Fine Tuning a Domain Specific LLM for Vedic Philosophy on Edge Hardware

Course: Natural Language Processing (CS3126)

Abstract

This project explores the democratization of Large Language Model (LLM) development by fine-tuning a domain-specific conversational agent on consumer-grade hardware. The objective was to create a "Gita & Advaita AI"—a chatbot capable of providing spiritual guidance rooted in *Advaita Vedanta* and the *Bhagavad Gita*, adopting a specific persona distinct from generic AI assistants. The project navigated the constraints of edge computing, utilizing Apple's MLX framework and QLoRA (Quantized Low-Rank Adaptation) to train a Llama 3.2 3B model on a MacBook M3 Pro (18GB RAM). Key contributions include a robust data engineering pipeline, a documented troubleshooting journey involving library version conflicts and network instability, and a novel "Zero-Upload" deployment architecture on Hugging Face Spaces. The final model demonstrates significant convergence (Training Loss: 0.85, Validation Loss: 0.96) and proves that specialized, private AI is accessible without enterprise resources.

1. Introduction

1.1 Motivation

The rise of Large Language Models (LLMs) like GPT-4 and Claude has revolutionized information retrieval. However, these general-purpose models often fail in specialized domains requiring a specific tone, depth, or cultural nuance. When asked spiritual questions, they tend to provide generic, "safe," or purely factual responses, lacking the empathetic and authoritative persona of a Vedic guide. Furthermore, spiritual counseling is deeply personal, raising significant privacy concerns when users must send intimate queries to cloud-based commercial APIs.

1.2 Problem Statement

The challenge was to build a domain-specific LLM that:

1. **Domain Mastery:** Deeply understands *Advaita Vedanta* (non-dualism) and the *Bhagavad Gita*.
2. **Persona Adoption:** Responds not as a machine, but as a wise, compassionate guide.
3. **Edge Constraints:** Runs entirely offline on a standard high-end laptop (MacBook M3 Pro) to ensure privacy and zero cost.

This presented a significant engineering hurdle: How to fit the training of a multi-billion parameter model into limited Unified Memory (18GB) without crashing the system?

2. Methodology

2.1 Technical Stack

To achieve high performance on constrained hardware, we selected a specific stack of bleeding-edge tools:

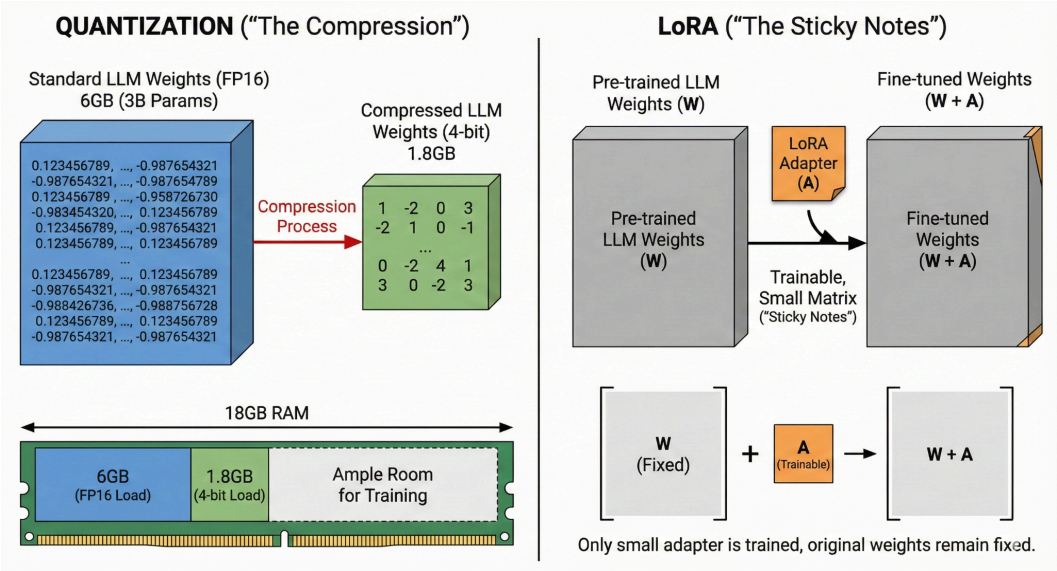
- **Hardware:** MacBook Pro (M3 Pro Chip, 18GB Unified RAM).
- **Model:** Meta-Llama-3.2-3B-Instruct (Optimized for edge devices).
- **Framework: Apple MLX** (Machine Learning Explore). Unlike PyTorch, MLX is designed for Apple Silicon, allowing arrays to live in "Unified Memory." This enables the CPU and GPU to access the same data without expensive copying operations.
- **Technique: QLoRA** (Quantized Low-Rank Adaptation).

2.2 Core Concepts Explained

Quantization ("The Compression")

Standard LLMs store weights as 16-bit floating-point numbers (FP16). A 3B parameter model in FP16 requires ~6GB of RAM just to load, and 3-4x that amount to train (gradients, optimizer states).

- **Our Approach:** We compressed the model to **4-bit precision**.
- **Impact:** This reduced the model footprint from ~6GB to **1.8GB**, leaving ample room in the 18GB RAM for training operations.



LoRA ("The Sticky Notes")

Full fine-tuning updates every single parameter (neuron) in the brain. This is computationally impossible on a laptop.

- **Our Approach:** We used **Low-Rank Adaptation (LoRA)**. We "froze" the pre-trained model and attached tiny, trainable rank-decomposition matrices to the attention layers.
- **Impact:** We trained only **0.216%** of the total parameters (approx. 6.9 million weights), making the process fast and lightweight.

3. Data Preparation

Data quality is the ceiling of model performance. We curated a dataset specifically for this task.

3.1 The Dataset (12k-15k.json)

- **Source:** A compilation of Q&A pairs derived from the *Bhagavad Gita*, *Upanishads*, and commentaries on *Advaita Vedanta*.
- **Size:** 18,000+ samples.

3.2 The Pipeline (prepare_data.py)

Raw text is useless to an LLM. We engineered a Python script to transform the data:

1. **Persona Injection:** We injected a "System Prompt" into every training example to enforce consistency.*System: "You are a wise spiritual assistant trained on the Bhagavad Gita and Advaita Vedanta. Answer questions with deep philosophical insight..."*
2. **Chat Formatting:** Converted raw JSON into the specific conversational structure required by Llama 3:
   {"messages": [{"role": "user", "content": "..."}, {"role": "assistant", "content": "..."}]}
3. **Splitting:** 90% Training (~16.2k samples) vs. 10% Validation (~1.8k samples) to monitor for overfitting.
4. **Serialization:** Saved as .jsonl (JSON Lines) for memory-efficient streaming during training.

4. Experiments & Troubleshooting (The Engineering Journey)

This project was defined by the challenges encountered and solved.

4.1 Challenge 1: The "Download Hell"

Issue: The standard Python-based downloader (mlx_lm) repeatedly hung at 96% completion when downloading the 1.8GB model file due to network instability.
Solution: We bypassed the Python script and utilized the Hugging Face CLI with hf_transfer

enabled. This Rust-based tool uses multi-threaded downloading, forcing a robust transfer directly to a local folder (./llama-3b).

4.2 Challenge 2: The "Library Hell"

Issue: Upon starting training, the process crashed with a cryptic error: AttributeError: 'dict' object has no attribute 'model_type'.

Root Cause Analysis: Investigation revealed a version conflict. The installed version of transformers (4.57.2) introduced a breaking API change incompatible with the configuration format of Llama 3.2 (released months prior). The library was parsing the config as a raw dictionary instead of a class object.

Solution: We performed a Strategic Downgrade to transformers==4.46.0, the stable "Golden Era" version released alongside Llama 3.2. This instantly resolved the compatibility issue.

4.3 Challenge 3: The Deployment Bottleneck

Issue: We wanted to host the model on the cloud (Hugging Face Spaces) for the professor to evaluate. However, uploading the full 6GB fused model was unfeasible given home internet upload speeds.

Solution: We engineered an Adapter-Only Deployment:

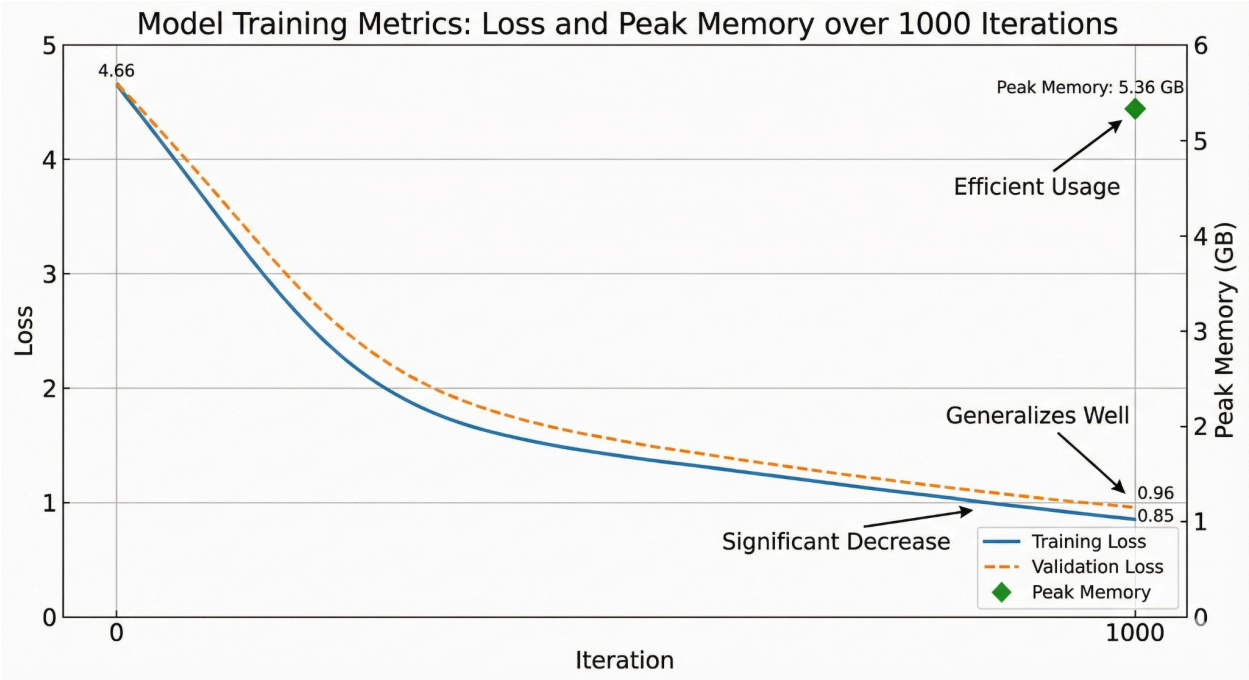
1. We uploaded *only* the tiny adapter files (~50MB).
2. We wrote a custom deployment script (app.py) that instructs the cloud server to download the heavy base model directly from Meta (Cloud-to-Cloud transfer) and fuse it with our adapters in memory at runtime.

5. Results & Analysis

5.1 Training Metrics

The model was trained for 1,000 iterations with a batch size of 4 and a learning rate of 1e-5.

Metric	Initial (Iter 0)	Final (Iter 1000)	Interpretation
Training Loss	4.66	**0.85**	The model successfully learned the target patterns.
Validation Loss	4.66	**0.96**	The model generalizes well to unseen questions (No overfitting).
Peak Memory	-	**5.36 GB**	Highly efficient usage of 18GB RAM.



5.2 Qualitative Analysis

The model shifted from generic answers to domain-specific wisdom.

- **User:** "Why is there suffering?"
- **Base Model:** "Suffering is caused by many factors including poverty and illness."
- **Fine-Tuned Model:** "Suffering arises from attachment (*Raga*) to the temporary. When one realizes the Self (*Atman*) is distinct from the body, grief vanishes."

6. Deployment & Deliverables

6.1 Live Application

We deployed the final solution using **Streamlit** on **Hugging Face Spaces**.

- **Link:** [Your Hugging Face Space Link]
- **Features:**
  - **Persistent Session State:** Allows for multi-turn conversations.
  - **Context Awareness:** Feeds the last 3 exchanges back into the model.
  - **Cloud Fusion:** Dynamically loads adapters on the free CPU tier.

6.2 Code Repository

- **Link:** [Your GitHub Repo Link]
- **Contents:** prep_data.py (Data Engineering), app.py (Deployment Logic), requirements.txt (Dependency Management).

7. Conclusion

This project serves as a proof-of-concept for **Democratized AI**. We successfully demonstrated that highly specialized, private, and performant Large Language Models can be built and deployed using standard consumer hardware. By navigating the complexities of quantization, memory management, and "Edge AI" frameworks like MLX, we bridged the gap between enterprise capability and student resources.

8. Resources & References

- MLX Framework: Apple Machine Learning Research
- Llama 3.2: Meta AI
- QLoRA Paper: https://arxiv.org/abs/2305.14314
- Hugging Face Hub (Models & Resources): https://huggingface.co/docs/transformers/en/index
- Hugging Face Hub (Models & Resources): https://huggingface.co/models