Homework 2 - Implementing the MLOps CI/CD Pipeline
Deadline : 11.01.2026 Sunday 23:59

Objective: To transition from a manual ML workflow (MLOps Level 0) to an automated pipeline (MLOps Level 1 & 2). You will implement a "Commit Stage" that enforces code quality and a "Deployment Stage" that verifies system health via smoke testing.

Context: This homework focuses on the infrastructure required for your Term Project. You are building the **Continuous Integration** backbone that ensures your High-Cardinality Prediction Service remains deployable at all times.

Part 1: The Commit Stage (Continuous Integration) (4 points)

According to the principles of CI, developers must integrate their work frequently, and every integration must be verified by an automated build.

Tasks:

1. **Version Control Setup:** Create a repository (e.g., GitHub/GitLab). Ensure all assets (code, Dockerfiles, database schema scripts, test data) are in the repository. "Everything you need to build the software must be contained in the version control repository".
2. **Automated Unit Testing:** Implement **Unit Tests** for your feature engineering logic.
    - *Requirement:* These must be fast, isolated tests with no external dependencies (no database/network calls).
    - *MLOps Context:* Test your hashing or embedding logic. For example, ensure your Hashed Feature function returns the correct bucket index for a known input string.
3. **Code Analysis/Linting:** Integrate a static analysis tool (e.g., Pylint, Flake8) to check for code style and syntax errors. Failure to meet these thresholds must fail the build.

Part 2: The Automated Acceptance Gate (CD) (3 points)

You must prove that your application works from a user's perspective before it is considered "done."

Tasks:

1. **Component/Integration Testing:** Implement at least one **Component Test** that verifies the interaction between your model serving logic and a data source (or a mock of it).
    - *Requirement:* Unlike unit tests, this can involve a database or a file system to ensure data consistency.

2. **Build & Package:** Create a build script (e.g., a Docker build command) that packages your model and serving code into a deployable artifact (binary/container). "Only build your binaries once".
3. **Smoke Test:** Write a script that spins up your container and sends a single prediction request (e.g., using curl or a Python script) to verify the service is up and responding (returning a 200 OK). This is the critical "Deployment Test".

Part 3: The "Stop the Line" Simulation (3 points)

A core practice of CI/CD is that **if any part of the pipeline fails, you stop the line**. You must demonstrate that your pipeline actively prevents bad code from entering production.

Tasks:

1. **The Sabotage:** Intentionally introduce a bug into your feature engineering code *OR* a syntax error.
2. **The Block:** Commit this broken code and show that your CI pipeline detects the failure and stops the deployment process.

Deliverables

Please submit a PDF report containing the following evidence:

1. **Pipeline Configuration:** A screenshot or snippet of your pipeline configuration file (e.g., .github/workflows/main.yml, Jenkinsfile, or .gitlab-ci.yml) showing the stages: Build $\rightarrow$ Unit Test $\rightarrow$ Lint $\rightarrow$ Package $\rightarrow$ Smoke Test.
2. **Test Results:**
    - **Evidence A (Success):** A screenshot of a "Green" build where all unit tests, component tests, and the final smoke test passed.
    - **Evidence B (Failure/Stop the Line):** A screenshot showing the pipeline **failing** and blocking deployment after you introduced the intentional bug.
3. **Test Code:** Paste the code for your **Unit Test** (feature engineering logic) and your **Smoke Test** (deployment verification). Explain *why* the unit test is considered "fast" and the smoke test is considered "end-to-end" based on the course definitions,.

Tools & References

- **CI/CD Platform:** You may use GitHub Actions, GitLab CI, or Jenkins.
- **Testing Frameworks:** Pytest or Unittest for Python code.
- **Orchestration (Optional but recommended):** If your logic is complex, you may incorporate Airflow or Prefect for the workflow, but the CI trigger must come from your Git provider,.