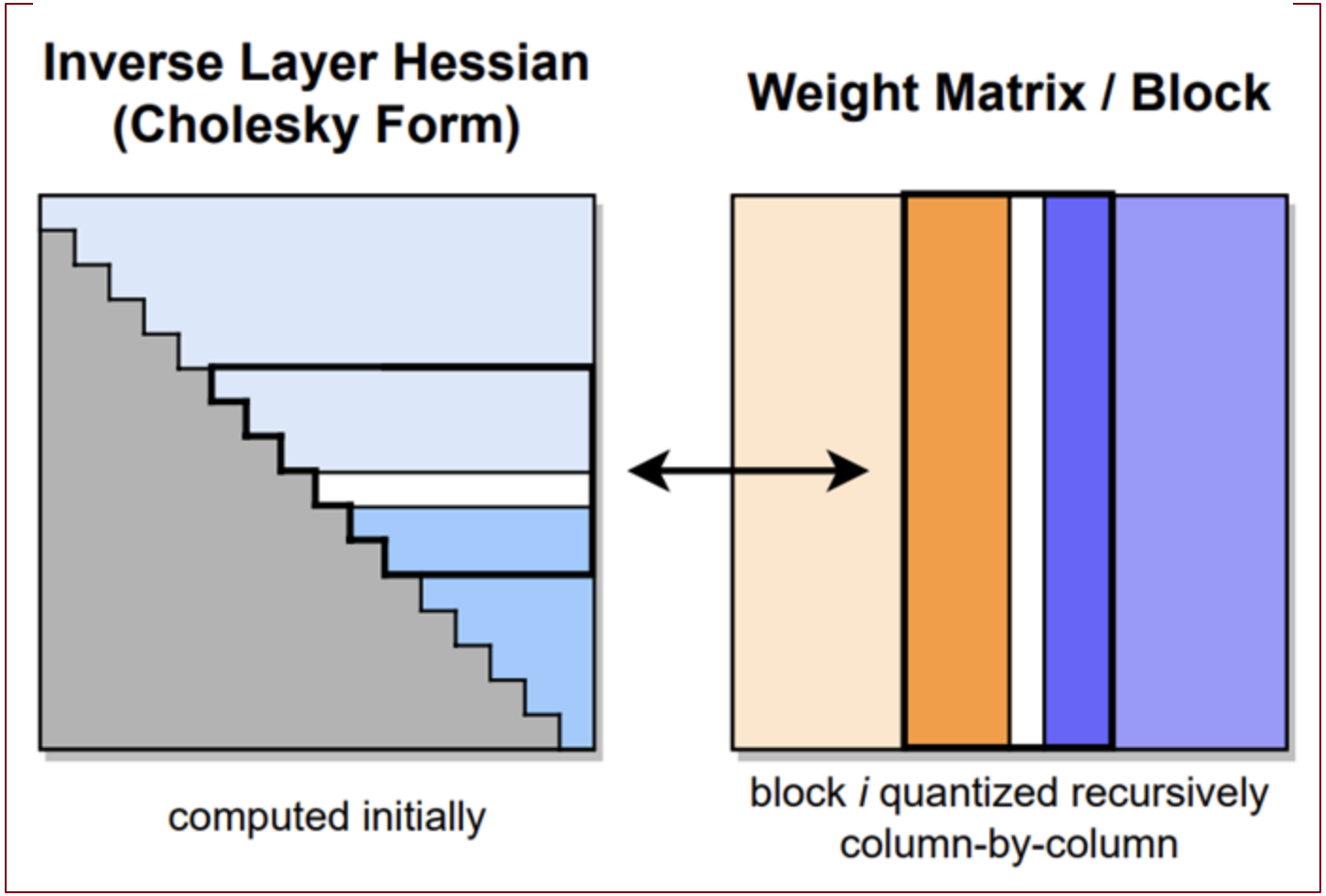


AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et. al., arxiv 2023]

GPTQ

$$\text{argmin}_{\hat{\mathbf{W}}} ||\mathbf{WX} - \hat{\mathbf{W}}\mathbf{X}||_2^2.$$

- Quantizes a block of each weight matrix at a time.
- Uses a loss term + the inverse hessian for the layer to update the non-quantized weights
- Takes a *long* time



GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers **[Frantar et al., arXiv 2022]**

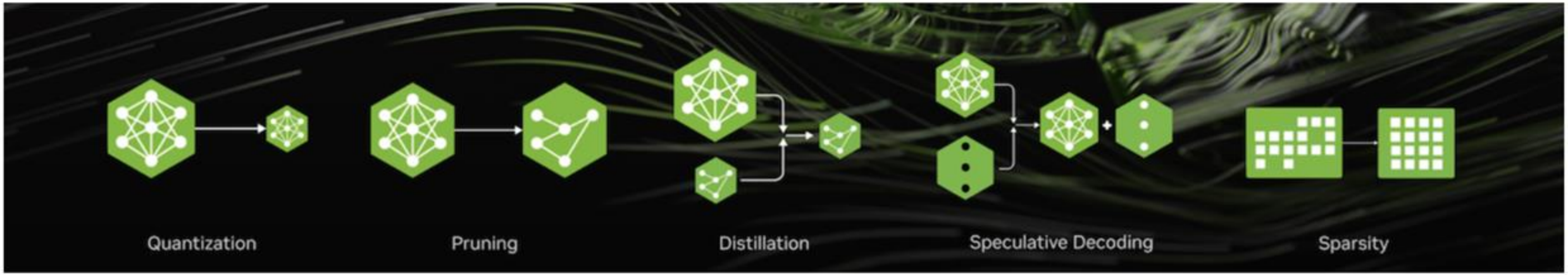
GGUF Quantization

- No model retraining as is done with GPTQ
- A model is quantized by taking blocks of the input matrix, then performing linear quantization on each block separately
- Uses several different methods based on level of quantization
- Will come with names such as IQ2_XS–IQ4_XS, Q2_K_S–Q5_K_S
- You can use unsloth to create models with this type

GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers **[Frantar et al., arXiv 2022]**

FP8 Quantization (Nvidia)

- Offered by the NVIDIA TensorRT Model Optimizer library
- Performs a separate per-channel post-training quantization
- Supported by hardware
- This library also performs a number of other functions for improving model efficiency



Modern Quantization (Open Source Models)

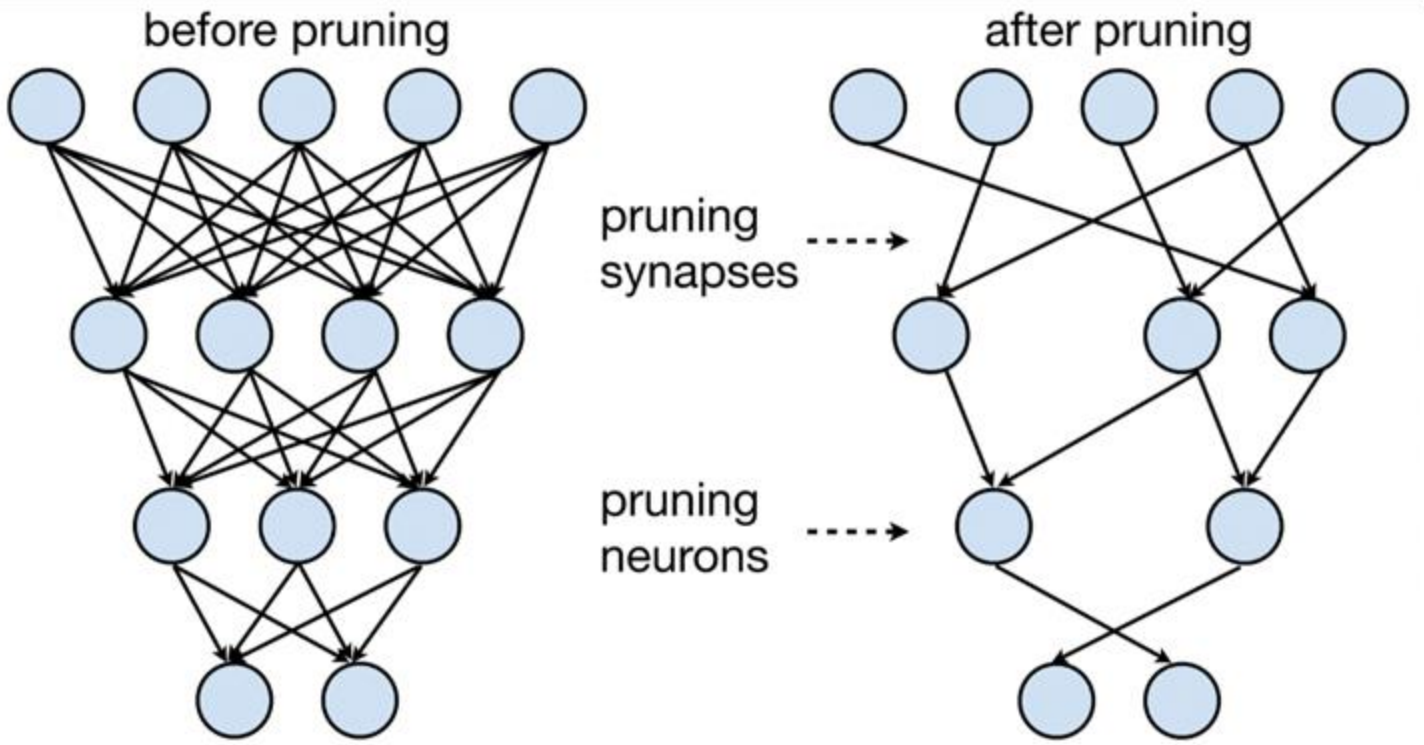
Oftentimes, open source LLM releases from model providers will release versions of models that use several of the above quantization techniques

Efficient LLMs

- ❑ Quantization
- ❑ **Sparsity**
- ❑ Long Context
- ❑ Serving & Systems
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

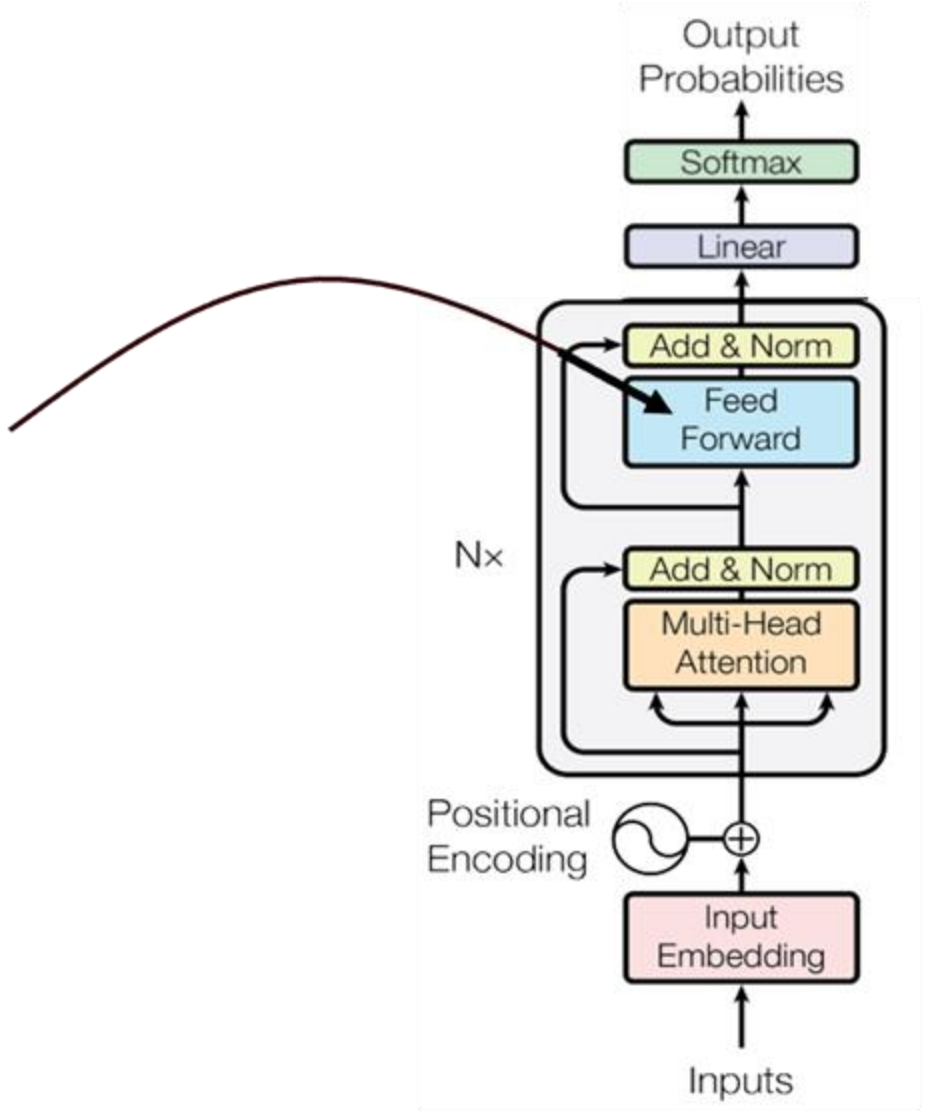
Sparsity

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

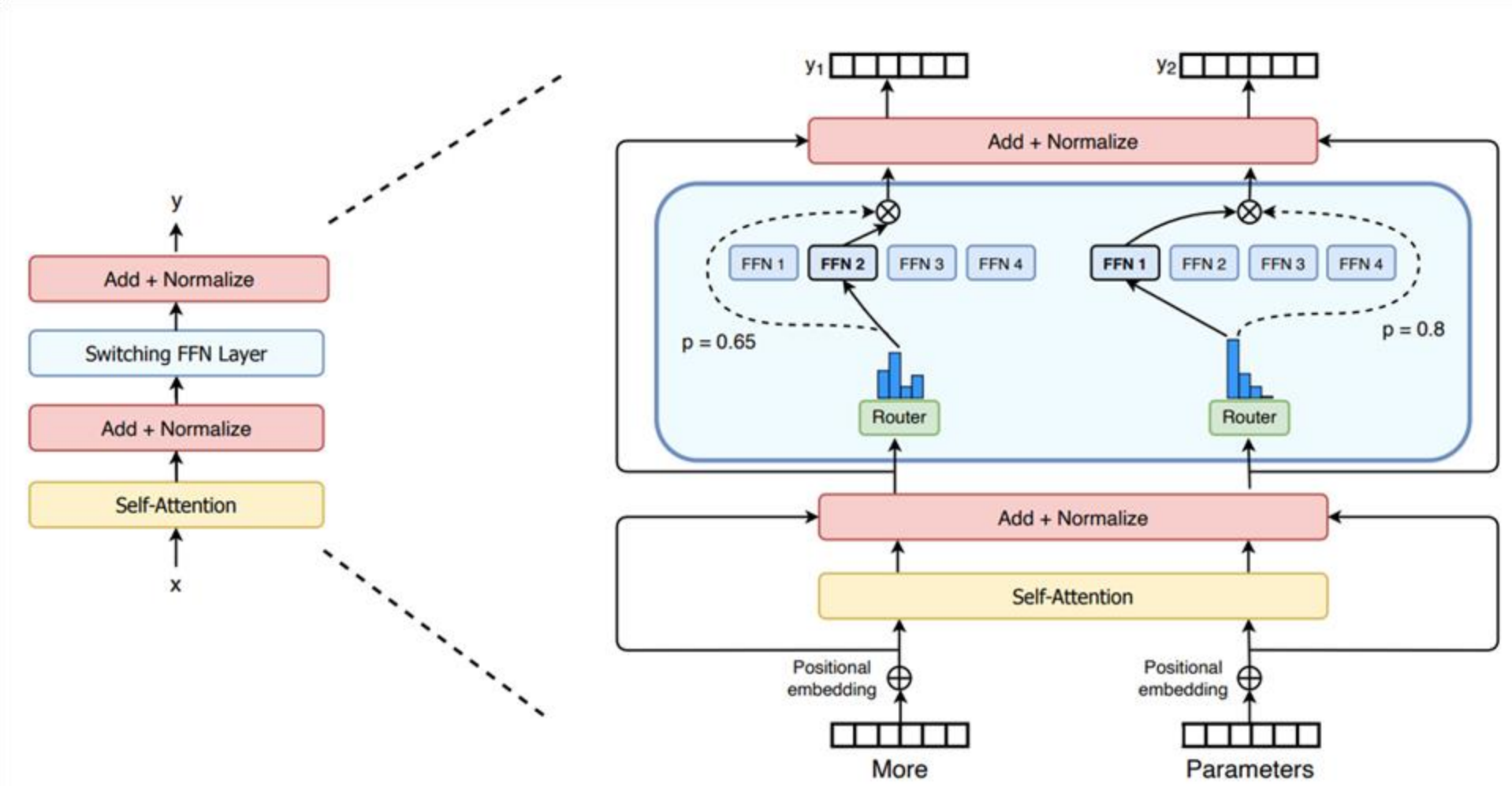


Mixture of Experts (MoE)

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



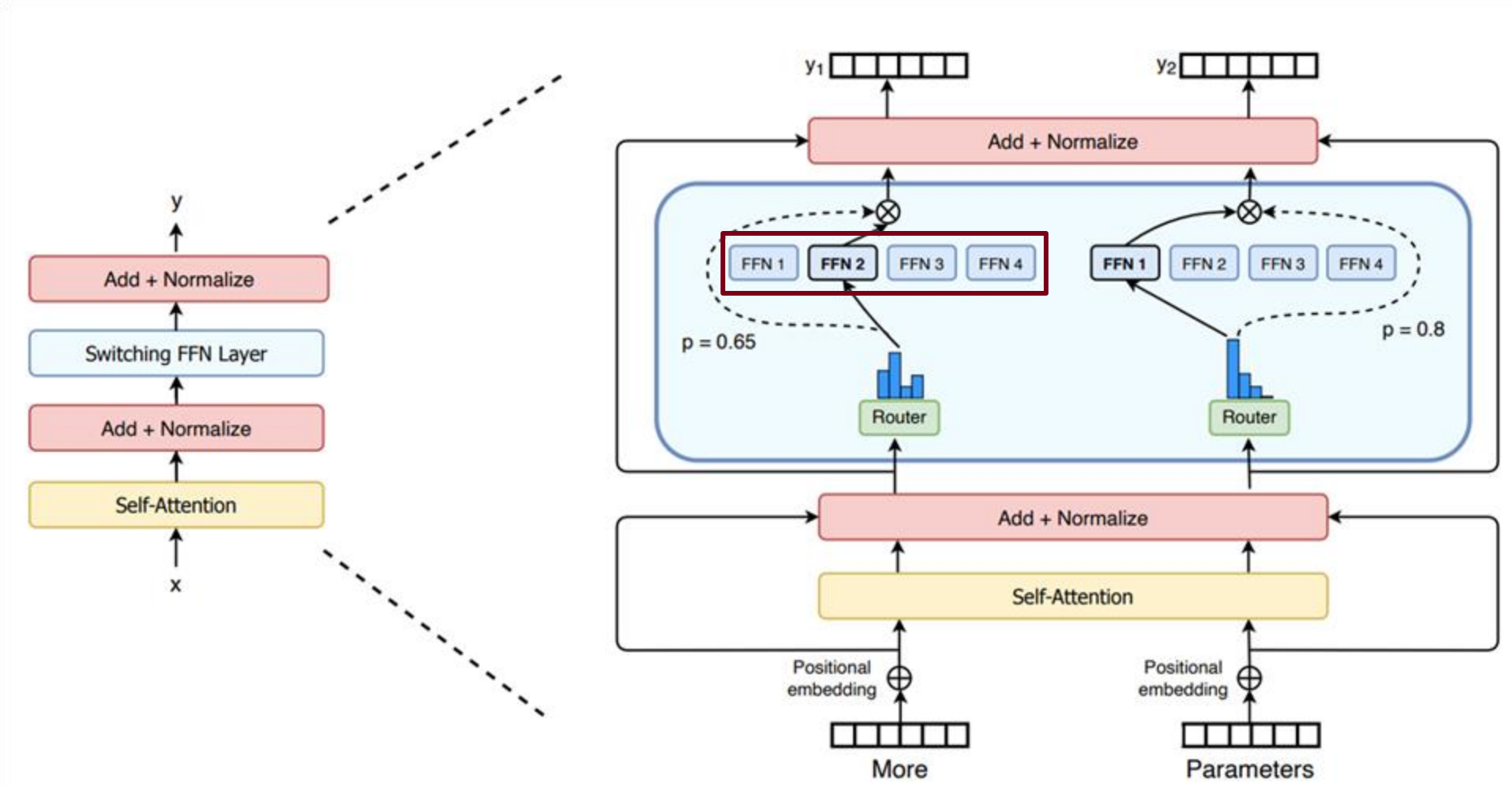
Mixture of Experts (MoE)



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

Mixture of Experts (MoE)

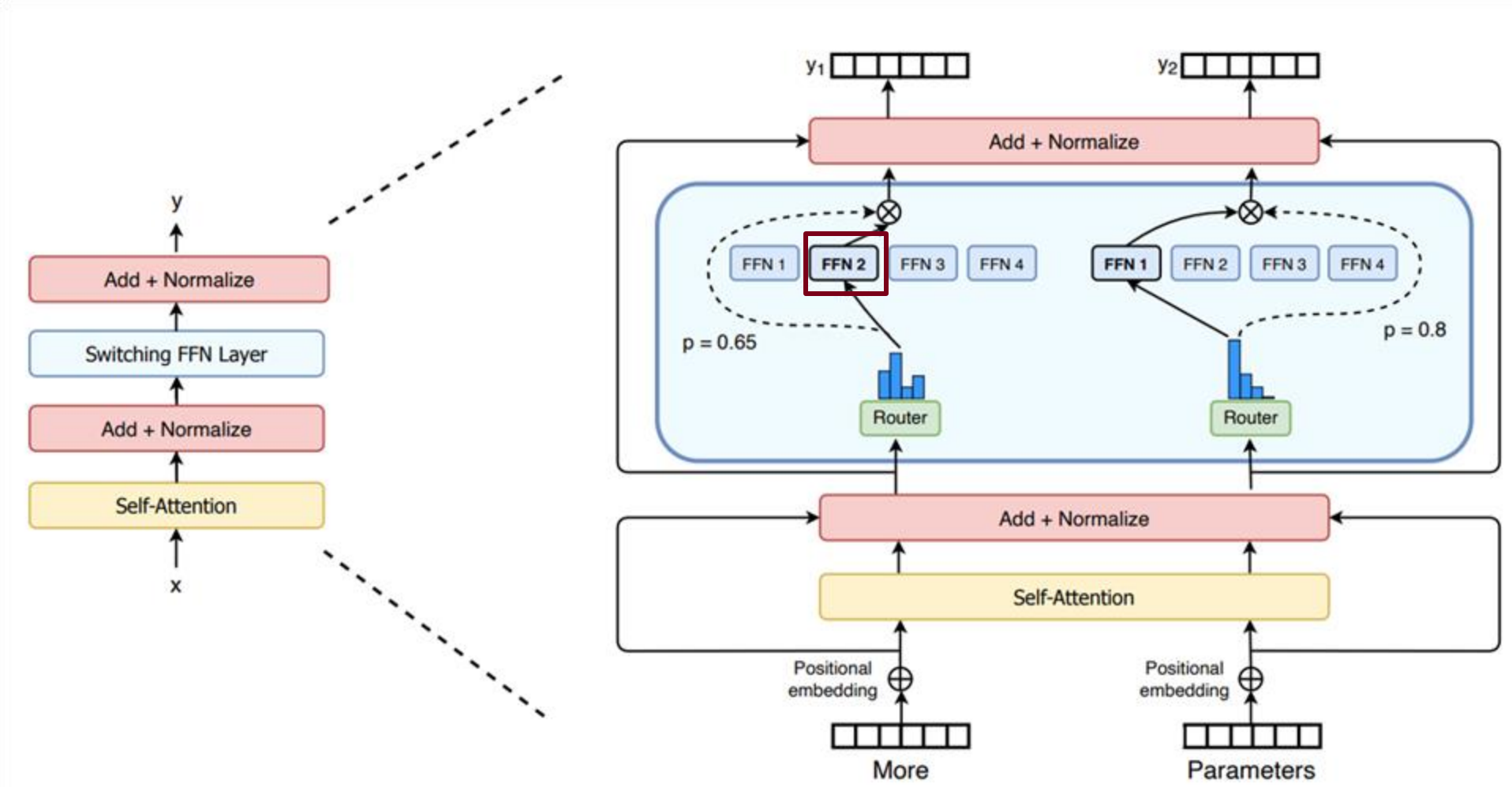
Four FFN layers



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

Mixture of Experts (MoE)

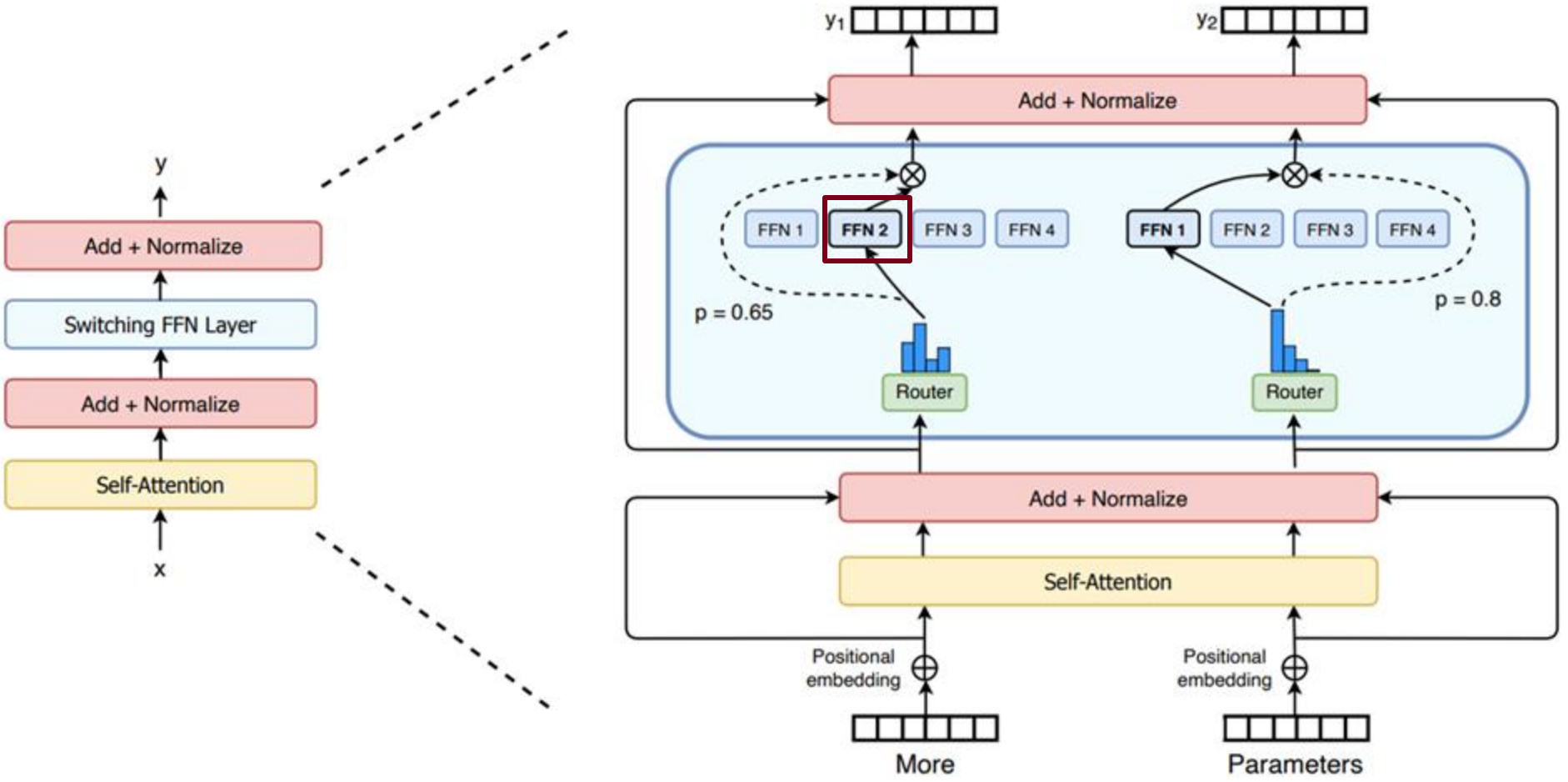
Only one is used per token



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



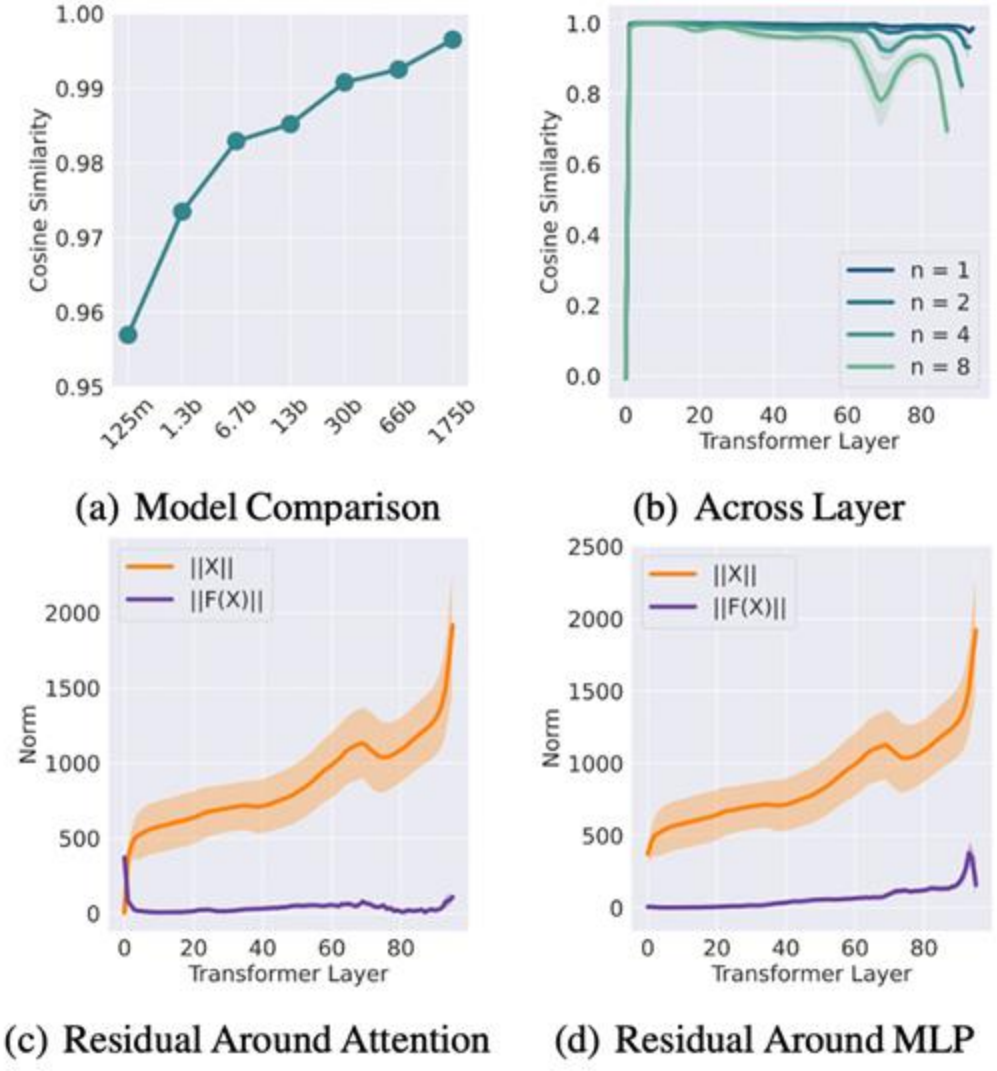
Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., CoRR 2021]

Mixture of Experts (MoE)

Most of the best-performing open source models produced today are mixture of experts models

Deja Vu: Contextual Sparsity

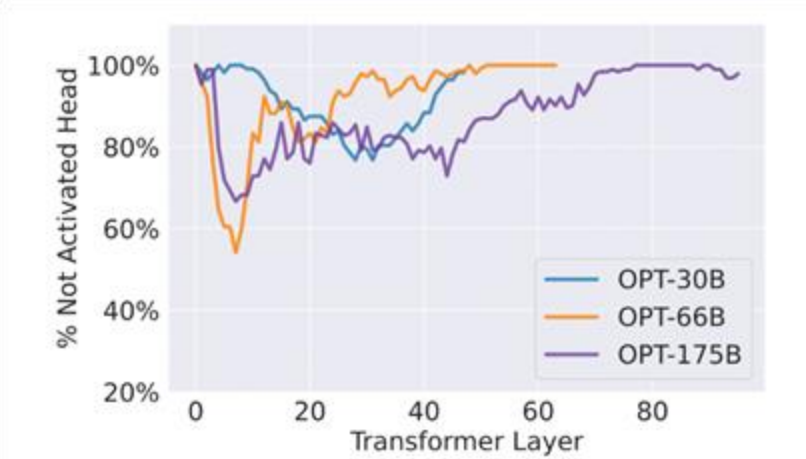
Observation 1: Model activations change very little between consecutive layers of a network



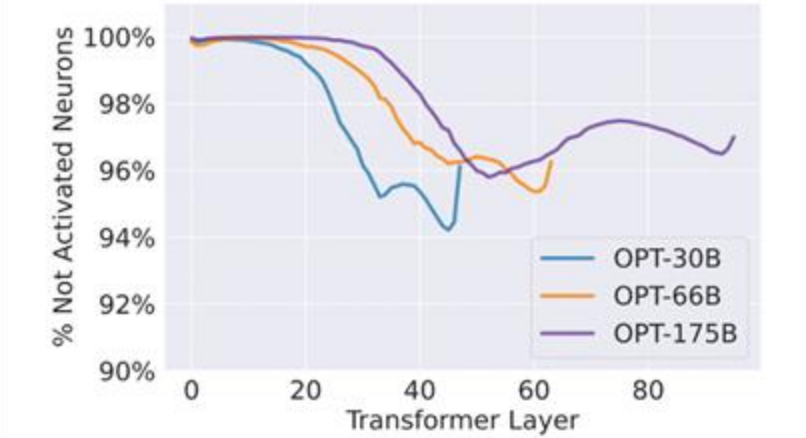
Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time [Liu et al., 2023]

Deja Vu: Contextual Sparsity

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



(a) Contextual sparsity in Attention Head

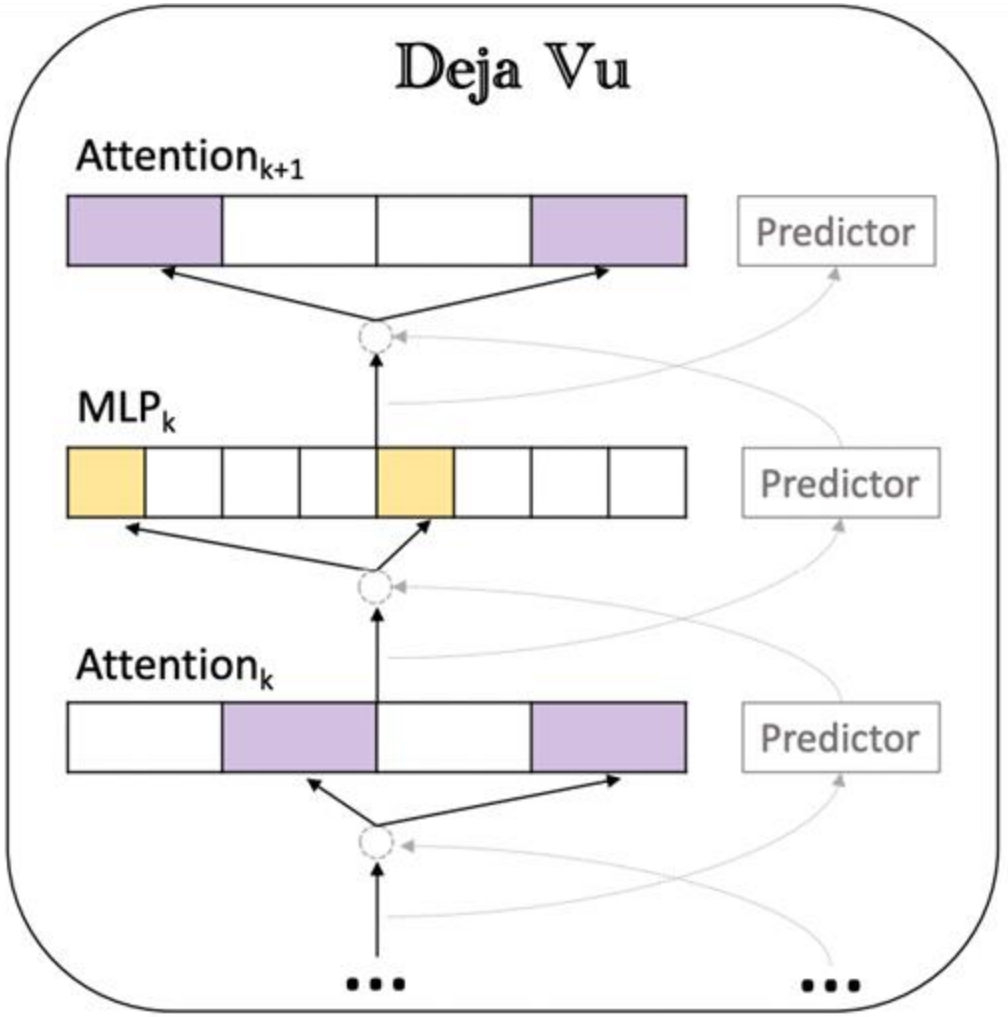


(b) Contextual sparsity in MLP Block

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

Deja Vu: Contextual Sparsity

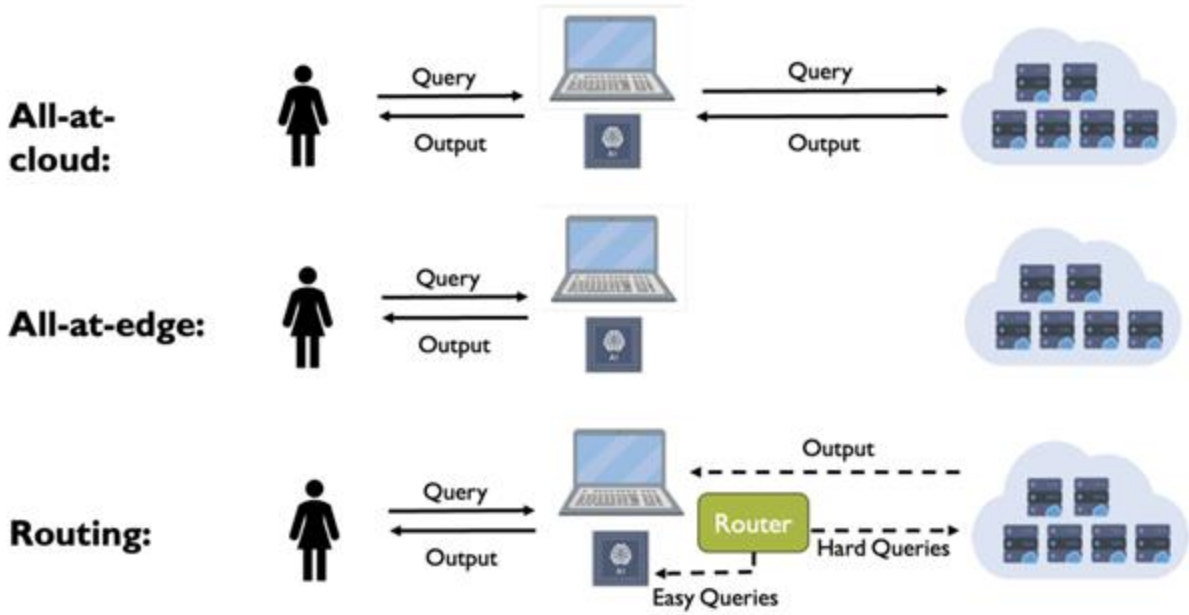
Sparsification: Use predictors in each layer to determine which neurons to activate and which attention heads to use – ignore all unpredicted heads/neurons



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

Model Routing

Route easier queries to smaller models, harder queries to larger models

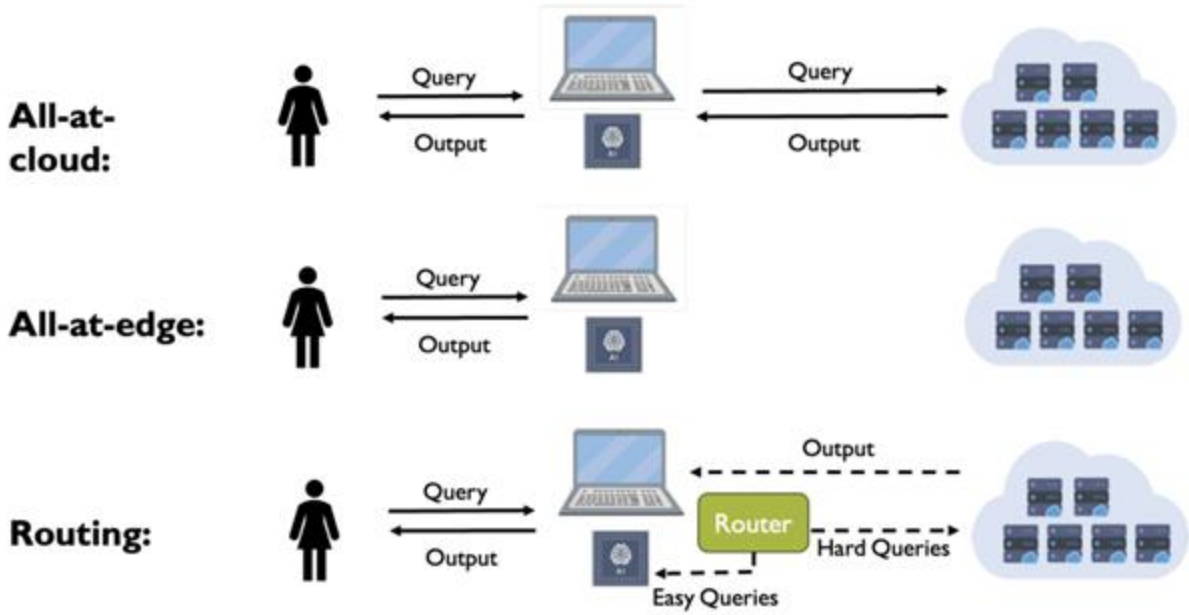


Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing [Ding et al., 2024]

Model Routing

Valuable for companies like OpenAI/Anthropic when they serve multiple models of different capabilities & sizes

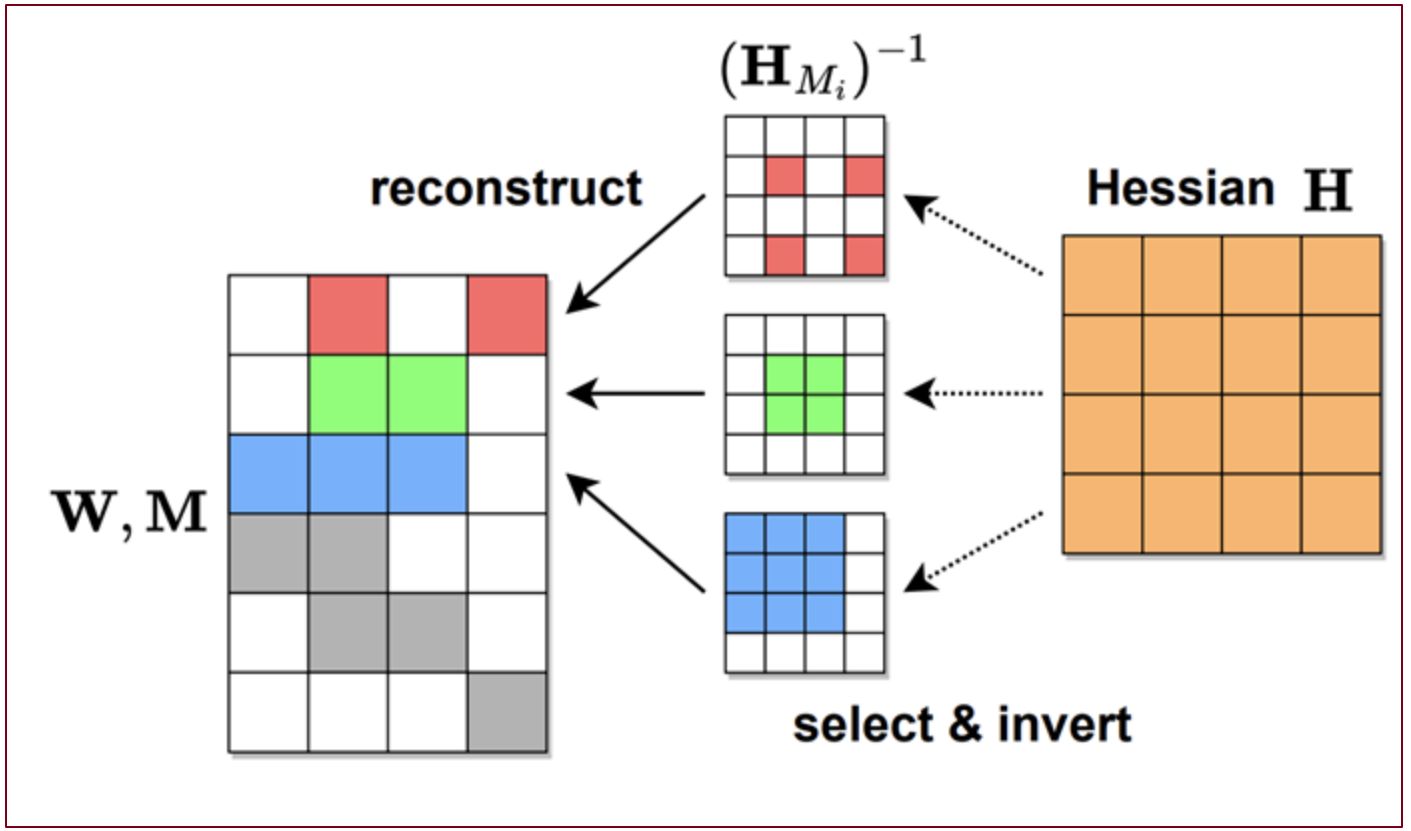
Route easier queries to smaller models, harder queries to larger models



Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing [Ding et al., 2024]

SparseGPT

- Similar to GPTQ, uses a reconstruction loss term to prune entries from each weight matrix
- This can be taken advantage of with either 2:4, 4:8 sparsity on Nvidia GPUs



Efficient LLMs

- ❑ Quantization
- ❑ Sparsity
- ❑ **Long Context**
- ❑ Serving & Systems
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

What is a central issue with having longer context models?

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

What is a central issue with having longer context models?

The storage costs are linear in generation length and quadratic in computation

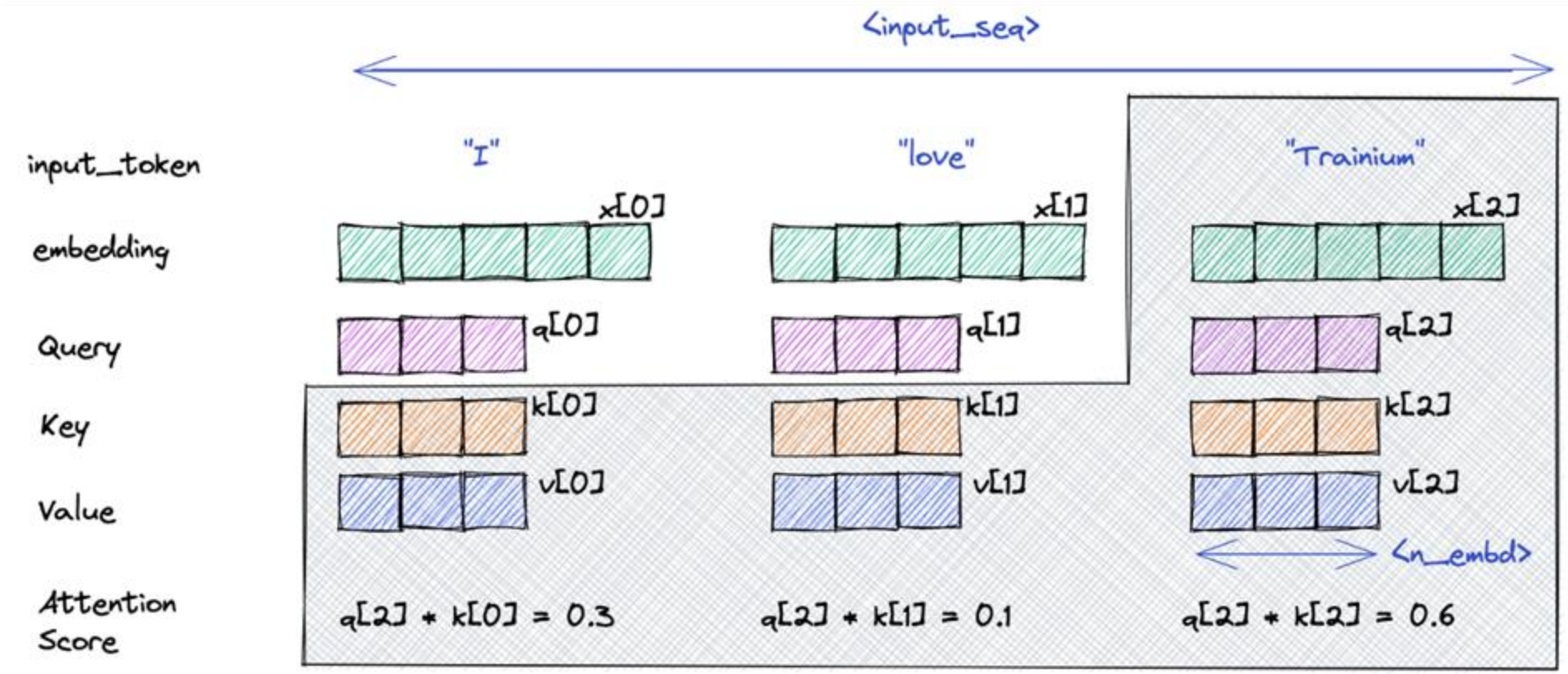
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

What is a central issue with having longer context models?

The **storage** costs are linear in generation length and quadratic in **computation**

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

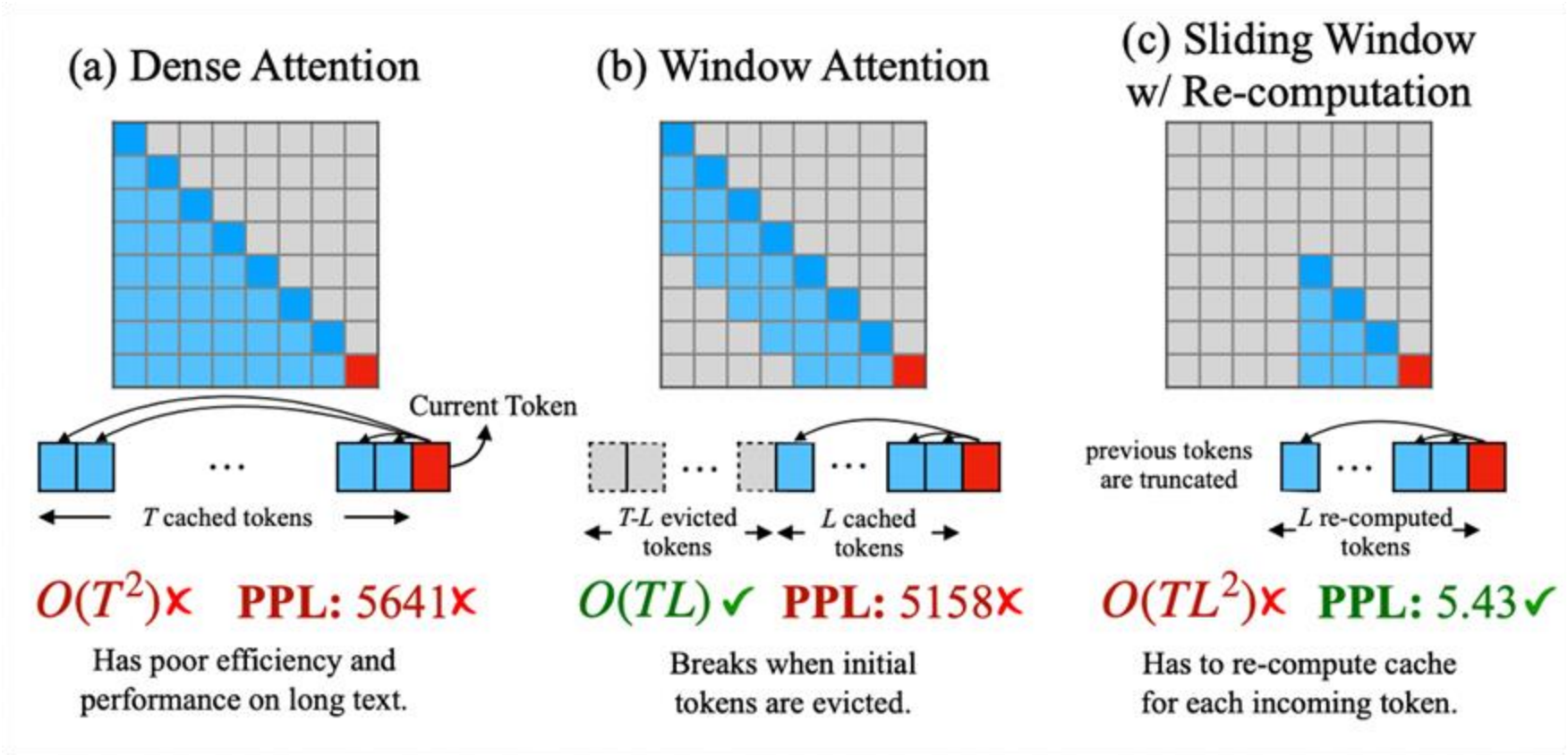
In total, we must store

$$\text{Batch\_size} * \text{seq\_len} * \text{num\_heads} * \text{num\_layers} * \text{emb\_dim} * 2$$

separate values in the kv cache

StreamingLLM

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

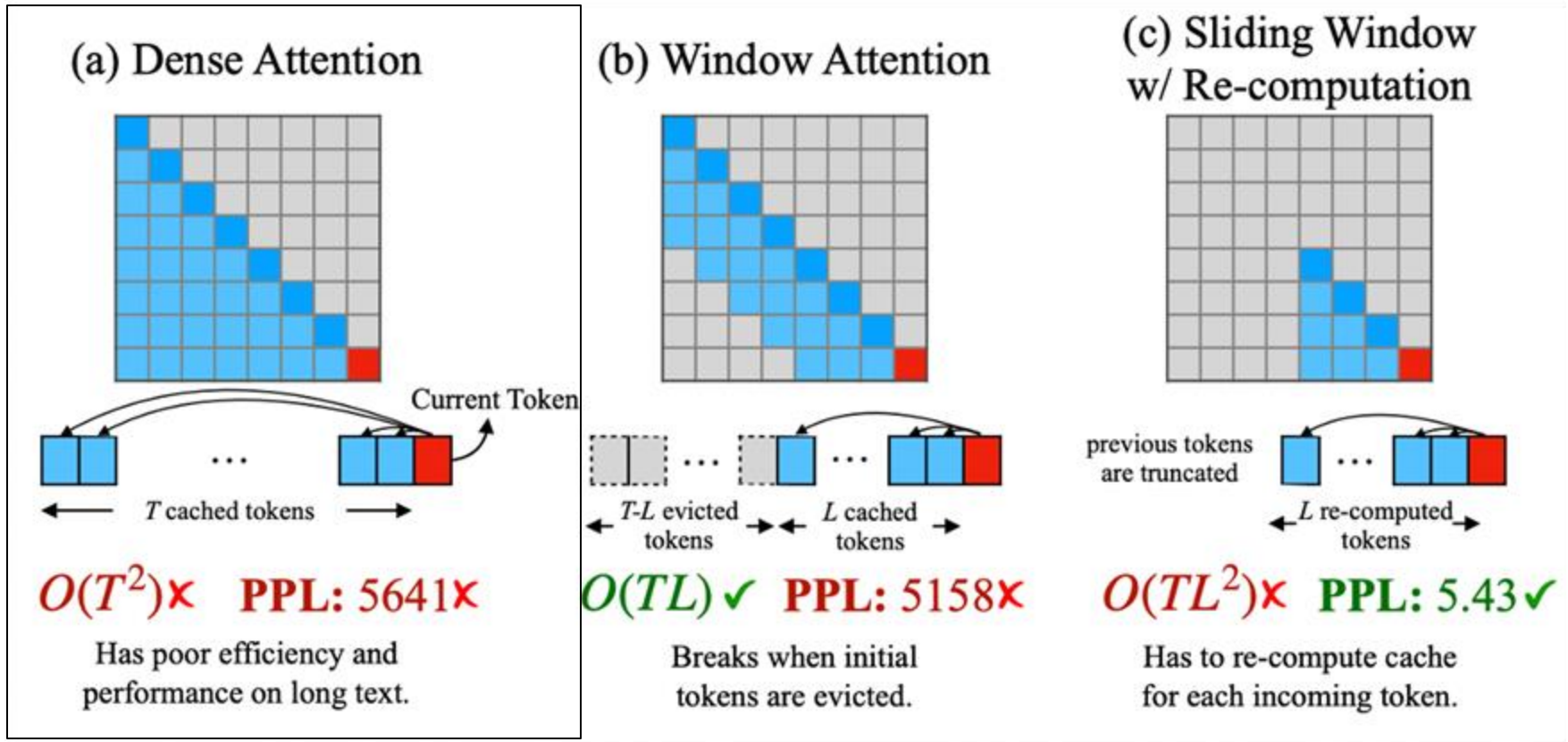


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

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

Too much storage

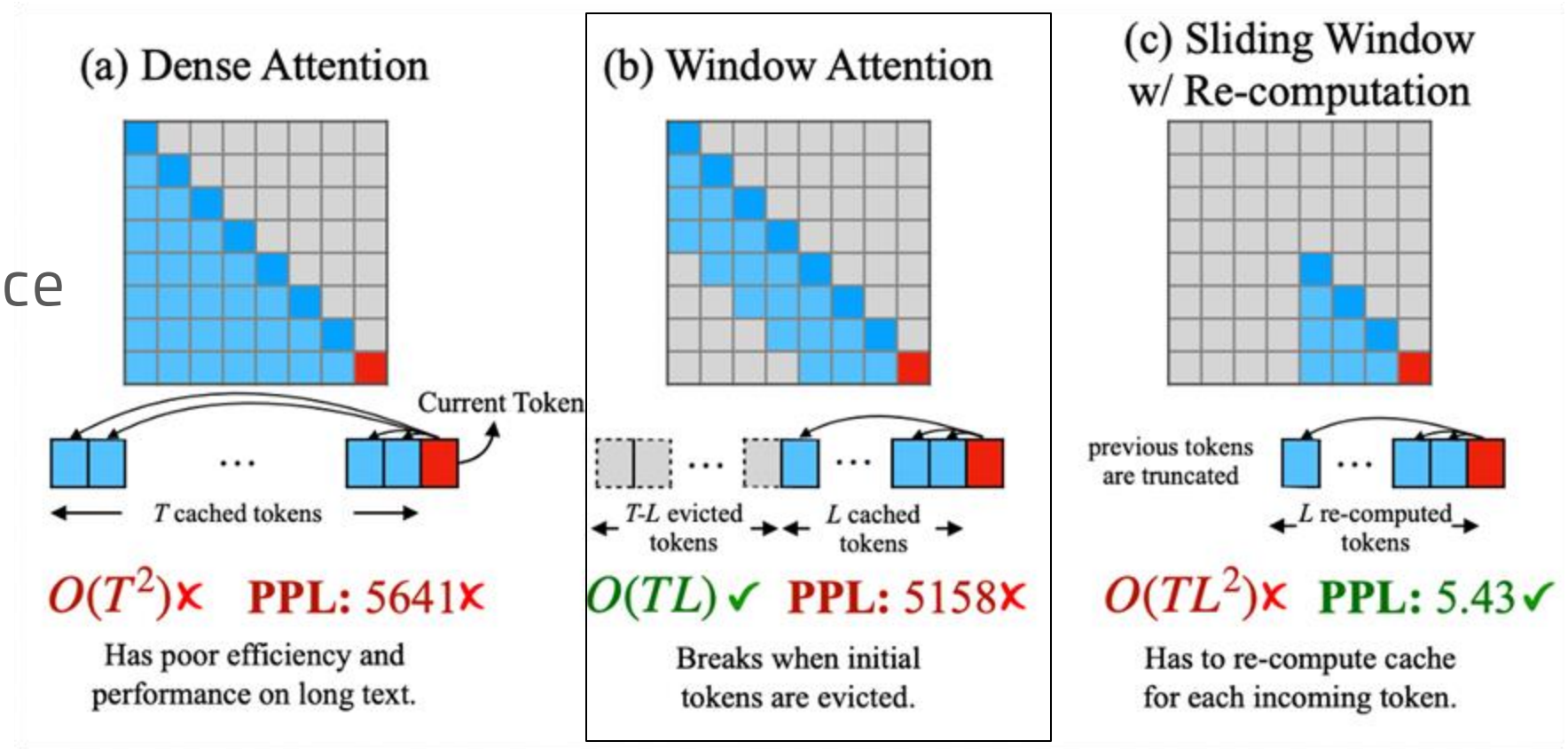


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

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

Bad performance

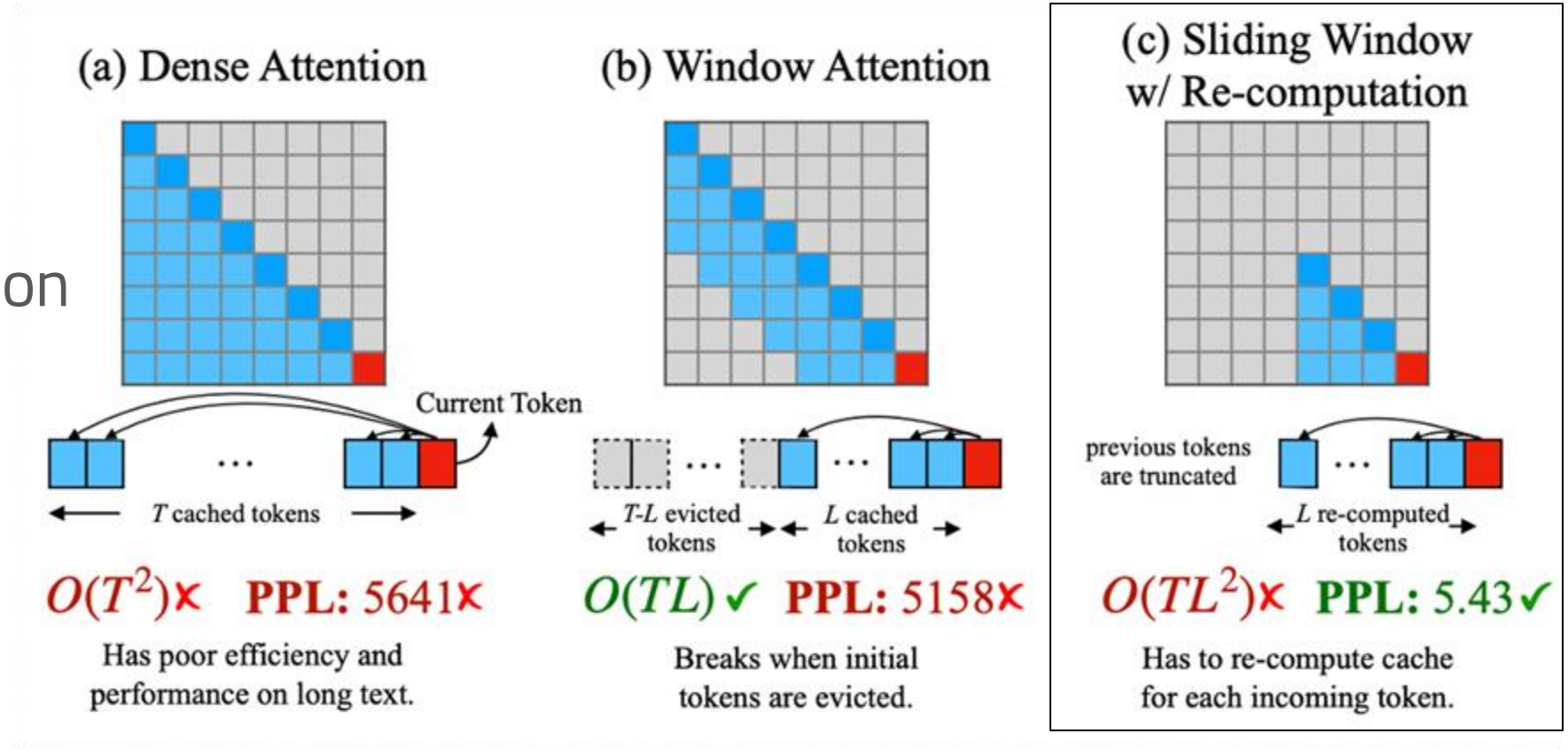


Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM

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

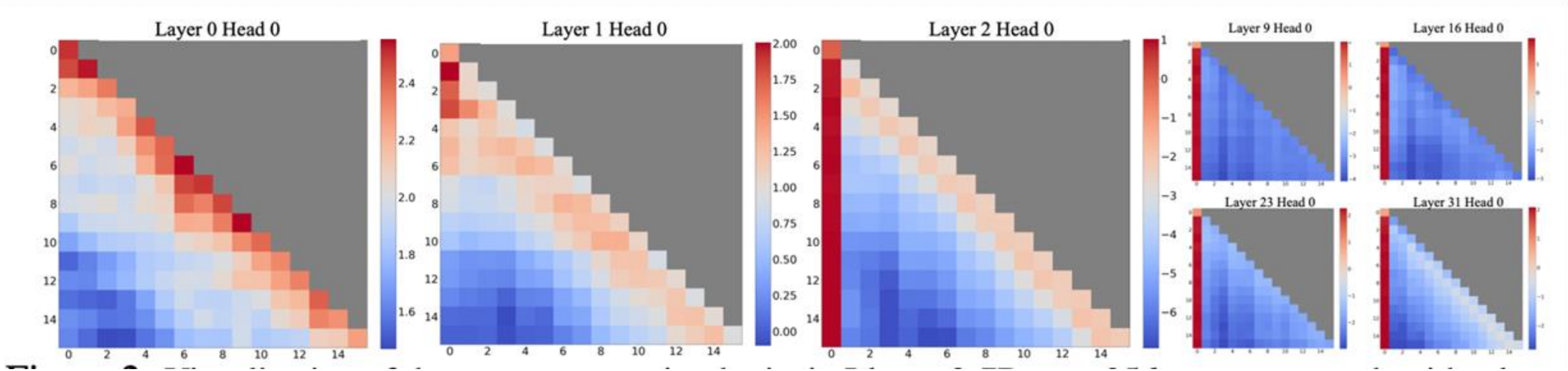
Too much recomputation



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

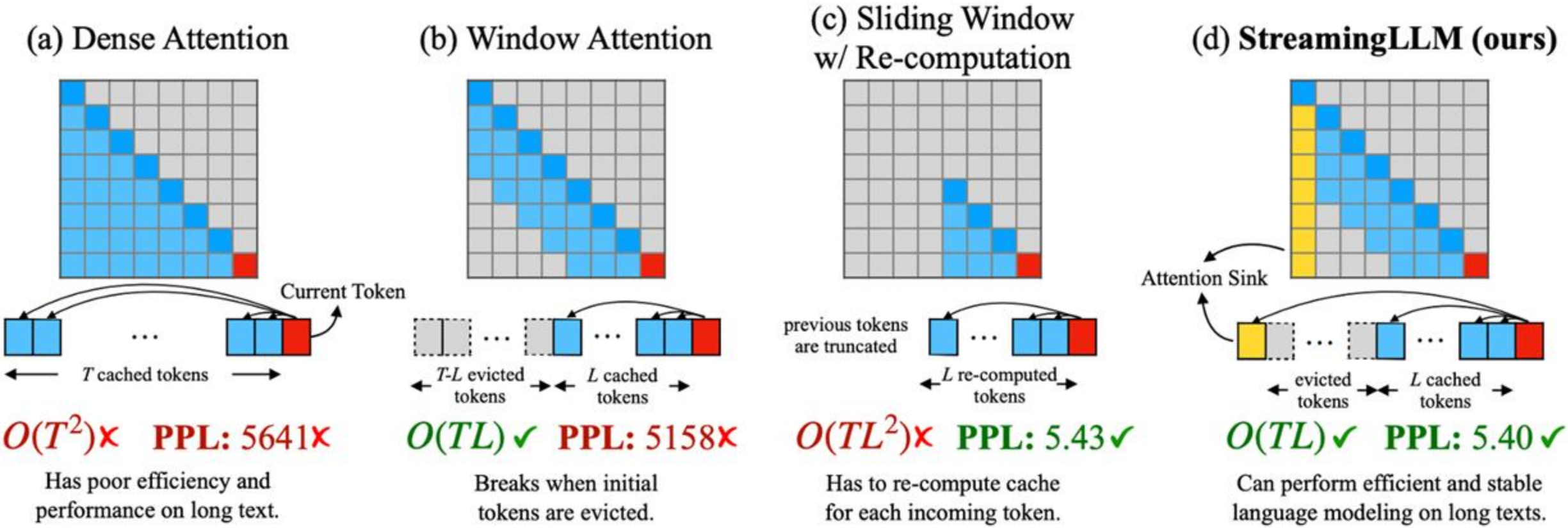
StreamingLLM

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



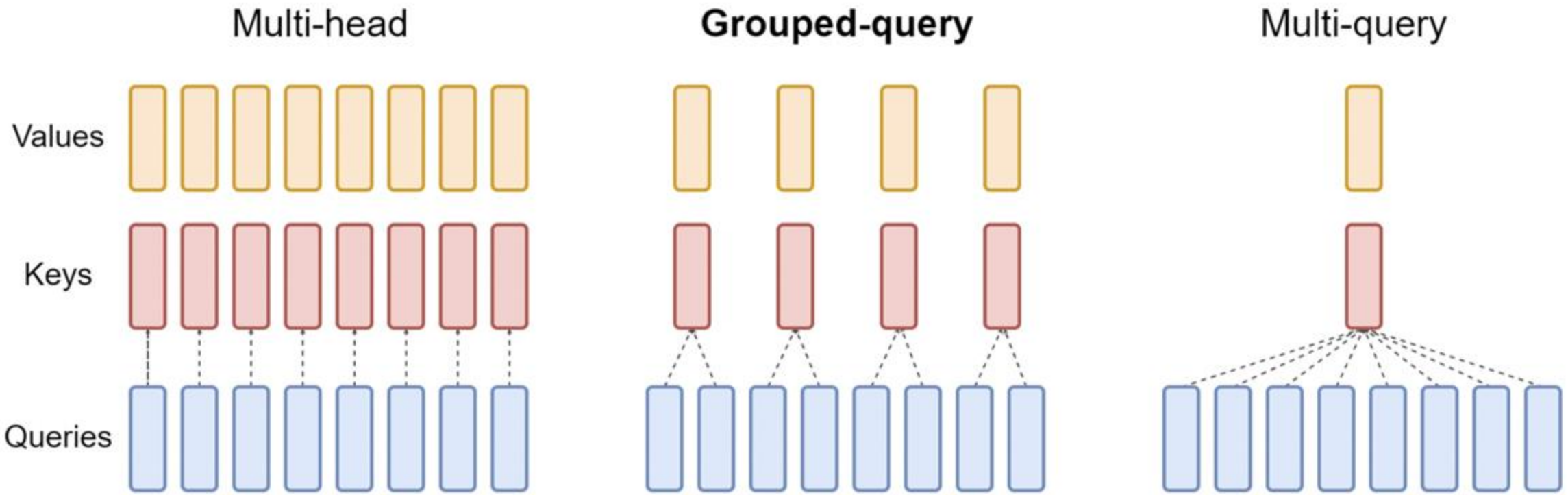
Efficient Streaming Language Models with Attention Sinks [Xiao et al., 2023]

StreamingLLM



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

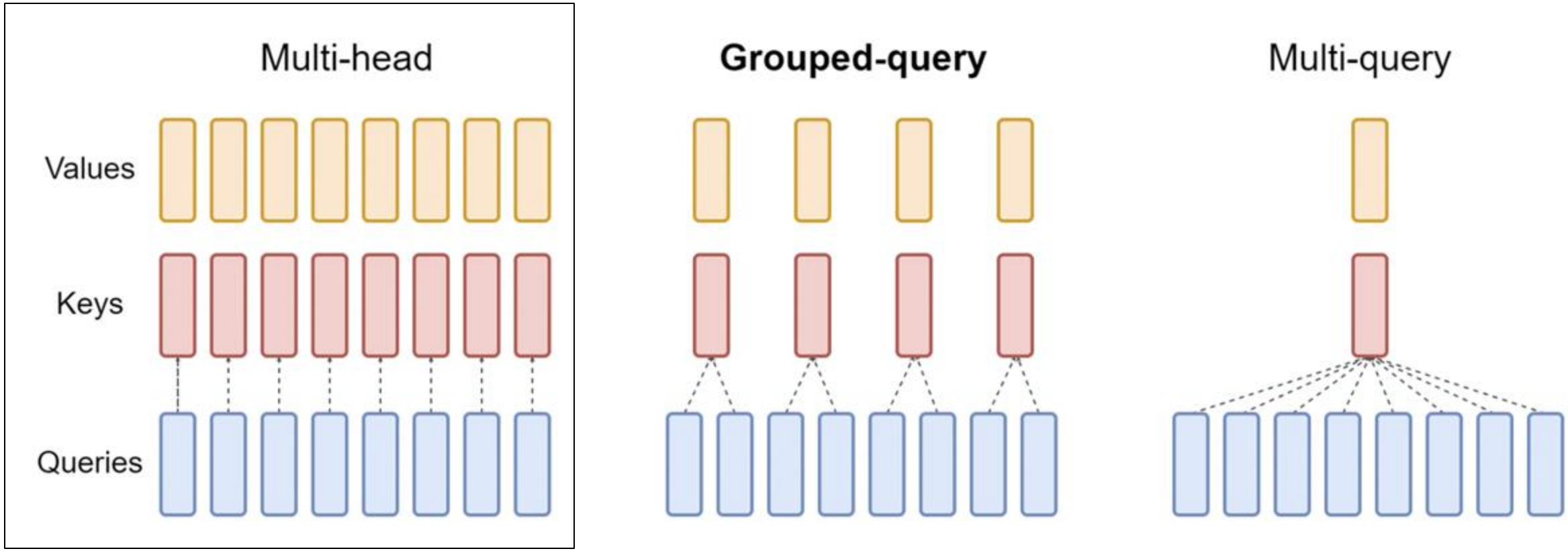
MHA/GQA/MQA



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

MHA/GQA/MQA

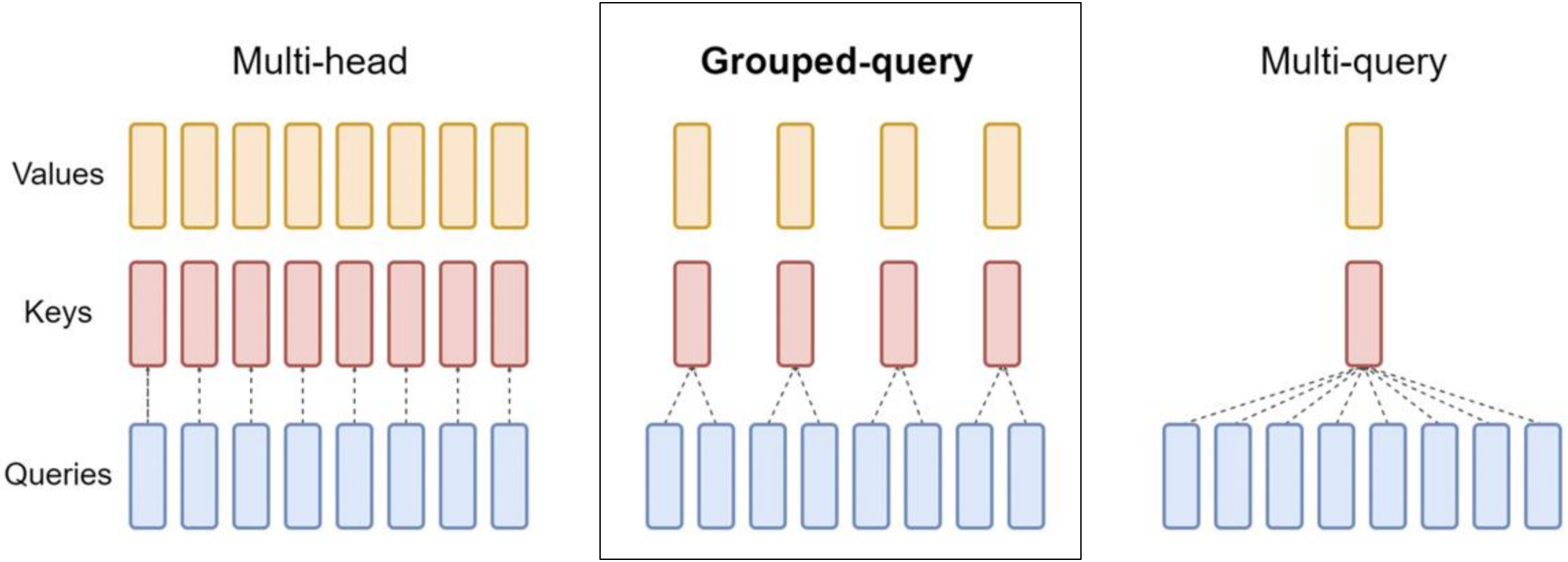
Each attention head calculates separate keys and values for each token



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

MHA/GQA/MQA

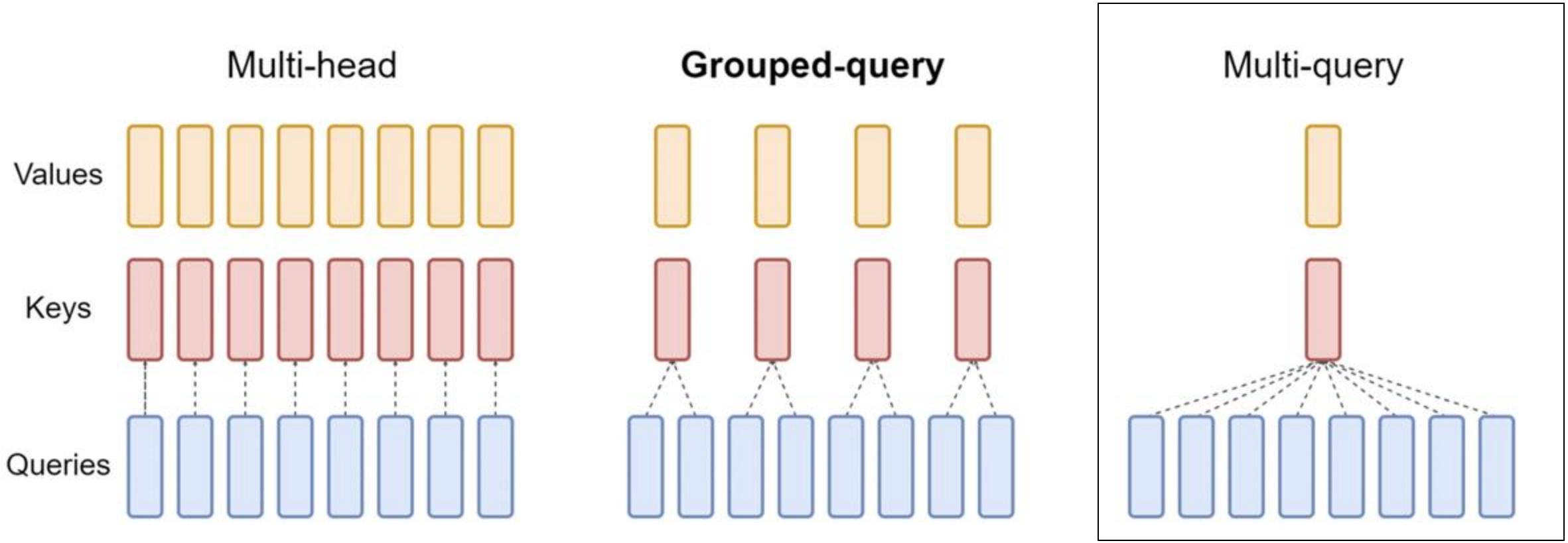
Attention heads are split into groups.
Each group has one key/value per token.



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

MHA/GQA/MQA

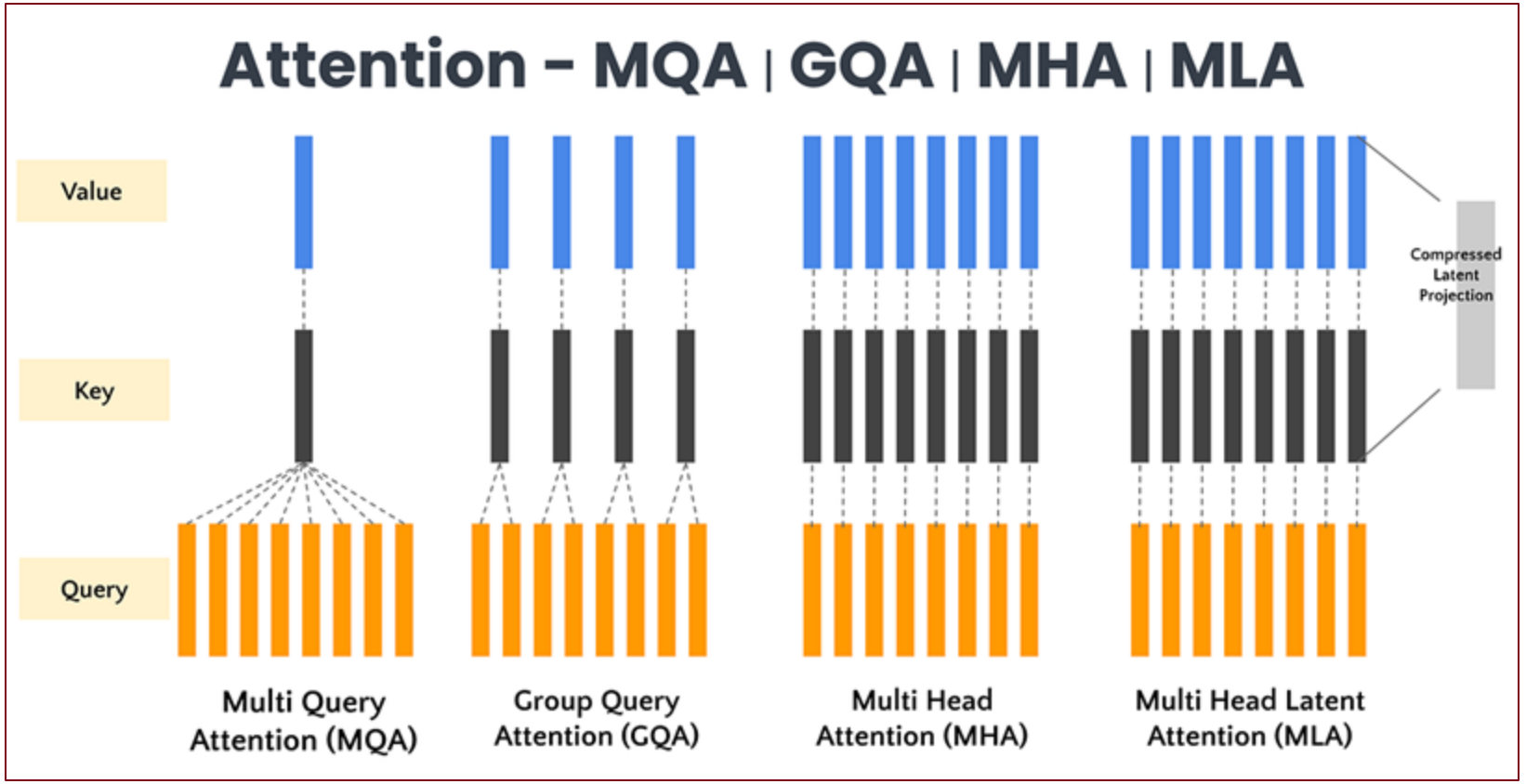
Attention heads share the same keys and values for each token



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

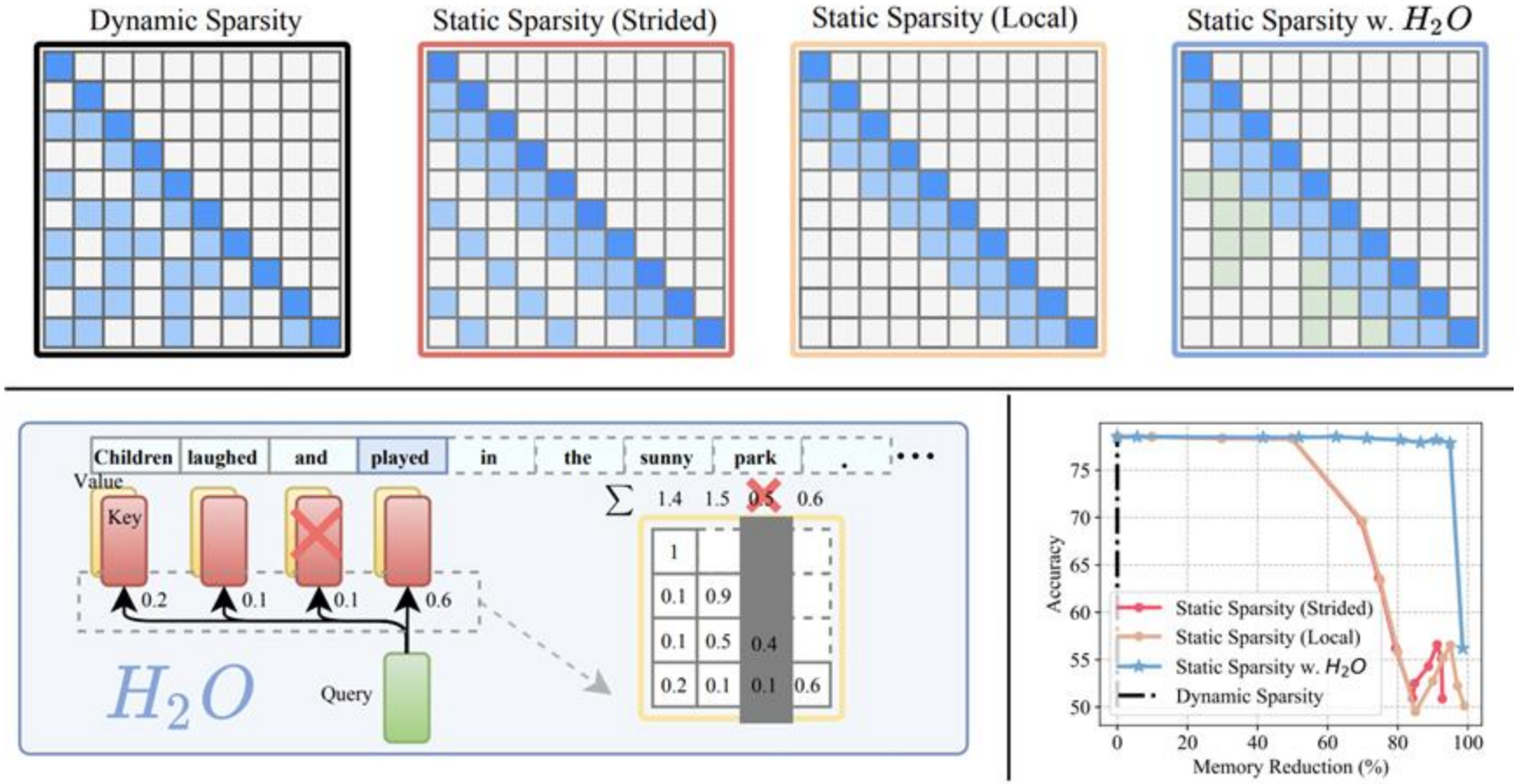
MHA Latent Query Attention

Some modern models (such as DeepSeek) train with what is called latent query attention → Keys + Values are compressed to a lower dimension to reduce storage costs



H_2O: Heavy Hitter Oracle

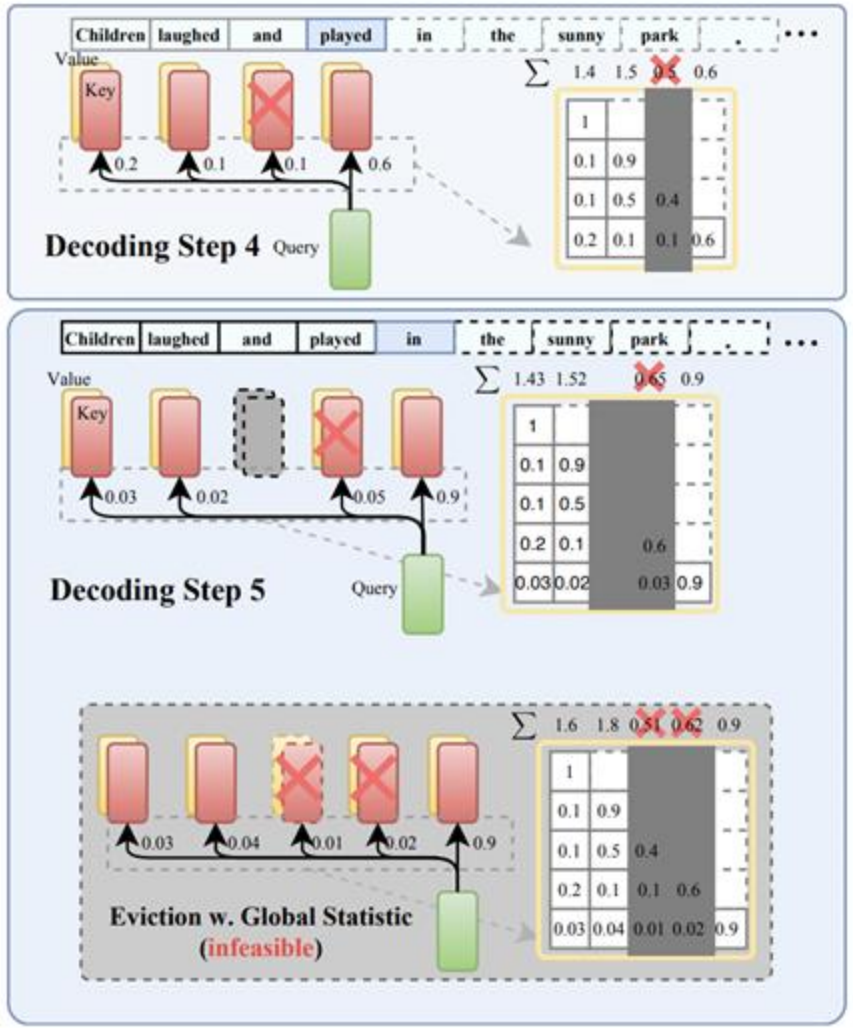
Evict all but k-highest cumulative attention tokens from cache



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]

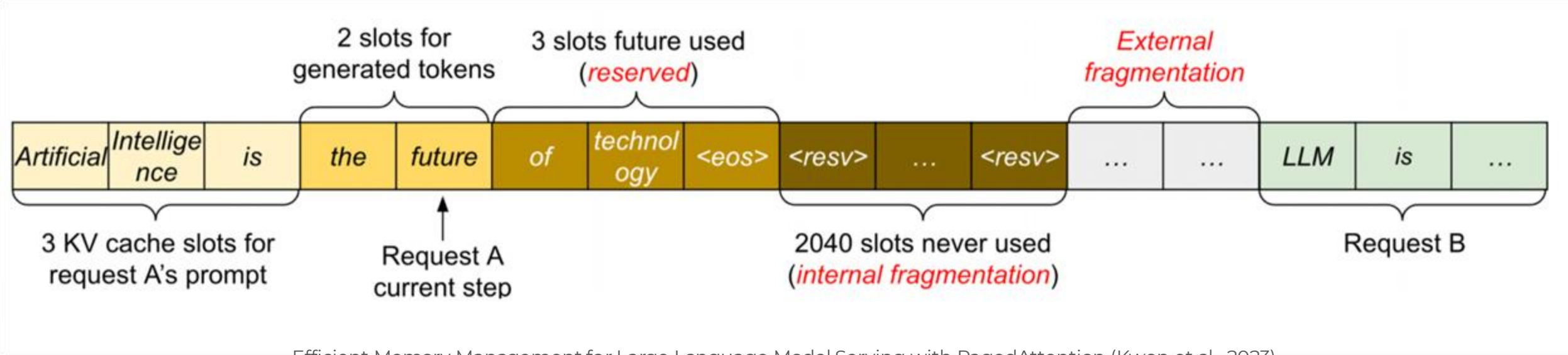
Efficient LLMs

- ❑ Quantization
- ❑ Sparsity
- ❑ Long Context
- ❑ **Serving & Systems**
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

PagedAttention

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

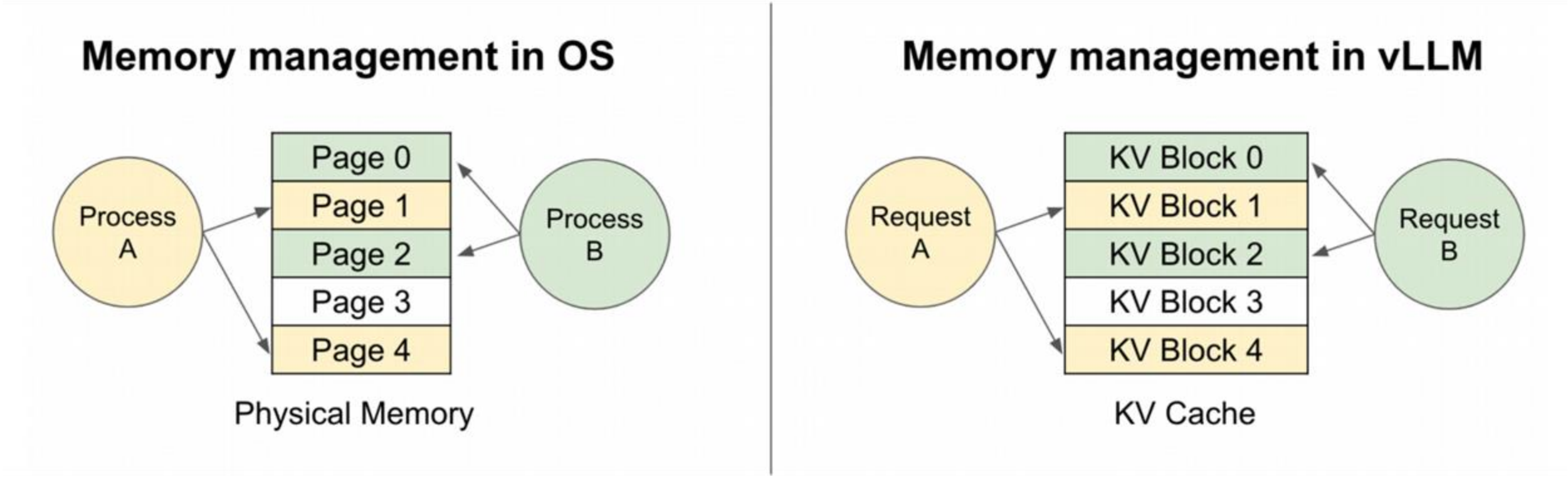
-Originally, most systems just assign fixed sized blocks of memory to each incoming request. How to improve?



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

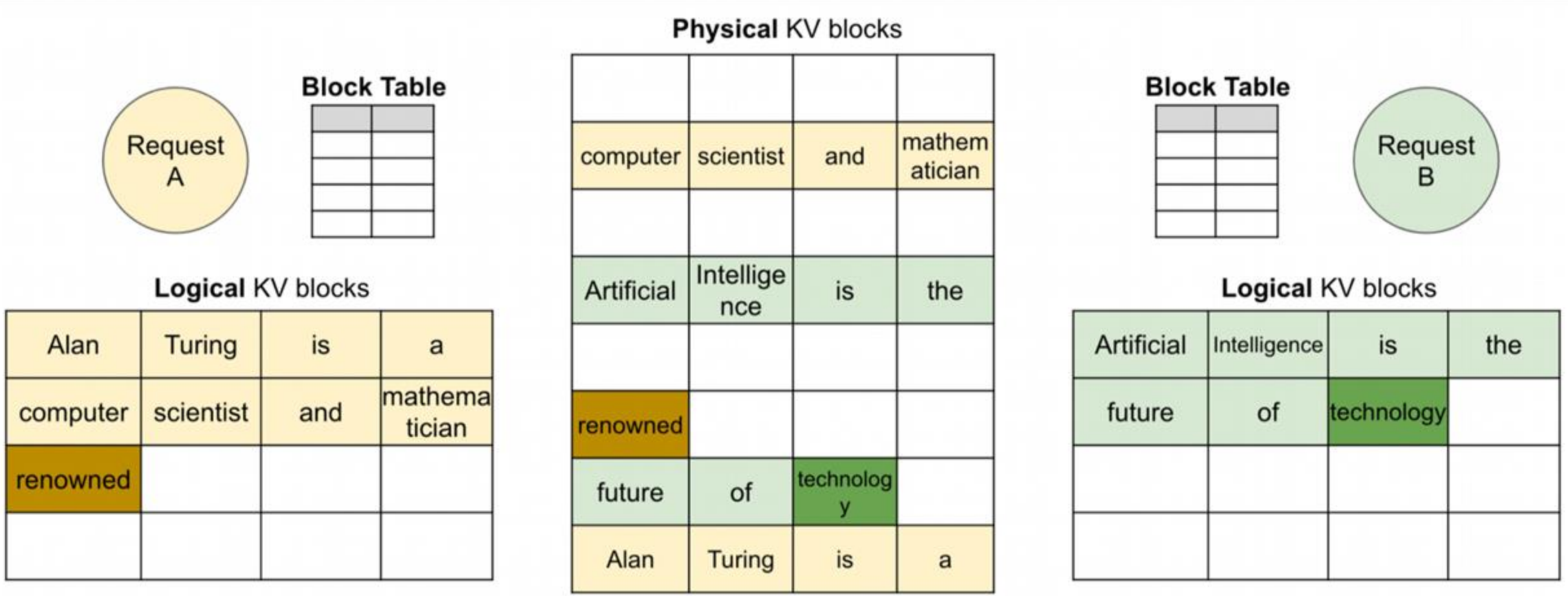
PagedAttention

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



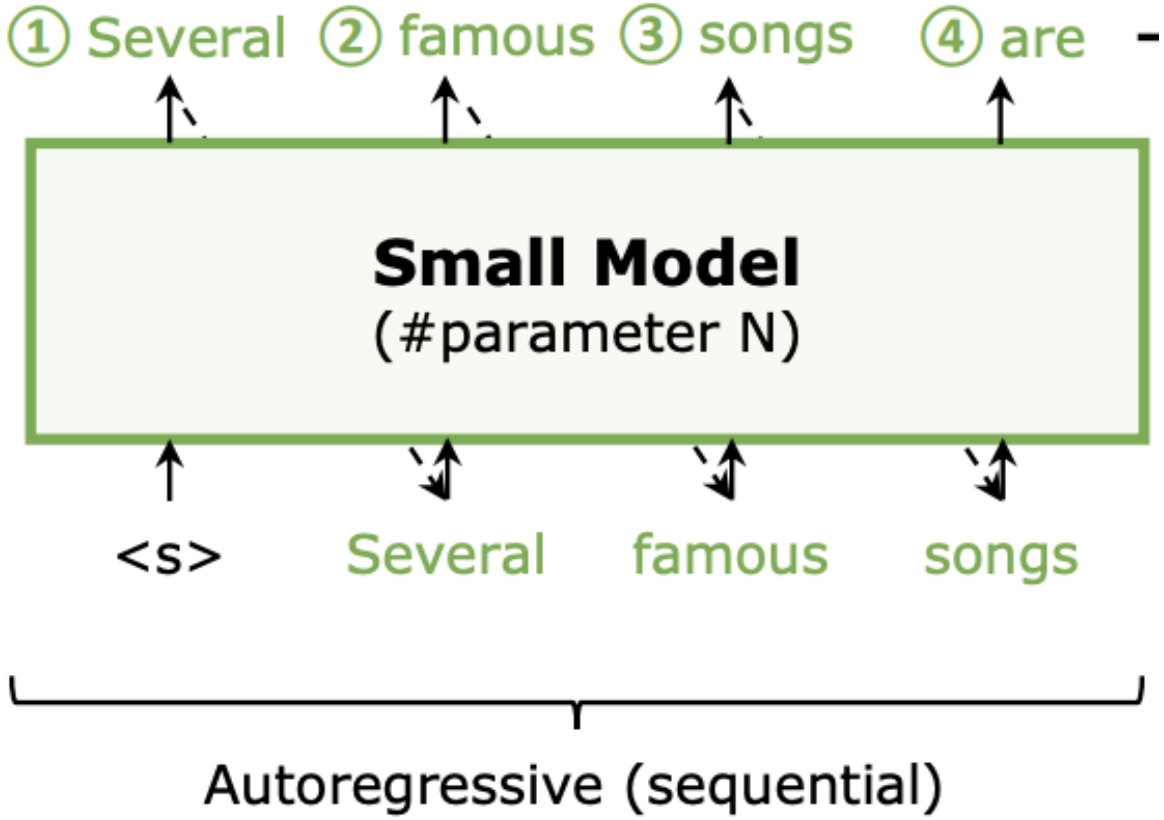
Efficient Memory Management for Large Language Model Serving with PagedAttention (Kwon et al., 2023)

Examples of Modern Inference Systems

vLLM, sglang, and TensorRT-LLM are all examples of inference engines which take advantage of this approach. If you are serving models in any setting, you should use one of the above (rather than directly instantiating some huggingface or pytorch model)

Speculative Decoding

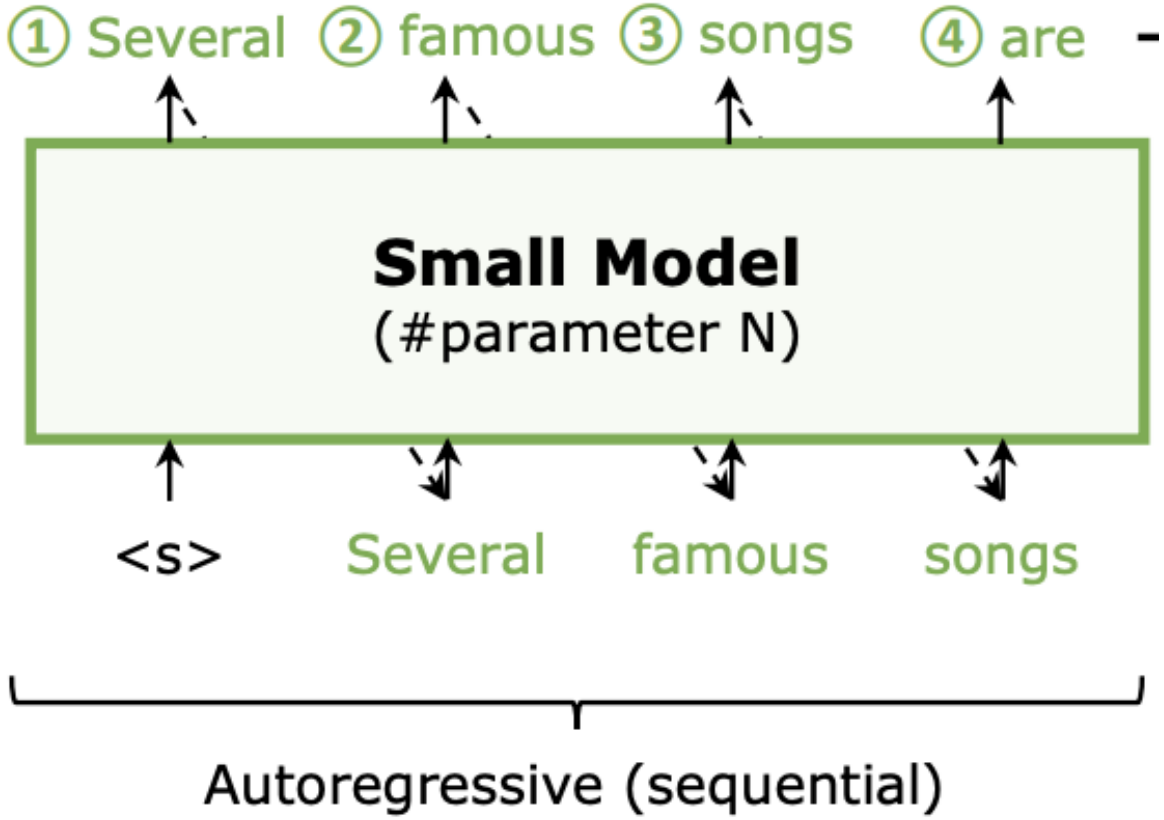
- In standard autoregressive decoding, we are only using each parameter ***one time*** when the batch size is 1
- This means standard decoding has a ***low*** arithmetic intensity and is memory bound
- We have a bunch more compute we could be getting for free given how massively parallel GPUs are



Big Little Decoder [Kim, et. al]

Speculative Decoding

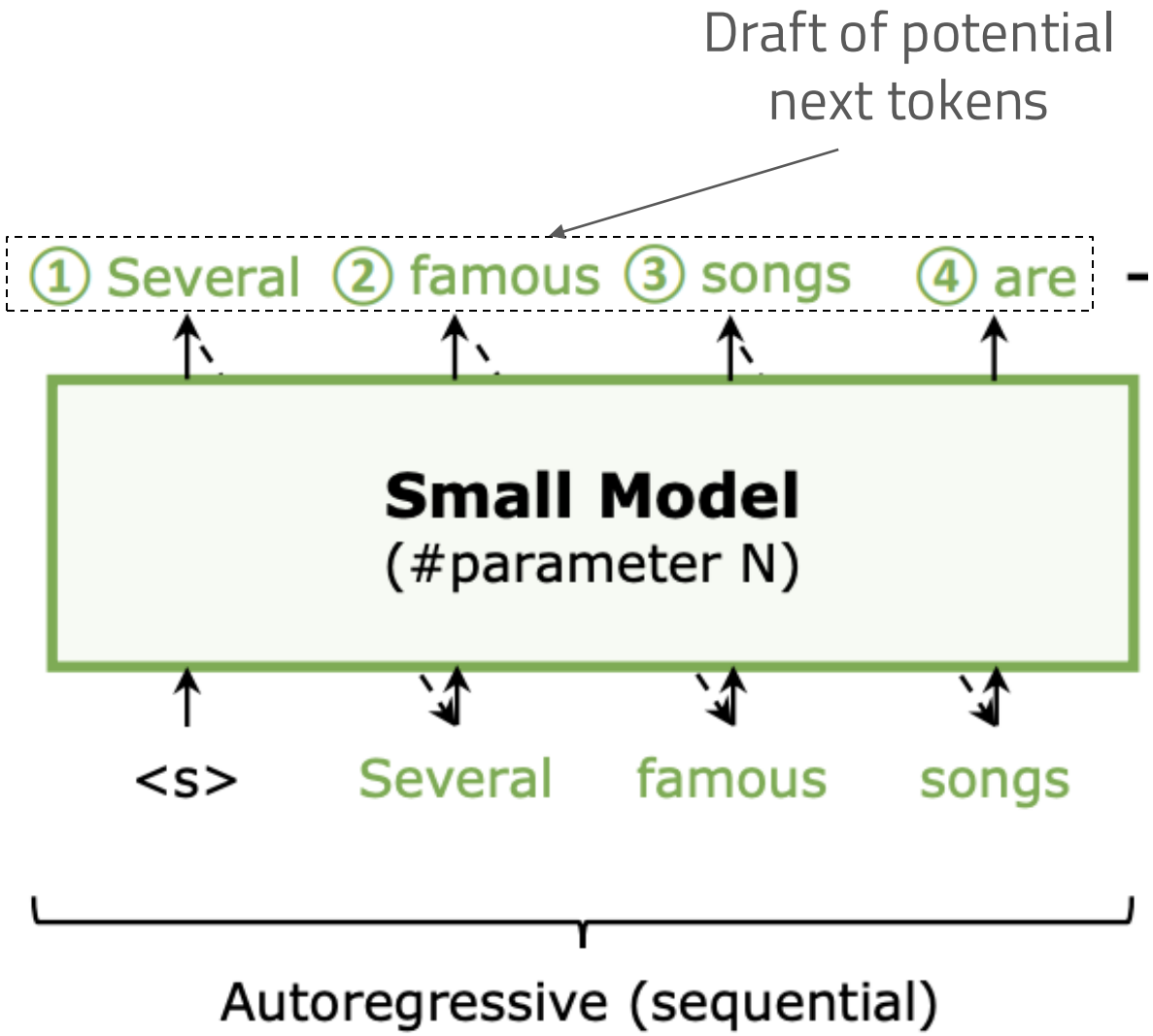
- Speculative decoding resolves this by 'speculating' multiple tokens into the future with a smaller, cheaper model



Big Little Decoder [Kim, et. al]

Speculative Decoding

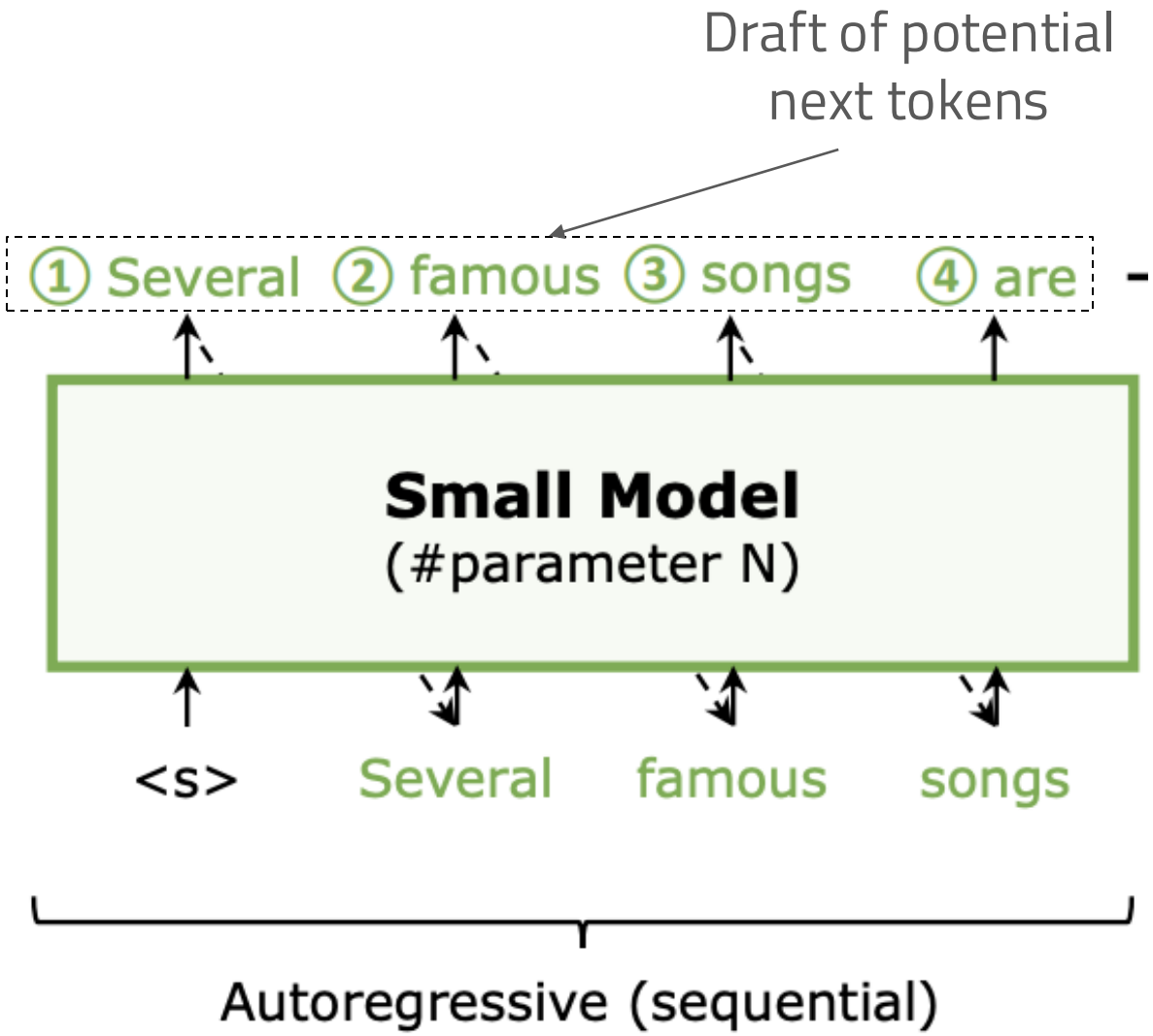
- Speculative decoding resolves this by ‘speculating’ multiple tokens into the future with a smaller, cheaper model



Big Little Decoder [Kim, et. al]

Speculative Decoding

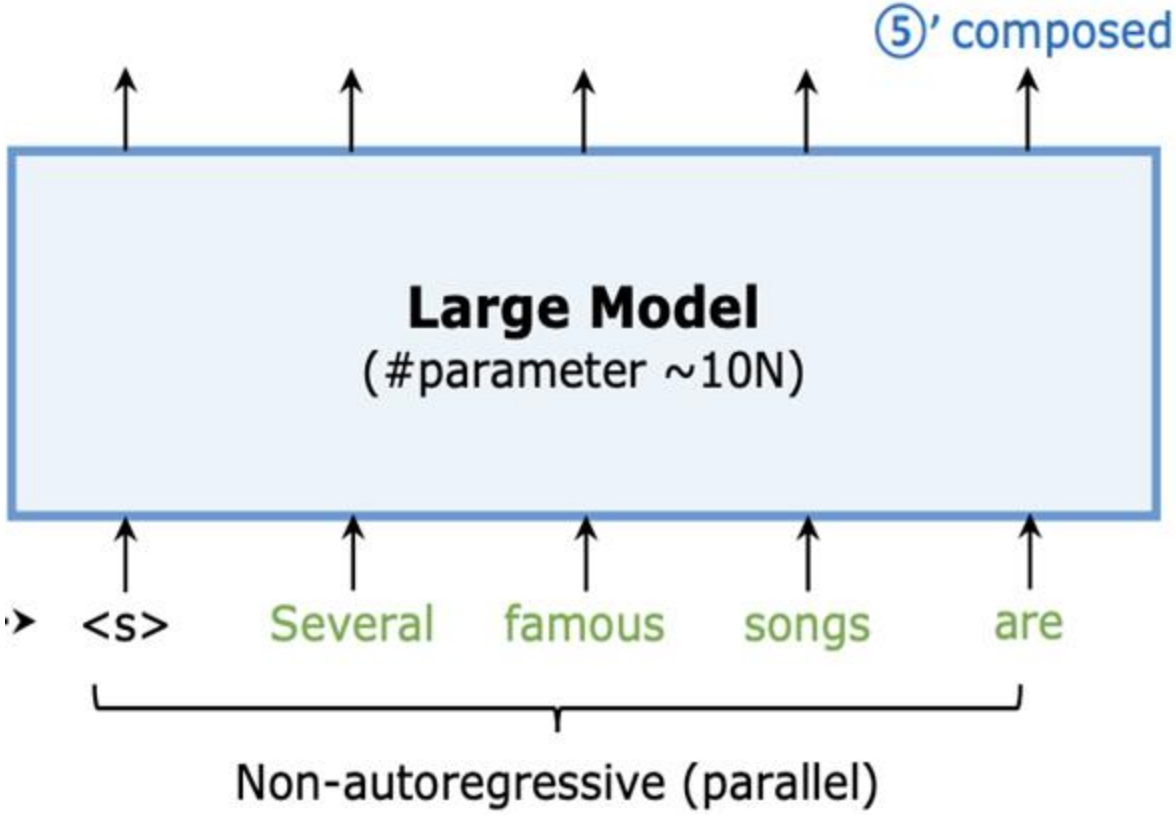
- Speculative decoding resolves this by 'speculating' multiple tokens into the future with a smaller, cheaper model
- We can now send this set of tokens on to a much larger model to verify the sequence



Big Little Decoder [Kim, et. al]

Speculative Decoding

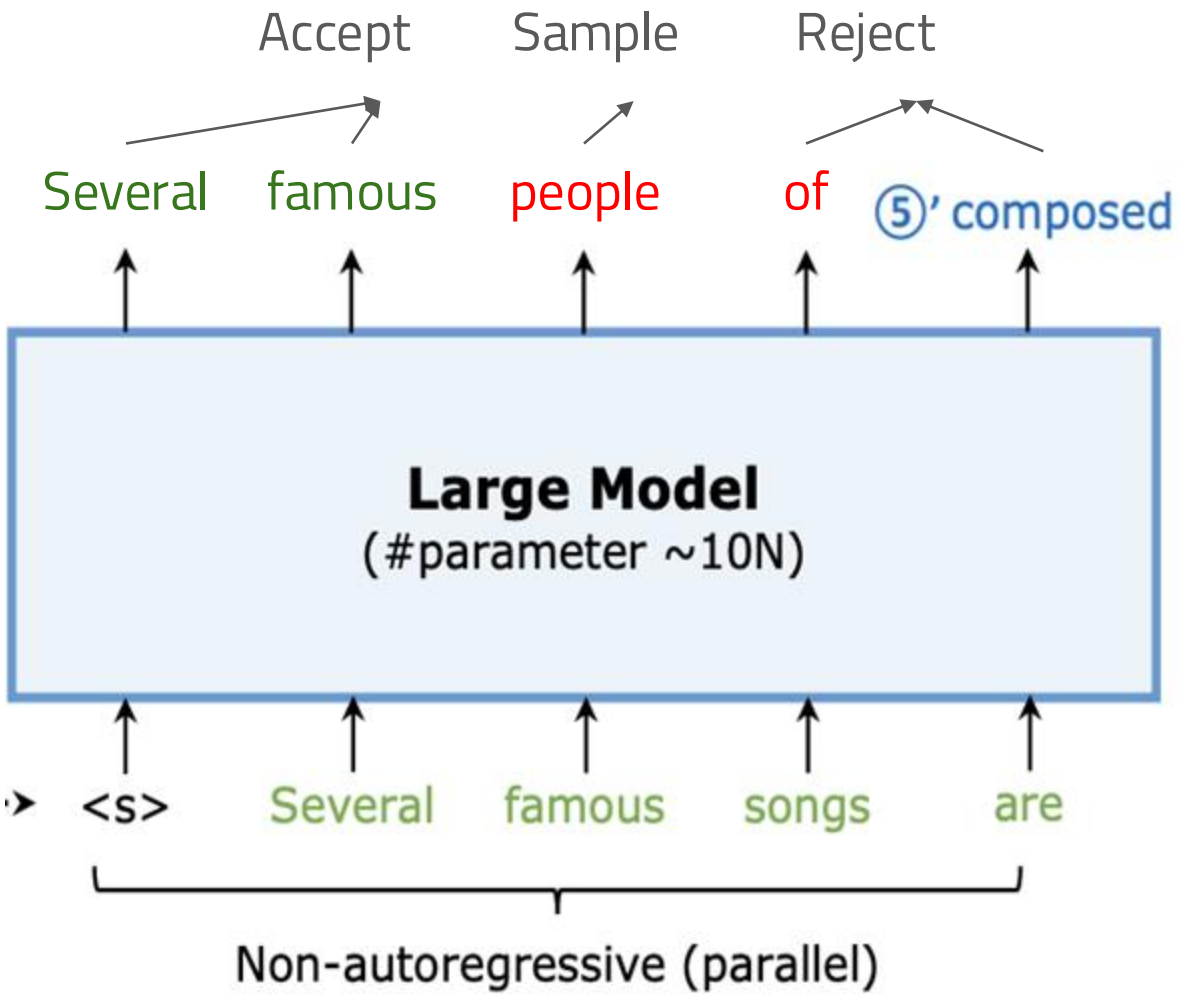
- Because the sequence will be run in parallel the arithmetic intensity will be proportional to the number of draft tokens
- We run each token through and see if the output of the large model matches that of the smaller, draft model



Big Little Decoder [Kim, et. al]

Speculative Decoding

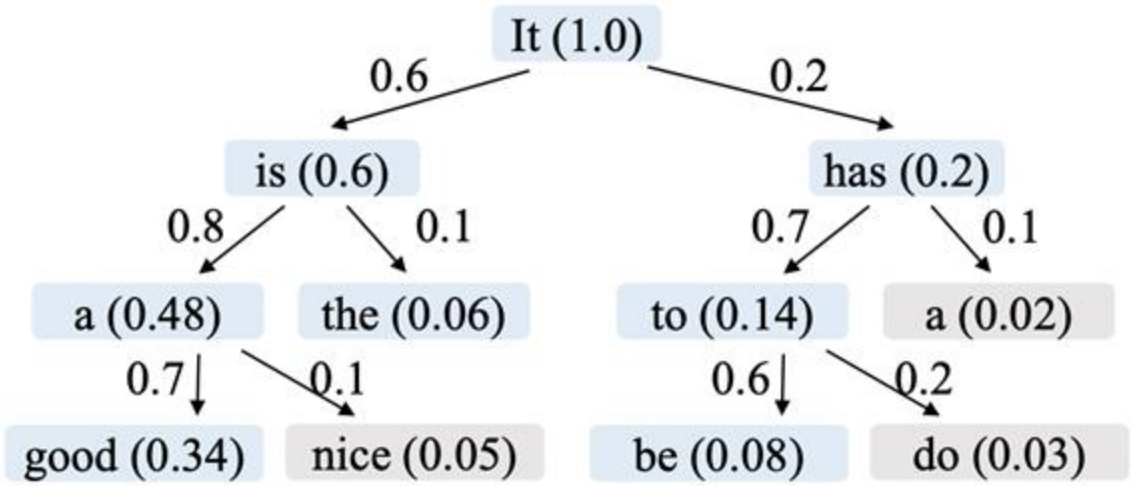
- Because the sequence will be run in parallel the arithmetic intensity will be proportional to the number of draft tokens
- We run each token through and see if the output of the large model matches that of the smaller, draft model
- We accept the matching tokens



Big Little Decoder [Kim, et. al]

Advanced Approaches

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



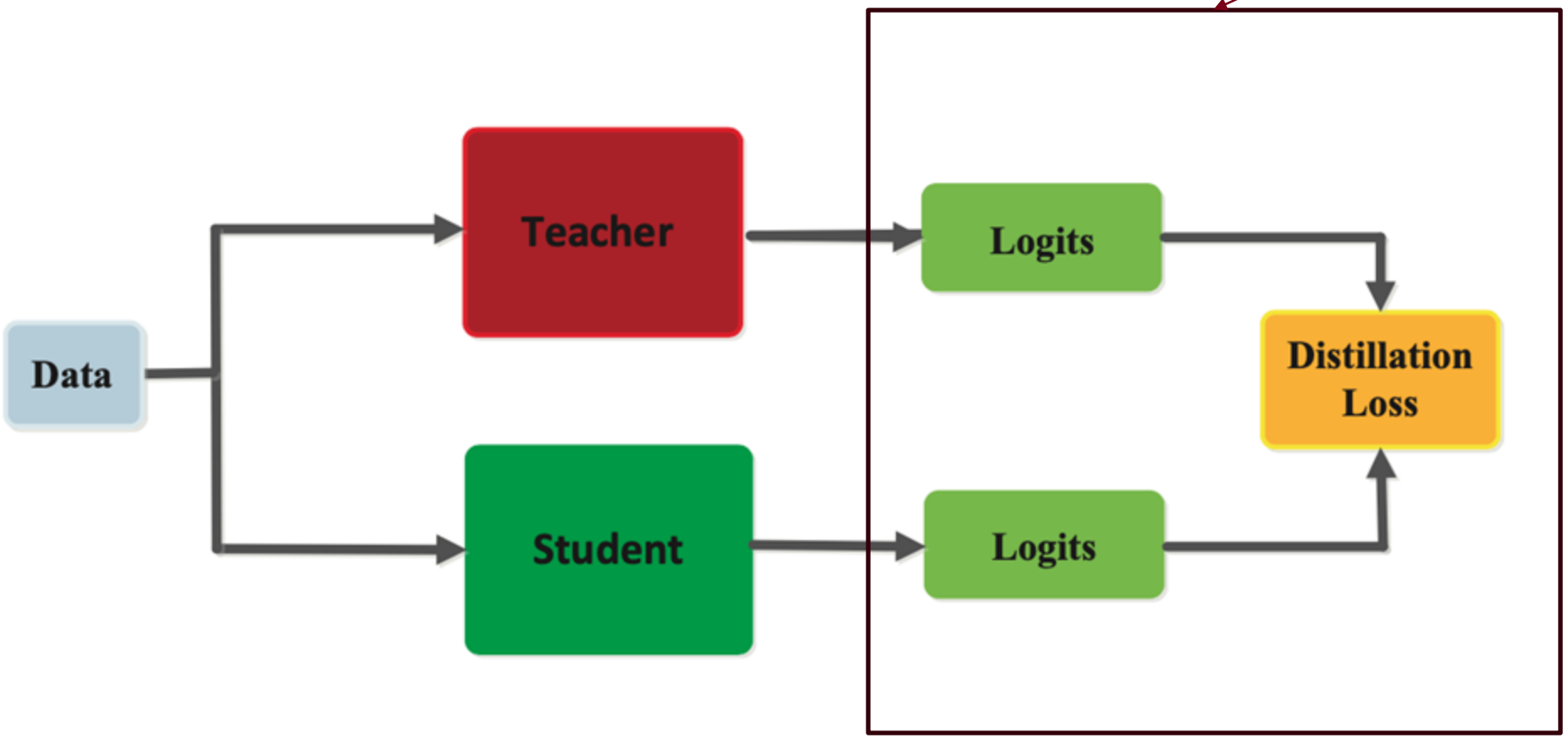
Big Little Decoder [Kim, et. al]

Efficient LLMs

- ❑ Quantization
- ❑ Sparsity
- ❑ Long Context
- ❑ Serving & Systems
- ❑ **Distillation**
- ❑ Parameter Efficient Fine-Tuning
- ❑ Alternative Paradigms

Logit Distillation

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

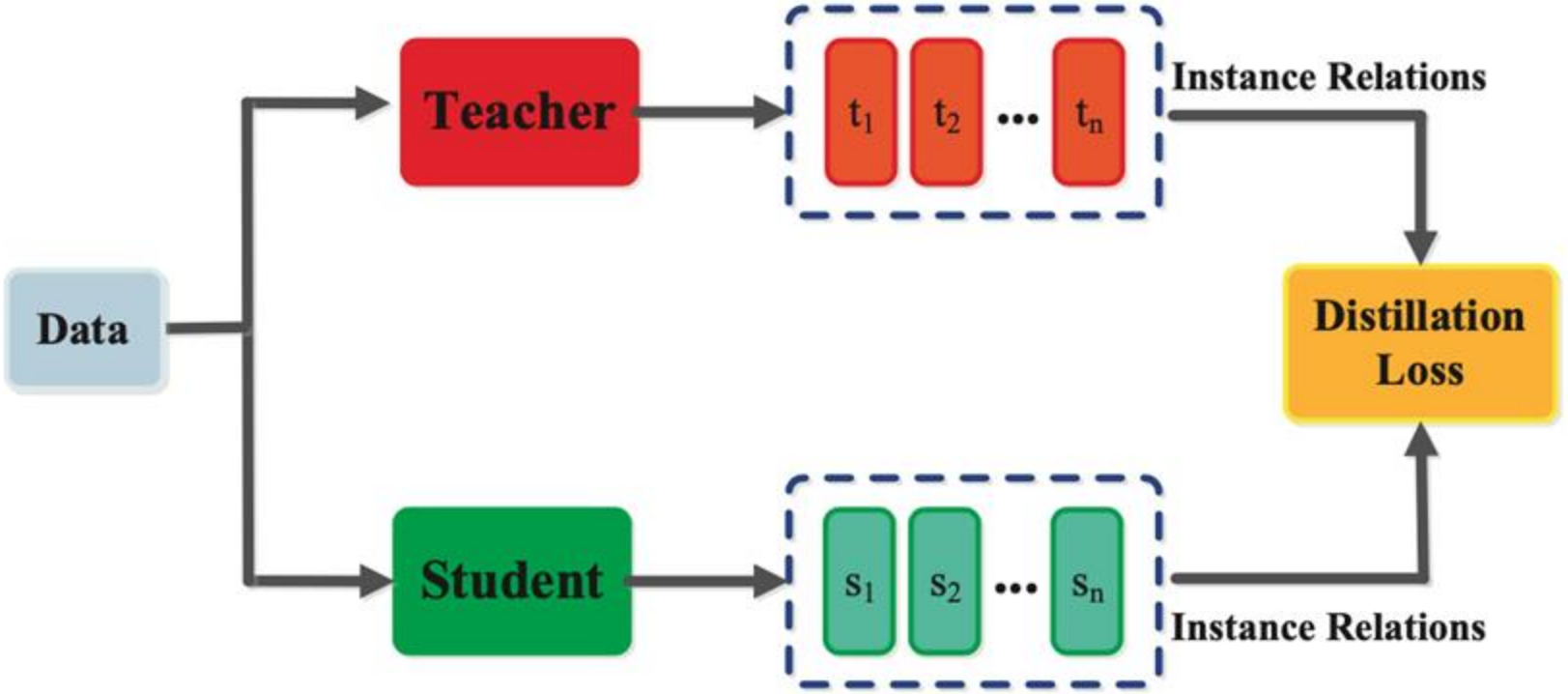


Logit Distillation

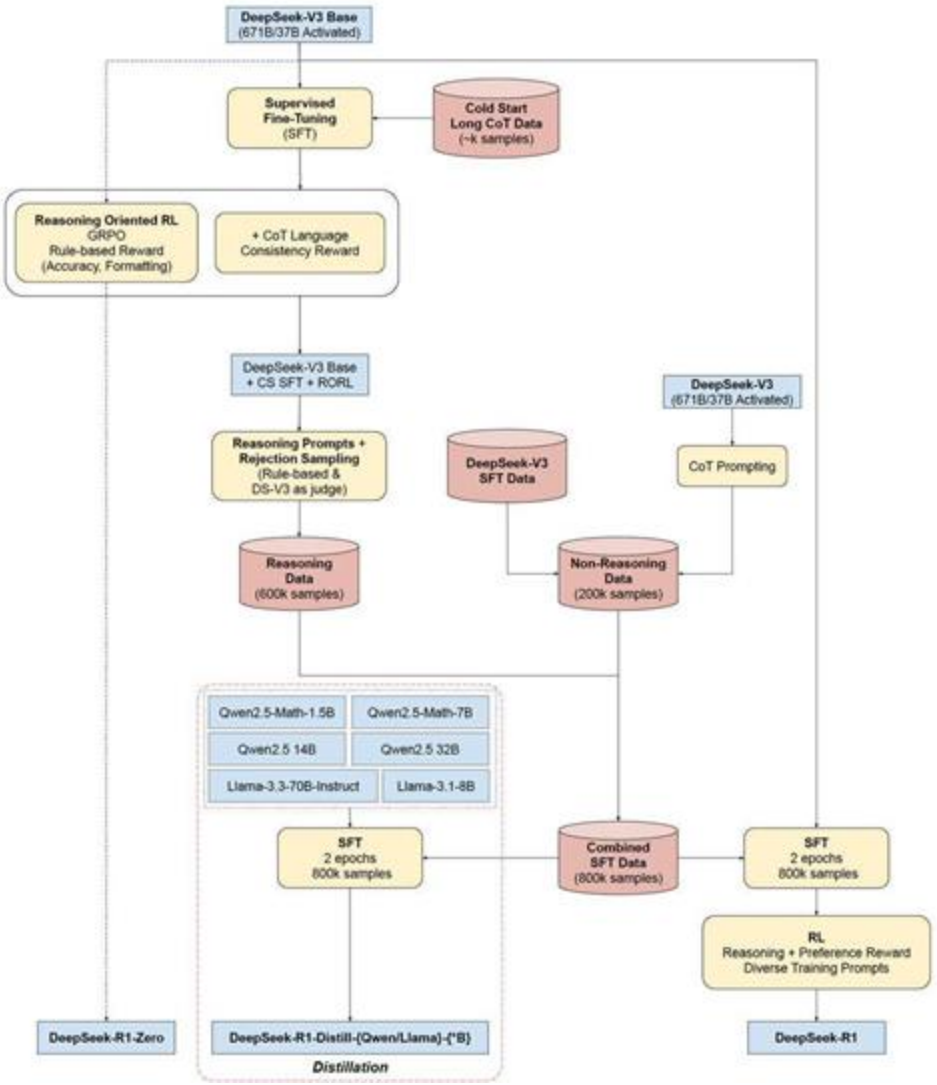
Both Gemma3 & Llama4 use logit distillation during pretraining

Layer Wise Distillation (LWD)

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

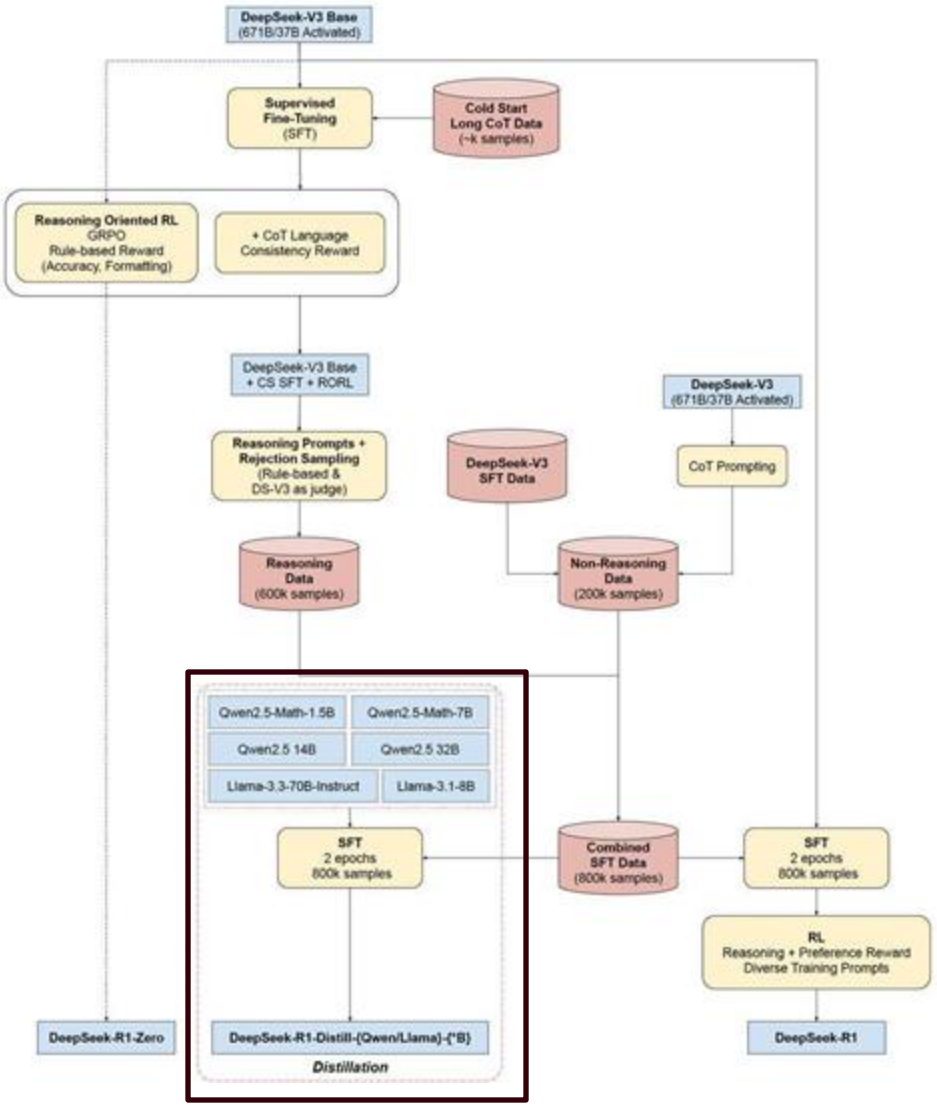


Output Tokens as Distillation (Synthetic Data)



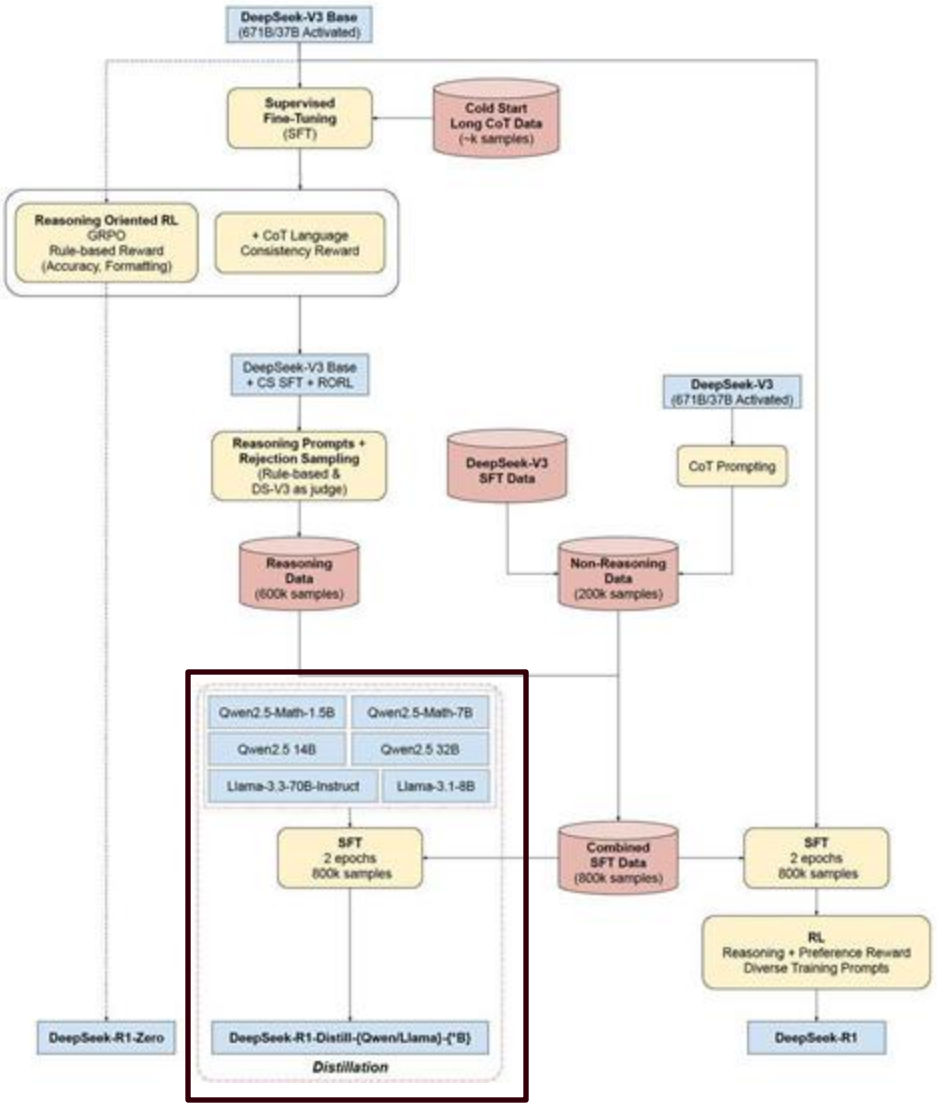
DeepSeekR1 outputs tokens for a given set of problems. In this case ‘distillation’, just means performing Supervised fine tuning on the tokens create by the larger V3 Models

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

Output Tokens as Distillation (Synthetic Data)



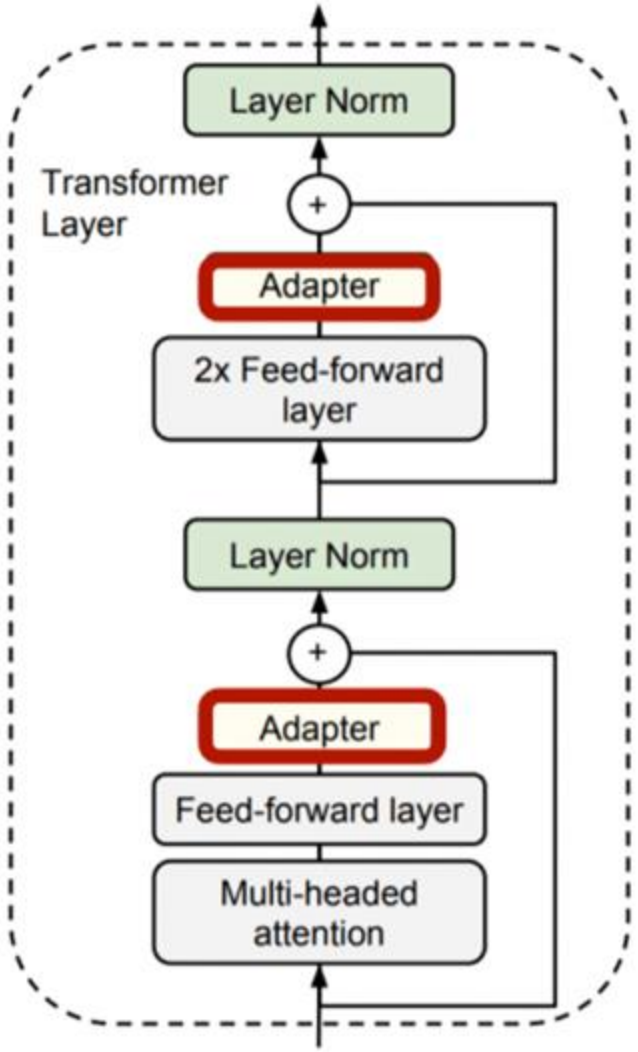
DeepSeekR1 outputs tokens for a given set of problems. In this case ‘distillation’, just means performing Supervised fine tuning on the tokens create by the larger V3 Models

Most small-model training today employs similar methods as these

Efficient LLMs

- ❑ Quantization
- ❑ Sparsity
- ❑ Long Context
- ❑ Serving & Systems
- ❑ Distillation
- ❑ **Parameter Efficient Fine-Tuning**
- ❑ Alternative Paradigms

Adapter



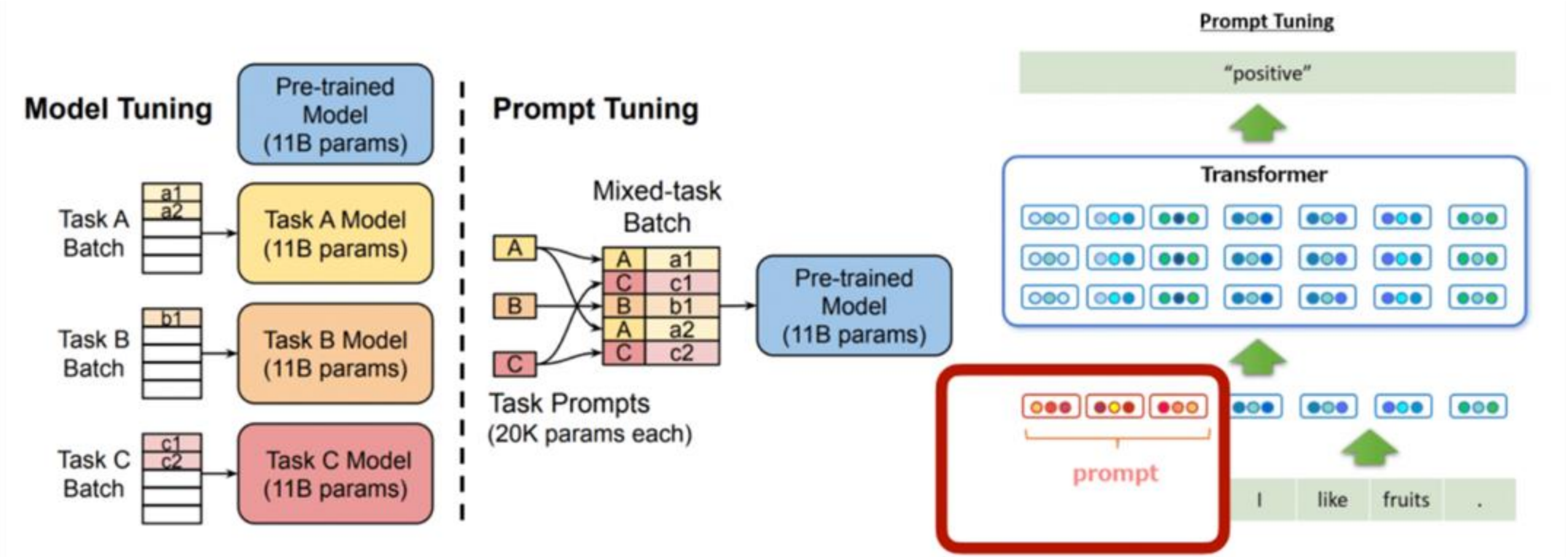
Add trainable layers after each feedforward layer

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	$MNLI_m$	$MNLI_{mm}$	QNLI	RTE	Total
$BERT_{LARGE}$	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]

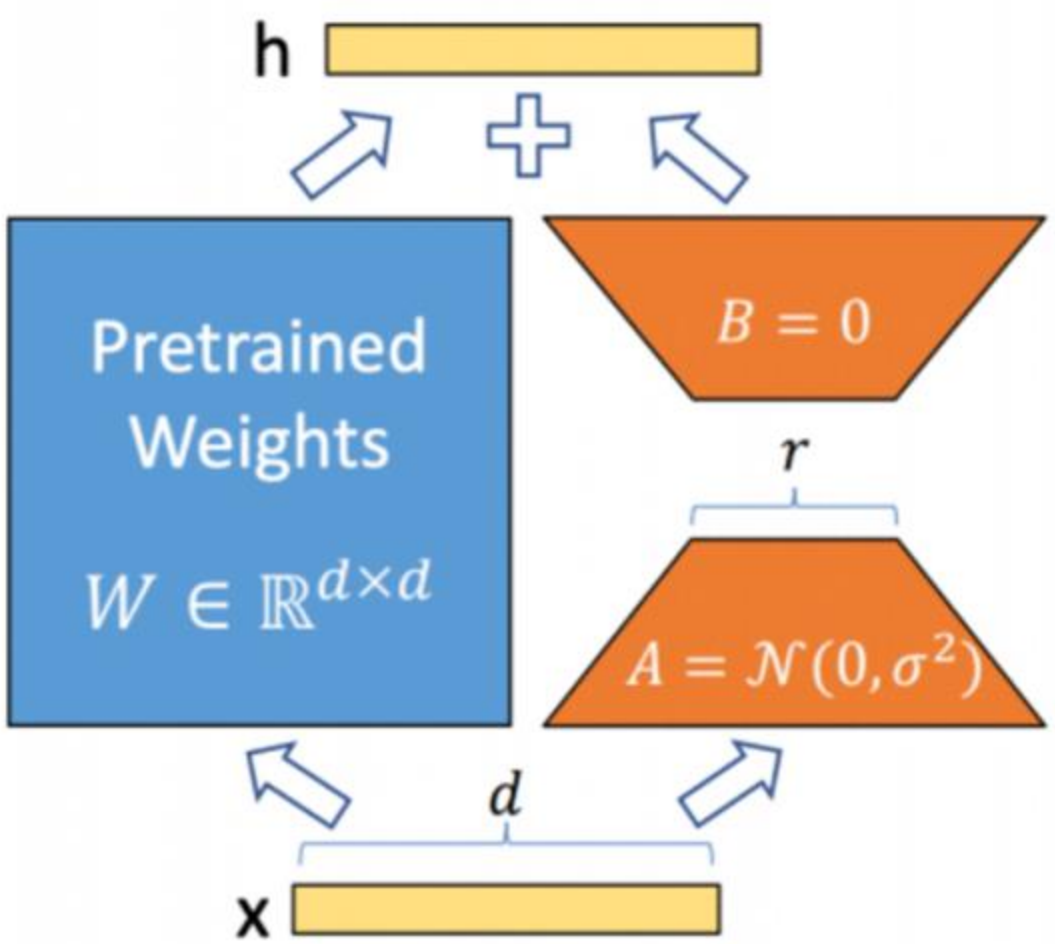
Prompt Tuning (Soft Prompting)

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]

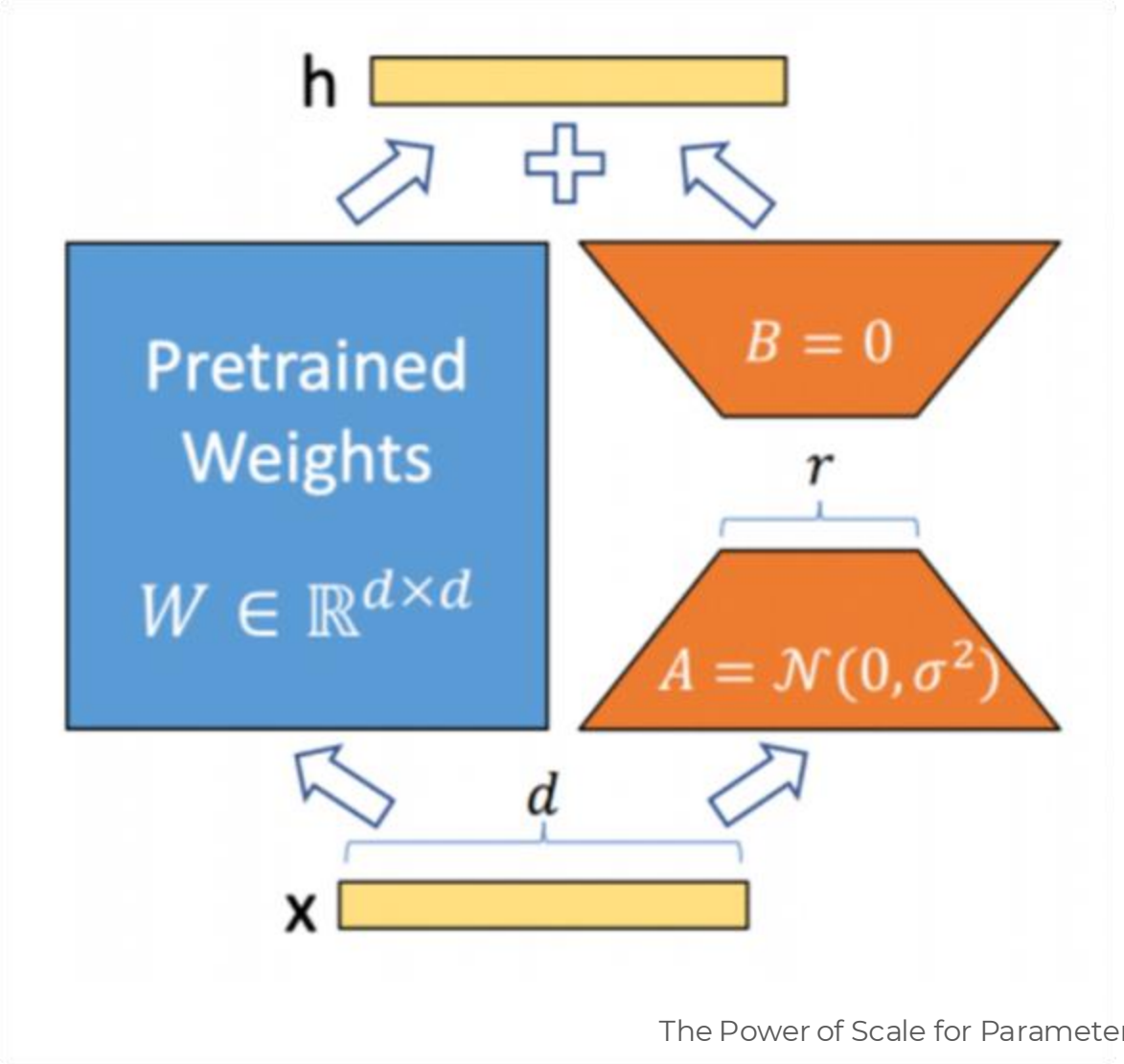
LoRA



- 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



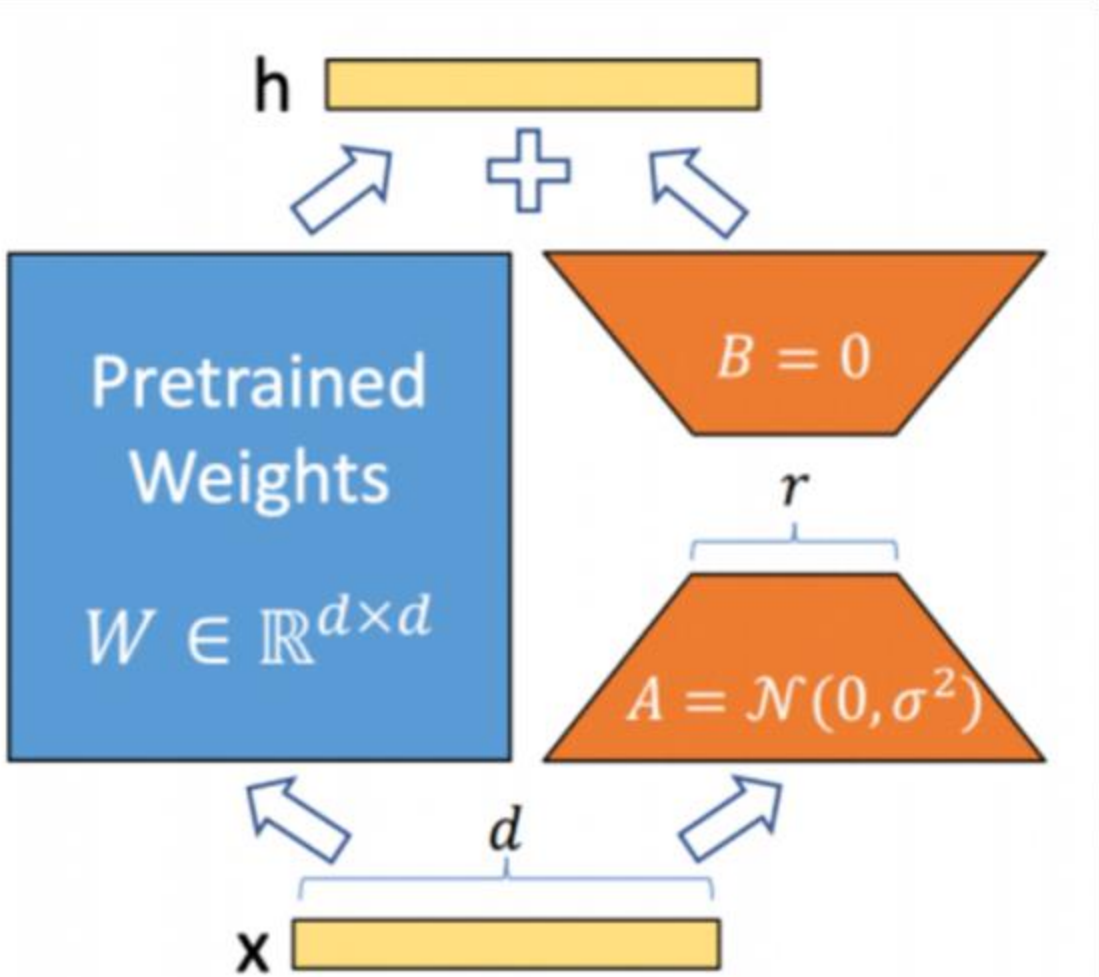
Normal Finetuning:

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

Update W

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

LoRA



Normal Finetuning:

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

Update W

LoRA Finetuning:

$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$

Update B,A
Leave W unchanged

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

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	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**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	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	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	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**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	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**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.

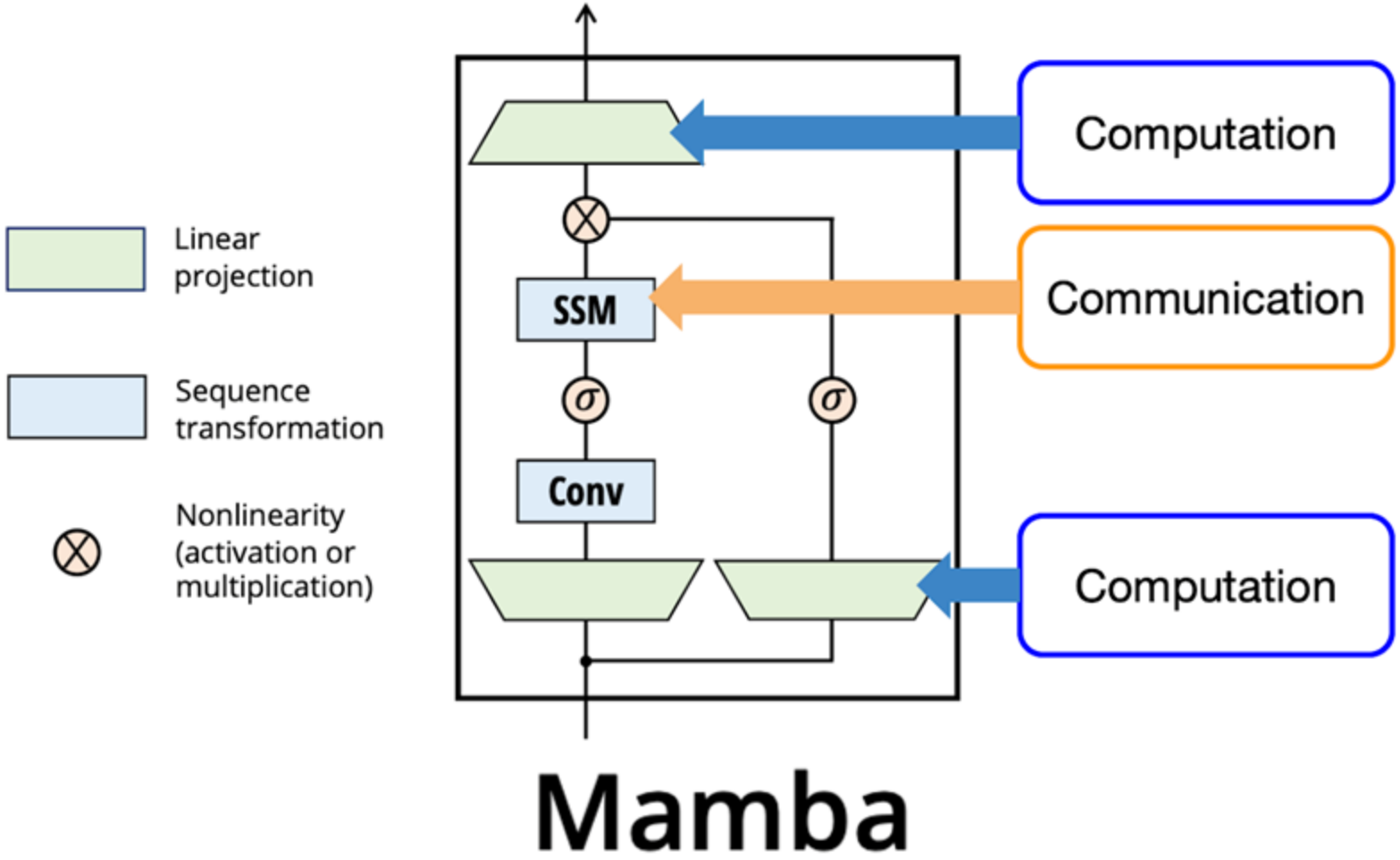
The Power of Scale for Parameter-Efficient Prompt Tuning [Lester, ACL 2021]

Efficient LLMs

- ❑ Quantization
- ❑ Sparsity
- ❑ Long Context
- ❑ Serving & Systems
- ❑ Distillation
- ❑ Parameter Efficient Fine-Tuning
- ❑ **Alternative Paradigms**

State Space Model (SSM)

Linearity Strikes back



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

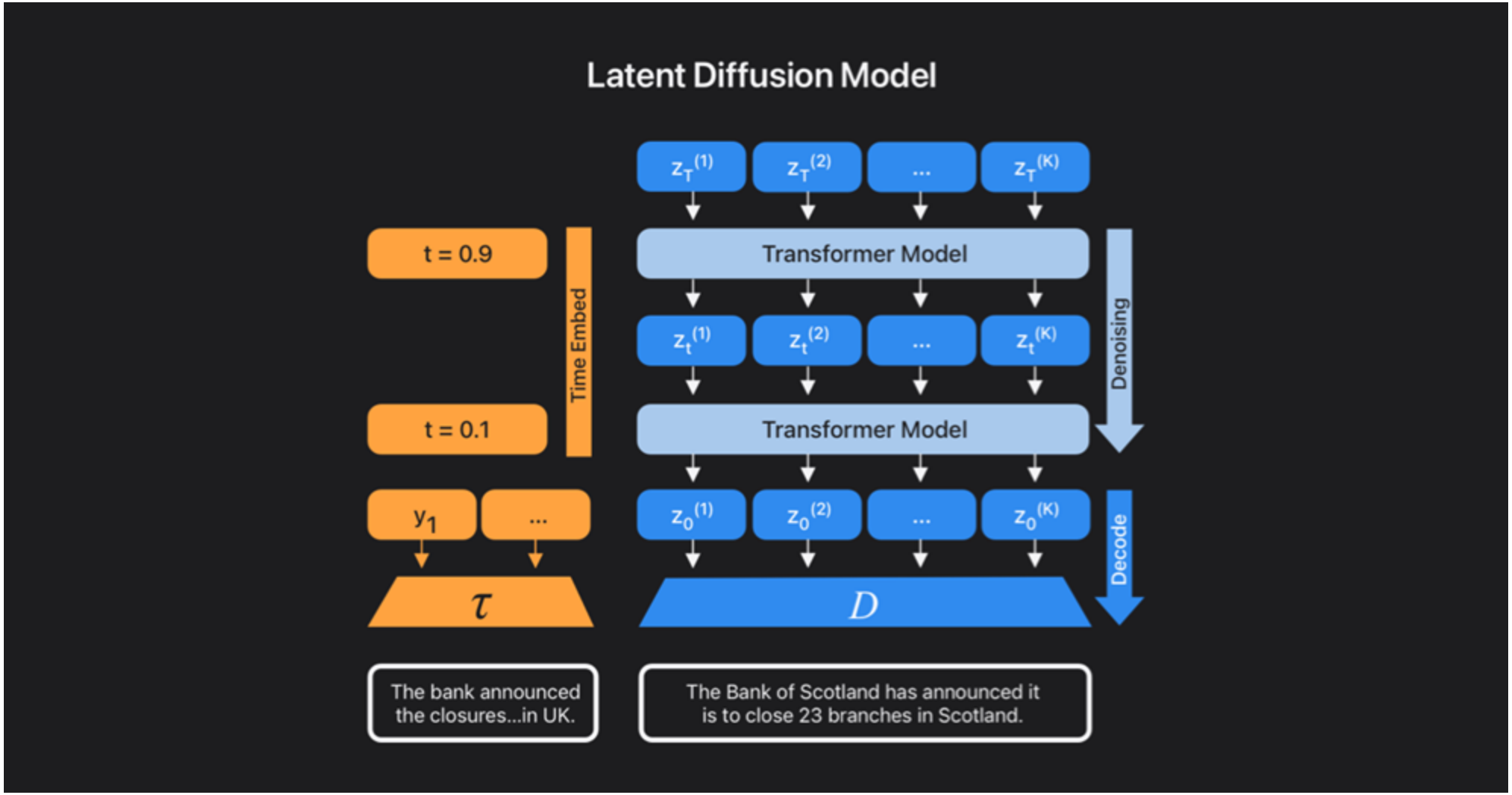
SSM

- 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

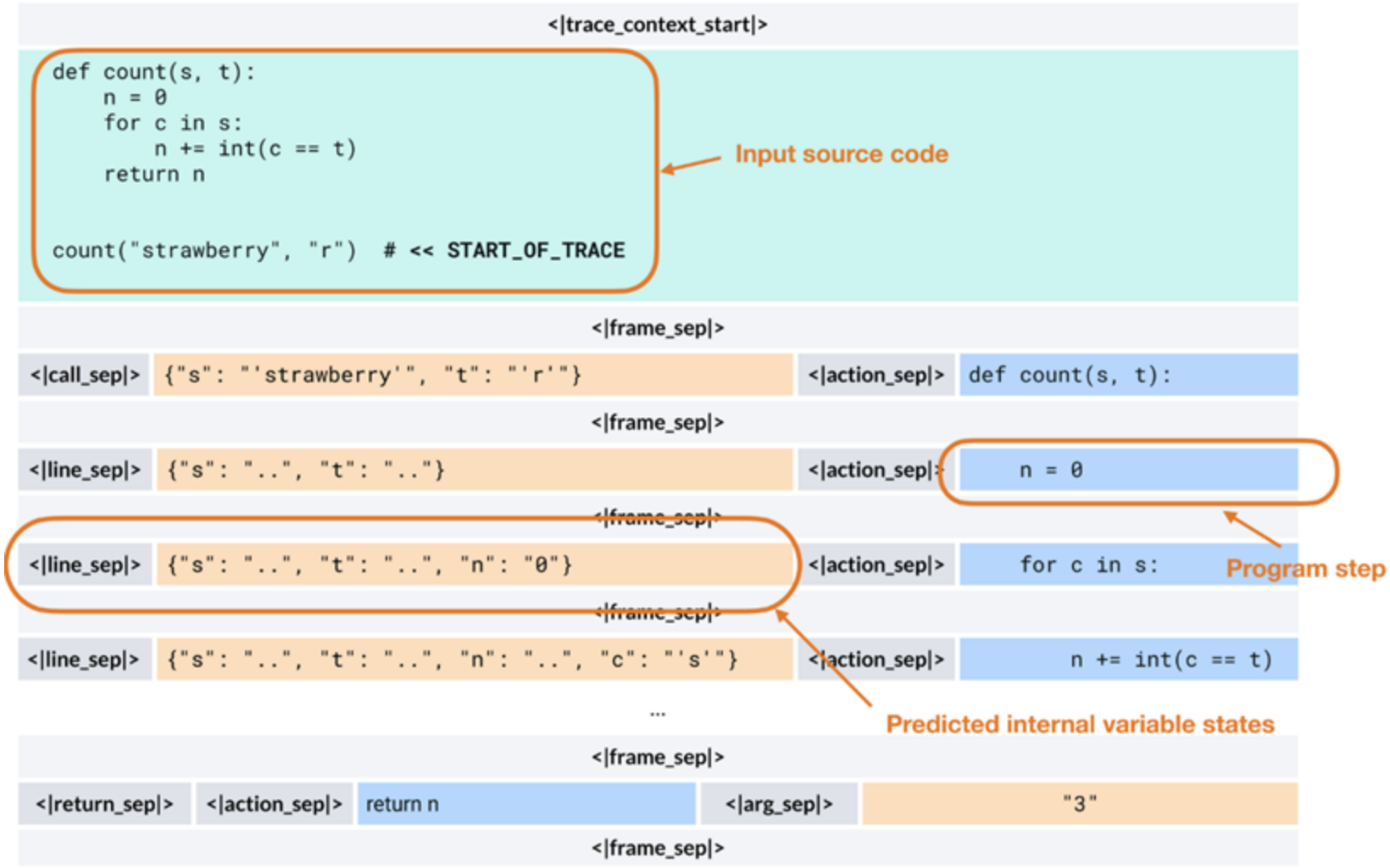
Text Diffusion

Predict multiple tokens as the same time using a diffusion model



World Models

Use some dense environment signals to predict what the consequences of each token prediction is



Recursive Transformers

- Use transformers recursively to iteratively refine model outputs
- This approach is still mostly useful in toy settings at the moment