Enhanced Real-Time Air Quality Monitoring System

Interim Report

Team AAA

Pongsakorn Supakpanichkul (psupkpa), Samuel Lin (szl), Vitchakorn Sayanwisuttikam (vsayanwi), Wan-Ting Tsai (tinat), Luis Carlos Solis Ochoa (lsolisoc)

Introduction

This project presents an end-to-end, production-ready air quality monitoring system designed to process, predict, and monitor pollution levels using real-time data streams and machine learning. The system is built on a robust MLOps pipeline, integrating modern technologies such as Kafka, Docker, Kubernetes, MLflow, and Evidently to ensure scalability, consistency, and transparency across the model lifecycle.

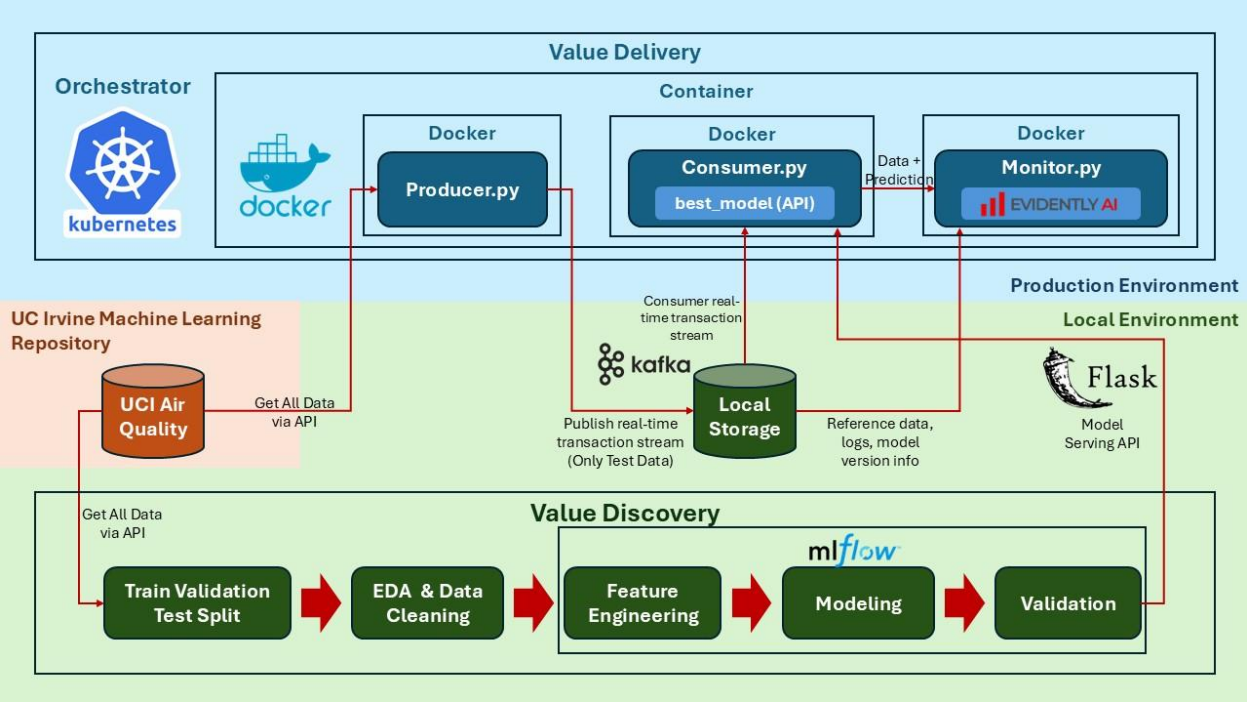
The foundation of the system begins with the UCI Air Quality dataset, which is programmatically accessed using an API. The dataset is used to develop and validate machine learning models in a local development environment, where features are engineered, models are trained and tracked using MLflow, and the final model is served via a Flask-based API. Once validated, the system is containerized and deployed to a production environment orchestrated by Kubernetes.

In production, Kafka simulates a real-time transaction stream of air quality readings, which are processed by the consumer service that calls the deployed model API for predictions. These predictions are then passed to a monitoring component powered by Evidently, which evaluates potential data or prediction drift against reference distributions captured during training. By combining real-time inference, automated monitoring, and modular deployment, this architecture provides a scalable and maintainable solution for environmental data intelligence.

The dataset contains 9358 instances of hourly average responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level,within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.[1]

[1] https://archive.ics.uci.edu/dataset/360/air+quality

System Architecture Design



Local Environment (Value Discovery)

In the local environment, the system begins by acquiring the UCI Air Quality dataset through the ucimlrepo API. This dataset includes hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device recorded from March 2004 to February 2005.

The data is first passed through a train-validation-test split to ensure proper model evaluation. Following this, EDA and data cleaning are performed to handle missing values, detect outliers, and transform raw input features. Once cleaned, the data proceeds to the feature engineering stage where meaningful attributes such as lags, rolling statistics, and temporal encodings are generated.

The engineered dataset is then used in the modeling phase, where multiple machine learning models are trained and tracked using MLflow. Key artifacts — including the trained model, StandardScaler, and column order — are saved and versioned in MLflow. After model training, a final validation step confirms the model's performance using holdout data. Once the best model is selected, it is exported and served through a Flask-based Model Serving API, ready for use in production.

Production Environment (Value Delivery)

In the production environment, the system transitions into real-time inference and monitoring. A Dockerized producer (Producer.py), orchestrated by Kubernetes, fetches test data from the UCI repository via API and streams it as real-time events to Kafka. Kafka publishes each record into a topic, simulating a real-time transaction stream.

A Dockerized consumer (Consumer.py) subscribes to this Kafka stream. Rather than directly using a saved .pkl file, it queries the best model from the Flask Model Serving API (labeled best_model (API)) to make predictions. These predictions, along with input data, are stored in local storage for monitoring and auditing purposes.

Parallel to this, a third container — Monitor.py with Evidently — reads prediction results and reference data to generate drift reports. This component detects any distributional changes or performance degradation between the model's training data and incoming real-time data. The reports can be visualized via an Evidently AI dashboard to support model governance and alerting.

All containers — producer, consumer, and monitor — are managed by Kubernetes, ensuring scalability, fault tolerance, and isolation in the production environment.

Model Experimentation with MLFlow

To select the best machine learning model, we evaluate the most accurate model by testing several models and fine-tuning hyper parameters. We run different models and track the experimentation using MLFlow. The modeling approach, model tuning, and evaluation are explained further in the sections below.

Data Splitting Strategy

To simulate a real-world scenario, we use chronological train-validation-test split to ensure robust model evaluation and avoid data leakage in this temporal context. The dataset was divided as follows:

- **Training Dataset**: From March 11, 2004 to October 31, 2004 — used to train predictive models. 5,646 records (60.34%)
- **Validation Dataset**: From November 1, 2004 to December 31, 2004 — used to tune hyperparameters and compare model performance. 1,464 records (15.65%)
- **Testing Dataset**: From January 1, 2005 to April 4, 2005 — used to evaluate the final model performance on unseen data. 2,247 records (24.01%)

To ensure that our modeling approach can operate in a real-time environment, we need to design the pipeline to integrate with Kafka's streaming architecture. Therefore, feature engineering will be the most important aspect that we need to consider whether it can be implementable in real-time situations.

Data Cleaning & Exploration Data Analysis

Before developing any analytical or predictive models, it is essential to thoroughly understand the dataset. This section presents key insights uncovered through exploratory data analysis (EDA) of the Air Quality dataset obtained from the UCI Machine Learning Repository.

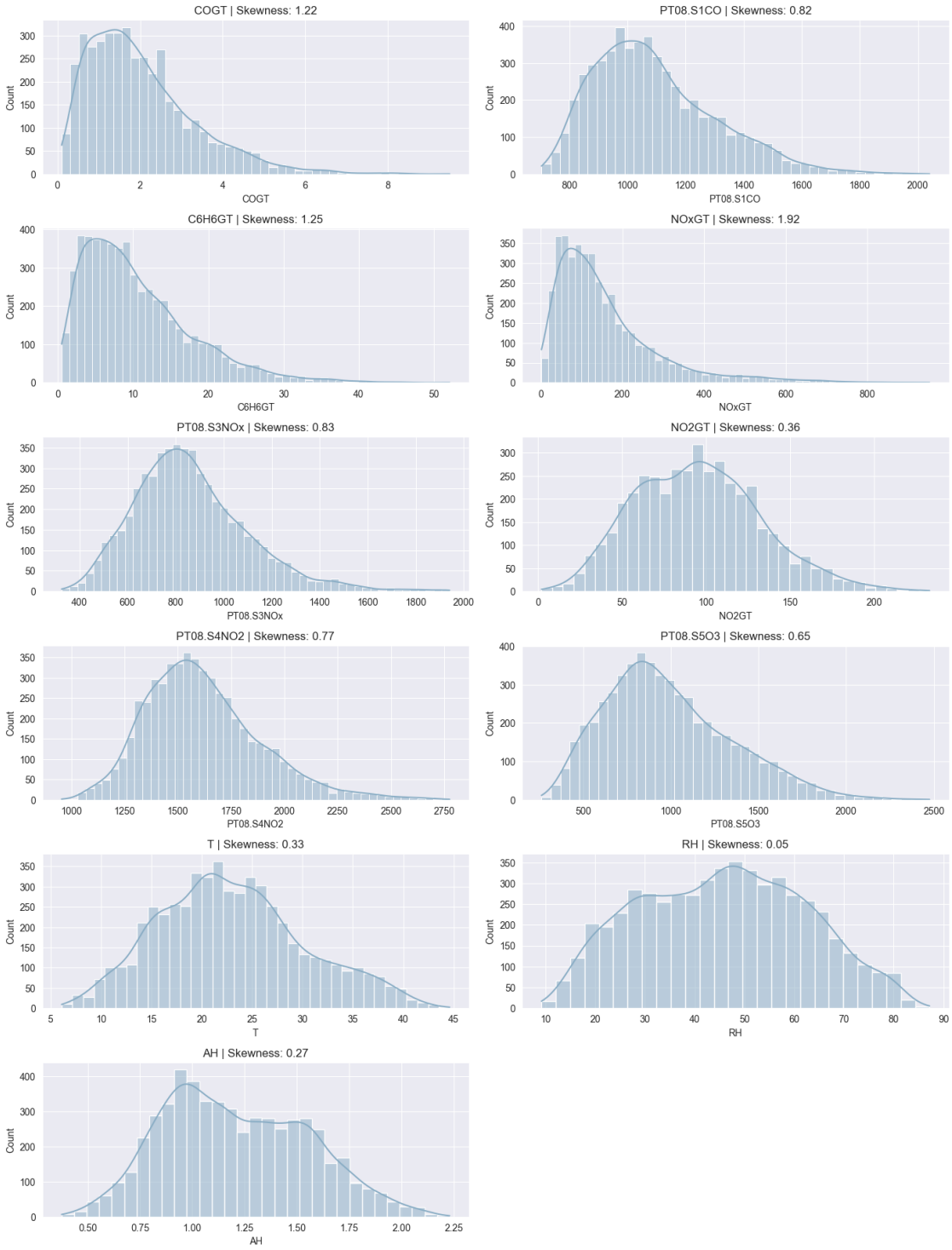
In general, we first replace the default placeholder for missing values (−200) with actual null values (NaN) to more accurately reflect the true nature of the data. All EDA is then conducted exclusively on the training dataset to prevent data leakage and maintain the integrity of model evaluation.

1. **Null Value**
   The dataset uses -200 as the default placeholder for missing values. We first replaced all instances of -200 with actual null values and conducted exploratory data analysis (EDA) to identify patterns in the missing data. Ultimately, we chose to forward-fill the missing values using the previous valid observation, as this approach preserves daily and weekly trends.

   Moreover, since NMHC(GT) contains high null values (83.90%), we decided to drop this pollutant from our analysis. And because the ground truth of NMHC contains a lot of null values, we cannot validate the sensor related to it, then we also drop PT08.S2NMHC from our analysis.

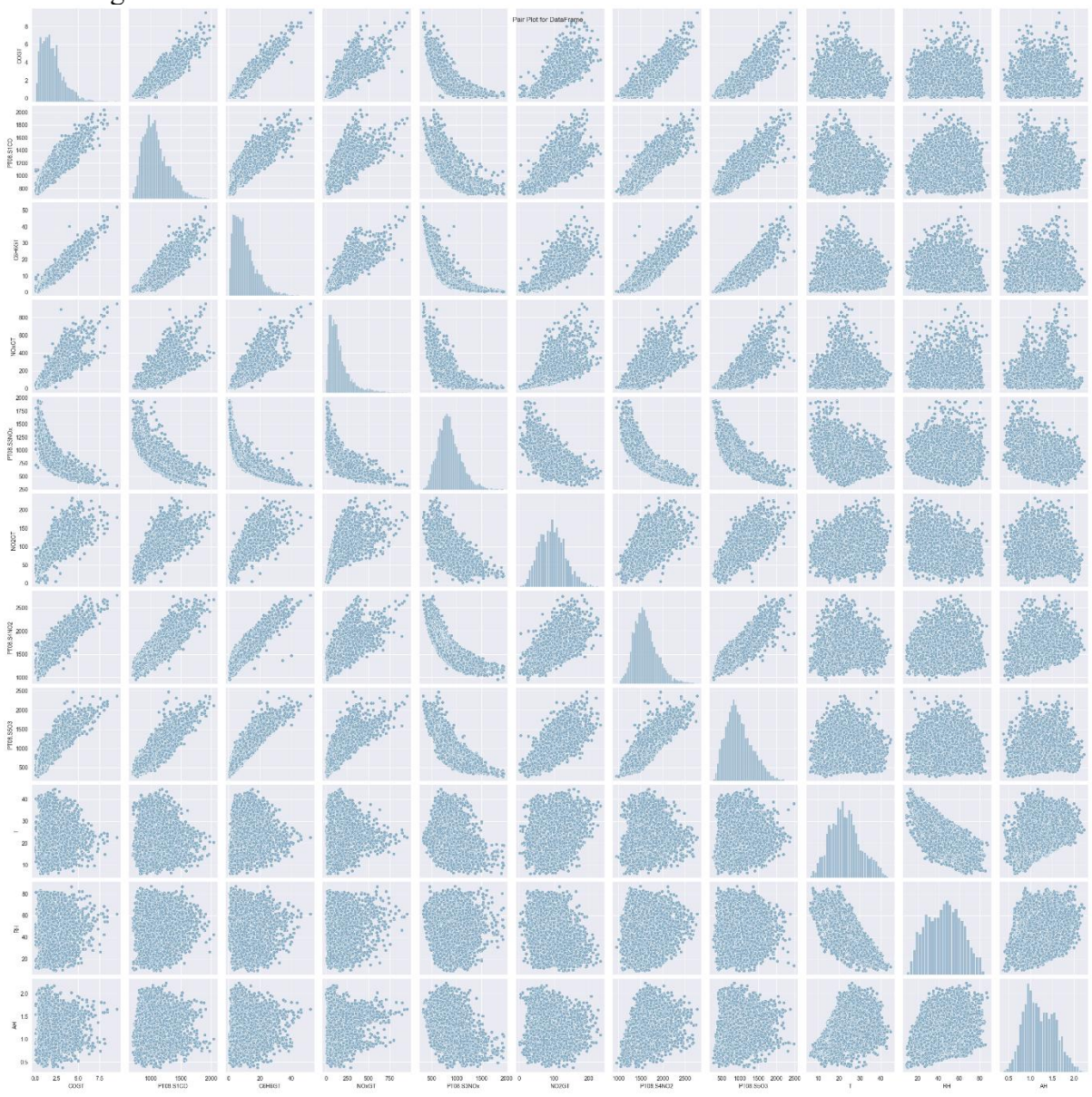
2. Distribution

Based on the distribution plot, we can notice that most of the data follow Gaussian distribution except for CO(GT), C6H6(GT), NOx(GT). The criteria are that if skewness is more than 1, we will determine the data is not following Gaussian distribution. This suggests that for these skewed features, we may need to apply some transformation to make it follow Gaussian distribution before modeling.

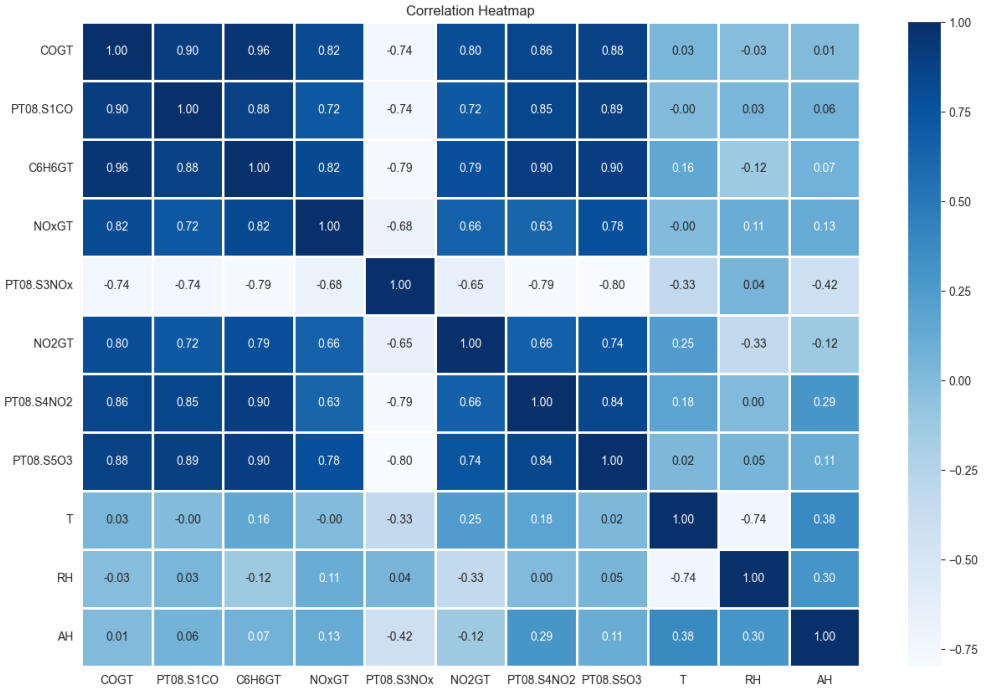


3. **Pair Plot**

Based on the pair plot, we can observe that most of the data are linearly related to each other, especially among the pollutant concentrations. This suggests that models well-suited for capturing linear dependencies, such as linear regression, may be effective for this dataset. Alternatively, it also indicates that certain pollutants could serve as useful predictors for estimating the concentrations of others.



4. **Correlation heatmap between different pollutants**
   Based on the heat map of correlation, we can observe that there are high correlations between the pollutant concentrations and its sensor readings. This suggests that we need to handle multicollinearity when we build a regression model.



5. **Time-series plots of CO, NOx, and Benzene concentrations**
   Based on the time-series plots for each pollutant, we can observe fluctuations over time but no clear long-term trend. This suggests that when building a SARIMA model, the trend component can likely be omitted. Alternatively, the ADF test can be conducted under the assumption of a constant mean without a trend.
   *Due to the limitation of the page, please refer to the full result in 02_DataCleaningAndEDA_for_Train.ipynb

6. **Daily/weekly patterns (average by hour of day, day of week)**
   Based on the daily and weekly patterns, we observe significant variations throughout the day and across different days of the week. This suggests that missing values should be filled using the previous valid observation, as this method helps preserve both hourly and weekly trends in the data.
   *Due to the limitation of the page, please refer to the full result in 02_DataCleaningAndEDA_for_Train.ipynb

7. **Autocorrelation and partial autocorrelation plots**
   Based on the autocorrelation plots, we observe that past values have a strong relationship with current values, indicating temporal dependence in the data. The partial autocorrelation plots further confirm that a few recent lags contribute significantly to the current value. This suggests that incorporating lagged features into our model is important for capturing the underlying time-dependent structure of the data, and that time series models like SARIMA are well-suited for this task.
   *Due to the limitation of the page, please refer to the full result in 02_DataCleaningAndEDA_for_Train.ipynb

8. **Decomposition of time series into trend, seasonality, and residuals**
   Based on the decomposition of the time series into trend, seasonality, and residuals both 24 lags and 168 lags, we observe clear seasonal patterns and some level of noise, but no strong long-term trend. This suggests that modeling efforts should focus on capturing seasonal effects and minimizing residual variance, while the trend component can be considered omitted.
   *Due to the limitation of the page, please refer to the full result in 02_DataCleaningAndEDA_for_Train.ipynb

Feature Engineering Technique

1. **Time-based features**
   Based on the timestamp provided in the dataset, we extracted several time-based features to help the model learn patterns related to time. These include:
   - Hour of the day to capture daily cycles
   - Day of the week to capture weekly cycles

   However, we cannot perform the monthly features because it will not be implementable in the real-time environment. This happens because the training dataset only contains the data from March to October. If the real data happens during November to February, the model will not know the pattern during that period. This leads to the decision not to develop the monthly feature to capture the yearly cycle.

2. **Lagged features**
   Based on the features provided in the dataset, we extracted several lagged features to represent pollutant values from previous time steps and are essential for capturing temporal dependencies. Specifically, we generated lag features for each pollutant variable as follows:
   - 1 to 3 hours lag: Capture immediate temporal dependencies
   - 6 to12 hours lag: Capture intra-day patterns
   - 24 hours lag: Captures daily seasonality and repeated behaviors from the previous day

   This allows the model to learn from a full day historical data when making predictions.

3. **Rolling statistics features**
   Based on the features provided in the dataset, we extracted several rolling statistics features including mean, standard deviation, maximum and minimum to help smooth out short-term noise and highlight consistent patterns over recent time windows. We generated rolling features for each pollutant variable using the following window sizes: 3, 6, 12, and 24 hours.
   - **Rolling Mean**: Captures the average pollutant level over a specific recent window. Helps identify underlying trends and persistent pollution levels.
   - **Rolling Standard Deviation**: Captures the volatility of pollutant levels within the rolling window. Useful for detecting unstable or highly variable conditions.
   - **Rolling Maximum**: Captures the highest pollutant level observed within the window. Highlights potential peaks or short-term spikes in pollution.
   - **Rolling Minimum**: Captures the lowest pollutant level within the window. Useful for identifying diurnal dips or short-term recovery in air quality.

To ensure that data leakage is prevented during model training, rolling features are created by shifting the data forward in time. This means that, at any given timestamp, the model only has access to information from the past—never from the future. For example, the features to predict the data at 2004-03-11 02:00:00 will use the average from 2004-03-11 00:00:00 to 2004-03-11 01:00:00.

	date_time	COGT	rolling_COGT_mean_2	rolling_COGT_mean_3
0	2004-03-11 00:00:00	1.2	NaN	NaN
1	2004-03-11 01:00:00	1.0	NaN	NaN
2	2004-03-11 02:00:00	0.9	1.10	NaN
3	2004-03-11 03:00:00	0.6	0.95	1.033333
4	2004-03-11 04:00:00	0.6	0.75	0.833333

*Validation dataset will be used in consumer.py as the historical context for calculating lagged and rolling features.

4. **Scaling features**
   To ensure feature magnitude consistency and improve the performance of linear models, we applied StandardScaler to all features. Instead of saving the scaler separately, we log the trained scaler as an MLflow artifact during model training. This artifact is then downloaded and reused by consumer.py to transform test data using the same scaling strategy.

5. **Column Ordering**
   To maintain feature order consistency between training and inference, we store the list of training feature columns in a .json file. This column schema file is also logged as an MLflow artifact, ensuring that consumer.py uses the exact column structure for prediction.

Modeling

To ensure optimal performance, each selected model was fine-tuned to minimize prediction error. Model tuning involves identifying the best combination of hyperparameters that improve accuracy and generalization.

- Simple Linear Regression: No specific setting
- Linear Regression with Elastic Net
  - Performed grid search to tune the balance between L1 (Lasso) and L2 (Ridge) regularization using the l1_ratio and alpha hyperparameters.
    - l1_ratio: [0.1, 0.5, 0.9]
    - alpha: [0.01, 0.1, 1.0, 10]
- Light Gradient Boosting Machine (LightGBM)
  - n_estimators=1000
  - learning_rate=0.01

Evaluation Methodology

To evaluate the accuracy of the model, Simple Linear Regression is used as our baseline model. Using the same features and dataset, we run ElasticNet and LightGBM using different hyper parameters; l1_ratio and alpha for ElasticNet and n_estimators and learning rate for LightGBM. The experiment results are logged in MLFlow for tracking and visualization purposes. We train each model using a training dataset and evaluate the model using a validation dataset. We compare the accuracy of different models using Root Mean Squared Error (RMSE).

Model Experiment Result

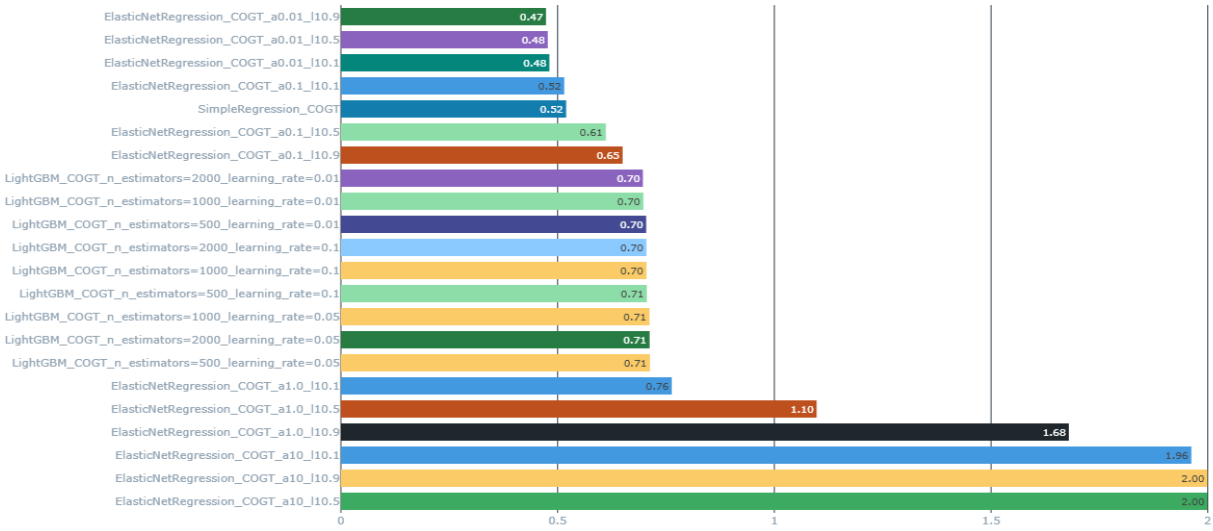


Figure 1: RMSE from validation dataset of CO(GT) prediction using various models

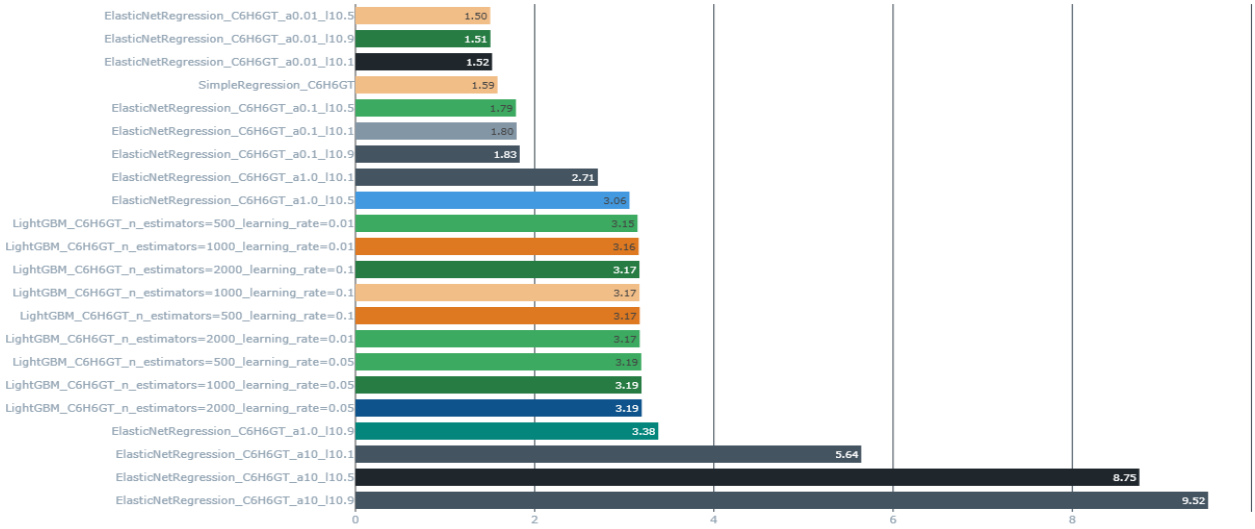


Figure 2: RMSE from validation dataset of C6H6(GT) prediction using various models

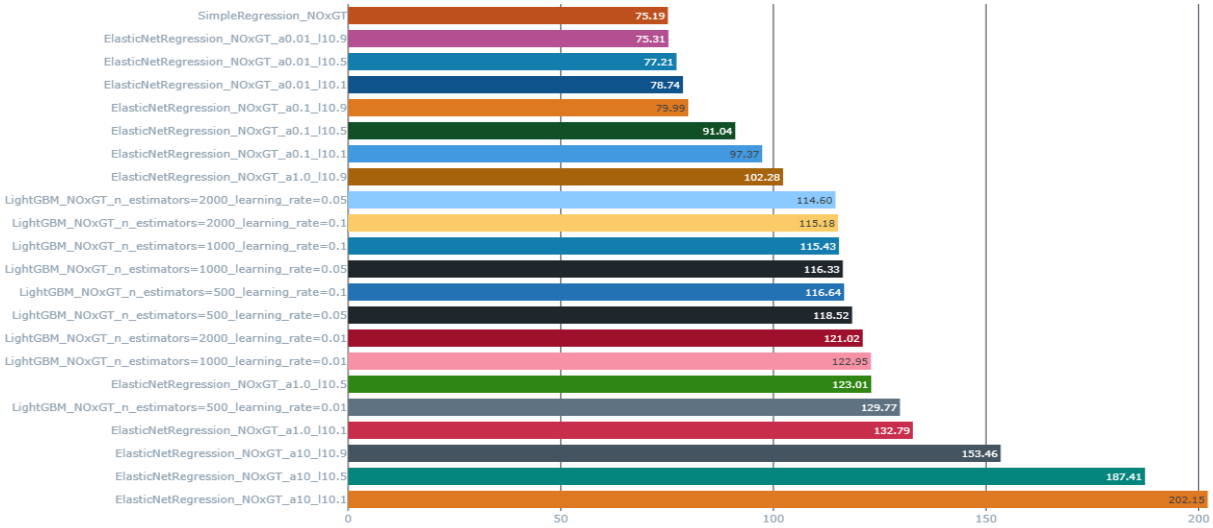


Figure 3: RMSE from validation dataset of NOx(GT) prediction using various models

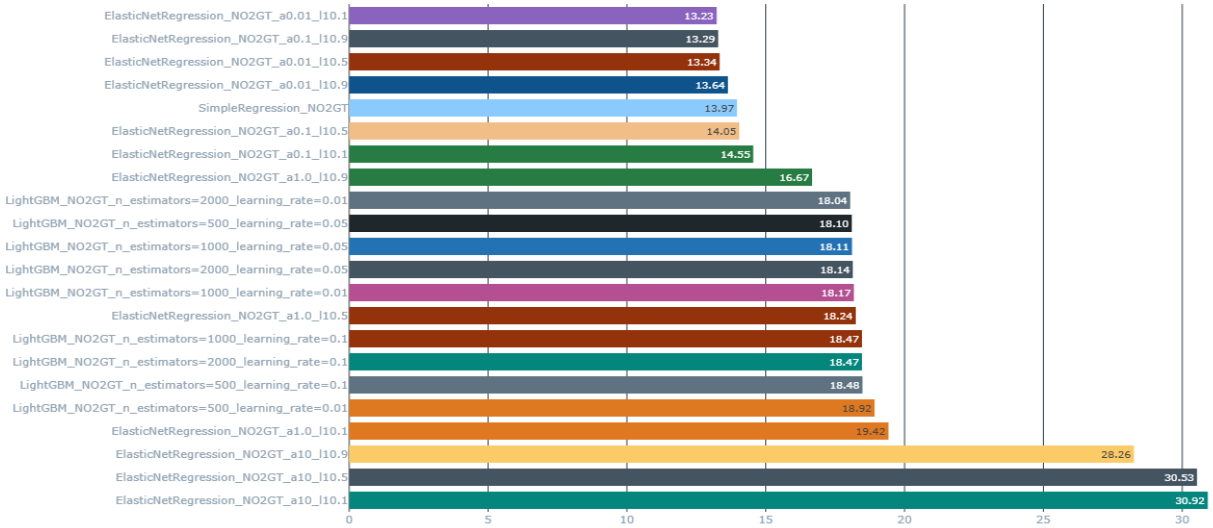


Figure 4: RMSE from validation dataset of NO2(GT) prediction using various models

The results visualized by MLFlow are shown in figure 1-4. The model that predicts CO(GT) with the least RMSE in the validation dataset is Linear Regression with Elastic Net with alpha = 0.01 and l1_ratio = 10.9. Linear Regression with Elastic Net with alpha = 0.01 and l1_ratio = 10.5 gives the least RMSE for predicting C6H6(GT) in the validation dataset. For NOx(GT), our baseline mode, Simple Linear Regression, has the lowest RMSE. Linear Regression with Elastic Net is also the model that achieves the lowest RMSE for predicting NO2(GT), using hyper parameters as alpha = 0.01 and l1_ratio = 10.1.

Project Plan & Timeline

No	Task	Description	PIC	Due Date
1	Model Experimentation	Select model approach (Best model tuning, Ensemble, or New model)	Tua	4/3/2025
2	MLFlow Setup	Install MLFlow and configure tracking server	Tua	4/3/2025
3	Kafka Producer Setup	Fetch UCI Air Quality dataset and implement producer.py to publish data to Kafka	Luis	4/4/2025
4	Kafka Consumer Setup	Implement consumer.py to consume Kafka messages and send them to the model	Luis	4/5/2025
5	MLFlow Integration	Implement model_training.py to train, validate, and log models in MLFlow	Tua	4/6/2025
6	Feature Engineering	Conduct preliminary feature engineering for model improvement	Tua	4/6/2025
7	System Architecture	Draft system architecture diagram and finalize key components	Ice	4/7/2025
8	Project Plan	Develop project timeline, assign roles, and outline learning objectives	Sam	4/7/2025
9	Monitoring Dashboard	Integrate Evidently for monitoring data drift and model performance	Tua	4/8/2025
10	Model Evaluation	Log training metrics in MLFlow and refine model selection	Tua	4/8/2025
11	Interim Report	Write model experimentation results section	Tua	4/9/2025
12	Interim Report	Write system architecture explanation	Ice	4/9/2025
13	Interim Report	Complete project plan and timeline section	Sam, Ting	4/13/2025
15	Code Repository	Push MLFlow experiments, Kafka scripts, and README setup guide	Luis, Ice, Tua	4/13/2025
16	Interim Deliverable Submission	Submit report	Sam	4/14/2025
17	Interim Deliverable Submission	Submit code repository	Tua	4/14/2025
17	Model Refinement	Perform hyperparameter tuning and final model selection	Tua	4/16/2025
18	Model Training	Train final model on full dataset	Tua	4/16/2025
19	Model Conversion	Convert trained model into deployable format (e.g., .pkl, ONNX)	Tua	4/16/2025
20	Model Serving API	Implement FastAPI/Flask for serving model inference	Ice	4/18/2025

21	Kafka Integration	Connect API with Kafka consumer	Luis, Ice	4/18/2025
22	Deployment Setup	Containerize model using Docker	Ice	4/19/2025
23	Kubernetes Deployment	Deploy model using Kubernetes	Ice	4/21/2025
24	Monitoring Dashboard	Finalize Evidently monitoring setup	Luis	4/22/2025
25	Final Report (Rough Draft)	Refine and complete interim report (20-25 pages)	Sam, Ting	4/23/2025
26	Final Report (Final Draft)	Refine and complete final report (20-25 pages)	Sam, Ting	4/25/2025
27	Code Cleanup	Ensure repository is well-documented and structured	Luis, Ice, Tua	4/25/2025
28	Presentation & Video	Create slides and record project presentation video	Ting, Sam	4/25/2025
29	Final Testing	Perform end-to-end testing of pipeline and fix bugs	Luis, Ice, Tua	4/26/2025
30	Final Deliverable Submission	Submit final report, code, and presentation	Entire Team	4/28/2025

*Cells highlighted in yellow indicate that these steps have been completed.
*PIC: Person in Charge

Team Structure & Responsibilities

Team Member

Name	Andrew ID	Nick Name
Pongsakorn Supakpanichkul	psupkpa	Tua
Samuel Lin	szl	Sam
Vitchakorn Sayanwisuttikam	vsayanwi	Ice
Wan-Ting Tsai	tinat	Ting
Luis Carlos Solis Ochoa	lsolisoc	Luis

Role & Responsibilities

Role	Responsibilities	PIC
Data Engineer	1. Fetch the UCI Air Quality dataset. 2. Implement producer.py to publish data records to Kafka. 3. Implement consumer.py to consume messages and send them to the model for inference. 4. Set up and manage Kafka topics. 5. Ensure robust data pipeline with retries/logging.	Luis
Data Scientist	1. Overall System Architecture 2. Implement model_training.py to train, validate, and select the best ML model. 3. Integrate MLFlow for experiment tracking and model versioning. 4. Integrate Evidently for monitoring data quality, drift, and model performance. 5. Output a model artifact ready for deployment (e.g., .pkl, .onnx, etc.).	Tua
MLOps Engineer	1. Containerize the trained model using Docker. 2. Create and manage Kubernetes manifests for scalable deployment. 3. Build and expose a prediction API endpoint (e.g., via Flask or FastAPI). 4. Connect the deployed service to receive requests from consumer.py. 5. Automate deployment with CI/CD if time allows.	Ice
Data Analyst	1. Research the meaning, normal ranges, and health implications of pollutants 2. Write descriptive text for the EDA section, helping tell the story behind the graphs. 3. Project presentation & Video	Ting
Project Management	1. Keep the team on track with the 3 phases + report timeline 2. Schedule and document meetings 3. Communicate with TA and Professor 4. Project presentation & Video	Sam

*PIC: Person in Charge

Our project team has clearly delineated roles to ensure a structured and efficient workflow in executing the tasks related to air quality data analysis and model deployment.

Luis, serving as the Data Engineer, is responsible for initiating the data handling process by fetching the UCI Air Quality dataset. He implements producer scripts to publish data records to Kafka and consumer scripts to consume these messages, forwarding them for inference purposes. Additionally, Luis manages Kafka topics to maintain data integrity and ensures the robustness of the data pipeline by incorporating retries and comprehensive logging.

Tua, our Data Scientist, oversees the overall system architecture, ensuring that all components work cohesively. Tua is tasked with developing model_training scripts to train, validate, and ultimately select the most effective machine learning model. He leverages MLFlow to facilitate experiment tracking and manage model versioning and integrates Evidently to monitor data quality, drift, and model performance comprehensively. The culmination of Tua's work is the delivery of a model artifact suitable for deployment, formatted appropriately as a .pkl or .onnx file.

Ice, our MLOps Engineer, takes the finalized model from Tua and prepares it for deployment. He containerizes the trained model using Docker to enhance portability and ease of deployment. Ice creates and manages Kubernetes manifests to ensure scalable and reliable deployment environments. He also develops and exposes a prediction API endpoint, typically leveraging frameworks like Flask or FastAPI, connecting it to the consumer scripts implemented by Luis. When time permits, Ice aims to automate the deployment process further using CI/CD pipelines.

As the team's Data Analyst, Ting dives deeply into the dataset's implications, researching the meanings, normal ranges, and potential health impacts of the pollutants measured. Ting crafts descriptive text to accompany exploratory data analysis (EDA), effectively narrating the data's insights and enhancing the interpretability of graphical presentations. Furthermore, Ting contributes significantly to the project's final presentation and accompanying video, clearly communicating findings to broader audiences.

Finally, Sam, as Project Manager, orchestrates the team's progress, ensuring adherence to the project's structured timeline, including its three distinct phases and the final report. Sam schedules and meticulously documents meetings, ensuring transparency and effective communication among team members. Additionally, Sam serves as the primary liaison with the teaching assistant and professor, facilitating smooth communication and providing regular updates. He will take responsibility for compiling, editing, and finalizing all deliverables. He also takes an active role in the project's final presentation and video, consolidating the team’s collective efforts into a coherent narrative.