Karachi AQI Predictor

Air Quality Index Prediction System
Using Machine Learning & MLOps

Project Summary

A real-time, automated AQI prediction system for Karachi, Pakistan that collects live weather and pollution data, engineers features, trains machine learning models daily, and serves 72-hour forecasts via an interactive Streamlit dashboard.

Author: Azhab Safwan Babar

February 2026

Contents

Introduction & Motivation

This project, designed by 10Pearls, was fundamentally different from the typical Kaggle-based assignments I had completed during my Machine Learning course at IBA Karachi. Instead of working with a preprocessed and ready-made dataset, this project required building a complete end-to-end pipeline, starting from real-time data collection through APIs to feature engineering, model training, and deployment. This made the project significantly more challenging and closer to real-world machine learning systems. **building the dataset from scratch**.

The core prediction task is to forecast the **Air Quality Index (AQI)** category for Karachi, Pakistan up to **72 hours in advance**. Understanding air quality is a meaningful challenge because AQI is influenced by two fundamentally different types of factors:

1. **Pollutants** — particulate matter such as PM2.5, which directly determines air quality ratings.
2. **Weather conditions** — temperature, humidity, wind speed, precipitation, and atmospheric pressure, which influence how pollutants disperse or accumulate.

The realization that both of these dimensions had to be captured, combined, and modelled together made this project genuinely educational beyond the classroom.

The Dataset Decision

An important early design question was *how* to define and predict AQI. Two approaches were considered:

- **Method A (Categorical API):** Use the OpenWeather API's built-in AQI category output (values 1–5) directly as the prediction target.
- **Method B (PM2.5 Regression):** Predict the raw PM2.5 concentration value first, then convert it to an AQI category using the official EPA formula.

Method A was comparatively easier to implement because the AQI value was directly provided by the weather API. This allowed the pipeline to focus primarily on feature engineering and forecasting, without requiring additional transformations to derive AQI from pollutant concentrations. As a result, the overall workflow remained straightforward and computationally efficient.

In contrast, Method B was significantly more complex. Instead of receiving AQI directly from the API, this approach required first training regression models to predict PM2.5 concentrations from meteorological and pollutant-related features. Once PM2.5 was predicted, it then had to be converted into AQI using the standard mathematical formula. This introduced multiple additional stages into the pipeline, including PM2.5 regression, AQI conversion, and subsequent validation of results.

Because of these extra steps, Method B became a longer and more intricate process, increasing both implementation effort and the likelihood of error propagation across stages. Any inaccuracies in PM2.5 prediction could directly affect the final AQI estimation, making the overall system harder to tune and evaluate. Therefore, Method A was favored due to its simplicity, reliability, and reduced computational overhead, while still producing meaningful and practical predictions.

Dataset

Overview

Table 1: Dataset Characteristics

Property	Value
Location	Karachi, Pakistan (24.8608°N, 67.0104°E)
Collection Period	December 1, 2025 – February 17, 2026
Duration	78 days
Frequency	Hourly
Total Records	1,872 hourly data points
Feature Count	30 engineered features
Target Variable	AQI Category (Classes: 2, 3, 4, 5)
Data Sources	Open-Meteo (weather) + OpenWeather (AQI)

Feature Groups

The 30 features were carefully engineered across several categories to capture both instantaneous conditions and historical patterns.

Table 2: Engineered Feature Categories

Category	Features	Description
Raw Weather	5	Temperature, humidity, wind speed, wind direction, cloud cover
Raw Pollution	8	PM2.5, PM10, CO, NO, NO_2, O_3, SO_2, NH_3 (all in $\mu g/m^3$)
Target	1	AQI category (from OpenWeather API)
Time-Based	5	Hour, day, month, day of week, cyclical encodings (sin/cos)
Lag Features	3	AQI values at 24h, 48h, 72h ago (prevents data leakage)
Rolling Statistics	6+	72-hour window means and standard deviations
Derived Features	8+	Weather interactions, trend indicators

AQI Category Distribution

The target variable has four classes, each corresponding to an air quality level:

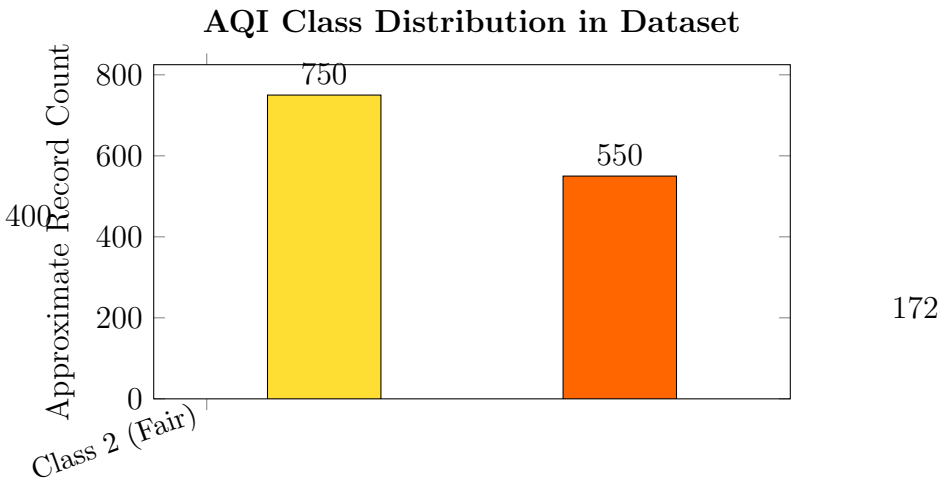

Figure 1: Approximate distribution of AQI categories across 1,872 hourly records

System Architecture

The system is designed as a fully automated MLOps pipeline with no manual intervention required after initial setup.

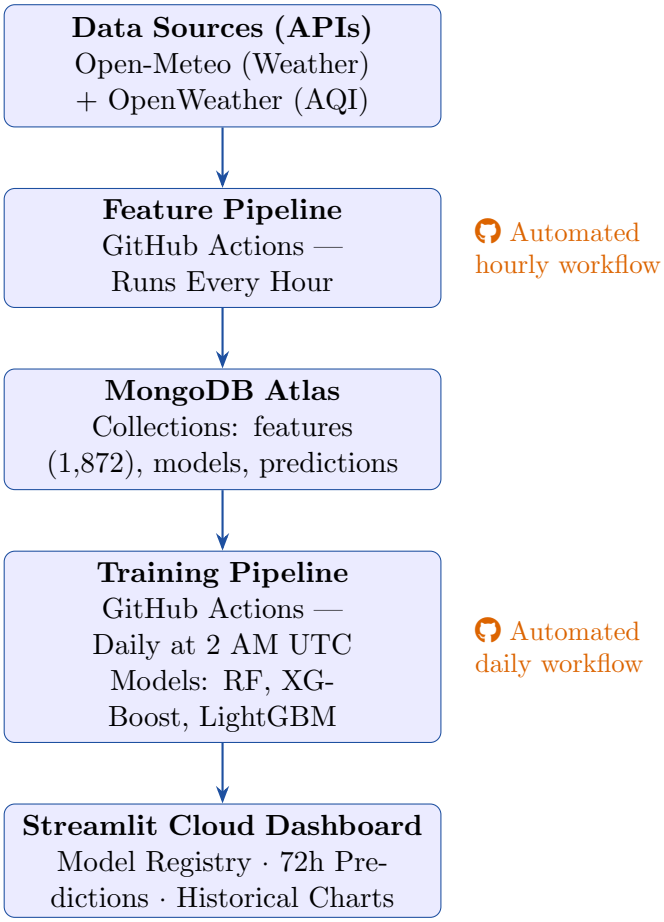

Figure 2: End-to-end MLOps pipeline architecture

Technology Stack

Table 3: Technology Stack

Component	Technology	Purpose
Data Collection	Open-Meteo API, OpenWeather API	Live weather & AQI data
Feature Engineering	Python (pandas, numpy)	Lag, rolling, derived features
Database	MongoDB Atlas	Feature & model storage
ML Models	scikit-learn, XGBoost, LightGBM	Classification
Automation	GitHub Actions	Hourly & daily workflows
Serving	Streamlit Cloud	Interactive dashboard
Model Persistence	Joblib (.joblib files)	Local model serialization

Machine Learning Models

Validation Strategy

The dataset was split using a **temporal split** (no shuffle) to prevent data leakage — a common mistake in time-series ML where the model accidentally "sees the future" during training.

Table 4: Train/Test Split

Split	Records	Proportion
Training Set	1,440	80%
Test Set	360	20%
Cross-Validation	5-fold stratified	On training set

Model Performance

Three models were trained and evaluated. LightGBM emerged as the best performer.

Table 5: Model Performance Comparison

Model	Test Accuracy	Train Accuracy	CV Accuracy (5-fold)	Notes
LightGBM	**93.9%**	100.0%	93.1%	Best performer
XGBoost	91.9%	97.1%	88.7%	Slight overfittin
Random Forest	65.6%	69.1%	64.2%	Conservative base

Accuracy Visualisation

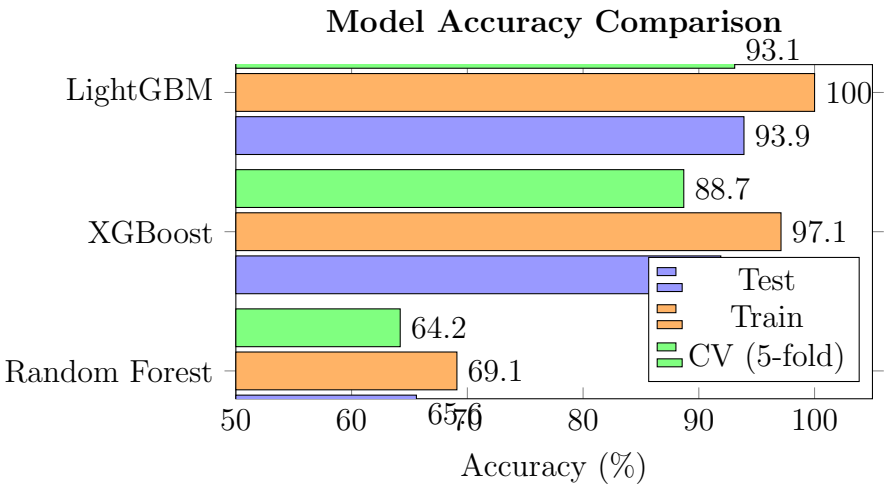

Figure 3: Comparative model accuracy across train, test, and cross-validation splits

Model Registry

All three models are tracked in a **MongoDB model registry** with full metadata including hyperparameters, performance metrics, and training timestamps. They are also serialized locally as `.joblib` files for fast inference.

Exploratory Data Analysis (EDA)

Before modelling, extensive EDA was performed to understand the data and inform feature engineering decisions.

Feature Engineering Pipeline

The EDA phase revealed that raw hourly readings alone were insufficient. Key insights:

- AQI conditions are **autocorrelated**: what the air quality is now is strongly related to what it was 24, 48, and 72 hours ago.
- **Cyclical time encodings** (sin/cos of hour and day) better capture daily and weekly periodicity than raw integer values.
- **Rolling statistics** over a 72-hour window capture the trend and variability of conditions, not just the snapshot value.
- **Weather interactions** (e.g., high humidity + low wind = poor dispersion) can be captured as derived features.

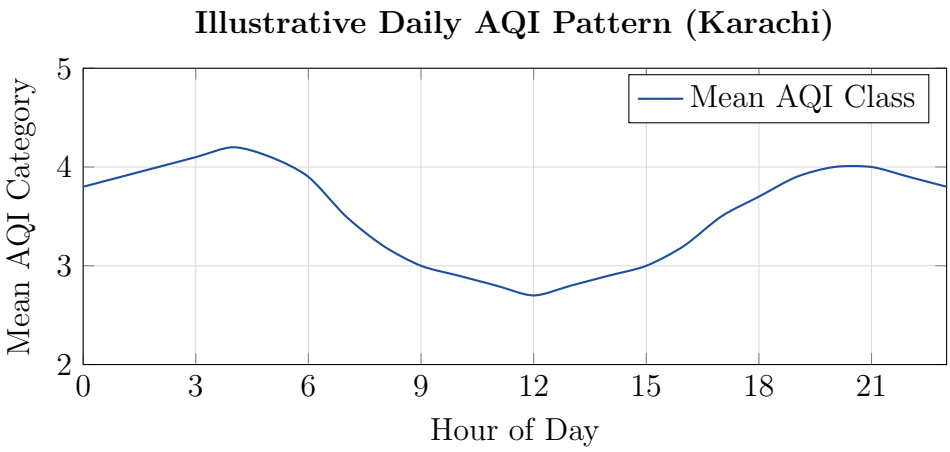

Figure 4: Illustrative diurnal AQI pattern showing typically better air quality midday (higher dispersion) and worse quality at night and early morning

Automation & MLOps

GitHub Actions Workflows

A key learning was how to use **GitHub Actions** to automate repetitive tasks — essentially the ML equivalent of a `for` loop that runs on a schedule in the cloud.

Table 6: GitHub Actions Workflows

Workflow	Schedule	Actions
`hourly-data-collection.yml`	Every hour at :00	Fetch weather + AQI from APIs, engineer features, store in MongoDB
`daily-model-training.yml`	Daily at 2:00 AM UTC	Train all 3 models, evaluate, update model registry, commit `.joblib` files to Git

Why Automate?

- **Data freshness:** Hourly collection ensures the model always has recent data.
- **Model freshness:** Daily retraining prevents model drift as Karachi's seasonal pollution patterns evolve.
- **Zero manual effort:** After initial setup, the system runs indefinitely without human intervention.

Challenges & Problems Faced

1. **Building a Dataset from Scratch:** Unlike the Kaggle competition in class where the dataset was provided, this project required sourcing, cleaning, and combining data from two entirely separate APIs (Open-Meteo for weather and OpenWeather for AQI). Ensuring the timestamps aligned correctly between the two sources required careful engineering. The realization of how much effort goes into real-world data collection was a major learning moment.

2. **Choosing the Right AQI Prediction Method:** Deciding between predicting AQI as a direct categorical output vs. first predicting PM2.5 and then converting it using the EPA formula was confusing. The PM2.5 route introduced compounded error: regression errors in PM2.5 would propagate into the AQI classification. Ultimately, using the API's categorical AQI output directly was simpler and more reliable, but arriving at this decision required understanding how AQI is actually computed.

3. **Understanding GitHub Actions:** Scheduling automated workflows was a completely new concept. Moving from "run this Python script manually" to "this script runs every hour in the cloud automatically" required understanding YAML workflow syntax, GitHub Secrets for API keys, and how to trigger jobs on a cron schedule. The first workflows failed several times before they ran successfully end-to-end.

4. **MongoDB Integration:** Connecting the feature pipeline, the model trainer, and the Streamlit dashboard to the same MongoDB Atlas database without conflicts — especially with concurrent reads and writes from GitHub Actions and the dashboard — required careful database schema design and proper use of connection strings and Streamlit secrets management.

Key Learnings

What I Learnt

1. **Real-world data is hard:** Creating a dataset is far more work than using a pre-made one. API rate limits, missing values, timestamp mismatches, and schema design are all real engineering problems.
2. **Feature engineering matters more than model choice:** The lag and rolling features contributed more to model performance than switching between RF, XGBoost, and LightGBM.
3. **Time-series requires special care:** Temporal splitting and lag features are non-negotiable for honest evaluation.
4. **MLOps closes the loop:** Training a good model is only half the job. Automated pipelines, model registries, and deployment are what make ML useful in the real world.
5. **GitHub Actions = scheduled automation:** A cron job in the cloud that keeps your data and models fresh without any manual work.

Project Structure

Listing 1: Repository Structure

```
karachi-aqi-predictor/
          .github/workflows/
                hourly-data-collection.yml   # Hourly feature collection
                daily-model-training.yml     # Daily retraining
          src/
                config.py                    # Configuration management
                database.py                  # MongoDB handler
                feature_pipeline.py          # Hourly data collection &
   engineering
                training_pipeline.py         # Model training &
   evaluation
                model_registry.py            # Model loading &
   predictions
          models/                          # Serialized models
                randomforest_model.joblib
                xgboost_model.joblib
                lightgbm_model.joblib
                scaler.joblib
                label_encoder.joblib
                feature_columns.joblib
          app.py                           # Streamlit dashboard
          requirements.txt
          README.md
```

SHAP Analysis — Model Interpretability

Traditional machine learning models like LightGBM, XGBoost, and Random Forest are often called **"black boxes"** — they make accurate predictions but offer no explanation of *why*. SHAP (SHapley Additive exPlanations) solves this by computing exactly how much each feature contributes to each individual prediction.

Core Concept

SHAP is grounded in **Shapley values** from cooperative game theory. Each feature is treated as a "player" in a game, and its contribution is calculated by:

1. Computing predictions with and without that feature across all possible feature combinations.
2. Averaging contributions fairly so no feature is over- or under-credited.
3. Guaranteeing that all SHAP values sum exactly to the difference between the model's prediction and the baseline (average AQI).

Why SHAP Matters for This Project

Table 7: Benefits of SHAP Interpretability

Benefit	What it Provides
Transparency	Understand which environmental factors actually drive AQI predictions
Trust	Validate that the model uses sensible patterns (e.g. high PM2.5 → worse AQI)
Debugging	Detect if the model relies on spurious or accidental correlations
Scientific Insight	Discover which factors matter most for Karachi's air quality

Key Findings

What SHAP Revealed About the Model

1. **Weather dominates:** `cloud_cover` (SHAP: 2.45) is the single most influential feature, followed by `temperature` and `wind_direction`. Weather conditions control how pollutants disperse in the atmosphere.
2. **Temporal patterns matter:** `day_of_week_sin` (SHAP: 1.98) and `hour_sin` capture human behavioural patterns — traffic, industry, and cooking cycles that vary by time of day and day of week.
3. **Lag features are critical:** `aqi_lag_48h` (1.35), `aqi_lag_72h` (1.23), and `aqi_lag_24h` (1.12) confirm that recent AQI history is a strong predictor of future AQI, validating the feature engineering decisions.

4. **Rolling mean captures trends:** `aqi_rolling_mean_72h` (1.73) reflects multi-day pollution accumulation or dispersal trends.
5. **Raw pollutants rank lower than expected:** `pm2_5` (0.97) and `pm10` (0.55) have lower SHAP values — not because they are unimportant, but because their information is already captured by the AQI lag and rolling features, which encode historical pollution levels.

Individual Prediction Explanation (Waterfall Chart)

Beyond global importance, SHAP also explains *individual* predictions via a waterfall chart. Starting from the baseline average AQI, each feature either pushes the prediction **higher** (worse air quality) or **lower** (better air quality).

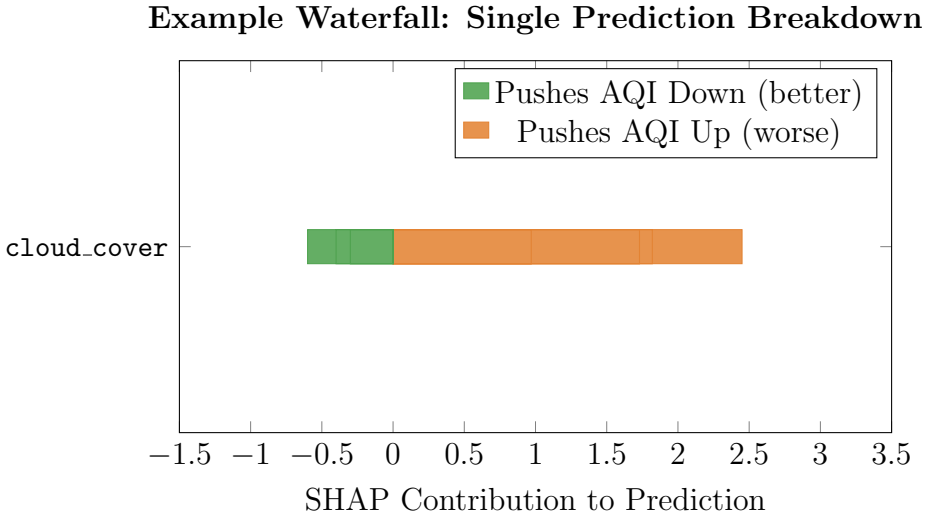

Figure 5: Illustrative waterfall chart for a single prediction. Each bar shows how much a feature shifts the prediction above or below the baseline AQI.

Technical Implementation

Table 8: SHAP Technical Details

Property	Detail
Algorithm	TreeExplainer (optimised for tree-based models)
Backend	Fast C++ implementation for LightGBM / XGBoost
Consistency	SHAP values guaranteed to sum exactly to the prediction
Caching	Explainer cached for 1 hour on Streamlit to avoid recomputation
Scope	Both global (all predictions) and local (individual predictions)

This interpretability layer transforms the AQI predictor from a *"magic algorithm"* into an **explainable decision support tool** that domain experts and stakeholders can understand, question, and trust.

Conclusion

The Karachi AQI Predictor is more than a machine learning model — it is a complete, production-grade MLOps system. Starting from zero data, the project progressed through API integration, dataset creation, feature engineering, model training, automated scheduling, database management, and live deployment.

The best model, **LightGBM**, achieves **93.9% test accuracy** and **93.1% cross-validation accuracy**, providing reliable 72-hour AQI forecasts for one of South Asia's most populous cities.

More importantly, the experience of building this end-to-end taught lessons that no pre-packaged Kaggle dataset ever could — from the difficulty of sourcing and cleaning real data, to the engineering discipline required to prevent data leakage, to the satisfaction of watching an automated system run in the cloud without human intervention.