]

Optimizing the Deployment of Large Language Models on Edge Devices

A Quantization Approach

Tharun Reddy Banda

Matricula: 51793A

Master's Degree Program: Computer Science
Università degli Studi di Milano
Academic Year: 2025/2026
Course: Natural Language Processing

July 30, 2025

Contents

List of Tables

List of Figures

Project Repository

Project Repository

This research project's complete implementation, including source code, datasets, experimental scripts, and additional documentation, is available in the following repository:

https://github.com/BandaTharun/Optimizing-the-Deployment-of-llm-models

The repository contains:

- Complete source code for all experiments
- Performance benchmarking scripts
- Deployment configuration files
- Additional technical documentation

Note: Please refer to the repository's README file for detailed setup instructions and usage guidelines.

Abstract

The deployment of Large Language Models (LLMs) on edge devices presents significant challenges due to resource constraints. This research investigates optimization through quantization techniques, which reduce model precision without substantial performance degradation.

The study explores post-training quantization and quantization-aware training, analyzing their effectiveness in reducing model size and computational requirements. Five inference tools (Langchain, Ctransformers, Llama-cpp, Ollama, and Llamafile) were evaluated measuring response time, CPU usage, and memory consumption.

Results indicate that Llamafile achieved superior performance with 39-46× faster response times compared to slower alternatives and lowest resource consumption. Successful deployment validation on edge hardware demonstrated the practical viability of quantized LLM inference.

This research provides practical deployment strategies and tool selection guidelines, demonstrating the feasibility of sophisticated language models on resource-constrained devices while maintaining privacy and reducing latency.

Keywords: Large Language Models, Edge Computing, Quantization, Model Optimization, Resource Constraints

1 Introduction and Problem Statement

1.1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing through sophisticated understanding and generation capabilities. However, their deployment traditionally relies on cloud infrastructure, creating dependencies on internet connectivity and raising privacy concerns.

Edge computing offers reduced latency, enhanced privacy, and improved reliability by bringing computation closer to data sources. The deployment of LLMs on edge devices holds potential for democratizing AI access while addressing cloud-based limitations, enabling offline functionality and eliminating sensitive data transmission to remote servers.

1.2 Problem Statement

The primary challenge in deploying LLMs on edge devices stems from the fundamental mismatch between model requirements and device capabilities. Modern LLMs typically contain billions of parameters, requiring substantial memory (often 4-16GB) and computational resources that exceed the capacity of most edge devices.

1.2.1 Resource Constraints

- **Memory Limitations:** Edge devices typically have 1-8GB RAM compared to cloud servers with 32GB or more
- **Storage Constraints:** Limited storage capacity conflicts with large model sizes (2-10GB)
- **Computational Power:** Reduced processing capabilities compared to high-performance cloud infrastructure
- **Energy Efficiency:** Battery-powered devices require power-conscious operation

1.2.2 Performance Requirements

- **Latency Sensitivity:** Real-time applications demand low response times
- **Quality Maintenance:** Model compression must preserve output quality
- **Hardware Utilization:** Efficient use of available computational resources

1.3 Research Objectives

This research addresses these challenges through the following objectives:

1. Investigate quantization techniques for reducing LLM size and computational complexity while maintaining performance integrity
2. Evaluate inference tools to identify optimal solutions for edge deployment based on resource efficiency and response time
3. Demonstrate practical deployment on real edge devices with comprehensive performance analysis

Research Impact

This research provides crucial insights for practitioners seeking to deploy sophisticated language models in edge computing environments while maintaining efficiency, privacy, and performance standards.

2 Literature Review and Background

2.1 LLM Fundamentals

Large Language Models are deep learning systems based on transformer architecture that process and generate human language through self-attention mechanisms. Transformers employ self-attention to weigh the importance of different words when processing each token, enabling effective capture of long-range dependencies within text sequences.

Modern state-of-the-art models like GPT-3 (175 billion parameters), BERT, and T5 demonstrate remarkable capabilities but require substantial computational resources. The parameter count directly correlates with model size and memory requirements, creating significant deployment challenges for resource-constrained environments.

2.2 Edge Computing Challenges

Edge devices operate under strict resource constraints that fundamentally conflict with LLM requirements. These limitations manifest across multiple dimensions:

2.2.1 Hardware Limitations

- Processing power typically 10-100× lower than cloud servers
- Memory constraints ranging from 1-8GB compared to cloud infrastructure with 32GB+
- Storage limitations affecting model loading and caching capabilities
- Limited bandwidth for model updates and data synchronization

2.2.2 Energy Considerations

Battery-powered edge devices face critical energy efficiency requirements. Running resource-intensive LLMs can rapidly drain batteries, making energy optimization essential for practical deployment. The computational intensity of transformer models, particularly during inference, presents significant power consumption challenges.

2.2.3 Performance Requirements

Edge applications often demand real-time responses with latency requirements under 100ms. However, large model inference can introduce substantial computational delays, potentially compromising user experience and application viability.

2.3 Quantization Overview

Model quantization emerges as the primary solution for addressing LLM deployment challenges on edge devices. Quantization reduces model precision by converting weights and activations from high-precision floating-point (typically 32-bit) to lower-precision representations (16-bit, 8-bit, or even 4-bit integers).

2.3.1 Quantization Benefits

- **Size Reduction:** 2-8× model size reduction depending on precision level
- **Memory Efficiency:** Proportional reduction in memory requirements during inference
- **Computational Speed:** Faster integer operations compared to floating-point calculations
- **Energy Savings:** Lower precision operations consume less power

The quantization process involves mapping continuous floating-point values to discrete integer representations through mathematical transformations that preserve model functionality while reducing computational overhead. This technique enables deployment of sophisticated models on resource-constrained hardware without complete architectural redesign.

3 Methodology - Quantization Techniques

3.1 Quantization Theory

Quantization transforms continuous floating-point values into discrete integer representations through a mathematical mapping process. The fundamental quantization equation maps real values (r) to quantized values (q) using scaling factors and zero points:

$$\boxed{q = \text{round}\left(\frac{r}{s}\right) + z} \tag{1}$$

Where:

- q = quantized integer value
- r = original floating-point value
- s = scaling factor
- z = zero point offset

3.1.1 Data Representation

Modern computing systems represent numbers using fixed bit allocations. Floating-point numbers follow IEEE 754 standards with separate bits for sign, exponent, and mantissa. Integer representations use two's complement encoding, providing more efficient storage and computation for quantized values.

The quantization process divides the continuous value range into discrete intervals, mapping original values to the nearest interval boundaries. This discretization introduces quantization error, but careful parameter selection minimizes accuracy loss while achieving significant compression.

3.2 Quantization Approaches

3.2.1 Symmetric Quantization

Symmetric quantization centers intervals around zero with equal positive and negative ranges. The quantization mapping maintains symmetry: $q = \text{round}(r/s)$, where the zero point $z = 0$. This approach provides balanced representation for positive and negative values but may waste representational capacity if data distribution is asymmetric.

3.2.2 Asymmetric Quantization

Asymmetric quantization adapts to actual data distributions by allowing non-zero offset points. This method can more efficiently represent data with skewed distributions, utilizing the full quantization range more effectively. However, it requires additional zero-point storage and slightly more complex computations.

3.2.3 Post-Training Quantization (PTQ)

PTQ applies quantization to pre-trained models without additional training. The process analyzes model weights and activations using representative datasets to determine optimal quantization parameters. PTQ offers simplicity and rapid deployment but may suffer accuracy degradation for precision-sensitive models.

PTQ Analysis

Advantages:

- No additional training required
- Fast implementation process
- Compatible with existing pre-trained models

Limitations:

- Potential accuracy loss
- Limited optimization for quantization effects
- May require careful calibration dataset selection

3.2.4 Quantization-Aware Training (QAT)

QAT incorporates quantization considerations during model training by adding fake quantization operations to the computation graph. These operations simulate quantization effects during forward passes while maintaining full precision gradients during backpropagation. This approach trains models that are inherently robust to quantization effects.

QAT Analysis

Advantages:

- Superior accuracy preservation
- Model optimization for quantized inference
- Better handling of quantization noise

Limitations:

- Requires complete retraining process
- Increased training complexity and time
- May need specialized training frameworks

The selection between PTQ and QAT depends on available computational resources, time constraints, and accuracy requirements. PTQ provides rapid deployment for most applications, while QAT offers optimal performance for critical accuracy requirements.

4 Experimental Setup

4.1 Hardware Specifications

The experimental evaluation was conducted on a MacBook M2 Pro to establish baseline performance metrics for tool comparison before edge deployment.

Testing Platform Specifications

MacBook M2 Pro:

- **Processor:** Apple M2 Pro chip
- **Memory:** 16GB unified memory
- **Storage:** SSD storage with high-speed access
- **Operating System:** macOS
- **Use Case:** Controlled testing environment for systematic tool evaluation

This configuration provides a stable, high-performance platform for conducting fair comparisons between inference tools without hardware-related performance variations affecting the results.

4.2 Software Tools Evaluated

Five popular open-source inference tools were systematically evaluated to identify optimal solutions for edge LLM deployment:

4.2.1 LangChain Library

- Framework for building LLM applications with extensive integration capabilities
- Provides high-level abstractions for model interaction and chain composition

- Rich ecosystem but potentially higher overhead for simple inference tasks

4.2.2 Ctransformers Library

- Python bindings for transformer models implemented in C/C++ using GGML library
- Optimized for CPU inference with quantized model support
- Lightweight implementation focused on efficiency

4.2.3 Llama-cpp

- Open-source C/C++ implementation optimized for local deployment
- Supports multiple quantization formats and hardware acceleration
- Direct model execution without framework overhead

4.2.4 Ollama

- Streamlined tool for running open-source LLMs locally
- Unified package management through Modelfiles
- Supports multiple model formats with simplified deployment workflow

4.2.5 Llamafile

- Single-file executable combining llama.cpp with Cosmopolitan Libc
- Zero-installation deployment with built-in web interface
- Cross-platform compatibility with minimal dependencies

4.3 Testing Methodology

The experimental evaluation employed a systematic approach to ensure reliable and comparable results across all tools and hardware configurations.

4.3.1 Test Scenarios

1. **Single Prompt Test:** Individual question processed 10 times for statistical accuracy
2. **Multi-Prompt Test:** Three questions in single request processed 10 times

4.3.2 Performance Metrics

- **Response Time:** Total time from query submission to complete response generation
- **CPU Usage:** Processor utilization percentage during inference
- **Memory Usage:** RAM consumption percentage during model execution
- **Tokens per Second:** Output generation rate for throughput analysis

4.3.3 Testing Protocol

1. System resource monitoring initiated before each test
2. Identical prompts submitted to ensure fair comparison
3. Hardware metrics recorded throughout inference process
4. Statistical analysis performed on 10-iteration datasets
5. Environment reset between tool evaluations to prevent interference

4.3.4 Quality Assurance

- Consistent prompt format across all evaluations
- Hardware thermal management to prevent throttling
- Network isolation to eliminate external interference
- Systematic tool configuration to ensure optimal performance

This methodology ensures robust comparison data while accounting for system variability and providing statistically significant results for optimal tool selection.

5 Results - Tool Performance Comparison

5.1 Performance Metrics Analysis

The systematic evaluation of five inference tools revealed significant performance variations across response time, resource utilization, and efficiency metrics. All tests were conducted using identical hardware configurations and standardized prompts to ensure fair comparison.

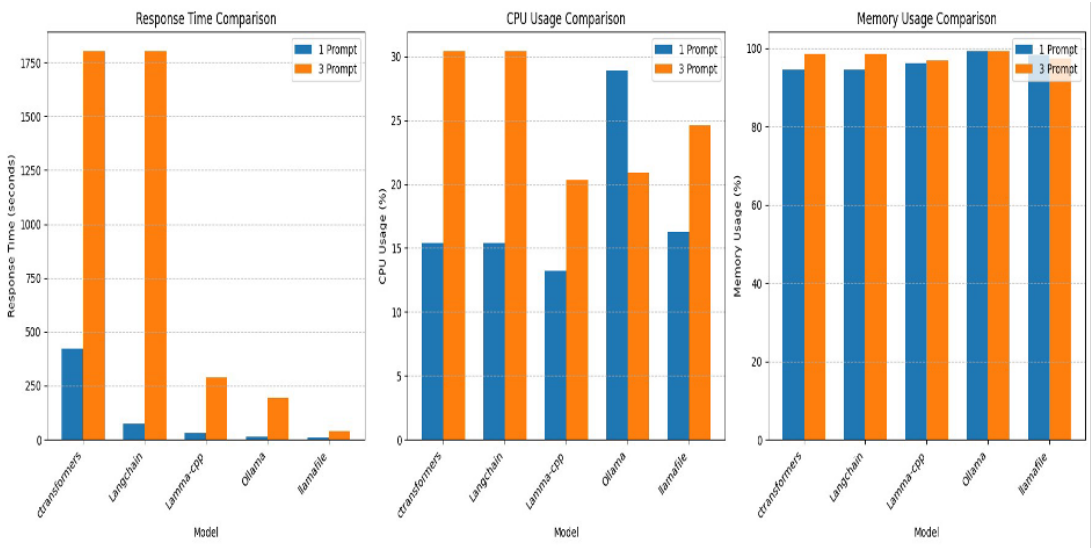

Figure 1: Comprehensive Performance Comparison

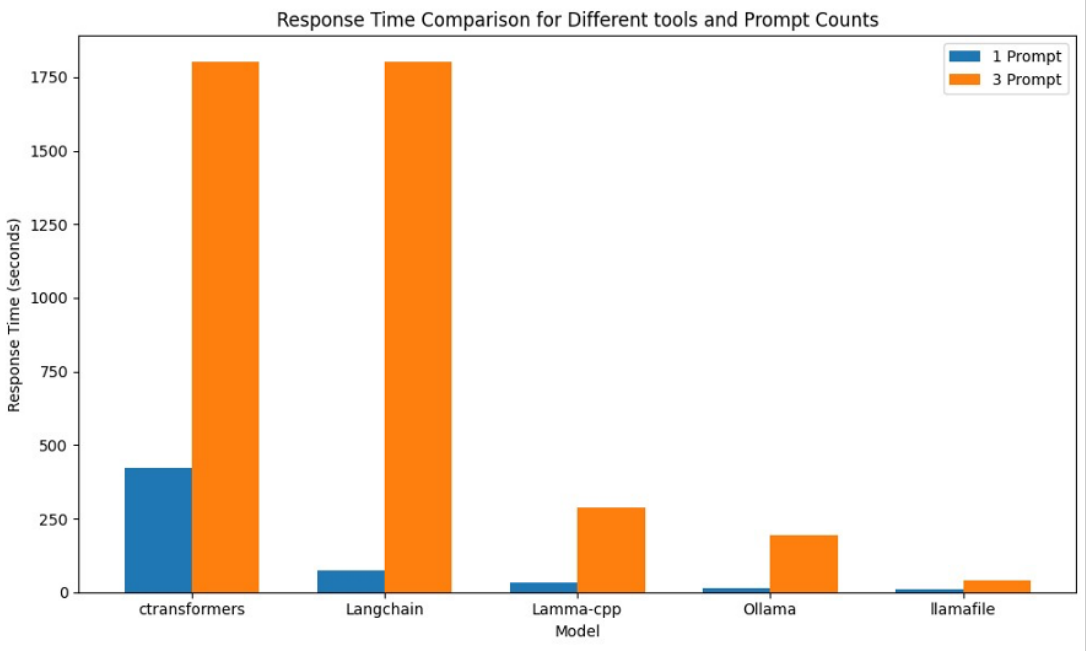

Figure 2: Response Time Comparison for Different Tools and Prompt Counts

Table 1: Single Prompt Performance Results

Tool	Response Time (s)	CPU Usage (%)	Memory Usage (%)
LangChain	73.82	15.4	94.6
Ctransformers	420.82	19.4	94.6
Llama-cpp	30.99	13.2	96.2
Ollama	13.42	28.9	99.2
Llamafile	**10.71**	16.3	98.3

Table 2: Multi-Prompt Performance Results

Tool	Response Time (s)	CPU Usage (%)	Memory Usage (%)
LangChain	1899.82	30.4	96.6
Ctransformers	1998.82	39.4	98.6
Llama-cpp	287.25	20.3	96.8
Ollama	193.24	20.9	99.2
Llamafile	**38.72**	24.6	97.4

5.2 Comparative Analysis

5.2.1 Response Time Performance

As demonstrated in Figure 2, Llamafile demonstrated superior performance with the fastest response times in both test scenarios. For single prompts, Llamafile achieved 10.71

seconds compared to the slowest performer (Ctransformers at 420.82 seconds), representing a 39× performance improvement. In multi-prompt scenarios, Llamafile completed tasks in 38.72 seconds versus 1800+ seconds for the slowest tools, achieving approximately 46× faster execution.

The visual comparison clearly illustrates the dramatic performance gap between tools, with Ctransformers and LangChain showing particularly poor multi-prompt performance, while Llamafile maintains consistently low response times across both scenarios.

5.2.2 Resource Utilization

Figure 1 provides a comprehensive view of all performance metrics. Memory usage remained consistently high across all tools (94-99%), indicating the fundamental memory requirements of LLM inference regardless of implementation. CPU usage varied significantly, with Llamafile maintaining moderate CPU consumption (16.3-24.6%) while achieving optimal performance, demonstrating efficient processor utilization.

The three-panel comparison reveals that while memory usage remains relatively consistent across tools, there are significant variations in CPU efficiency and dramatic differences in response times, highlighting the importance of tool selection for edge deployment scenarios.

5.2.3 Efficiency Analysis

The performance data reveals Llamafile's optimal balance between speed and resource consumption. While Ollama achieved competitive response times, it exhibited higher memory usage peaks. Llama-cpp provided moderate performance with lower CPU usage but significantly slower response times than the leading tools. The graphical representation clearly demonstrates why Llamafile emerged as the superior choice for resource-constrained edge environments.

5.3 Tool Selection Rationale

Based on comprehensive performance analysis, Llamafile emerged as the optimal choice for edge deployment due to:

Llamafile Advantages

- **Superior Speed:** Consistently fastest response times across all test scenarios
- **Balanced Resource Usage:** Efficient CPU utilization without excessive overhead
- **Deployment Simplicity:** Single-file executable requiring no complex installation
- **Cross-Platform Compatibility:** Unified solution across different edge hardware
- **Minimal Dependencies:** Reduced complexity for edge device deployment

The significant performance advantages of Llamafile, combined with its deployment simplicity, make it the recommended solution for edge LLM deployment in resource-constrained environments.

6 Edge Deployment Validation

Following the comprehensive tool comparison that identified Llamafile as the optimal inference solution, practical edge deployment validation was conducted. The quantized model and Llamafile tool were successfully deployed on resource-constrained edge devices, demonstrating real-world feasibility.

6.1 Key Achievements

Deployment Success Metrics

- **Functional Inference:** Complete LLM inference capability achieved on edge hardware
- **Memory Efficiency:** 4GB RAM requirement successfully met through quantization
- **Response Quality:** Generated outputs maintained coherence comparable to full-precision models
- **Stable Operation:** Sustained inference without thermal throttling or system instability

The deployment successfully executed both single-prompt and multi-prompt inference tasks, validating the scalability of the approach across different workload scenarios while operating within hardware constraints.

7 Conclusion

This research successfully demonstrated the feasibility of deploying Large Language Models on resource-constrained edge devices through quantization techniques and systematic tool evaluation. The comprehensive comparison of five inference tools revealed Llamafile as the optimal solution, achieving 39-46× superior performance compared to slower alternatives.