CPE 393 Machine Learning Operations Project Report

Prepared by

Chayawat	Anaroch	65070501012
Siwarat	Laoprom	65070501052
Kamolpop	Poonsawat	65070501065
Poomrapee	Moungnoi	65070501084
Bhagya	Saranunt	65070501092

Present to

Dr. Aye Hninn Khine

Asst. Prof. Dr. Santitham Prom-on

This report is part of the CPE 393 Machine Learning Operations course

Faculty of Engineering, Department of Computer Engineering

King Mongkut's University of Technology Thonburi

2nd semester, academic year 2025

Abstract

Air pollution, particularly PM2.5, poses a serious health threat in Bangkok. This project develops a machine learning model to forecast PM2.5 concentration levels using comprehensive meteorological and air quality data. The dataset was collected from OpenMeteo's Air Quality and Historical Weather APIs, including features such as temperature, humidity, PM10, CO, NO_2, O_3, and CH_4. These variables were selected for their proven relevance to PM2.5 prediction patterns. The cleaned data are split into three contiguous blocks by date. There is 70% for training, the next 10% for validation, and the final 20% for testing. No shuffling is applied so that each set preserves the true chronological order required for time-series forecasting. In this project, three models were evaluated: XGBoost, Random Forest, and LSTM, assessed using accuracy metrics and mean absolute percentage error (MAPE) for both single-step and multi-step predictions. Experiment tracking was performed using Weights & Biases (wandb) to ensure reproducibility, version control, and efficient comparison of model performance across different training runs. The project achieved the highest overall performance using the XGBoost model, with 93.4% single-step and 91.7% multi-step accuracy, corresponding to MAPE values of 6.6% and 8.3% respectively. Random Forest, while slightly lower in single-step accuracy, excelled in long-term forecasting with the best multi-step MAPE of 2.9%. LSTM underperformed both tree-based models, particularly in single-step forecasting. Finally, the system incorporates a FastAPI-based API for serving prediction results. The model is containerized using Docker to ensure consistency across environments. A Streamlit web application provides an interactive interface for visualizing PM2.5 forecasts.

Architecture

Techstack

Data and Model Development

Tool	Description
	Weight & Bias • Model Monitoring • Logging Key Metrics for evaluation • distribution changes across training and inference runs
	Apache Airflow (AWS service) • Data query. Airflow is integrated with OpenMeteo-API for daily data streaming. • Data preprocessing. The preprocessing pipeline is scheduled in Airflow, after data query.
	OpenMeteo is an open-source weather API.It offers free access for non-commercial use. In this project, we are using the apis that's listed from OpenMeteo for Dataset: • Air Quality API • Historical Weather API

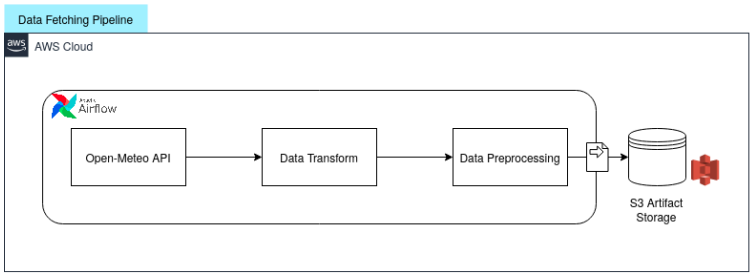
Version Control

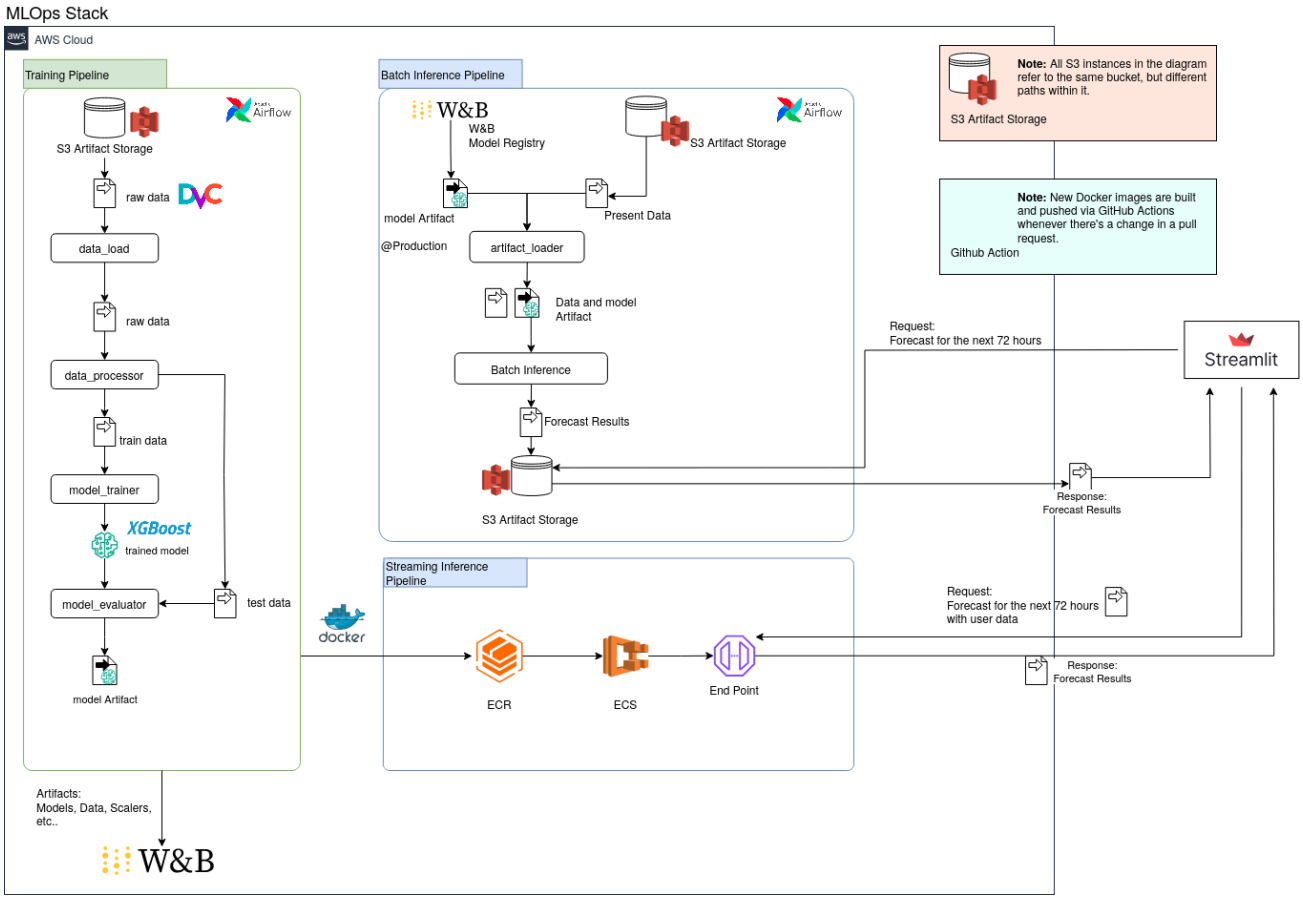
Tool	Description
	Github & Github Action • Hosting and managing source code, tracking changes • collaborating via pull requests • integrating with CI/CD workflows.
	UV package manager • Managing Python virtual environments and package dependencies
	DVC (Data Version Control) • Track changes in datasets and model artifacts

Application & Deployment

Tool	Description
	Docker • Containerized service is used before the deployment of forecasting API.
	Fast API • Create python Forecasting-API. Using models from W&B and Data from Apache Airflow.
	Streamlit • Create a Fast Interactive dashboard for our application.
	Amazon Elastic Container Service • Easily deploy a docker container on a public network.
	Amazon S3 • Cloud storage service for storing preprocessed datasets and model artifacts before deployment.

Overall Architecture





System Architecture Overview

This system is designed to support both real-time and batch forecasting of PM2.5 air quality levels in Bangkok. It integrates machine learning models, data pipelines, and interactive visualization through a modular and reproducible architecture. The core components and their interactions are as follows:

1. Data and Model Storage (Artifacts)

Main Repo and DATA DAG are the sources of code and raw or processed data respectively.

These artifacts (models, processed datasets) are stored in:

- W&B (Weights & Biases): for model versioning, training logs, and tracking production-ready models.
- S3 (AWS S3 Storage): for storing raw data, prediction outputs, and feature-engineered datasets.

2. Batch Prediction Pipeline

- A Batch Prediction DAG (e.g., managed by Apache Airflow) pulls:
    - The latest production model from W&B
    - The required data from S3
- It runs batch inference on historical or incoming data, and stores the results (e.g., as .csv) back to S3.
- This pipeline supports scheduled batch forecasting, useful for daily reporting or analytics.

3. Real-Time Prediction API

- The API Server serves the current production model (fetched from W&B) and provides stream inference capability.
- It accepts incoming data from the frontend or external systems and returns real-time prediction results.
- This API enables the Streamlit app to display live forecasts (e.g., next 1–48 hours of PM2.5).

4. Frontend Dashboard (Streamlit)

- The Streamlit web app acts as the primary interface for end-users.
- It communicates with the API server to retrieve real-time predictions.
- It also fetches historical and batch prediction data from S3 to visualize:
    - Trends
    - KPI summaries
    - Raw tables
    - Time-based analytics

5. User Interaction

- Users interact directly with the Streamlit dashboard.
- They can monitor air quality, view forecast visualizations, explore historical trends, and trigger data refreshes.

Fast-API

All source code for API can be access through this repository
https://github.com/Saranunt/baq-api

Docker Image For our API can be found in this repository docker hub
https://hub.docker.com/repository/docker/saranunt/baq-api-v3

To make the PM2.5 forecasting system easily accessible for downstream applications and users, the Forecasting API was developed using **FastAPI**. This API serves real-time and batch predictions from the production machine learning model–present in **Weight and Bias**, offering a reliable interface for integrating forecasting functionality into broader systems, such as dashboards or alerting mechanisms.

Endpoint Summary

<table>
<tr><th>Endpoint</th><th>Description</th></tr>
<tr><td>POST /predict/onetime</td><td>Single-Step Forecast
Predict pm2.5 value for next one hour.

Output format (JSON)
<pre>{
    "predictions": [
        {"predicted_value": 12.3}
    ]
}</pre>Curl Testing
<pre>curl -X POST http://13.228.168.101:9000/predict/onetime</pre></td></tr>
<tr><td>POST /predict/next</td><td>Multi-Step Forecast
Predict pm2.5 value for next forecast_horizon argument

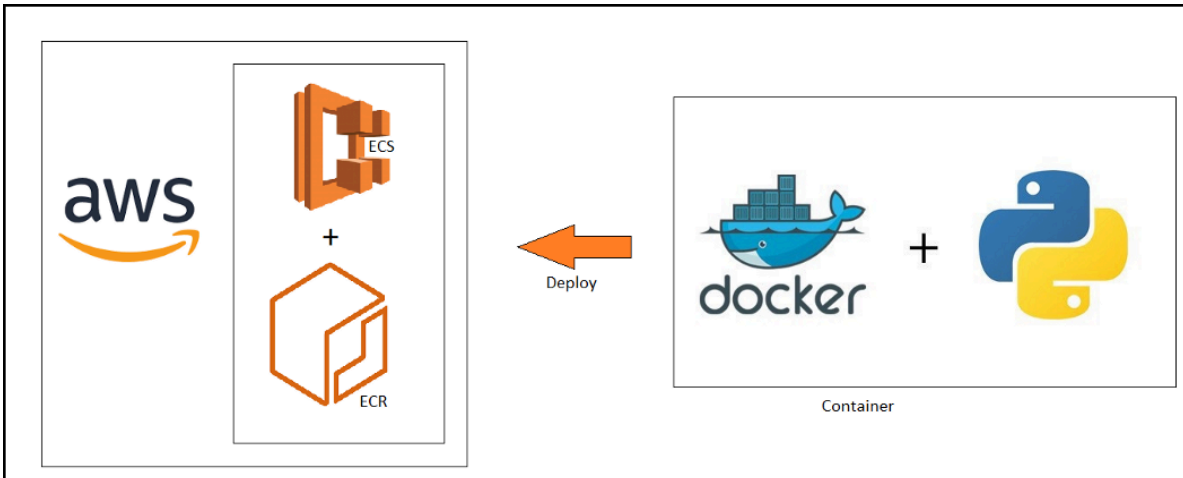
Output format (JSON)
<pre>{
    "predictions": [
        {"predicted_value": 12.3},
        ...
    ]
}</pre>Curl Testing
<pre>curl -X POST http://13.228.168.101:9000/predict/next -H
"Content-Type: application/json" -d '{"forecast_horizon": 5}'</pre></td></tr>
</table>

Endpoint	Description
`POST /predict/cache`	**Multi-Step Forecast with Batch Caching in S3** Predict pm2.5 value for next **forecast_horizon** argument, and save prediction result to **S3** filebase **Output format (csv in S3 Database)** time,predicted_value 2025-05-26 08:00:00,12.43 2025-05-26 09:00:00,12.43 ... **Curl Testing** `curl -X POST http://13.228.168.101:9000/predict/cachet -H "Content-Type: application/json" -d '{"forecast_horizon": 5}'`

Supporting Features

- Model Loading: W&B is directly connected via wandb-API to download production stage models directly from ML-engineer.
- Forecasting Logic: Multi-step forecasting is handled by a custom `multi_step_forecasting()` utility which supports autoregressive predictions.
- S3 Integration: Raw, processed, and prediction output data are all read from and written to Amazon S3, ensuring cloud-based scalability and persistence.

Fast-API Deployment



The PM2.5 forecasting API is containerized using Docker to ensure consistent behavior across environments. A production-ready Docker image is built and pushed to Docker Hub, serving as the source for deployment. The container is then deployed to Amazon ECS (Elastic Container Service), enabling scalable, reliable, and managed hosting of the API. This setup allows for seamless integration with other AWS services, automated updates, and high availability of the prediction service.

Frontend Dashboard (Streamlit)

All source code for Streamlit can be access through this repository
https://github.com/tawayahc/baq-frontend

To make the forecasting system accessible and interactive, we developed a web-based dashboard using Streamlit, an open-source Python framework for creating data applications with minimal frontend code. The dashboard visualizes air quality conditions and PM2.5 forecasts in real time, offering users an intuitive interface to explore predictions, historical patterns, and raw data.

Architecture Overview

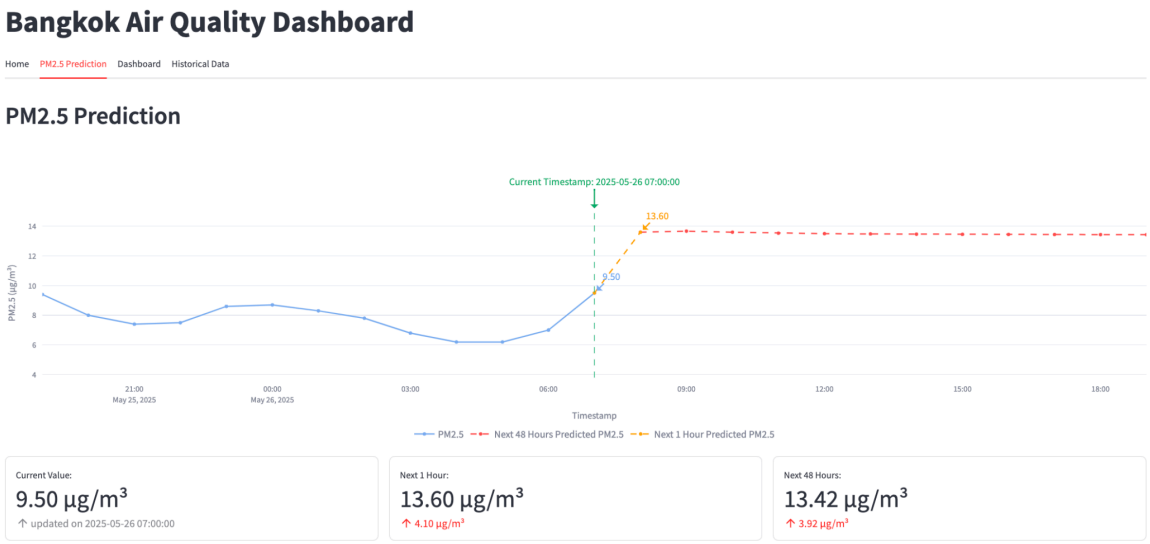
The application integrates with two external sources:

1. Prediction API: A backend service that returns model-generated PM2.5 forecasts. It is queried periodically to obtain the latest predictions for both the next 1 hour and the next 48 hours.
2. AWS S3 Storage: Cleaned raw data and statistical summaries are stored in an S3 bucket. The dashboard retrieves and loads this data to populate historical charts and detailed tables.

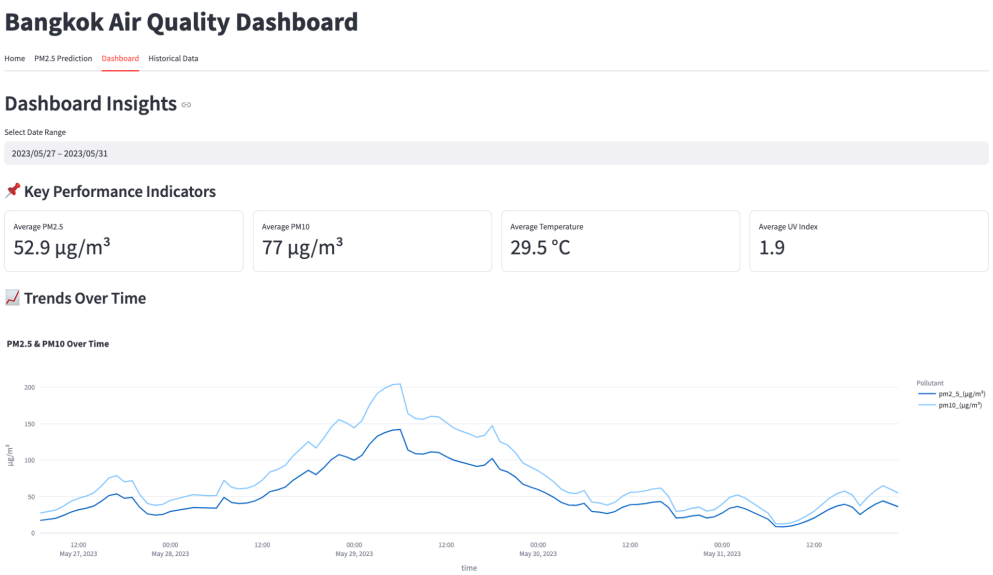
Key Functionalities

The web app is organized into three main sections:

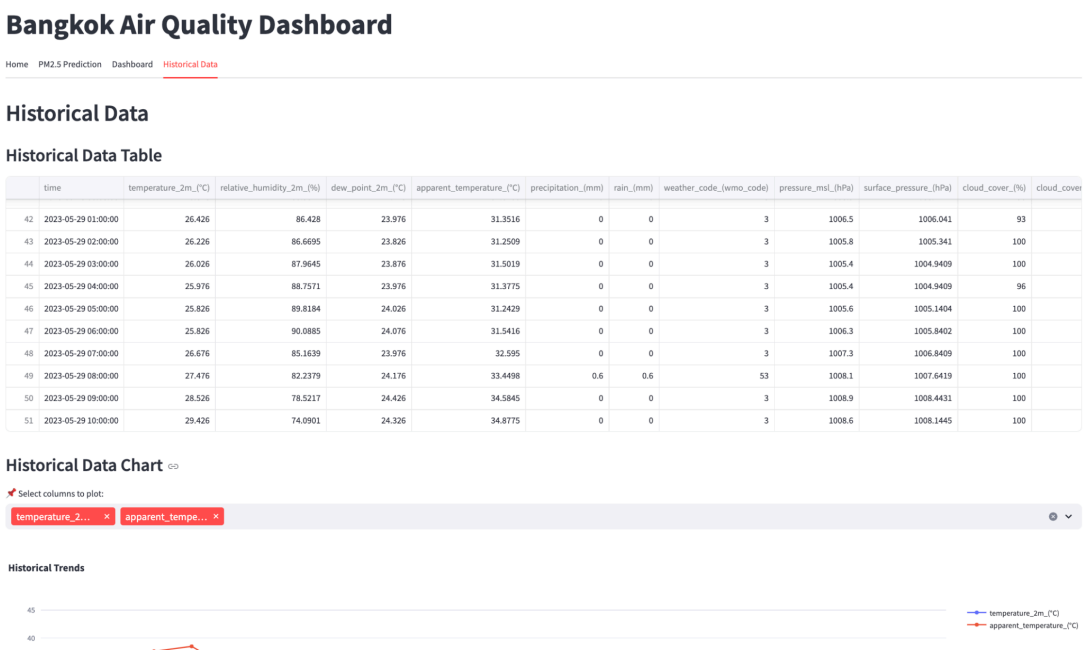
- PM2.5 Prediction: This page displays real-time PM2.5 readings and model predictions for the upcoming 1–48 hours. Forecast data is retrieved from the backend API and visualized alongside recent ground truth measurements. Key metrics (current value, predicted change, forecast horizon) are clearly presented to enhance situational awareness.



- Dashboard Insights: Users can explore trends and patterns across selected date ranges, including key performance indicators (e.g., average PM2.5, PM10, temperature) and time-series visualizations. This section allows for analysis of correlations, distributions, and seasonal fluctuations.



- Historical Data: A searchable data table and interactive plot area allow users to browse raw measurements retrieved from S3. Variables such as temperature, humidity, cloud cover, and pollutant levels can be visualized and analyzed through selectable chart options.



	time	temperature_2m_(°C)	relative_humidity_2m_(%)	dew_point_2m_(°C)	apparent_temperature_(°C)	precipitation_(mm)	rain_(mm)	weather_code_(wmo_code)	pressure_msl_(hPa)	surface_pressure_(hPa)	cloud_cover_(%)	cloud_cover
42	2023-05-29 01:00:00	26.426	86.428	23.976	31.3516	0	0	3	1006.5	1006.041	93	
43	2023-05-29 02:00:00	26.226	86.6695	23.826	31.2509	0	0	3	1005.8	1005.341	100	
44	2023-05-29 03:00:00	26.026	87.9645	23.876	31.5019	0	0	3	1005.4	1004.9409	100	
45	2023-05-29 04:00:00	25.976	88.7571	23.976	31.3775	0	0	3	1005.4	1004.9409	96	
46	2023-05-29 05:00:00	25.826	89.8184	24.026	31.2429	0	0	3	1005.6	1005.1404	100	
47	2023-05-29 06:00:00	25.826	90.0885	24.076	31.5416	0	0	3	1006.3	1005.8402	100	
48	2023-05-29 07:00:00	26.676	85.1639	23.976	32.595	0	0	3	1007.3	1006.8409	100	
49	2023-05-29 08:00:00	27.476	82.2379	24.176	33.4498	0.6	0.6	53	1008.1	1007.6419	100	
50	2023-05-29 09:00:00	28.526	78.5217	24.426	34.5845	0	0	3	1008.9	1008.4431	100	
51	2023-05-29 10:00:00	29.426	74.0901	24.326	34.8775	0	0	3	1008.6	1008.1445	100	

Backend Integration Details

- Prediction Retrieval: The app uses scheduled API calls (e.g., every 60 seconds) to fetch the latest PM2.5 prediction values, which are then rendered dynamically in Streamlit.
- Data Loading from S3: The dashboard uses a custom S3DataLoader class to load CSV files from a designated S3 path, based on the most recent date available. These files include both raw sensor data and engineered features.

Dataset

This project uses environmental datasets focusing on the Bangkok metropolitan area, combining historical weather and air quality data. The data were collected from Open-Meteo, a free and publicly available API service providing high-resolution meteorological and air pollution information. The dataset includes hourly records of temperature, humidity, wind speed, precipitation, and air pollutants such as PM10, ozone (O_3), PM2.5, CO, NO_2, and CH_4. These variables were selected for their known influence on PM2.5 concentration levels.

The task is a regression problem since the objective is to predict future PM2.5 concentrations as continuous numerical values. The PM2.5 levels in the dataset range approximately from 0 to 300 $\mu g/m^3$, covering a broad spectrum of air quality conditions from low pollution to hazardous levels.

The cleaned dataset contains approximately 17,000 hourly samples collected over several months. As the original dataset does not include predefined training or testing splits, the data were divided into three chronologically ordered blocks: 70% for training, 10% for validation, and 20% for testing. No shuffling was applied, in order to maintain the temporal consistency essential for time-series forecasting.

To prepare the data for supervised learning, a sliding window approach was employed for the train-test split. In this method, a fixed-length sequence of past time steps is used as input features, while the subsequent time step serves as the prediction target. This technique is particularly well-suited for time series forecasting, as it preserves the temporal ordering of data and avoids information leakage from future observations into the training set. The dataset was split chronologically into training and testing sets to reflect real-world forecasting conditions.

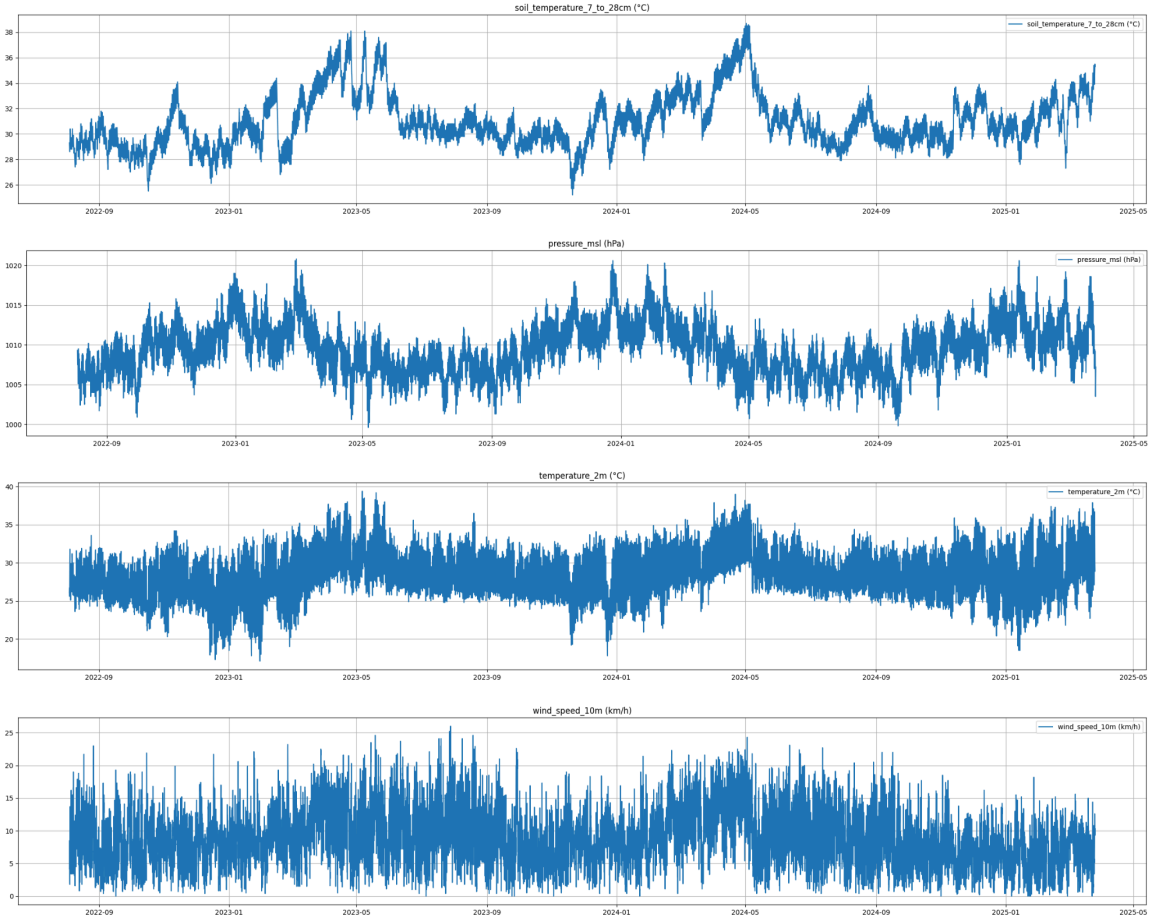
Data Analysis

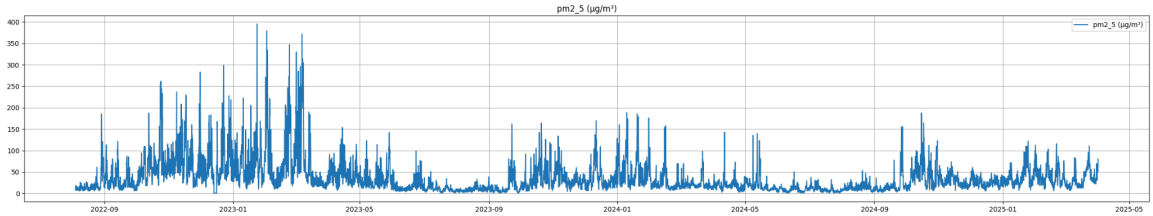
- Priminarly EDA
  - Statistics

	temperature_2m (°C)	relative_humidity_2m (%)	dew_point_2m (°C)	...	uv_index ()	dust (μg/m³)	aerosol_optical_depth ()
count	23312.000000	23312.000000	23312.000000	...	23312.000000	23312.000000	23312.000000
mean	28.264310	73.092527	22.485540	...	1.897027	0.698653	0.407597
std	3.125795	16.615385	3.493947	...	2.975091	1.806950	0.307136
min	17.100000	21.000000	7.400000	...	0.000000	0.000000	0.030000
10%	24.600000	48.000000	16.700000	...	0.000000	0.000000	0.140000
25%	26.100000	62.000000	21.100000	...	0.000000	0.000000	0.200000
50%	28.000000	76.000000	23.900000	...	0.000000	0.000000	0.310000
75%	30.400000	87.000000	24.800000	...	3.050000	1.000000	0.520000
90%	32.400000	92.000000	25.500000	...	7.150000	2.000000	0.800000
99%	35.900000	97.000000	26.800000	...	10.650000	10.000000	1.510000
max	39.400000	100.000000	28.000000	...	13.150000	20.000000	3.290000

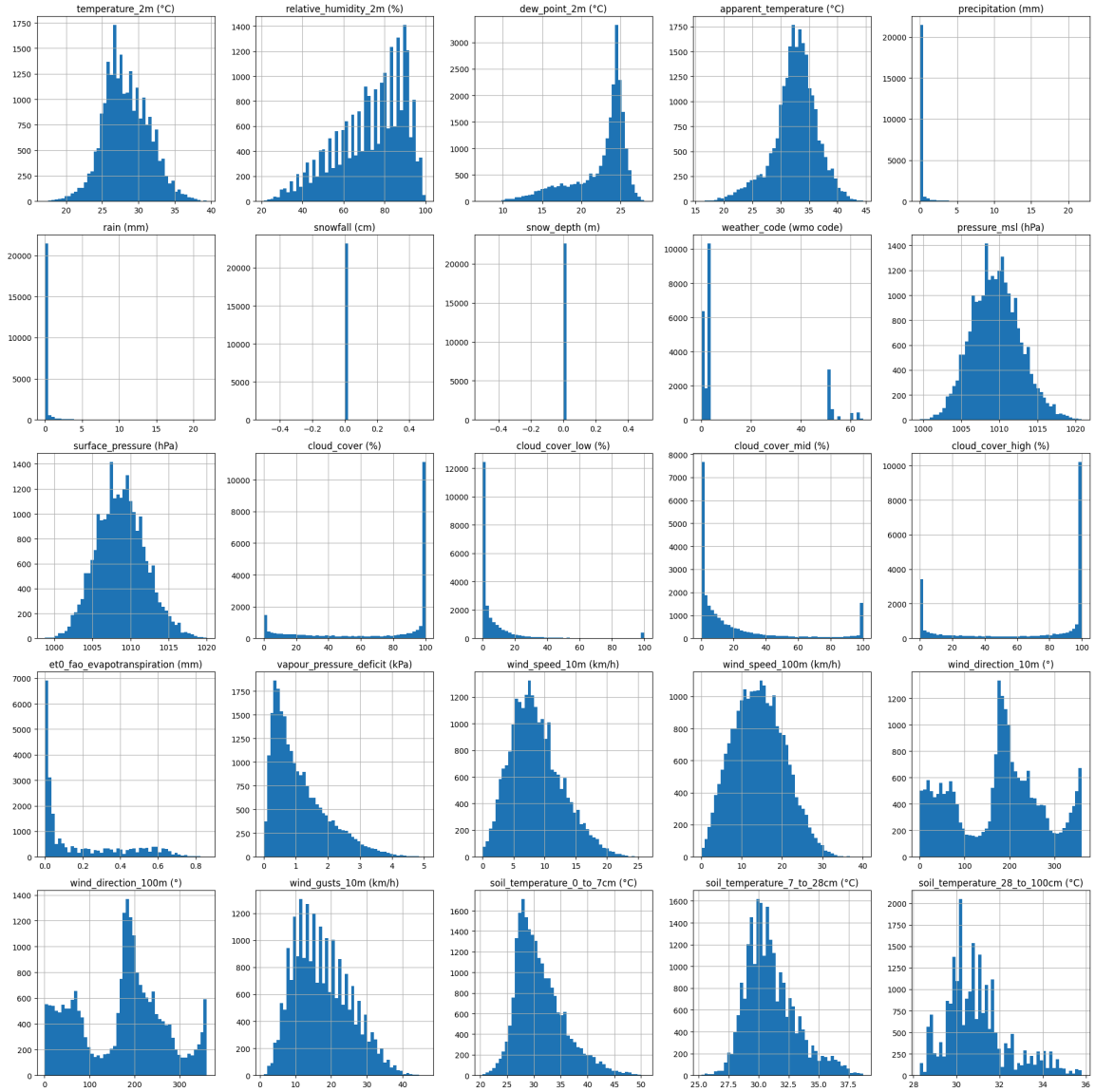
To better understand the characteristics and distribution of our input features, we performed exploratory data analysis (EDA) by calculating summary statistics for each variable in the dataset. This includes metrics such as count, mean, standard deviation, min, max, and various percentiles (10%, 25%, 50%, 75%, 90%, 99%).

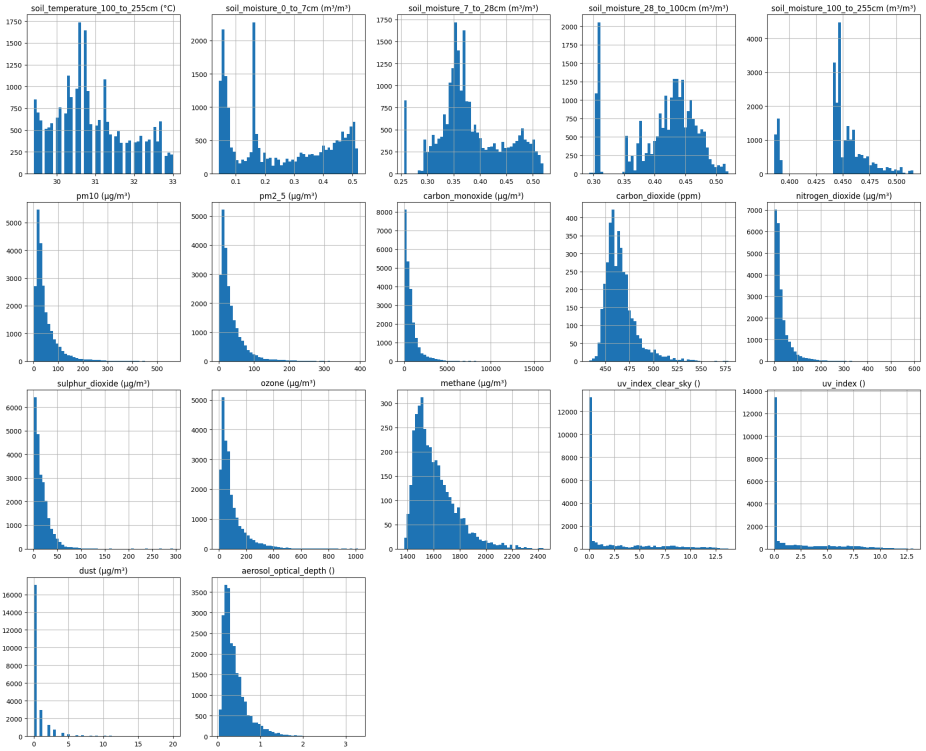
- Example of features





- Example of distributions





- Correlation with PM2.5

From the preliminary EDA, we observed that most of the features are approximately normally distributed, with only a few showing slight skewness. This suggests that the dataset is relatively clean and of reasonably good quality at this initial stage. The meteorological variables like temperature, pressure, and wind show typical bell-curve distributions, while precipitation and air quality parameters exhibit expected right-skewed patterns with occasional spikes. Cloud cover variables display bimodal distributions between clear and overcast conditions.

Also, From the correlation heatmap, we observe several key insights regarding features that are correlated with PM2.5 levels:

Strong Positive Correlations:

- pm10 (μg/m³) shows an extremely high positive correlation (0.99) with pm2_5, indicating that both pollutants tend to vary together and may share similar sources or environmental influences.
- Other air pollutants such as sulphur_dioxide (0.80), carbon_monoxide (0.76), and nitrogen_dioxide (0.65) also show significant positive correlations. This suggests that PM2.5 levels are closely linked to other gaseous pollutants, reflecting potential sources like traffic emissions or industrial activity.
- aerosol_optical_depth (0.59) and ozone (0.54) also contribute positively, indicating their atmospheric interplay with particulate matter.

Low to Moderate Positive Correlations:

- Variables like methane, carbon_dioxide, and pressure_msl show weak positive correlations (around 0.2–0.3), which may have limited direct influence on PM2.5 levels but could still reflect background environmental factors.

Negative Correlations:

- Several meteorological features such as wind_speed, cloud_cover, relative_humidity_2m, and dew_point_2m show moderate negative correlations, ranging from about -0.2 to -0.3. This implies that increases in wind or humidity might help disperse or dilute PM2.5 concentrations.
- The strongest negative correlation is with oil_temperature_100_to_255cm (°C) at -0.45, which might be more indirectly related and could indicate seasonal or deeper environmental trends.

Near-zero Correlations:

- Features such as uv_index, temperature_2m, and some soil moisture/temperature levels show very low or near-zero correlation, suggesting little to no direct relationship with PM2.5 in this dataset.

Data Preprocessing

Before model training, extensive data preprocessing was performed to ensure temporal consistency, handle missing values, and normalize features. These steps were essential to prepare the dataset for both traditional machine learning models and sequence-based deep learning models like LSTM. The preprocessing pipeline included the following components:

1. Timestamp Parsing and Indexing

The original dataset included a time column in string format. This column was converted to datetime and set as the DataFrame index to facilitate time-based operations such as resampling, lag generation, and rolling statistics. The data was sorted chronologically to maintain its temporal order.

2. Column Cleaning and Reduction

To standardize naming conventions, column names were reformatted by replacing spaces and parentheses with underscores. Irrelevant or incomplete columns (e.g., carbon_dioxide_(ppm), methane_(μg/m³), snowfall_(cm), and snow_depth_(m)) were removed from the dataset to reduce noise and dimensionality.

3. Resampling and Interpolation

The dataset was resampled to an hourly frequency to ensure consistent time intervals. Linear interpolation was then applied to fill short gaps in the time series, ensuring continuity and completeness for model input.

4. Seasonal Median Imputation

To handle residual missing values after interpolation—particularly in pollutant and meteorological measurements—a custom imputation method was applied. For each missing value, a median was computed from historical observations that occurred at the same hour, day, and month, but in different years. This approach preserves seasonal and diurnal trends while minimizing imputation bias.

5. Data Splitting

We divided the cleaned dataset into three non-overlapping subsets: training, validation, and test sets. The split was performed based on fixed chronological order rather than random sampling, which is essential for time series forecasting tasks where the order of observations carries important sequential information.

We allocated:

- 70% of the data for training, used to fit the model parameters
- 10% for validation, used for hyperparameter tuning and early stopping
- 20% for testing, used for final model evaluation on unseen future data

The splitting process was performed by indexing the dataset along the time axis:

- The training set contains the earliest observations
- The validation set follows immediately after the training set
- The test set includes the most recent data, simulating real-world forecasting scenarios where future values are predicted from past trends

No shuffling was applied during the split to preserve the integrity of the temporal relationships. Each subset was copied explicitly to prevent unintended side effects during model development. This approach ensures that the models are trained only on past information and evaluated on future observations, mimicking the actual deployment conditions for air quality forecasting.

6. Feature Scaling

All input features were scaled to a range of [0, 1] using Min-Max normalization. This is especially important for neural networks, which are sensitive to the magnitude of input features. Separate scalers were used for the feature set and the PM2.5 target column. Scaling was fit on the training set and applied consistently to the validation and test sets to avoid data leakage.

7. Sequence Preparation for LSTM

For the LSTM model, data was converted into fixed-length input sequences. Specifically, a sliding window approach was used to generate sequences of 24 consecutive hourly time steps, with each sequence predicting the PM2.5 value for the hour immediately following the sequence. This format enables the LSTM model to learn temporal dependencies across multiple input features over time.

Feature Engineering

To enhance the model's ability to capture both temporal patterns and pollution dynamics, we performed extensive feature engineering. The goal was to enrich the dataset with information reflecting time-based behaviors, short-term trends, and health-related thresholds. The resulting features can be grouped into six main categories.

1. Air Quality Index Tier

PM2.5 concentration values were transformed into categorical tiers based on widely recognized air quality standards. These tiers represent different levels of health risk:

- Tier 0 (Good): PM2.5 ≤ 12.0 μg/m³
- Tier 1 (Moderate): 12.1–35.4 μg/m³
- Tier 2 (Unhealthy for Sensitive Groups): 35.5–55.4 μg/m³
- Tier 3 (Unhealthy): 55.5–150.4 μg/m³
- Tier 4 (Very Unhealthy): 150.5–250.4 μg/m³
- Tier 5 (Hazardous): PM2.5 > 250.4 μg/m³

This categorization simplifies the learning task by allowing the model to associate pollutant levels with discrete health impact categories, supporting both regression and classification objectives.

2. Raw Timestamp Features

From each timestamp, we extracted the hour of day, day of week, and month. These features provide the model with information about daily cycles (e.g., traffic peaks), weekly patterns (e.g., weekday vs. weekend behavior), and seasonal variations (e.g., monsoon effects).

3. Temporal Flags

We introduced two binary indicators to highlight specific time-related contexts:

- is_weekend: Identifies Saturdays and Sundays, which often exhibit different pollution patterns due to reduced traffic and industrial activity.
- is_night: Flags nighttime hours (before 6:00 or after 20:00), when lower atmospheric mixing may cause pollutant accumulation.

4. Cyclical Encoding of Hour

Since the hour of day is cyclical in nature (e.g., 23:00 is close to 00:00), we applied sine and cosine transformations to encode it. This method ensures continuity in the representation and enables the model to learn smooth diurnal patterns without discontinuities.

5. Lagged Pollution Features

To capture short-term temporal dependencies, we created lag features for PM2.5, PM10, ozone, and coarse dust at various intervals (1, 3, 6, 12, and 24 hours). These lag values help the model detect recent trends, sudden spikes, or decays in pollutant concentrations.

6. Rolling Averages

Rolling means were calculated over 3-, 6-, and 12-hour windows for selected pollutants (PM2.5, PM10, and ozone). These smoothed features reduce noise in the data and highlight sustained pollution trends, which are valuable for forecasting over extended periods.

Collectively, these engineered features provide a robust temporal and contextual foundation for the forecasting models. They capture daily, weekly, and seasonal cycles, distinguish between normal and atypical conditions (such as weekends or night time), and incorporate recent pollutant history. This comprehensive feature set enhances the models' ability to learn complex temporal dependencies in PM2.5 data, ultimately improving both short-term (single-step) and long-term (multi-step) prediction performance.

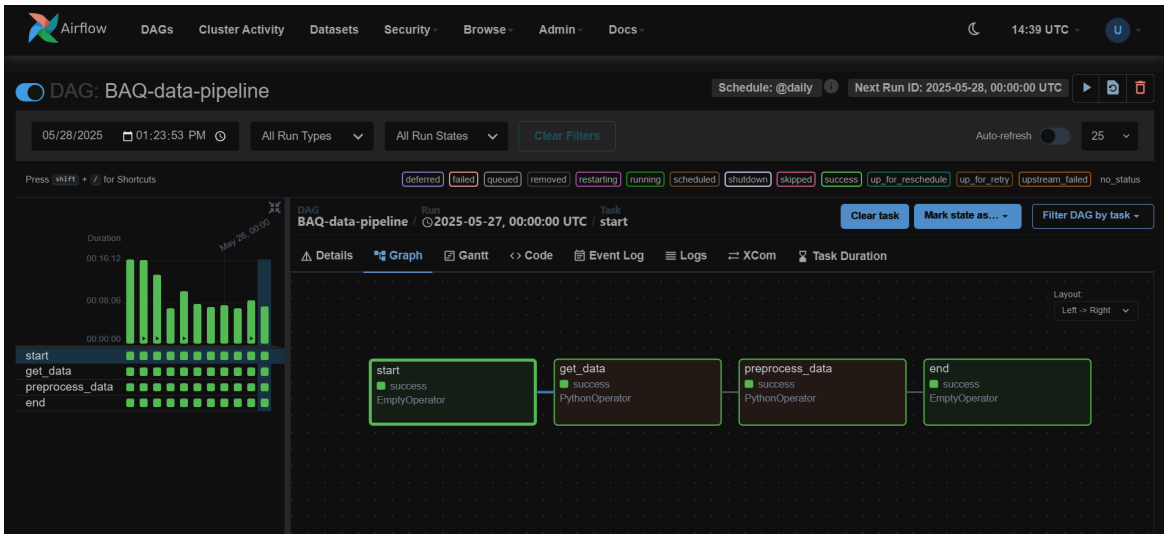
Data Pipeline in Apache Airflow

All source code for Apache-Airflow can be access through this repository
https://github.com/Saranunt/baq-airflow

To support the continuous forecasting and evaluation of PM2.5 levels, this project integrates an automated data pipeline using **Apache Airflow**, deployed on **AWS**, with **S3** as a filebase storage. The pipeline ensures reliable and up-to-date ingestion and preprocessing of weather and pollution data retrieved daily from Open-Meteo's APIs.

The Airflow DAG (Directed Acyclic Graph) is scheduled to run daily and performs the following key tasks:

1. **Data Retrieval**: The pipeline begins by querying Open-Meteo's Air Quality and Historical Weather APIs to daily collect the most recent measurements.
2. **Preprocessing**: Raw data will be passing through data pipeline & feature engineering
3. **Data Storage**: Both raw and preprocessed datasets are automatically uploaded to **Amazon S3**. This centralized storage supports accessibility for downstream tasks and serves as the source of truth for model inputs.



While data versioning is not directly handled within the Airflow DAG, it is a critical part of the ML engineering workflow. Machine learning engineers manage dataset versions manually or through dedicated tools when loading data from S3 for model training and evaluation. This ensures that experiments remain reproducible and traceable over time.

By automating these processes, Apache Airflow enables scalable, maintainable, and production-ready data operations that are integral to the project's MLOps strategy.

Machine Learning Model Development / Training

1. Model Selection

To address the task of forecasting PM2.5 concentrations in Bangkok, we selected three distinct machine learning models: Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Long Short-Term Memory (LSTM). This combination was chosen to explore both traditional ensemble methods and deep learning techniques, allowing for a comparative evaluation across different modeling paradigms.

- Random Forest is a robust ensemble of decision trees that is well-suited for handling structured tabular data. It is capable of capturing complex nonlinear relationships between input features and the target variable while being relatively resistant to overfitting. Its interpretability and low sensitivity to feature scaling make it a strong baseline for regression tasks.
- XGBoost extends the decision tree ensemble approach through gradient boosting, where trees are built sequentially to minimize a loss function. Known for its superior predictive accuracy and speed, XGBoost handles feature interactions and temporal patterns effectively. It is particularly advantageous in time series problems when engineered features (such as lags and rolling statistics) are used.
- LSTM is a type of recurrent neural network specifically designed to model sequential data. Unlike tree-based methods, LSTM learns temporal dependencies directly from raw time-ordered inputs. This makes it well-suited for time series forecasting, especially when long-range dependencies are present. However, LSTMs generally require more data and careful tuning to perform optimally.

By selecting these three models, we aim to balance interpretability, accuracy, and the ability to learn time-dependent patterns. The comparative analysis provides insights into the strengths and limitations of each approach in the context of short-term air quality prediction.

2. Training Procedure

Each model was trained using procedures tailored to its architecture and strengths. While tree-based models focused on ensemble learning with structured features, the LSTM model relied on sequential inputs and neural network optimization. The training strategies for each model are summarized as follows:

2.1. Random Forest

The Random Forest model was trained as a baseline ensemble method due to its robustness and ease of use. It builds multiple decision trees using bootstrapped samples of the data and averages their predictions to reduce variance. Key training characteristics include:

- Number of estimators: 100
- Max depth: 10

This model does not require feature scaling and is relatively insensitive to hyperparameter tuning, making it a strong and interpretable baseline.

2.2. XGBoost

XGBoost was used as a high-performance gradient boosting method. It trains decision trees sequentially, with each new tree correcting the residuals of the previous ones. Training was configured as follows:

- Number of estimators: 100
- Max depth: 10

XGBoost requires careful hyperparameter tuning to balance learning speed and generalization, and it benefits significantly from engineered lag and time-based features.

2.3. LSTM

The Long Short-Term Memory (LSTM) network was trained to model sequential dependencies in the data. It uses a sliding window of 24 hours of input to predict the PM2.5 value for the next hour. Training was conducted as follows:

- Input shape: sequences of 24 time steps with 37 features each
- Network architecture:
    - Two LSTM layers with 128 and 64 units respectively
    - Dropout layer (rate = 0.2) after every LSTM layer
    - Fully connected (Dense 1 unit) output layer
- Optimizer: Adam
- Loss function: Mean Squared Error
- Batch size: 32
- Epochs: Up to 200 with early stopping (patience = 10)
- Validation: Loss monitored on the validation set to prevent overfitting

The LSTM model required more training time and hyperparameter sensitivity but offered the ability to learn complex temporal patterns directly from raw sequences.

3. Evaluation

Model performance was evaluated on both single-step and multi-step forecasting tasks using a comprehensive set of regression and custom metrics. The primary target was hourly PM2.5 concentration, and the goal was to assess how accurately each model could predict future values under different time horizons.

We used the following evaluation metrics:

- Accuracy: Defined as the percentage of predictions that fall within an acceptable deviation from the true PM2.5 value
- Mean Absolute Error (MAE): Measures the average magnitude of errors in the predictions

- Mean Absolute Percentage Error (MAPE): Reflects the average percentage deviation and is scale-independent
- Mean Squared Error (MSE) and Root Mean Squared Error (RMSE): Penalize larger errors more heavily
- Coefficient of Determination (R^2): Indicates the proportion of variance in PM2.5 that is explained by the model

The results are summarized below:

Single-Step Forecasting

In the single-step setup, where the model predicts the PM2.5 value for the next hour, XGBoost achieved the best performance with an accuracy of 93.45% and the lowest MAPE of 6.55%, followed closely by Random Forest with 93.09% accuracy and 6.91% MAPE. The LSTM model, while achieving a relatively high R^2 of 0.898, underperformed in comparison with an accuracy of 79.99% and MAPE of 20.00%, indicating less consistency in short-term predictions.

Multi-Step Forecasting

In the multi-step forecasting task, where the model predicts several future time steps, Random Forest significantly outperformed other models with an accuracy of 97.15% and an exceptionally low MAPE of 2.85%, demonstrating its strength in stable, extended predictions. XGBoost also maintained strong performance, achieving 91.67% accuracy and 8.33% MAPE. Interestingly, LSTM showed considerable improvement in multi-step accuracy (91.10%) compared to its single-step results, suggesting better learning of long-term temporal patterns despite higher training complexity.

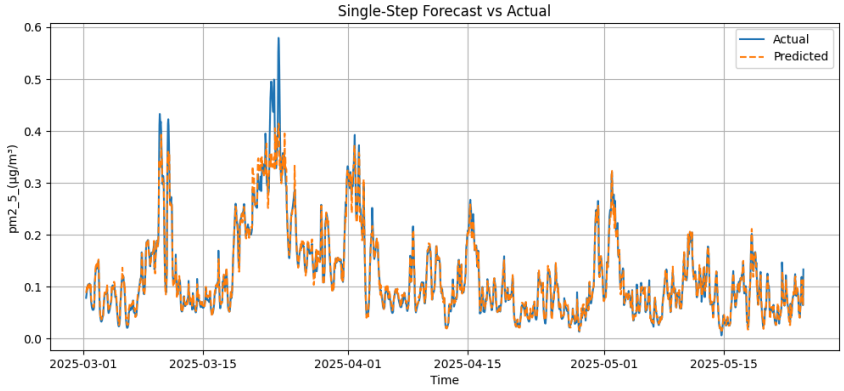
Model	Forecast Type	Accuracy	MAE	MAPE	MSE	RMSE	R^2
XGBoost	Single-step	0.9345	0.00745	0.0655	0.00027	0.01648	0.9640
	Multi-step	0.9167	0.00609	0.0833	0.00006	0.00744	0.8620
Random Forest	Single-step	0.9309	0.00813	0.0691	0.00030	0.01740	0.9598
	Multi-step	0.9715	0.00218	0.0285	0.00001	0.00283	0.9800
LSTM	Single-step	0.8000	0.02040	0.2000	0.00077	0.02780	0.8983
	Multi-step	0.9110	0.00666	0.0890	0.00006	0.00739	0.8639

Summary

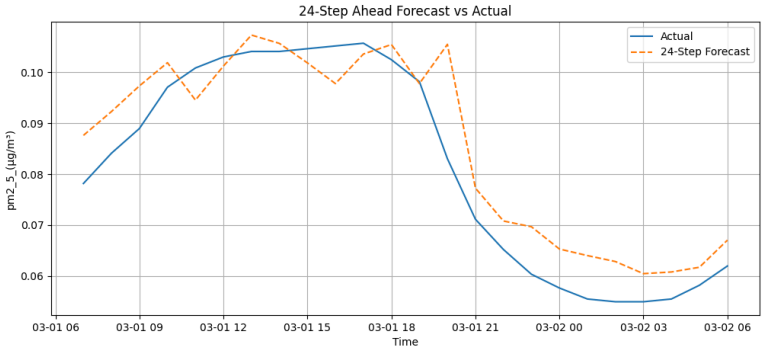
- Random Forest emerged as the most consistent and accurate model for both single-step and multi-step forecasting, especially excelling in extended horizon predictions.

- XGBoost performed best in the single-step task, combining high accuracy with competitive error metrics.
- LSTM, despite its deep learning capacity, required more tuning and showed lower performance in the single-step task, though it improved markedly in multi-step scenarios.

These findings suggest that tree-based ensemble models, particularly Random Forest and XGBoost, are highly effective for short-term air quality forecasting when combined with carefully engineered features. While LSTM may capture temporal dependencies, it may need further optimization or larger datasets to outperform traditional models in this domain.



Example Evaluation Plot of XGBoost Model for single-step forecast

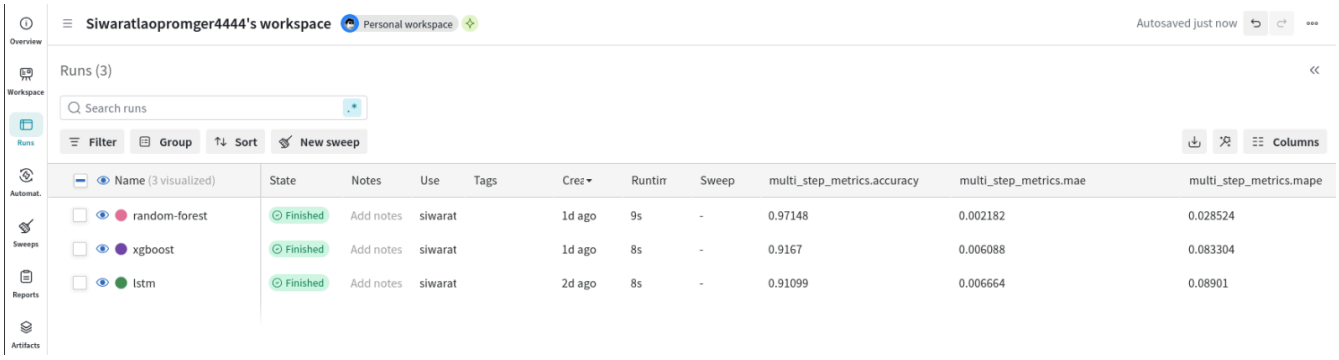


Example Evaluation Plot of XGBoost Model for multi-step forecast

Machine Learning Model Monitoring

In this project, we utilize Weights & Biases (W&B) for comprehensive machine learning model monitoring to ensure performance and reliability throughout the model lifecycle.

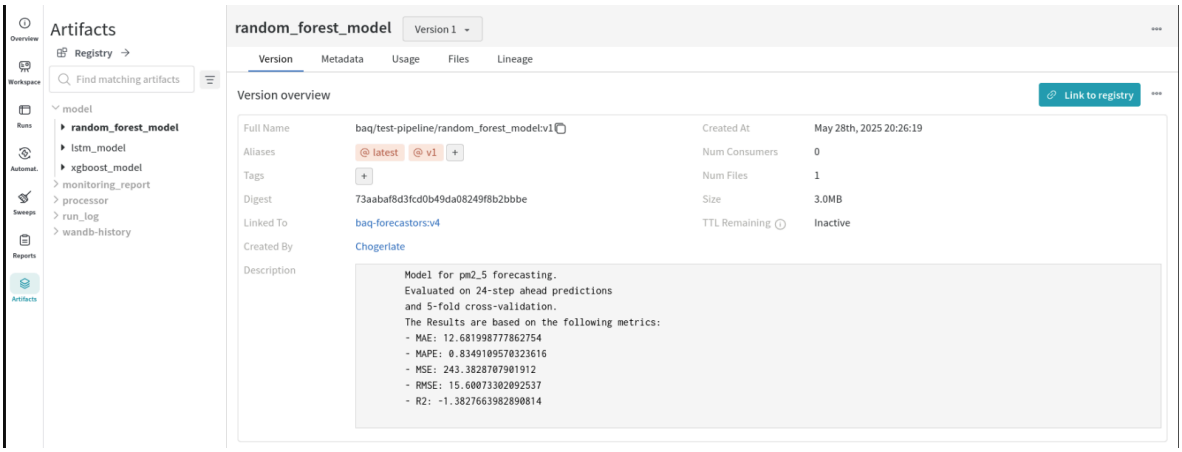
Model Training & Experiment Tracking



Experiment Runs Management

- Run Tracking: We maintain detailed records of all training runs with comprehensive metadata
- Model Variants: Multiple model configurations are tested including:
    - random-forest: Random Forest classifier with optimized hyperparameters
    - xgboost: XGBoost model for enhanced performance comparison
    - lstm: Long Short-Term Memory network for sequential pattern recognition

Performance Metrics Logging



W&B automatically logs and tracks key evaluation metrics during model training:

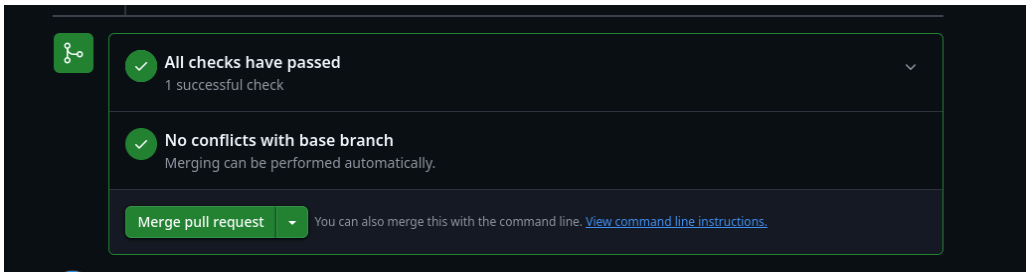
- Accuracy: Defined as the percentage of predictions that fall within an acceptable deviation from the true PM2.5 value
- Mean Absolute Error (MAE): Measures the average magnitude of errors in the predictions
- Mean Absolute Percentage Error (MAPE): Reflects the average percentage deviation and is scale-independent

- Mean Squared Error (MSE) and Root Mean Squared Error (RMSE): Penalize larger errors more heavily
- Coefficient of Determination (R^2): Indicates the proportion of variance in PM2.5 that is explained by the model

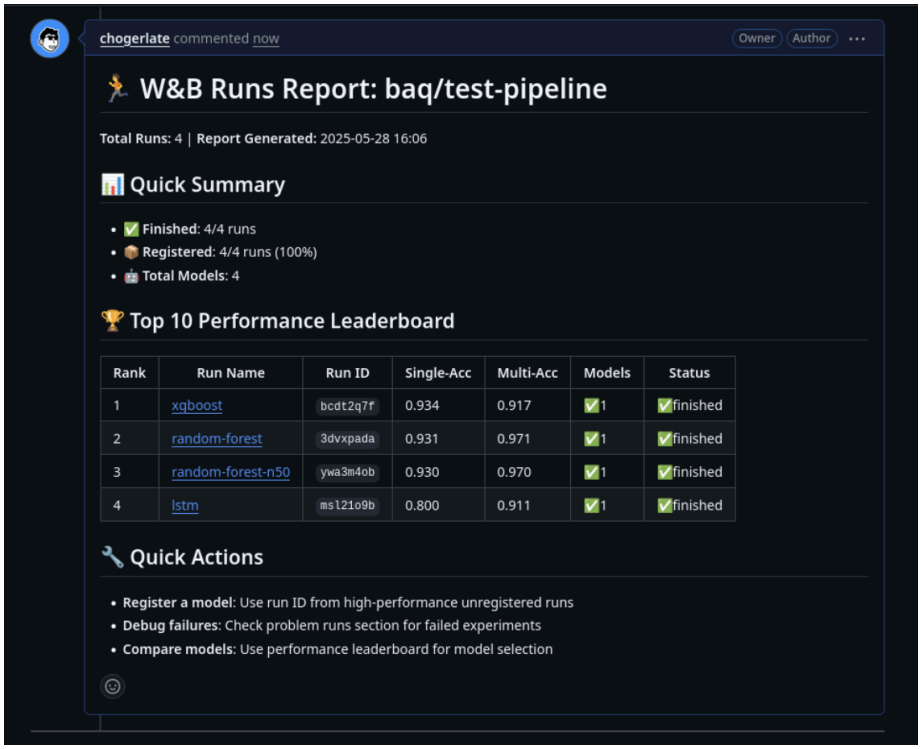
CI/CD Integration with GitHub Actions and Weights & Biases

The project integrates several custom GitHub Actions to streamline experiment tracking, model registration, and deployment workflows. These actions interact directly with the Weights & Biases (WandB) platform and are triggered through pull request comments on GitHub. The following functionalities are supported:

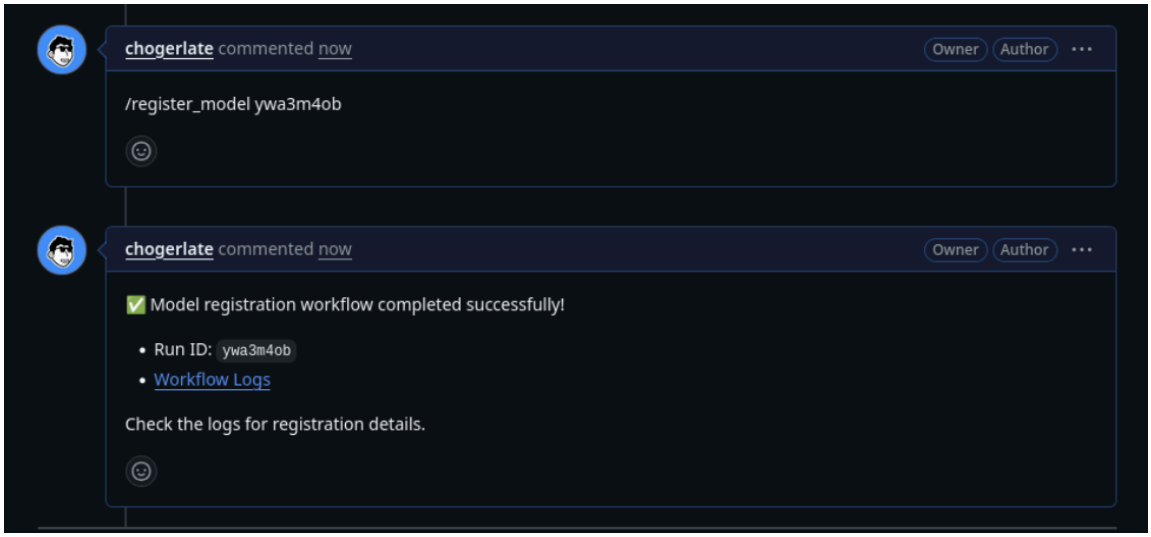
- **Dependency Checking**
  Automatically checks project dependencies (e.g., from requirements.txt) during pull requests to ensure all packages are valid and installable.



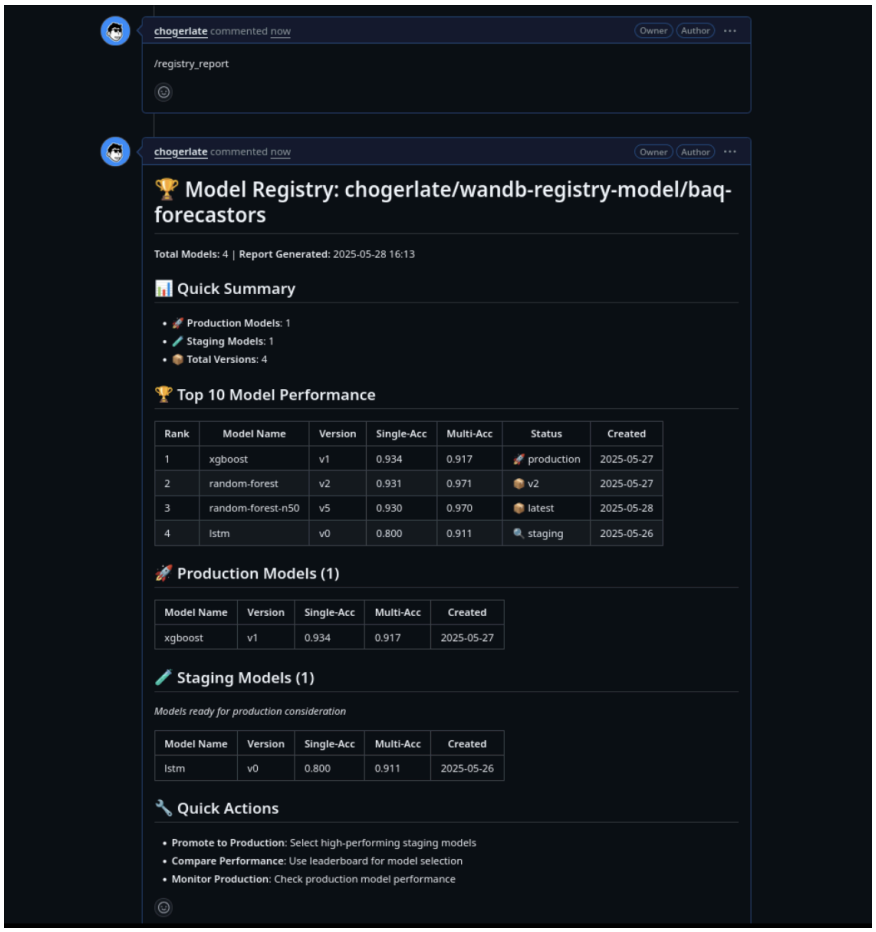
- **/runs_report Command**
  Fetches and displays a summary of recent WandB experiment runs, including key metrics and hyperparameters, as a comment in the pull request.tates easy comparison and selection of model candidates during code review.



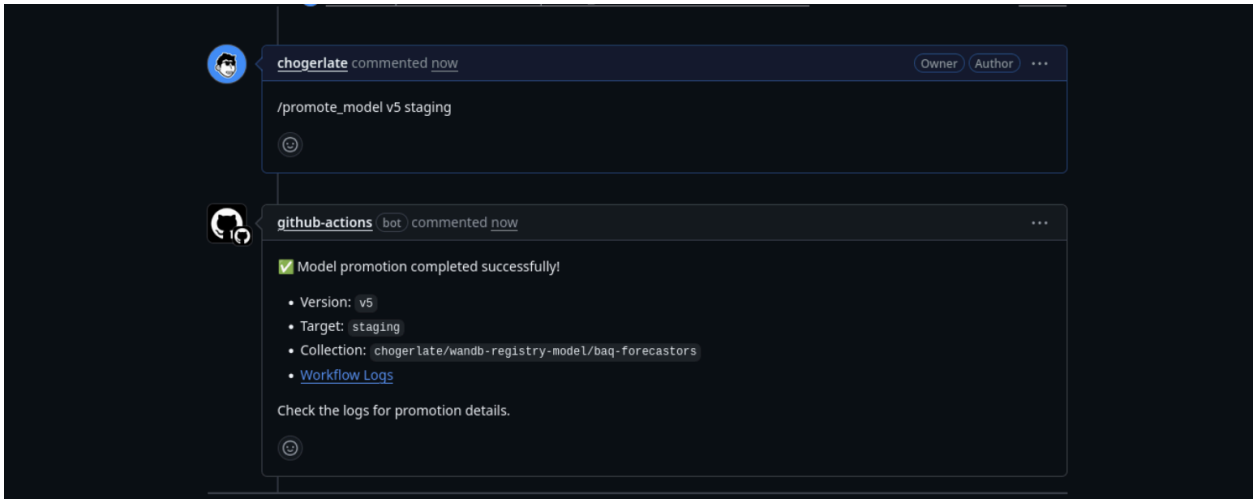
- **/register_model {run_id} Command**
  Registers a trained model from the specified WandB run_id into the WandB model registry and confirms the registration in the pull request.

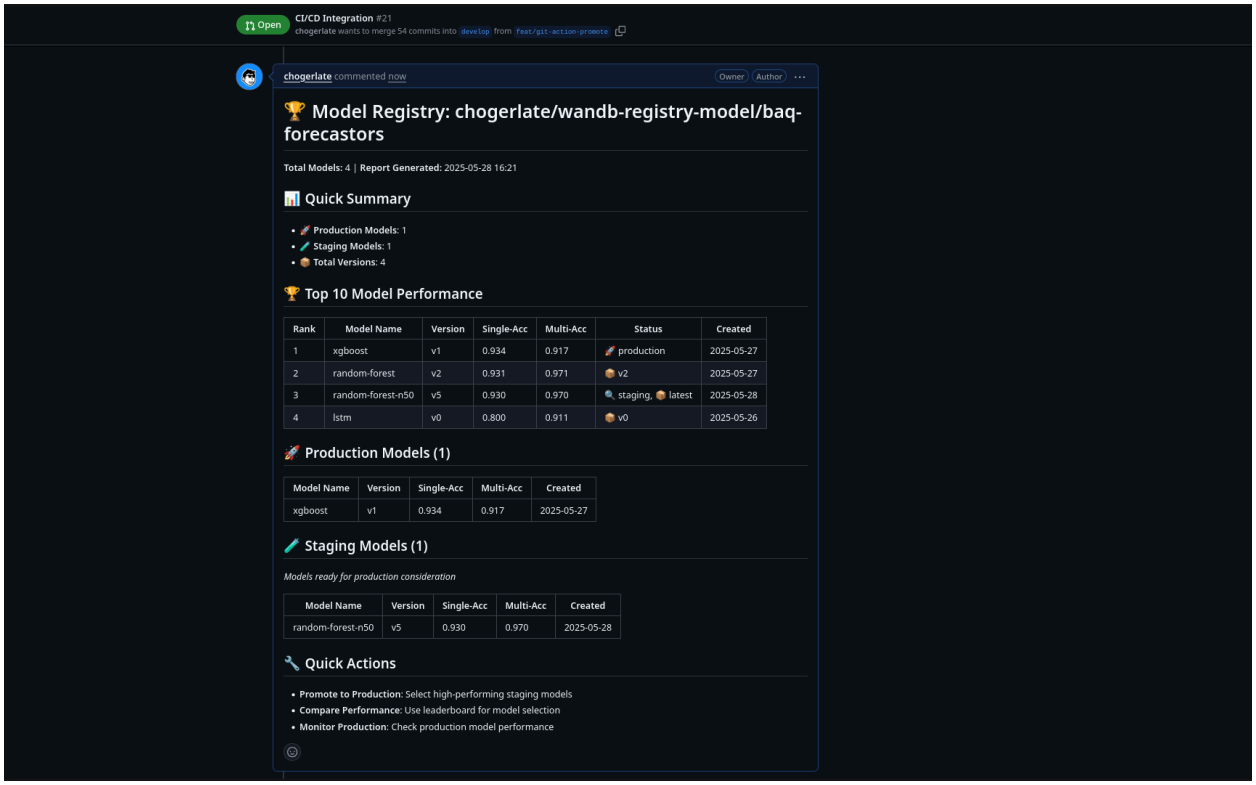


- **/registry_report Command**
  Lists all registered models in the WandB registry, showing names, versions, and aliases in the pull request for easy reference.



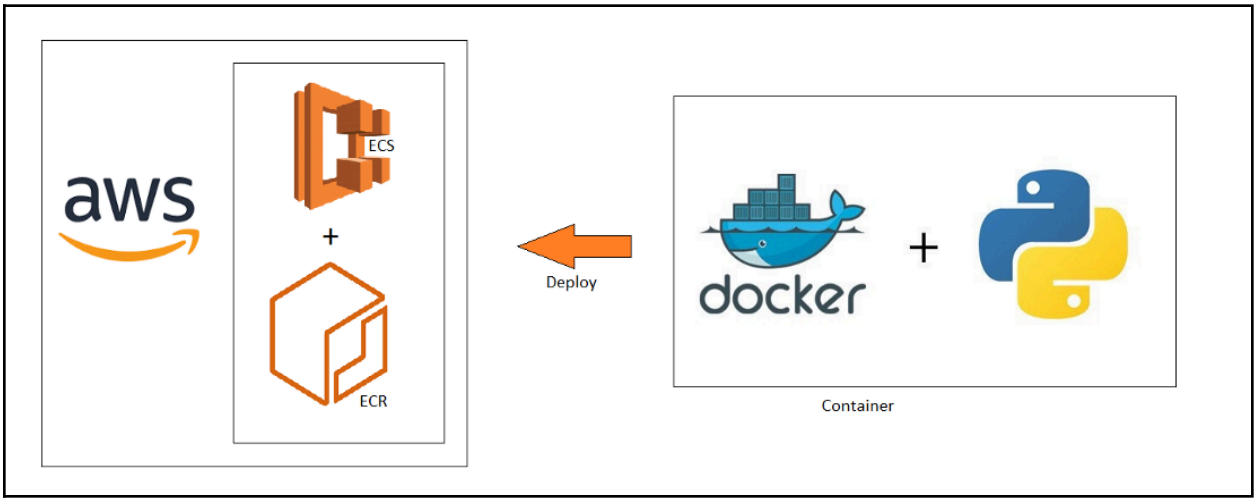
- **/promote_model {version} {target_alias} Command**
  Promotes a registered model version by assigning it a new alias (e.g., production), and confirms the update in the pull request.





Deployment

Fast-Api Deployment



The PM2.5 forecasting API is containerized using Docker to ensure consistent behavior across environments. A production-ready Docker image is built and pushed to Docker Hub, serving as the source for deployment. The container is then deployed to Amazon ECS (Elastic Container Service), enabling scalable, reliable, and managed hosting of the API. This setup allows for seamless integration with other AWS services, automated updates, and high availability of the prediction service.

Streamlit Deployment

To make the air quality forecasting dashboard accessible to end-users and stakeholders, the application was deployed using Streamlit Cloud, a fully-managed hosting service specifically designed for Streamlit apps. This platform allows for rapid and seamless deployment directly from a GitHub repository, eliminating the need for complex infrastructure setup.

Deployment Process

The deployment process consists of the following steps:

1. Codebase Integration: The entire project—including the Streamlit app, data loader classes, API client, and utility modules—was organized into a GitHub repository.
2. Streamlit Cloud Configuration:
    a. A requirements.txt file was provided to specify all necessary Python dependencies.
    b. The main entry point (app/main.py) was selected via the Streamlit Cloud interface.
    c. Environment variables (e.g., API keys, AWS credentials) were configured securely through the platform's settings interface.

3. Automatic Updates: Streamlit Cloud automatically redeploys the app whenever changes are pushed to the repository's main branch, ensuring that users always access the latest version.

Reference

- MLOps-Basics. (n.d.). GitHub repository. Retrieved from graviraja/MLOps-Basics
- machine-learning-apps/actions-ml-cicd. (n.d.). GitHub repository. Retrieved from machine-learning-apps/actions-ml-cicd: A Collection of GitHub Actions That Facilitate MLOps
- Open-Meteo. (n.d.). Retrieved from Open-Meteo.com

Appendix

GitHub Repositories

Main Repository
https://github.com/chogerlate/baq

DAG Airflow Repository
https://github.com/Saranunt/baq-airflow

FastAPI Repository
https://github.com/Saranunt/baq-api

Streamlit Repository
https://github.com/tawayahc/baq-frontend

Model Experiment Repository
https://github.com/tawayahc/baq-experiment

Docker Image

Docker Image For API
https://hub.docker.com/repository/docker/saranunt/baq-api-v3