Air Quality Index (AQI) Predictor

Introduction

Air pollution has become one of the most pressing environmental challenges affecting urban populations across the world. Accurate forecasting of the Air Quality Index (AQI) is essential for public health awareness, policy planning, and early warning systems. The objective of this project was to design and implement a complete end-to-end AQI forecasting system that not only predicts future air quality levels but also follows modern MLOps practices for automation, reproducibility, and scalability.

The system was developed to forecast AQI values for the next 72 hours using historical pollutant and weather data. In addition to model development, the project includes automated data ingestion, feature engineering, feature storage using Hopsworks Feature Store, model versioning using a Model Registry, daily retraining through GitHub Actions, and an interactive frontend dashboard built using Streamlit. The final system resembles a real-world production pipeline rather than just a standalone machine learning model.

Problem Statement

The primary goal of this project was to build a robust forecasting model capable of predicting AQI values up to 72 hours ahead. Since air quality is highly dependent on past pollutant concentrations and weather patterns, the problem was treated as a supervised regression task with time-series feature engineering. Beyond model accuracy, the system was designed to continuously update itself by fetching new data, retraining models daily, and maintaining version control for reproducibility.

Data Collection and Automated Pipeline

The system collects hourly pollutant and weather data through external APIs. Pollutant variables include PM2.5, PM10, CO, SO2, NO2, and O3, while weather variables include temperature, humidity, wind speed, and atmospheric pressure. These environmental indicators directly influence AQI behavior and therefore form the backbone of the predictive system.

To ensure automation, GitHub Actions workflows were implemented. On an hourly basis, the workflow fetches the latest pollutant and weather data and appends it to the existing historical dataset. On a daily schedule, another workflow retrains all machine learning models using the updated dataset. This automation ensures that the system continuously learns from new environmental patterns without manual intervention. The integration between GitHub Actions and Hopsworks was carefully configured using secure API authentication to maintain a reliable production workflow.

Feature Engineering

Air pollution exhibits strong temporal dependency, meaning that present AQI levels are heavily influenced by past pollutant concentrations and previous AQI values. To capture this behavior, lag-based feature engineering was implemented. For AQI and each pollutant, lag features were created for 1 hour, 24 hours, and 72 hours. These lag values allow the model to understand short-term fluctuations, daily cyclic patterns, and longer-term trends.

Each feature included in the Feature Store was carefully selected based on environmental relevance. AQI itself was included not only as the target variable but also in lagged form to support autoregressive forecasting. PM2.5 was incorporated because it is one of the most significant contributors to AQI calculations, particularly in urban areas. PM10 captures the effect of coarse particulate matter, while CO indicates combustion-related pollution, often linked to traffic congestion. SO2 represents industrial emissions, NO2 reflects vehicular activity, and O3 accounts for photochemical reactions influenced by sunlight.

Weather variables were equally important. Temperature affects chemical reaction rates in the atmosphere, humidity influences particulate suspension, wind speed determines pollutant dispersion, and atmospheric pressure can trap pollutants near the surface. By combining pollutant and weather data with temporal lags, the model was provided with a comprehensive representation of AQI dynamics.

Feature Store Implementation

All engineered features were stored in the Hopsworks Feature Store. Using a centralized feature store ensured consistency, reproducibility, and proper version control. The Feature Store architecture separates offline training data from potential online serving use cases, making the system scalable and production-ready.

During implementation, strict naming conventions had to be followed since Hopsworks enforces schema validation rules. Column names were sanitized to lowercase and formatted to comply with database constraints. The Feature Store became the single source of truth for training data used by all models.

Model Development and Comparison

Three machine learning models were implemented to compare performance and ensure robustness: Random Forest Regressor, Gradient Boosting Regressor, and XGBoost Regressor.

Random Forest was used as a strong baseline model because of its stability and ability to handle non-linear relationships. Gradient Boosting was implemented to improve sequential learning by correcting errors from previous trees. XGBoost was selected due to its regularization capability, optimized boosting strategy, and strong performance on structured tabular datasets. Each model was trained using time-aware train-test splitting to avoid data leakage.

Model Registry Implementation

To maintain proper version control and lifecycle management, a Model Registry was implemented in Hopsworks. All three models, Random Forest, Gradient Boosting, and XGBoost were registered along with their evaluation metrics and training metadata. Each retraining cycle generates a new version of the models, allowing historical comparison and rollback if necessary.

The Model Registry ensures that the best-performing model can be promoted to production. By storing evaluation metrics such as RMSE and MAE, the system automatically identifies which model performs best on updated data. This approach follows modern MLOps standards where model management is treated as a structured engineering process rather than a one-time experiment.

Daily Retraining and Continuous Learning

One of the most important aspects of this project is its automated retraining pipeline. Every day, the updated dataset stored in the Feature Store is used to retrain all three models. This continuous retraining mechanism allows the system to adapt to seasonal changes, pollution spikes, or unexpected environmental events.

By combining GitHub Actions automation with the Hopsworks Model Registry, the system achieves continuous integration and continuous training (CI/CT). This significantly reduces model drift and ensures consistent predictive performance over time.

72-Hour Forecasting Strategy

Since the system uses lag-based features, multi-step forecasting was implemented recursively. The model first predicts the AQI for the next hour. That prediction is then appended to the dataset, new lag features are recalculated, and the next prediction is generated. This process continues until 72 hours of forecasts are produced. Although recursive forecasting requires careful management of feature updates, it allows dynamic and realistic long-term prediction.

Streamlit Frontend Dashboard

To make the system accessible and user-friendly, a frontend interface was developed using Streamlit. The dashboard allows users to visualize real-time AQI predictions and view a complete 72-hour forecast in graphical form. Pollutant trends are displayed through interactive plots, and AQI categories such as Good, Moderate, and Unhealthy are clearly indicated.

The Streamlit interface connects directly to the trained model and presents predictions in a clean and intuitive layout. This transforms the project from a backend machine learning pipeline into a fully usable application suitable for real-world deployment.

Challenges Faced During Implementation

Several technical challenges were encountered during the project. Initially, authentication and region mismatches in Hopsworks caused connection failures, which were resolved by configuring the correct backend host and API keys. Spark engine misconfiguration also created execution issues, which were addressed by forcing the Python execution engine. Feature name validation errors required renaming columns to comply with Feature Store constraints. Client–backend version mismatches introduced compatibility errors, which were fixed by upgrading the Hopsworks client library.

Additionally, configuring automated workflows through GitHub Actions required careful handling of environment variables and secure API credentials. Recursive forecasting logic also required meticulous implementation to correctly update lag features at each prediction step.

Conclusion

This project successfully demonstrates a complete, production-oriented AQI forecasting system built using modern MLOps principles. It integrates automated data ingestion, feature engineering, Feature Store management, multi-model comparison, Model Registry versioning, daily retraining, recursive forecasting, and an interactive Streamlit dashboard.

The system is scalable, reproducible, and aligned with real-world deployment practices. By combining environmental science with machine learning and cloud-based MLOps tools, this project represents a comprehensive solution for intelligent air quality monitoring and forecasting.