MLOps Architecture for Automated Retraining and Model Serving

Davide Rossetti

1 Introduction

This document describes the theoretical foundations and engineering design behind an end-to-end MLOps system that performs automated model retraining, drift detection, experiment tracking, and model serving. The system integrates Prefect for orchestration, MLflow for experiment tracking and model versioning, FastAPI for model serving, and Docker for containerization. The goal is to build a reproducible, scalable and production-ready ML architecture that mirrors the workflow used in modern companies.

2 Data Processing and Feature Engineering

The dataset used in this project is the *AirPassengers* time series. The system performs preprocessing and the construction of lag-based features:

$$x_t = [y_{t-1}, y_{t-2}, \dots, y_{t-12}],$$

where the model predicts the next future value:

$$\hat{y}_t = f(x_t).$$

Lag features capture temporal dependencies, a common approach in traditional time-series forecasting models when deep learning architectures (e.g., LSTM, Transformers) are not required.

3 Model Training

A Random Forest Regressor is employed due to its robustness, interpretability, and strong performance on tabular time-series features. The model minimizes the mean squared error:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Training is orchestrated by Prefect and logged with MLflow. At each training run, MLflow stores:

- Model parameters and hyperparameters,
- Performance metrics (MAE, RMSE),
- The trained model artifact,
- Input feature signature,
- A reproducibility snapshot for future reference.

4 Drift Detection

To ensure the model remains reliable over time, the system performs a drift detection step using the two-sample Kolmogorov–Smirnov test. Given two samples X and Y with empirical distributions F_X and F_Y, the KS statistic is defined as:

$$D = \sup_x |F_X(x) - F_Y(x)|.$$

If the corresponding p-value is below a threshold (e.g., 0.05), we conclude that the distributions differ significantly, indicating drift. If drift is detected, the Prefect flow triggers automatic retraining.

5 Model Serving

The latest trained model is exposed through a REST API built with FastAPI. The API accepts a vector of 12 lag values and returns a numerical forecast. FastAPI provides:

- Automatic schema validation via Pydantic,
- Interactive Swagger documentation,
- High performance asynchronous request handling.

This makes it suitable for both batch and real-time inference.

6 Experiment Tracking with MLflow

MLflow serves as the experiment tracking backbone of the system. It maintains full metadata for each training run, ensuring:

- Reproducibility,
- Comparability across model versions,
- Transparent lineage of data, parameters, and artifacts,
- A centralized registry of models.

Each retraining operation appears as a new run in the MLflow UI, enabling visual comparison of performance metrics over time.

7 Orchestration with Prefect

Prefect coordinates the full retraining pipeline, defining the system as a declarative flow:

1. Load or update raw data,
2. Generate features,
3. Check statistical drift,
4. Retrain the model if necessary,
5. Log results to MLflow,
6. Save the new model artifact.

This allows the pipeline to be scheduled, automated, and monitored.

8 Containerization with Docker

All components (API, Prefect, MLflow) are encapsulated inside Docker containers. Docker Compose orchestrates them, creating a multi-service environment with shared volumes:

- `models/` — the latest trained model,
- `mlruns/` — MLflow tracking data,
- `data/` — input data.

Containerization ensures full reproducibility and makes deployment trivial on any cloud provider.

9 Conclusion

This project implements a complete MLOps system that covers the entire machine learning lifecycle: data ingestion, feature engineering, drift detection, retraining, model tracking, and API deployment. The architecture is modular, scalable, and production-ready.