

# “Small” LLMs

---

CMSC 473/673 - NATURAL LANGUAGE PROCESSING

# Learning Objectives

---

Examining...

- Methods for shrinking pre-existing models
- Methods for mimicking pre-existing models with smaller models
- Methods for faster/smaller finetuning of pre-existing models
- Methods for training new models more efficiently

Finding where to implement these methods

Recognizing when to implement them

# Efficient LLMs

---

Methods for shrinking pre-existing models

Methods for mimicking pre-existing models with smaller models

Methods for faster/smaller finetuning of pre-existing models

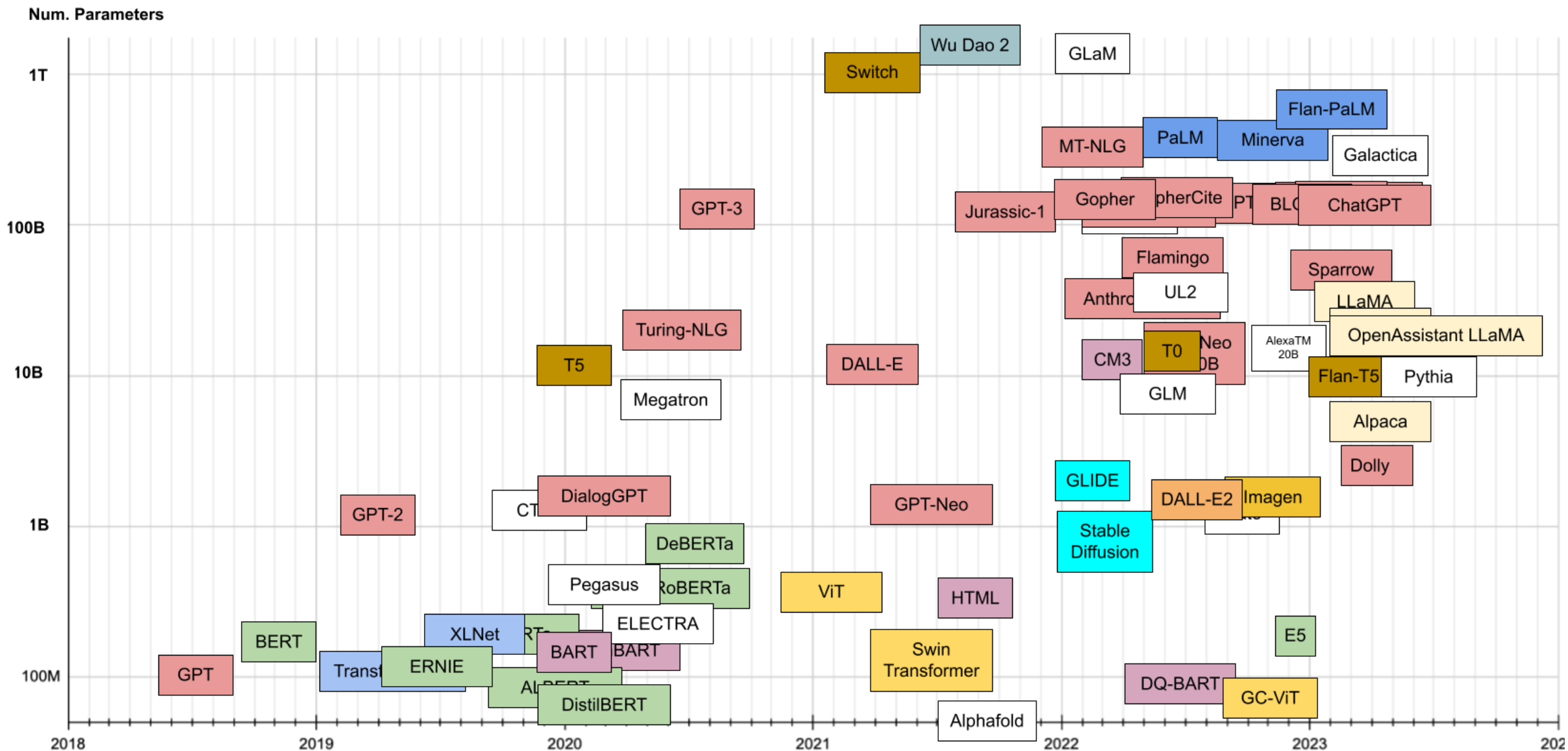
Methods for training new models more efficiently

Model Compression

Knowledge Distillation

PEFT

Efficient Training  
e.g., Sparse Mixture of Experts



<https://amatriain.net/blog/transformer-models-an-introduction-and-catalog-2d1e9039f376/>

Category	Benchmark (Metric)	Claude-3.5-Sonnet-1022	GPT-4o 0513	DeepSeek V3	OpenAI o1-mini	OpenAI o1-1217	DeepSeek R1
	Architecture	-	-	MoE	-	-	MoE
	# Activated Params	-	-	37B	-	-	37B
	# Total Params	-	~1.8 Trillion	671B	-	-	671B
English	MMLU (Pass@1)	88.3	87.2	88.5	85.2	<b>91.8</b>	90.8
	MMLU-Redux (EM)	88.9	88.0	89.1	86.7	-	<b>92.9</b>
	MMLU-Pro (EM)	78.0	72.6	75.9	80.3	-	<b>84.0</b>
	DROP (3-shot F1)	88.3	83.7	91.6	83.9	90.2	<b>92.2</b>
	IF-Eval (Prompt Strict)	<b>86.5</b>	84.3	86.1	84.8	-	83.3
	GPQA-Diamond (Pass@1)	65.0	49.9	59.1	60.0	<b>75.7</b>	71.5
	SimpleQA (Correct)	28.4	38.2	24.9	7.0	<b>47.0</b>	30.1
	FRAMES (Acc.)	72.5	80.5	73.3	76.9	-	<b>82.5</b>

<https://huggingface.co/deepseek-ai/DeepSeek-R1#4-evaluation-results>

## Nvidia's new Llama-3.1 Nemotron Ultra outperforms DeepSeek R1 at half the size

<https://venturebeat.com/ai/nvidias-new-llama-3-1-nemotron-ultra-outperforms-deepseek-r1-at-half-the-size/>

Carl Franzen

@carlfransen

April 8, 2025 8:08 AM

f X in

# Model Compression

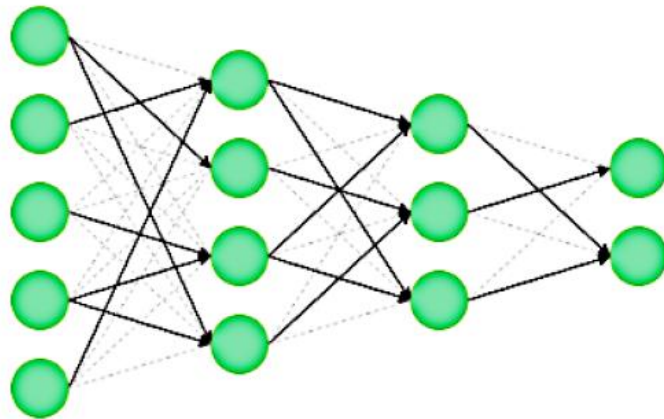
---

# Pruning

---

Remove parts of the model

Unstructured Pruning



Structured Pruning

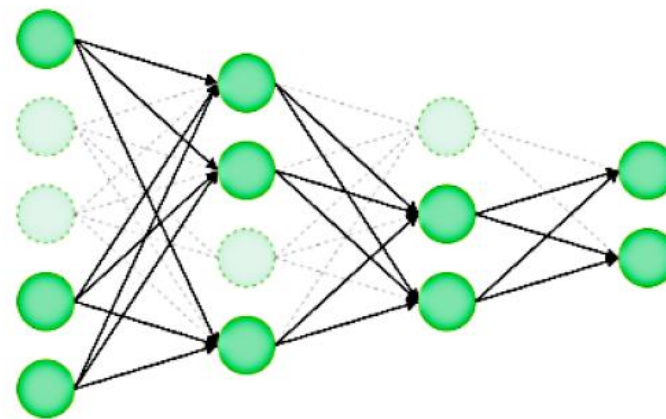
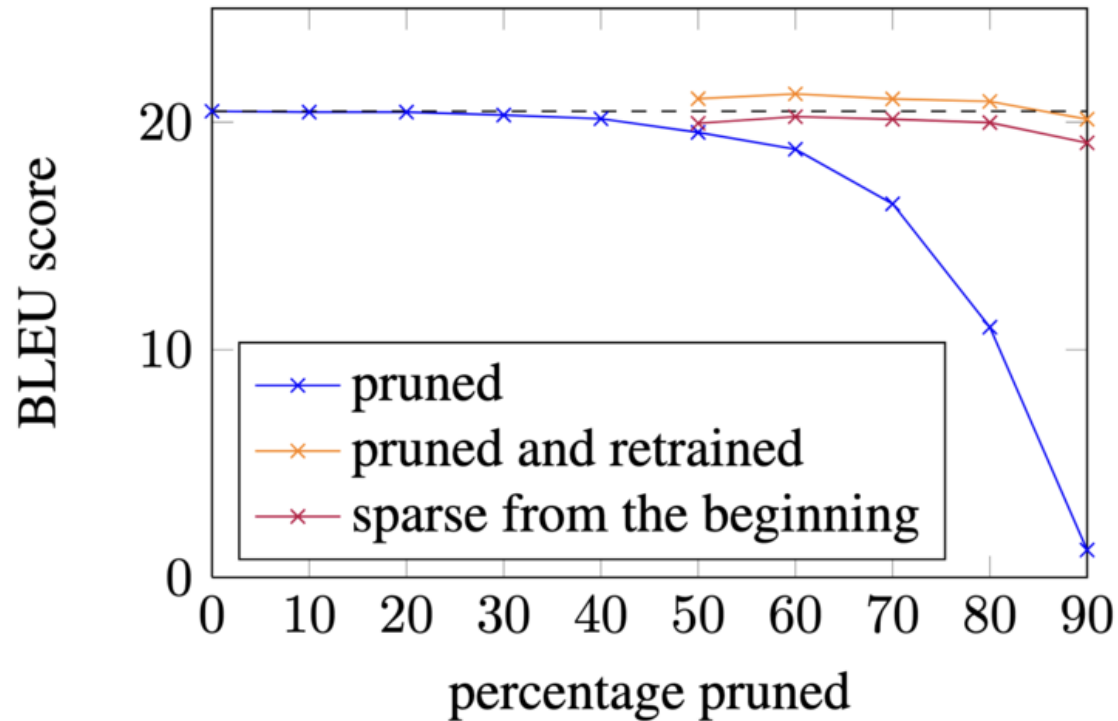


Image credits: neuralmagic.com

Implementation Tutorial: [https://pytorch.org/tutorials/intermediate/pruning\\_tutorial.html](https://pytorch.org/tutorials/intermediate/pruning_tutorial.html)

# Magnitude Pruning



- prune weights with smallest absolute value
- prunes 40% of the weights with negligible performance loss
- by adding a retraining phase after pruning, we can prune 80% with no performance loss

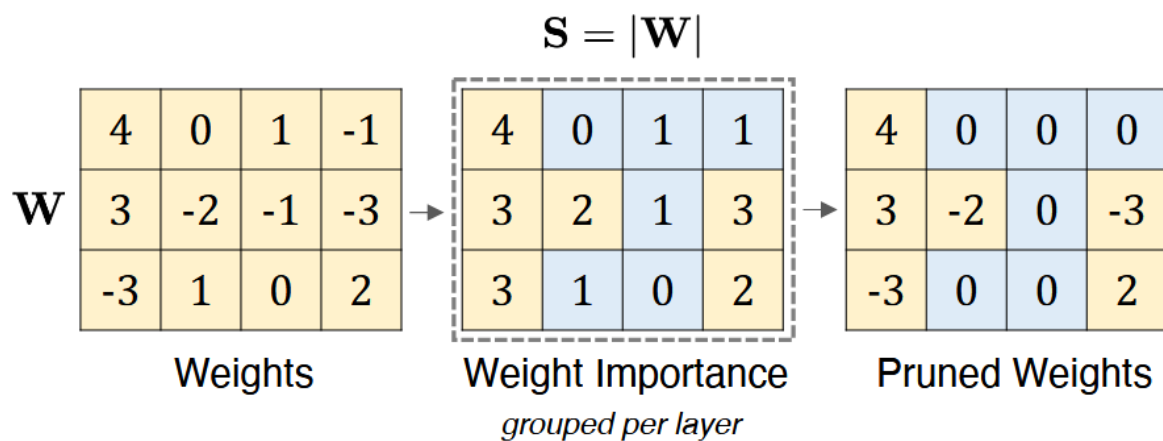
Image credits: [See et al. 2016](#)

Slide by Dinesh Raghu

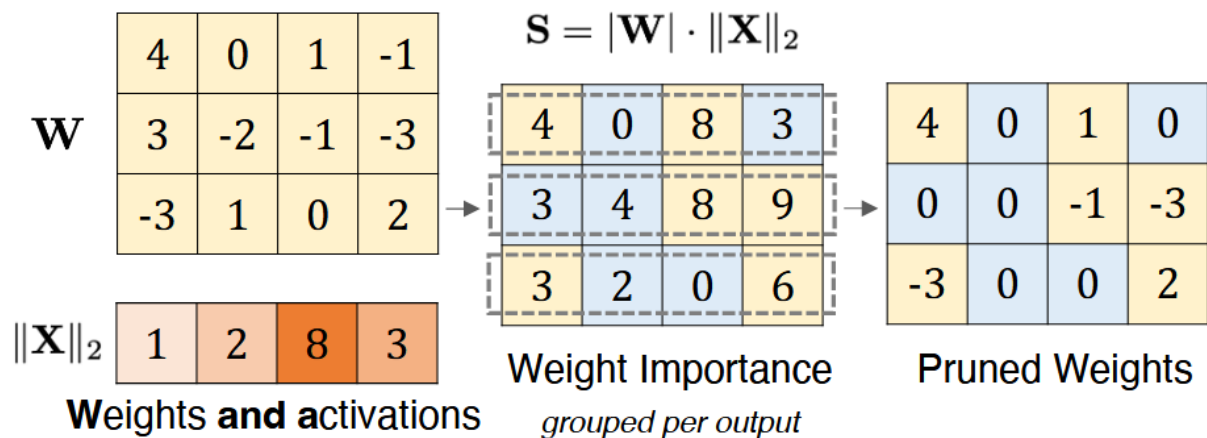


# Wanda

## Magnitude Pruning



## Wanda



# Quantizing Models

---

Compresses weights and activations from floating point numbers to integers (e.g., 4-bit, 8-bit)

Then use the scaling factor to get the “original” value

Implementation:

<https://pytorch.org/docs/stable/quantization.html>

<https://pypi.org/project/bitsandbytes/>

Learn more here:

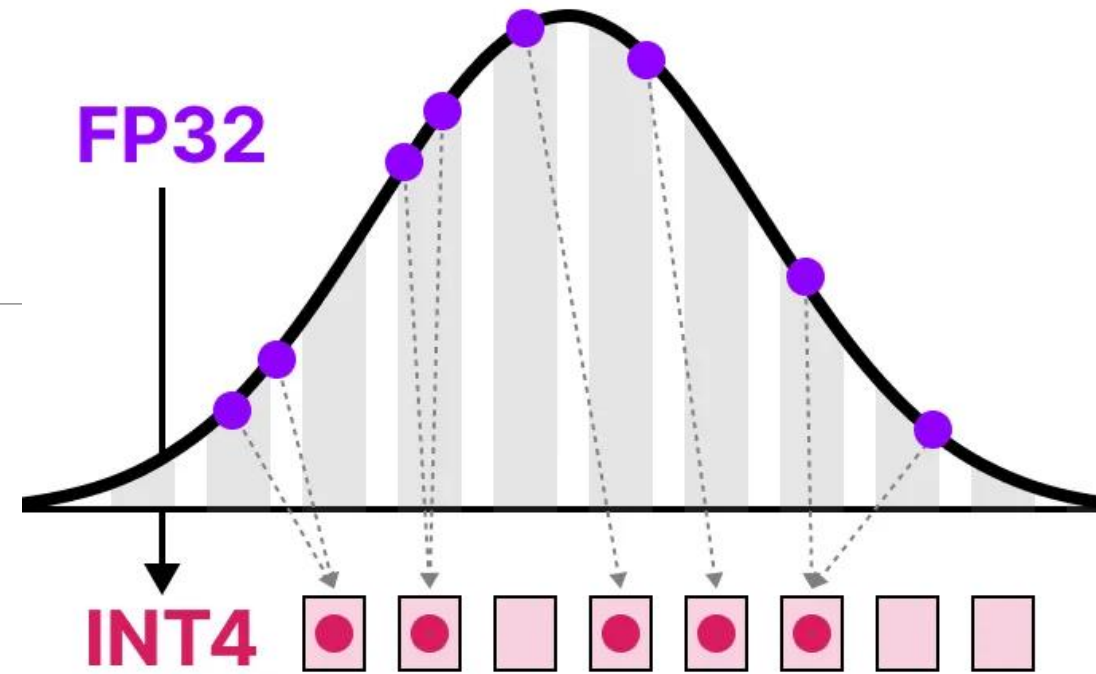
<https://huggingface.co/blog/hf-bitsandbytes-integration>

# Quantizing Models

Given a layer  $l$  with weight matrix  $W_l$  and layer input  $X_l$ , find quantized weight  $\hat{W}_l$ :

$$(\hat{W}_l^* = \operatorname{argmin}_{\hat{W}_l} \|W_l X - \hat{W}_l X\|_2^2)$$

<https://huggingface.co/blog/merve/quantization>

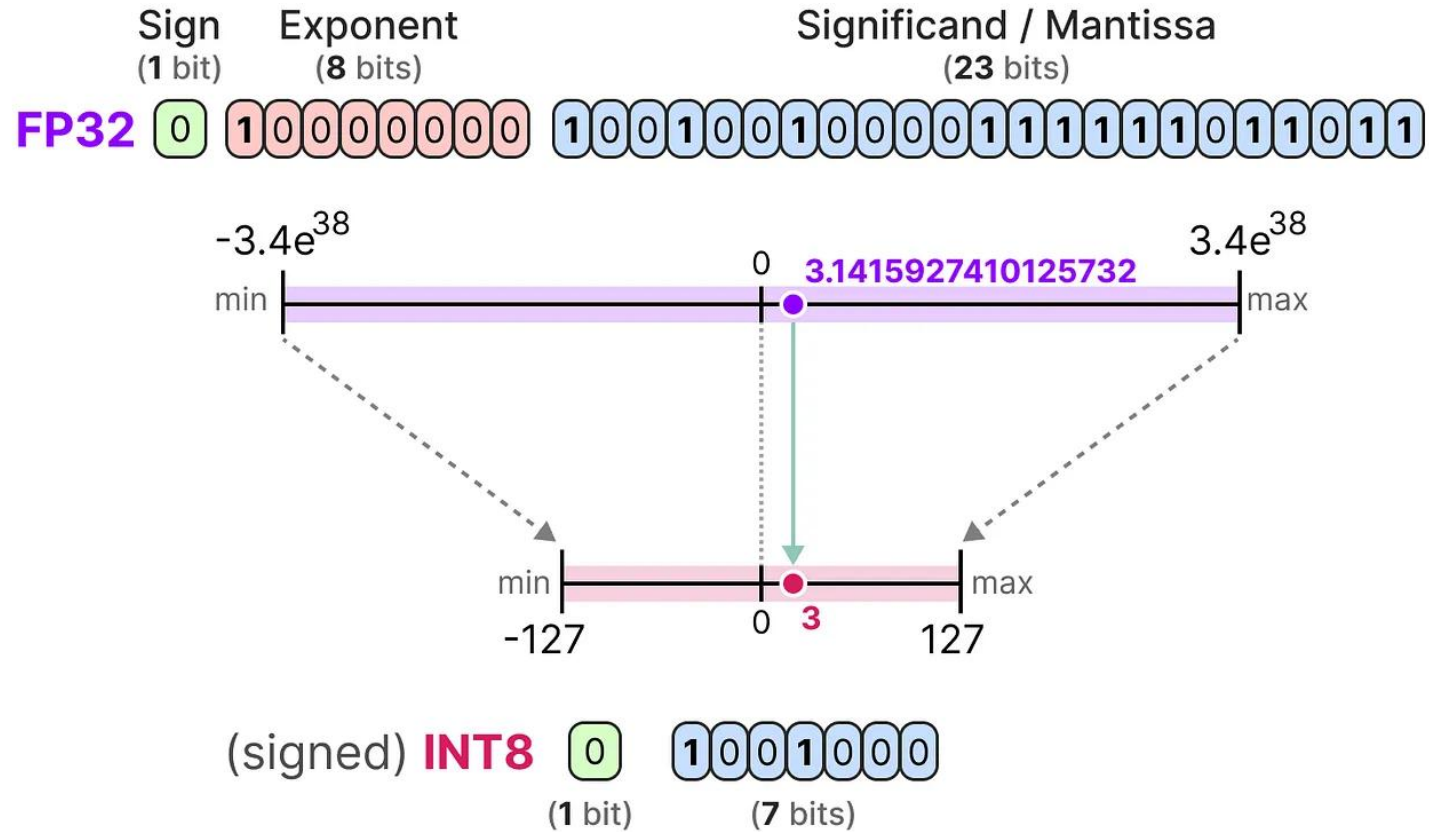


<https://substack.com/home/post/p-145531349>

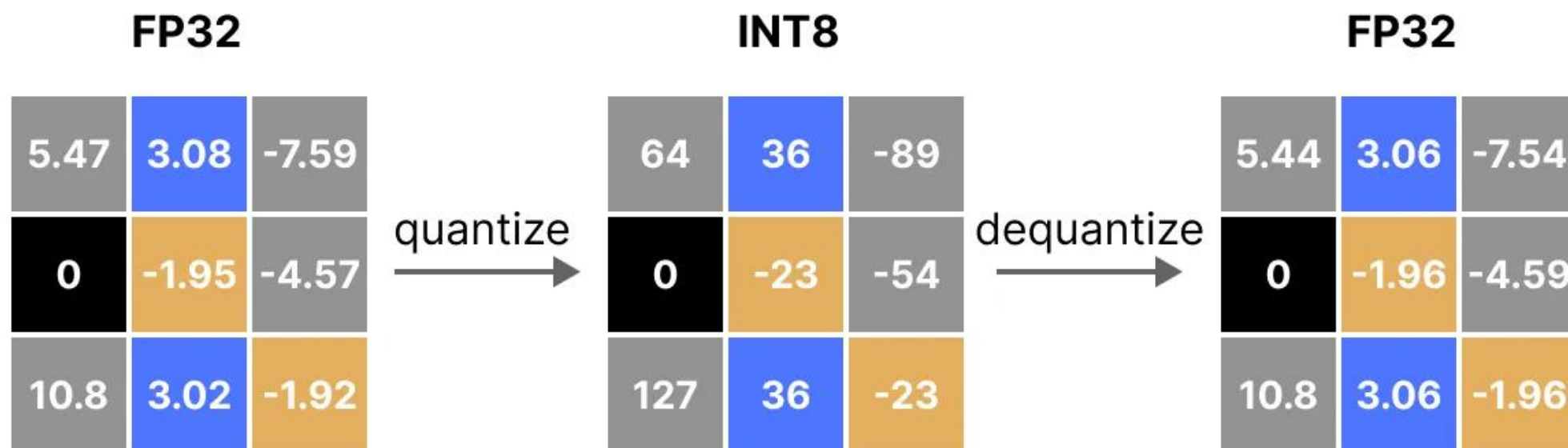
$$\mathbf{X}^{\text{Int8}} = \operatorname{round} \left( \frac{127}{\operatorname{absmax}(\mathbf{X}^{\text{FP32}})} \mathbf{X}^{\text{FP32}} \right) = \operatorname{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}), \text{ where } c \text{ is the quantization constant}$$

Dettmers, Tim, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. "QLoRA: Efficient Finetuning of Quantized LLMs." *Advances in Neural Information Processing Systems*

# Mapping floating point to integer



# Quantizing → Dequantizing



# Quantizing/Pruning (Model Compression)

---

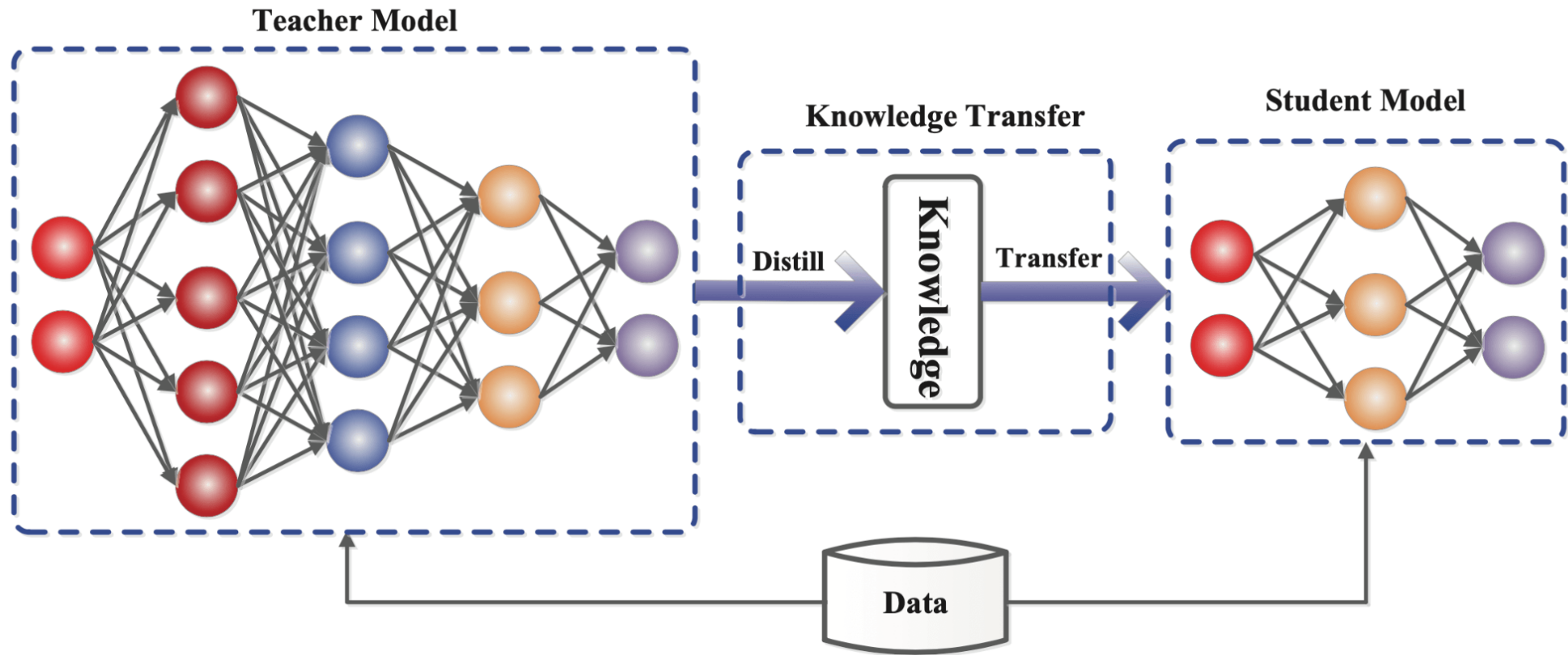
PROS

CONS

# Knowledge Distillation

---

# Knowledge Distillation

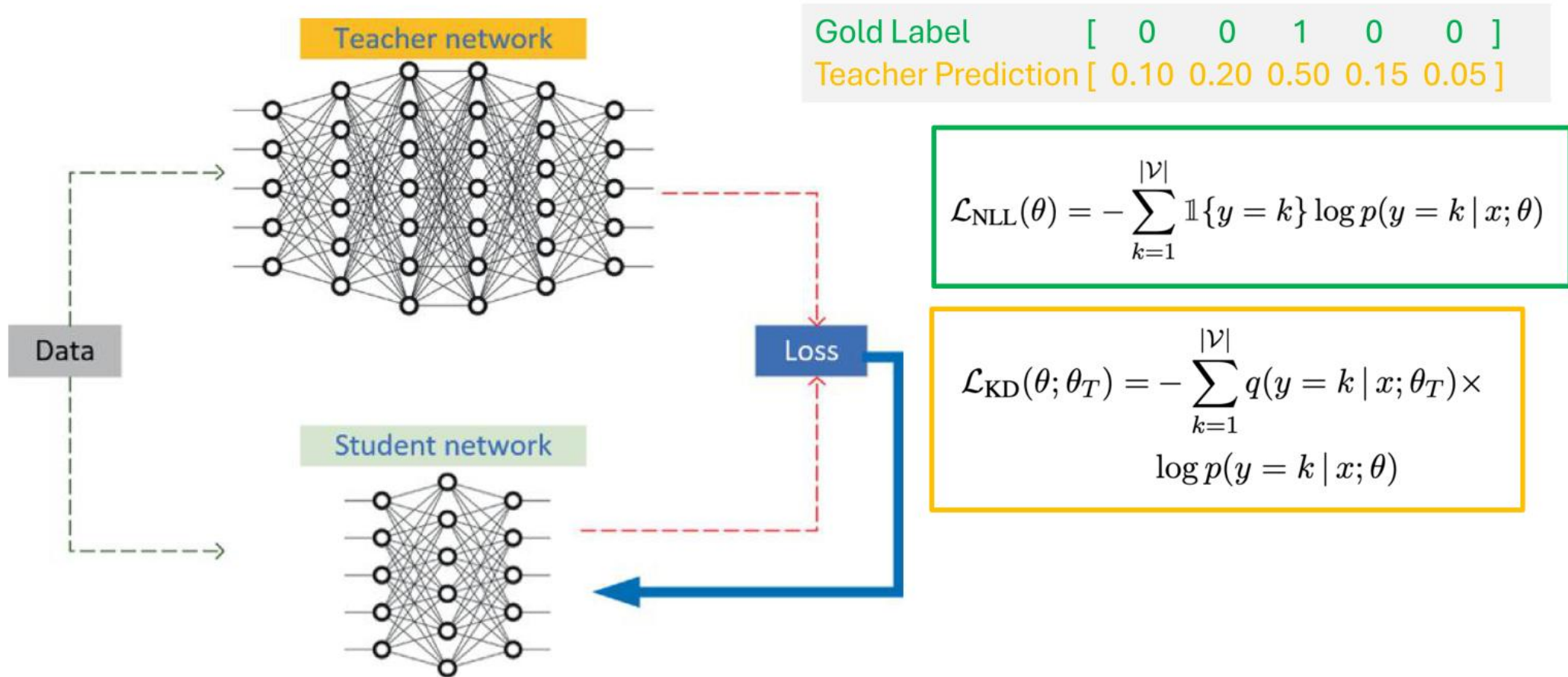


<https://neptune.ai/blog/knowledge-distillation>

Implementation Tutorial: [https://pytorch.org/tutorials/beginner/knowledge\\_distillation\\_tutorial.html](https://pytorch.org/tutorials/beginner/knowledge_distillation_tutorial.html)



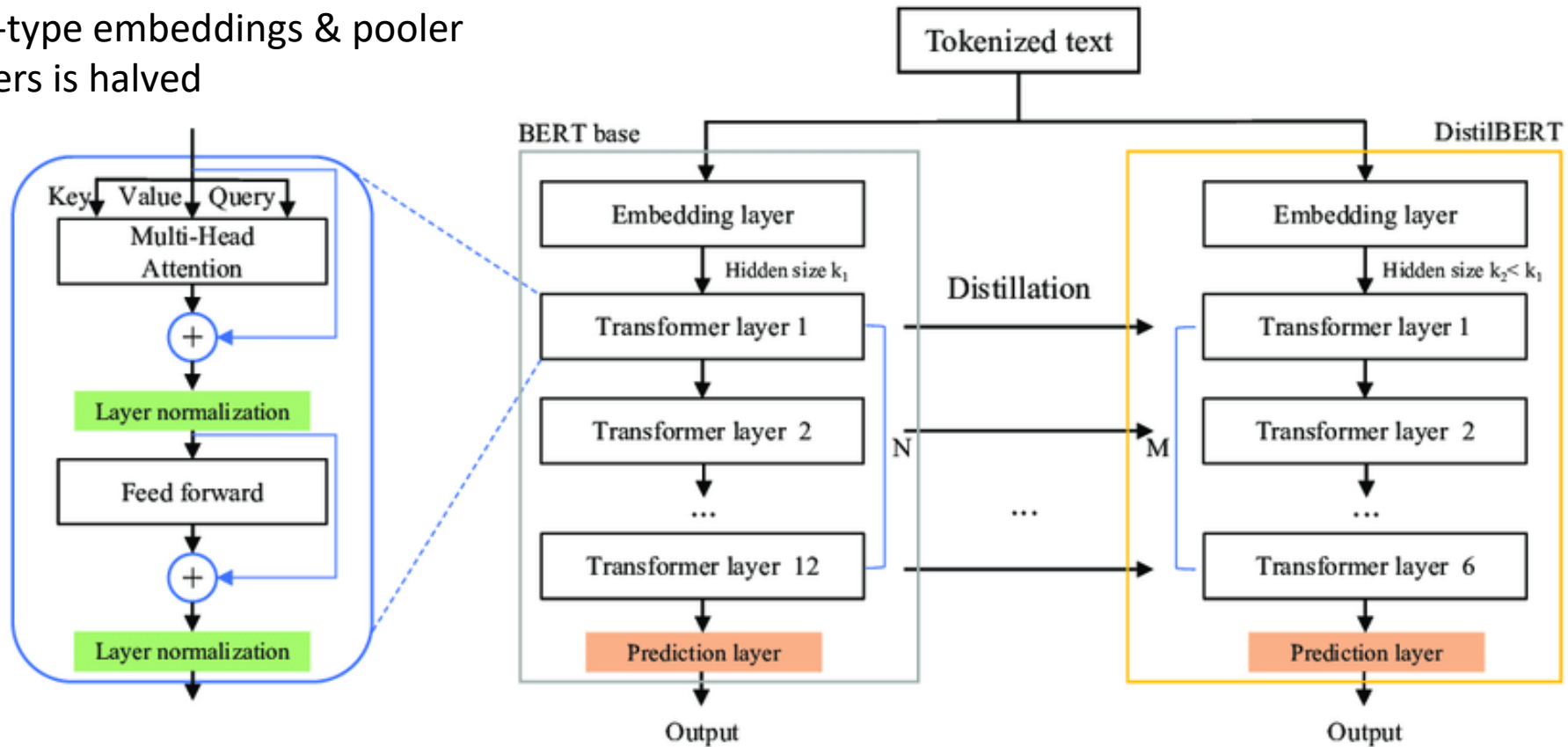
# Training the Student Network



Slide by Dinesh Raghu

# DistilBERT

Removed token-type embeddings & pooler  
Number of layers is halved



[https://www.researchgate.net/figure/The-DistilBERT-model-architecture-and-components\\_fig2\\_358239462](https://www.researchgate.net/figure/The-DistilBERT-model-architecture-and-components_fig2_358239462)

# Knowledge Distillation

---

PROS

CONS

# PEFT

---

# Parameter-efficient Fine-tuning (PEFT)

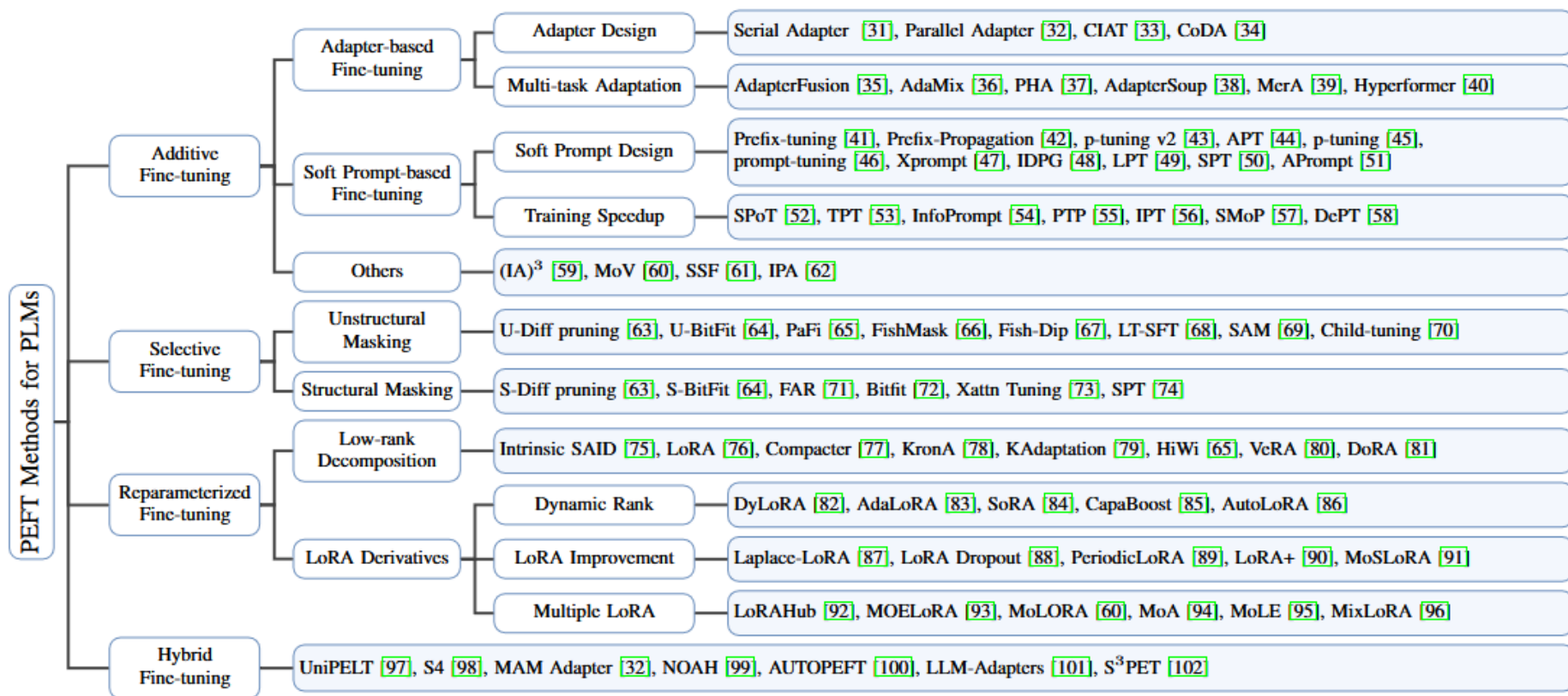
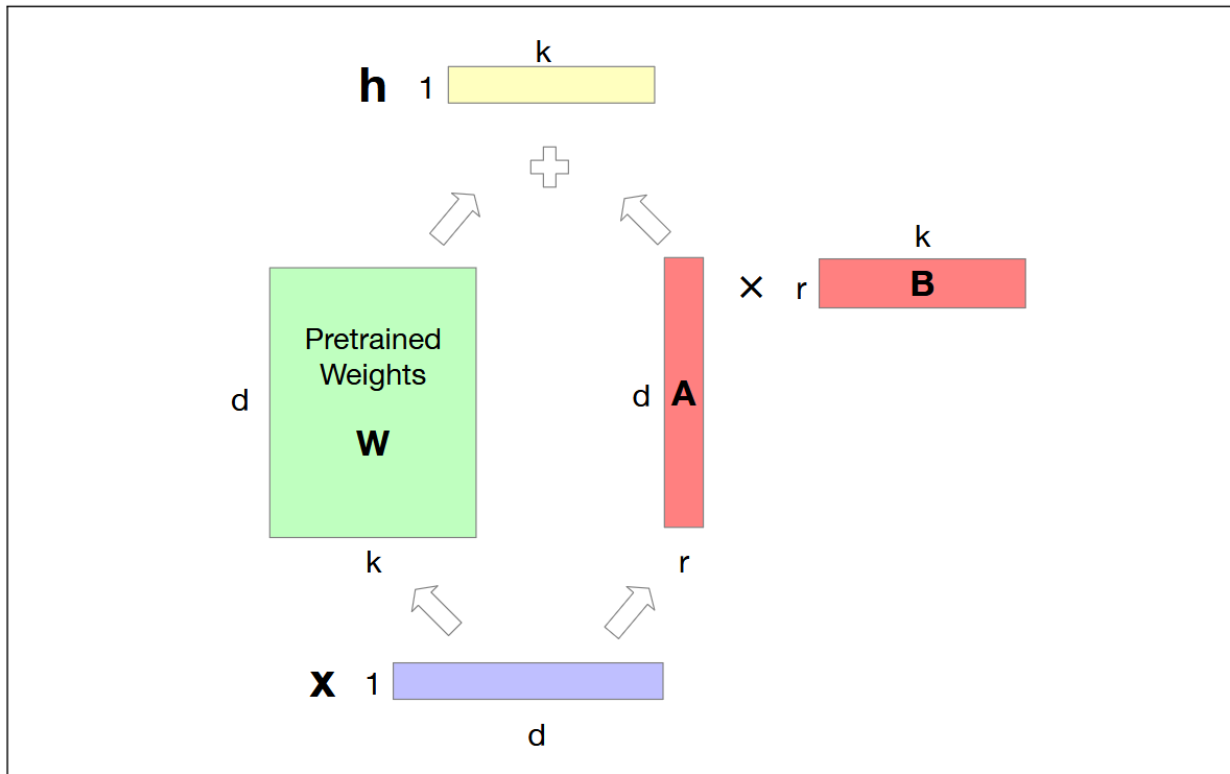


Fig. 3: Taxonomy of Parameter-Efficient Fine-Tuning Methods for Large Models.

# LoRA (Low-Rank Adaptation)



**Figure 10.8** The intuition of LoRA. We freeze  $W$  to its pretrained values, and instead fine-tune by training a pair of matrices  $A$  and  $B$ , updating those instead of  $W$ , and just sum  $W$  and the updated  $AB$ .

Train a model using a pretrained LLM but give the new model fewer parameters  $\rightarrow$  a low-rank decomposition of the original weight matrix

$$h = xW + xAB$$

$$r \ll \min(d, N)$$

From SLP book Chapter 10

# LoRA (Low-Rank Adaptation)

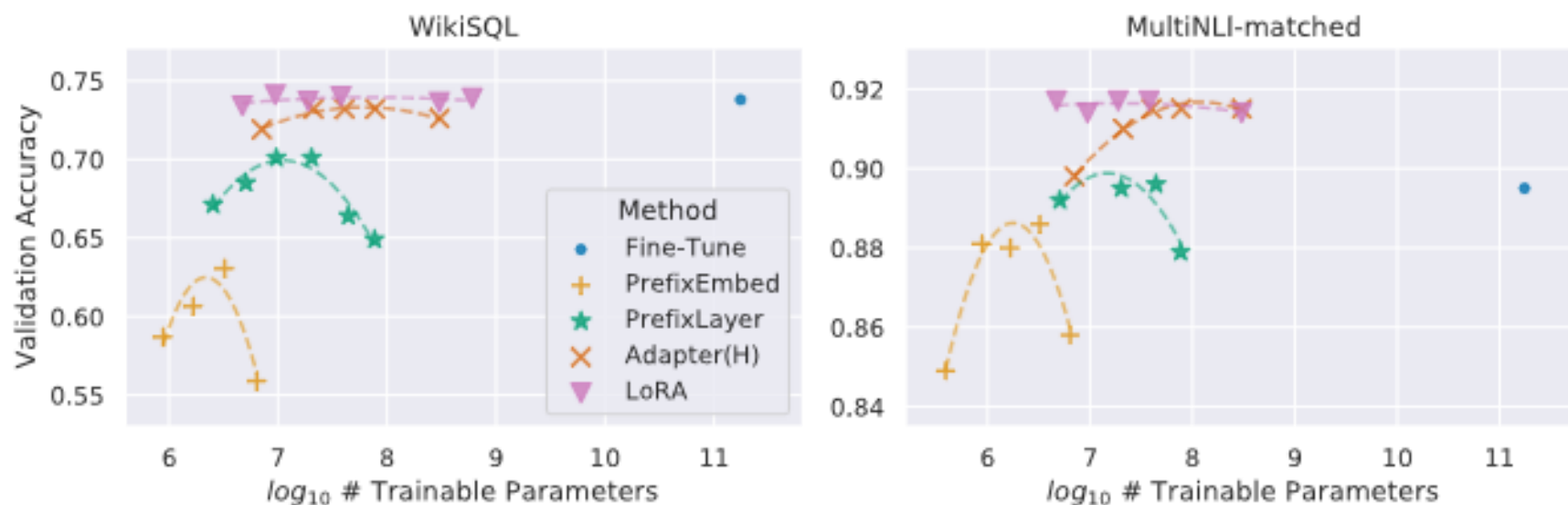
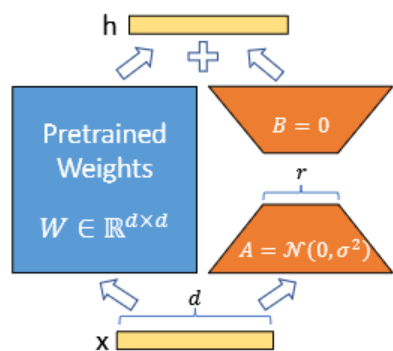


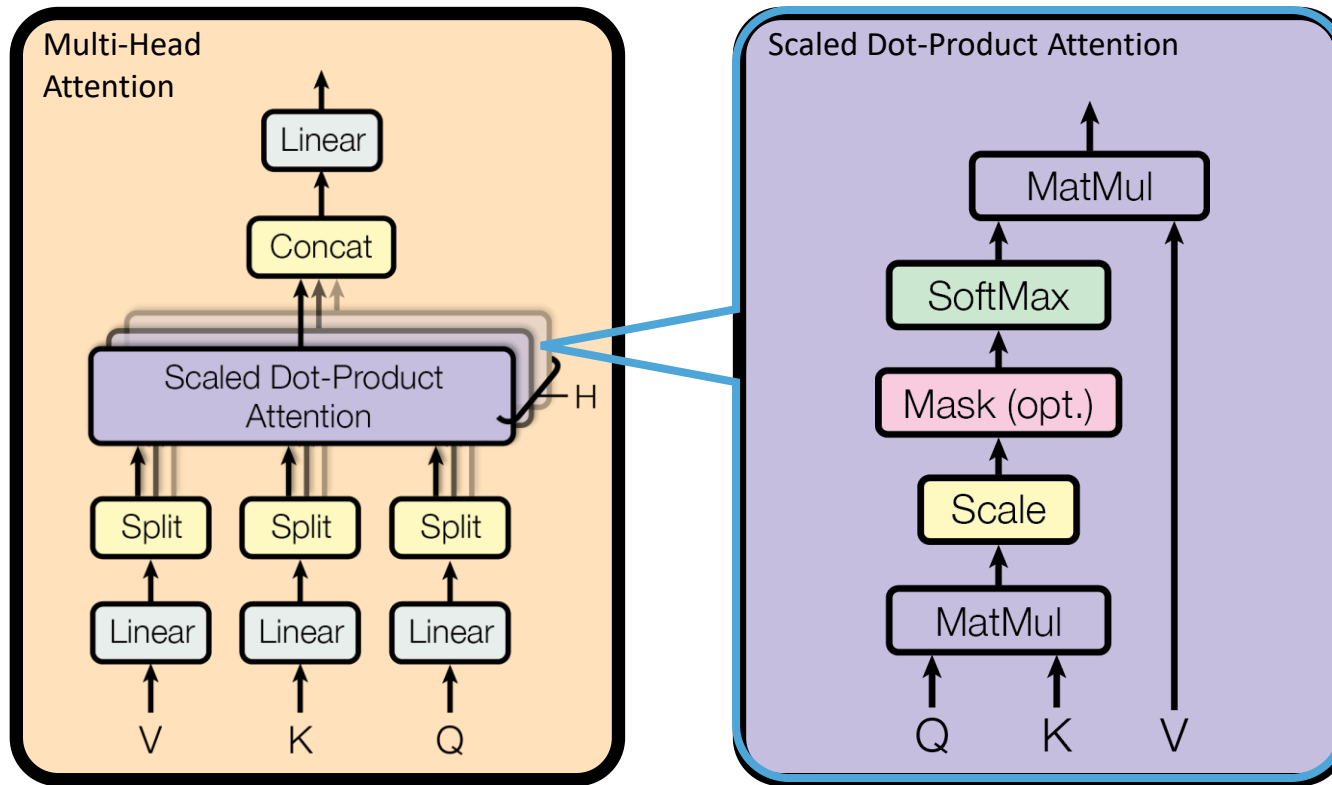
Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See [Section I.2](#) for more details on the plotted data points.

Implementation:

<https://github.com/microsoft/LoRA>

<https://huggingface.co/docs/diffusers/training/lora>

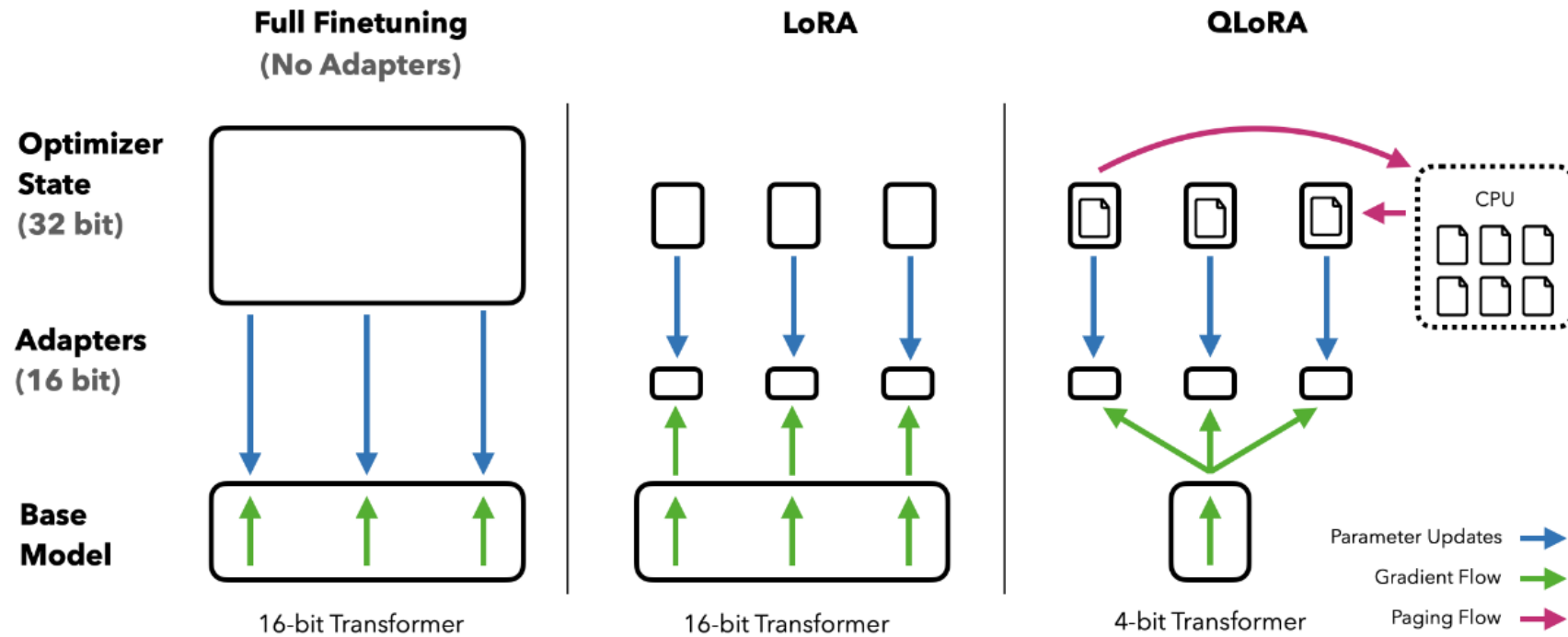
# Review: Attention Mechanism



Original LoRA was just applied to the attention weights:  
 $W_Q$ ,  $W_K$ ,  $W_V$ , and  $W_O$



# Guanaco: QLoRA



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

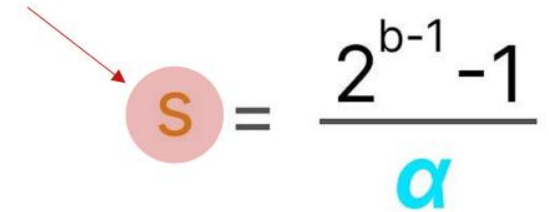
# Guanaco: QLoRA

---

4-bit NormalFloat quantization

Uses *double quantization* (quantizing the quantization constants)

Stored as FP32


$$s = \frac{2^{b-1} - 1}{\alpha}$$

Dinesh Raghu

# Guanaco: QLoRA

**Table 3:** Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset Model	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

**Table 8:** Evaluation of biases on the CrowS dataset. A lower score indicates lower likelihood of generating biased sequences. Guanaco follows the biased pattern of the LLaMA base model.

	LLaMA-65B	GPT-3	OPT-175B	Guanaco-65B
Gender	70.6	62.6	65.7	<b>47.5</b>
Religion	79.0	73.3	68.6	<b>38.7</b>
Race/Color	57.0	64.7	68.6	<b>45.3</b>
Sexual orientation	81.0	76.2	78.6	<b>59.1</b>
Age	70.1	64.4	67.8	<b>36.3</b>
Nationality	64.2	61.6	62.9	<b>32.4</b>
Disability	66.7	76.7	76.7	<b>33.9</b>
Physical appearance	77.8	74.6	76.2	<b>43.1</b>
Socioeconomic status	71.5	73.8	76.2	<b>55.3</b>
Average	66.6	67.2	69.5	<b>43.5</b>

# LoRA

---

PROS

CONS

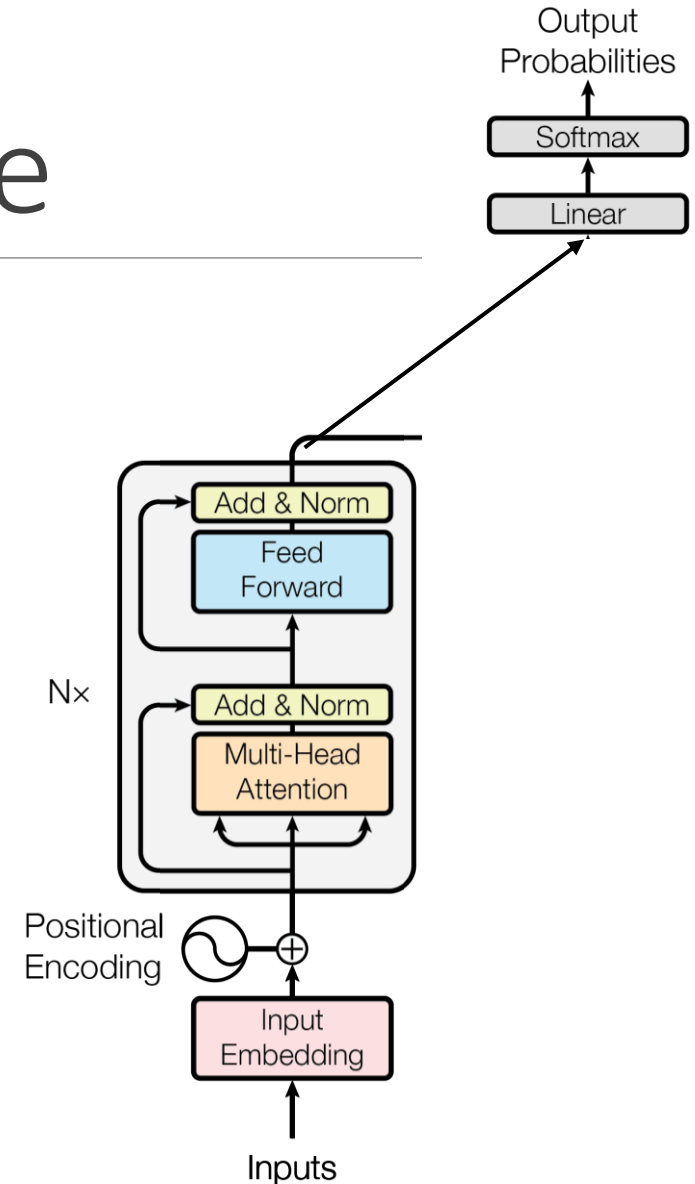
# Efficient Training

---

# Review: Transformer Architecture

$$\begin{aligned}
 \text{Multi-Head Attention} &= \text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}) \\
 \text{Add \& Norm} &= \text{LayerNorm}(\text{Multi-Head Attention} + \mathbf{H}_i^{enc}) \\
 \text{Feed Forward} &= \max(0, \text{Add \& Norm} \mathbf{W}_1 + b_1) \mathbf{W}_2 + b_2 \\
 \text{Add \& Norm (2)} &= \text{LayerNorm}(\text{Feed Forward} + \mathbf{H}_i^{enc}) \\
 \mathbf{H}_{i+1}^{enc} &= \text{Add \& Norm (2)}
 \end{aligned}$$

A decoder-only architecture is very similar to the encoder of the original transformer architecture



# Sparse Mixture-of-Experts

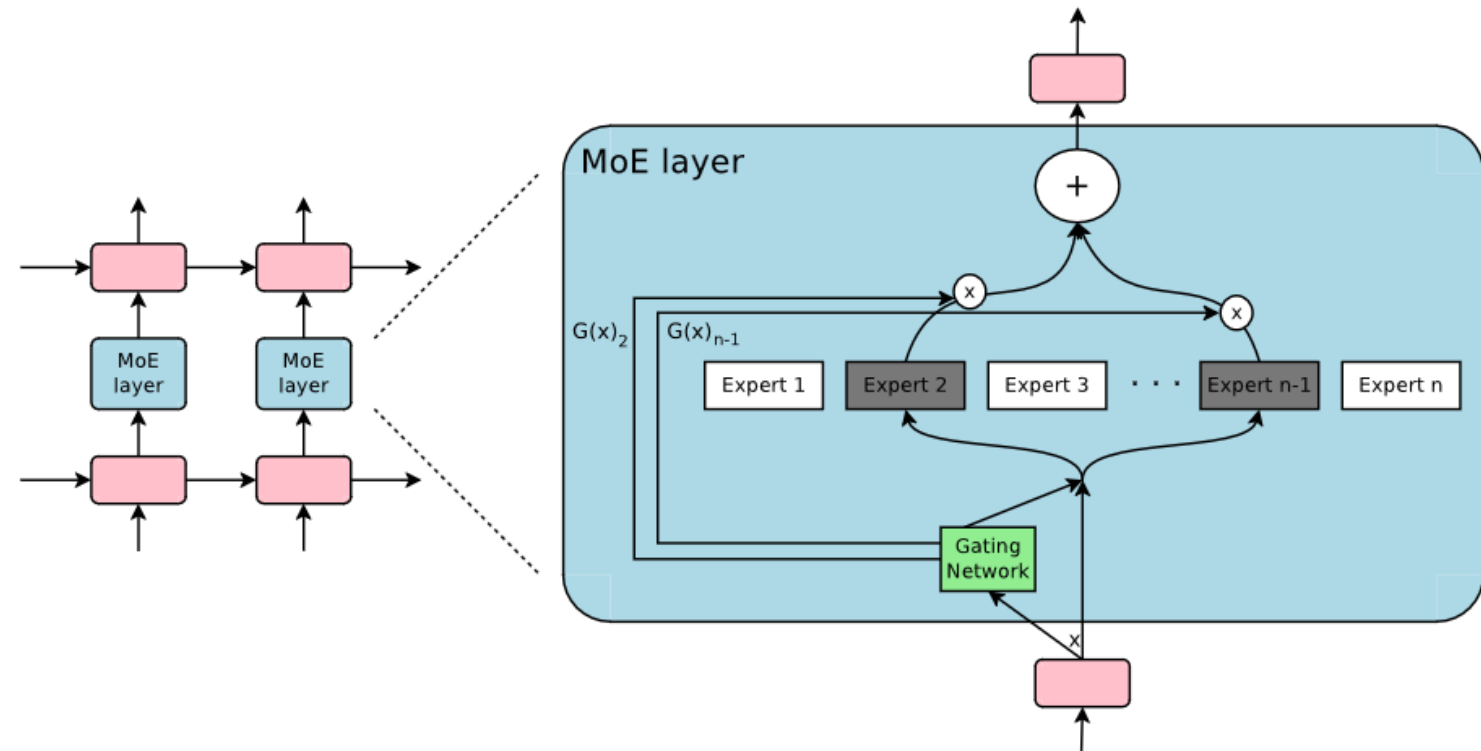


Figure 1: A Mixture of Experts (MoE) layer embedded within a recurrent language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

## Implementations

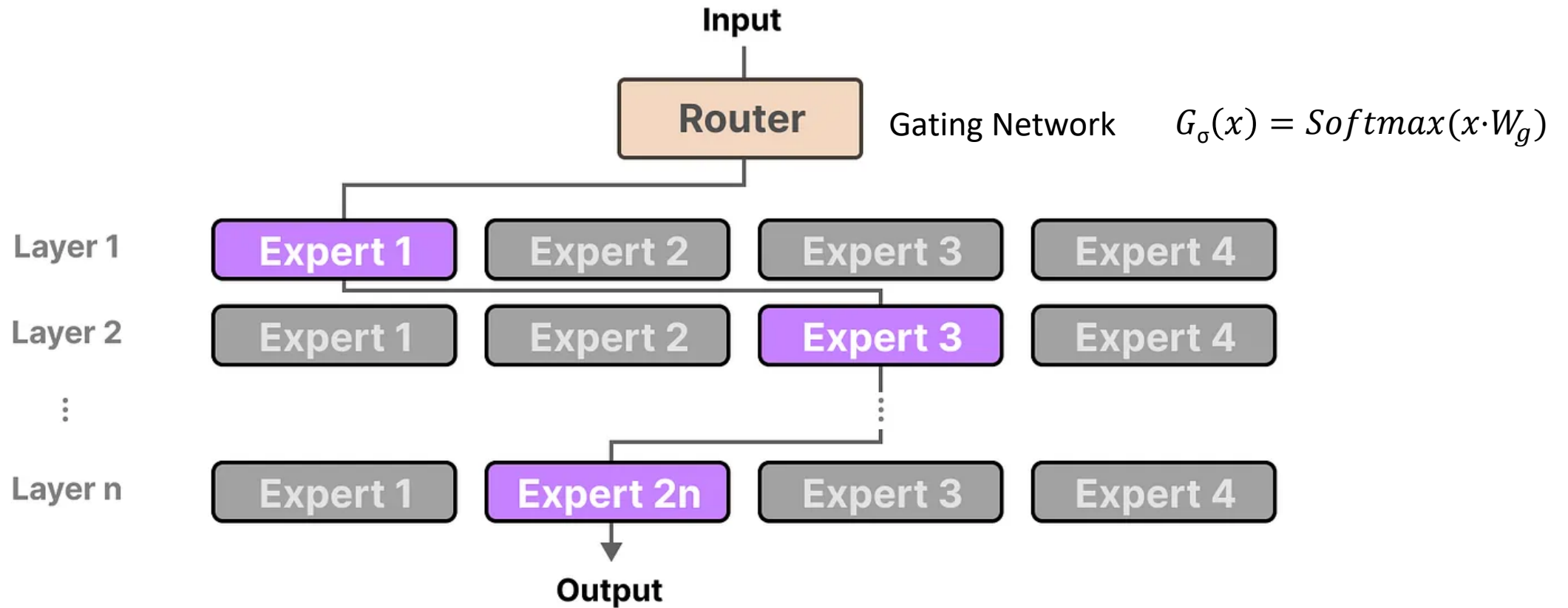
Megablocks: <https://github.com/stanford-futuredata/megablocks>

Fairseq: [https://github.com/facebookresearch/fairseq/tree/main/examples/moe\\_lm](https://github.com/facebookresearch/fairseq/tree/main/examples/moe_lm)

OpenMoE: <https://github.com/XueFuzhao/OpenMoE>

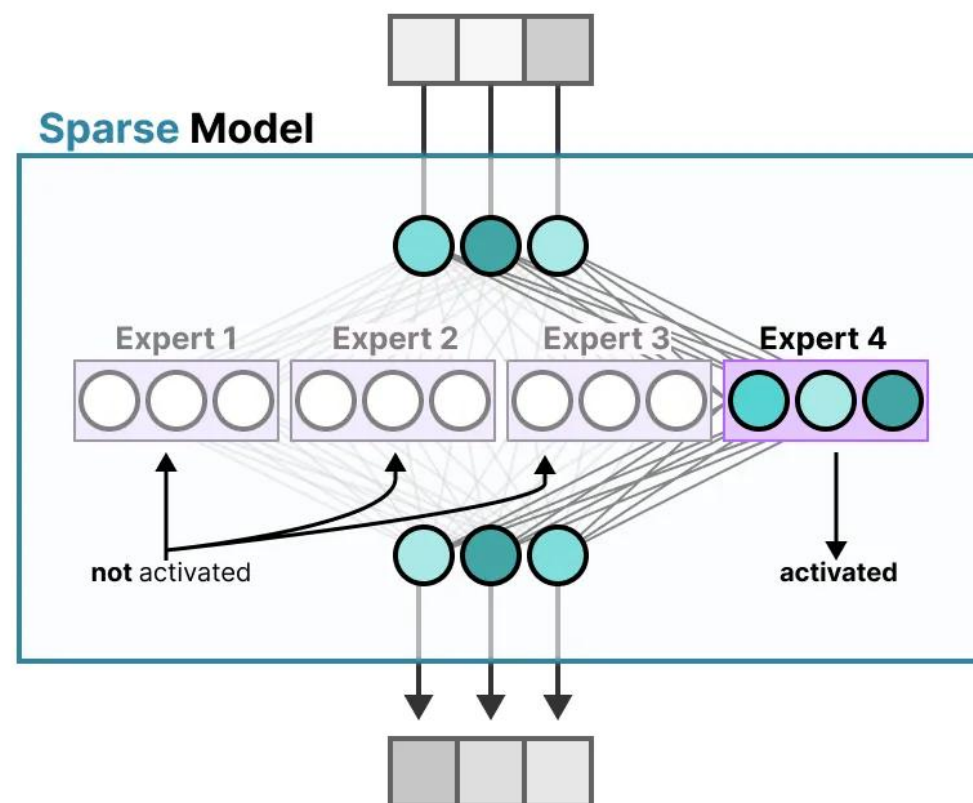
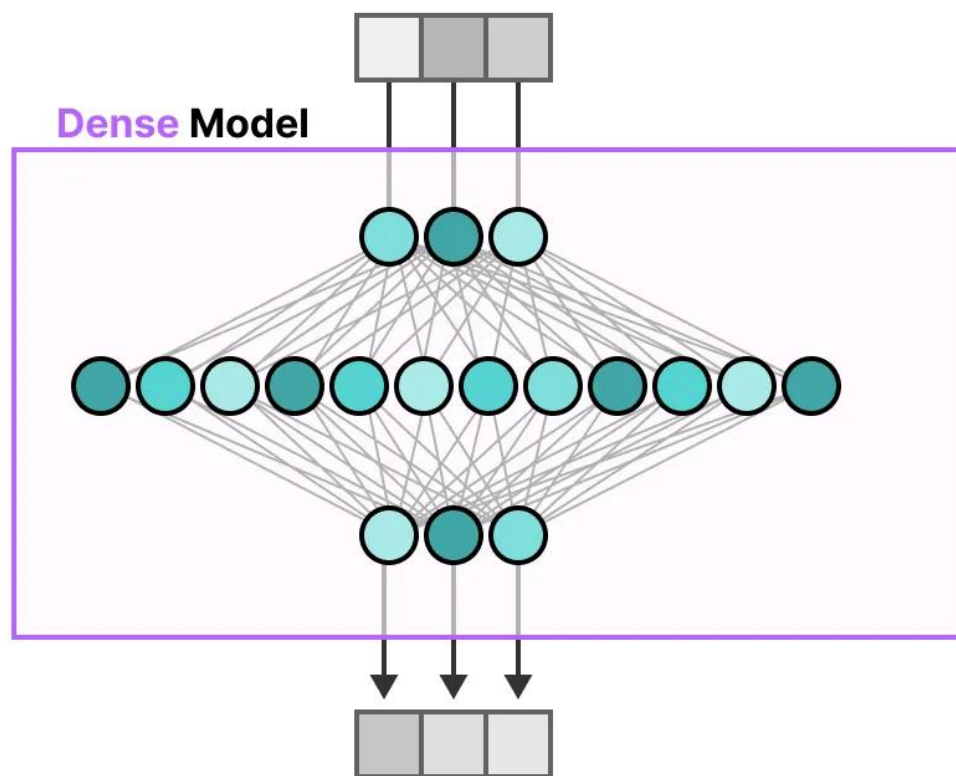
# Mixture-of-Experts

$$y = \sum_{i=1}^n G(x)_i E_i(x)$$

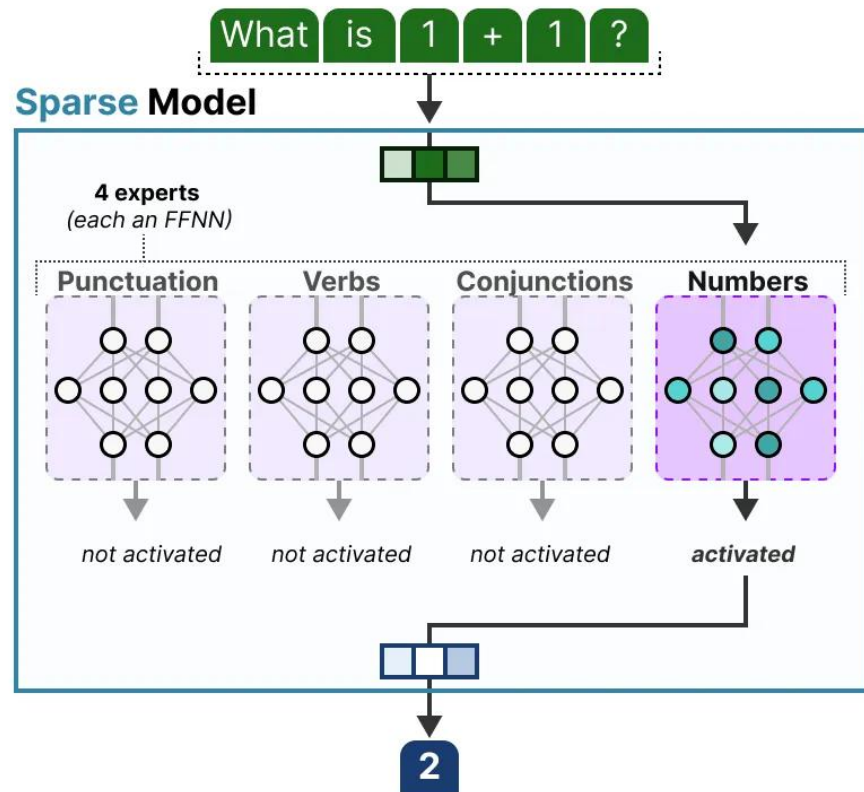




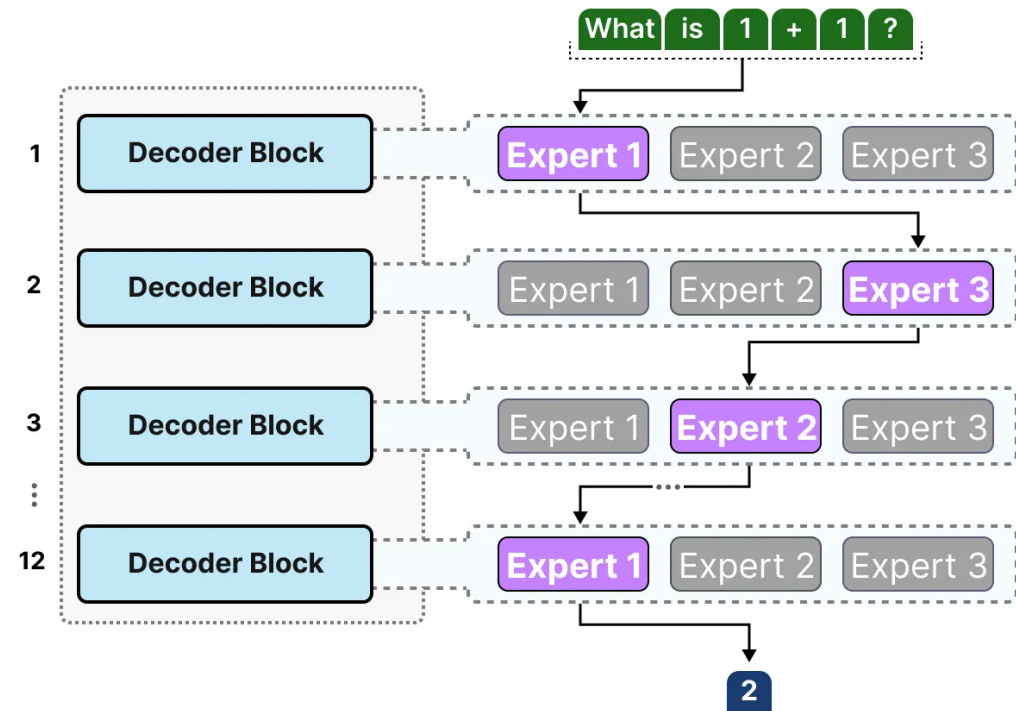
# Sparse MoE



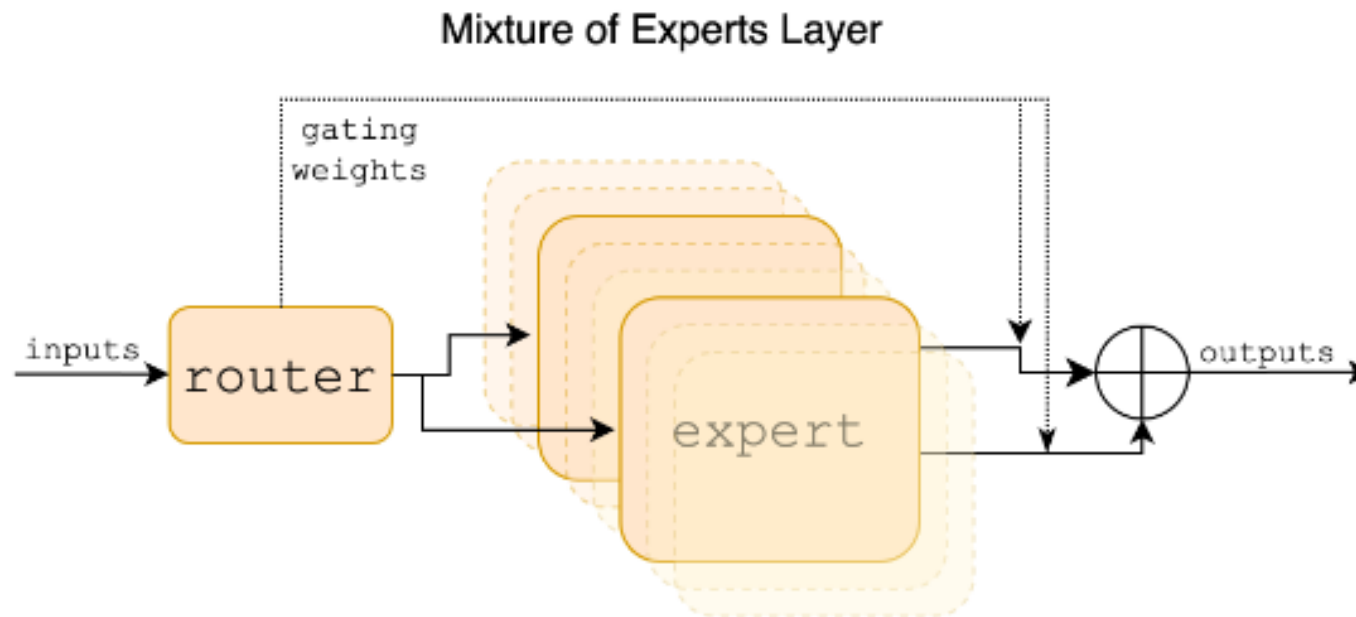
# MoE Example



Most transformers have multiple decoder blocks

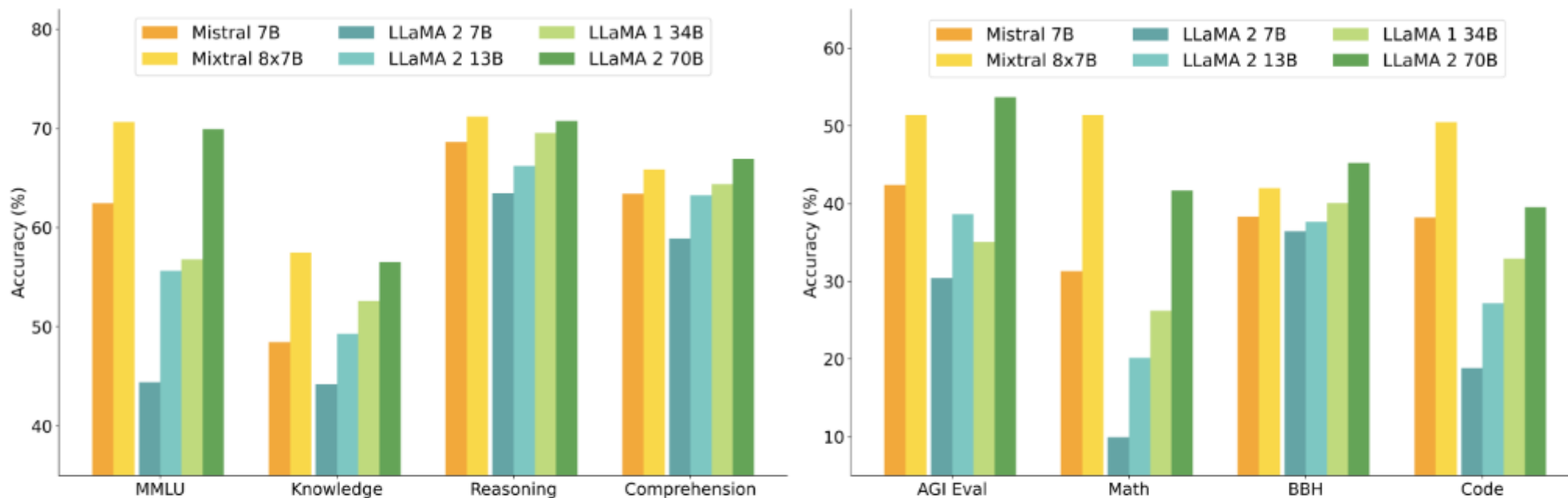


# Mixtral 8x7B



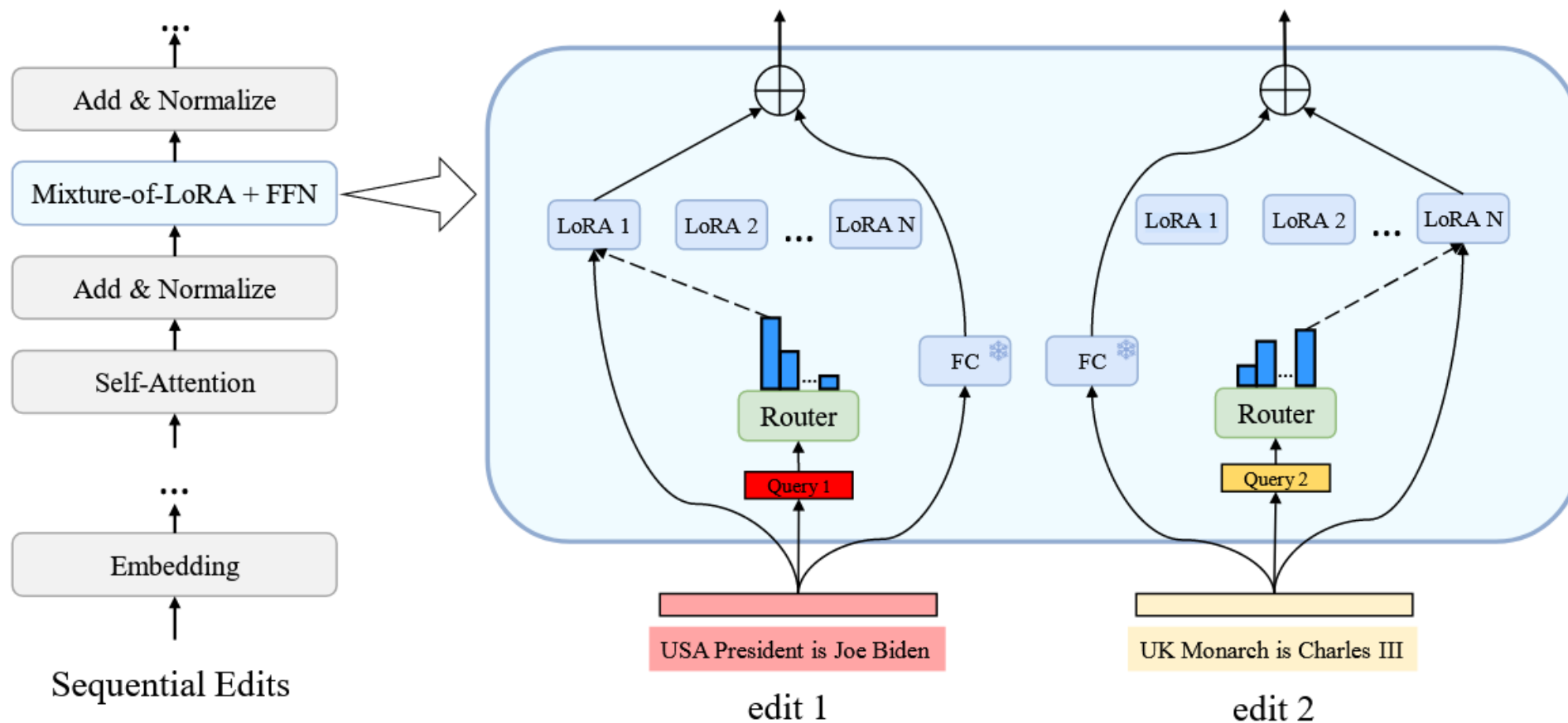
**Figure 1: Mixture of Experts Layer.** Each input vector is assigned to 2 of the 8 experts by a router. The layer's output is the weighted sum of the outputs of the two selected experts. In Mixtral, an expert is a standard feedforward block as in a vanilla transformer architecture.

# Mixtral 8x7B



**Figure 2: Performance of Mixtral and different Llama models on a wide range of benchmarks.** All models were re-evaluated on all metrics with our evaluation pipeline for accurate comparison. Mixtral outperforms or matches Llama 2 70B on all benchmarks. In particular, it is vastly superior in mathematics and code generation.

# ELDER: Mixture-of-LoRA



# Mixture of Experts

---

PROS

CONS