Real-Time Air Quality Prediction with Apache Kafka

Anshika Shukla

Carnegie Mellon University, M.S. Healthcare Data Analytics and Information Technology

Fundamentals of Operationalizing AI

Dr. Anand S. Rao

Heinz College of Information Systems and Public Policy

September 23rd, 2025

This report is submitted in partial fulfillment of the 94879 Individual Project requirements.

1. Executive Summary

Air pollution poses a serious challenge to public health and urban sustainability. Carbon Monoxide (CO), Nitrogen Oxides (NOx), Benzene (C6H6), and related pollutants are primarily driven by traffic emissions and industrial activity. Exposure to elevated pollutant levels can aggravate respiratory conditions, increase hospital admissions, and reduce overall quality of life. Traditional monitoring approaches are reactive, offering limited foresight into short-term spikes. This creates a pressing need for real-time, predictive air quality monitoring systems that can forecast pollution levels and support timely decision-making.

The problem addressed in this project is: How can we leverage streaming data and machine learning models to forecast air quality with a focus on CO levels in real-time, enabling operational interventions before harmful exposure occurs?

Leveraging the UCI Air Quality dataset, my analysis showed that CO levels are highly correlated with Benzene and moderately correlated with NOx, indicating common sources such as traffic and industrial emissions. Controlling these sources can reduce multiple pollutants at once. It was also observed that pollution levels follow predictable cycles, with higher levels during commuting hours and on weekdays compared to weekends, highlighting the significant influence of human activity. The presence of outliers in pollutant concentrations signalled the unusual pollution events (e.g., weather inversions, industrial incidents), which real-time detection can help mitigate. Lastly, it was also observed that the past pollutant levels are strong predictors of future values, supporting the use of ensemble learning models with lag features in real-time forecasting.

The integration of Kafka-based streaming pipelines with machine learning models (Random Forest, SARIMA, XGBoost) demonstrates the feasibility of predicting air quality in real time. This capability has direct business and societal impact. For policymakers, this enables dynamic traffic management where appropriate congestion charges, alternate-day driving policies, etc., can be enforced. For Healthcare Systems, it can act as an early warning system to vulnerable populations, reducing emergency admissions. For Industry & Environment Agencies, this can facilitate compliance monitoring and proactive emission control. Lastly, for Citizens, it can deliver actionable insights, such as avoiding outdoor activity during forecasted pollution peaks.

Overall, this indicates that by forecasting pollutant levels in advance, we are taking a step towards transforming air quality monitoring from reactive observation to proactive prediction. By focusing on CO forecasting while leveraging correlations with other pollutants, the approach provides a scalable framework that can guide interventions, safeguard health, and support smarter urban planning.

2. Technical Architecture and Infrastructure Implementation

The diagram below shows an end-to-end real-time air quality prediction pipeline that utilizes Apache Kafka. The system ingests environmental sensor data, performs exploratory data analysis (EDA), and applies machine learning models (Random Forest, XGBoost, SARIMA) to predict Carbon Monoxide (CO(GT)) levels. The design integrates data streaming with predictive analytics for operational decision-making.

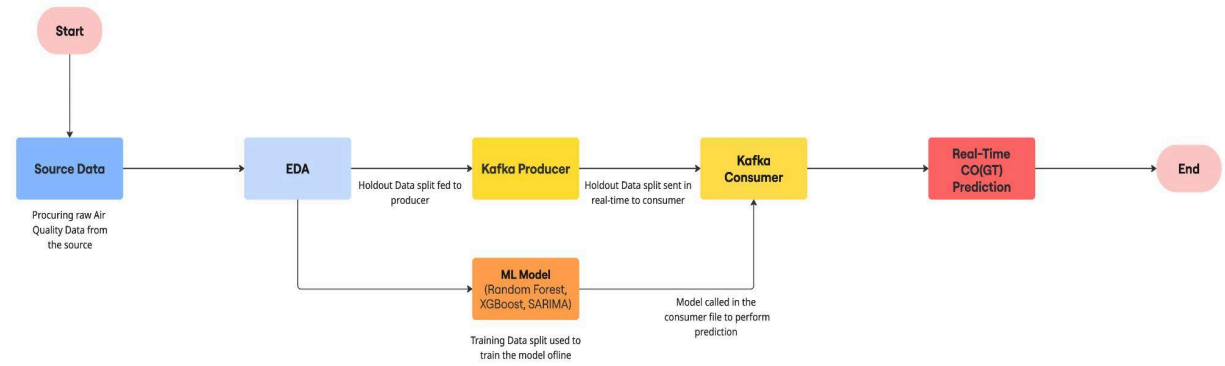

Figure 1. Architecture of Pipeline

2.1 Architecture Configuration Overview

The raw data is sourced from the UCI Air Quality dataset, containing hourly sensor readings. This data goes through Exploratory Data Analysis (EDA), where preprocessing and the cleaning of missing and imputation of missing values are taken care of. We also did some additional feature engineering where features like lag features, temporal features, and rolling averages were computed.

This data was then split further into a training dataset (90%), which is used to train our various ML models, and a holdout dataset (10%) that was fed to the Kafka producer to test our model on real-time data. These models are trained offline (Random Forest, XGBoost, SARIMA). The Kafka consumer reads the streamed data, transforms it, and calls the trained machine learning model for predictions. The Consumer outputs CO(GT) predictions for each incoming data record from the producer in real-time.

2.2 Kafka Ecosystem Design

Leveraged Docker to containerize and orchestrate all components of the Kafka ecosystem, ensuring portability and ease of deployment. The holdout datasets (holdout_data.csv for Random Forest/XGBoost and sarima_holdout_data.csv for SARIMA) were used to simulate real-time sensor data. The producer.py script streamed records row-by-row into Kafka topics, with dataset

selection based on the model. On the consumer side, consumer.py (for Random Forest/XGBoost) and consumer_sarima.py (for SARIMA) subscribed to topics and processed incoming messages. The consumer also loaded pre-trained model artifacts (.joblib, .json) to generate real-time forecasts on the streamed data. Predictions were logged and evaluated for performance, enabling operational decision-making for proactive air quality monitoring.

2.3 Infrastructure Challenges and Solutions Implemented

One of the main infrastructure challenges was understanding the Kafka producer-consumer workflow and its interaction with the predictive models. Initially, there was confusion regarding whether the producer, consumer, and model should all handle parts of the training and prediction cycle. This was resolved by designing a sound architecture diagram that clarified responsibilities. Where the producer was responsible solely for streaming the data, the consumer loads the pre-trained model, and the real-time prediction is generated at the consumer end.

Another challenge was determining how to correctly incorporate the holdout dataset into Kafka without inadvertently retraining inside the consumer. The solution was to train the model offline using the training data, then configure the producer to stream only the holdout test data, while the consumer loads the saved pre-trained model artifacts and performs real-time predictions. This separation ensured a clean architecture with no data leakage and a reliable simulation of real-world streaming analytics.

During the real-time prediction pipeline, I faced issues while ensuring feature consistency between incoming Kafka messages and the trained model. Initially, predictions were inaccurate because the features were not aligned in the same order as the model's training data. To address this, consumer.py was updated to explicitly reorder the incoming data using df = df[feature_cols] and added further reiterative preprocessing steps in the consumer file. This ensured that all features matched the training order before being passed to the model.

There were occasions when the consumer stopped reading Kafka messages for one of the models after splitting the scalers. Upon investigation, it was confirmed that the Kafka topic and scaler paths were correct. The root cause was that each model required a separate configuration in the consumer scripts. Updating the consumer to include distinct configs for each model resolved the issue and restored proper message processing.

Lastly, evaluating model performance during live Kafka streaming posed a challenge. Initially, only individual predictions were printed, making it difficult to assess overall accuracy. To address this, consumer.py was updated to collect y_true and y_pred values for all incoming messages and calculate aggregate metrics such as MAE and RMSE once streaming was complete. This enabled proper performance evaluation in real-time.

3. Data Intelligence and Pattern Analysis

Exploratory data analysis revealed strong correlations among pollutants, with CO highly correlated with Benzene (C6H6, r ≈ 0.93) and moderately correlated with NOx (r ≈ 0.79). This indicates that urban pollution sources, such as traffic and industrial activity, often emit multiple pollutants simultaneously. This shows, monitoring CO levels can provide indirect insights into other pollutants, enabling city regulators to implement targeted interventions.

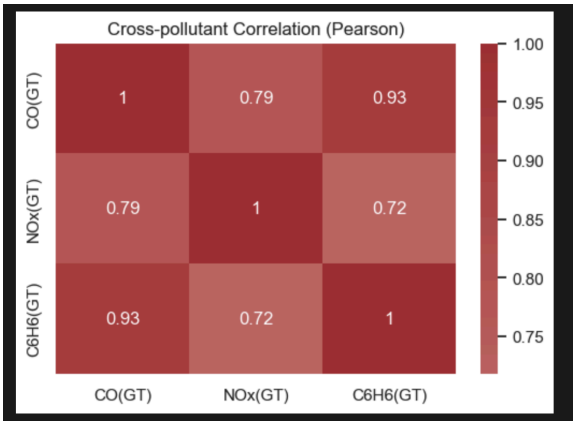

Figure 2. Pollutant Correlation Heatmap

Analysis of temporal patterns highlighted clear daily and weekly cycles. CO, NOx, and Benzene levels consistently peak during morning and evening hours, reflecting human activity patterns like commuting, and differ significantly between weekdays and weekends. This emphasizes the importance of incorporating time-of-day and day-of-week features into real-time predictive models.

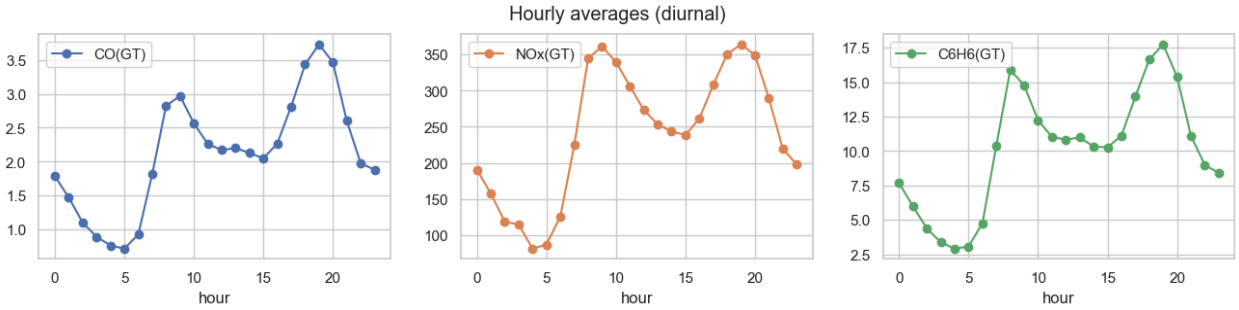

Figure 3. Hourly Temporal Patterns

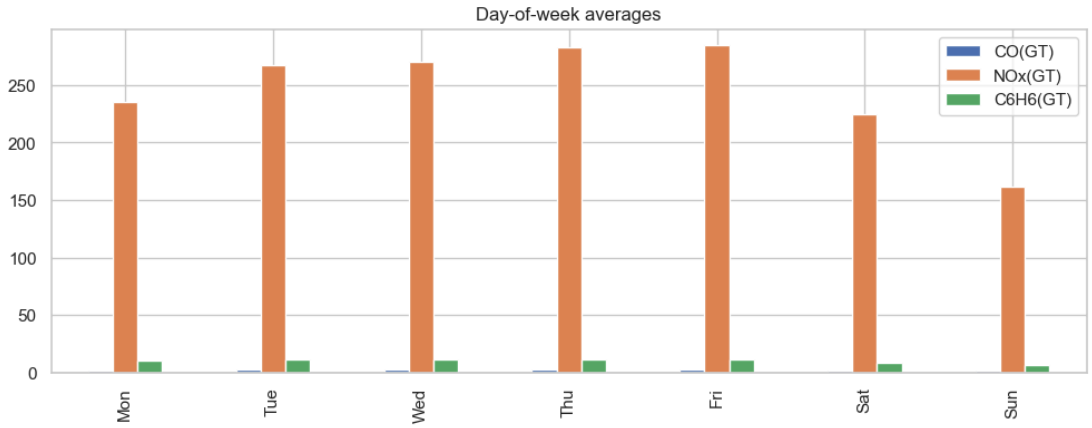

Figure 4. Weekly Temporal Patterns

Seasonality in CO was also evident, with STL decomposition confirming strong 24-hour cycles. Such predictable patterns suggest that forecasting models like SARIMA or Random Forest with lag features are particularly well-suited for anticipating daily pollutant fluctuations. This allows public health agencies to issue proactive advisories.

Anomaly detection further enhanced the understanding of extreme events. Several global anomalies (z-score > 3) and local anomalies (via rolling IQR) were observed in CO levels, likely representing events such as industrial accidents or weather-driven pollution spikes. These findings underscore the value of real-time monitoring and alerts for city planners and health agencies to respond swiftly to hazardous conditions.

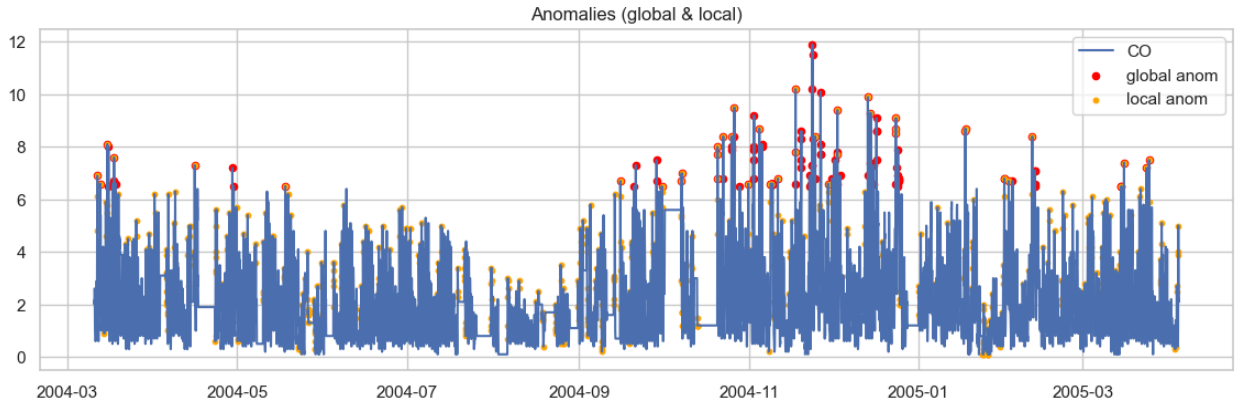

Figure 3. Global and Local Anomalies in CO

Statistical testing confirmed a significant difference between weekday and weekend CO levels (t-test, p ≈ 4e-116), reinforcing the notion that traffic is a dominant pollution source. Additionally, lag analysis showed strong autocorrelation in CO, particularly at lag-1, lag-6, lag-12, and lag-24, indicating that past pollutant levels are powerful predictors of near-future concentrations. This validates the use of lagged features in predictive models and demonstrates that real-time streaming predictions can be reliably generated from recent sensor data.

Overall, these analyses reveal that air quality is structured rather than random, closely tied to human activity cycles and industrial patterns. Strong correlations among pollutants suggest that interventions targeting a single source can have broader multi-pollutant impacts. By leveraging predictive modeling with lag features, seasonality, and anomaly detection, city planners, policymakers, and health agencies can make data-driven, targeted decisions to mitigate pollution exposure and protect public health effectively.

4. Predictive Analytics and Model Performance

4.1 Feature Engineering

Feature engineering played a critical role in capturing the temporal dynamics and autocorrelation present in CO levels. Temporal features such as hour of the day and day of the week were included to account for predictable patterns driven by human activity and industrial schedules, which influence daily and weekly pollution cycles. Lagged features, including CO_lag_1, CO_lag_3, CO_lag_6, CO_lag_12, and CO_lag_24, were incorporated to leverage the strong autocorrelation observed in CO levels, allowing the model to use past pollutant measurements as predictors for near-future concentrations.

To better capture the cyclical nature of daily pollution patterns, the hour of the day was encoded using sine and cosine transformations (hour_sin and hour_cos). Traditional numeric encoding of hours treats 23 and 0 as far apart, but in reality, they are adjacent in the daily cycle. Additionally, rolling statistics, such as the rolling mean and standard deviation of CO over 6- to 24-hour windows, were computed to capture short and medium-term trends, smoothing out transient spikes while retaining meaningful variations. Together, these engineered features provided the model with a rich, temporally aware dataset, enhancing its ability to generate accurate and timely real-time predictions.

4.2 Model Development

Random Forest Training and Evaluation

The Random Forest Regressor was trained using a time-series-aware cross-validation approach with TimeSeriesSplit to preserve temporal ordering. Preprocessing included median imputation for sensor features, forward- and backward-filling for lag and rolling features, and optional scaling. Hyperparameters were optimized via grid search across n_estimators, max_depth, and min_samples_split.

XGBoost Training and Evaluation

XGBoost followed a similar preprocessing pipeline and was trained using gradient boosting with TimeSeriesSplit cross-validation. A grid search over learning_rate, max_depth, n_estimators, and subsample identified the optimal parameters (learning_rate=0.05, max_depth=10, n_estimators=200, subsample=0.8).

SARIMA Training and Evaluation

The SARIMA (Seasonal ARIMA) model was trained to explicitly leverage daily and weekly seasonality, incorporating exogenous (external factors affecting CO level) predictors, including sensor measurements and cyclical temporal features (hour_sin, hour_cos). Lagged CO features were excluded, making the model purely dependent on current exogenous inputs and seasonality patterns.

Summary of Model Performance

Across all models, the baseline lag-1 forecast established a reference point for predictive accuracy. Random Forest and XGBoost both captured complex feature interactions, with XGBoost demonstrating superior generalization on holdout data. SARIMA offered interpretable, seasonally-informed predictions, useful for understanding underlying temporal patterns but less robust to unseen variations. We saw the importance of combining machine learning models for high-accuracy real-time forecasting with classical time-series models for interpretable seasonality insights. XGBoost, with its low holdout error and stable performance, emerges as the best candidate for integration into the live Kafka streaming prediction pipeline.

4.3 Model Evaluation and Comparative Analysis

To ensure temporal integrity, the dataset was split chronologically into training and holdout data sets using a 90/10 split, respectively. The first 90% of observations were used for offline model training and validation. The holdout dataset comprising the final 10% of the timeline was kept entirely separate to simulate real-time streaming predictions and prevent any information leakage from the training phase. Model performance was consistently evaluated using standard regression metrics, specifically Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), providing interpretable measures of prediction accuracy and error magnitude across both test and holdout datasets.

Random Forest, on the training/test split (last 14 days as test), the model significantly outperformed the naïve lag-1 baseline, achieving a test MAE of 0.198 and RMSE of 0.351, compared to baseline values of MAE 0.512 and RMSE 0.767. These results demonstrate that the model effectively captures nonlinear interactions among temporal, lagged, and rolling features. However, on holdout data, the Random Forest MAE increased to 1.643 and RMSE to 1.987, suggesting some overfitting to the training period or sensitivity to unseen temporal patterns.

On the training/test split, XGBoost achieved an MAE of 0.179 and an RMSE of 0.332, outperforming both the baseline and Random Forest. On holdout data, XGBoost maintained strong performance with an MAE of 0.151 and an RMSE of 0.248, indicating excellent generalization and robustness in predicting unseen pollutant trends. Its ability to capture complex nonlinear relationships and temporal dependencies makes it the most reliable candidate for real-time CO forecasting.

SARIMA, on the training/test split, showed modest improvements over the baseline, achieving a MAE of 0.504 and RMSE of 0.659, compared to baseline MAE 0.506 and RMSE 0.760. While less effective than machine learning models in capturing nonlinear interactions, SARIMA provides interpretable, seasonally-aware forecasts. On holdout data, SARIMA's performance decreased (MAE 1.078, RMSE 1.154), reflecting sensitivity to shifts in external conditions and highlighting its role as a complementary model alongside more flexible approaches like XGBoost.

4.4 Production Deployment Strategy and Monitoring Framework

The production deployment leverages a Kafka-based streaming architecture to operationalize the offline-trained models (Random Forest, XGBoost, and SARIMA) for real-time air quality prediction. Raw sensor data from the UCI Air Quality dataset is preprocessed, with temporal, lagged, and rolling features computed offline for Random Forest and XGBoost, while SARIMA relies only on exogenous predictors and cyclical temporal features.

The holdout dataset (10% of the chronological data) is streamed via a Kafka producer into designated topics corresponding to each model. On the consumer side, separate scripts (consumer.py for Random Forest/XGBoost and consumer_sarima.py for SARIMA) subscribe to their respective topics, preprocess incoming messages to ensure feature alignment and scaling, and call the trained model artifacts (.joblib, .json) to generate predictions in real time.

To maintain robustness and prevent errors, each consumer script is configured with model-specific settings, ensuring proper handling of multiple models and avoiding misalignment or scaling conflicts. The consumer also implements a monitoring framework: predictions and corresponding actual values are collected to compute aggregate evaluation metrics (MAE and RMSE) over the streaming period, allowing continuous performance tracking.

This framework enables early detection of prediction drift or model degradation, ensuring that forecasts remain reliable. The combination of containerized Kafka services, dedicated producer-consumer scripts, and a real-time evaluation pipeline provides a scalable, fault-tolerant, and interpretable deployment that supports proactive decision-making for urban air quality management.

5. Strategic Conclusions and Future Enhancements

5.1 Lessons Learned

The project highlighted that while Kafka provides a robust and scalable streaming infrastructure, careful attention to data quality and preprocessing is critical. Feature consistency, proper scaling, and alignment with the offline-trained model are essential to ensure accurate real-time predictions. Additionally, the predictive performance of the models is heavily influenced by feature engineering, including the use of lagged features, rolling statistics, and temporal encodings. Among the models evaluated, XGBoost offered a strong baseline due to its ability to handle nonlinear relationships and missing values; however, ongoing monitoring of streaming data for drift is necessary to maintain model reliability in a live environment.

5.2 Limitations

This implementation focused exclusively on predicting CO levels, despite the availability of multiple pollutants in the dataset, which limits the breadth of actionable insights. The current deployment relies on offline-trained, static models, with no automated retraining pipeline to adapt to changing data patterns. Furthermore, the Kafka setup is single-broker. It is lacking features such as replication, failover, and high availability, which are essential for robust real-time operations in a live monitoring system.

5.3 Future Enhancements

Future work can expand the system to multi-output forecasting, simultaneously predicting CO, NOx, Benzene, and NO2 to provide a more comprehensive view of urban air quality. Integrating a retraining loop with streaming feedback would enable models to adapt to changing pollution patterns in real time. Additionally, exploring hybrid architectures such as LSTM-ARIMA models could better capture long-term temporal dependencies and seasonality. Finally, deploying production-grade Kafka infrastructure with multiple brokers, replication, and enhanced monitoring will improve reliability, scalability, and operational robustness for a fully operational real-time air quality prediction system.

6. References and Professional Resources

- Kramer, Kathryn. "Stream Processing: A Game Changer for Air Quality Monitoring." *RisingWave Blog*, 24 July 2024, https://risingwave.com/blog/stream-processing-a-game-changer-for-air-quality-monitoring/ *Industry perspective on streaming technologies for environmental monitoring applications.*
- Shapira, Gwen, Todd Palino, Rajini Sivaram, and Krit Petty. *Kafka: The Definitive Guide*. 2nd Edition. O'Reilly Media, Inc. November 2021. *Comprehensive technical reference for Apache Kafka architecture and implementation patterns.*

- Apache Kafka Documentation: https://kafka.apache.org/documentation/ *Official technical documentation and configuration reference.*
- Docker, Inc. (2025). Docker documentation. Retrieved September 23, 2025, from https://docs.docker.com
- OpenAI. (2023). ChatGPT [Large language model]. Retrieved September 23, 2025, from https://chat.openai.com

Appendix A. ChatGPT Usage History

ChatGPT was primarily used to debug code, GitHub CLI codes, to do various setups (Docker/kafka/joblib files), write the .md files/report, SARIMA model code, and write integrated code that can be used to build an ML pipeline.

>> Please Access Chats here:

- https://chatgpt.com/share/68d356a7-3b28-800a-b33a-0f874dfcddd2
- https://chatgpt.com/share/68d35518-1d90-800a-b223-e64cb451c6bc
- https://chatgpt.com/share/68d36144-969c-800a-91a3-05eb9ccbc0b2
- https://chatgpt.com/share/68d36190-95cc-800a-bcaf-c07db4a36276

Appendix B. Repository

https://github.com/anshika2698/Real-Time-Air-Quality-Prediction-with-Apache-Kafka