Capstone Project: End-to-End MLOps Pipeline

Goal

Students will design, implement, and deploy a complete MLOps pipeline for a machine learning project of their choice. The focus is not on the specific ML model, but on building a reproducible, automated, and deployed system that demonstrates version control, experiment tracking, orchestration, model registry usage, deployment, and serving.

Students are free to use **any tools** beyond the ones covered in class. For example:

- If you prefer Hugging Face for model serving, you may use that.
- If you prefer GitHub for dataset versioning instead of W&B, you may use that.
- The important part is that you can **justify your design choices** in your final report.

There is no “right” or “wrong” pipeline design — what matters is reproducibility, automation, and clarity of reasoning.

Project Requirements

1. Model Training

- Select a dataset and ML problem of your choice (classification, regression, image analysis, text analysis, etc.).
- Choose an architecture suitable for the problem. Note: try to select a model that can be trained on CPU with limited compute resources.
- Ensure reproducibility (requirements.txt, Docker, or equivalent).

2. Version Control

- **Code**: Use Git and GitHub for managing source code, scripts, and notebooks.
- **Data and Models**: Use W&B Artifacts and Model Registry (or an alternative you justify) for dataset and model versioning.

3. Experiment Tracking

- Track experiments in W&B (or alternative):
    - Metrics (accuracy, loss, RMSE, etc.).

    - Hyperparameters and configurations.
    - Artifacts: datasets, checkpoints, trained models.

4. Orchestration (CI/CD)

- Use GitHub Actions (or an alternative orchestrator) to:
    - Automate training pipelines on push/merge.
    - Automate model registration into the model registry.

5. Inference Service + Frontend

- Build an inference service (FastAPI, Hugging Face Inference, or equivalent) that:
    - Loads the latest version of the model from the registry.
    - Exposes REST endpoints for inference (`/predict`) and health checks (`/health`).
- Connect the service to a simple frontend (Streamlit, Gradio, or equivalent) where users can input data and view predictions.
- The frontend should interact with the inference API (not run the model directly).

6. Deployment

- Containerize backend and frontend.
- Deploy using Google Cloud Run or another production environment.
- Share a public URL that demonstrates a fully functional app.
- Deployment must be CI/CD-driven: when a new model variant is registered, the deployed application should automatically begin serving the latest version.

Deliverables

1. **GitHub Repository** containing:
    - Code, datasets (or dataset links), Dockerfiles, orchestrator workflows.
    - Documentation (README with setup and usage instructions).
2. **Deployed Application**:
    - Public link to the working app (frontend + backend).
3. **Experiment Tracking**:
    - W&B (or alternative) project with logged experiments and model versions.
4. **Pipeline Design Diagram**:
    - A clear diagram showing how the pipeline is structured (from data ingestion → training → versioning → CI/CD → deployment → inference).
5. **Final Report**:
    - Justify your pipeline design decisions:
        - Why did you choose a particular tool for each component?

        - Why is the pipeline configured in its current form?
    - Reflect on trade-offs: reproducibility, scalability, ease of use, and resource constraints.