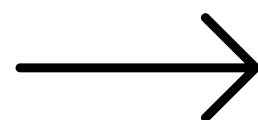


# **PEFT and Family of LoRA Techniques**

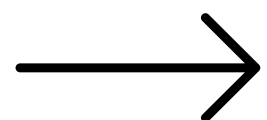
Swipe to next slide



# What is PEFT

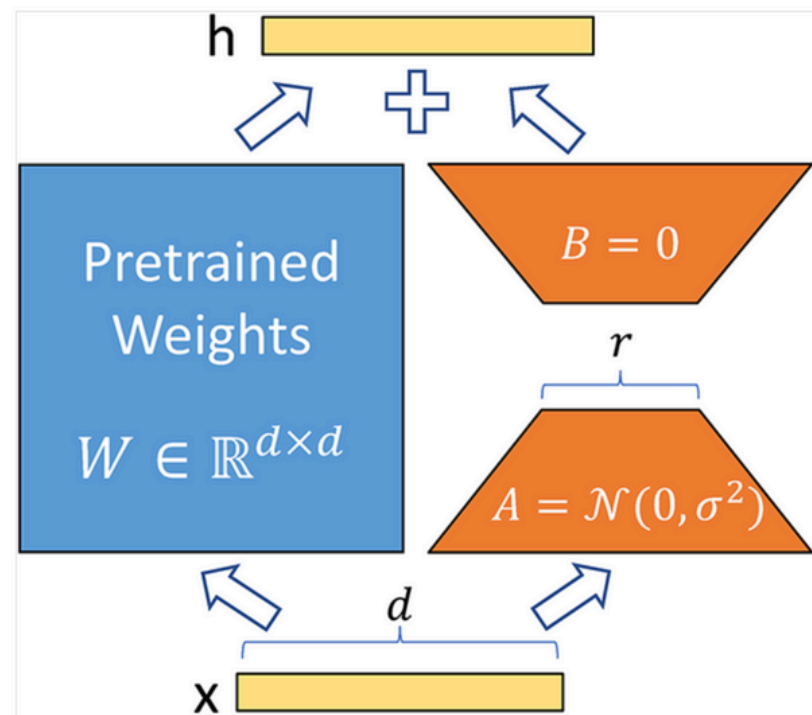
Parameter-Efficient Fine-Tuning (PEFT) enables fine-tuning of LLMs for specific tasks, significantly reducing both computational overhead and storage demands.

Swipe to next slide



# LoRA

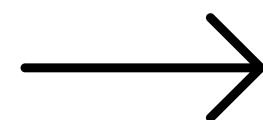
## Low Rank Adaptation



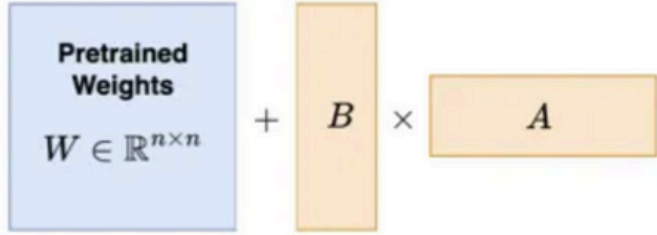
Source: Fine-Tuning LLMs: LoRA or Full-Parameter? An in-depth Analysis with Llama 2

LoRA is a fine-tuning technique for quickly adapting LLMs to specific domains. It adds lightweight pieces to the original model, as opposed to changing the entire model.

Swipe to next slide



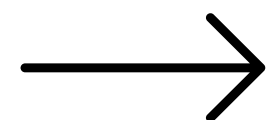
# LoRA +

	LoRA	LoRA+
Parameterization		
Training	$A \leftarrow A - \eta \times G_A$ $B \leftarrow B - \eta \times G_B$	$A \leftarrow A - \eta \times G_A$ $B \leftarrow B - \lambda \eta \times G_B$ $\lambda \gg 1$

LoRA+ introduces different learning rates for the two matrices A and B, here indicated by the parameter  $\lambda$ .  
Image from [2].

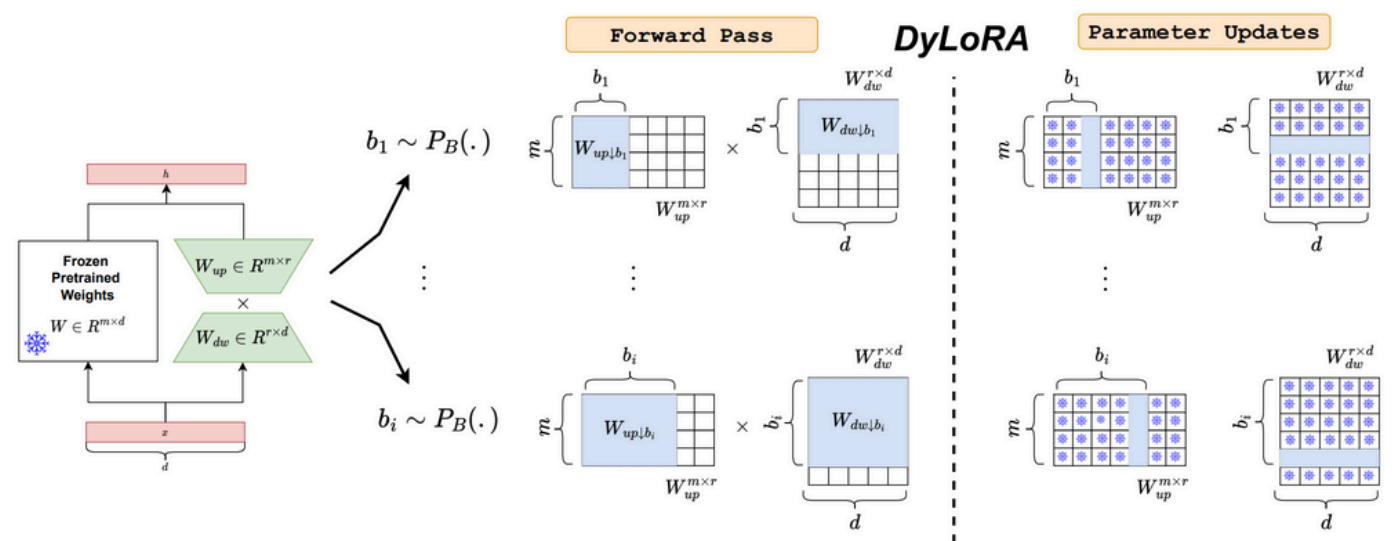
LoRA+ introduces a more efficient way of training LoRA adapters by introducing different learning rates for matrices A and B. LoRA+ improves the performance just a little but speeds up the finetuning process, at the same computational cost as LoRA.

Swipe to next slide



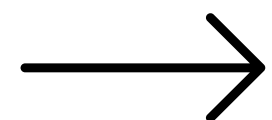
# DyLoRA

## Dynamic Low Rank Adaptation



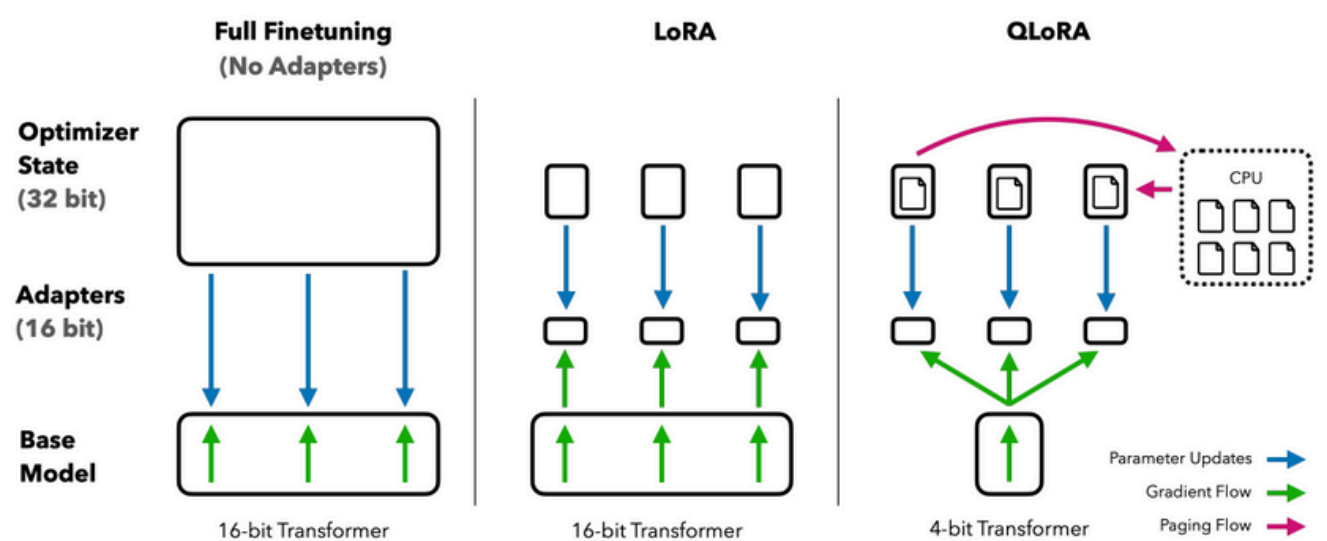
DyLoRA allows for the dynamic adaptation of Low ranks adapters during training. This means that instead of being limited to a single rank, DyLoRA can optimize performance across a range of ranks without requiring multiple re-training sessions

Swipe to next slide



# QLoRA

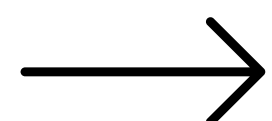
## Quantized Low Rank Adaptation



**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.

QLoRA is an extension of LoRA method which further reduces the memory footprint of fine-tuning by quantizing the low-rank matrices used in LoRA. It significantly reduces the memory usage making it possible to fine-tune LLMs on a single GPU.

Swipe to next slide



# LoRA-drop

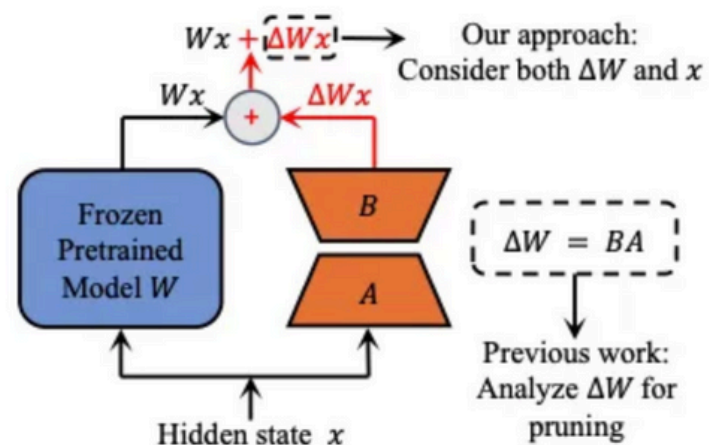
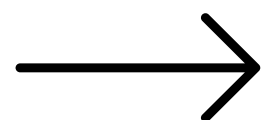


Figure 1: The diagram of LoRA

LoRA-drop uses the output of  $B \cdot A$  to decide, which LoRA-layers are worth to be trained at all. Image from [5].

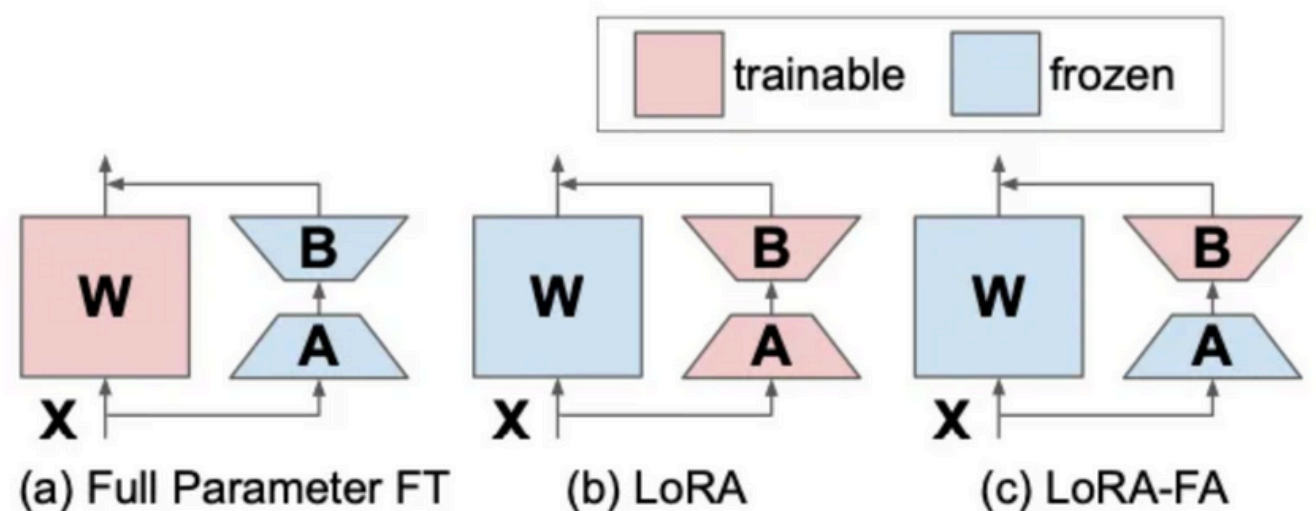
LoRA-drop optimizes which model layers should use LoRA adapters to reduce training costs. After an initial pass with a data subset, it calculates each adapter's impact on frozen layers, selecting only the most important ones for full training.

Swipe to next slide



# LoRA-FA

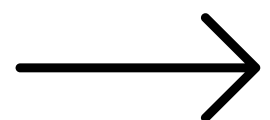
## Freezing part of the Adaptation



LoRA-FA freezes matrix A and only trains matrix B. Image from [4].

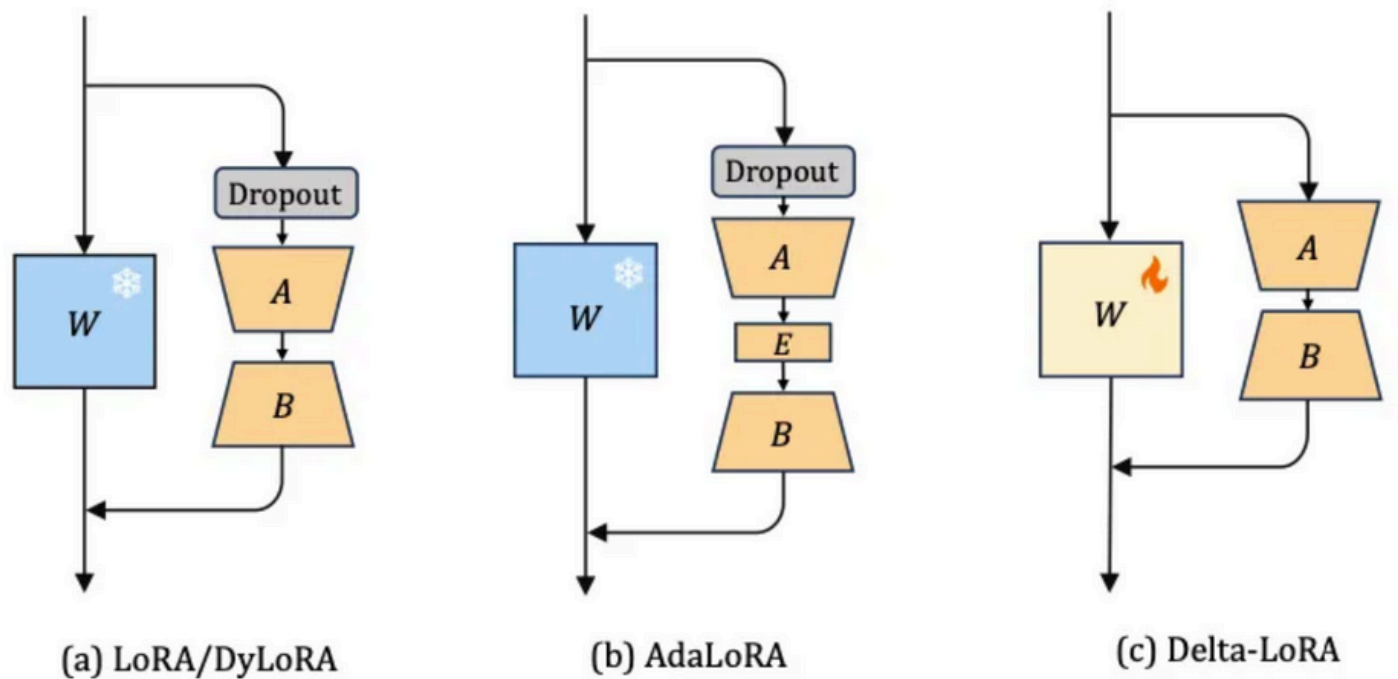
LoRA-FA freezes the parameters of the A matrix along with the original matrix W, further reducing the number of trainable parameters while maintaining performance similar to LoRA.

Swipe to next slide





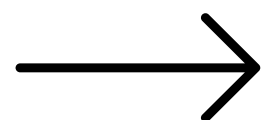
# Delta-LoRA



Delta-LoRA doesn't freeze the matrix  $W$  but updates it with the gradient obtained from  $B \cdot A$ . Image from [8].

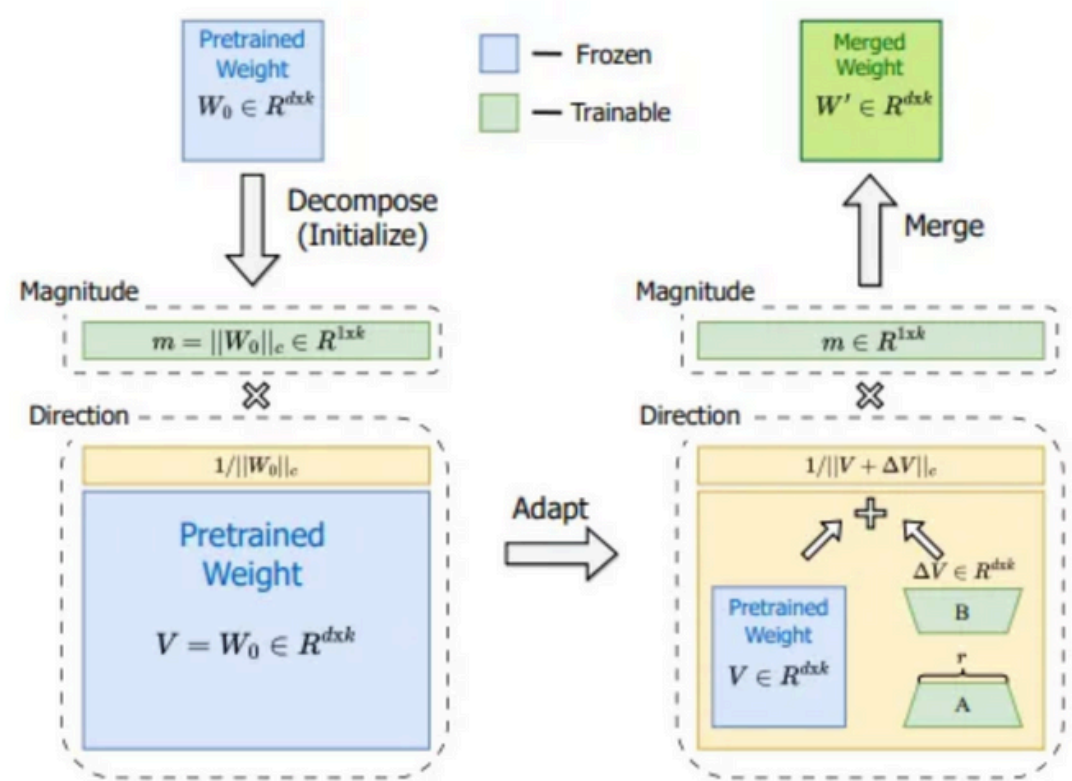
Delta-LoRA improves LoRA by updating the pre-trained matrix  $W$  using the gradient of  $A \cdot B$  between consecutive steps, scaled by a hyperparameter  $\lambda$ . This allows  $W$  to be updated with minimal computational cost, improving performance compared to standard LoRA.

Swipe to next slide



# DoRA

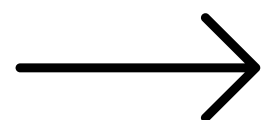
## Decomposed Low-Rank Adaptation

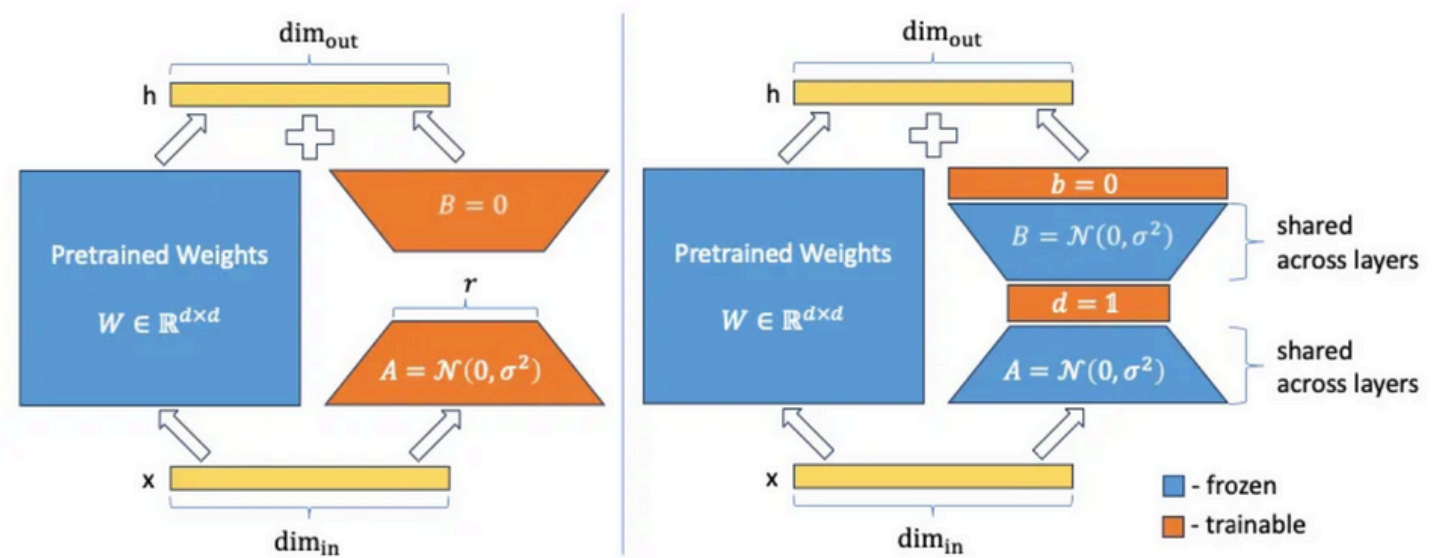


In DoRA, the weight matrix  $W$  is decomposed into magnitude  $m$  and direction  $V$ , which are tuned independently. Image from [7].

DoRA improves LoRA by decomposing each matrix into its magnitude and direction. This adds granularity to the training process, reducing large, destabilizing updates. While there's slight computational overhead, this approach provides better control and performance.

Swipe to next slide





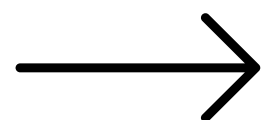
VeRA doesn't train  $A$  and  $B$ , but initializes them to a random projection and trains additional vectors  $d$  and  $b$  instead. Image from [3].

# VeRA

## Vectorized Random Matrix Adaptation

VeRA uses the same frozen  $A$  and  $B$  matrices for all layers, while only training small, layer-specific scaling vectors ( $b$  and  $d$ ). This approach significantly reduces the number of trainable parameters.

Swipe to next slide



**Check Out**

**hub.athina.ai**

**Let's Connect**