

---

## ♦ 1. PEFT (Parameter-Efficient Fine-Tuning)

- Instead of updating all model weights, you **only tune a small number** of parameters.
- **Goal:** Save memory, speed up training, work with big LLMs even on small GPUs.
- **Popular PEFT methods:**
  - LoRA
  - Prefix-Tuning
  - Adapter-Tuning

---

## ♦ 2. LoRA (Low-Rank Adaptation)

- A special PEFT method.
- Instead of changing the full weight matrices, it **injects small low-rank matrices** into the model.
- **Memory efficient, fast, can work with big models like LLaMA, GPT-2/3.**

---

## ♦ 3. Quantization

- **Reduce model size** by lowering precision (like **float32** → **int8** or **int4**).
- Helps to **save RAM, speed up inference**, and sometimes even **finetune on small hardware**.
- Popular tools:
  - **bitsandbytes** (8-bit and 4-bit training)
  - **QLoRA** (Quantized LoRA) → combines quantization + LoRA!

---

## ♦ 4. QLoRA (Quantized LoRA)

- **Best of both worlds:**
    - **Quantize the base model** (to 4-bit) → saves memory.
    - **Apply LoRA adapters** → allows fine-tuning.
  - You can finetune huge models (like 65B) even with a single 24GB GPU!
- 

## ♦ 5. Prefix-Tuning

- You **freeze** the model and **only train some special tokens** (prefix embeddings) that are prepended to inputs.
  - Good for **task-specific tuning** without touching main model.
- 

## ♦ 6. Adapter-Tuning

- **Insert small bottleneck layers** (adapters) between the model layers.
  - Only train adapters, not the full model.
  - Very modular — you can **plug/unplug** different task adapters.
- 

## ♦ 7. Full Fine-Tuning

- Old-school method.
  - Update **all** weights of the model.
  - Very expensive, needs a lot of compute (multi-GPU/TPU setups).
- 

### ✓ In short:

- **LoRA** and **QLoRA** are kings today for efficient finetuning.
- **PEFT** is the general category that includes LoRA, Prefix, Adapters.

- **Quantization** is a trick to handle model size/memory issues.

Great! Let me break down the **architectures** behind each major LLM finetuning technique mentioned earlier. These are high-level views of how these methods modify or extend a base LLM like GPT, BERT, or LLaMA.

---

## LLM Fine-Tuning Architectures

---

### ♦ 1. Full Fine-Tuning Architecture

- **What happens?** Every parameter of the original model is updated.

**Structure:**

[ Input ] → [ Embedding ] → [ Transformer Blocks ] → [ Output ]  
                                  ↑  
                              All weights updated ( $W_q$ ,  $W_k$ ,  $W_v$ , FFN, etc.)

- 

---

### ♦ 2. LoRA (Low-Rank Adaptation) Architecture

- **Key idea:** Replace expensive weight updates with **low-rank matrix inserts** in attention layers.
- **LoRA modifies:**
  - Query ( $W_q$ )
  - Value ( $W_v$ )
  - (Sometimes  $W_k$  and  $W_o$ )

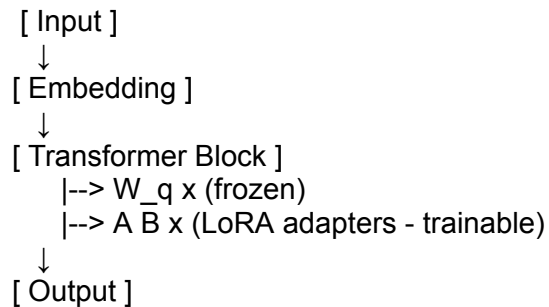
**LoRA injection:**

$$\begin{aligned} W_q x &= (W_q + \Delta W_q) x \\ &= (W_q + A_q B_q) x \end{aligned}$$

Where:  $A_q \in \mathbb{R}^{(d \times r)}$ ,  $B_q \in \mathbb{R}^{(r \times d)}$ ,  $r \ll d$

- 

**Structure:**



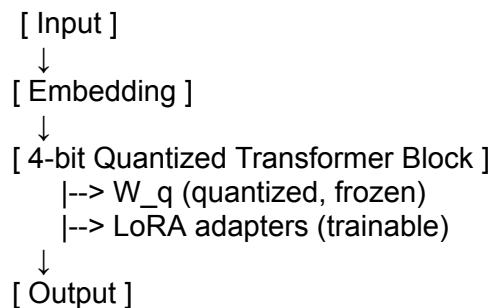
- 

---

### ♦ 3. QLoRA (Quantized + LoRA) Architecture

- **Key idea:** Combine LoRA with **4-bit quantization** of base model weights.
- Quantization helps reduce memory, and LoRA adapters let you finetune.

**Structure:**



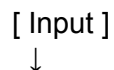
- 

---

### ♦ 4. Prefix Tuning Architecture

- **Key idea:** Keep model weights frozen, but **prepend special tunable prefix vectors** to attention keys and values.

**Structure:**



[ Learned Prefix Tokens ]  
↓  
[ Embedding + Prefix ]  
↓  
[ Transformer Blocks (Frozen) ]  
↓  
[ Output ]

- 
- Prefix vectors are only added to **self-attention mechanism**.

---

## ♦ 5. Adapter Tuning Architecture

- **Key idea:** Inject **small trainable bottleneck MLPs** inside each transformer layer.
- These adapters are the only parts trained.

**Structure:**

[ Transformer Layer ]  
↓  
[ Attention ]  
↓  
[ Adapter Layer ]  
↓  
[ FFN ]

- 

Adapter =

down\_proj ( $W_{\text{down}}$ ) → nonlinearity → up\_proj ( $W_{\text{up}}$ )

- 

---

## ♦ 6. Quantization-Only Architecture

- **Key idea:** Lower precision (like 8-bit, 4-bit) for weights to save memory.
- Doesn't change model architecture.
- May or may not allow training, depending on method (e.g., 8-bit with **bitsandbytes**, 4-bit with **QLoRA**).

---

## ✓ Summary Table

Method	Model Weights	Extra Params	Trainable	Hardware Need	Best For
Full Finetune	All	None	All	High	Full control, small models
LoRA	Frozen	A, B matrices	Few %	Low/Med	Efficient finetuning
QLoRA	Quantized	LoRA adapters	Few %	Low	Very large models (13B-65B)
Prefix-Tuning	Frozen	Prefix vectors	Very Few	Very Low	Task adaptation
Adapter-Tuning	Frozen	Adapter MLPs	Few %	Medium	Modular fine-tuning
Quantization	Quantized	None	Optional	Very Low	Inference & storage efficiency

---

Would you like me to generate a **visual diagram** comparing these architectures side-by-side?