Quantization, Pruning and Distillation

Dinesh Raghu

Senior Researcher, IBM Research

Introduction to Large Language Models

LLM Sizes

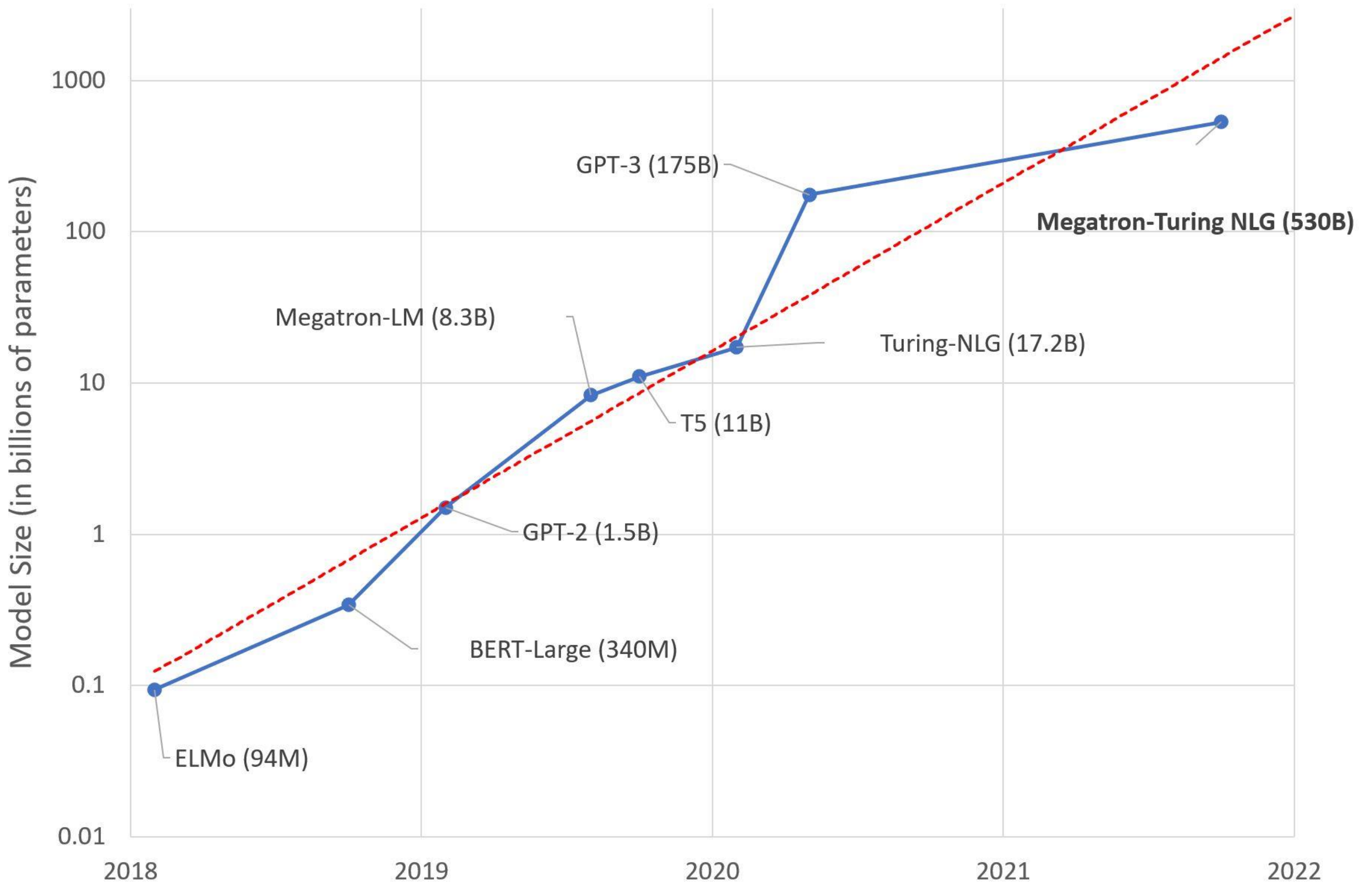
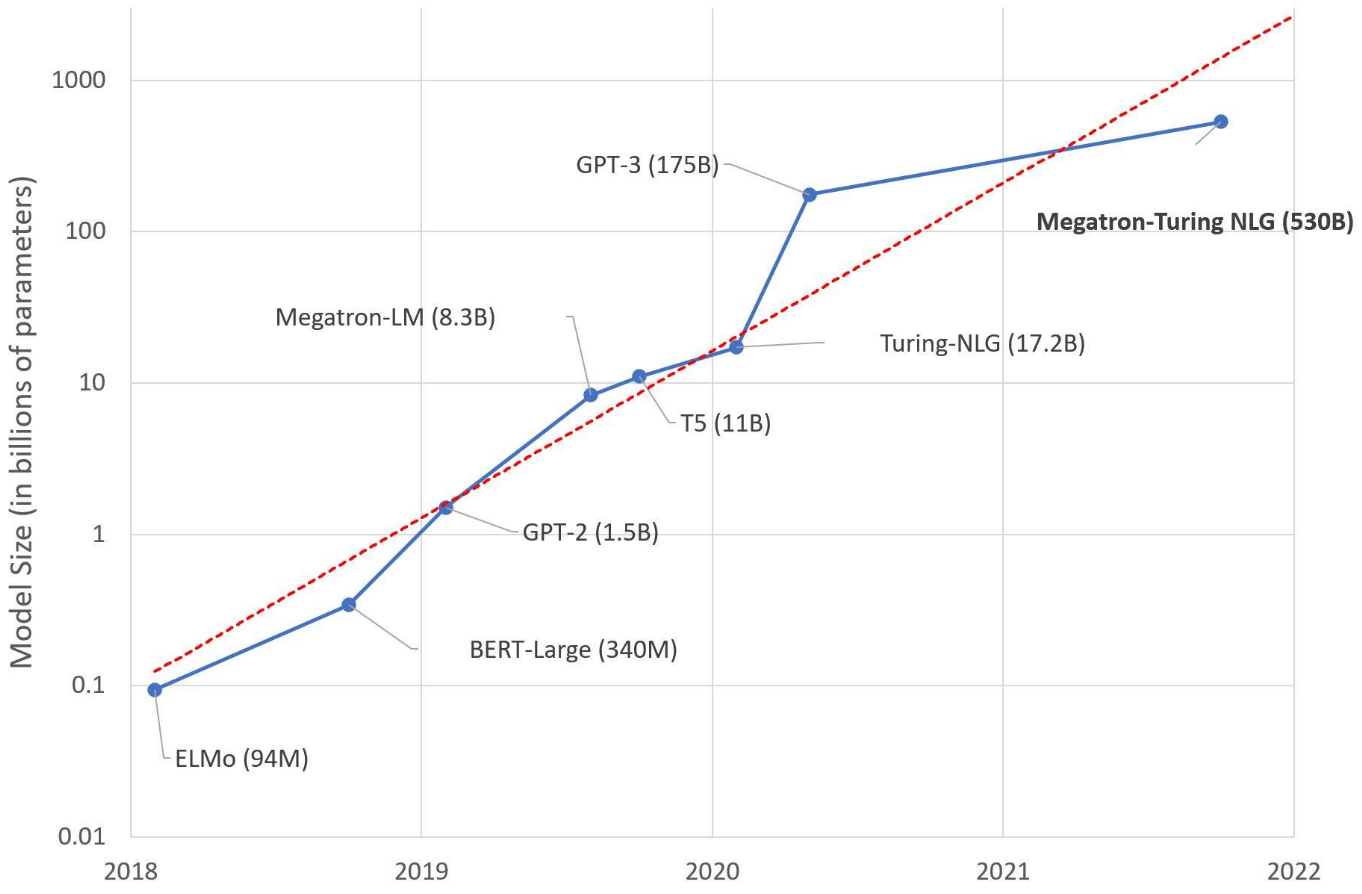


Image credits: https://huggingface.co/blog/large-language-models

LLM Sizes



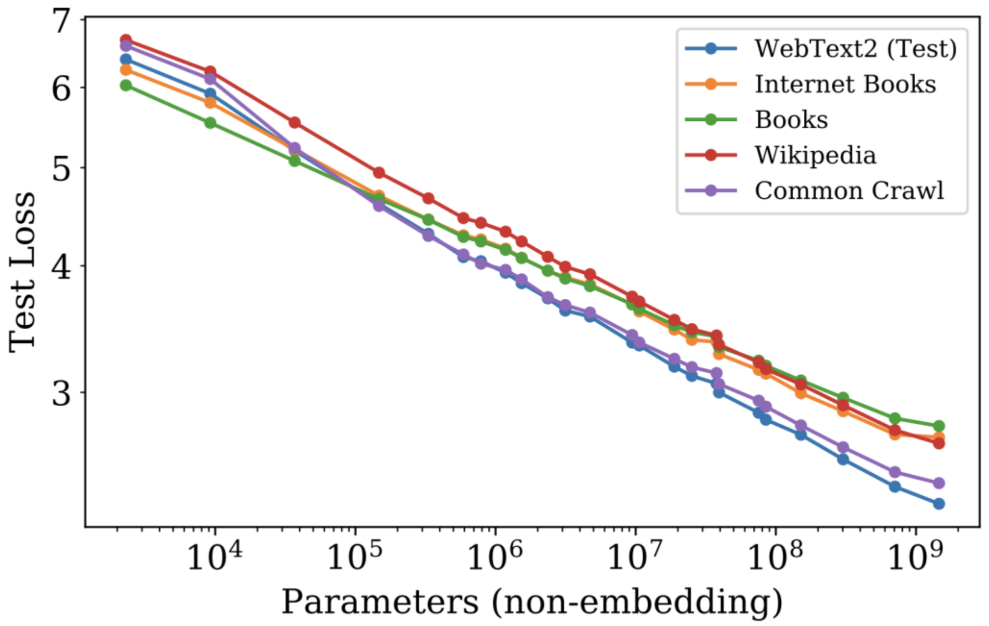
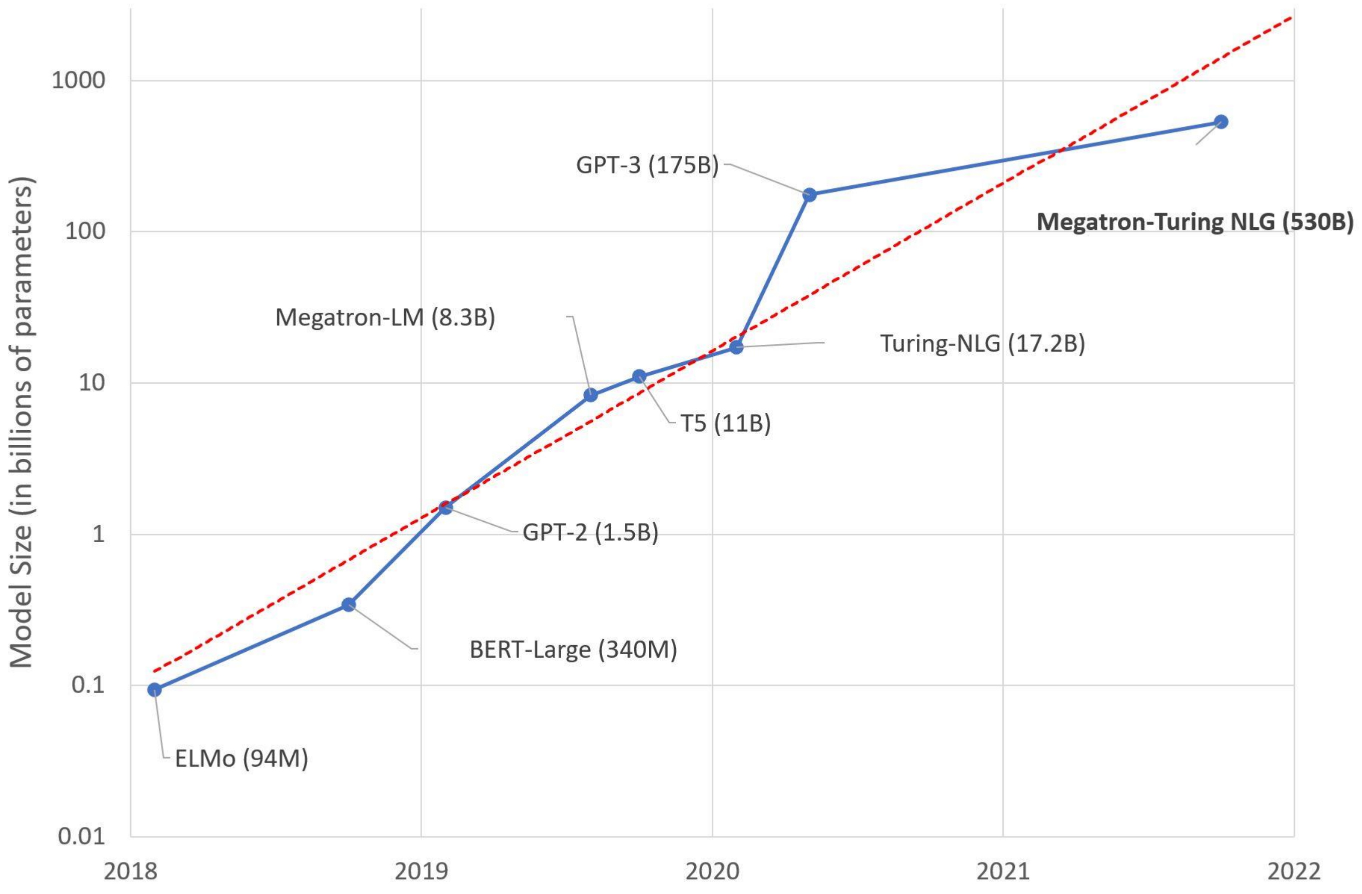


Image credits: https://huggingface.co/blog/large-language-models

LLM Inference



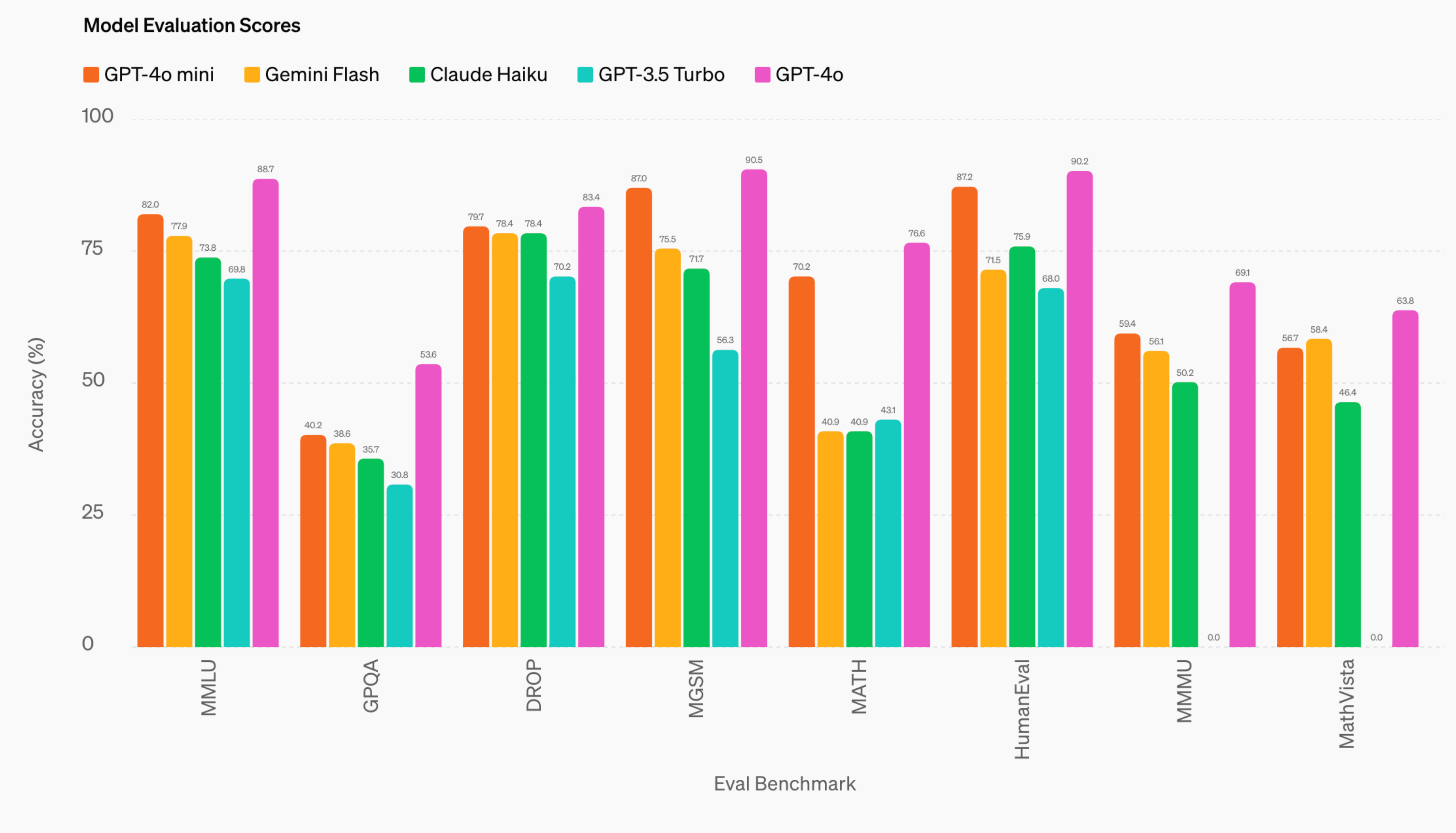
Larger the model, larger the

1. GPU memory requirement
2. latency
3. inference cost
4. environmental concerns

Image credits: https://huggingface.co/blog/large-language-models

LLM Inference

Image credits: openai.com



LLM Inference

Calculate by: Tokens | Words | Characters

Input tokens: 100

Output tokens: 500

Number of API calls: 10000

Provider	Model	Input price for 1M tokens	Output price for 1M tokens	Price per API call	Total price
OpenAI	gpt-4o	$5.00	$15.00	$0.0080	**$80.00**
OpenAI	gpt-4o-mini	$0.15	$0.60	$0.0003	**$3.15**

Why is gpt-4o-mini so cheap when compared to gpt-4o?

Image credits: gptforwork.com

LLM Inference

Calculate by: Tokens Words Characters

Input tokens: 100

Output tokens: 500

Number of API calls: 10000

Provider	Model	Input price for 1M tokens	Output price for 1M tokens	Price per API call	Total price
OpenAI	gpt-4o	\$5.00	\$15.00	\$0.0080	**\$80.00**
OpenAI	gpt-4o-mini	\$0.15	\$0.60	\$0.0003	**\$3.15**

~~Why is gpt-4o-mini so cheap when compared to gpt-4o?~~

How can we deploy LLMs in a cost-effective manner while maintaining high performance?

Image credits: gptforwork.com

Cost Effective Inference

1. Model Compression (lossy)

2. Efficient Engineering (lossless)

Cost Effective Inference

1. Model Compression (lossy)

2. Efficient Engineering (lossless)

Cost Effective Inference

1. Model Compression (lossy)
    1. Quantization
    2. Pruning
    3. Distillation

2. Efficient Engineering (lossless)

Cost Effective Inference

1. Model Compression (lossy)
   1. Quantization: keep the model the same but reduce the number of bits
   2. Pruning: remove parts of a model while retaining performance
   3. Distillation: train a smaller model to imitate the bigger model

2. Efficient Engineering (lossless)

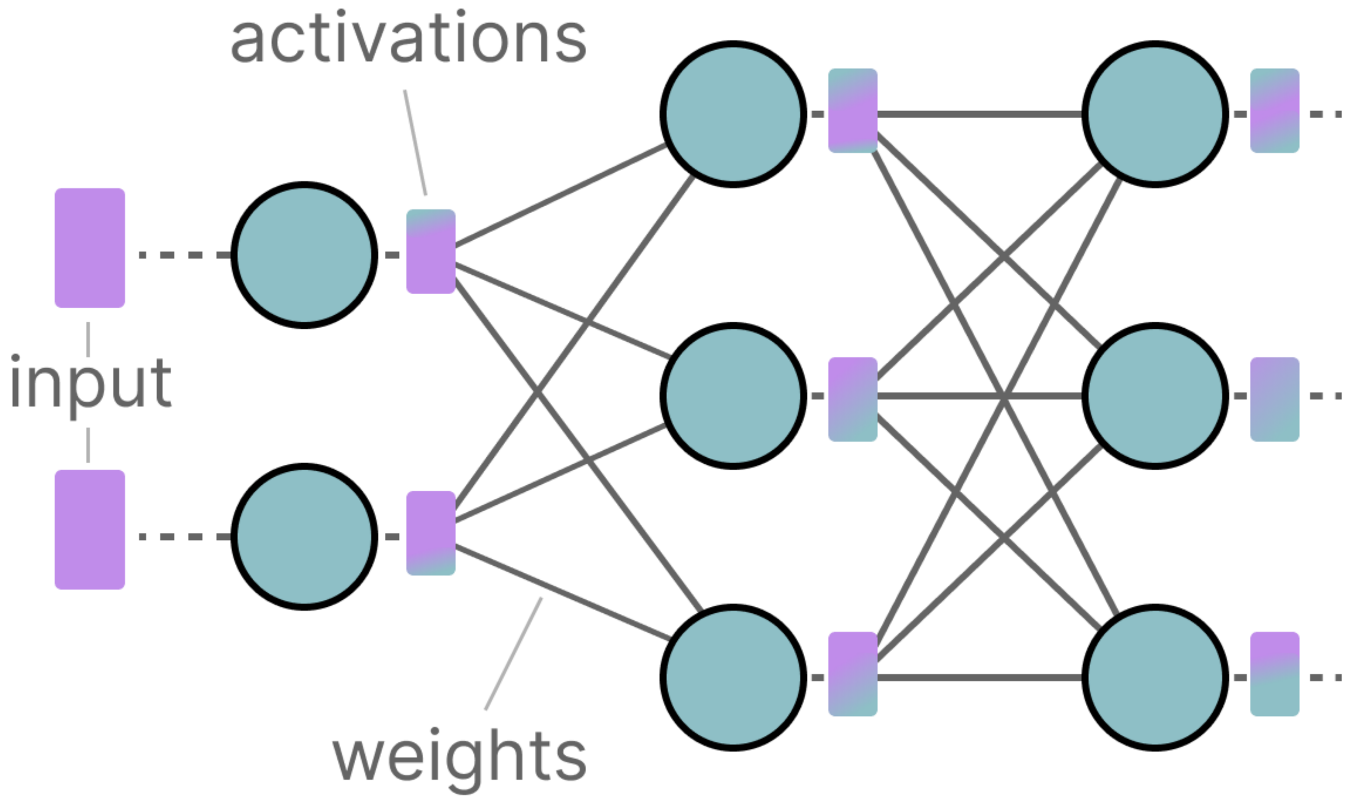
Model Compression

1. **Quantization:** keep the model the same but reduce the number of bits

2. **Pruning:** remove parts of a model while retaining performance

3. **Distillation:** train a smaller model to imitate the bigger model

Quantization: Problem with LLMs



- LLMs have billions of parameters which are expensive to store
- During inference, activations are created as a product of the input and the weights, which similarly are expensive to store
- The goal is to represent billions of values as efficiently as possible

Image credits: [Maarten Grootendorst]

Quantization: Numerical Values Representation

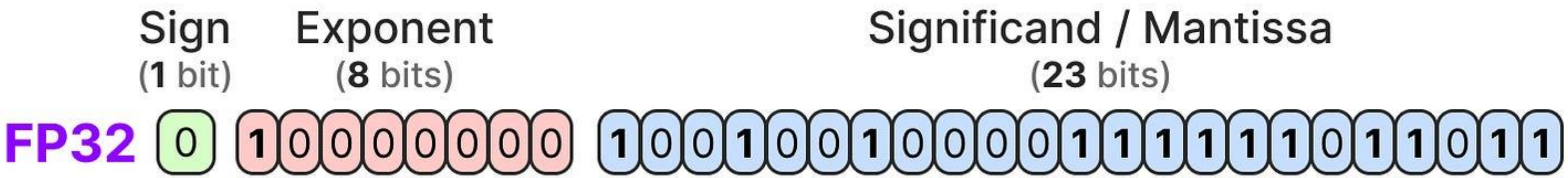


Image credits: [Maarten Grootendorst]

Quantization: Numerical Values Representation

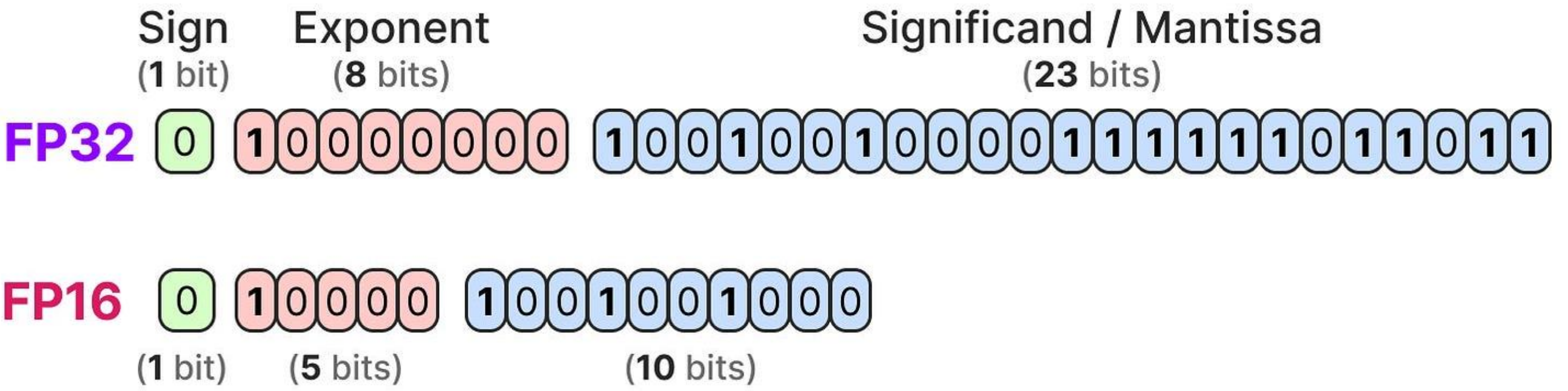


Image credits: Maarten Grootendorst

Quantization: Numerical Values Representation

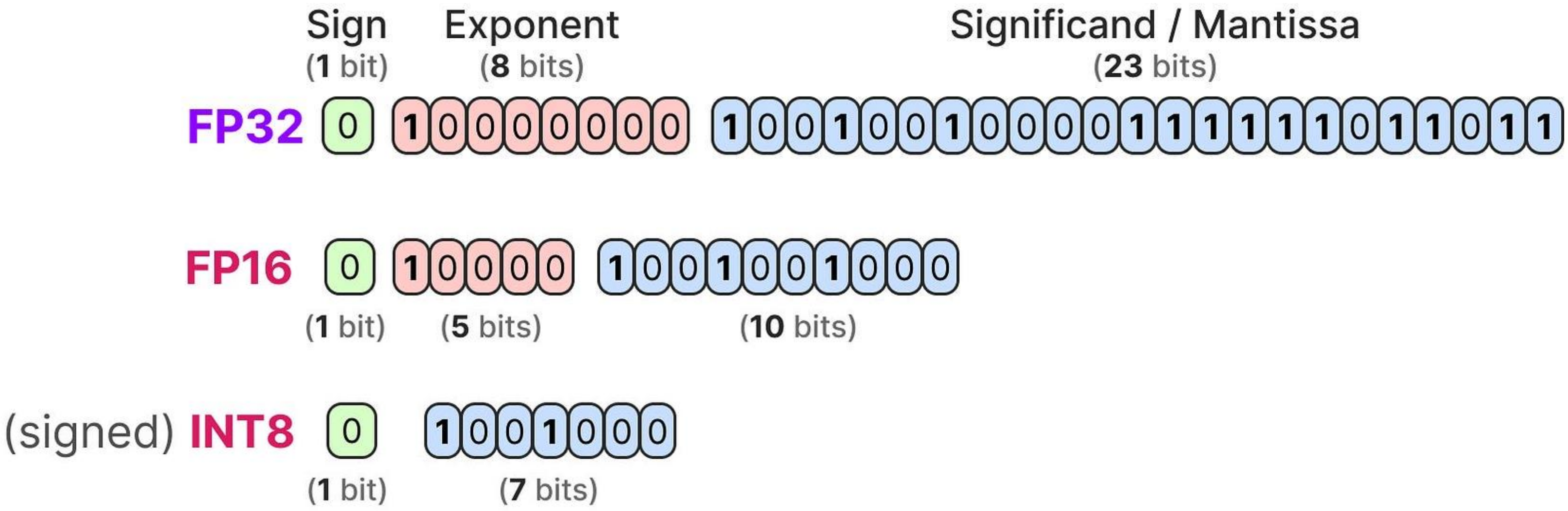


Image credits: [Maarten Grootendorst]

Quantizing FP32 to INT8

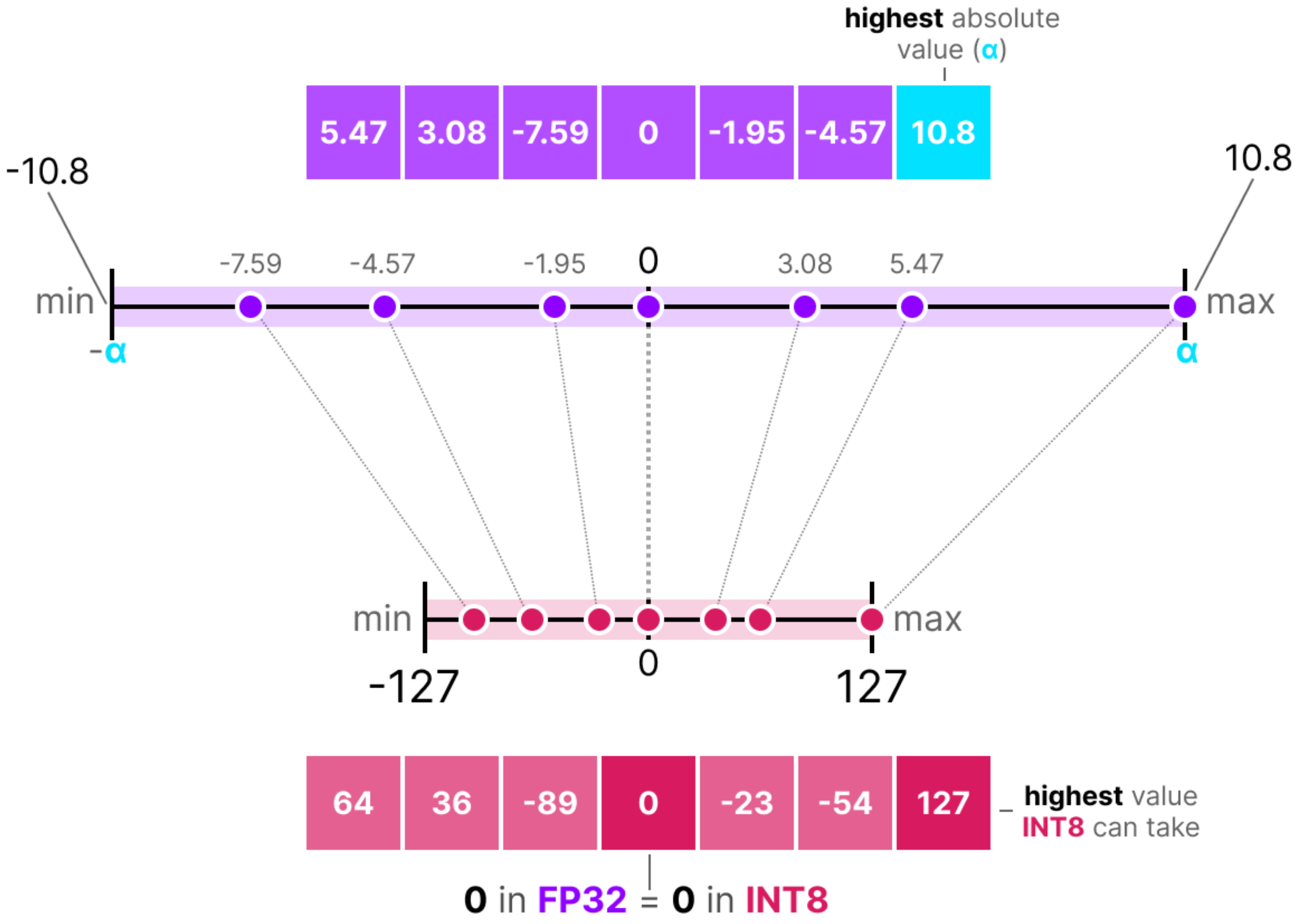
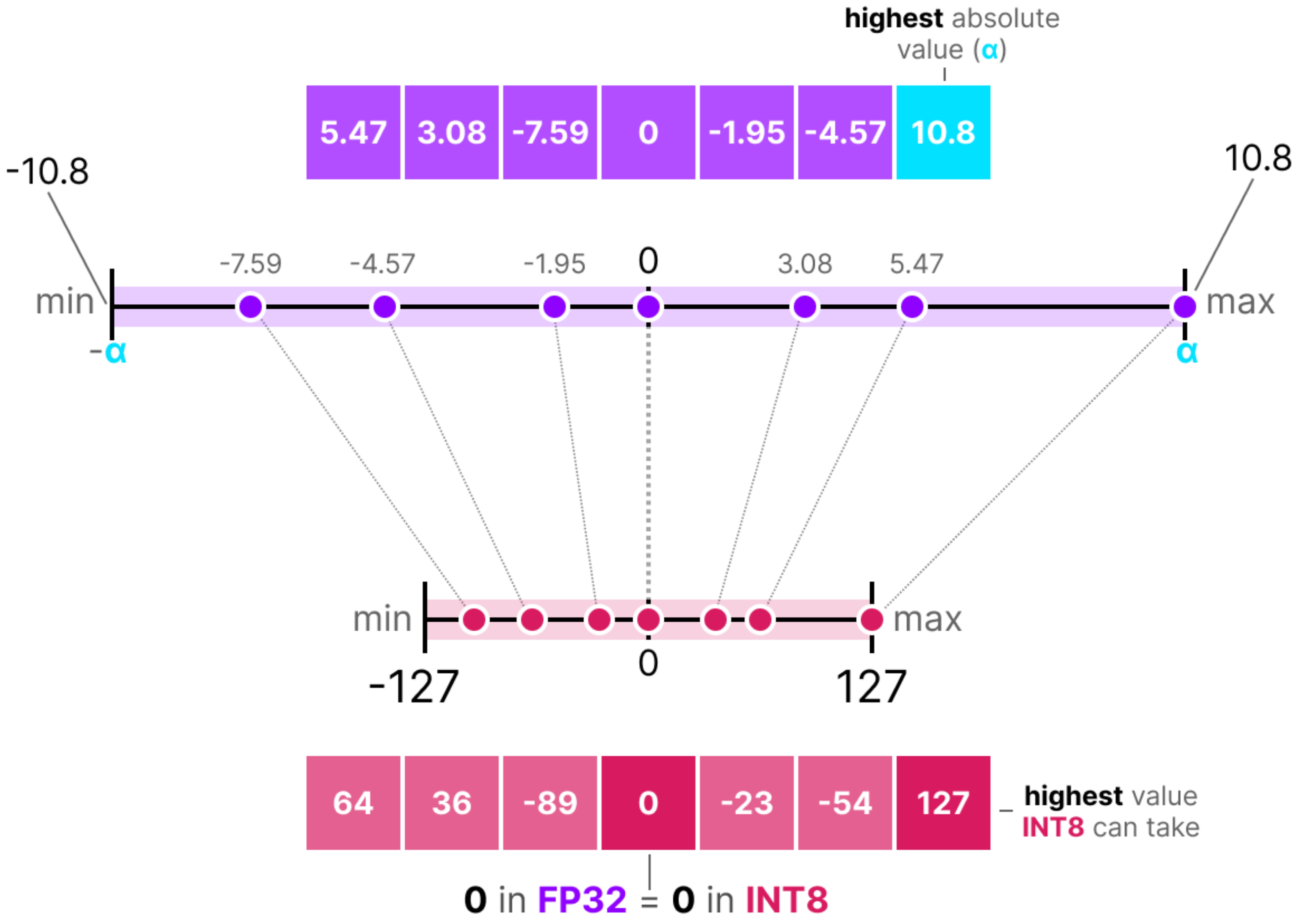


Image credits: [Maarten Grootendorst]

Quantizing FP32 to INT8



$$s = \frac{2^{b-1}-1}{\alpha}$$ (scale factor)

$$X_{quantized} = \text{round}\left(s \cdot X\right)$$ (quantization)

$$s = \frac{127}{10.8} = 11.76$$ (scale factor)

$$X_{quantized} = \text{round}\left(11.76 \cdot \blacksquare\right)$$ (quantization)

Image credits: [Maarten Grootendorst]()

Dequantizing INT8 to FP32

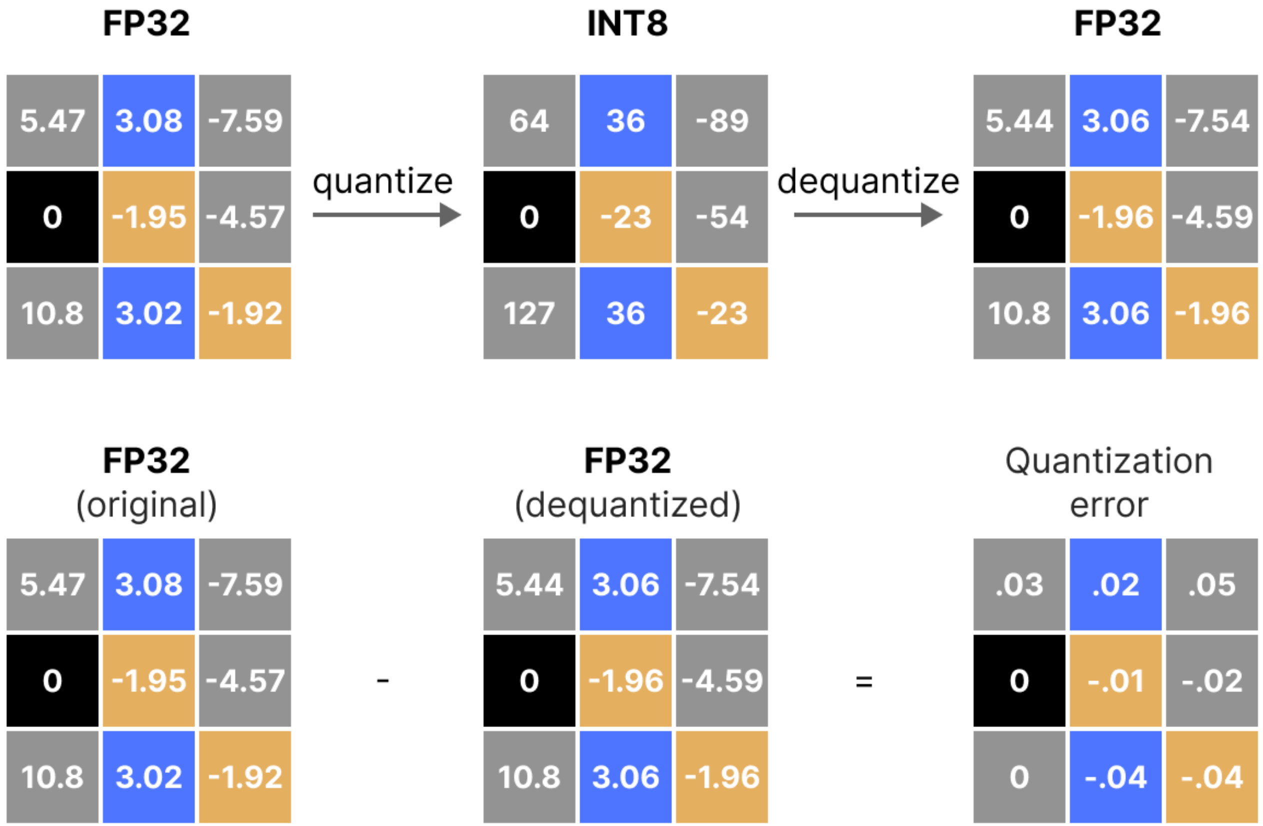
$$s = \frac{2^{b-1} - 1}{\alpha} \quad \text{(scale factor)}$$

$$X_{quantized} = \text{round}\left(s \cdot X\right) \quad \text{(quantization)}$$

$$X_{dequantized} = \frac{\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare}{s} \quad \text{(dequantize)}$$

Image credits: [Maarten Grootendorst]

Dequantizing INT8 to FP32



$$s = \frac{2^{b-1} - 1}{\alpha} \qquad \text{(scale factor)}$$

$$X_{quantized} = \text{round}\left(s \cdot X\right) \qquad \text{(quantization)}$$

$$X_{dequantized} = \frac{\quad}{s} \qquad \text{(dequantize)}$$

Image credits: [Maarten Grootendorst]

Model Compression

1. **Quantization:** keep the model the same but reduce the number of bits
    1. Post Training Quantization
    2. Quantization Aware Training

2. **Pruning:** remove parts of a model while retaining performance

3. **Distillation:** train a smaller model to imitate the bigger model

Post Training Quantization (PTQ)

- Reduce the model size without altering the LLM architecture and without retraining

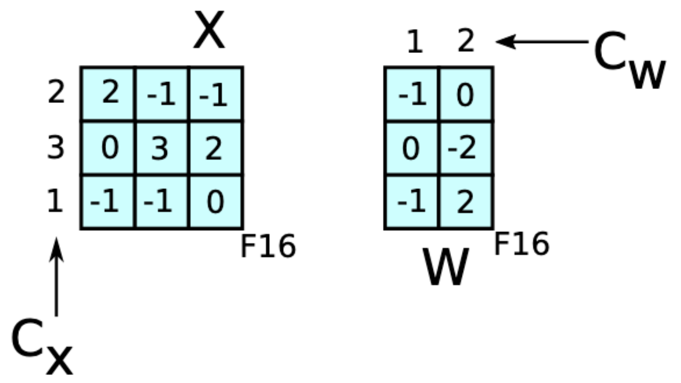
- Weights and biases are constants. Easy to compute the scale factor(s).

- Model input and activations are variable. Use a calibration dataset to compute the scale factor(s).

Post Training Quantization (PTQ)

8-bit Vector-wise Quantization

(1) Find vector-wise constants: C_w & C_x



(2) Quantize

$$X_{F16} * (127/C_x) = X_{I8}$$

$$W_{F16} * (127/C_w) = W_{I8}$$

(3) Int8 Matmul

$$X_{I8} W_{I8} = Out_{I32}$$

(4) Dequantize

$$\frac{Out_{I32} * (C_x \otimes C_w)}{127*127} = Out_{F16}$$

Image credits: [Dettmers et al., 2022]

Post Training Quantization (PTQ)

Technical Specifications		
	H100 SXM	**H100 NVL**
FP64	34 teraFLOPS	30 teraFLOPS
FP64 Tensor Core	67 teraFLOPS	60 teraFLOPS
FP32	67 teraFLOPS	60 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS	835 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS	3,341 teraFLOPS
INT8 Tensor Core*	3,958 TOPS	3,341 TOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	3.35TB/s	3.9TB/s

Datasheet

NVIDIA

NVIDIA H100 Tensor Core GPU

Extraordinary performance, scalability, and security for every data center.

Image credits: nvidia.com

PTQ: LLM.int8() [Dettmers et al., 2022]

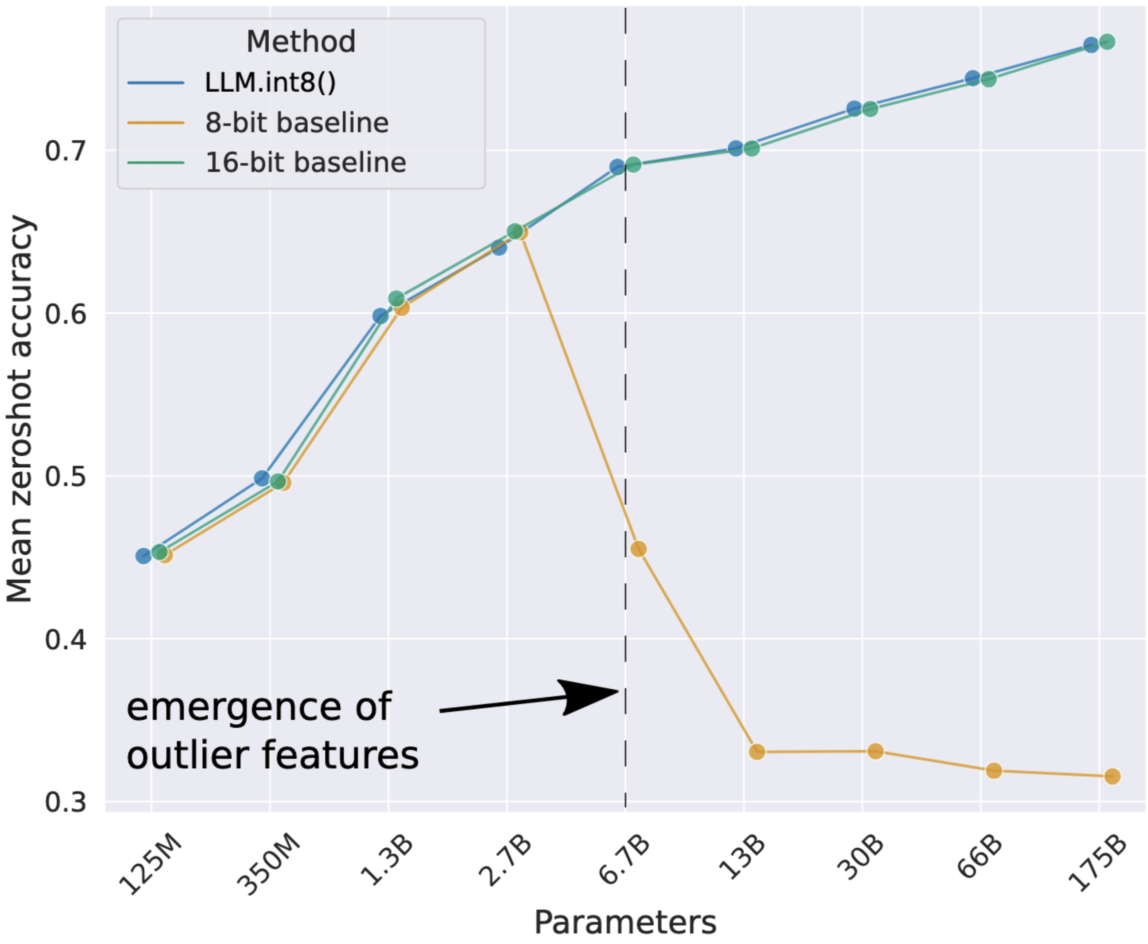


Image credits: [Dettmers et al., 2022]

- regular quantization retains performance at scales up to 2.7B parameters
- once systematic outliers occur at a scale of 6.7B parameters, regular quantization methods fail
- Irrespective of the scale, LLM.int8() maintains 16-bit accuracy

PTQ: LLM.int8()

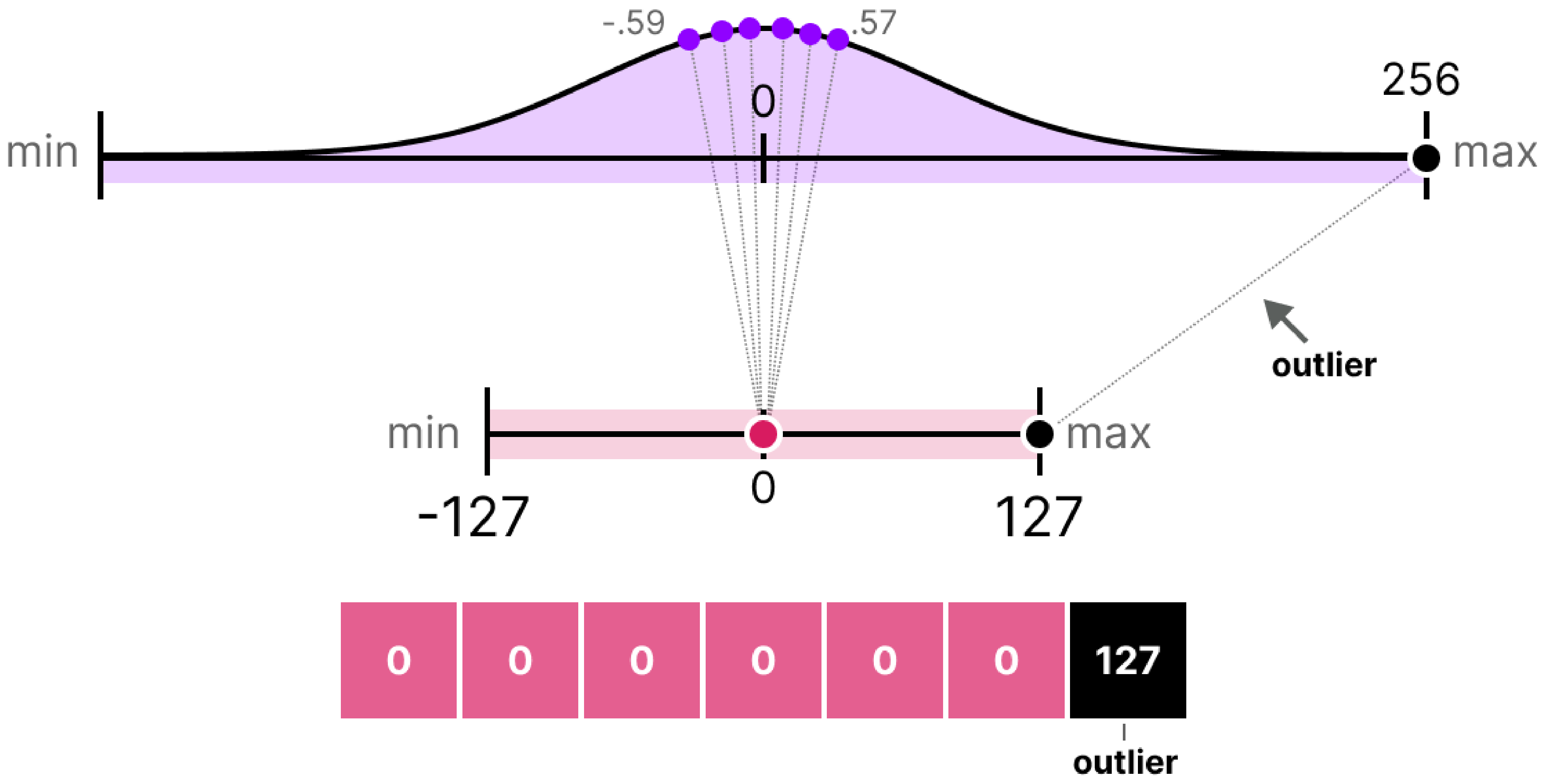


Image credits: Maarten Grootendorst

PTQ: LLM.int8()

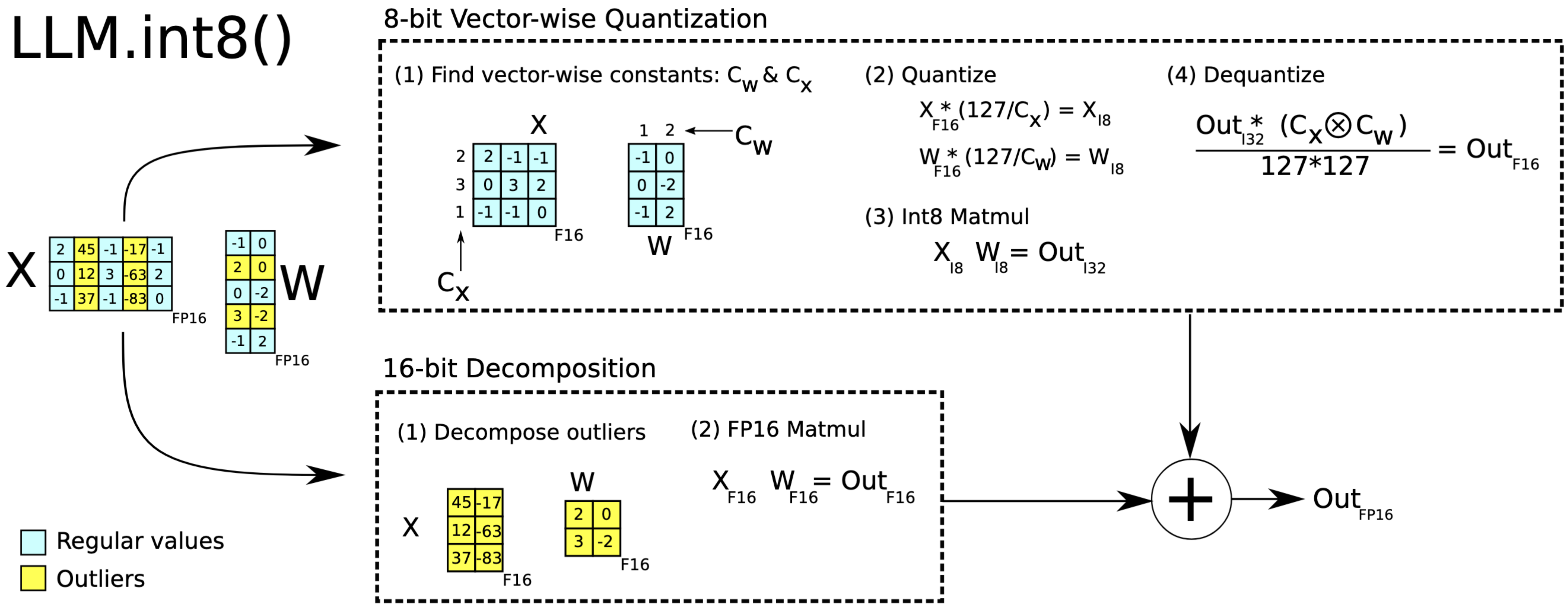


Image credits: Dettmers et al., 2022

Model Compression

1. **Quantization:** keep the model the same but reduce the number of bits
    1. Post Training Quantization
    2. Quantization Aware Training

2. **Pruning:** remove parts of a model while retaining performance

3. **Distillation:** train a smaller model to imitate the bigger model

QLoRA [Dettmers et al. 2023]

- Average memory requirements of finetuning a 65B parameter model is >780GB

- QLoRA reduces the memory requirement to <48GB without degrading the predictive performance

QLoRA [Dettmers et al. 2023]

1. 4-bit NormalFloat (NF4) Quantization
2. Double Quantization
3. Paged Optimizers
4. LoRA

QLoRA [Dettmers et al. 2023]

1. NF4 Quantization
2. Double Quantization
3. Paged Optimizers
4. LoRA

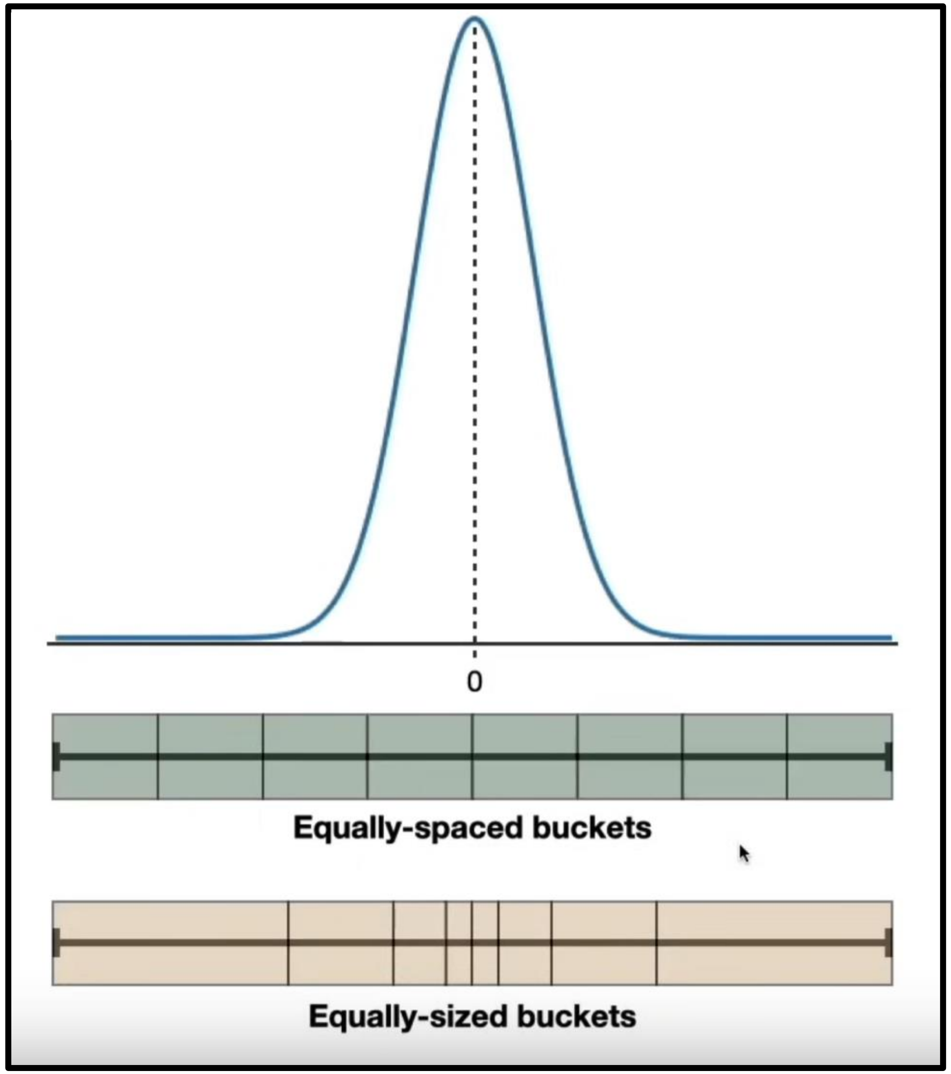


Image credits: Shaw Talebi

QLoRA [Dettmers et al. 2023]

Double Quantization is the process of quantizing the quantization constants for additional memory savings

Stored as FP32

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

QLoRA [Dettmers et al. 2023]

1. NF4 Quantization
2. Double Quantization
3. Paged Optimizers
4. LoRA

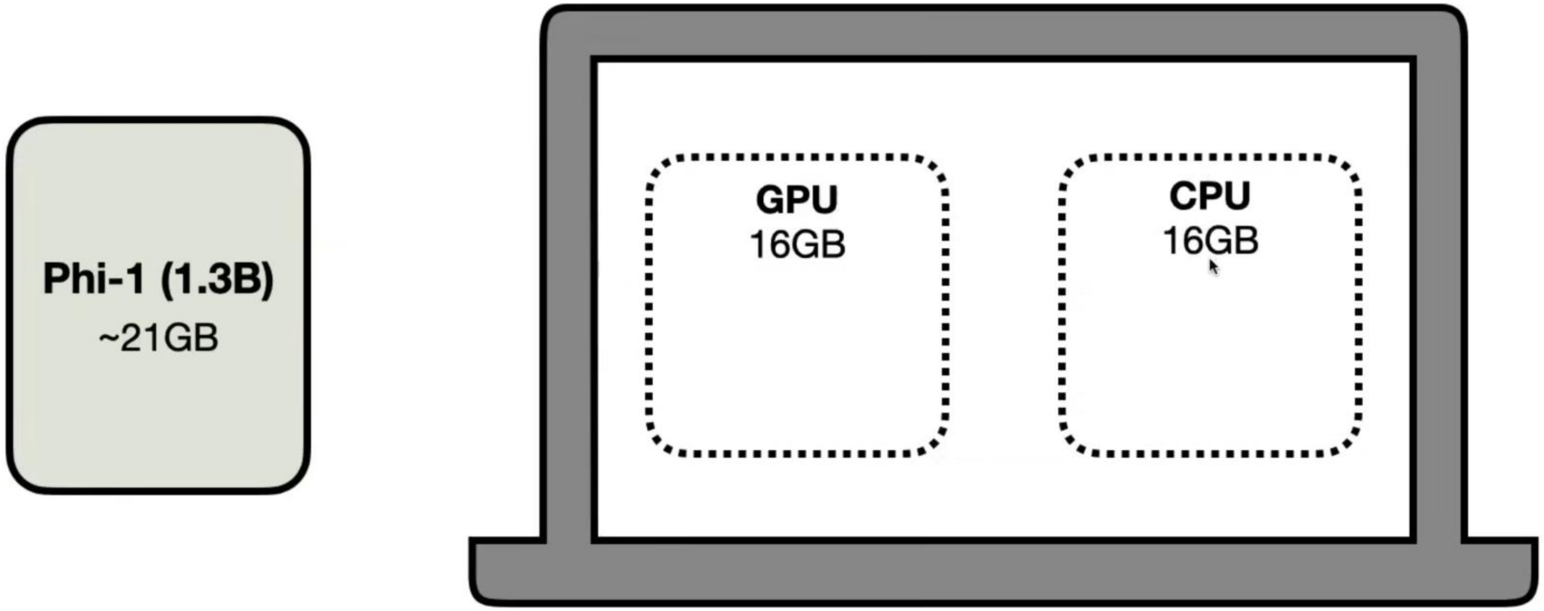


Image credits: Shaw Talebi

QLoRA [Dettmers et al. 2023]

1. NF4 Quantization
2. Double Quantization
3. Paged Optimizers
4. LoRA

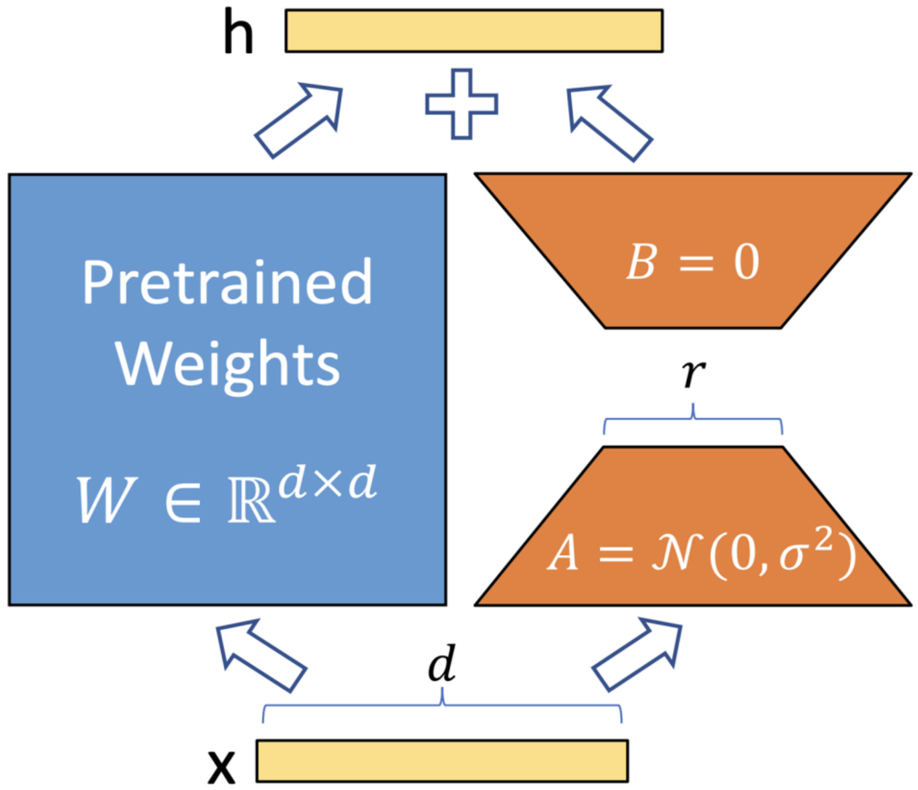


Image credits: [Hu et al., 2022]

QLoRA [Dettmers et al. 2023]

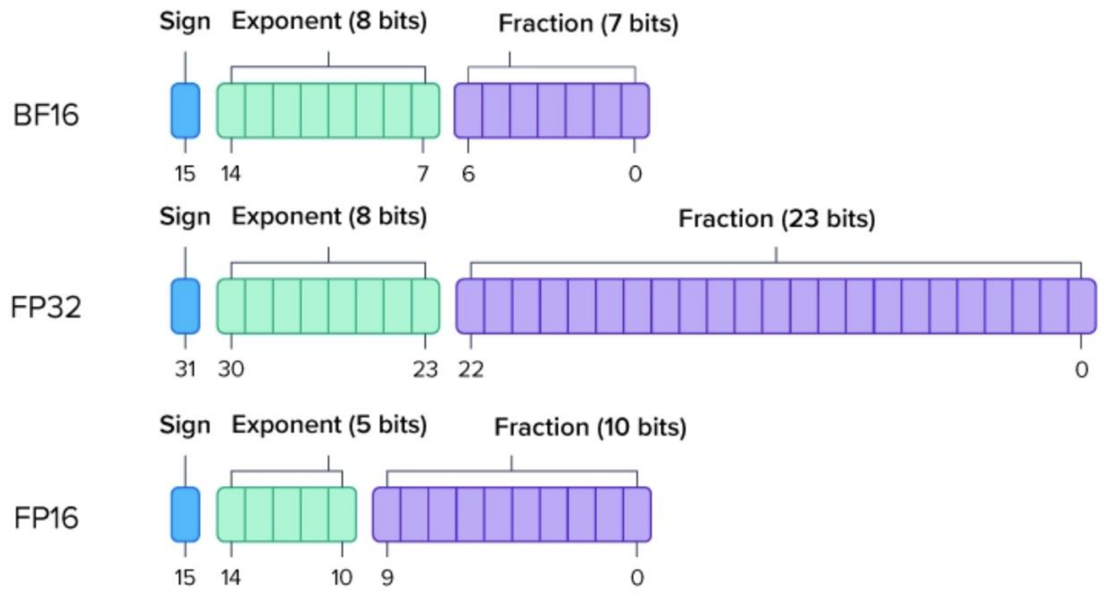
Image credits: Dettmers et al. 2023, exxactcorp.com

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + s\mathbf{X}\mathbf{L}_1\mathbf{L}_2$$

QLoRA [Dettmers et al. 2023]

Image credits: Dettmers et al. 2023, exxactcorp.com

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + s\mathbf{X}\mathbf{L}_1\mathbf{L}_2$$



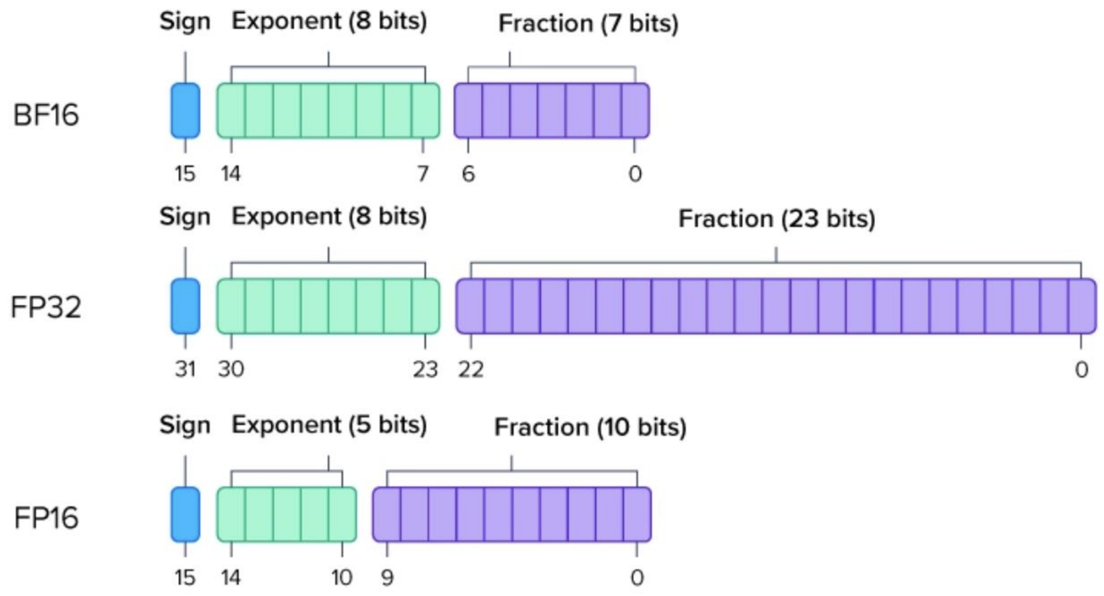
QLoRA [Dettmers et al. 2023]

Image credits: Dettmers et al. 2023, exxactcorp.com

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + s\mathbf{X}\mathbf{L}_1\mathbf{L}_2$$

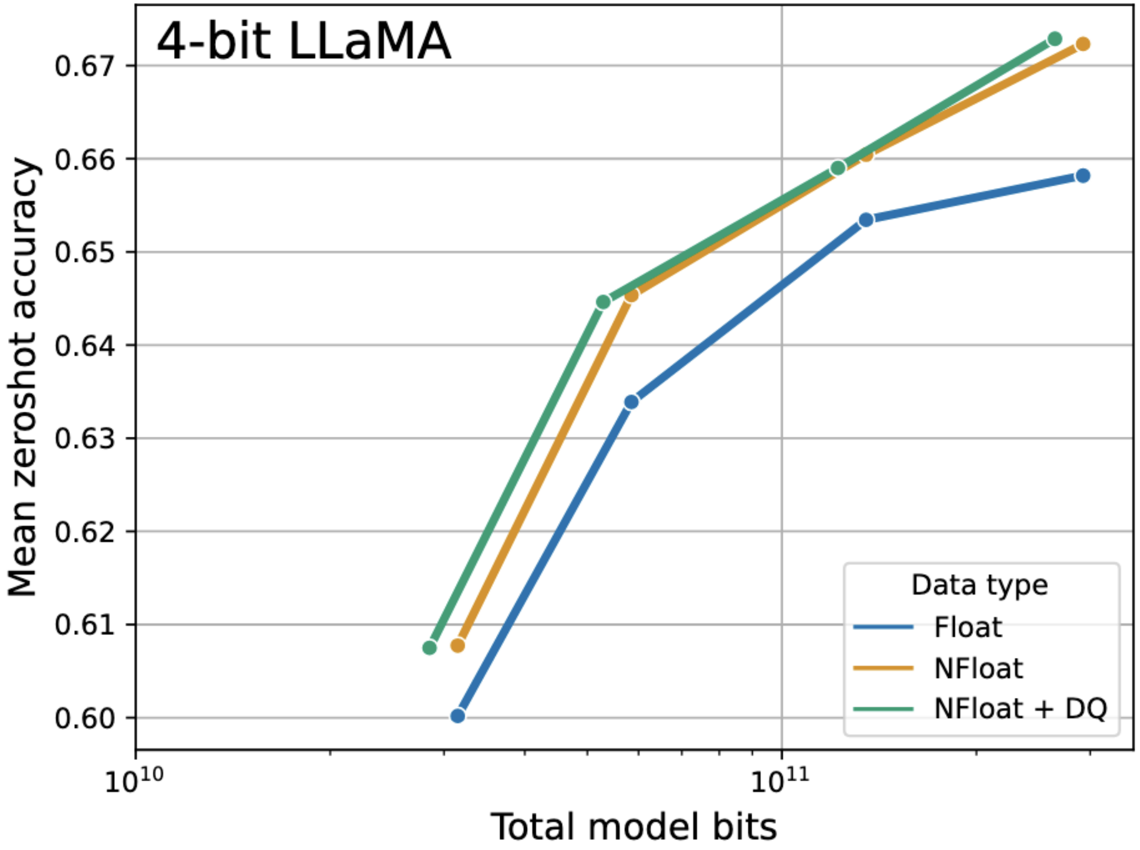
$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}},$$

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}$$



QLoRA [Dettmers et al. 2023]

Image credits: Dettmers et al. 2023



Mean zero-shot accuracy over Winogrande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types.

- NFloat data type improves the bit-for-bit accuracy gains compared to regular 4-bit Floats
- Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint

QLoRA [Dettmers et al. 2023]

Image credits: Dettmers et al. 2023

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

Mean 5-shot MMLU test accuracy for LLaMA models finetuned with adapters on Alpaca and FLAN v2 for different data types.

Model Compression

1. Quantization: keep the model the same but reduce the number of bits

2. Pruning: remove parts of a model while retaining performance

3. Distillation: train a smaller model to imitate the bigger model

Pruning

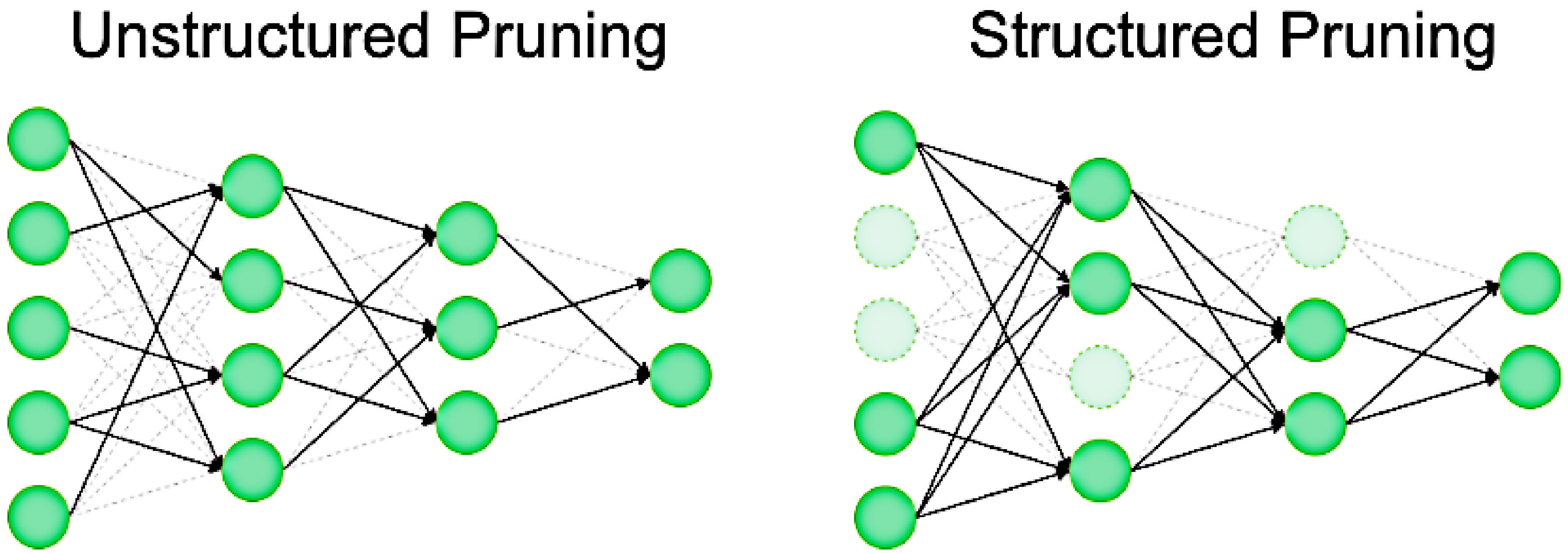
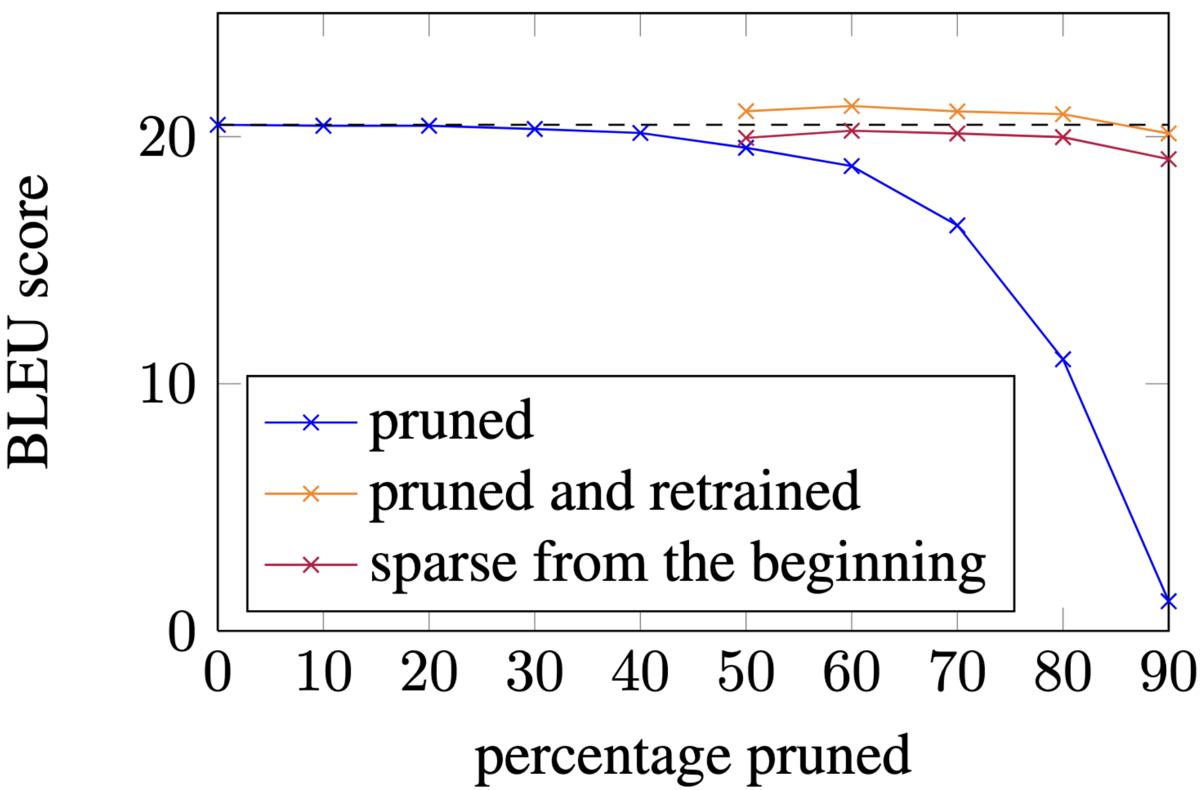


Image credits: neuralmagic.com

Magnitude Pruning [Han et al. 2015, See et al. 2016]



- 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

Wanda [Sun et al. 2023]

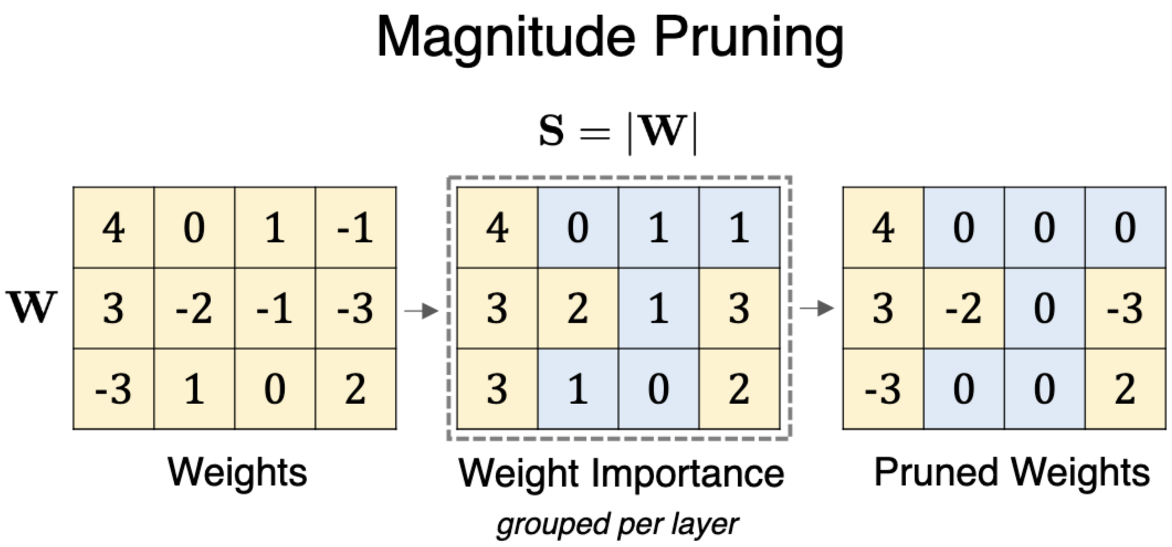


Image credits: Sun et al. 2023

Wanda [Sun et al. 2023]

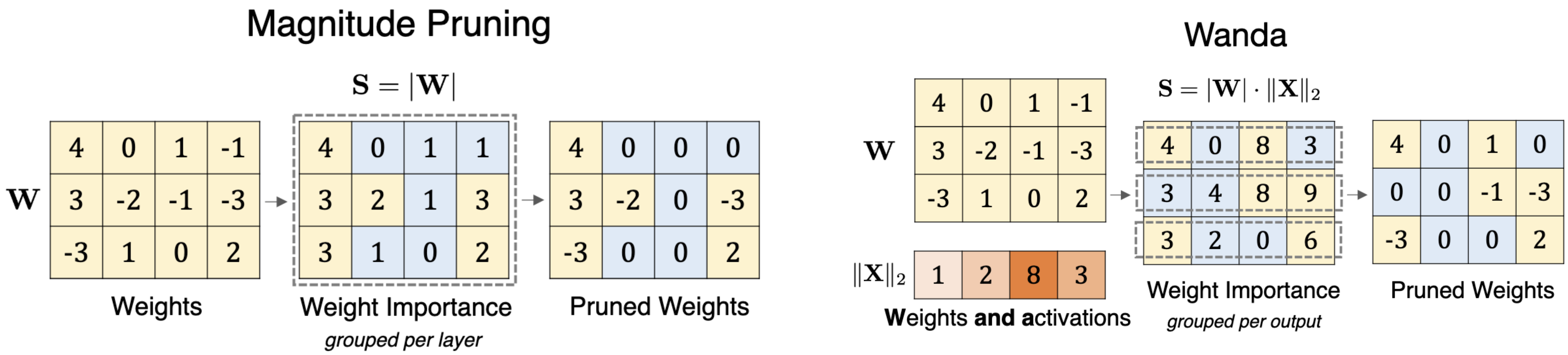


Image credits: Sun et al. 2023

Unstructured Pruning

Unstructured pruning can work only if the hardware supports.

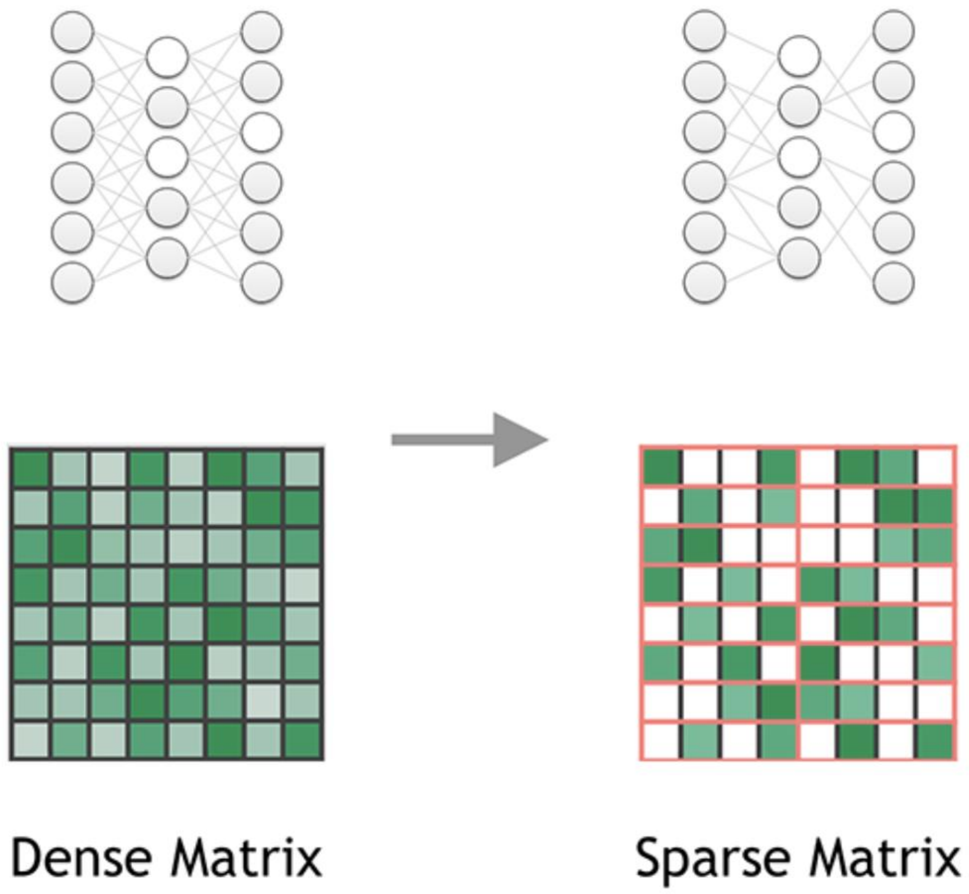
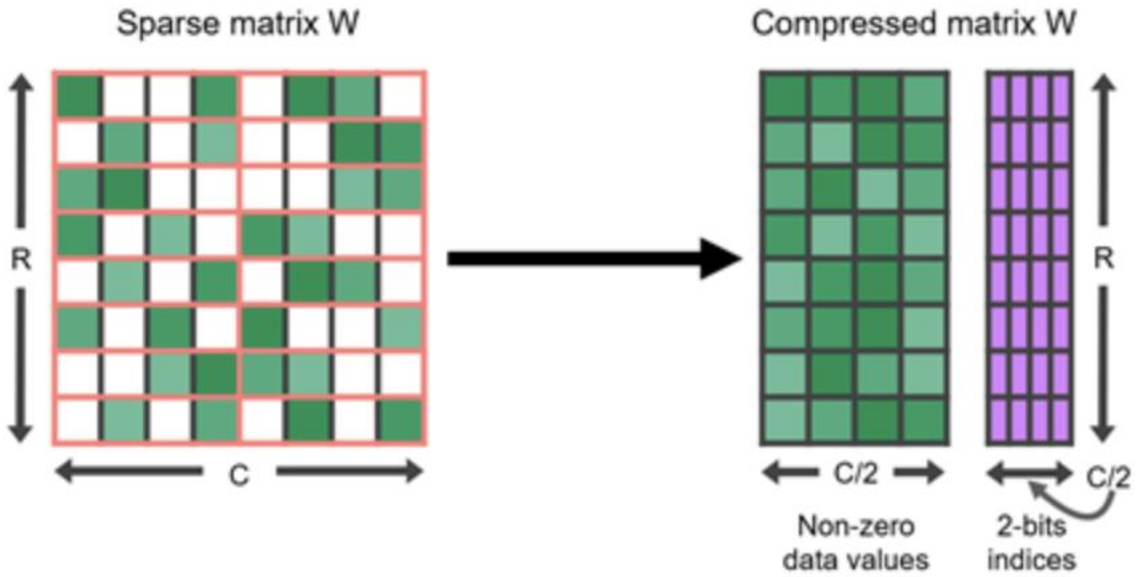


Image credits: nvidia.com

Structured Pruning

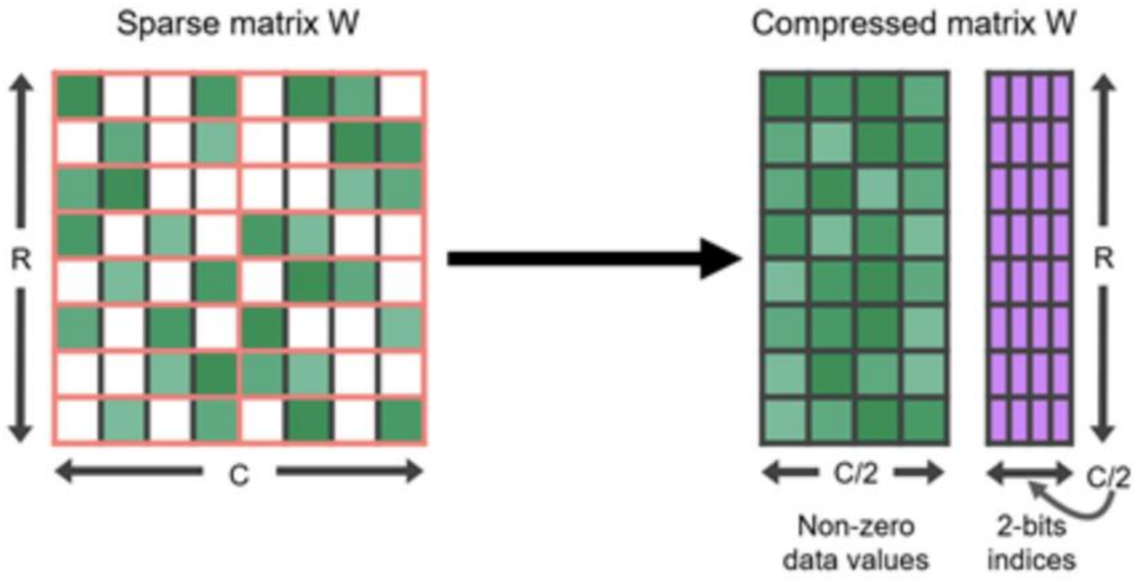


- NVIDIA A100 GPU supports fine-grained structured sparsity to its Tensor Cores
- Sparse Tensor Cores accelerate a 2:4 sparsity pattern.

Image credits: nvidia.com

Structured Pruning

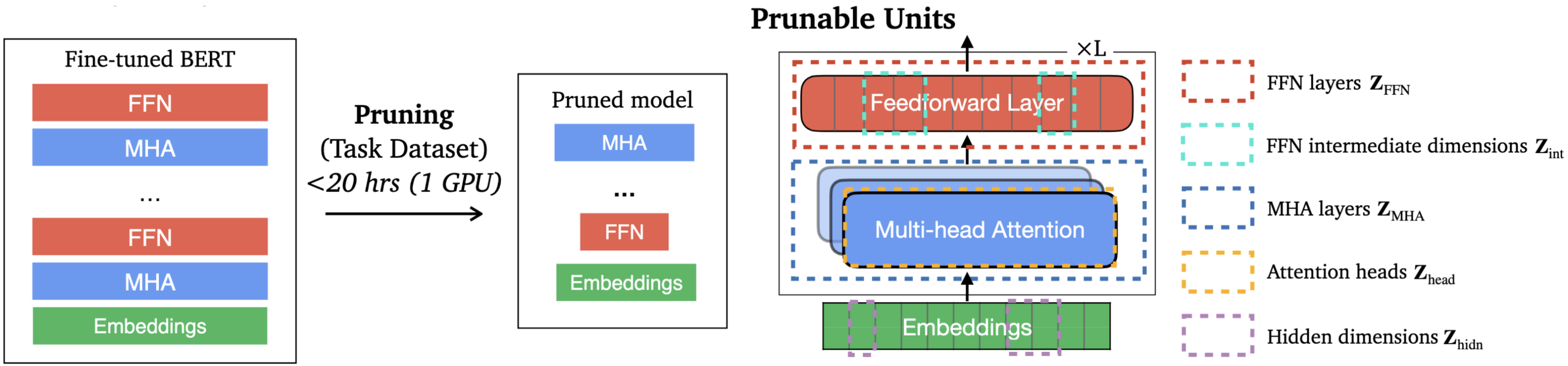
Input Operands	Accumulator	Dense TOPS	vs. FFMA	Sparse TOPS	vs. FFMA
FP32	FP32	19.5	–	-	-
TF32	FP32	156	8X	**312**	**16X**
FP16	FP32	312	16X	**624**	**32X**
BF16	FP32	312	16X	**624**	**32X**



- NVIDIA A100 GPU supports fine-grained structured sparsity to its Tensor Cores
- Sparse Tensor Cores accelerate a 2:4 sparsity pattern.

Image credits: nvidia.com

Structured Pruning [Xia et al. 2022]



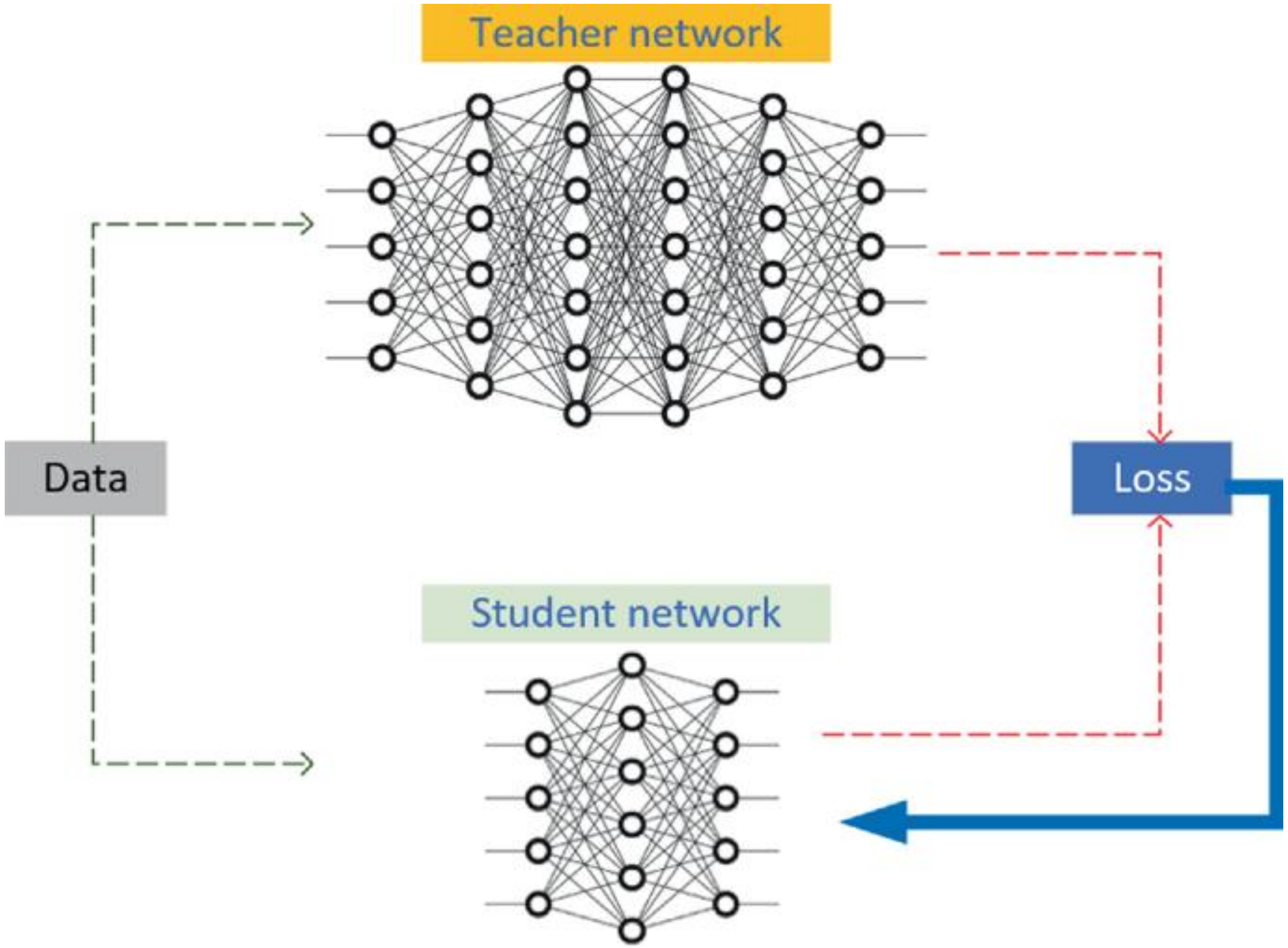
Model Compression

1. **Quantization:** keep the model the same but reduce the number of bits

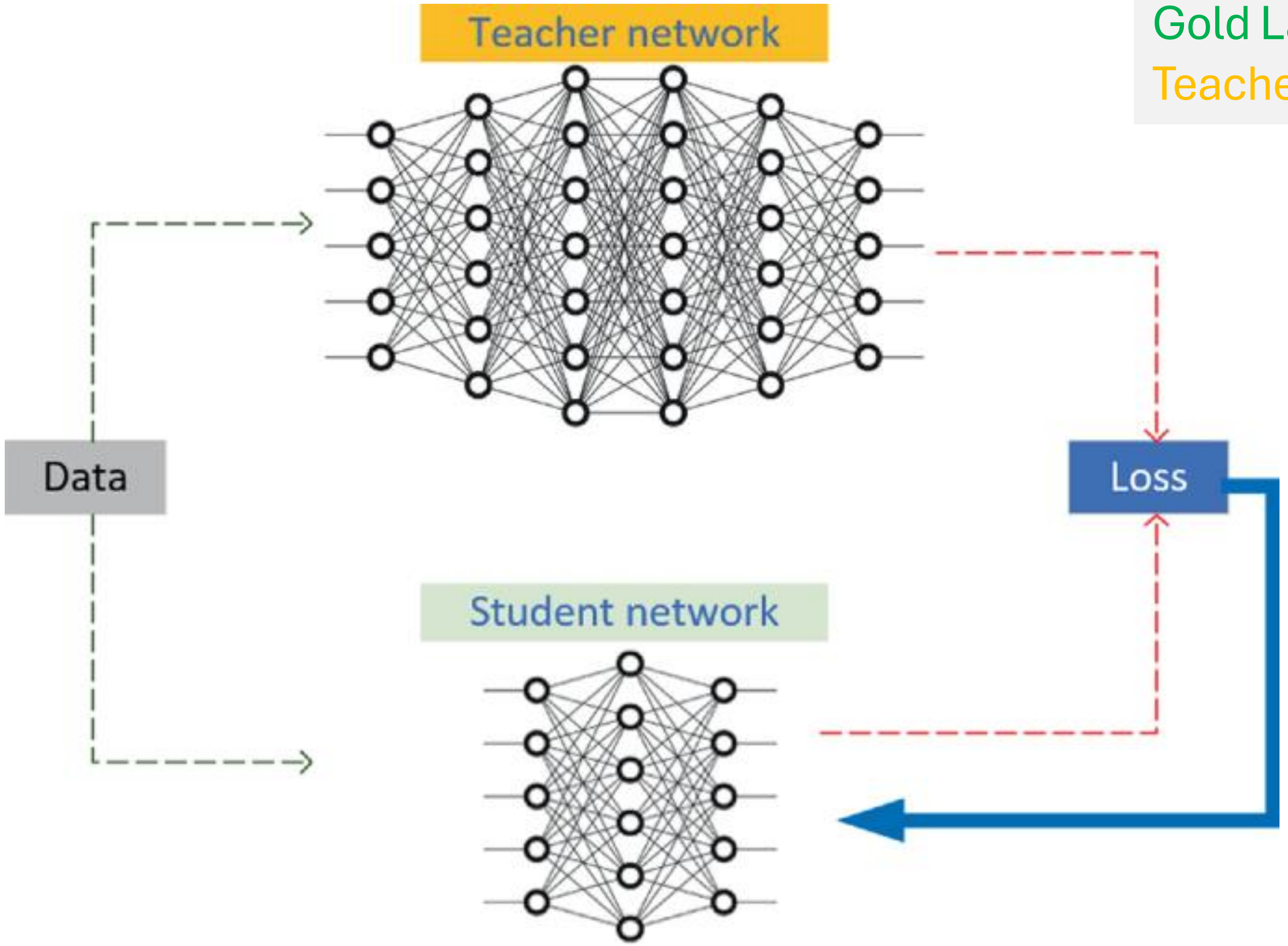
2. **Pruning:** remove parts of a model while retaining performance

3. **Distillation:** train a smaller model to imitate the bigger model

Distillation [Hinton et al 2015]



Distillation [Hinton et al 2015]

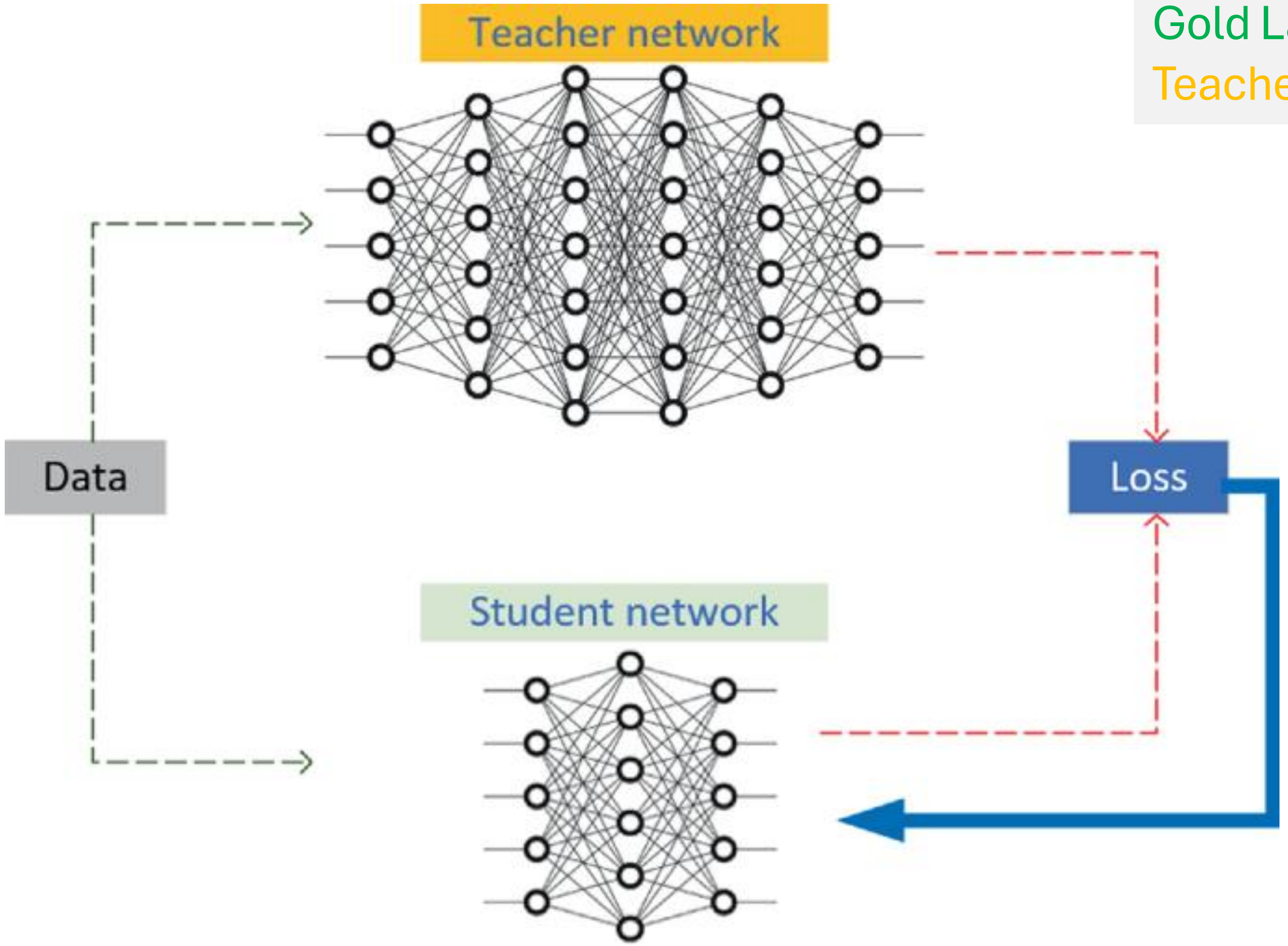


Gold Label	0	0	1	0	0
Teacher Prediction	0.10	0.20	0.50	0.15	0.05

$$\mathcal{L}_{\text{NLL}}(\theta) = -\sum_{k=1}^{|\mathcal{V}|} \mathbb{1}\{y = k\} \log p(y = k \mid x; \theta)$$

$$\mathcal{L}_{\text{KD}}(\theta; \theta_T) = -\sum_{k=1}^{|\mathcal{V}|} q(y = k \mid x; \theta_T) \times \log p(y = k \mid x; \theta)$$

Distillation [Hinton et al 2015]



Gold Label	[	0	0	1	0	0	]
Teacher Prediction	[	0.10	0.20	0.50	0.15	0.05	]

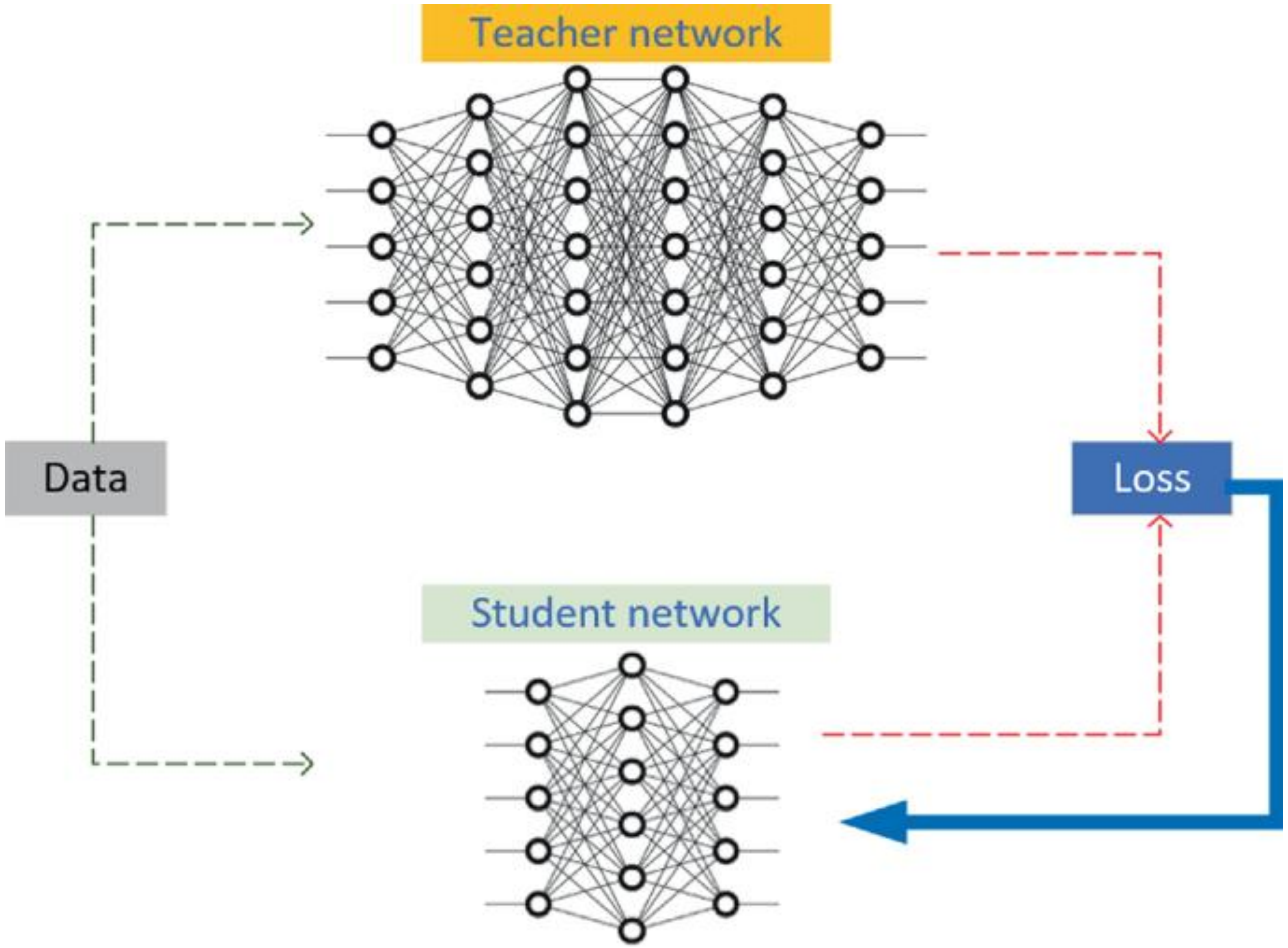
Pros:

- No restriction on student network structure
- Biggest potential gain in speed

Cons:

- Needs training data
- Expensive to train student and get soft labels from the teacher

Distillation [Hinton et al 2015]



Gold Label	[0	0	1	0	0]	
Soft Target	[0.90	0.01	0.05	0.01	0.03]	
Hard Target	[1.	0.	0	0	0]	

Sequence Level Distillation [Kim et al. 2016]

$$\mathcal{L}_{\text{KD}}(\theta;\theta_T) = -\sum_{k=1}^{|\mathcal{V}|} q(y=k\,|\,x;\theta_T)\times \log p(y=k\,|\,x;\theta)$$

Sequence Level Distillation [Kim et al. 2016]

1. Word-Level Knowledge Distillation

$$\mathcal{L}_{\text{WORD-KD}} = -\sum_{j=1}^{J}\sum_{k=1}^{|\mathcal{V}|} q(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j}) \times \log p(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j})$$

$$\boxed{\mathcal{L}_{\text{KD}}(\theta; \theta_T) = -\sum_{k=1}^{|\mathcal{V}|} q(y = k \mid x; \theta_T) \times \log p(y = k \mid x; \theta)}$$

Sequence Level Distillation [Kim et al. 2016]

1. Word-Level Knowledge Distillation

$$\mathcal{L}_{\text{WORD-KD}} = -\sum_{j=1}^{J} \sum_{k=1}^{|\mathcal{V}|} q(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j}) \times \log p(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j})$$

$$\mathcal{L}_{\text{KD}}(\theta; \theta_T) = -\sum_{k=1}^{|\mathcal{V}|} q(y = k \mid x; \theta_T) \times \log p(y = k \mid x; \theta)$$

2. Sequence-Level Knowledge Distillation

$$\begin{aligned}
\mathcal{L}_{\text{SEQ-KD}} &= -\sum_{\mathbf{t} \in \mathcal{T}} q(\mathbf{t} \mid \mathbf{s}) \log p(\mathbf{t} \mid \mathbf{s}) \\
&\approx -\sum_{\mathbf{t} \in \mathcal{T}} \mathbb{1}\{\mathbf{t} = \hat{\mathbf{y}}\} \log p(\mathbf{t} \mid \mathbf{s}) \\
&= -\log p(\mathbf{t} = \hat{\mathbf{y}} \mid \mathbf{s})
\end{aligned}$$

Sequence Level Distillation [Kim et al. 2016]

1. Word-Level Knowledge Distillation

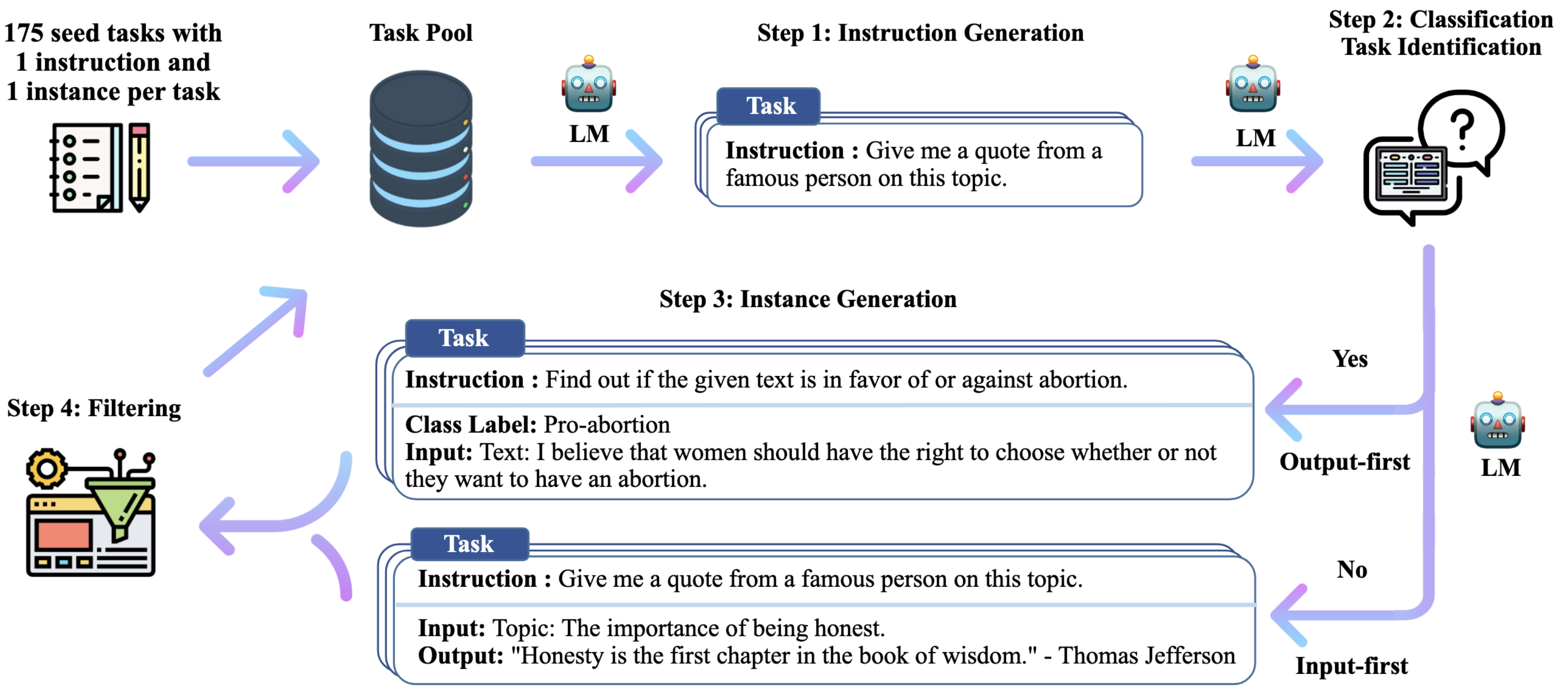
$$\mathcal{L}_{\text{WORD-KD}} = -\sum_{j=1}^{J}\sum_{k=1}^{|\mathcal{V}|} q(t_j = k \,|\, \mathbf{s}, \mathbf{t}_{<j}) \times \log p(t_j = k \,|\, \mathbf{s}, \mathbf{t}_{<j})$$

$$\boxed{\mathcal{L}_{\text{KD}}(\theta; \theta_T) = -\sum_{k=1}^{|\mathcal{V}|} q(y = k \,|\, x; \theta_T) \times \log p(y = k \,|\, x; \theta)}$$

2. Sequence-Level Knowledge Distillation

$$\mathcal{L}_{\text{SEQ-KD}} = -\sum_{\mathbf{t}\in\mathcal{T}} q(\mathbf{t} \,|\, \mathbf{s}) \log p(\mathbf{t} \,|\, \mathbf{s})$$

Self-Instruct [Wang et al. 2023]



Model Compression

1. **Quantization:** keep the model the same but reduce the number of bits

2. **Pruning:** remove parts of a model while retaining performance

3. **Distillation:** train a smaller model to imitate the bigger model