Comparing Quantization Strategies Across Hardware: A Comprehensive Study of LLM Inference on Edge Devices

Mike Doan[1], Jenny Dinh[1]

[1]Department of Computer Science, Georgia State University
Atlanta, GA, USA
mdoan@student.gsu.edu, jdinh@student.gsu.edu

Abstract

[TODO: Write abstract (150-250 words)]

The growing interest in local deployment of Large Language Models (LLMs) on edge devices necessitates efficient compression techniques to manage memory constraints and computational limitations. This paper presents a comprehensive empirical study comparing various quantization strategies across diverse hardware platforms, ranging from mobile devices to consumer-grade laptops. We systematically benchmark multiple quantization approaches (FP16, INT8, INT4, NF4) using different inference backends (llama.cpp, tinygrad) on models including Llama-3.2-1B and Qwen2.5-1.5B. Our evaluation encompasses both raw performance metrics (throughput, latency, memory usage) and downstream task accuracy using standardized benchmarks. We present a reproducible benchmarking framework with standardized data schemas, enabling community-driven expansion of our dataset. Our findings reveal... [key results]. The codebase and collected data are publicly available under MIT license.

Keywords: Large Language Models, Quantization, Edge Computing, Mobile Inference, Benchmarking

Introduction

Motivation

[TODO: Expand on the motivation for local LLM deployment]

The rapid advancement of Large Language Models (LLMs) has led to remarkable capabilities in natural language understanding and generation. However, the deployment of these models has traditionally been confined to cloud-based services due to their substantial computational and memory requirements. Recently, there has been growing interest in running LLMs locally on edge devices for several compelling reasons:

- **Privacy and Security**: Sensitive data remains on-device
- **Latency**: Elimination of network round-trips
- **Cost**: Reduced API costs for frequent inference
- **Availability**: Offline functionality in network-constrained environments

The primary challenge in local LLM deployment is the massive memory footprint. For instance, Llama-3.1-8B requires approximately 16GB of memory in FP16 precision, exceeding the capacity of most consumer devices and mobile platforms.

Problem Statement

[TODO: Define the specific research problem]

While quantization techniques have emerged as a solution to reduce model size, there exists limited systematic comparison of these techniques across diverse hardware platforms. Existing work typically focuses on:

- Single hardware platforms (e.g., NVIDIA GPUs only)
- Single quantization methods
- Synthetic benchmarks without downstream task evaluation

This fragmentation makes it difficult for practitioners to make informed decisions about quantization strategies for their target deployment scenarios.

Contributions

This work makes the following contributions:

1. **Comprehensive Benchmarking Framework**: We develop a reproducible, extensible benchmarking suite with standardized data schemas for cross-platform comparison.
2. **Cross-Platform Empirical Study**: We systematically evaluate multiple quantization strategies across diverse hardware (Apple Silicon, Android devices with ARM/Adreno) and inference backends (llama.cpp, tinygrad).
3. **Downstream Task Evaluation**: Beyond raw performance metrics, we assess the impact of quantization on actual task performance using standardized benchmarks (GSM8K, MATH, etc.).
4. **Open-Source Release**: We release our complete codebase, collected benchmark data, and analysis tools under MIT license to enable community contributions.

Paper Organization

The remainder of this paper is organized as follows: Section 2 reviews related work on quantization and edge deployment. Section 3 details our experimental methodology and benchmarking framework. Section 4 presents our experimental setup. Section 5 reports our results. Section 6 discusses implications and limitations. Section 7 concludes and outlines future work.

Background and Related Work

Large Language Models on Edge Devices

[TODO: Literature review on edge LLM deployment]

- Mobile deployment challenges
- Memory bandwidth limitations
- Power consumption constraints
- Previous work on on-device NLP models

Quantization Techniques

Weight Quantization

[TODO: Technical background on weight quantization]

Post-Training Quantization (PTQ): Quantization applied after model training without requiring access to training data or retraining.

Quantization-Aware Training (QAT): Models trained with quantization in mind, typically achieving better accuracy at lower precision.

Common Quantization Formats

[TODO: Expand on each format with citations]

- **INT8 (W8A8)**: 8-bit integer weights and activations
- **INT4 (W4A8)**: 4-bit weights, 8-bit activations
- **NF4**: 4-bit NormalFloat used in QLoRA [1]
- **GPTQ**: Group-wise post-training quantization [2]

- **AWQ**: Activation-aware weight quantization [3]

KV-Cache Quantization

[TODO: Discuss KV-cache compression techniques]

Key-value cache quantization reduces memory requirements during long-context inference.

Inference Frameworks

[TODO: Compare different inference frameworks]

llama.cpp

High-performance C++ implementation with GGUF model format. Supports multiple backends including Metal, OpenCL, CUDA.

tinygrad

Pure Python deep learning framework with emphasis on simplicity and hackability.

MLC-LLM

TVM-based compiler for deploying LLMs across platforms.

Benchmarking Methodologies

[TODO: Review existing benchmarking work]

- MLPerf Inference benchmarks
- Previous quantization studies
- Gap in cross-platform, cross-quantization comparisons

Methodology

Benchmarking Framework Design

Design Principles

Our framework is designed with the following principles:

1. **Reproducibility**: Seeded random generation, deterministic model loading
2. **Extensibility**: Modular design for adding new backends and devices
3. **Standardization**: Consistent data schema across all experiments
4. **Automation**: Minimal manual intervention required

Data Schema

[TODO: Explain the rationale behind the schema design]

We define a standardized schema for all benchmark runs:

```
class BenchmarkRow(TypedDict):
    step: int                      # Token generation step
    enqueue_latency_ms: float      # Time to enqueue operation
    total_latency_ms: float        # Total time for step
    tokens_per_sec: float          # Throughput
    memory_throughput_gb_s: float  # Memory bandwidth utilization
    param_throughput_gb_s: float   # Parameter throughput
    generated_text: str            # Incrementally generated text
    platform: str                  # OS (Darwin, Linux, Android)
    release: str                   # OS version
    device: str                    # Hardware device identifier
    username: str                  # User identifier
    hostname: str                  # Machine hostname
```

```
    size: str                    # Model size (1B, 3B, 8B)
    quantize: str                # Quantization strategy
    seed: int                    # Random seed
    uuid: str                    # Unique run identifier
```

This schema captures both performance metrics and environmental metadata necessary for reproducible analysis.

Benchmark Implementation

Backend-Specific Implementations

[TODO: Detail each backend implementation]

llama.cpp Backend

Implementation details:
- Model format: GGUF
- Quantization: Built into model files
- Metrics collection: Custom parsing of llama-cli output

tinygrad Backend

Implementation details:
- Server mode with OpenAI-compatible API
- Runtime quantization selection
- Direct metric extraction from framework

Metrics Collection

[TODO: Explain how each metric is computed]

Latency Metrics:
- Time to First Token (TTFT): Measured from prompt submission to first token
- Per-Token Latency: Time between consecutive token generations

Throughput Metrics:
- Tokens per second (tok/s)
- Memory bandwidth (GB/s): Computed from model size and throughput
- Parameter throughput (GB/s)

Memory Metrics:
- Model load size
- Peak memory utilization during inference

Downstream Task Evaluation

Verifiers Framework Integration

[TODO: Explain the verifiers framework and why it was chosen]

We integrate the Prime Intellect Verifiers framework [4] for downstream task evaluation. This framework provides:
- Standardized benchmark environments
- Consistent evaluation protocols
- OpenAI-compatible API interface

OpenAI Proxy Implementation

[TODO: Explain the proxy design]

To bridge non-streaming backends with the Verifiers framework, we implement an OpenAI-compatible proxy server that converts between streaming and non-streaming APIs.

Benchmark Tasks

[TODO: Describe each benchmark and what it measures]

- **GSM8K**: Grade school math problems (arithmetic reasoning)
- **MATH**: Higher-level mathematics (advanced reasoning)
- **GPQA**: Graduate-level science questions (domain knowledge)
- **SimpleQA**: Factual knowledge retrieval

Experimental Setup

Hardware Platforms

[TODO: Detail specifications of each device]

Device	Processor	Memory	GPU	Backend
MacBook (M-series)	Apple Silicon	16-32GB	Integrated	Metal
Google Pixel 7	Tensor G2	8GB	Mali-G710	OpenCL
Google Pixel 8	Tensor G3	8-12GB	Mali-G715	OpenCL

Table 1: Hardware platforms evaluated in this study

Model Selection

[TODO: Justify model choices]

We evaluate the following models:

- **Llama-3.2-1B-Instruct**: Small instruction-tuned model (1.2B parameters)
- **Qwen2.5-1.5B**: Multilingual model with strong performance (1.5B parameters)

These models were chosen for their:
- Size compatibility with edge devices
- Strong baseline performance
- Availability in multiple quantization formats

Quantization Strategies Evaluated

[TODO: Explain implementation details for each strategy]

Strategy	Bits	Description
FP16	16	Half-precision floating point (baseline)
INT8	8	8-bit integer quantization
INT4	4	4-bit integer quantization
NF4	4	4-bit NormalFloat (QLoRA)

Table 2: Quantization strategies under evaluation

Inference Configurations

[TODO: Document experimental parameters]

- **Context lengths**: 256, 512, 1024 tokens
- **Generation lengths**: 128, 256 tokens

- **Batch size**: 1 (single-query inference)
- **Temperature**: 0.7 (for downstream tasks)
- **Random seed**: 42 (for reproducibility)

Experimental Procedure

[TODO: Step-by-step procedure]

For each combination of (device, model, quantization, backend):

1. Load model and measure initialization time and memory
2. Warm-up: Generate 10 tokens to stabilize performance
3. Execute 100 token generations with standardized prompt
4. Record metrics for each token generation step
5. Compute aggregate statistics (mean, median, p95, p99)
6. For subset of configurations, run downstream task evaluation

Results

Performance Metrics

Throughput Analysis

[TODO: Present throughput results with figures]

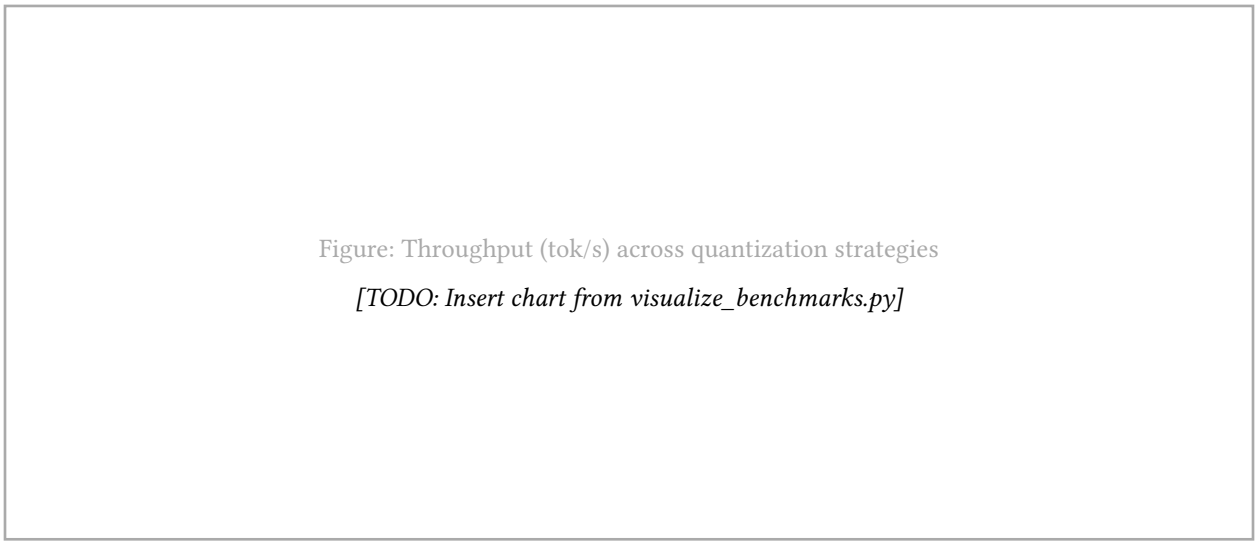

Figure 1: Token generation throughput across devices and quantization strategies

Key Findings:

- [TODO: Bullet points of key throughput findings]
- NF4 achieves X tok/s on MacBook M-series
- INT8 provides Y% improvement over FP16
- Diminishing returns observed on memory-bandwidth limited devices

Latency Analysis

[TODO: Present latency results]

Figure: Per-token latency distribution

[TODO: Insert latency distribution chart]

Figure 2: Per-token latency across quantization strategies (box plot)

Time to First Token (TTFT):
- [TODO: TTFT results]

Generation Latency:
- [TODO: Per-token latency results]

Memory Utilization
[TODO: Memory usage results]

Model	FP16	INT8	INT4	NF4
Llama-3.2-1B	TBD GB	TBD GB	TBD GB	TBD GB
Qwen2.5-1.5B	TBD GB	TBD GB	TBD GB	TBD GB

Table 3: Peak memory usage during inference

Downstream Task Performance

[TODO: Accuracy results from Verifiers benchmarks]

GSM8K Results

Quantization	Accuracy	Avg. Tokens	Time (s)	Score
FP16	TBD%	TBD	TBD	TBD
INT8	TBD%	TBD	TBD	TBD
NF4	TBD%	TBD	TBD	TBD

Table 4: GSM8K benchmark results across quantization strategies

Accuracy vs. Efficiency Trade-off
[TODO: Pareto frontier analysis]

Figure 3: Trade-off between downstream task accuracy and inference throughput

Cross-Device Comparison

[TODO: Compare performance across devices]

Device-Specific Observations

MacBook (Metal):

- [TODO: Key findings for MacBook]

Google Pixel (OpenCL):

- [TODO: Key findings for Pixel devices]

Backend Comparison

[TODO: Compare llama.cpp vs tinygrad]

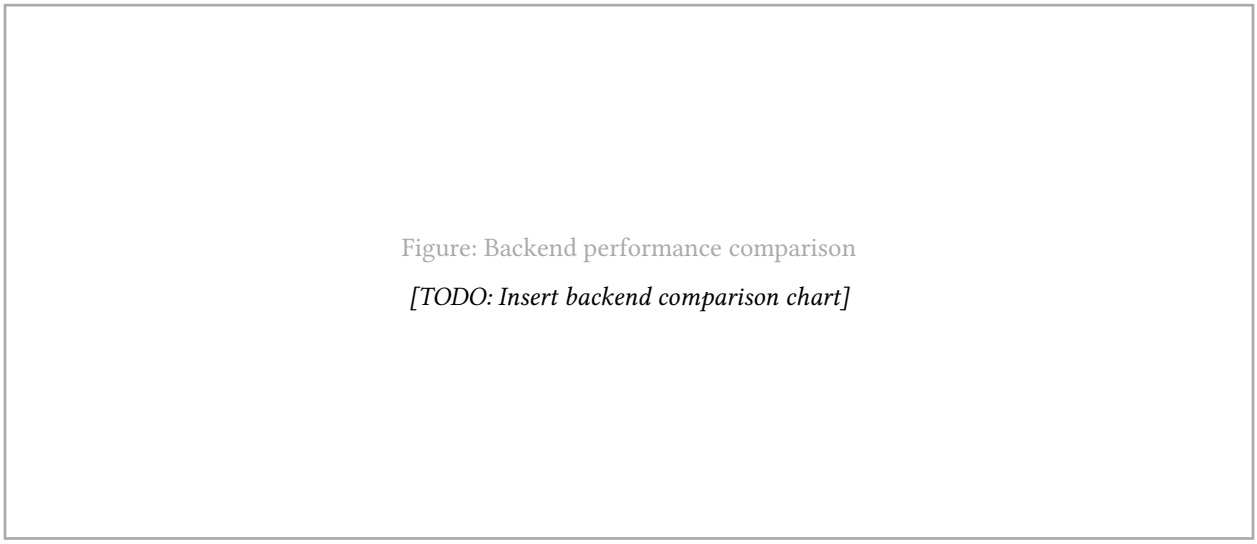

Figure 4: Performance comparison between llama.cpp and tinygrad backends

Discussion

Key Findings Summary

[TODO: Synthesize main results]

1. **Quantization enables practical edge deployment**: 4-bit quantization reduces memory footprint by 75% with acceptable accuracy degradation.
2. **Device-specific optimal strategies**: Memory-bandwidth characteristics heavily influence optimal quantization choice.
3. **Backend implementation matters**: Significant performance variations observed between inference frameworks.

Accuracy-Efficiency Trade-offs

[TODO: Discuss when to use which quantization]

Recommendations:
- **Memory-constrained devices** (< 8GB): NF4 or INT4
- **Accuracy-critical applications**: INT8 or FP16
- **Balanced deployment**: INT8 provides good compromise

Limitations

[TODO: Acknowledge limitations]

1. **Limited model sizes**: Focused on 1-1.5B parameter models
2. **Batch size 1**: Did not evaluate batched inference scenarios
3. **Specific hardware**: Limited to available devices
4. **Context lengths**: Did not extensively test long-context scenarios (>2K tokens)
5. **MLC-LLM excluded**: Build complexity prevented comprehensive evaluation

Implications for Practitioners

[TODO: Practical guidance]

This work provides practitioners with:
- Data-driven quantization selection criteria
- Realistic performance expectations for edge deployment
- Reproducible framework for evaluating new devices

Threats to Validity

[TODO: Discuss validity concerns]

Internal Validity:
- Variability in system background processes
- Thermal throttling on sustained workloads

External Validity:
- Generalization to larger models
- Different workload patterns (e.g., long conversations)

Construct Validity:
- Benchmark tasks may not represent all real-world use cases

Related Work

[TODO: Comprehensive related work section]

Quantization for LLMs
- QLoRA [1]
- GPTQ [2]
- AWQ [3]
- SmoothQuant
- LLM.int8()

Edge LLM Deployment
- MLC-LLM
- Mobile inference optimizations
- On-device NLP systems

Benchmarking Studies
- MLPerf Inference
- Previous quantization comparisons
- Mobile ML benchmarks

Conclusion and Future Work

Conclusion
[TODO: Summarize contributions and impact]

This paper presented a comprehensive empirical study of quantization strategies for LLM inference on edge devices. Through systematic benchmarking across multiple hardware platforms, models, and quantization approaches, we provide actionable insights for practitioners deploying LLMs locally. Our open-source framework enables reproducible research and community-driven expansion of performance data.

Key takeaways:
- 4-bit quantization (NF4/INT4) enables practical edge deployment with manageable accuracy trade-offs
- Hardware characteristics significantly influence optimal quantization strategy
- Standardized benchmarking frameworks are essential for meaningful cross-platform comparisons

Future Work
[TODO: Outline future research directions]

1. **Larger Models**: Extend evaluation to 3B, 7B, 8B parameter models
2. **Energy Profiling**: Comprehensive watt/token measurements and battery life impact
3. **KV-Cache Quantization**: Evaluate cache compression for long-context scenarios
4. **Batched Inference**: Multi-query performance on edge devices
5. **Enterprise GPUs**: Extend to A100, H100 for datacenter deployment guidance
6. **Quantization-Aware Training**: Compare PTQ vs QAT on edge devices
7. **Additional Backends**: Include MLC-LLM once build process is stabilized
8. **Production Deployment**: Real-world case studies and user studies

Reproducibility Statement
All code, data, and analysis scripts are available at: https://github.com/spikedoanz/t-eai-project

The repository includes:
- Complete benchmarking framework
- Data collection and parsing scripts
- Visualization and analysis code
- Raw benchmark data (CSV format)
- Setup instructions for all evaluated platforms

Acknowledgments

[TODO: Add acknowledgments]

We thank the developers of llama.cpp, tinygrad, and the Verifiers framework for their excellent open-source tools. We also acknowledge [funding sources, if any] and [individuals who provided feedback].

References

[1] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.

[2] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[3] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han, "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration," *arXiv preprint arXiv:2306.00978*, 2023.

[4] P. Intellect, "Verifiers: A framework for LLM evaluation." GitHub, 2024.