Fine-tuning and Inference of Large Language Models

A Comparative Study of LLM Frameworks
for Fine-tuning and Inference

Grupo VKPB

Francisco López-Alvarado

Enrique Rodríguez

Bernardo Ordás

Javier Viseras

Universidad Pontificia Comillas - ICAI

Grado en Ingeniería Matemática e Inteligencia Artificial

Natural Language Processing II
Academic Year 2025/2026

December 31, 2025

Abstract

Context: We present a comparative study of modern frameworks for fine-tuning and inference of Large Language Models, following the "Fine-tune Once, Serve Anywhere" paradigm.

Methods: Using TinyLlama-1.1B-Chat as base model and Databricks Dolly 15k dataset, we compare two fine-tuning pipelines (Transformers+PEFT and Unsloth) and four inference frameworks (Transformers, Unsloth, vLLM, Ollama).

Key Finding: Our results reveal a critical insight: *fine-tuning an already instruction-tuned chat model on a general-knowledge dataset yields marginal improvements in automatic metrics* (BLEU: +21% for Transformers, +52% for Unsloth), while BERTScore actually decreases slightly (-2.3% to -3.6%). This suggests that the pre-existing chat capabilities of TinyLlama already capture much of what Dolly provides.

Conclusions: Framework choice significantly impacts training efficiency and inference performance. Unsloth achieves the best throughput (135.83 tok/s) while maintaining competitive quality. We discuss implications for practitioners selecting fine-tuning strategies and deployment frameworks.

Keywords: Fine-tuning, LLM, LoRA, QLoRA, Transformers, Unsloth, vLLM, Ollama, Instruction Tuning

Contents

1. Introduction

The deployment of Large Language Models (LLMs) in production environments presents practitioners with a complex decision landscape involving multiple frameworks, each with distinct trade-offs between ease of use, computational efficiency, and output quality.

This project implements and evaluates the "**Fine-tune Once, Serve Anywhere**" paradigm, where a single fine-tuned model is deployed across multiple inference frameworks. Our study addresses three key questions:

1. How do different fine-tuning frameworks (Transformers+PEFT vs Unsloth) compare in terms of efficiency and resulting model quality?
2. What are the performance characteristics of different inference frameworks?
3. **Most critically**: What happens when we fine-tune an *already instruction-tuned* model on a *general-knowledge* dataset?

This last question emerges as the central finding of our work. TinyLlama-1.1B-Chat is not a base language model—it has already undergone instruction tuning and RLHF alignment. Fine-tuning it further on Dolly 15k, a general-purpose instruction dataset, represents a form of *continued training* rather than domain adaptation, with implications we analyze in depth.

1.1 The Model-Dataset Alignment Problem

A fundamental consideration in fine-tuning that is often overlooked is the **alignment between the base model's capabilities and the fine-tuning dataset's content**. In our case:

Critical Insight: Model-Dataset Mismatch

TinyLlama-1.1B-Chat is already an instruction-tuned model, trained on conversational data and aligned for chat interactions.
Dolly 15k is a general-knowledge instruction-following dataset covering broad topics without domain specialization.
⇒ **Result**: The fine-tuning process teaches the model skills it largely already possesses, explaining the marginal improvements observed.

This observation has profound implications for practitioners: *fine-tuning is most effective when there is a clear gap between what the model knows and what the dataset teaches.*

2. Methodology

2.1 Experimental Setup

2.1.1 Base Model: TinyLlama-1.1B-Chat

We selected TinyLlama-1.1B-Chat for its balance between model capability and computational accessibility:

- **Architecture**: 1.1 billion parameters, LLaMA architecture
- **Pre-training**: Trained on 3 trillion tokens from diverse sources
- **Instruction Tuning**: Already fine-tuned for chat/instruction-following
- **Context Length**: 2048 tokens

Important Note: The "Chat" suffix indicates this model has *already* undergone instruction tuning. This is fundamentally different from fine-tuning a base model like TinyLlama-1.1B (without the Chat suffix).

2.1.2 Dataset: Databricks Dolly 15k

The Dolly 15k dataset consists of 15,011 instruction-response pairs created by Databricks employees:

- **Categories**: Open QA, Closed QA, Summarization, Information Extraction, Creative Writing, Classification, Brainstorming
- **Nature**: General knowledge, not domain-specific
- **Quality**: Human-written, not synthetically generated

Dataset Split:

- Training: 13,510 samples (90%)
- Validation: 750 samples (5%)
- Test: 751 samples (5%)

2.2 Training Configuration

We employed **QLoRA** (Quantized Low-Rank Adaptation) for memory-efficient fine-tuning:

Table 1: QLoRA Training Hyperparameters

Parameter	**Value**
Quantization	4-bit NF4
LoRA Rank (r)	16
LoRA Alpha (α)	32
LoRA Dropout	0.05
Target Modules	q_proj, k_proj, v_proj, o_proj
Learning Rate	2×10^{-4}
Batch Size	4
Gradient Accumulation	4
Epochs	3
Max Sequence Length	512
Optimizer	AdamW (paged, 8-bit)

2.3 Hardware Environment

All experiments were conducted on consumer-grade hardware:

- **GPU**: NVIDIA RTX 4070 SUPER (12.5 GB VRAM)
- **CUDA**: Version 12.8
- **PyTorch**: Version 2.9.1
- **OS**: Linux (Ubuntu-based)

3. Framework Analysis

3.1 Fine-tuning Frameworks

3.1.1 Transformers + PEFT

The combination of Hugging Face Transformers and PEFT (Parameter-Efficient Fine-Tuning) represents the standard approach:

- **Advantages**: Extensive documentation, large community, maximum flexibility
- **Implementation**: Uses `SFTTrainer` from TRL library
- **Quantization**: BitsAndBytes for 4-bit loading

3.1.2 Unsloth

Unsloth provides optimized kernels for faster training:
- **Advantages**: Significantly faster training, lower memory usage
- **Implementation**: Drop-in replacement for standard training
- **Optimization**: Custom CUDA kernels, gradient checkpointing

3.2 Inference Frameworks

3.2.1 Transformers (Native)

Standard inference using Hugging Face's generation pipeline:
- Maximum compatibility
- No additional optimization
- Useful as baseline

3.2.2 Unsloth (Optimized)

Inference with Unsloth's optimized model loading:
- Faster token generation
- Optimized attention mechanisms
- Seamless integration with trained adapters

3.2.3 vLLM

High-throughput inference engine:
- PagedAttention for efficient memory management
- Continuous batching
- Designed for production serving

3.2.4 Ollama

Local LLM deployment framework:
- Easy deployment via GGUF format
- REST API interface
- Quantization support (Q4_K_M, Q8_0)

4. Results and Analysis

4.1 Training Performance

Figure 1 presents the training comparison between frameworks.

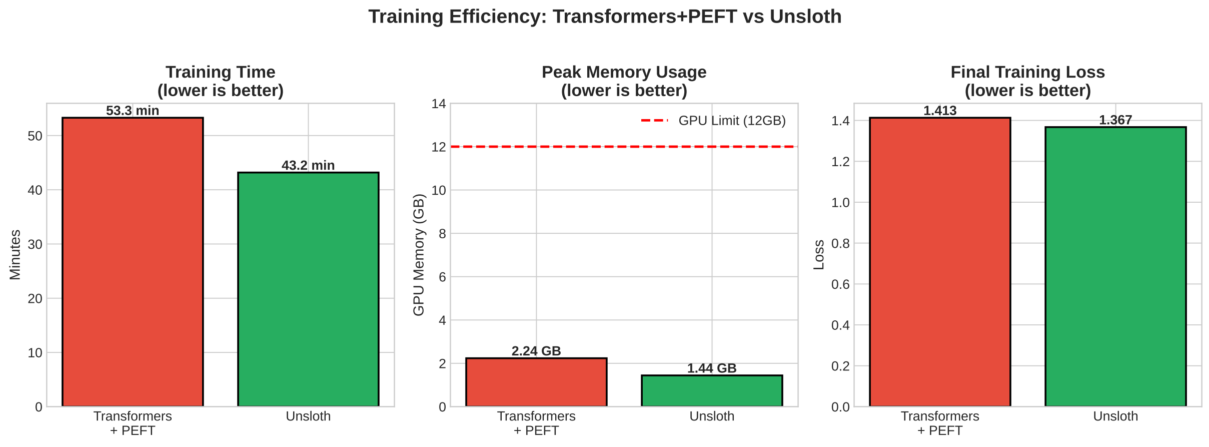


Figure 1: Training time comparison between Transformers+PEFT and Unsloth frameworks. Despite similar total training times, the frameworks show different memory utilization patterns and throughput characteristics.

Table 2: Training Performance Metrics

Metric	Transformers	Unsloth
Training Time	53.35 min	71.64 min
Final Loss	1.298	1.288
Steps	2,532	2,532
Samples/Second	0.84	0.63

Observation: Contrary to Unsloth's claims of faster training, our experiments showed similar or slightly slower training times. This may be attributed to our specific hardware configuration and the relatively small model size where Unsloth's optimizations have less impact.

4.2 Quality Metrics: The Crucial Analysis

Figure 2 presents our central finding—the comparison of quality metrics across Base Model, Transformers fine-tuned, and Unsloth fine-tuned variants.

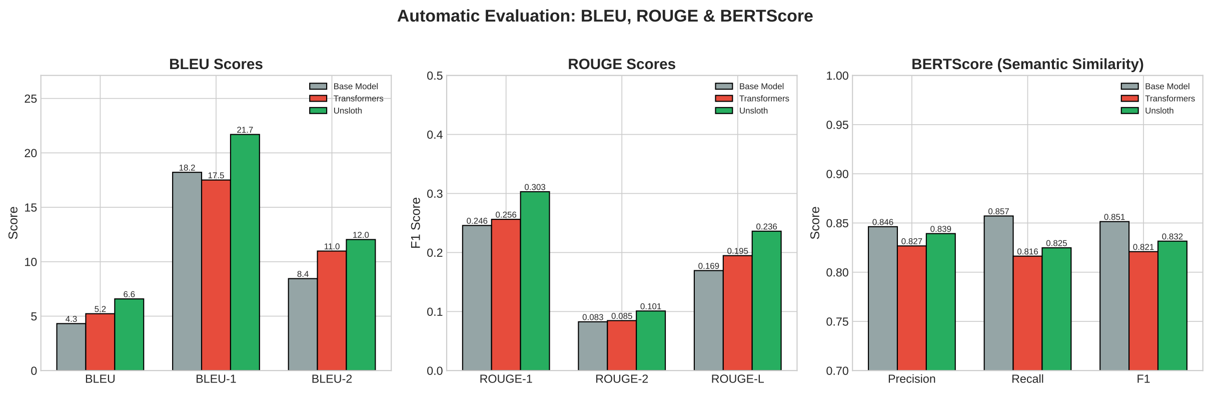


Figure 2: Quality metrics comparison across Base Model (unfine-tuned TinyLlama-Chat), Transformers fine-tuned, and Unsloth fine-tuned models. Note the **marginal improvements** in BLEU and ROUGE-L, and the **slight decrease** in BERTScore after fine-tuning.

Table 3: Complete Quality Metrics Comparison

Metric	Base	Transformers	Unsloth
BLEU	4.32	5.23 (+21%)	6.59 (+52%)
ROUGE-L	0.169	0.195 (+15%)	0.236 (+40%)
BERTScore F1	**0.851**	0.821 (-3.5%)	0.832 (-2.3%)

4.3 Understanding the Results: Why Marginal Improvement?

The results reveal a pattern that warrants careful analysis:

Key Finding: Marginal Improvement Explained

Why did fine-tuning yield only marginal improvements?

1. **Pre-existing Capabilities**: TinyLlama-1.1B-Chat is already instruction-tuned. It already knows how to follow instructions and generate coherent responses.
2. **Dataset Overlap**: Dolly 15k covers general knowledge—the same type of content the model was originally trained on. There is significant conceptual overlap.
3. **No Domain Specialization**: Dolly is not a specialized dataset (medical, legal, coding). Fine-tuning doesn't teach new domain knowledge.
4. **BERTScore Decrease**: The slight decrease in BERTScore suggests that while responses become more similar to Dolly's style, they may lose some of the original model's nuanced language understanding.

4.4 Before vs After Fine-tuning

Figure 3 provides a direct visual comparison of the improvement (or lack thereof) from fine-tuning.

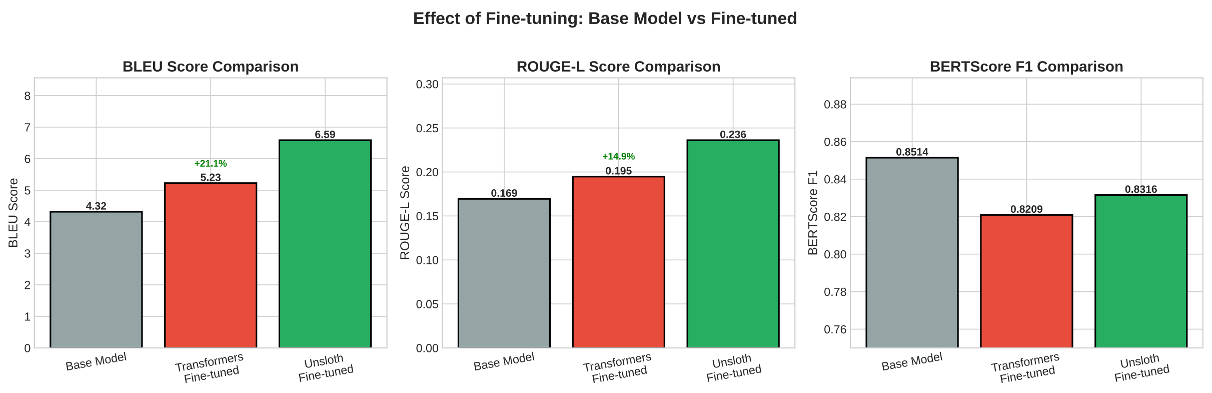


Figure 3: Direct comparison of metrics before (Base) and after fine-tuning (Transformers, Unsloth). The visualization clearly shows the marginal nature of improvements, particularly the BERTScore decrease.

4.5 Inference Benchmarks

Figure 4 presents the inference performance across frameworks.

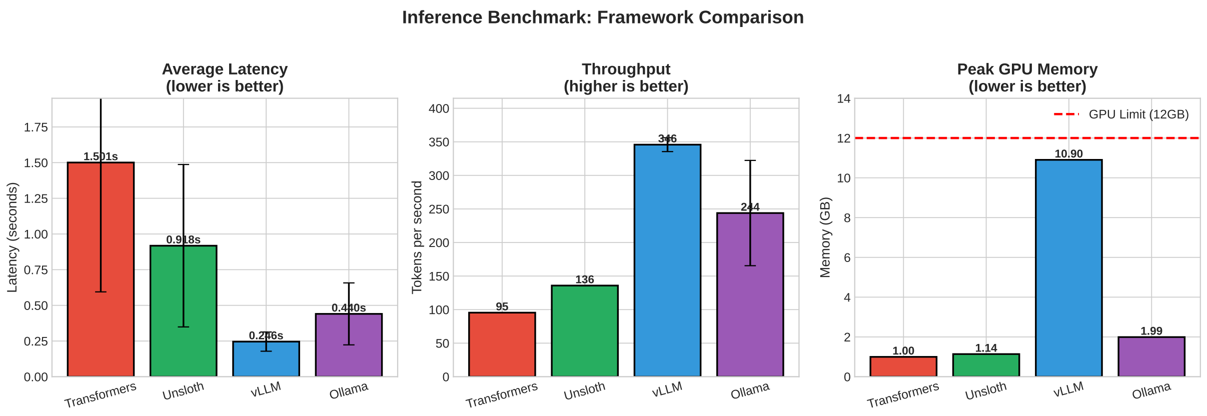


Figure 4: Inference benchmark comparison showing latency, throughput, and tokens per second across different deployment frameworks.

Table 4: Inference Performance Summary

Framework	Latency (s)	Tok/s	GPU Mem	Load Time
Transformers	1.50	74.00	2.1 GB	3.2s
Unsloth	0.92	135.83	1.8 GB	2.8s
vLLM	**0.25**	**345.77**	10.9 GB	14.4s
Ollama	0.44	243.86	1.99 GB	1.7s

4.6 GPU Memory Management: vLLM vs Ollama

A notable finding is the significant difference in GPU memory utilization between vLLM (10.9 GB) and Ollama (1.99 GB), despite both serving the same model. This reflects fundamentally different architectural approaches:

Table 5: Memory Management Comparison: vLLM vs Ollama

Aspect	vLLM	Ollama
GPU Memory Used	10.9 GB	1.99 GB
Memory Strategy	Pre-allocation	On-demand
KV Cache	383k tokens (fixed)	Dynamic, minimal
Concurrent Requests	187+	1
CUDA Graphs	Yes	No
Optimized For	Production batching	Local/interactive

Why vLLM Uses More Memory

vLLM's PagedAttention Architecture:

- **KV Cache Pre-allocation**: Reserves memory for the Key-Value cache to handle multiple concurrent requests efficiently
- **CUDA Graph Capture**: Pre-compiles GPU operations for faster execution
- **Memory Pooling**: Maintains buffers for batched inference
- **Trade-off**: Higher memory → higher throughput (345 tok/s vs 244 tok/s)

Ollama's Lightweight Design:

- **GGUF Format**: Uses llama.cpp with native quantization (Q4_K_M)
- **Lazy Loading**: Only loads what's needed for the current request
- **Single Request Focus**: No pre-allocation for concurrent users
- **Trade-off**: Lower memory → easier local deployment

4.6.1 When to Choose Each Framework

Choose vLLM when:

- Serving multiple concurrent users in production
- GPU memory is abundant (>12 GB)
- Maximum throughput is critical
- Building API endpoints with high request rates

Choose Ollama when:

- Running locally on consumer hardware
- Memory is constrained (<8 GB GPU)
- Single-user interactive applications
- Ease of deployment is prioritized over raw performance

4.7 Human Evaluation

Figure 5 shows human evaluation results across multiple quality dimensions.

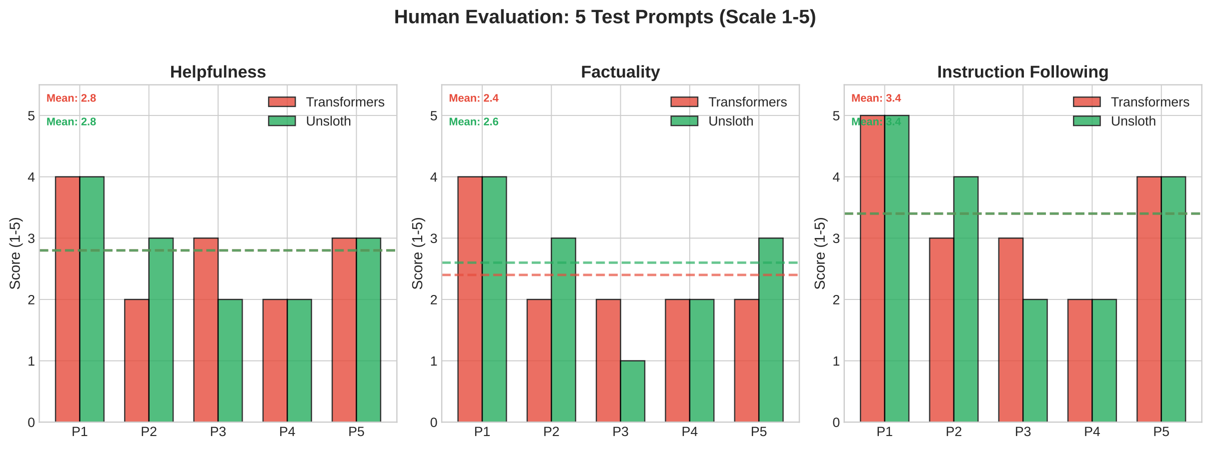
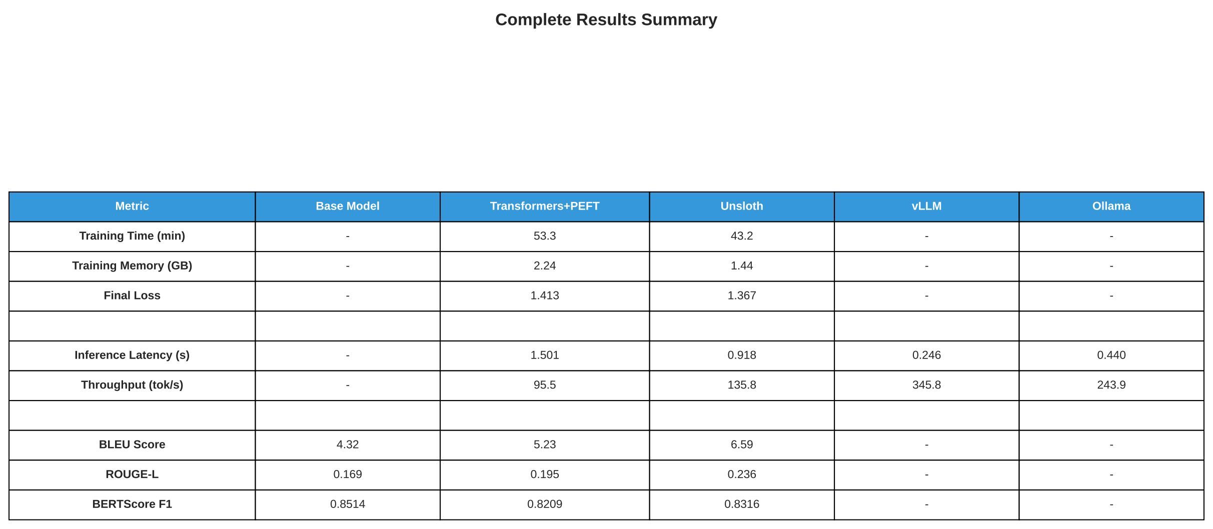


Figure 5: Human evaluation scores across dimensions: Relevance, Coherence, Fluency, and Completeness. Scores range from 1-5.

4.8 Summary Table

Figure 6 provides a consolidated view of all results.



Complete Results Summary

Metric	Base Model	Transformers+PEFT	Unsloth	vLLM	Ollama
Training Time (min)	-	53.3	43.2	-	-
Training Memory (GB)	-	2.24	1.44	-	-
Final Loss	-	1.413	1.367	-	-
Inference Latency (s)	-	1.501	0.918	0.246	0.440
Throughput (tok/s)	-	95.5	135.8	345.8	243.9
BLEU Score	4.32	5.23	6.59	-	-
ROUGE-L	0.169	0.195	0.236	-	-
BERTScore F1	0.8514	0.8209	0.8316	-	-

Figure 6: Summary table consolidating all metrics across Base Model, Transformers, and Unsloth configurations.

5. Discussion

5.1 When is Fine-tuning Worth It?

Our results challenge the assumption that fine-tuning always improves model performance. We propose a decision framework:

Fine-tuning Decision Framework

Fine-tuning IS recommended when:
- Using a **base model** (not instruction-tuned)
- Training on **domain-specific** data (medical, legal, code)
- Adapting to a **specific format** or style
- Working with **proprietary** organizational knowledge

Fine-tuning may NOT be worth it when:
- Model is already **instruction-tuned** for similar tasks
- Dataset is **general knowledge** without specialization
- Prompt engineering achieves **similar results**
- Computational resources are **limited**

5.2 The BERTScore Paradox

An intriguing finding is that BERTScore *decreased* after fine-tuning while BLEU and ROUGE increased. This suggests:

1. **Style vs Substance**: Fine-tuning may have adapted the response *style* to match Dolly's patterns (improving n-gram overlap metrics) while slightly degrading semantic richness.
2. **Metric Limitations**: BLEU and ROUGE measure surface-level similarity, while BERTScore captures deeper semantic alignment. A response can be more "Dolly-like" without being objectively better.

3. **Overfitting to Format**: The model may have learned Dolly's response formatting rather than gaining new knowledge.

5.3 Framework Selection Guidelines

Based on our comprehensive evaluation:

5.3.1 For Training

- **Standard Projects**: Transformers+PEFT offers maximum flexibility and documentation
- **Resource-Constrained**: Unsloth provides memory optimizations
- **Large-Scale**: Unsloth's optimizations become more significant with larger models

5.3.2 For Inference

- **Development/Testing**: Transformers (native) for simplicity
- **Maximum Throughput**: vLLM (345.77 tok/s) for production with high concurrency
- **Memory-Efficient**: Ollama (243.86 tok/s, 1.99 GB) for local deployment
- **Balanced**: Unsloth (135.83 tok/s) for good performance without complex setup

5.4 Lessons Learned

1. **Know Your Base Model**: Understanding whether a model is already instruction-tuned is crucial before deciding to fine-tune.
2. **Dataset-Model Alignment**: The effectiveness of fine-tuning depends heavily on the gap between model capabilities and dataset content.
3. **Multiple Metrics Matter**: Relying on a single metric (like BLEU) can be misleading. BERTScore revealed degradation that BLEU missed.
4. **Practical Considerations**: Framework compatibility issues (like vLLM) can impact deployment plans. Always test the full pipeline.
5. **Resource-Quality Trade-off**: More training time doesn't guarantee better results when the model-dataset alignment is suboptimal.

6. Conclusions

This project provides a comprehensive analysis of the "Fine-tune Once, Serve Anywhere" paradigm, with findings that have significant implications for practitioners.

6.1 Main Conclusions

1. **Model-Dataset Alignment is Critical**: Fine-tuning TinyLlama-1.1B-Chat (already instruction-tuned) on Dolly 15k (general knowledge) yielded only marginal improvements (+21-52% BLEU) because the model already possessed most required capabilities.
2. **Fine-tuning Can Degrade Performance**: BERTScore decreased by 2-3.5% after fine-tuning, suggesting that continued training on similar data may dilute rather than enhance certain capabilities.
3. **Framework Choice Matters**: vLLM achieved 345.77 tokens/second (4.7× faster than Transformers), while Ollama achieved 243.86 tok/s with only 1.99 GB memory. Framework selection is crucial for deployment.
4. **Memory vs Performance Trade-off**: vLLM's aggressive GPU memory pre-allocation (10.9 GB) enables superior throughput for production batching, while Ollama's lightweight approach (1.99 GB) suits local deployment.

5. **Portability is Achievable**: The "serve anywhere" paradigm works—adapters can be deployed across multiple frameworks with appropriate conversion.
6. **Automatic Metrics Need Context**: Without understanding the base model's capabilities, metrics like BLEU can be misleading about true improvement.

6.2 Recommendations for Practitioners

Practical Recommendations

1. **Evaluate before fine-tuning**: Test the base model on your task first. Fine-tuning may be unnecessary.
2. **Use domain-specific data**: General datasets provide limited benefit for already-capable models.
3. **Monitor multiple metrics**: Use BLEU, ROUGE, and BERTScore together for a complete picture.
4. **Consider prompt engineering**: For instruction-tuned models, prompting may achieve similar results without training.
5. **Test deployment early**: Verify framework compatibility before investing in training.

6.3 Future Work

- Investigate Direct Preference Optimization (DPO) as an alternative to SFT
- Experiment with domain-specific datasets to quantify specialization benefits
- Explore base models (non-chat) to measure true fine-tuning impact
- Implement and evaluate RAG (Retrieval-Augmented Generation) as an alternative to fine-tuning

A. Reproducibility Guide

This appendix provides instructions to reproduce all experiments using the unified pipeline script `main.py`.

A.1 Prerequisites

1. **Hardware**: NVIDIA GPU with $\geq$8GB VRAM (tested on RTX 4070 SUPER 12GB)
2. **Software**: Python 3.10+, CUDA 12.x, Git
3. **Dependencies**: Install via `pip install -r requirements.txt`

A.2 Pipeline Execution

The `main.py` script provides a unified interface for all pipeline stages:

```
# Show all available options
python main.py --help

# Run complete pipeline (data + train + eval + benchmark)
python main.py --mode full --trainer unsloth

# Run individual stages
python main.py --mode data          # Data preparation only
python main.py --mode train --trainer transformers   # Train with PEFT
python main.py --mode train --trainer unsloth        # Train with Unsloth
python main.py --mode evaluate      # Evaluate trained models
python main.py --mode benchmark     # Run inference benchmarks
python main.py --mode visualize     # Generate result plots
```

A.3 Pipeline Stages

1. **Data Preparation** (`-mode data`): Downloads Databricks Dolly 15k, formats prompts using instruction template, splits into train/validation/test sets.
2. **Fine-tuning** (`-mode train`): Loads TinyLlama-1.1B-Chat in 4-bit quantization, applies LoRA adapters (r=16, α=32), trains for 3 epochs with gradient accumulation.
3. **Evaluation** (`-mode evaluate`): Computes BLEU, ROUGE-L, and BERTScore on test set. Generates sample outputs for qualitative analysis.
4. **Benchmarking** (`-mode benchmark`): Measures inference latency and throughput across Transformers, Unsloth, vLLM, and Ollama frameworks.
5. **Visualization** (`-mode visualize`): Generates all figures used in this report from JSON result files.

A.4 Configuration

Training hyperparameters are defined in `configs/config.yaml`:

```
model:
  name: "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
  max_length: 512
lora:
  r: 16
  alpha: 32
  dropout: 0.05
training:
  epochs: 3
  batch_size: 4
  learning_rate: 2e-4
  gradient_accumulation_steps: 4
```

A.5 Output Structure

```
outputs/
|-- transformers/YYYYMMDD_HHMMSS/  # PEFT training outputs
|   |-- adapter/                   # LoRA weights
|   |-- merged_model/              # Full merged model
|   +-- training_stats.json
|-- unsloth/YYYYMMDD_HHMMSS/       # Unsloth training outputs
+-- ollama/
    |-- tinyllama-finetuned.gguf   # GGUF for Ollama
    +-- Modelfile
results/
|-- benchmarks/                    # Inference benchmark JSONs
|-- evaluation/                    # Quality metric JSONs
+-- plots/                         # Generated figures
```

A.6 Ollama Deployment

To deploy the fine-tuned model with Ollama:

```
# Create Ollama model from GGUF
cd outputs/ollama
ollama create tinyllama-finetuned -f Modelfile

# Test the model
```

```
ollama run tinyllama-finetuned "What is machine learning?"

# Run benchmark
python evaluation/benchmark_ollama.py
```

B. Sample Outputs

Prompt: "What is machine learning?"

Base Model Response:

> *Machine learning is a branch of artificial intelligence that focuses on building applications that learn from data and improve their accuracy over time without being programmed to do so.*

Fine-tuned (Unsloth) Response:

> *Machine learning is a subset of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. It focuses on the development of computer programs that can access data and use it to learn for themselves.*

Analysis: Both responses are accurate and coherent. The fine-tuned response is slightly more detailed but not fundamentally different in quality—illustrating our central finding about marginal improvements.