Project – End-to-End MLOps Pipeline for Taxi Trip Duration Prediction

This project aims to provide students with practical experience in designing, implementing, and managing a complete Machine Learning Operations (MLOps) pipeline. Through the NYC Taxi Trip Duration Prediction task, students will apply best practices in automation, reproducibility, and monitoring to deliver a fully functional predictive system. The assignment emphasizes the engineering principles—such as versioning, testing, and continuous integration—that ensure reliability and scalability in real-world machine learning environments.

1. Project Overview

The project involves building an end-to-end MLOps workflow to predict the duration of taxi trips in New York City using real data from the Taxi and Limousine Commission (TLC). Each team will develop and maintain its own repository with the complete workflow, while a centralized server will act as the deployment and evaluation environment, simulating a client production system.

To mirror real-world operational challenges, the data will be released in sequential rounds representing future time periods. This approach allows for the simulation of data drift, performance degradation, and delayed label availability, which are typical in production ML systems.

2. Dataset Description

The dataset (https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page) consists of trip-level records from the NYC Taxi and Limousine Commission (TLC). Students should use the Yellow Taxi data, from 2010–2011 for model development and retain 2012 for validation and testing. Each record includes various features such as:

- Pickup and dropoff datetimes and locations (longitude, latitude, and zone)
- Passenger count
- Trip distance
- Vendor ID and payment information
- Fare amount, tolls, and surcharges (depending on the year)

The predictive target is the trip duration, calculated from pickup and dropoff times. Feature engineering should consider temporal (hour, day, season), spatial (distances, zones), and contextual (rush hour, weather, congestion pricing) variables.

3. Project Requirements

Students must implement a complete MLOps pipeline emphasizing automation, reproducibility, and continuous integration. The project is divided into three main components:

A. Data and Model Management

- Implement reproducible data preprocessing pipelines.
- Track experiments and models using MLflow.
- Use Git and GitHub for source control and collaboration.
- (Bonus) Integrate DVC for data versioning and provenance tracking.

B. Automation and Deployment

- Implement automated testing and training workflows using GitHub Actions.
- Package the inference service as a Docker container.
- Deploy the model and necessary artifacts via automated CI/CD pipelines.
- Implement an inference API (e.g., using FastAPI) exposing a `/predict` endpoint that accepts pickup details and returns the predicted trip duration. The endpoint will receive input data in its raw form, requiring your system to perform all necessary preprocessing steps prior to inference. Each team must ensure that their deployed model correctly handles feature preparation, encoding, and normalization as defined in their preprocessing pipeline. The expected input format is as follows:

```
{
  "data": [
    {
      "vendor_id": "CMT",
      "pickup_datetime": "2010-01-26 07:41:00",
      "dropoff_datetime": "2010-01-26 07:55:00",
      "passenger_count": 1,
      "trip_distance": 3.4,
      "pickup_longitude": -73.985428,
      "pickup_latitude": 40.748817,
      "rate_code": "1",
      "store_and_fwd_flag": "N",
      "dropoff_longitude": -73.977222,
      "dropoff_latitude": 40.752998,
      "payment_type": "CRD",
      "fare_amount": 12.5,
      "surcharge": 0.5,
      "mta_tax": 0.5,
      "tip_amount": 2.3,
      "tolls_amount": 0.0,
      "total_amount": 15.8
    },
    {
      ...
    }
  ]
}
```

The endpoint should return data in the following format:

```
{
  "predictions": [734.5, 652.3]
}
```

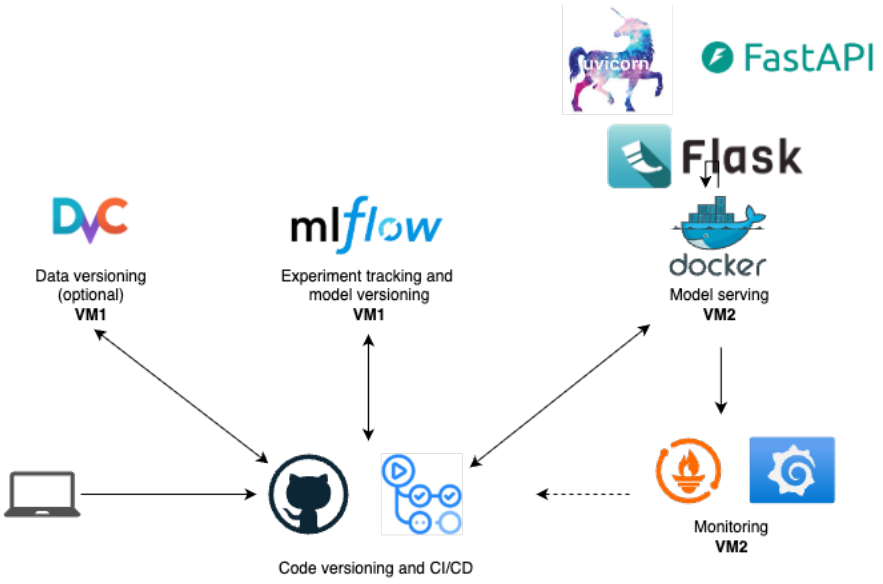
Your system will undergo evaluation across multiple rounds once the inference endpoint is available. A registration placeholder will be provided to allow you to submit your endpoint for centralized testing. Each evaluation round will introduce new datasets representing future months or years, thereby simulating real-world data drift arising from factors such as seasonal variations, traffic patterns, or changes in pricing policies. Following each evaluation round, the corresponding data will be released to your team, enabling the implementation and demonstration of monitoring mechanisms, drift detection strategies, and (optionally) continuous training procedures.

C. Monitoring and Drift Detection

- Monitor deployed models using evaluation metrics such as RMSE, MAE, and related business metrics.
- Detect feature or performance drift caused by temporal and contextual changes.
- Define and demonstrate appropriate alerting strategies.
- (Bonus) Implement automated retraining triggered by drift detection.

4. Technologies and Architecture

Students may use any programming language of their choice; however, Python is strongly recommended due to its extensive ecosystem for data science and MLOps development. Code versioning should be managed through GitHub, with GitHub Actions employed to implement CI/CD workflows. For experiment tracking and model versioning, teams are expected to use MLflow, while Prometheus and Grafana are suggested for system monitoring and performance visualization. In addition to the local development environment, each team will require two virtual machines (VMs): one dedicated to running the MLflow server (and optionally DVC for data versioning), and another to serve as the deployment environment, hosting the production version of the model and monitoring tools. A simplified diagram illustrating the overall architecture and key components is provided below.



5. Milestones

Checkpoint	Date	Deliverable
Checkpoint 1	3 November	Baseline model and versioning setup (Git, MLflow).
Checkpoint 2	24 November	CI/CD automation with containerized model and registration.
Final Submission	12 December	End-to-end system with deployment, monitoring, and drift management.

Students are encouraged to develop the project incrementally according to the checkpoints. Adjustments to intermediate deliverables may be made upon request.

The outcome of this project extends beyond the final artifacts themselves, encompassing the demonstrable evidence that the concepts taught throughout the semester were effectively applied, such as consistent code commits, detailed experiment tracking logs, and comprehensive model versioning records.

6. Deliverables

Each team must submit a complete record of their work demonstrating both the technical artifacts and the application of MLOps principles taught throughout the semester. The deliverables are divided into required and optional (bonus) components.

A. Required Deliverables

GitHub Repository

Organized, public or shared repository containing all source code, configuration files, and documentation. Demonstrated use of Git for version control (regular commits, branches, and commit messages reflecting the development process).

Experiment Tracking and Model Versioning

MLflow logs of experiments, including metrics, parameters, and model artifacts. Clear evidence of model registry usage with proper versioning and promotion between stages (e.g., "Staging", "Production").

CI/CD Pipeline

Fully functional Continuous Integration, Continuous Delivery, and Continuous Deployment workflow implemented via GitHub Actions (or equivalent). Automatic testing, training, model packaging, and deployment steps documented and reproducible.

Monitoring

Evidence of deployed system monitoring (metrics dashboard, logs, or reports) with support for drift detection mechanisms (or similar).

Documentation

Technical report describing:
- Dataset and preprocessing steps
- Model architecture and training strategy
- MLOps pipeline design and implementation
- Monitoring and drift detection approach
- Discussion of results and reflections on challenges faced

Evidence of MLOps Practices

Proof that the team applied the concepts taught during the semester, such as:
- Meaningful code commits and pull requests
- Experiment tracking logs
- Model versioning history
- Workflow automation scripts

Self- and Peer-Assessment

A self-assessment report summarizing how MLOps principles were implemented and lessons learned. An individual peer assessment of team member contributions, evaluating collaboration, effort, and adherence to project goals.

B. Optional (Bonus) Deliverables

Continuous Training (CT)

Automated retraining mechanism triggered by performance degradation or data drift.

Data Versioning (DVC Integration)

Implementation of data lineage and version control using DVC, integrated into the CI/CD workflow.

7. Evaluation and Grading Matrix

Assessment will focus on the adequacy of the model training process and the completeness of the MLOps implementation. The grading emphasizes reproducibility, automation, and the engineering quality of the pipeline rather than only predictive accuracy. Continuous Training (CT) and Data Versioning (DVC) are optional components that can provide bonus credit.

Evaluation Dimension	Weight	Description
Model Training Process (Design & Adequacy)	20%	Sound feature engineering, appropriate validation design, and clear documentation of the training process.
Code Versioning & Reproducibility	20%	Consistent and correct use of Git and GitHub for version control, collaboration, and documentation.
Experiment & Model Versioning (MLflow or equivalent)	20%	Accurate logging of parameters, metrics, and artifacts to ensure full traceability of experiments.
CI/CD Automation	20%	Reliable implementation of automated testing, training, and deployment pipelines.
Deployment & Monitoring	20%	Functional containerized deployment, live inference API, and effective monitoring and drift detection mechanisms.
Bonus: Continuous Training & Data Versioning (CT, DVC)	+10%	Optional advanced integration of continuous retraining and/or data versioning pipelines using DVC or automated workflows.

The final grade is determined by the weighted average of the categories listed above. Projects demonstrating comprehensive automation, rigorous experiment tracking, and effective monitoring will achieve the highest marks. Bonus points are awarded for technically advanced features that go beyond core requirements.

8. Constraints and Notes

- Teams: 4 students each.
- Computing resources: CPU-only environments with limited memory. Focus on efficient models such as Linear Regression, Gradient Boosting, or Random Forests. Heavy deep

neural networks are not recommended.
- Tools: GitHub, GitHub Actions, MLflow, Docker. DVC is optional for bonus.