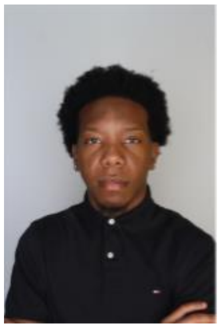
P10-T1 LLM Compression Final Report

Course: CS4850-03 | Semester: Fall 2025 | Professor Perry | Date 12/3/25

Ryan Tran Developer/Documentation	Aldi Susanto Developer/Documentation/Team Lead	 Thomas Graddy Developer/Documentation	Pranav Kartha Developer/Documentation

Team Members:

Name	Role	Cell Phone / Alt Email
Ryan Tran	Documentation/Development	678-670-9868 Concepting@protonmail.com
Aldi Susanto	Team Lead/Documentation/Development	404-834-9304 aldisusanto01@gmail.com
Thomas Graddy	Documentation/Development	229-661-5041 Thomas.graddy3@gmail.com
Pranav Kartha	Documentation/Development	943-268-4909 pranav.kartha1950e@gmail.com
Sharon Perry	Project Owner or Advisor	770-329-3895 Sperry46@kennesaw.edu
Taylor Cuffie	Academic Program Support Specialist	470-578-5760 tcuffie1@kennesaw.edu

GitHub Repo Link:

https://github.com/RyanTren/AFB-LLM-Compression-Trade-off-Evaluator

Stats and Status

- This project is under NDA
- The number of lines of code in our project varies per branch (we have 3) (put estimation of all combined lines of code here)
- Number of Project Components/tools in project
- Total Man Hours:
    - Estimated:
    - Actual:
- Status: % complete for project (we can probably say like 85%)

Table of Contents

Introduction

This project investigates how model compression techniques affect the balance between speed and accuracy for large language models (LLMs) on code-generation and debugging tasks. We build a baseline inference environment and implement several compression methods, including quantization, pruning, and knowledge distillation. Using a suite of code-focused prompts, we evaluate each model's quality with BLEU and CodeBLEU, along with latency, throughput, memory usage, and cost.

Based on these results, we developed a prototype system that dynamically routes queries between the full model and compressed variants. The router estimates query complexity and selects the smallest model that can maintain a target accuracy, which improves efficiency while limiting quality loss on difficult inputs. We compare this dynamic approach with static single-model deployments and analyze Pareto frontiers that relate model quality to performance.

The project concludes with a reporting dashboard, evaluation harnesses, analysis scripts, and system documentation that support reproducibility and cloud deployment. Sponsor guidance informs benchmark selection, compression settings, and routing criteria, with the overall goal of enabling practical and scalable LLM inference for code-related workloads.

Requirements

Project Overview

The project aims to build a **Large Language Model (LLM) Compression Accelerator** for **Robins AFB**, focusing on optimizing LLM usage offline through compression techniques. It evaluates trade-offs between **model size, cost, and accuracy** for code-related tasks (debugging and cross-language translation). The solution includes a **dynamic query routing prototype** that selects between full and compressed models based on query complexity.

Key Objectives

- Implement and compare **compression methods**: quantization, pruning, and knowledge distillation.

- Measure performance using **BLEU/CodeBLEU**, HumanEval, latency, throughput, GPU hours, and memory usage.
- Develop a **dashboard** for visualizing trade-offs and exporting results.
- Ensure reproducibility via **Dockerized environments** and CI/CD workflows.

Scope

In-Scope:

- Containerized setup for LLaMA 3.3 (8B) and DeepSeek-Coder V2 Lite (16B).
- Compression experiments and evaluation metrics.
- Dashboard with filtering, comparison, and export features.
- Prototype dynamic query router.

Out-of-Scope:

- Production deployment in Air Force systems.
- Proprietary datasets or advanced compression beyond the three chosen methods.

Success Criteria

- Compressed models retain **≥95% baseline CodeBLEU accuracy** while reducing latency/GPU usage by **≥2×**.
- Dashboard supports comparison of at least **three configurations**.
- Prototype routing system demonstrates dynamic allocation.

Design & Architecture

- **Environment:** GPU-enabled VM (RTX 4090 or A100), Docker containers for LLM and dashboard.
- **System:** Modular architecture with Python (PyTorch/Transformers), compression tools, and full-stack dashboard (React/Angular or Streamlit/Dash).
- **Interfaces:** Dashboard for metrics visualization; export to PDF/CSV/JSON.

Functional Requirements

- Baseline LLM setup.
- Implement compression techniques.
- Dynamic query routing.
- Evaluation and reporting dashboard.
- Export functionality and milestone reporting.

Non-Functional Requirements

- **Security:** Open-source datasets only; containerized isolation.
- **Capacity:** Handle models up to 16B parameters; demonstrate 2× efficiency gains.
- **Usability:** Intuitive dashboard for non-ML experts.
- **Reproducibility:** Dockerized deployment and CI/CD pipelines.

Risks & Mitigations

- **Accuracy degradation:** Conservative compression thresholds.
- **Hardware constraints:** Use smaller models or optimization techniques.
- **Time constraints:** Scope routing system as prototype.
- **Generalizability:** Document assumptions clearly.
- **Operational integration:** Emphasize modular design for future adoption.

Analysis / design

Purpose

This SDD translates the requirements from the SRS into a concrete design for implementing the **LLM Compression Trade-Off Accelerator**. It defines architecture, components, interfaces, workflows, and quality attributes to ensure reproducibility, scalability, and maintainability.

System Overview

The system accelerates LLM inference for **code-related tasks** by:

- Establishing a baseline inference environment.
- Applying **compression techniques**: quantization, pruning, and knowledge distillation.
- Dynamically routing queries between baseline and compressed models.
- Presenting results in an interactive dashboard. Deployment uses **Docker** for containerization and supports GPU-enabled VMs or cloud environments.

Design Considerations

- **Assumptions:** Focus on code generation accuracy (evaluated via CodeBLEU), using DeepSeek V2 and LLaMA models; requires GPU-enabled VM; reproducibility via Docker and Conda.
- **Constraints:** Heavy resource requirements; goal is to reduce hardware needs while maintaining accuracy; SSH/FTP for VM communication; GitHub for version control.
- **Compression Methods:**
    - **Quantization:** Reduces precision for smaller size and faster inference.
    - **Pruning:** Removes redundant neurons for efficiency.
    - **Knowledge Distillation:** Transfers knowledge from large to smaller models.
- Each method has trade-offs in complexity, accuracy, and performance.

Architectural Strategies

- **Containerized Microservices:** Separate services for Model, Compression, Routing, Evaluation, Dashboard, and Results Store.
- **Stateless Compute & Versioned Artifacts:** All compute is stateless; artifacts and datasets are version-controlled for reproducibility.
- **Experiment Orchestration:** Controller manages runs, retries, and artifact persistence.
- **Routing Strategy:** Rule-based (prompt length, complexity) and learned classifier for dynamic model selection.
- **Observability:** Logging, metrics (latency, throughput, GPU hours), tracing, dashboards (Grafana), and alerts.
- **Failure Safety:** Fallback to baseline model, checkpointing, retries, and container isolation.

System Architecture

Four main subsystems:

1. **Model Compression Engine:** Applies quantization, pruning, and distillation.
2. **Data Processing & Loader:** Handles tokenization, batching, and dataset preparation.
3. **Trade-Off Analyzer:** Evaluates accuracy, latency, memory, and cost; generates reports.
4. **User Interface & Control Module:** CLI and optional web dashboard for configuration, monitoring, and exporting results.

Detailed Design

- **Interfaces:** Functions and endpoints for compression (`compress_model`), evaluation (`evaluate_tradeoff`), dataset loading, and reporting.
- **Resources:** GPU compute (CUDA/cuDNN), PyTorch, Hugging Face, Docker, GitHub for version control.
- **Constraints:** Must maintain functional correctness, support standard formats (.pt, .bin), and avoid deadlocks in parallel jobs.
- **Exports:** Reports in PDF, CSV, JSON; metrics and manifests for reproducibility.

Glossary & References

Includes definitions for LLM, SRS, SDD, compression techniques, BLEU/CodeBLEU, Docker, PyTorch, CI/CD, and routing concepts.

Development

Test (plan and report)

Version control

For version control, we chose GitHub as our platform and in our repository structure we have 4 branches. (Main, lora, quantization, and knowledge distillation)

- The main branch contains the context and documentation of our project
- In the other 3 branches has in-depth and documented setups and logs of our development work between the three different compression methods we choose.

Conclusion

Appendix – training verification (if mobile, web or desktop apps), project plan (optional), etc.

Item – See assignment for details – list may vary by type of project	**Points**
Final Report	**70**
Cover page (with all required information)	✓
Table of Contents	✓
Sections	
Run Spell Check	
Final Report filename	✓
Website – must be public and all links must work	**25**
Home page layout – all info and bio pics	
Link to Final Report - that works and displays report	
Link to GitHub	
Link to Final Presentation Video	
Source code (zip file) submitted to D2L	**5**
Source code filename	
Total	**100**