Lecture 7.3 - Parameter-Efficient Fine-tuning (PEFT) Methods

Generative AI Teaching Kit

This lecture

- Motivation for Parameter Efficient Fine-tuning (PEFT)
- Quantization and Pruning to reduce memory footprint
- Distillation of models
- Low Rank Adapters (LoRA/DoRA/QLoRA)

Why Parameter Efficient Fine-tuning?

GPUs don’t grow on trees

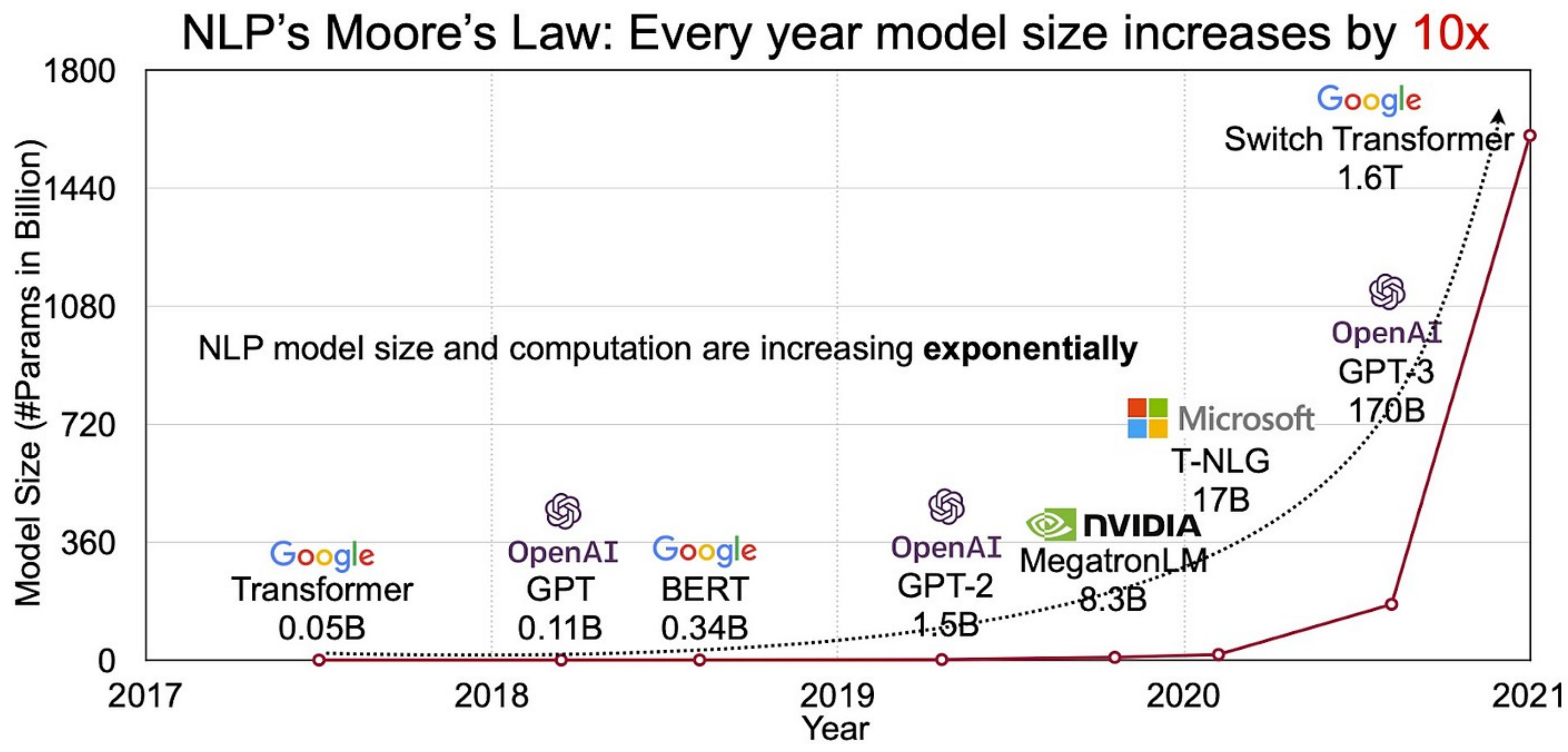
The cost of Large LLMs

LLMs have achieved state-of-the-art results in various Natural Language Processing tasks.

They have also started foraying into other domains, such as Computer Vision (CV) (VIT, Stable Diffusion, LayoutLM) and Audio (Whisper, XLS-R).

The conventional paradigm is large-scale pretraining on generic web-scale data, followed by fine-tuning to downstream tasks.

Fine-tuning these pretrained LLMs on downstream datasets results in huge performance gains when compared to using the pretrained LLMs out-of-the-box (zero-shot inference, for example).



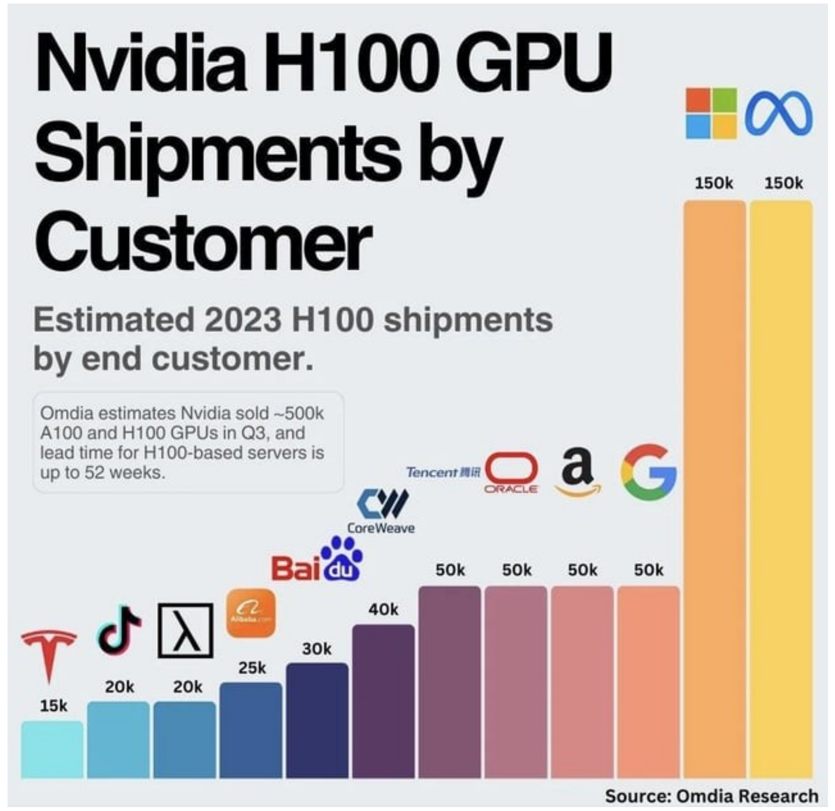
However, as models get larger and larger, full fine-tuning becomes infeasible to train on consumer hardware. In addition, storing and deploying fine-tuned models independently for each downstream task becomes very expensive, because fine-tuned models are the same size as the original pretrained model.

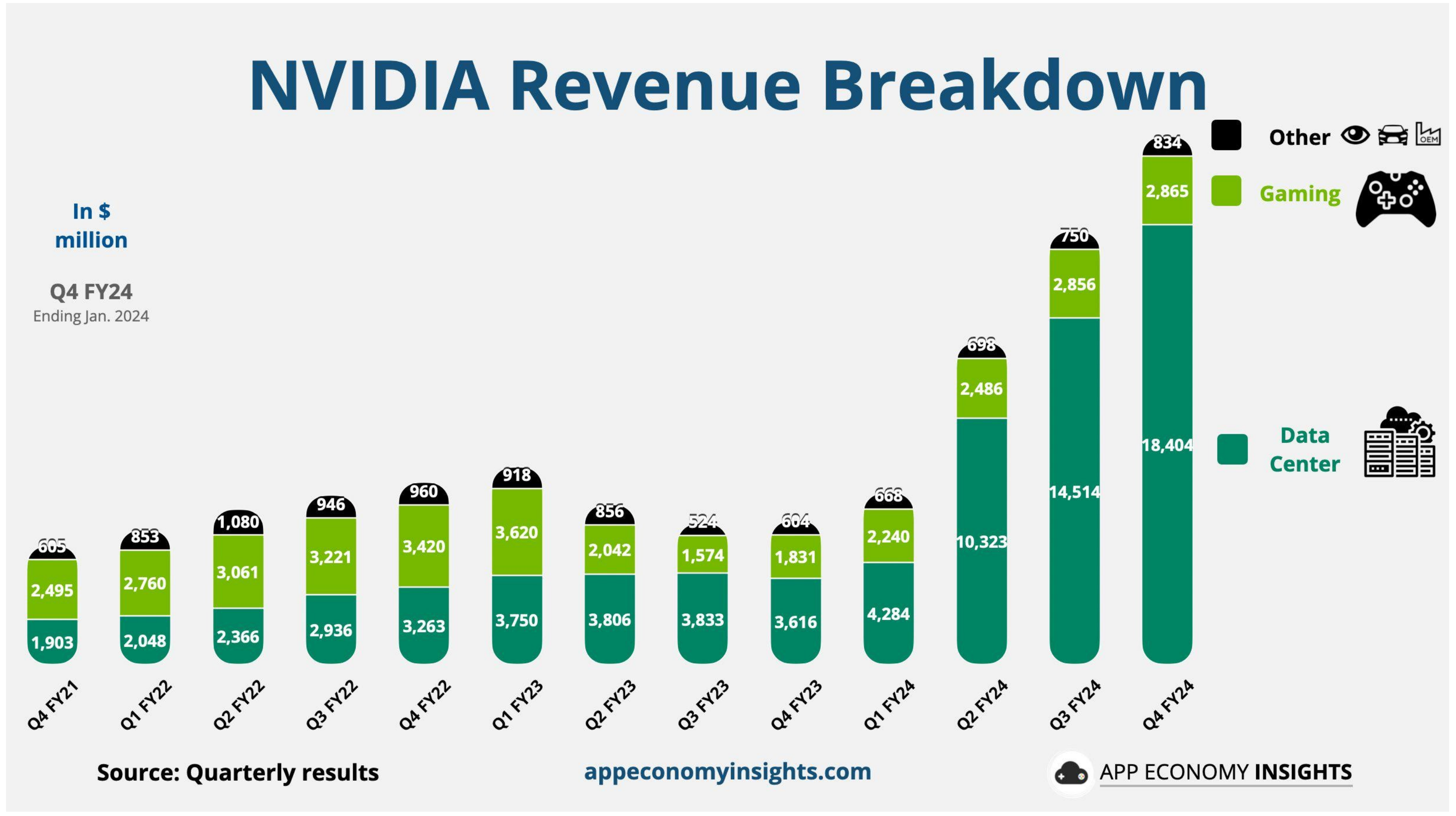
GPU Rich vs. GPU Poor

Consumers face challenges in LLM development due to the significant gap between consumer and enterprise GPUs.

Enterprise GPUs (like A100s or H100s) offer higher memory, faster processing, and better optimization for AI tasks compared to consumer GPUs (e.g., RTX series).

This disparity limits consumers' ability to train large models, as enterprise hardware is more suited for handling the massive parallelism and memory bandwidth required.





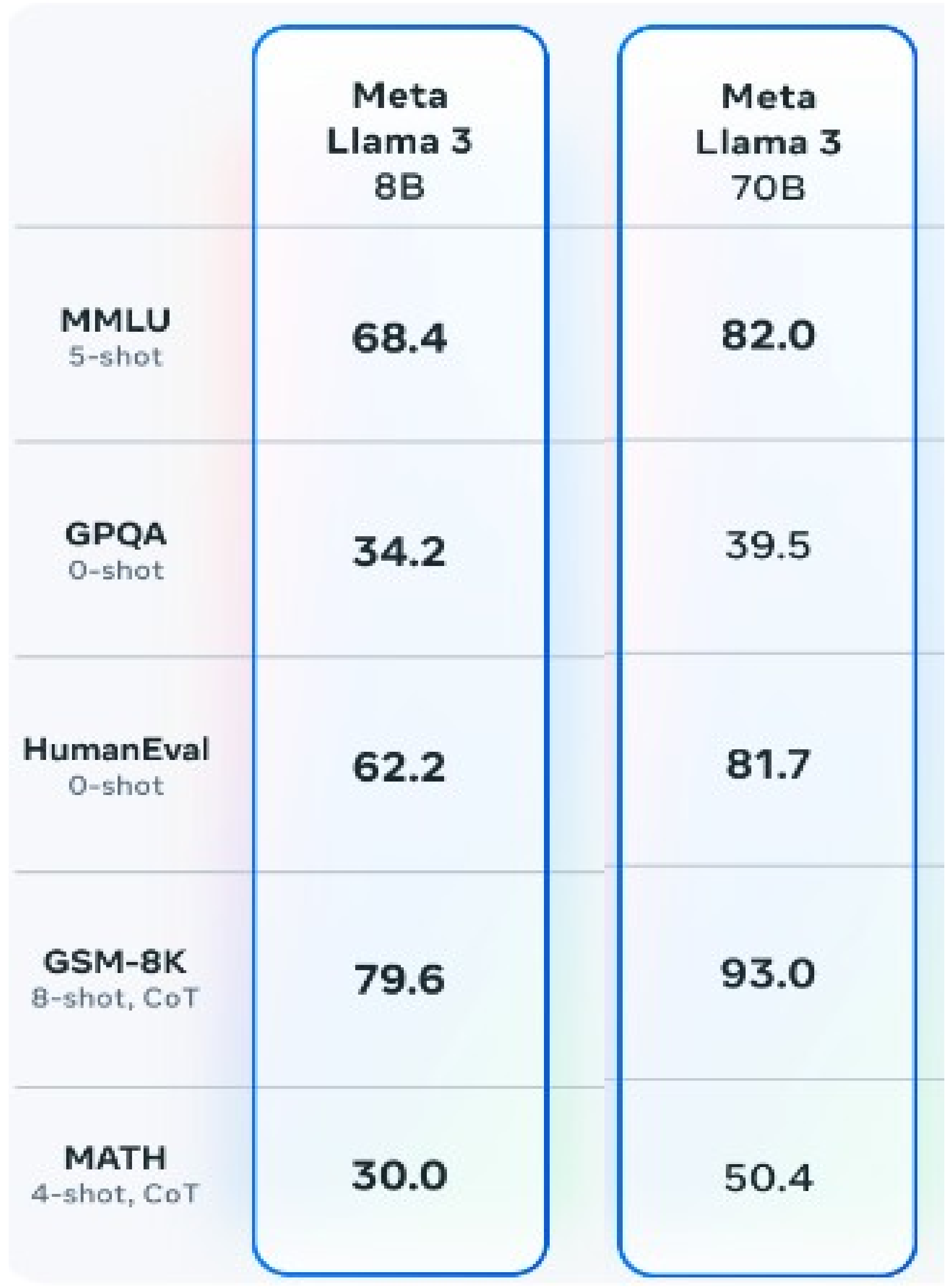
The tradeoff of accuracy and memory footprint

Performance

- Higher memory usage allows faster training and inference by reducing data shuffling and offloading.
- Larger models with more parameters improve performance but require significantly more memory.
- Optimizing for speed (e.g., larger batch sizes, higher parallelism) increases memory demand.

Memory Footprint

- Reducing memory usage often requires model compression techniques (e.g., pruning, quantization), which can lead to reduced accuracy or slower processing.
- Smaller memory footprint allows for deployment on consumer GPUs but might sacrifice performance, particularly for complex tasks.
- For consumers, lower memory often means lower performance compared to enterprise-grade systems.



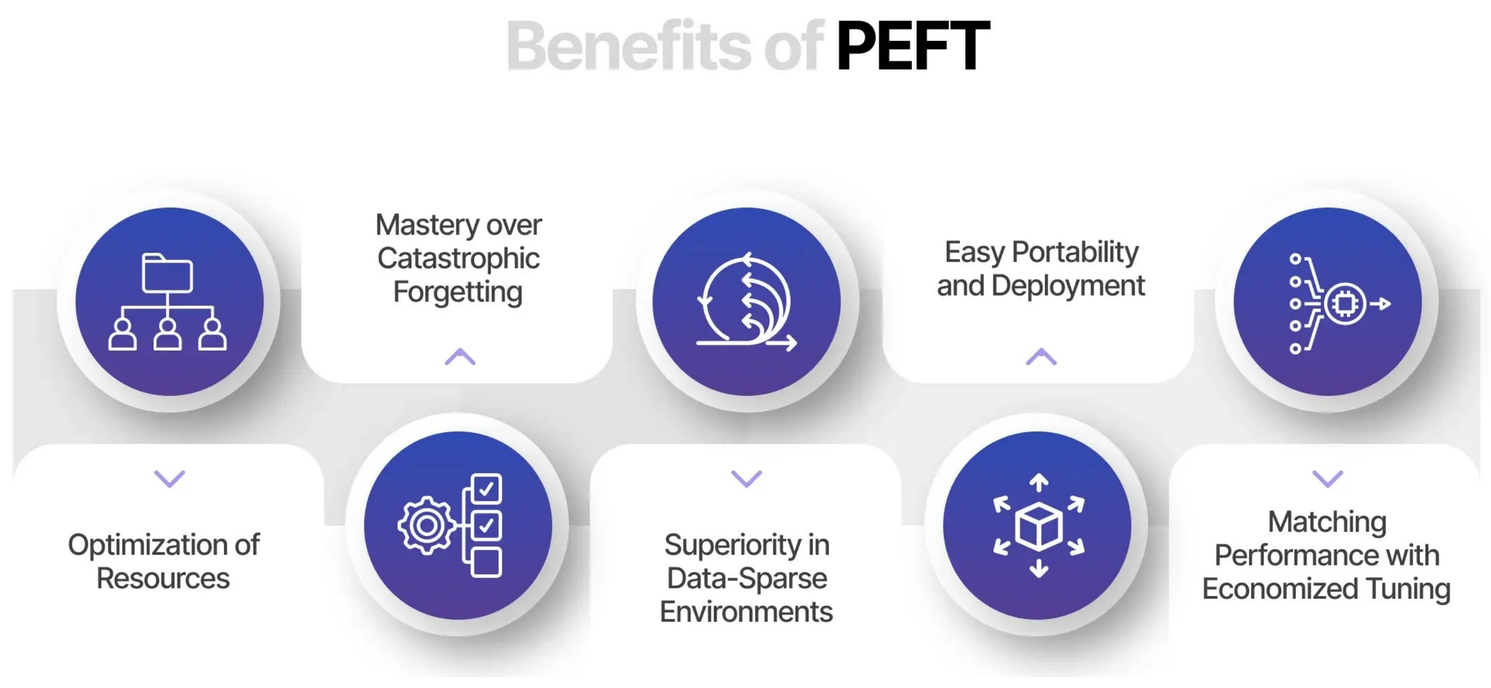
	Meta Llama 3 8B	Meta Llama 3 70B
MMLU 5-shot	68.4	82.0
GPQA 0-shot	34.2	39.5
HumanEval 0-shot	62.2	81.7
GSM-8K 8-shot, CoT	79.6	93.0
MATH 4-shot, CoT	30.0	50.4

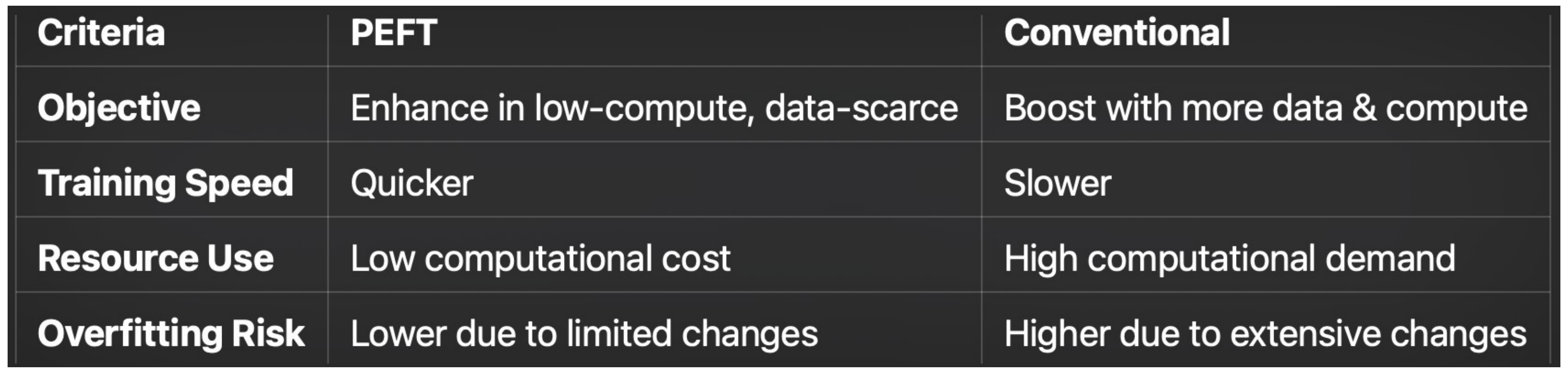
Parameter Efficient Fine-tuning (PEFT)

Fine-tuning large pretrained models is often prohibitively costly due to their scale.

PEFT methods enable efficient adaptation of large pretrained models to various downstream applications by only fine-tuning a small number of (extra) model parameters instead of all the model's parameters.

This significantly decreases the computational and storage costs. Recent state-of-the-art PEFT techniques achieve performance comparable to fully fine-tuned models.


Benefits of PEFT
Mastery over Catastrophic Forgetting
Easy Portability and Deployment
Optimization of Resources
Superiority in Data-Sparse Environments
Matching Performance with Economized Tuning

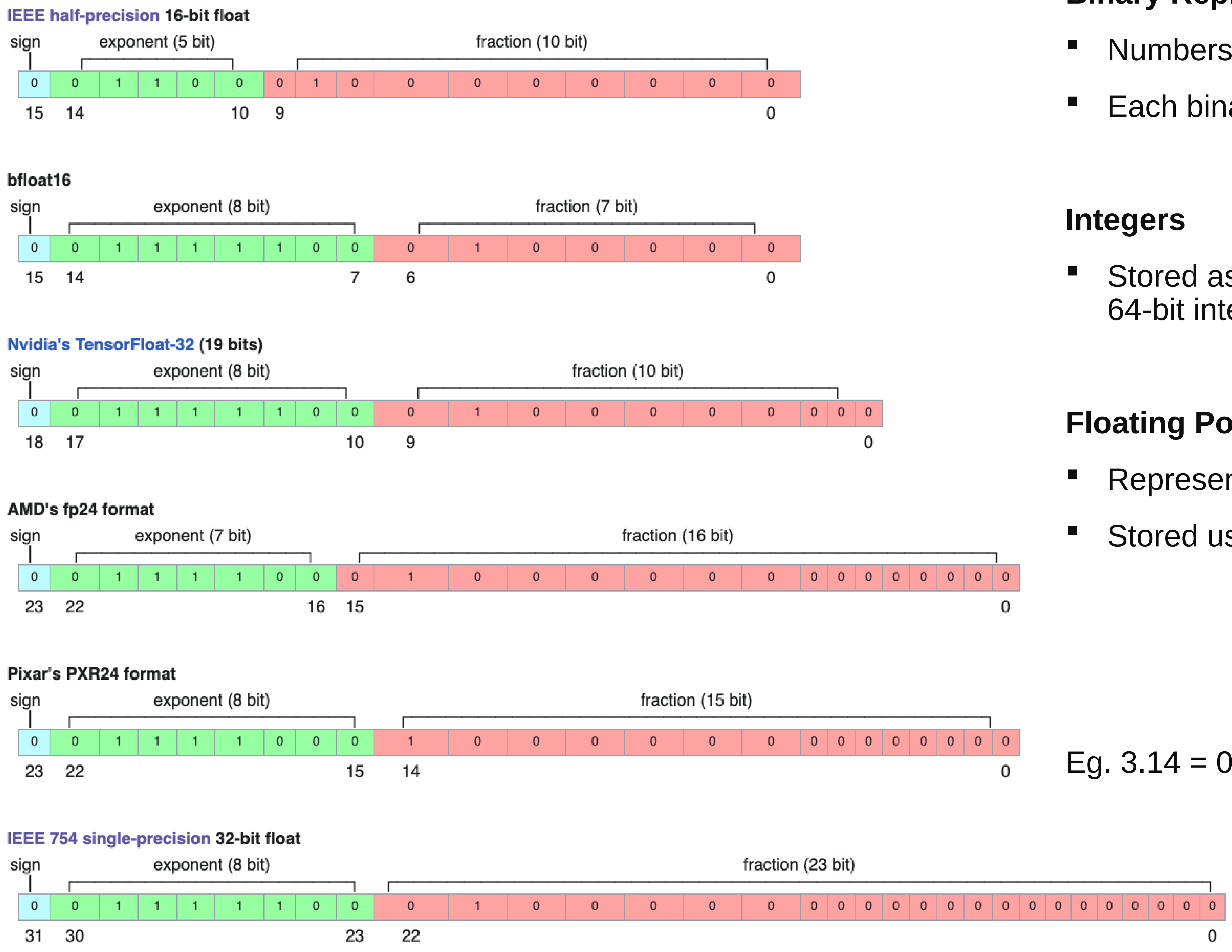


Criteria	PEFT	Conventional
Objective	Enhance in low-compute, data-scarce	Boost with more data & compute
Training Speed	Quicker	Slower
Resource Use	Low computational cost	High computational demand
Overfitting Risk	Lower due to limited changes	Higher due to extensive changes

Minimizing Footprint of Floating Points

Storing numbers

How numbers are stored in a computer



Binary Representation

- Numbers are stored as binary (0s and 1s).
- Each binary digit (bit) is a unit of information.

Integers

- Stored as whole numbers using a fixed number of bits (e.g., 32-bit or 64-bit integers).

Floating Point Numbers

- Represent real numbers (with decimals).
- Stored using scientific notation (mantissa and exponent) in binary.

Eg. 3.14 = 0 10000000 10010001111010111000011 in FP32 format

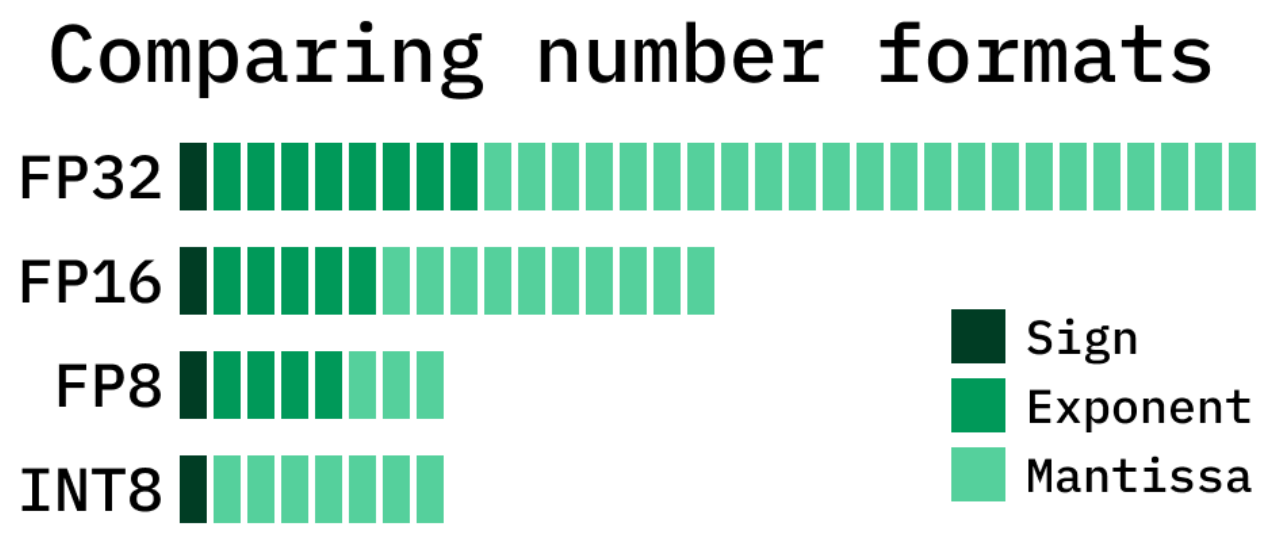
Integers and Floating Point Values

Floating Point Values (FPs):

- The most common formats include Half Precision (FP16), Single Precision (FP32), and Double Precision (FP64), each suited for different computational needs and applications, with trade-offs between memory usage and precision.
- FPs are designed to express a wide range of values, from very large to very small.

Integers:

- While floating-point numbers are essential for handling large and precise values, integers are simpler and often more efficient for discrete tasks.
- Integers are stored exactly without the need for approximations, making them ideal for indexing, counting, and control flows in LLM algorithms.



LLM Variables

M GPU memory required, in **Gigabytes**. This is the total memory needed to load and compute the model on a GPU.

P The total **number of parameters** in the model, which determines its overall size and memory footprint.

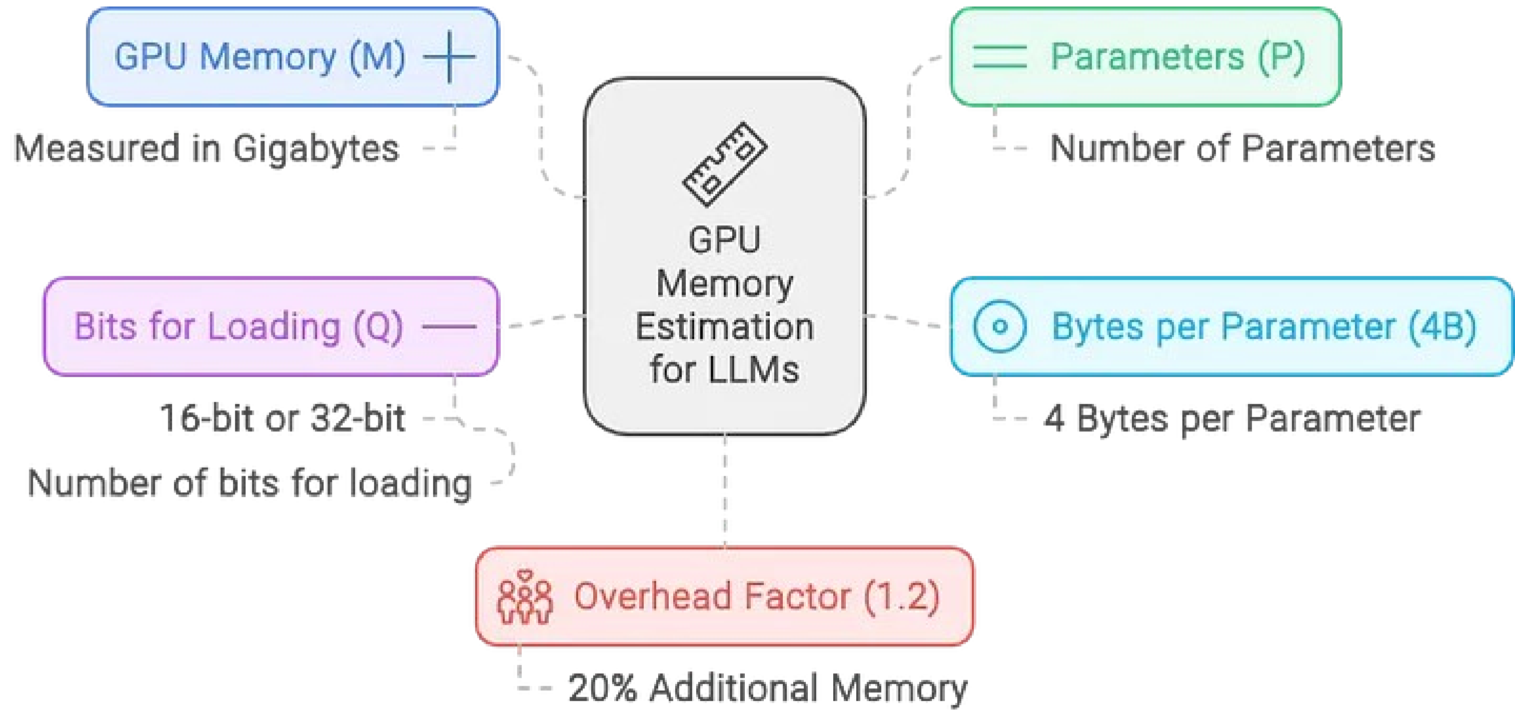
4B 4 bytes per parameter. This represents the storage size for each parameter in **FP32** (32-bit floating point), typically used for higher precision.

32 There are **32 bits** in 4 bytes, which defines the full precision of FP32, the standard used in most floating-point calculations.

Q The number of **bits per parameter** used for loading the model (e.g., 16 bits for FP16, 8 bits, or 4 bits), which reduces memory usage at lower precision.

1.2 A **20% overhead** factor, accounting for extra memory needed when loading the model into the GPU, such as additional metadata or optimization layers.

$$M = \left(\frac{P \times 4B}{32/Q}\right) \times 1.2$$



Approximation with Quantization

Quantization is the process of reducing the precision of a digital signal, typically from a higher-precision format to a lower-precision format.

This technique is widely used in various fields, including signal processing, data compression and machine learning.

Quantization Algorithm Steps:

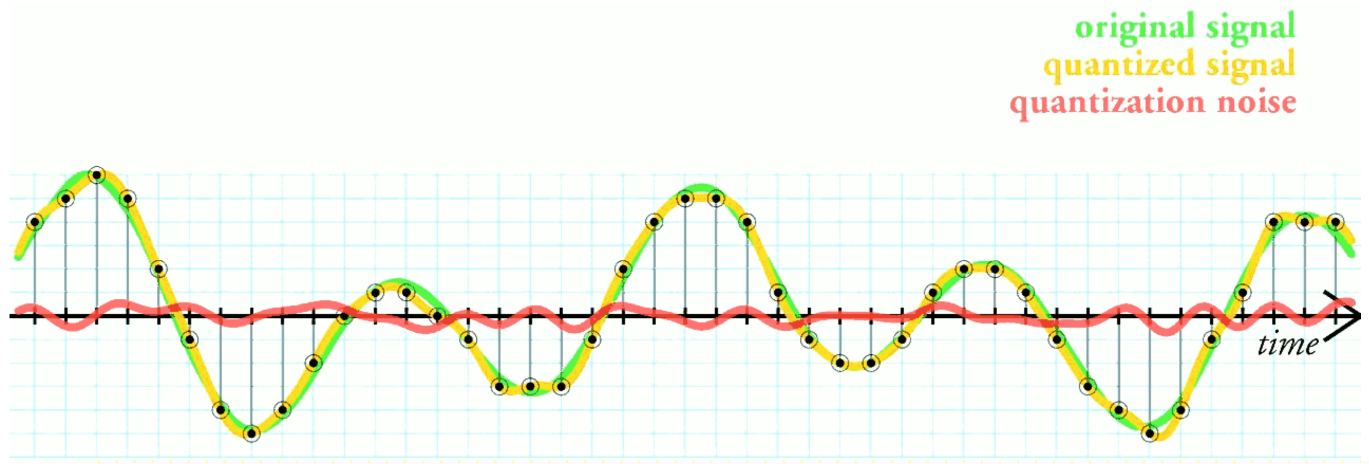
1. Determine Range:
Identify the range of floating-point values (e.g., weights from -1.5 to +1.5).

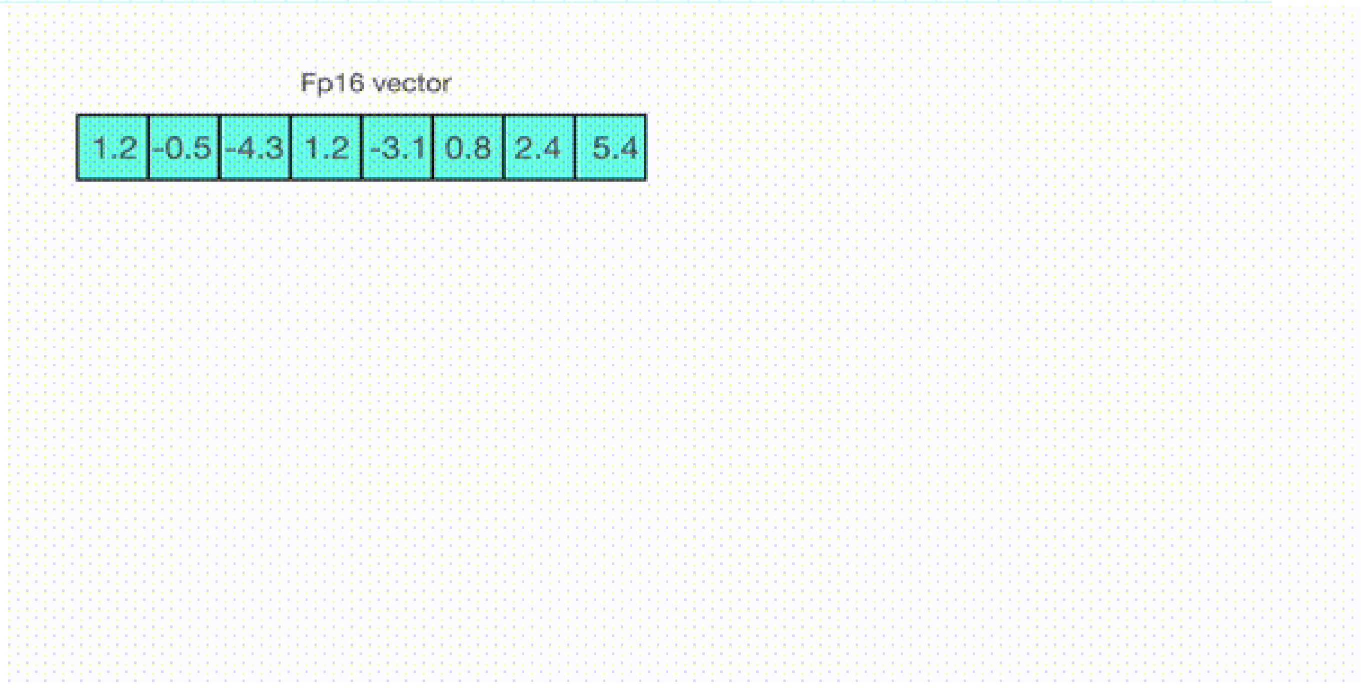
2. Select Format:
Choose a lower precision format (e.g., INT8 or FP16) for quantization.

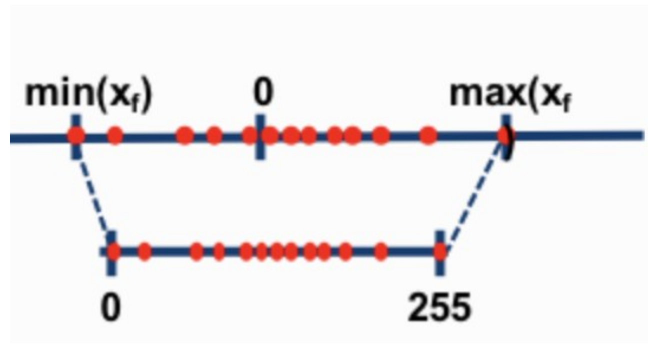
3. Calculate Scaling Factor:
Compute the scaling factor to map the original values to the integer range (e.g., -128 to 127 for INT8).

4. Quantize:
Scale and round each value to the nearest integer using the scaling factor.

5. Dequantize (for computations):
Convert the quantized values back to floating point when necessary by multiplying with the scaling factor.







Pros and Cons of LLMs with Quantization

Pros of Quantization

- **Smaller Models**:

Reduced weight size enables deployment on less powerful hardware and lowers storage costs.

- **Scalability**:

Smaller memory footprint makes models easier to scale across various infrastructures.

- **Faster Inference**:

Lower bit-widths improve computational efficiency and speed.

- **Power Efficiency**:

Reduces energy usage, beneficial for mobile and edge devices.

- **Hardware Optimization**:

Modern GPUs support low-precision operations (e.g., INT8), enhancing performance.

Cons of Quantization

- **Accuracy Loss**:

Lower precision can degrade model performance, especially with aggressive quantization (e.g., 4-bit).

- **Training Complexity**:

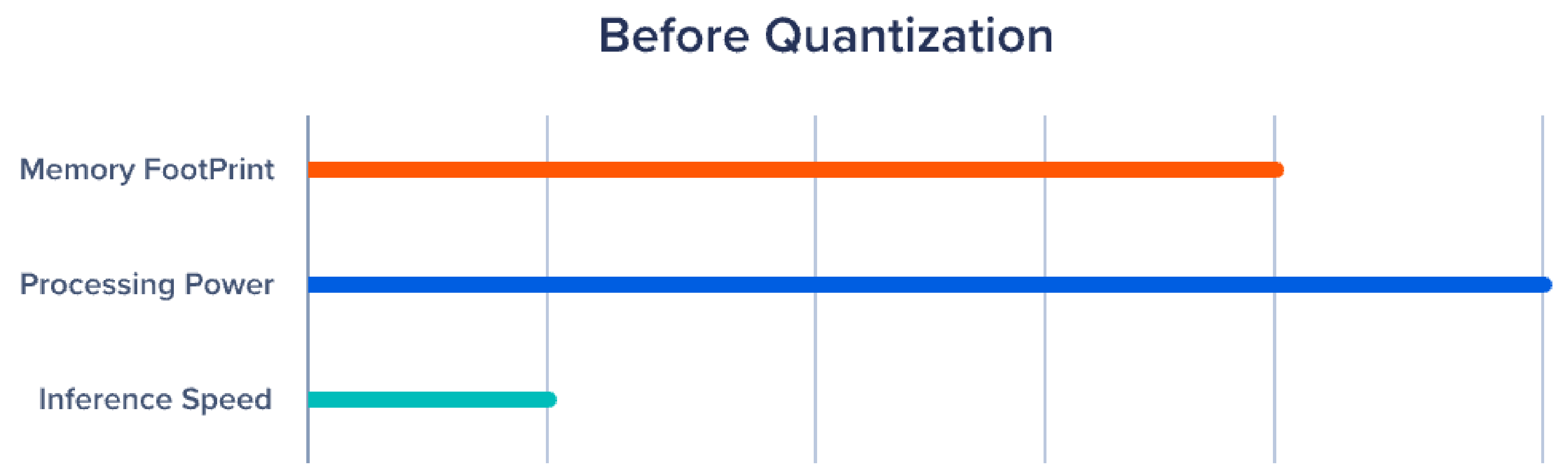
Quantization-aware training adds complexity to the model development process.

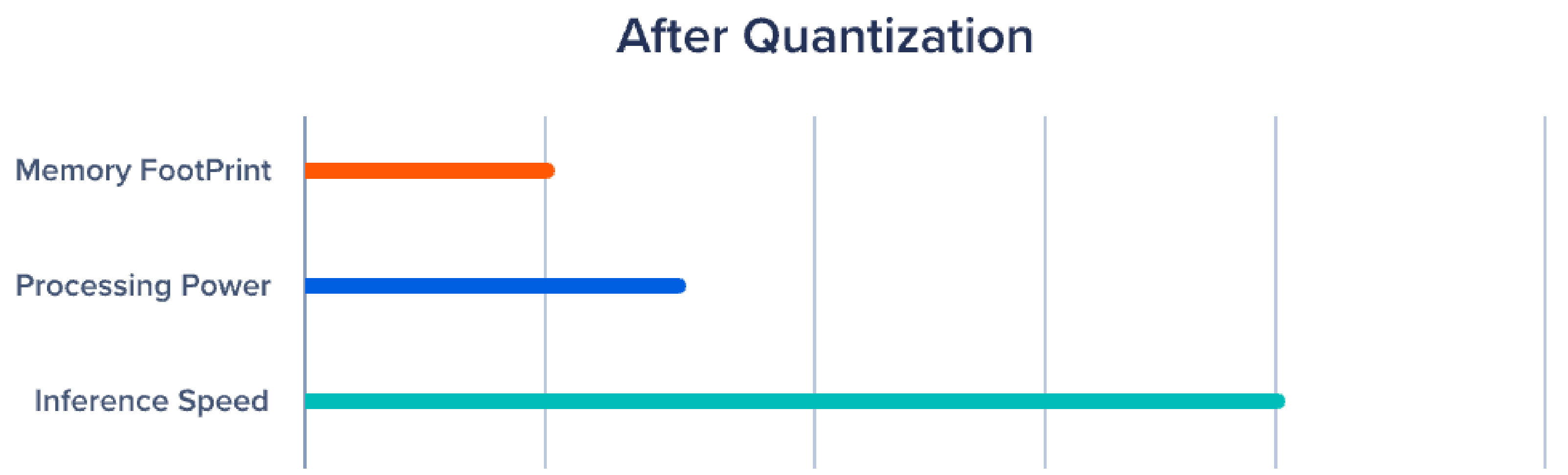
- **Limited Use Cases**:

Not all models/tasks work well with reduced precision, especially high-precision tasks.

- **Hardware Compatibility**:

Older hardware may not fully support low-precision operations.





Distilling information with Teaching

Learning from the teacher

Simple Data Savings - Pruning

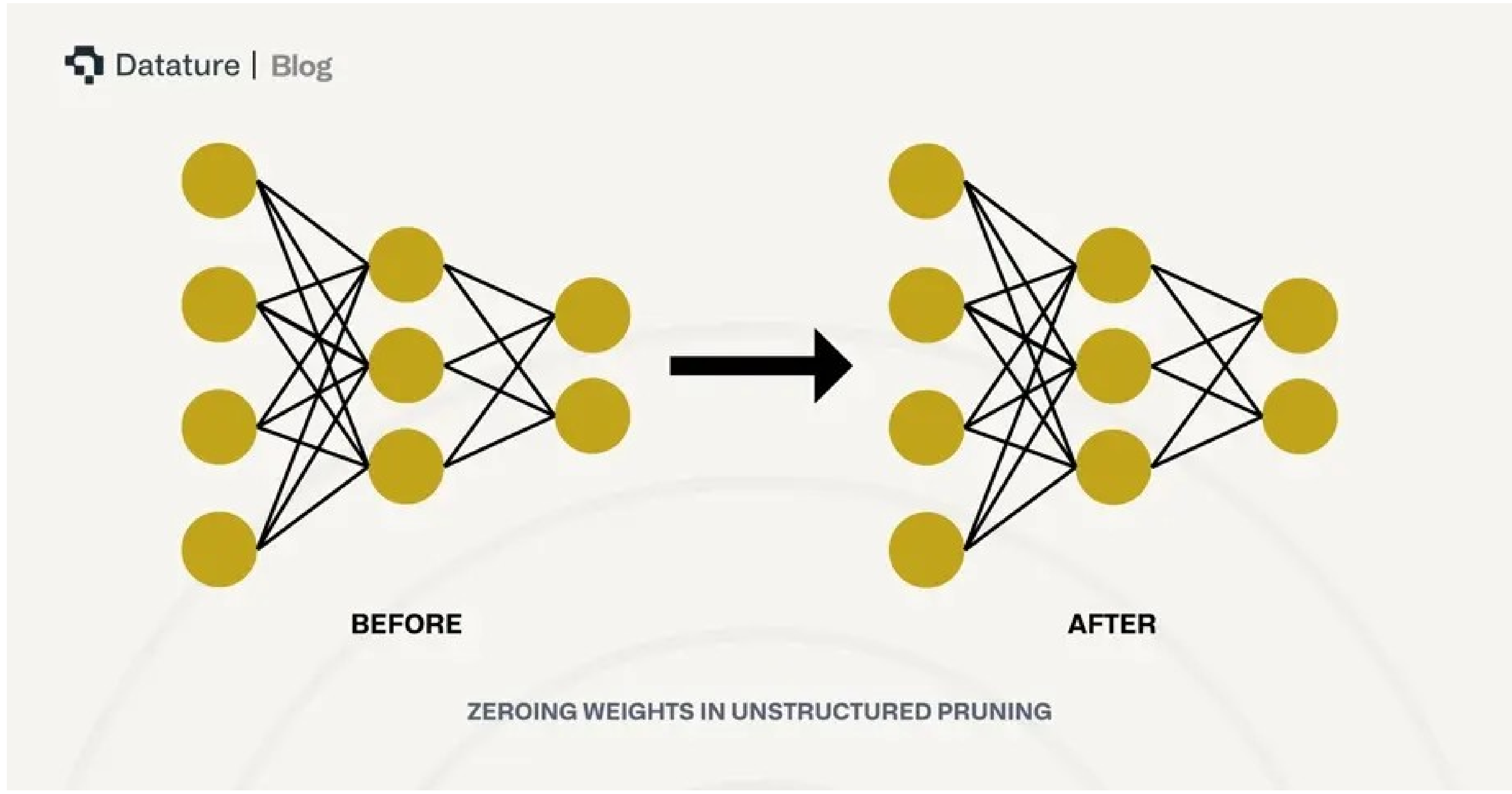
Pruning is the process of reducing the size of a neural network by removing unnecessary or less important parameters, improving efficiency without significantly impacting performance.

Train-Time Pruning
Pros:
Models are trained with sparsity in mind, leading to more efficient models.
Pruning decisions are made during training, optimizing model parameters along the way.

Cons:
Adds complexity to training.
Pruning parameter changes may require retraining the entire model.

Post-Training Pruning
Pros:
Simpler to implement after model training.
Pruning parameters can be adjusted for different inference requirements.

Cons:
May require fine-tuning to restore performance if accuracy degrades.
Pruning decisions are user-defined and may not be optimal.

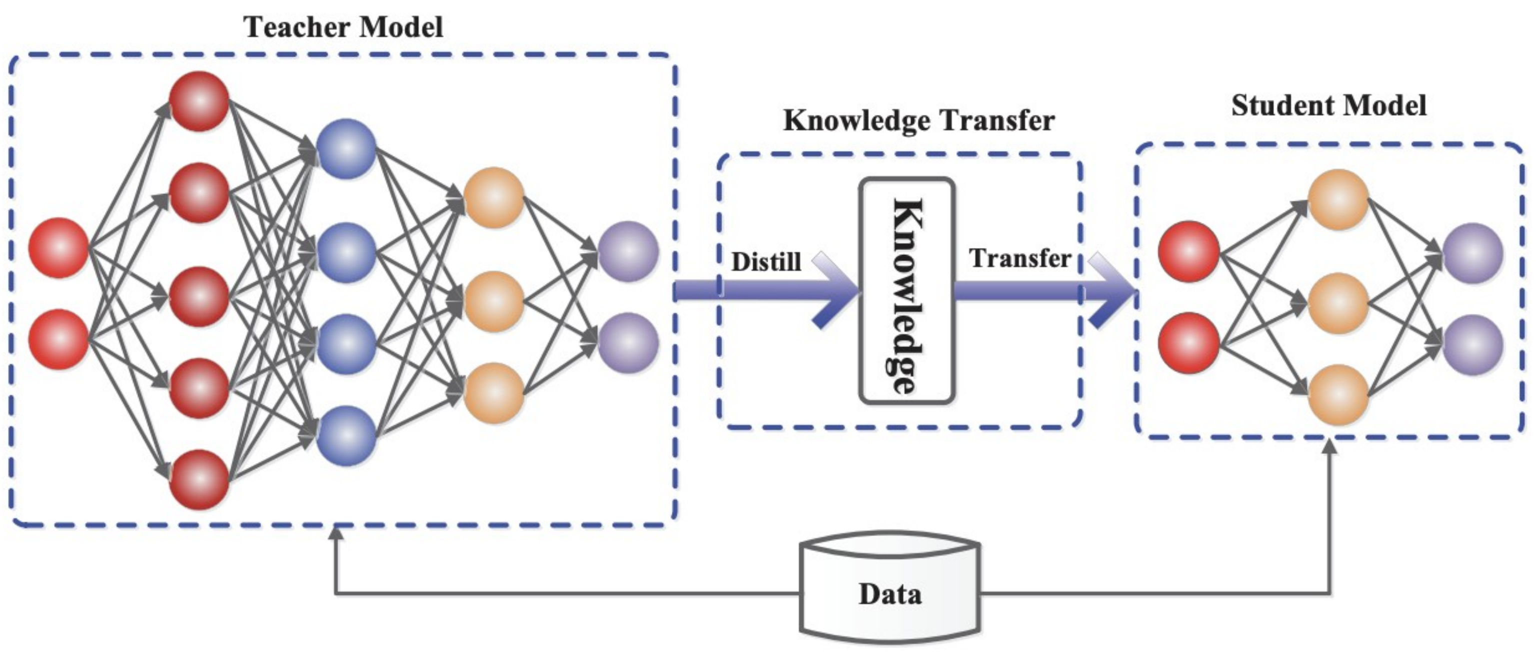


Trade-offs
Pruning can reduce model size and inference time, making models more scalable and deployable, especially on resource-limited devices. However, it can introduce challenges in training complexity or accuracy loss, depending on the pruning approach.

Distillation a more complex approach

Knowledge distillation is a technique where a smaller, simpler model (the “student”) is trained to replicate the behavior of a larger, more complex model (the “teacher”).

How It Works:
1. Teacher Model:
A large, pre-trained model provides predictions, usually with high accuracy but requires substantial computational resources.
2. Student Model:
A smaller, lightweight model learns to mimic the teacher’s behavior by matching its predictions, allowing faster inference with reduced complexity.



Why Use Knowledge Distillation?
Model Compression:
The student model is much smaller, reducing storage and memory requirements.
Faster Inference:
Smaller models run faster and are more suited for real-time or resource-constrained environments.
Deployment Efficiency:
Student models are ideal for mobile devices, edge computing, and cloud applications due to their lightweight nature.

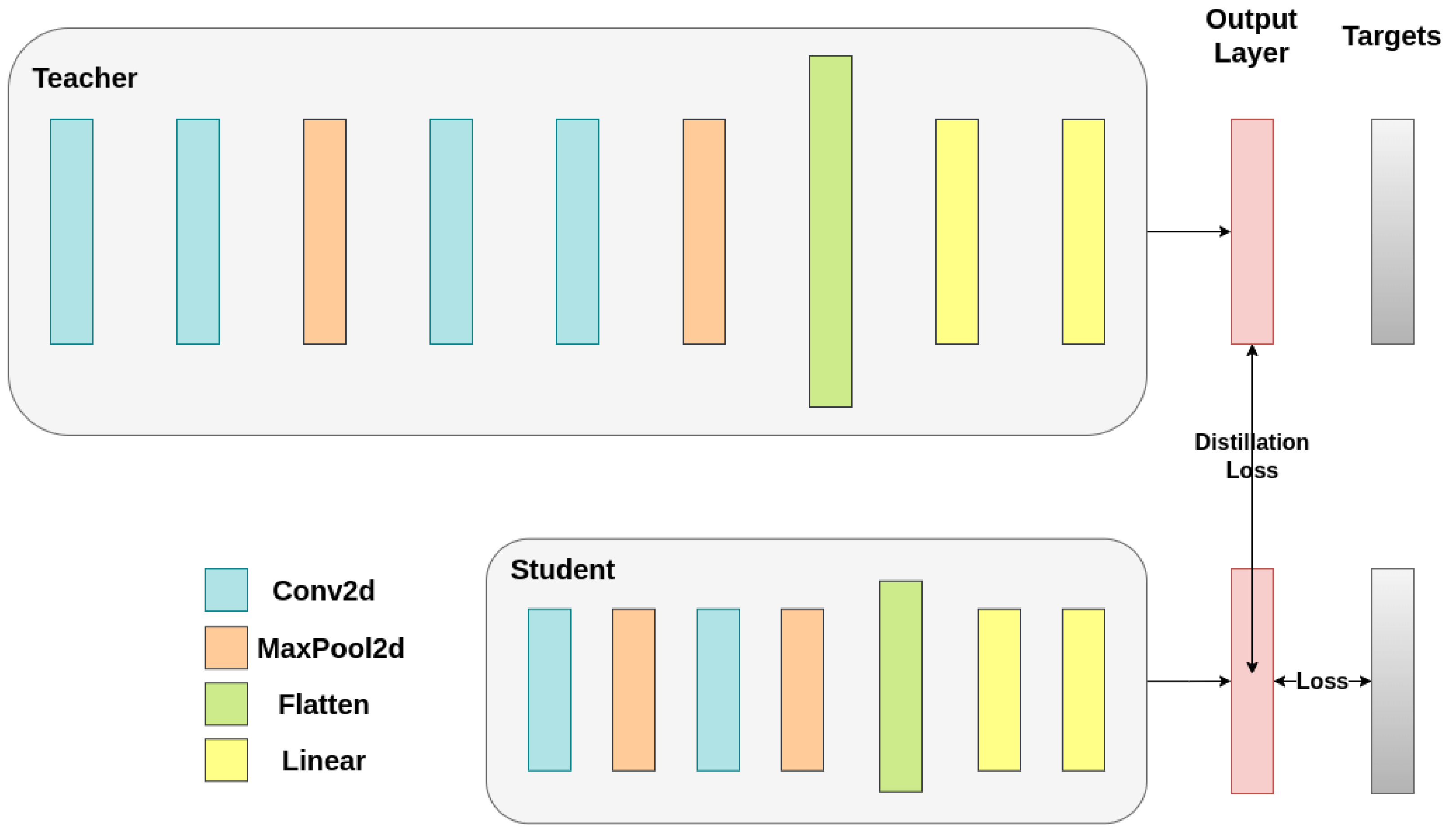
Teaching a smaller model with soft targets

Soft Targets in Knowledge Distillation:
Soft targets are the probabilities output by the teacher model for each class in a classification task, as opposed to hard targets, which are binary labels (e.g., "cat" or "dog"). Instead of just telling the student model the correct class, the teacher provides a probability distribution over all classes, giving the student more nuanced information about the teacher's knowledge.

For example, instead of saying "90% cat, 10% dog," soft targets might look like: [0.70 cat, 0.25 dog, 0.05 rabbit]

This extra information helps the student model understand **class relationships** and **relative confidences** that aren't reflected in hard labels. The **temperature** parameter is used to control how smooth the soft targets are, with higher values producing softer probability distributions.

Explanation:
1. **Teacher Model**: Provides logits (pre-softmax outputs), which are softened using the temperature.
2. **Student Model**: Produces its logits and softens them using the same temperature.
3. **Distillation Loss**: KL divergence between teacher's soft targets and student's predictions.
4. **Supervised Loss**: Cross-entropy loss between student's predictions and hard labels.
5. **Final Loss**: Combines distillation and supervised loss to train the student.
This pseudo code outlines the key steps in knowledge distillation, particularly focusing on how soft targets and the distillation process work.



Pros and Cons of Distilled Models

Pros of Distillation:

Model Compression:
Reduces model size while retaining most of the teacher's accuracy.
Faster Inference:
Smaller models have lower latency, making them ideal for deployment on limited hardware.
Energy Efficient:
Consumes less power, important for mobile and edge devices.

Cons of Distillation:

Potential Accuracy Loss:
The student model may not achieve the same performance as the teacher, especially for more complex tasks.

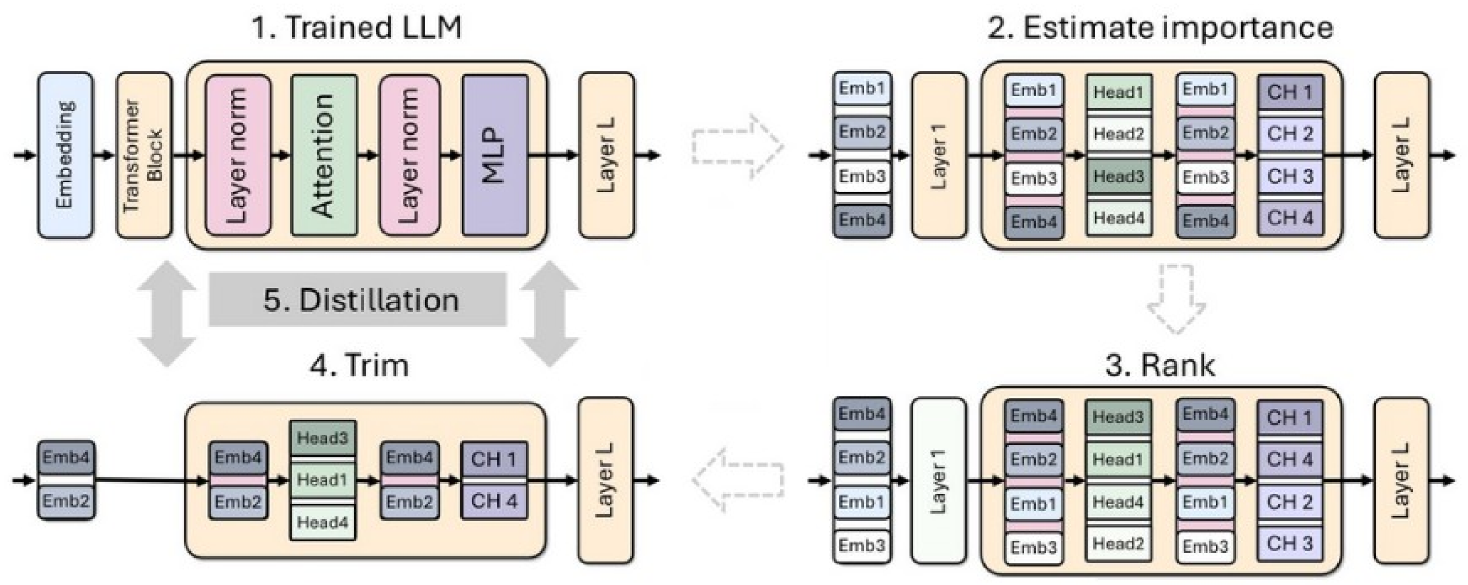
Combing methods - Pruning + Distillation for Efficient LLMs

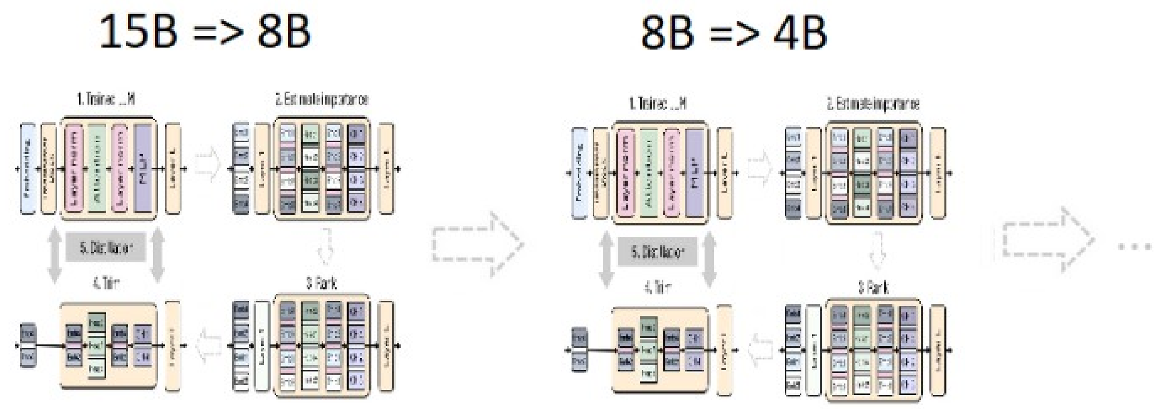
Why Combine?
Pruning reduces model size by removing unnecessary layers or neurons, while distillation transfers knowledge from a larger teacher model to a smaller student model. Together, they create smaller, faster models with minimal performance loss.

Steps to Compress Llama 3.1:
Pruning:
Depth Pruning: Remove entire layers (e.g., 50% of layers in Llama 3.1 8B → 4B).
Width Pruning: Trim neurons, attention heads, and embedding dimensions.

Knowledge Distillation:
The pruned model (student) is retrained using soft targets from the original model (teacher) to retain performance.

Benefits:
16% improvement in MMLU scores vs. training from scratch.
40x reduction in training tokens needed for smaller models.
1.8x compute cost savings across model families.
Comparable performance to larger models (e.g., Mistral 7B, Gemma 7B).
This provides a clean, high-level explanation ideal for a slide presentation.





Linear Algebra and Minimal Updates

What we can use in mathematics to fuel PEFT

Fine-tuning in the frame of Linear Algebra

What is Fine-Tuning?
Fine-tuning is the process of adjusting a pre-trained model's weights to improve performance on a specific task.

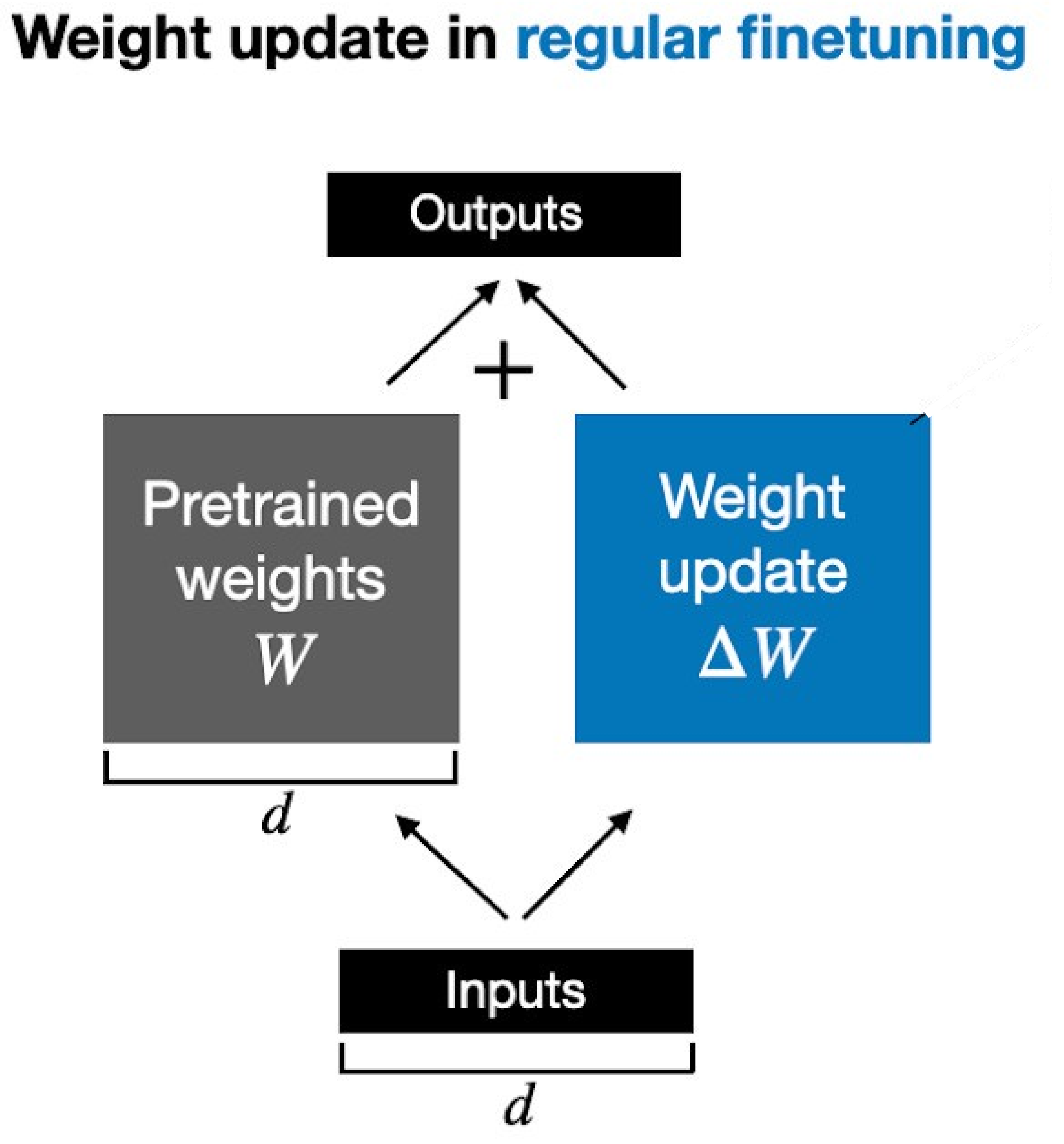
Linear Algebra in Fine-Tuning:

Matrix Operations:
Weights in neural networks are represented as matrices.
Fine-tuning adjusts these matrices through matrix multiplication and linear transformations during backpropagation.

Eigenvalues & Eigenvectors:
Fine-tuning explores the model's parameter space, where understanding eigenvalues helps in adjusting directions that significantly influence model performance.

Matrices in LLMs can be massive, containing millions of parameters each, resulting in a very computationally expensive operation when performing fine-tuning.

We can view fine-tuning as taking the original weights and adding a new matrix to it, representing the changes.



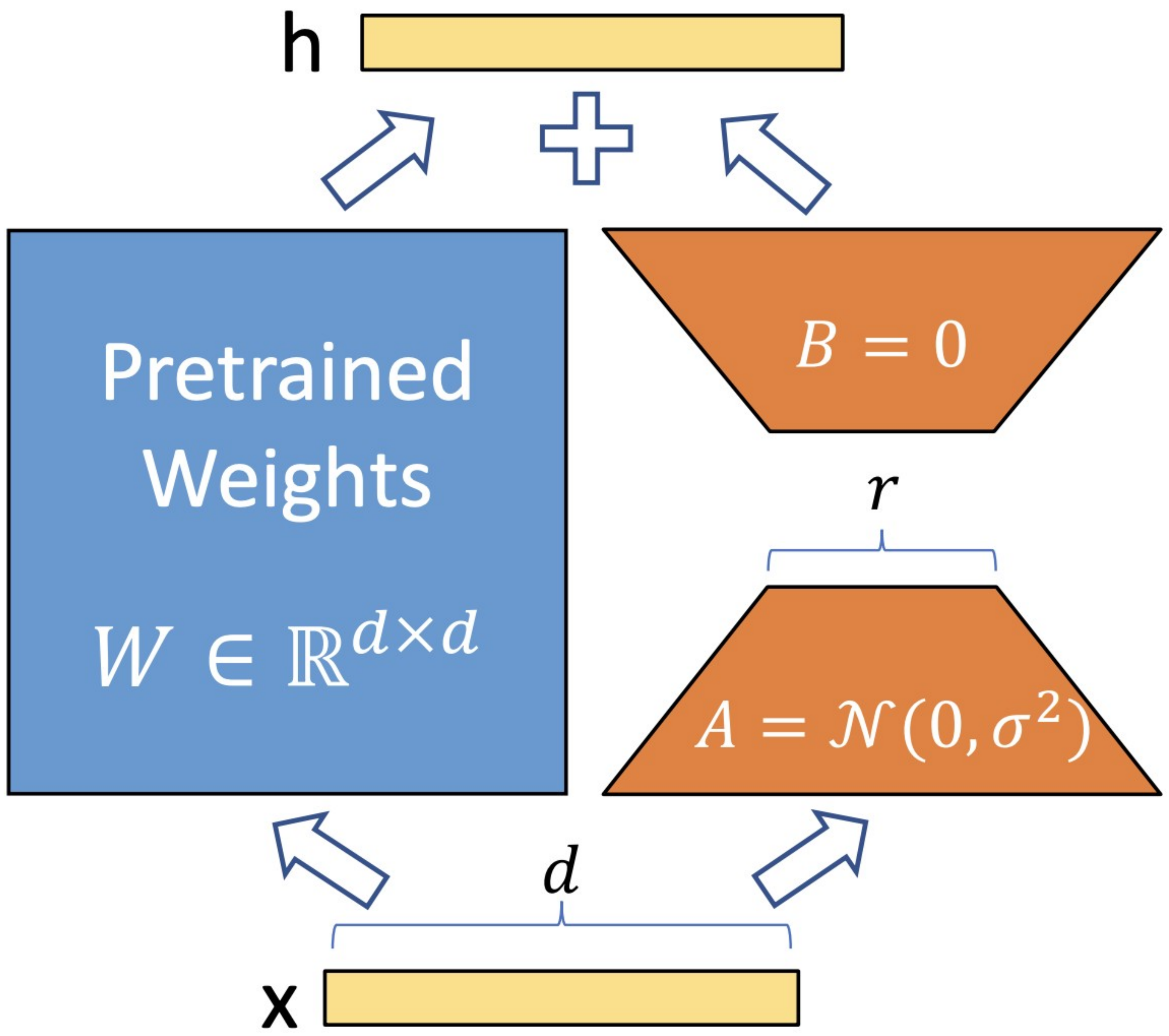
Low **R**ank **A**dapters - LoRA

LoRA is a technique for fine-tuning large language models (LLMs) by injecting low-rank matrices into the model's architecture. It allows efficient adaptation of models without retraining all weights.

Key Concepts:
Low-Rank Matrices:
LoRA introduces low-rank decomposition to the model's weight matrices, significantly reducing the number of parameters needed to be fine-tuned.

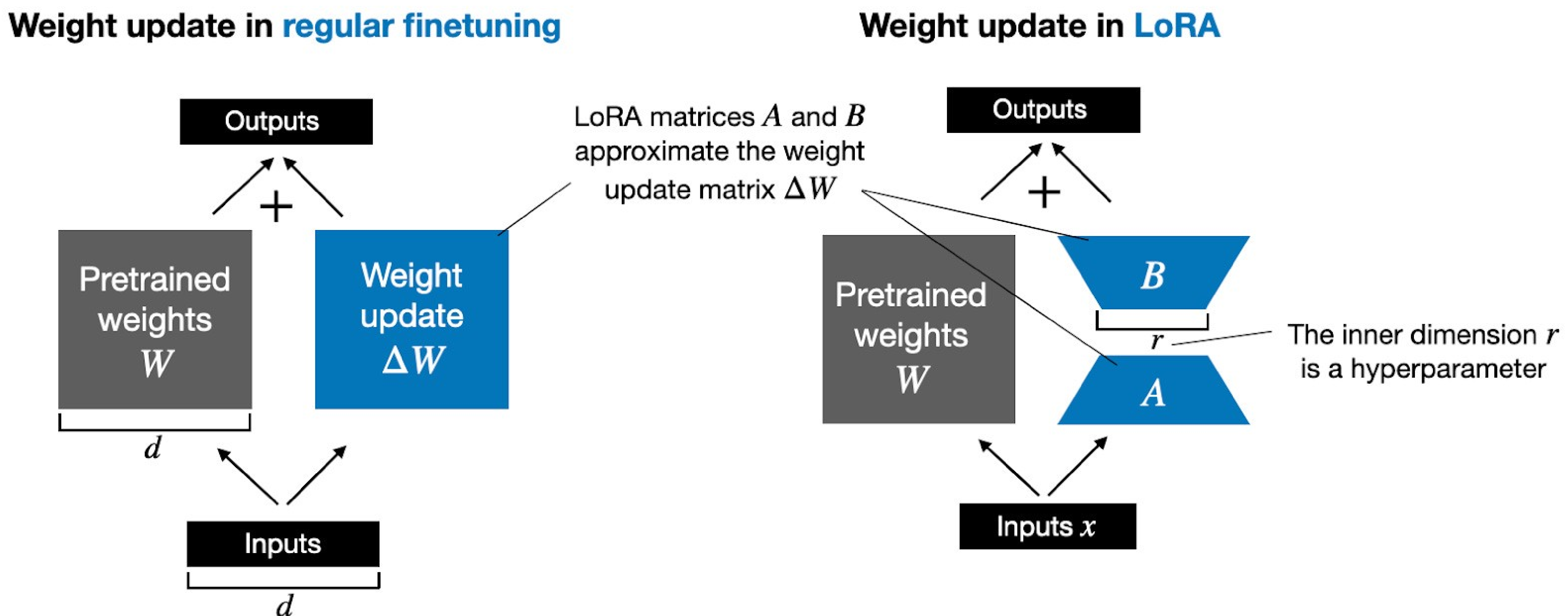
Parameter Efficiency:
Only a small fraction of the original model's parameters are updated, making fine-tuning faster and more memory-efficient.

No Full Model Retraining:
Instead of updating all parameters, LoRA focuses on modifying just the low-rank adaptation matrices, keeping the rest of the model frozen.


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Fine-tuning vs. LoRA

	Fine-Tuning	LoRA (Low-Rank Adaptation)
Parameter Updates	Updates all model parameters, requiring large-scale weight adjustments.	Updates only small, low-rank matrices, keeping most of the model frozen.
Memory Usage	High memory usage, adjusting all parameters consumes significant resources.	Low memory footprint, since only a subset of parameters is updated.
Training Speed	Slower, as all parameters need to be optimized, taking more time and resources.	Faster, as fewer parameters are updated, leading to quicker fine-tuning.
Compute Resources	Requires substantial compute power, often with multiple GPUs.	Requires fewer compute resources, ideal for faster adaptation and resource-limited environments.

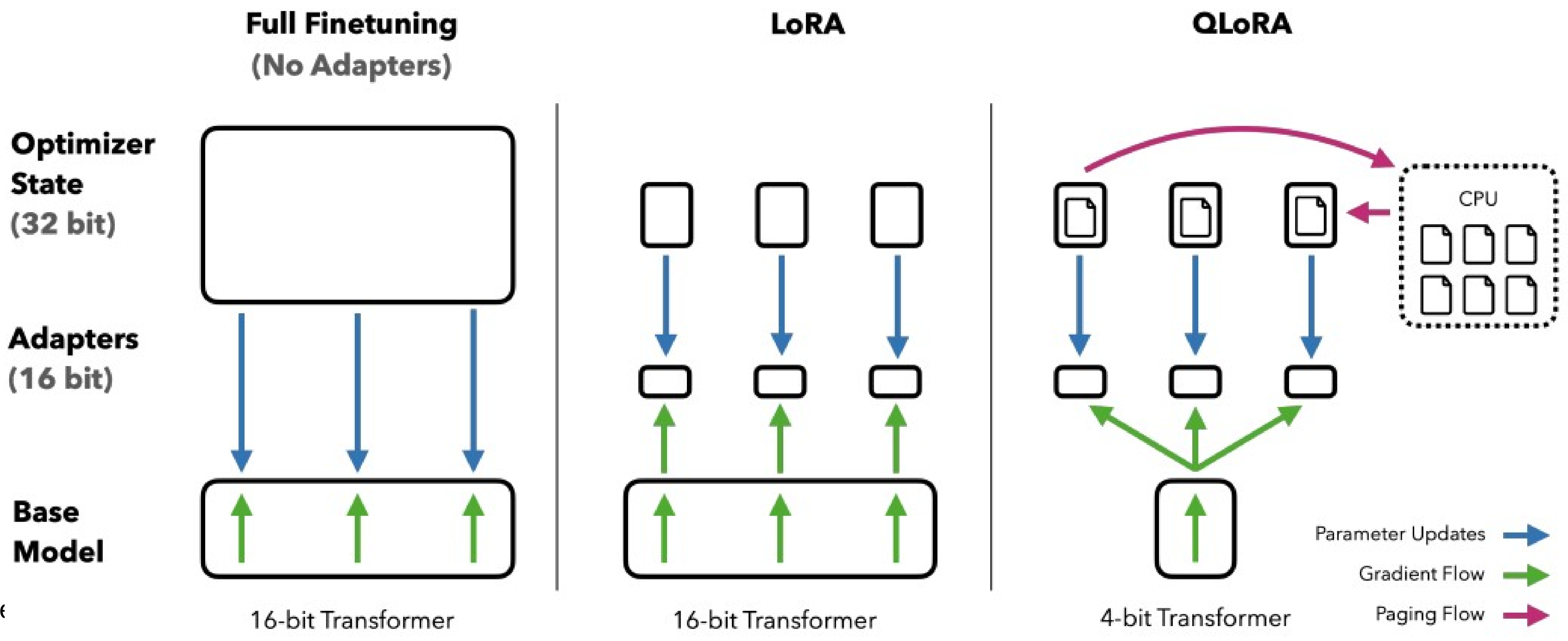

Weight update in regular finetuning
Weight update in LoRA
Outputs
+
Pretrained weights W
Weight update ΔW
d
Inputs
d
LoRA matrices A and B approximate the weight update matrix ΔW
Outputs
+
Pretrained weights W
B
r
A
The inner dimension r is a hyperparameter
Inputs x

Quantization of Low Rank Adapters - QLoRA

QLoRA is LoRA with quantized linear layers in the base model. The adapters are identical to those of LoRA and kept in higher precision (BF16) during QLoRA training.

Compared to LoRA, QLoRA is:

- Up to 60% more memory-efficient, allowing for fine-tuning large models with smaller/less GPUs and/or higher batch size.
- Able to achieve the same accuracy, although a different convergence recipe is required.
- Between 50% and 200% slower than LoRA.

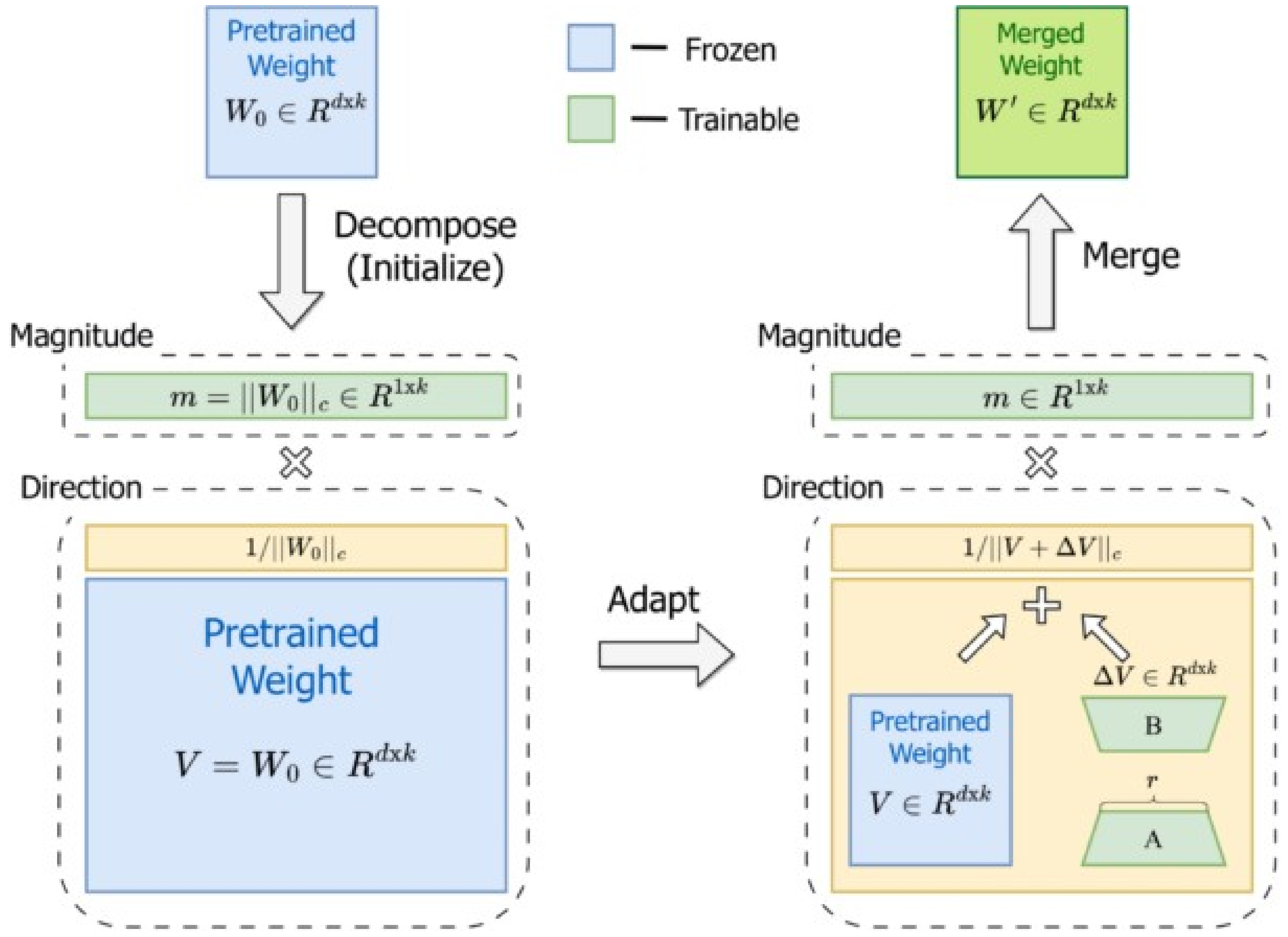


This can make QLoRA a viable choice in compute constrained environments

Alternatives to LoRA: DoRA

Weight-Decomposed Low-Rank Adaptation **(DoRA)**, is an enhanced alternative to LoRA.

- DoRA improves learning capacity and model stability, while maintaining the same inference speed, avoiding any additional overhead during inference.
- DoRA operates by decomposing the pre-trained model's weights into **magnitude** and **directional** components, and fine-tuning both.
- By leveraging the relatively small size of the directional component, DoRA enhances LoRA's efficiency in fine-tuning. Importantly, DoRA can merge with the original pre-trained weights before inference, ensuring no additional latency is introduced.
- DoRA builds on the success of LoRA by focusing on decomposing the weight matrices, targeting only the most important directional shifts. This approach reduces the computational complexity further without sacrificing the fine-tuning accuracy.

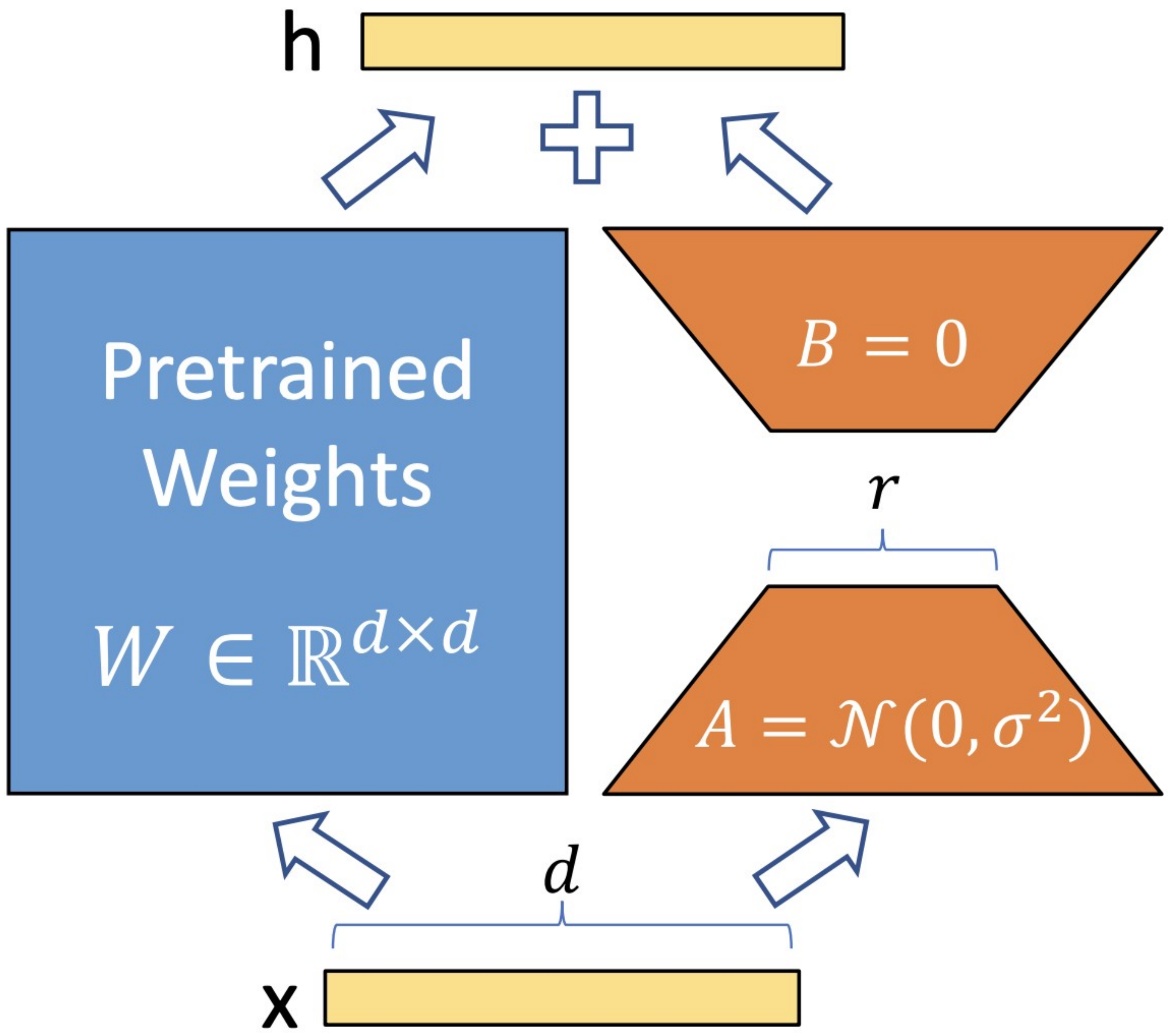


Wrap Up

Parameter-Efficient Fine-Tuning (PEFT)

- Today we covered some of the popular PEFT methods
- We motivated PEFT as a means for fine tuning and LLM interactions where compute resources are constrained
- A review of general methods such as quantization and pruning of neural networks was presented
- The Low-Rank Adaptors approach, LoRA, was presented as a means of more efficient fine-tuning of LLMs, exploiting the matrices that are involved in LLM fine tuning
- Quantized versions, and alternatives to LoRA, QLoRA and DoRA were also introduced as other areas to explore as PEFT methods gain popularity in the community and industry

----



NVIDIA

DARTMOUTH ENGINEERING

1867

Thank you!