■ Final Project: End-to-End MLOps System

Goal

Build a fully containerized, production-grade machine learning system — from model training to deployment and monitoring — using modern MLOps tools. The outcome will be a complete GitHub repository demonstrating your ability to integrate all components into a cohesive and automated pipeline.

1. Project Overview

You will design and implement an end-to-end machine learning system that includes MLflow, Airflow, Docker & Docker Compose, FastAPI, Grafana, and Pytest. Your project should reflect real-world MLOps practices and be reproducible from scratch by running a few commands.

A. Experiment Tracking with MLflow

Integrate MLflow into your training pipeline. Log parameters, metrics, and artifacts. Register the best-performing model in the MLflow Model Registry and manage version promotion to Production.

B. Model Serving with FastAPI

Create a REST API service that loads the latest Production model from MLflow, exposes /predict and /health endpoints, validates input with Pydantic, and returns JSON results. Include Pytest-based endpoint tests.

C. Pipeline Orchestration with Apache Airflow

Design a DAG for data ingestion, model training, evaluation, and promotion. Each step should be a separate task, scheduled automatically.

D. Containerization with Docker & Docker Compose

Containerize MLflow, FastAPI, Airflow, and Grafana. Orchestrate all services using Docker Compose to launch the full system with one command.

E. Monitoring with Grafana

Integrate Grafana to visualize system metrics (CPU, memory, API latency, request count). Expose metrics from FastAPI via Prometheus-compatible endpoints.

F. Testing with Pytest

Implement automated tests for the API, pipeline, and helper functions. Integrate Pytest into your CI/CD workflow so tests run on each code push.

3. Evaluation Criteria (100 points total)

Category	Description	Points
Functionality	All services (MLflow, Airflow, FastAPI, Grafana) work together seamlessly	20
Code Quality	Modular, clean, and well-documented Python code	10

MLflow Integration	Proper tracking of runs, metrics, and model versioning	10
Airflow Pipeline	Functional DAG with correct sequencing and automation	15
Model Serving API	Reliable, validated FastAPI service with working endpoints	10
Containerization	Correct use of Docker and Docker Compose	10
Monitoring	Functional Grafana dashboard with relevant metrics	10
Testing (Pytest)	Comprehensive test suite integrated into pipeline	10
Documentation	Clear README with setup, usage, and architecture overview	5

4. Bonus Challenges (Optional, +10 points)

- Add CI/CD automation with GitHub Actions to rebuild and test containers automatically.
- Use DVC for dataset versioning.
- Deploy your system to Google Cloud Run, AWS ECS, or Azure Container Apps.
- Implement automated retraining triggers based on model drift metrics.

5. Submission

Submit a public GitHub repository link containing source code, Docker configurations, and a README with setup instructions. Include screenshots or a short demo video showing MLflow UI, Airflow DAG, FastAPI /docs page, Grafana dashboard, and Pytest results.

6. Expected Outcome

By completing this project, you will demonstrate practical MLOps skills — designing a reproducible, automated ML pipeline with full lifecycle monitoring, testing, and orchestration.