Pearls AQI Predictor: End-to-End Serverless Air Quality Forecasting

Final Internship Project Report

Muhammad Akbar

Location: Karachi, Pakistan

1. Executive Summary

During this internship, I designed and deployed the **Pearls AQI Predictor**, a 100% serverless machine learning system built to forecast the Air Quality Index (AQI) for Karachi over a 72-hour horizon. The system automates the entire ML lifecycle, from real-time data ingestion and feature engineering to model training and interactive visualization. Using Hopsworks, GitHub Actions, and Streamlit, I built a fully automated and low-maintenance predictive pipeline with a final Root Mean Square Error (RMSE) of 26.89.

2. Project Objectives & Scope

The primary objective of this internship project was to build a robust, automated pipeline capable of predicting PM2.5-based AQI three days into the future. The specific requirements included:

- **Automated Data Pipelines:** Continuous fetching of historical and live weather/pollutant data.
- **Serverless Infrastructure:** Utilizing cloud-based feature stores and model registries without provisioning dedicated servers.
- **Machine Learning:** Training, evaluating, and deploying a highly accurate forecasting model.
- **Interactive Dashboard:** A web-based UI featuring real-time predictions, historical trends, and hazard alerts.
- **Explainable AI:** Integrating SHAP to provide transparency into the model's decision-making process.

3. System Architecture & Technology Stack

The system operates on a decoupled, serverless architecture, ensuring scalability and ease of maintenance.

Core Technologies Used:

- **Data Sources:** Open-Meteo API (unified weather and air quality historical/live data).
- **Feature Store & Model Registry:** Hopsworks.
- **Machine Learning:** Python, Scikit-learn, XGBoost, SHAP.
- **Orchestration (CI/CD):** GitHub Actions.
- **Frontend/Deployment:** Streamlit Cloud, Plotly for interactive data visualization.

4. Methodology & Implementation

4.1. Data Engineering (The Feature Pipeline)

I implemented a data ingestion script (fetch_features.py) as the foundation of the predictor. Initially, a backfill operation was performed to gather historical data to train the model.

- **Feature Engineering:** I transformed the raw data (Temperature, Humidity, Wind Speed, PM2.5, PM10, NO2, O3) into engineered temporal features designed to improve predictive performance. This included calculating cyclical time encoding (sine/cosine of hours and months) to capture seasonality, and computing dynamic rolling averages (3-hour and 24-hour PM2.5 means) and 12-hour trend differentials.
- **Storage:** The processed data is pushed to the Hopsworks Feature Store, serving as a single source of truth for both training and inference.

4.2. Model Development (The Training Pipeline)

I implemented a training pipeline (train_model.py) that pulls the latest features from Hopsworks, generate sequences for time-series forecasting (using a 24-hour input window to predict a 72-hour horizon), and evaluate multiple algorithms.

- **Algorithm Selection:** During development, several algorithms were evaluated, including Ridge Regression and Random Forest. While Random Forest captured some non-linearities, **XGBoost (wrapped in a MultiOutputRegressor)** was ultimately selected as the best performer. It effectively handled the complex, non-linear relationships in meteorological data and sudden pollution spikes, achieving a final **RMSE of 26.89**.
- **Artifact Management:** The trained model, along with its scaler and metadata, is automatically serialized and registered in the Hopsworks Model Registry.

4.3. Pipeline Automation (CI/CD)

To maintain model freshness and data accuracy, the pipelines were fully automated using GitHub Actions:

- **Hourly Feature Pipeline:** Runs every hour to fetch the latest live data from Open-Meteo and update the Hopsworks Feature Group.
- **Daily Training Pipeline:** Runs at midnight UTC to retrain the XGBoost model on the most recent dataset, automatically incrementing the model version in the registry if performance criteria are met.

4.4. Application & Explainability

The user-facing component (app_streamlit.py) dynamically pulls the latest model and data from Hopsworks.

- **Interactive Dashboard:** Displays current metrics, a 7-day historical trajectory, and the 72-hour AI forecast, automatically localized to Asia/Karachi time.
- **Alerting System:** Implemented logic to warn users if the forecasted AQI crosses into "Unhealthy" or "Hazardous" thresholds.
- **SHAP Integration:** Included an Explainability tab that computes SHAP values in real-time, displaying a bar chart of the top 15 features driving the immediate next hour's prediction.

5. Engineering Challenges & Solutions

Building a production-ready system introduced several technical hurdles that required strategic problem-solving:

- **Handling Timezone-Aware vs. Naive Datetime Conflicts:** Integrating UTC-based cloud data with local Karachi timezones caused significant synchronization bugs and "future data leakage" in the dashboard.
    - **Solution:** Enforced strict UTC localization during data ingestion within the Feature Store, and converted timestamps to Asia/Karachi only at the Streamlit visualization layer.
- **Dependency Conflicts During Deployment:** Deploying to Streamlit Cloud triggered silent C-extension clashes between newer NumPy versions and legacy Pandas distributions.
    - **Solution:** Implemented strict version pinning in the requirements.txt and pruned unnecessary deep-learning libraries to reduce bloat and ensure stable, lightweight container builds.

- **Validating Auto-Regressive Model Responsiveness:** A major risk in time-series forecasting is the model defaulting to flat, unresponsive historical averages.
    - **Solution:** I performed additional validation tests to ensure the auto-regressive lag features accurately captured sudden state corrections, allowing the XGBoost model to immediately adjust its 72-hour trajectory when real-world PM2.5 levels abruptly spiked.
- **Managing Data Continuity:** Transitioning from historical batch backfilling to live hourly API polling exposed a risk of missing temporal data.
    - **Solution:** I implemented interpolation and careful alignment of timestamp boundaries between the batch and streaming phases to ensure an unbroken sequence for the XGBoost sliding window.
- **Cold Start Latency in Serverless Compute:** Because the stack is entirely serverless, initial dashboard loads experienced cold starts while the container initialized, authenticated with Hopsworks, and downloaded artifacts.
    - **Solution:** Utilized Streamlit's @st.cache_resource and @st.cache_data decorators to heavily cache the Hopsworks connection, model artifacts, and batch data, significantly reducing latency for subsequent user interactions.

6. Conclusion

The Pearls AQI Predictor fulfills the defined project requirements, showing that a 100% serverless architecture can support a production-grade ML workflow. By combining automated data pipelines, the final XGBoost forecasting model, and an intuitive, explainable user interface, this project delivers a practical tool for monitoring and anticipating air quality in Karachi. This project significantly strengthened my understanding of production-level ML systems beyond model training, particularly in areas such as MLOps, automation, and deployment reliability.