QLoRA

EFFICIENT FINE-TUNING ON CONSUMER HARDWARE

FROM LORA TO QLORA: BREAKTHROUGH IN MODEL COMPRESSION

PROFESSIONAL TECHNICAL DOCUMENTATION

MADE BY SAFIA TIFOUR

July 31, 2025

Contents

1 From LoRA to QLoRA (Quantized Low-Rank Adaptation)

1.1 Transition: Why Do We Need QLoRA?

While **LoRA** significantly reduces the number of **trainable parameters**, it still requires:

- Loading the **full precision (FP16 or FP32) base model** into memory
- Using **expensive GPUs** (with lots of VRAM) for inference/training

So although LoRA is efficient in training, it still:

- **Uses large memory**
- **Limits fine-tuning to high-end hardware**

Problem: Can we do even better — fine-tune *full-sized models* on *low-end consumer GPUs (e.g., 24GB VRAM)*?

Solution: **QLoRA**, proposed in 2023 by Tim Dettmers et al.

2 What is QLoRA?

QLoRA stands for: **Quantized Low-Rank Adapter**

It combines:

- **Quantization (4-bit)** of the base model weights
- With **LoRA-style low-rank adapters** for fine-tuning

This allows:

- Training *very large models* (e.g., 65B parameters)
- On **a single GPU**
- **Without performance loss** compared to full precision training

3 High-Level Intuition

You know LoRA already: It adapts a frozen model by injecting **low-rank adapters**.
Now:

$$\text{QLoRA} = \text{LoRA} + \text{Quantization}$$

We go **a step further**:

1. **Compress** the frozen weights to **4-bit precision** (quantization)
2. **Avoid touching them** — just use them in forward pass
3. Train only the **LoRA adapters (A and B)** as before — **in full precision (FP16)**

So we:

- Save memory by quantizing base weights
- Retain flexibility by training small adapters

4 Motivation: Why Quantize?

4.1 Large models are memory-intensive

Even loading a 13B model in FP16 can require:

- $\sim$26GB VRAM (13B $\times$ 2 bytes/param)

With quantization (e.g., 4-bit):

- Only $\sim$6.5GB is needed

4.2 LoRA still loads weights in full precision

LoRA helps by reducing trainable parameters — **but not model size**
You still need a high-end GPU just to run the model

4.3 QLoRA enables

- Full **LoRA training on quantized weights**
- **Efficient memory use**
- No major loss in accuracy

5 Quantization: What Does 4-bit Mean?

Quantization = Reducing precision of numbers

Type	Bits	Value Range
FP32	32	Very high precision
FP16	16	Half precision
INT8	8	256 values
INT4	**4**	**Only 16 values!**

Table 1: Precision Comparison

In **QLoRA**, the base model weights are stored as **4-bit integers**, not floats.
This brings:

- 4× less memory usage
- Faster data transfer
- Smaller model storage

6 Challenge: Can we quantize AND still train effectively?

Yes — **but with care.**
QLoRA introduces two innovations to make this work:

1. **Double quantization**
2. **NF4 (Normalized Float 4-bit) quantization format**

7 NF4 Quantization: Why it Matters

Standard INT4 (linear mapping) performs poorly:

- Not enough dynamic range for neural network weights

NF4 (Normalized Float 4-bit) is a custom quantization scheme:

- Keeps the **distribution of values closer to real weights**
- Has better performance than INT4 or FP8

Benefits:

- Matches full-precision accuracy
- Requires only 4 bits per weight

8 Double Quantization

A second trick QLoRA uses:

Compress not just the weights, but also the **quantization constants (scales)**

This allows even more memory savings **without sacrificing accuracy**.

9 Final QLoRA Architecture

Summary of components:

Component	Precision	Trainable?	Notes
Base model weights	**4-bit (NF4)**	No - Frozen	Saved in quantized form
LoRA adapters (A, B)	FP16	Yes - Trainable	Low-rank matrices
Optimizer state	FP32 or FP16	Yes	Only for adapters

Table 2: QLoRA Architecture Components

10 Training Intuition

Just like LoRA:

- You train $\Delta W = BA$ (low-rank matrices) **only**
- But now the **base matrix W is quantized**

In forward pass:

$$y = \text{Quantized}(W) \cdot x + \frac{\alpha}{r} \cdot B(Ax)$$

In backward pass:

- Gradients only flow through A and B

The quantized W is never updated — it's just used in the computation