Project Report

Pearls AQI Predictor – AQI Forecasting System

Author: Neha Maqbool

Introduction

Air pollution has become a serious environmental and health concern in large cities such as Karachi. Increasing levels of pollutants like PM2.5, PM10, NO_2, and CO negatively affect human health and daily life. Therefore, predicting future Air Quality Index (AQI) values is important for early warnings and preventive actions.

This project, **Pearls AQI Predictor**, presents a **fully automated, serverless, and scalable AQI forecasting system**. The system collects real-time environmental data from APIs, performs feature engineering, trains machine learning models, and predicts AQI for the next **three days** using an interactive web dashboard.

The solution follows modern **MLOps practices**, including:

- Automated data pipelines
- Feature Store integration
- Model Registry
- Continuous training
- CI/CD automation
- Streamlit dashboard

The final system is designed to be **production-ready, scalable, and reliable**.

Problem Statement

Karachi's air quality changes frequently and manual monitoring cannot provide future forecasts. Static reports only show current values but do not predict upcoming pollution levels.

The goal of this project was to build:

✅ A fully automated AQI prediction system
✅ Forecast for next 3 days
✅ API-based ingestion (no CSV files)
✅ Feature Store & Model Registry
✅ Automated pipelines
✅ Interactive dashboard

System Architecture

The project is organized into modular components:

```
AQI_PROJECT/
├── data_pipeline/
├── feature_pipeline/
├── training_pipeline/
├── models/
├── app/
├── .github/workflows/
```

This modular structure improves maintainability, scalability, and reusability.

Data Collection (Data Pipeline)

File: `fetch_api_data.py`

Raw AQI and weather data were fetched directly from external APIs.

Pollutants collected:

- PM2.5, PM10
- NO_2, SO_2
- CO, O_3

Weather:

- Temperature
- Humidity
- Wind speed

Historical data for **3–4 months** was backfilled to create training datasets.

✅ Fully API-based ingestion
✅ No CSV files used

Feature Engineering

Files:

- `feature_engineering.py`
- `feature_store.py`
- `feature_view.py`

Raw data was converted into model-ready features.

Created features:

- Hour, Day, Month, Weekday
- Lag AQI values

- AQI change rate
- Rolling averages

These features help the model capture temporal trends and improve prediction accuracy.

All features were stored in **Hopsworks Feature Store**.

Model Training Pipeline

Folder: `training_pipeline/`

Steps:

1. Fetch historical features from Feature Store
2. Split dataset (80% train / 20% test)
3. Train multiple models
4. Evaluate performance
5. Register best model

Models used:

- Linear Regression
- Random Forest
- XGBoost

Metrics:

- RMSE
- MAE

- R² Score

The best-performing model was saved in the **Hopsworks Model Registry**.

Prediction Pipeline

The system:

- Loads the latest features from Feature Store
- Loads the best model from registry
- Predicts AQI for next 3 days
- Sends results to the dashboard

This enables **real-time forecasting**.

Web Application (Streamlit Dashboard)

Files: `dashboard.py`, `backend.py`

Features:

- Real-time AQI display
- 3-day forecast charts
- KPIs
- Interactive visualizations
- Clean and simple interface

The dashboard allows users to easily monitor air quality trends.

Automation (CI/CD)

File: `.github/workflows/aqi_pipeline.yaml`

Automation includes:

- Hourly feature updates
- Daily model retraining
- Automatic pipeline execution

This ensures continuous learning and fresh predictions without manual work.

Problems Faced & Solutions

During development, several practical issues were encountered and resolved:

1. Module Import Error

Problem: Streamlit could not detect project modules.
Solution: Added root path to `sys.path`, added `__init__.py`, and ran app from root directory.

2. Hopsworks Version Mismatch

Problem: Client and server versions were incompatible.
Solution: Installed matching versions of `hopsworks` and `hsfs`.

3. Missing Libraries

Problem: Training failed due to missing packages like xgboost.
Solution: Installed all required dependencies using pip.

---

4. API Authentication Errors

Problem: Feature Store access failed due to incorrect API key.
Solution: Configured API key correctly using environment variables.

---

5. Feature Store Saving Issues

Problem: Data insertion errors due to inconsistent schema.
Solution: Split dataset into 80% training and 20% testing and standardized feature structure before saving.

---

6. Duplicate Data Stored

Problem: Repeated API calls inserted duplicate rows, causing storage waste and biased training.
Solution: Removed duplicates and stored only new records using deduplication logic.

---

7. Model Training Failure

Problem: Duplicate and noisy data caused unstable training.
Solution: Cleaned dataset and retrained the model successfully.

---

8. Dashboard Not Fetching Data

Problem: Schema mismatch prevented Streamlit from retrieving features.
Solution: Corrected feature engineering code and aligned feature names with the Feature Store.

Outcome

✅ End-to-end AQI prediction system
✅ API-based ingestion
✅ Feature Store integration
✅ Model Registry
✅ Multiple ML models
✅ Automated CI/CD pipelines
✅ Interactive dashboard
✅ Clean and scalable architecture

Conclusion

The **Pearls AQI Predictor** successfully demonstrates a **production-grade MLOps workflow** for AQI forecasting. The system automates data collection, feature engineering, model training, and deployment, while providing real-time predictions through a dashboard.