J. Moeckel
ok, 140725

Master Thesis Exposé

Comparing Prompt Engineering, Fine-Tuning, and Retrieval-Augmented Generation for Task-Specific LLM Applications

Anirudh Deepak Bhargav, 11038508
M.Sc. Applied Computer Science
Supervisor: Prof. Dr. Moeckel, Gerd

---

1. Abstract

The proliferation of open-source Large Language Models (LLMs) like Mistral, Gemma, and LLaMA 2 has made it feasible for developers and researchers to build intelligent systems tailored to specific domains. However, choosing the most effective method to adapt an LLM for a specific task remains a major challenge.

Three widely used strategies—Prompt Engineering, Fine-Tuning, and Retrieval-Augmented Generation (RAG)—each offer unique advantages and trade-offs in terms of performance, cost, and complexity. This thesis aims to investigate and compare these approaches in a controlled, small-scale experimental setting. The comparison will be conducted through the implementation of a Python Programming Q&A Assistant focused on beginner-level coding errors.

2. Problem Statement

While all three adaptation methods are widely discussed in literature, there is a lack of hands-on comparative analysis based on practical metrics such as resource consumption, implementation feasibility, and output quality. This thesis seeks to fill that gap by applying each method to a consistent, domain-specific dataset and evaluating them both quantitatively and qualitatively.

3. Objectives

This thesis aims to accomplish the following objectives:

- Implement three distinct LLM adaptation strategies—Prompt Engineering, Fine-Tuning (LoRA), and RAG—on a domain-specific task using a consistent dataset.
- Evaluate and compare each approach across multiple dimensions:
    - Output quality using automated metrics (BLEU, ROUGE-L) and human evaluation
    - Computational and financial costs (inference vs training vs retrieval)

    - Technical complexity of setup and reproducibility
- Build lightweight, reproducible prototypes that demonstrate how each approach performs in real-world conditions
- Document trade-offs and best practices to assist researchers and developers in selecting the most suitable adaptation technique for different types of applications and constraints.

4. Methodology

The comparative evaluation will follow a structured experimental pipeline:

Step 1: Dataset Curation

A high-quality, medium-scale dataset (100–500 entries) will be created based on a selected task such as technical Q&A or news summarization. This dataset will be pre-processed and used as the uniform basis for all experiments.

Step 2: Prompt Engineering

Using a pre-trained LLM (e.g., Mistral 7B), carefully crafted prompts will be used to elicit task-specific responses. Different versions of prompt structures (e.g., few-shot vs zero-shot) will be tested and scored.

Step 3: Fine-Tuning (LoRA)

The model will be fine-tuned using Low-Rank Adaptation (LoRA) with HuggingFace's PEFT and Transformers libraries. This will be done in a Google Colab Pro environment with GPU access. The fine-tuning will use the same dataset and target output.

Step 4: Retrieval-Augmented Generation (RAG)

A RAG pipeline will be implemented using SentenceTransformers to encode the dataset, FAISS for vector indexing, and LangChain for retrieval and generation orchestration. This method adds context dynamically from the dataset during inference.

Step 5: Evaluation and Comparison

Each method will be evaluated using:

- Cost analysis: Time, hardware/GPU needs, storage
- Human evaluation: Accuracy, relevance, factuality, consistency
- Automated metrics: BLEU, ROUGE-L
- Complexity & Feasibility: Lines of code, reproducibility, documentation requirements

5. Expected Results and Deliverables

- A clean, task-specific dataset
- Three separate working LLM adaptation prototypes
- A detailed evaluation report comparing all approaches
- Final thesis including charts, diagrams, code snippets, and insights for future research

6. Use Case – Python Programming Q & A Assistant

The practical implementation in this thesis will be a lightweight Python Programming Q&A Assistant — a chatbot designed to answer common beginner-level coding questions in natural language. The assistant will specifically target typical problems faced by novice Python learners, such as:

- Syntax errors
- Indentation mistakes
- Type mismatches
- Basic logical or structural misunderstandings

By narrowing the focus to this clearly defined and realistic problem space, the models can be trained and tested effectively using a relatively small dataset (approximately 300–500 curated Q&A pairs). For example, when asked a question like *"Why does int("abc") throw an error?"*, the assistant will return a helpful, code-backed explanation with corrective suggestions.

This focused scope ensures measurable outcomes while allowing a fair and meaningful comparison of Prompt Engineering, Fine-Tuning, and Retrieval-Augmented Generation techniques.

7. Timeline (Tentative)

Phase	Duration (Approx.)	Description
Literature Review & Planning	Weeks 1–4 (Month 1)	Study LLM adaptation techniques, finalize task & dataset, outline structure
Dataset Collection & Preprocessing	Weeks 5–6 (Month 2)	Curate and clean dataset for use across all methods
Prompt Engineering Implementation	Weeks 7–9 (Month 2–3)	Design and test multiple prompt strategies using pre-trained models
Fine-Tuning with LoRA	Weeks 10–13 (Month 3–4)	Apply LoRA fine-tuning, run experiments, optimize hyperparameters

Phase	Duration (Approx.)	Description
RAG Pipeline Implementation	Weeks 14–17 (Month 4)	Setup vector DB, retrieval logic, and test RAG performance
Evaluation & Analysis	Weeks 18–20 (Month 5)	Apply metrics, perform human evaluation, document trade-offs
Thesis Writing (Draft & Review)	Weeks 21–23 (Month 5–6)	Write main thesis document, create visualizations, get feedback
Final Edits & Submission Prep	Week 24 (End of Month 6)	Final proofread, formatting, appendix, and submission checklist

8. Areas for Further Research

- How evaluation metrics (BLEU, ROUGE) correlate with human judgment
- Feasibility of fine-tuning with limited hardware (e.g., Google Colab Pro limits)
- Best practices in RAG pipeline construction
- Trade-offs between shallow vs deep prompt chains
- Dataset quality and preprocessing impacts on fine-tuning and RAG
- Use of lightweight models (e.g., distilled versions) vs full-scale models

9. Probable Bottlenecks & Resource Constraints

- Hardware Limitations: Fine-tuning even with LoRA may hit VRAM/memory limits on Colab Pro. Backup access to local or cloud GPU might be needed.
- Time Constraints: Implementing all three methods, especially RAG, requires tight project scheduling.
- Model Incompatibility: Some models may not support LoRA out-of-the-box or may be too large for hosted environments.
- Data Preprocessing: Creating a consistent and clean dataset that works for all three methods may require iterative refinement.
- Evaluation Challenges: Human scoring can be subjective and time-consuming, requiring multiple iterations and possibly external reviewers.
- Framework Complexity: RAG frameworks like LangChain may have a steep learning curve and integration quirks.

10. References (to be expanded)

- HuggingFace documentation (Transformers, PEFT)
- LangChain and FAISS official guides
- Scholarly articles on LoRA, RAG, and prompt design principles
- Research on evaluation metrics and human alignment