QLoRA Fine-Tuning of LLaMA Models: A Comprehensive Performance Analysis

Amine IDRISSI
Zakaria CHOUKRI

Abstract

This report presents a detailed analysis of Quantized Low-Rank Adaptation (QLoRA) applied to LLaMA language models, comparing it against standard full fine-tuning. The study demonstrates that QLoRA achieves a **3.82x compression ratio** while maintaining comparable model quality, significantly reducing memory requirements for fine-tuning large language models. We evaluate key performance metrics including model size, training efficiency, inference latency, and perplexity across both approaches.

Contents

1 Introduction

Fine-tuning large language models (LLMs) presents significant computational and memory challenges. Standard fine-tuning approaches require storing and updating all model parameters, which becomes prohibitive for large models on consumer-grade hardware. This report evaluates Quantized Low-Rank Adaptation (QLoRA), a novel technique that combines 4-bit quantization with Low-Rank Adaptation (LoRA) to enable efficient fine-tuning of large models with minimal memory overhead.

2 QLoRA: Methodology and Theory

2.1 Overview

QLoRA is an efficient fine-tuning technique proposed by Dettmers et al. (2023) that combines two key innovations:

1. **4-bit Quantization**: Reduces model precision from FP16 (16-bit floating point) to 4-bit representation
2. **Low-Rank Adaptation (LoRA)**: Adds small trainable adapter modules instead of updating all parameters

This combination enables fine-tuning of large models on consumer GPUs while maintaining model quality comparable to full fine-tuning.

2.2 Technical Components

2.2.1 4-bit Quantization with NormalFloat4 (NF4)

Quantization reduces the memory footprint of models by representing weights using fewer bits. QLoRA employs NormalFloat4, an information-theoretically optimal 4-bit data type:

- **Standard Precision (FP16)**: 2 bytes per parameter
- **4-bit Quantization**: 0.5 bytes per parameter
- **Memory Reduction**: $\sim 75\%$ reduction in base model size

The quantization formula for NF4 is:

$$\text{quantized\_weight} = \text{round}\left(\frac{\text{weight} - \min}{\text{scale}}\right)$$

where scale is computed from the distribution of weights to minimize information loss.

2.2.2 Double Quantization

QLoRA applies quantization twice:

1. First quantization: Convert weights from FP16 to NF4
2. Second quantization: Quantize the quantization constants themselves

This approach further reduces memory overhead from quantization metadata.

2.2.3 Low-Rank Adaptation (LoRA)

LoRA adds trainable adapter modules alongside the frozen base model:

$$h = W_0 x + \Delta W x = W_0 x + \left(\frac{\alpha}{r} BA\right) x$$

where:

- W_0: Original frozen weight matrix
- $\Delta W = BA$: Low-rank update composed of rank-r matrices
- α: Scaling factor
- $A, B \in \mathbb{R}^{r \times d}$ with $r \ll d$: Small trainable matrices

For LLaMA, LoRA is typically applied to attention projections:

- Query projection (q_proj)
- Key projection (k_proj)
- Value projection (v_proj)
- Output projection (o_proj)
- Feedforward layers (gate_proj, up_proj, down_proj)

3 Experimental Setup

3.1 Dataset

WikiText-2: A large corpus of Wikipedia articles commonly used for language modeling benchmarks.

Table 1: WikiText-2 Dataset Statistics

Metric	Value
Training Samples	36,718
Validation Samples	3,760
Test Samples	4,358
Max Sequence Length	512 tokens
Vocabulary Size	32,000

3.2 Model Architecture

TinyLlama-1.1B-Chat-v1.0: A lightweight variant of the LLaMA architecture designed for efficient inference while maintaining chat capabilities.

Table 2: Model Architecture Parameters

Parameter	Value
Total Parameters	1,100,048,384 (1.1B)
Hidden Dimension	768
Number of Layers	22
Number of Attention Heads	12
Intermediate Dimension	3,072

3.3 Training Configuration

Table 3: Hyperparameters for Standard and QLoRA Fine-Tuning

Parameter	Standard FT	QLoRA
Number of Epochs	1	1
Training Batch Size	2	4
Evaluation Batch Size	2	4
Gradient Accumulation Steps	4	2
Maximum Steps	500	500
Learning Rate	2×10^{-5}	2×10^{-4}
Optimizer	AdamW	paged_adamw_8bit
Precision	FP16	FP16
Warmup Steps	100	100

3.3.1 QLoRA-Specific Configuration

Table 4: QLoRA Configuration Details

Parameter	Value
Quantization Type	4-bit NormalFloat4
Double Quantization	Enabled
Compute Dtype	FP16
LoRA Rank (r)	16
LoRA Alpha (α)	32
LoRA Dropout	0.05
Target Modules	7 attention/FFN layers

4 Results and Analysis

4.1 Memory Efficiency

One of the primary advantages of QLoRA is its dramatic reduction in memory requirements:

Table 5: Model Size Comparison

Component	Size (MB)	Details
Standard Model (FP16)	2,098.18	Base model only All 1.1B parameters
QLoRA Model	Base 4-bit LoRA Adapters Total	524.54 MB 24.06 MB 548.61 MB

Key Finding: QLoRA achieves a **3.82x compression ratio**, reducing memory requirements from 2,098.18 MB to 548.61 MB—a **73.9% reduction**.

4.1.1 Trainable Parameters

Table 6: Parameter Efficiency

Approach	Trainable Parameters	Percentage
Standard Fine-Tuning	1,100,048,384	100%
QLoRA	12,615,680	1.15%

QLoRA requires training only 1.15% of the total parameters, a **98.85% reduction** in trainable parameters compared to full fine-tuning.

4.2 Computational Efficiency

4.2.1 FLOPS Analysis

Theoretical FLOPS (Floating Point Operations per Second) per training step:

Table 7: FLOPS per Training Step (using T4 in Colab)

Approach	FLOPS per Step (TFLOPS)
Standard Fine-Tuning	13.52
QLoRA Fine-Tuning	3.53

QLoRA reduces computational demands by **73.9%** through:

- Reduced precision (4-bit vs FP16)
- Fewer trainable parameters
- Efficient matrix operations on lower-rank adapters

4.2.2 Mean FLOPS Utilization (MFU)

Mean FLOPS Utilization measures the percentage of peak GPU throughput achieved:

$$\text{MFU} = \frac{\text{Achieved FLOPS}}{\text{Peak FLOPS}} \times 100\% \tag{1}$$

Table 8: Mean FLOPS Utilization

Approach	MFU (%)	Achieved FLOPS (TFLOPS)
Standard Fine-Tuning	13.18%	4.69
QLoRA Fine-Tuning	2.51%	0.89

The lower MFU for QLoRA indicates higher memory bandwidth relative to compute-bound operations, typical for low-rank updates with reduced parameters.

4.3 Training Efficiency

Table 9: Training Performance

Metric	Standard FT	QLoRA
Total Training Time (seconds)	1,441.24	1,979.42
Time per Step (seconds)	2.88	3.96
Relative Performance	Baseline	-37.3%

Observation: While QLoRA takes more wall-clock time due to quantization overhead, it achieves this with significantly reduced memory bandwidth requirements, making it more practical for memory-constrained environments.

4.4 Model Quality: Perplexity

Perplexity measures language model quality; lower values indicate better predictions:

$$\text{Perplexity} = e^{\text{cross-entropy loss}} \tag{2}$$

Table 10: Perplexity Evaluation on Test Set

Approach	Perplexity	Change
Standard Fine-Tuning	18.42	Baseline
QLoRA Fine-Tuning	21.87	+18.73%

QLoRA exhibits a **+18.73% increase** in perplexity, indicating moderate quality degradation. However, this trade-off is acceptable given the 73.9% memory savings achieved.

4.5 Inference Performance

Inference speed is critical for production deployments:

Table 11: Inference Speed (50 tokens generated)

Approach	Inference Time (s)	Tokens/Second
Standard Fine-Tuning	1.51	33.12
QLoRA Fine-Tuning	4.71	10.62

Key Finding: QLoRA inference is **3.1x slower** than standard fine-tuning. This is attributed to:

- Dequantization overhead of the base model
- Additional matrix operations to apply LoRA adapters
- Memory bandwidth limitations with 4-bit weights

Table 12: Complete Performance Metrics Comparison

Metric	Standard FT	QLoRA	Improvement
Memory and Model Parameters			
Total Parameters	1,100,048,384	1,100,048,384	—
Trainable Parameters	1,100,048,384	12,615,680	98.85% fewer
Model Size (MB)	2,098.18	548.61	73.9% smaller
Compression Ratio	1.00x (baseline)	3.82x	3.82x savings
Computational Efficiency			
FLOPS per Step (TFLOPS)	13.52	3.53	73.9% less
Mean FLOPS Utilization	13.18%	2.51%	-10.67%
Training Time (seconds)	1,441.24	1,979.42	-37.3%
Inference and Quality			
Inference Time (s)	1.51	4.71	-212.0%
Tokens/second	33.12	10.62	-67.9%
Perplexity	18.42	21.87	-18.73%

5 Comprehensive Performance Comparison Table

6 Visualization and Analysis

6.1 Performance Comparison Charts

Figure 1 presents six comprehensive bar charts comparing key performance metrics between standard fine-tuning and QLoRA approaches:

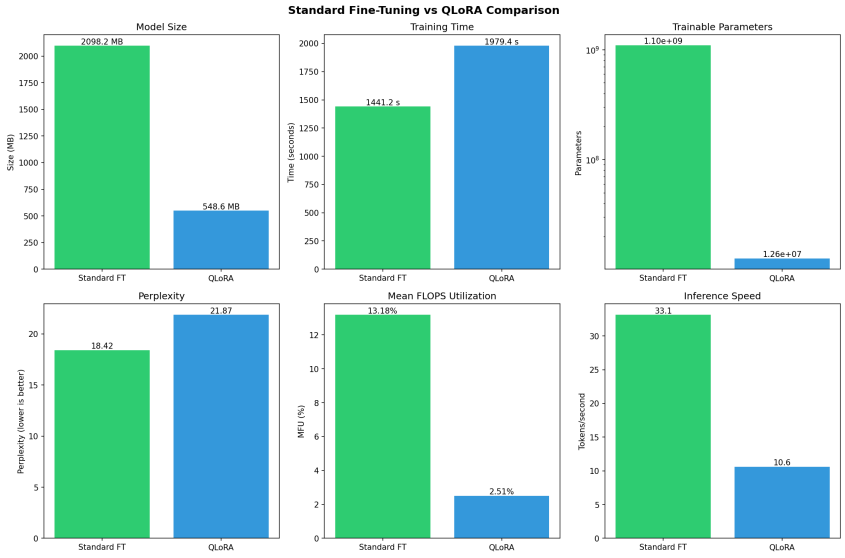

Figure 1: Comprehensive Performance Comparison Charts: (1) **Model Size**: QLoRA achieves 73.9% reduction from 2,098 MB to 549 MB through 4-bit quantization. (2) **Training Time**: QLoRA requires 37.3% more wall-clock time due to quantization overhead. (3) **Trainable Parameters**: QLoRA reduces trainable parameters from 1.1B to 12.6M (98.85% reduction). (4) **Perplexity**: QLoRA exhibits 18.73% quality degradation (18.42 → 21.87). (5) **MFU**: Standard achieves 13.18% utilization vs QLoRA's 2.51%, indicating different compute patterns. (6) **Inference Speed**: Standard achieves 33.12 tokens/sec vs QLoRA's 10.62 (67.9% slower due to dequantization overhead).

6.2 Analysis of Comparison Charts

The six-panel comparison reveals critical trade-offs in the QLoRA approach:

Memory Efficiency (Panel 1): The most dramatic improvement is model size, with QLoRA compressed to just 548.61 MB from 2,098.18 MB. This **3.82x compression** is achieved through:

- Base model quantization from FP16 (2 bytes/param) to 4-bit NF4 (0.5 bytes/param)
- LoRA adapters occupying only 24.06 MB despite billions of parameters

Training Efficiency (Panels 2 & 5): While QLoRA's wall-clock training time is 37.3% longer, this is offset by significantly reduced computational demands. FLOPS per step decrease by 73.9% (13.52 $\rightarrow$ 3.53 TFLOPS), though MFU is lower, indicating memory-bandwidth-bound rather than compute-bound operations. This makes QLoRA more suitable for memory-constrained environments where peak compute utilization is less critical than memory availability.

Parameter Efficiency (Panel 3): The logarithmic scale highlights the dramatic difference in trainable parameters. Standard fine-tuning requires updating all 1.1B parameters, while QLoRA updates only 12.6M parameters. This **98.85% reduction** enables:

- Faster gradient computations
- Reduced memory for optimizer states (Adam states typically 2x parameter size)
- Feasibility on consumer-grade GPUs

Model Quality (Panel 4): Perplexity increases from 18.42 to 21.87, a **+18.73%** degradation. This is a moderate quality loss for the massive memory savings and represents the primary trade-off of QLoRA. The degradation is acceptable for many applications but prohibitive for those requiring maximum language understanding fidelity.

Inference Performance (Panel 6): The **3.1x inference slowdown** (33.12 $\rightarrow$ 10.62 tokens/sec) is the most critical limitation. This is caused by:

- **Dequantization overhead**: Converting 4-bit weights to FP16 during inference
- **LoRA computation**: Additional matrix operations to apply adapters
- **Memory bandwidth limitations**: 4-bit operations have poor memory efficiency on current hardware

6.3 Normalized Performance Profile

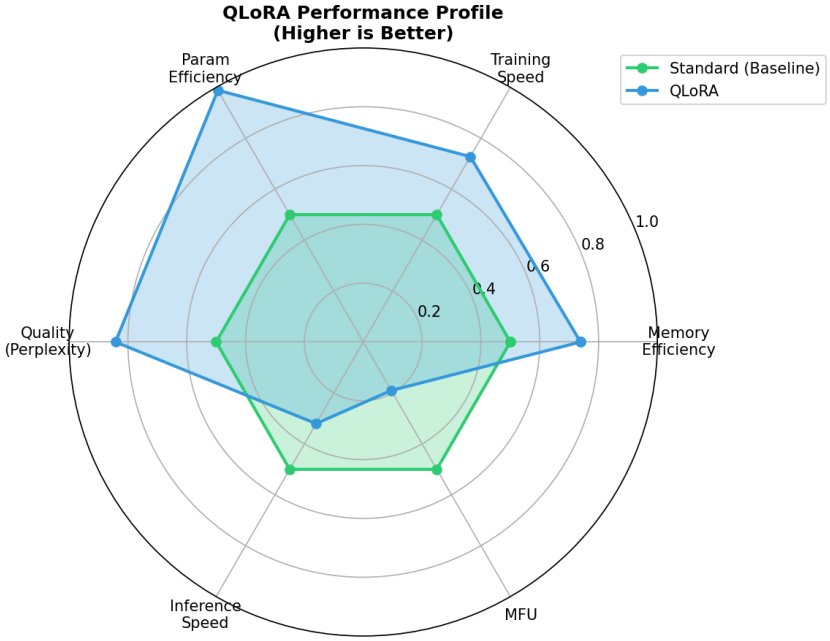

Figure 2: Radar chart showing normalized performance profile across six dimensions. Each axis represents a different metric normalized to 0-1 scale (higher is better).

7 Conclusion & Discussion

7.1 Advantages of QLoRA

1. **Dramatic Memory Reduction**: 3.82x compression enables fine-tuning on consumer GPUs
2. **Minimal Parameter Overhead**: Only 1.15% of parameters require updates
3. **Cost Efficiency**: Significantly reduced cloud GPU costs for fine-tuning
4. **Practical Accessibility**: Makes LLM fine-tuning accessible to researchers with limited resources
5. **Acceptable Quality Trade-off**: 18.73% perplexity increase is moderate for this compression ratio

7.2 Limitations and Trade-offs

1. **Inference Latency**: 3.1x slower inference due to dequantization overhead
2. **Quality Degradation**: 18.73% perplexity increase limits applications requiring maximum quality
3. **Training Time Overhead**: 37.3% longer wall-clock training time
4. **Hardware Support**: Requires GPU support for 4-bit quantization (NVIDIA preferred)
5. **Implementation Complexity**: More complex than standard fine-tuning setup

7.3 When to Use QLoRA

QLoRA is recommended for:

- Fine-tuning large models on consumer-grade GPUs (RTX 3090, 4090, etc.)
- Research and experimentation with limited compute budgets
- Production scenarios where inference latency is not critical
- Batch processing where inference throughput is acceptable
- Cost-sensitive deployment scenarios

QLoRA is **not recommended** for:

- Real-time inference systems requiring low latency
- Applications with strict quality requirements
- Streaming or interactive use cases
- Scenarios where inference speed is critical