Fine-Tuning LLaMA 2 with LoRA and QLoRA

LoRA (Low-Rank Adaptation)

- **What is LoRA?**

LoRA is a Parameter-Efficient Fine-Tuning (PEFT) method.
Instead of updating all the model parameters (which are billions in LLaMA 2),
LoRA freezes the model and injects small trainable matrices (adapters) into specific layers (usually the attention layers).

Intuition:

Full fine-tuning: Update all model parameters (heavy, slow, expensive).

LoRA fine-tuning: Only train small additional matrices (light, fast, cheap).

- **Where does LoRA Insert Adapters?**

Primarily inside Attention layers:

Query projections (q_proj)

Value projections (v_proj)

(Sometimes also k_proj, o_proj, but mostly q_proj, v_proj)

- **LoRA Key Hyperparameters**

Parameter	Meaning	Typical Values
r	Rank of the low-rank decomposition	4, 8, 16
lora_alpha	Scaling factor for the adapter updates	16, 32
target_modules	Parts of model where LoRA is applied	["q_proj", "v_proj"]
lora_dropout	Dropout rate on LoRA layers	0.05

🔹 Why LoRA?

Cuts VRAM usage by 10× to 100×.

Can fine-tune 7B models on 24GB GPUs (e.g., RTX 4090).

Saves training cost by a huge factor.

Keeps original model intact (only tiny adapters are saved).

🔹 Hardware Requirements for LoRA

Model	Min GPU	Notes
LLaMA 2 7B	**24GB VRAM**	**Very stable.**
LLaMA 2 13B	**48GB VRAM (A6000/A100)**	**Good performance.**
LLaMA 2 65B	**Multi-GPU only (8×A100 80GB)**	**Not practical for individuals.**

🔹 Detailed Time and Dataset Estimation (LoRA)

Model	GPU Needed	Training Time
7B	**1× A100 40GB**	**~8–10 hours**
13B	**2× A100 80GB**	**~18–22 hours**
65B	**8× A100 80GB**	**~2 days**

🔹 How Training Scales with Dataset Size (LoRA)

Dataset Samples	Dataset Size	Training Time Increase
50K	**~4 GB**	**~50% less time**
100K	**~8 GB**	**Base case**
200K	**~16 GB**	**~2× time**
500K	**~40 GB**	**~5× time**

🔹 Detailed Steps to Fine-Tune with LoRA

- Prepare dataset (instruction format: instruction, input, output).
- Load model normally (no quantization).
- Apply LoRA adapters on target modules.
- Use PEFT library to manage adapters.
- Train using Huggingface Trainer with mixed-precision (bf16/fp16).
- Save only LoRA adapters, not full model.
- Inference = Load base model + LoRA adapter → Serve.

QLoRA (Quantized Low-Rank Adaptation)

🔹 What is QLoRA?

QLoRA = 4-bit quantized LLaMA 2 + LoRA adapters fine-tuned.

It compresses the large model during training using 4-bit quantization to:

Reduce memory footprint.

Enable fine-tuning even bigger models on smaller GPUs.

Comparison:

Feature	LoRA	QLoRA
Base model size	**Full fp16/bf16**	**4-bit quantized**
Memory use	**Medium**	**Extremely low**
Training time	**Fast**	**Slightly slower**
Training complexity	**Low**	**Medium**

- **How does QLoRA Quantize?**

During training:

Model weights stored in 4-bit.

Computations in bf16/float16 for stability.

Double quantization (optional): Quantize quantization parameters.

- **QLoRA Key Hyperparameters**

Parameter	Meaning	Typical Value
bnb_4bit_use_double_quant	**Double-quantization for memory saving**	**True**
bnb_4bit_quant_type	**Type of quantization**	**"nf4" (best for NLP)**
bnb_4bit_compute_dtype	**Compute data type**	**torch.bfloat16**

- **Why QLoRA?**

Fine-tune LLaMA 2 13B on 1x 24GB GPU (unthinkable without QLoRA).

Fine-tune LLaMA 2 65B with multi-GPU, but ~50% less VRAM needed.

Huge cost saving without major performance loss.

🔹 Hardware Requirements for QLoRA

Model	Min GPU	Notes
LLaMA 2 7B	16GB VRAM (A6000, 3090)	Very smooth.
LLaMA 2 13B	24GB VRAM (A100, 4090)	Manageable.
LLaMA 2 65B	4x A100 80GB	Very heavy still.

🔹 Detailed Time and Dataset Estimation (QLoRA)

Model	GPU Needed	Training Time
7B	1× A100 40GB / RTX 4090	~9–11 hours
13B	2× A100 80GB	~20–26 hours
65B	8× A100 80GB	~2–3 days

🔹 How Training Scales with Dataset Size (QLoRA)

Same scaling as LoRA (dataset size linearly increases training time).
However, QLoRA memory usage stays low, so bigger batches are possible.

🔹 Detailed Steps to Fine-Tune with QLoRA

Prepare dataset (same instruction format).

Load model with 4-bit quantization (bnb_config in Huggingface).

Apply LoRA adapters using PEFT.

Use paged optimizers (paged_adamw_8bit) for memory efficiency.

Train using Trainer with bf16 precision.

Save adapters only.

Inference = Load base 4-bit model + LoRA adapter → Serve.

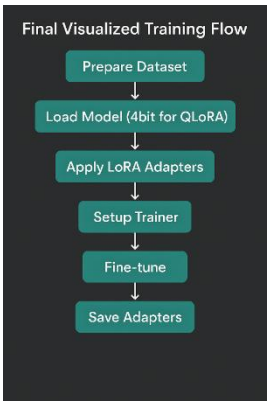
- **Key Efficiency Tricks (Both LoRA and QLoRA)**

Trick	Effect
Gradient checkpointing	Save VRAM by trading compute.
Gradient accumulation	Simulate large batch without big memory.
Mixed precision (bf16)	Speed up training by 2x–3x.
Smaller LoRA rank (r=4)	Faster training, lower memory.
Early stopping	Save time if validation loss plateaus.

Final Comparison Table

Feature	**LoRA**	**QLoRA**
Base Model	Full precision	4-bit quantized
Fine-tuning Cost	Low	Very Low
Memory Usage	Low	Very Low
Training Speed	Fast	Slightly slower
Use Case	Fine-tune with reasonable GPUs	Fine-tune bigger models on small GPUs

- **Final Visualized Training Flow**



Prepare Dataset -> Load Model (4bit for QLoRA) -> Apply LoRA Adapters ->

Setup Trainer -> Fine-tune -> Save Adapters -> Serve Model