SWE016 Machine Learning Operations Project
Deadline : 02.01.2026 Friday 12:00

1. MLOps Term Project Announcement: Building and Governing Production ML Pipelines

This project requires designing and implementing an end-to-end Machine Learning system demonstrating **MLOps Level 2 maturity: CI/CD Pipeline Automation**. Your final presentation must be a **business presentation** directed toward **senior management** , focusing on operational efficiency and risk mitigation.

Term Project Theme: Developing a Resilient, High-Cardinality Prediction Service

Your group must design, implement, deploy, and monitor an end-to-end ML system that tackles a supervised learning problem (classification or regression) involving complex or **high-cardinality categorical features**.

I. Mandatory Deliverables and Rules

Compliance with the following organizational and deliverable rules is mandatory:

Requirement	Details and MLOps Context
Group Size	Student groups must consist of **5 to 7 members**.
Defined Roles	Each student's **responsibilities must be clearly defined** . Map responsibilities to roles (e.g., Data Engineer, ML Engineer, DevOps).
Working Demo	The project **must include a working demo** . The demo should showcase the automated pipeline execution, the live prediction service, and the monitoring dashboard.
Individual Report	**Each student must upload their individual report** , detailing their contribution and technical implementation choices.
Business Presentation (PPT)	Prepare a PowerPoint presentation which should be a **business presentation**, assuming you will present this to **senior management** .
Presentation Flow	The presentation must strictly adhere to the **Introduction-Development-Conclusion flow** .
Video Presentation	Each group must prepare a **5-minute video presentation** at the end of the semester and share the link separately .
In-Class Presentation	The in-class presentation will be conducted **in person** , and the video presentation will also be required .

II. Required Tool Stack and MLOps Infrastructure

Students are required to implement a modern, hybrid MLOps tool stack. The core tooling components are:

1. Experiment Tracking and Model Governance

You may use **MLflow**.

- MLflow is the **dedicated experiment tracking tool** needed regardless of the chosen orchestrator.
- It must be used to log every parameter, metric, and model version.
- MLflow's **Model Registry** must be demonstrated as the central hub for version control and stage management (e.g., moving models from staging to production).

2. Workflow Orchestration and CI/CD Pipeline Engine (Choose One)

Your group must select and justify one of the following tools to manage the overall pipeline workflow (DAG):

- **Kubeflow Pipelines (KFP):** Ideal if you focus on Kubernetes-native scale and deep learning.
- **Apache Airflow:** Ideal if focusing on reliable time-based automation for generalized data jobs and complex ETL (Data Preparation) steps.
- **Prefect:** Preferred for dynamic workflows that respond intelligently to data changes and fail fast during complex retraining loops.

3. Core CI/CD and Auxiliary Tools

The demonstration of CI/CD automation and resilient serving will require the use of standard DevOps and testing tools:

Tool Category	Example Tools Mentioned in Sources	Application in Project
Containerization	**Docker, Kubernetes**	Used to package the ML model, dependencies, and serve the Stateless Serving Function.
CI/CD Execution	**Jenkins, GitLab CI/CD, Circle CI**	Defining the pipeline steps (Commit Stage, Acceptance Test Stage).
Unit/ Component Testing	**xUnit Frameworks (JUnit, NUnit), DbUnit**	Used for unit testing feature engineering logic and testing database interactions for repeatability.
Code Inspection/ Analysis	**Checkstyle, PMD, JDepend, NDepend**	Ensuring code quality and architectural adherence in the CI pipeline.
Cloud	**AWS, Azure, Google Cloud**	All services are allowed

III. Mandatory Technical Implementation Requirements

Your working demo must integrate and justify the use of specific Machine Learning Design Patterns, linking them to the chosen tool stack:

1. Data Representation (Handling High-Cardinality)

You must implement methods to transform complex inputs into features the model can operate on:

- **High Cardinality:** Implement and justify using either **EMBEDDINGS** (mapping high-cardinality data into a dense, lower-dimensional space, learning weights via gradient descent) or the **HASHED FEATURE** pattern (converting categorical input into a fixed number of buckets, accepting the trade-off of **bucket collision** and potential loss of model accuracy).
- **Feature Interactions:** Implement a **FEATURE CROSS** (concatenating two or more categorical features, potentially after **bucketing** numerical data) to allow the model to learn relationships between inputs faster.

2. Problem Representation and Training

Your solution must incorporate strategies for resilient modeling:

- **Problem Reframing:** Explore the use of **REFRAMING** (e.g., turning a regression into a classification problem by **bucketizing outputs**) to gain expressiveness of a full probability distribution or restrict the prediction range.
- **Performance Optimization:** Use the **ENSEMBLES** design pattern (such as Bagging or Boosting) to combine multiple models to decrease bias and/or variance, documenting the **Increased Training and Design Time** trade-offs.
- **Data Imbalance:** Apply the **REBALANCING** design pattern (e.g., Downsampling or Upsampling) to address highly skewed data, noting the importance of choosing an appropriate evaluation metric beyond simple accuracy.
- **Reproducibility:** Utilize **CHECKPOINTS** during training to ensure resilience (resuming training after interruption) and facilitate **fine-tuning** on fresh data by retraining from the last checkpoint.

3. Resilient Serving and Continuous Evaluation

The deployment must focus on reliability and monitoring:

- **Serving Architecture:** Implement the **STATELESS SERVING FUNCTION** pattern via a REST endpoint to handle prediction requests, noting its benefits for **autoscaling** and **language-neutral** operation. Discuss when the alternative, **BATCH SERVING**, would be more appropriate for asynchronous, non-latency-sensitive workloads.
- **Continuous Evaluation (CME):** Implement the **CONTINUED MODEL EVALUATION** design pattern to track model degradation caused by **data distribution shifts** (like concept drift or covariate shift).
    - Monitoring must track **ML-specific metrics** related to **features** and **predictions**.
    - The system must use evaluation metrics (like accuracy, precision, or AUC) to detect performance degradation.
    - Monitoring should incorporate statistical checks (like those available in tools such as **Great Expectations**) to perform feature validation and detect skew.
- **Algorithmic Fallback:** Present a strategy for **algorithmic fallbacks** (e.g., rolling back to an older model version or using a simpler, hard-coded solution) in case monitoring detects performance drops below acceptable thresholds.