P10-T1 LLM Compression Final Report

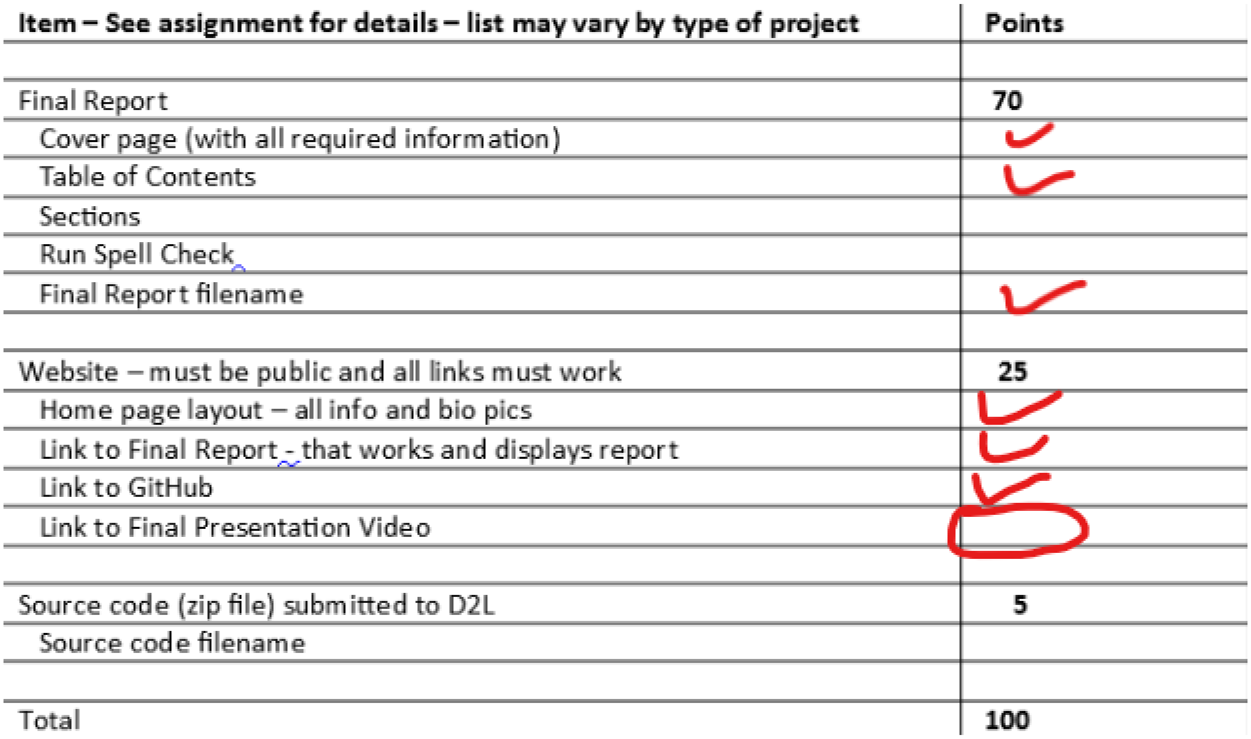
Course: CS4850-03 | Semester: Fall 2025 | Professor Perry | Date 12/3/25

 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

Checklist:



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**

GitHub Repo Link:

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

Final Report Website Link:

https://afb-llm-compression-trade-off-evalu-lac.vercel.app/

Stats and Status	Project is under NDA
# of Lines of Code	15k+ Lines of code across our main, lora, knowledge_distilation, and quantization branches.
Number of Project Components/tools in project	10+ React + TypeScript + Vite, StreamLit, Python, Docker, HuggingFace, TensorFlow/PyTorch, pandas, numPy, Git, PuTTY + KSU VM
Estimate Hours	300
Actual	342
Status	95% complete for project (need to record final demo)

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, LoRA, and knowledge distillation. Using a suite of code-focused prompts, we evaluate each model's quality with HumanEval 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 (StreamLit + Python for LoRA and Quantization branch in our repo), training/inference/evaluation pipeline, and system documentation that support reproducibility and containerized 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.

Challenges/Assumptions

(filler for now)

- Talk about VM downtimes, our original VM crashing
- Depreciated dependencies and other conflicts
- Limited by our VM Specs on what models we could train and how it impacted our training time when we did use smaller models.
- No Sudo Access

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) for Quantization and DeepSeek-Coder V2 1B/CodeParrot Small Model for LoRA.
- 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 (4x Tesla A100s), Docker containers for LLM Training/Compression/Evaluation Scripts and StreamLit 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.

Development

Model Research and Selection

Three models were initially chosen for their coding capabilities:

- **DeepSeek-Coder-V2**: Excellent reasoning/debugging, open weights, multiple sizes (16B–236B).
- **Qwen2.5-Coder**: Strong multilingual debugging and translation, scalable sizes (7B–32B).
- **StarCoder2**: Cost-effective, permissively licensed, smaller sizes (3B–15B).

Due to hardware limitations (Tesla M40 GPUs, 11 GB VRAM), the team pivoted to smaller models like **GPT-2**, **CodeParrot-small (~100M)**, and **Llama 3.1-8B** for realistic compression experiments.

Hardware Constraints

The VM provided had:

- **4× NVIDIA Tesla M40 GPUs (11 GB VRAM each)**
- **Compute capability 5.2**
- **503 GB system memory**
- **220 GB disk storage**

These specs were insufficient for large models requiring Ampere/Hopper GPUs and advanced low-precision kernels. This led to adopting smaller models and parameter-efficient fine-tuning.

LoRA Training Environment

Implemented using:

- **Transformers** (model loading/tokenization)
- **PEFT** (LoRA layers)
- **Accelerate** (multi-GPU orchestration)
- **DeepSpeed** (ZeRO-Offload for memory optimization)
- **Torch/CUDA 13.0**

This setup allowed fine-tuning on legacy GPUs without loading full model weights.

Evaluation Metrics

- **Execution-Based**: Compile success rate, unit test pass@k, latency, GPU memory usage.
- **Quality-Based**: CodeBLEU, BLEU, exact match, APR metrics (plausible/correct/regression patch rates).

Execution-based metrics were prioritized for functional accuracy.

Datasets

- **Code Translation**: CodeXGLUE (Java ↔ C#)
- **Debugging/APR**: Defects4J, QuixBugs, MdEval
- **LoRA Training**: codeparrot-clean, python_code_instructions_18k_alpaca, codecomplex

Implementation Summary

- **Architecture**: Modular, containerized (Docker + CUDA 13.0 + PyTorch 2.1).
- **Compression**: Custom 4-bit GPTQ quantizer script for quantization.
- **LoRA/PEFT Pipeline**: Fine-tuning with low-rank adapters, FP16 precision, DeepSpeed ZeRO-3 offloading.
- **Outputs**: Adapter weights, metrics, generation samples stored under `/src/lora_out_<model_name>/`.

Risk Mitigation

- Accuracy degradation: Controlled via conservative thresholds.
- Hardware limits: Addressed with 4-bit quantization.
- Time constraints: Prototype-first approach.
- VM restrictions: Workarounds for read-only access.
- Deprecated functions: Adjustments for Python 3.11.9 and legacy GPUs.

Setup Instructions

- Connect to VM via **GlobalProtect VPN** and **PuTTY**.
- Clone repo with SSH keys.
- Build local environment using Python venv and install dependencies.
- Configure Hugging Face cache in `/tmp` due to limited permissions.
- Training and inference scripts: `train_lora.py`, `run_inference.py`.

Test (STR and STP)

Scope

Testing covers:

- Baseline vs. compressed model performance for code generation.
- Validation of **quantization**, **LoRA**, and **knowledge distillation** workflows.
- Dynamic query-routing behavior.
- Evaluation metrics: BLEU/CodeBLEU, latency, resource usage.
- Dashboard functionality for visualization and data integrity.
- End-to-end integration of scripts, routing logic, and metrics pipeline.

Out of Scope: UAT by sponsor, usability beyond basic dashboard checks, external system integration, training full-scale LLMs from scratch.

Testing Summary

- **In Scope:** Baseline/compressed LLM performance, compression pipelines, routing logic, metric evaluation, dashboard validation.
- **Out of Scope:** External integrations, new LLM architectures, non-code tasks.

Test Focus Areas

- **Release Content:** Working version of LLM Compression Framework integrating GPTQ and LoRA workflows for training, inference, and evaluation.

- **Regression Testing:** Ensures stability after changes in compression workflows, evaluation scripts, or routing logic.
- **Platform Testing:** Conducted on Ubuntu VM with NVIDIA Tesla P100 GPU, validating Python, PyTorch, Transformers, GPTQ, PEFT, and evaluation utilities.

Test Objectives

- **Progression:** Validate CodeBLEU similarity for compressed models (GPTQ, LoRA, self-distilled Llama).
- **Regression:** Confirm BLEU metric accuracy and system stability under compression.

Other Testing

- **Security:** Validate permissions, prevent external calls, and ensure no credential exposure.
- **Stress & Volume:** Assess system under heavy inference loads and large prompt batches.
- **Connectivity:** Confirm internal component communication and dataset/model access.
- **Disaster Recovery:** Verify restoration of LoRA checkpoints, GPTQ artifacts, datasets, and configs.
- **Unit Testing:** Validate individual modules (data utilities, LoRA helpers, quantization utilities, metrics).
- **Integration Testing:** Ensure workflows (LoRA, GPTQ) and evaluation tools work together seamlessly.

Test Strategy

- **Responsibilities:** Project team handles unit/integration; external party assists with security; business handles UAT.
- **Approach:** Separate testing for each compression method; Docker-based deployment.
- **Environment:** Ubuntu VM with GPU support; isolated VPC; controlled access and monitoring.

Entry/Exit Criteria

- **Entry:** Environment ready, access granted, stable code baseline, artifacts prepared.

- **Exit:** All tests executed, defects logged, results validated, environment stable.

Testing Metrics

Includes:

- Defect density, severity mix, reopen rate, escape rate.
- Test case pass rate, requirements coverage, automation coverage.
- Latency (p50/p90/p99), throughput, error rate.
- Cost per query, model quality (CodeBLEU, pass@k), routing effectiveness, footprint reduction.

Software Test Results

Model	Compression	CodeBLEU	Status	Severity
Base Llama 3 8B	16 GB	0.67	Pass	Minor
GPTQ Llama	5.5 GB	0.33	Pass	Minor
CodeParrot LoRA	2 GB → 10–20 MB	0.31	Pass	Minor
DeepSeek LoRA	256 MB → 1 MB	0.59	Pass	Minor
Self-Distilled Llama3.2 1B	KL Loss: 6.688	N/A	WIP	Moderate

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 as well as the final project website built with React + TypeScript + Vite.
- The other 3 branches have in-depth documented setups and logs of our development work between the three different compression methods we choose: LoRA, Quantization, & Knowledge Distillation.

Conclusion

As a group, we all learned the importance of having a set and stone SDLC, communication, skill building with Streamlit, bash scripting, React/TypeScript, Python, Docker, etc. Being able to delegate and communicate project tasks and deadlines we're important to deliver quality work to present to Robins AFB Milestone Meetings on Teams.

We will all carry the skills acquired from this class and experience to our future endeavors whether that is a masters, PHD, internship, co-op, or full-time role. Thank you, Professor Perry, for your guidance and patience with not only our group but the other sections you had responsibility for.

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