Fundamentals of Operationalizing AI: Assignment 1 Report

Name: Dharmesh Agase
Andrewid: dagase

Index

1. Introduction

Urban air quality monitoring is essential for public health and urban planning. As cities grow, rising pollutant levels such as Carbon Monoxide (CO), Nitrogen Oxides (NOx), and Benzene pose significant health risks. To address these challenges, this project sets up a real-time data pipeline using Apache Kafka in KRaft mode for efficient streaming, analysis, and forecasting of sensor data.

This project uses the UCI Air Quality Dataset, which contains 9,358 hourly records (March 2004 – February 2005) across 15 columns including date/time and pollutant measurements. It has missing values, marked as –200, which are handled with robust cleaning and imputation techniques to ensure reliable analysis

The project begins by establishing a robust data streaming pipeline using Apache Kafka in KRaft mode. This approach simulates a continuous stream of real-time sensor data, with a dedicated Kafka producer that streams preprocessed air quality records into the system. The Kafka consumers are also deployed to process and store the incoming data, ensuring that the raw data flows seamlessly into the analytical and modeling layers.

To ensure the quality and reliability of the data, a comprehensive exploratory data analysis (EDA) was done on the dataset. Missing values, which were originally marked as –200, were imputed using a forward-fill method. In addition, we generated a suite of time-based features such as Hour, DayOfWeek, and Month, and engineered lag and rolling statistics to capture the temporal dynamics inherent in the dataset. This multi-layered preprocessing not only improved data consistency but also reduced multicollinearity among features.

On the modeling side, two predictive approaches were developed and trained to forecast CO concentrations. One branch of the project utilized a Random Forest regression model, which leverages the engineered features to capture nonlinear interactions among variables. In parallel, an LSTM neural network model was developed to specifically learn sequential dependencies from normalized sequences of CO values. This dual-model strategy helps to compare and contrast the strengths of traditional machine learning methods with those of deep learning approaches in handling complex temporal patterns.

For real-time prediction deployment, the trained models are integrated into the Kafka consumer pipeline. The consumer designed for the Random Forest model extracts the necessary features from each incoming JSON message, while the LSTM consumer employs a rolling buffer strategy to compile a sequence of recent CO readings. If a new value is missing, the LSTM consumer intelligently uses the last predicted value to fill the gap. This mechanism ensures the rolling buffer remains complete, enabling uninterrupted, real-time forecasting of air quality.

This document further details the Kafka setup process, the data exploration and cleaning strategy, the development and training of the Random Forest and LSTM models, and the overall implementation of the real-time prediction pipeline, thereby offering a comprehensive solution for urban air quality monitoring and forecasting.

2. Kafka Setup Description

2.1 Installing Kafka,Kraft,JDK17, Python, Virtual Env & WSL

The binary file was downloaded and extracted to a folder and made sure that there was no space in

1. Downloading and Extract Kafka & Kraft:
    a. In this project, the latest version of Apache Kafka 4.0.0 was downloaded from this link https://kafka.apache.org/downloads.. The archive was extracted, and the installation directory was configured appropriately
    b. A new folder was created in config with the name kraft & server.properties file was copied and the log.dirs was changed to new location
2. Installing WSL & Python V-ENV:
    a. As I was using a Windows machine and it was difficult to run the linux cmds in powershell or cmd, I installed WSL.
    b. After WSL, python and virtual environment was installed using cmd `sudo apt install python3 python3-venv python3-pip`.
    c. JDK 17 was also installed for the WSL user.

2.2 Starting Kraft & Creating Kafka Topic

- With the configuration updated for KRaft, Kafka was started directly with the command: bin/kafka-server-start.sh config/server.properties. Few errors were encountered like metadata.log.dir settings missing or not found, which were resolved by verifying and correcting the logs file by removing the existing file and creating new uuid. Exact steps mentioned in the Appendix
- After starting the Kafka server in KRaft mode, a Kafka topic named "air_quality_topic" was created using: bin/kafka-topics.sh --create --topic air_quality_topic --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1. Errors indicating that the topic already existed were handled by listing the existing topics and choosing a unique topic name or deleting the duplicate.

2.3 Kafka Producer and Consumer Scripts

Producer Script:

- Functionality: Reads the UCI Air Quality dataset from a CSV file, preprocesses each record (including handling missing values), and sends the record to the Kafka topic with a time delay to simulate real-time streaming.
- Preprocessing: A function (preprocess_record) replaces missing values (–200) with None to ensure data consistency.

- Error Handling: The script includes try/except blocks to manage Kafka connection issues, CSV file reading errors, and record transmission problems

Consumer Script:

- Functionality: Subscribes to the Kafka topic, continuously listens for messages, and logs the received data.
- Error Handling: Similar error handling was implemented to manage JSON deserialization and data processing issues.

2.4 Data preprocessing strategy

1. Replacement with None:
    a. The strategy adopted was to replace –200 with Python's None. This replacement is executed in the Kafka producer script using the preprocess_record function.
    b. Using None clearly signifies that the value is missing.Many data processing libraries, such as pandas, recognize None (or convert it to NaN) for proper handling of missing data.This approach lays the groundwork for applying various imputation strategies (e.g., mean, median, or advanced methods) during subsequent data analysis phases

3. EDA: Air quality time series analysis

EDA was performed on the air quality dataset streamed from Kafka. The goal of this EDA was to understand the temporal patterns in pollutant concentrations, examine relationships among pollutants, and identify potential factors that influence air quality. The insights derived here lay the groundwork for the subsequent predictive modeling phase

3.1 Data Preparation and Cleaning

3.1.1 Data Conversion and loading:

Data Source: The dataset, streamed via Kafka, was stored in a CSV file (`air_quality_streamed.csv`).

Datetime Conversion: The 'Datetime' column was converted from a string format into a proper datetime object using `pd.to_datetime()`. This allowed the data to be indexed by time, enabling time series analyses.

Sorting and Indexing**:** The dataset was sorted by the datetime column and then set as the index, ensuring that all subsequent analyses respect the chronological order

3.1.2 Handle Missing Values

Initial Inspection: An initial assessment of missing values revealed that some columns, such as `CO(GT)`, `NOx(GT)`, and `NO2(GT)`, had moderate missing percentages (around 18%), whereas others like `NMHC(GT)` had a missing rate exceeding 80%.
Strategy Applied:

- NMHC(GT): Dropped due to its high missing rate, as it would introduce too much noise into the analysis.
- Remaining Columns: Missing values were handled using forward-fill (`ffill`), which propagates the last valid observation forward. This approach is justified by the gradual nature of environmental changes in urban air quality

3.2 EDA Graphs

3.2.1 Time Series Plot of pollutants

a. CO(GT) Time Series

- Observations:
    - A clear cyclic pattern was evident, with noticeable peaks and troughs that may correspond to daily activities.
    - The gradual variation in CO levels suggests a strong correlation with urban traffic patterns and possible industrial emissions.
- Implications: The consistent trend makes CO(GT) a strong candidate for time series forecasting. The periodic behavior observed encourages the inclusion of lag features and seasonality components in future modeling.

b. NOx(GT) Time Series
i) Observations:

- NOx levels show fluctuations similar to CO,suggesting the idea that both may share common emission sources, such as vehicular exhaust.
- Peaks observed at specific hours suggest a relationship with rush-hour traffic.

ii) Implications: This reinforces the hypothesis that urban traffic is significantly influenced by NOx concentrations.

c. Benzene (C6H6(GT)) Time Series
Observations:

- Benzene levels appear more variable with sporadic spikes, which may indicate episodic industrial discharges or other irregular emission sources.
- Compared to CO and NOx, benzene exhibits sharper drops and rises.

Implications: The high variability in benzene levels suggests that any forecasting model must be robust to abrupt changes, potentially by including additional exogenous features or using ensemble methods.

3.2.2 Daily & Hourly Graphs

1) Hourly Patterns
Method: The data was grouped by the hour of the day, and mean concentration was calculated.
Observations:

- Hourly averages of CO, NOx, and Benzene reveal distinct peaks, likely corresponding to rush hours.
- A common pattern across pollutants is a gradual increase in the morning, peaking around midday or early evening, followed by a decline at night.

Implications: These hourly patterns highlight the need for including time-of-day features in predictive models. This periodicity suggests that the inclusion of lag features will enhance forecasting accuracy.

2) Weekly Patterns
Method: Data was grouped by the day of the week, with the days from Monday through Sunday.
Observations:

- A bar plot of daily averages indicated that weekdays exhibit higher pollutant concentrations compared to weekends.
- The differences could be attributed to higher traffic volumes and industrial activity during weekdays.

Implications: Weekday/weekend data will help in forecasting models to capture the variance induced by weekly activity cycles.

3.2.3 Correlation analysis

Observations:

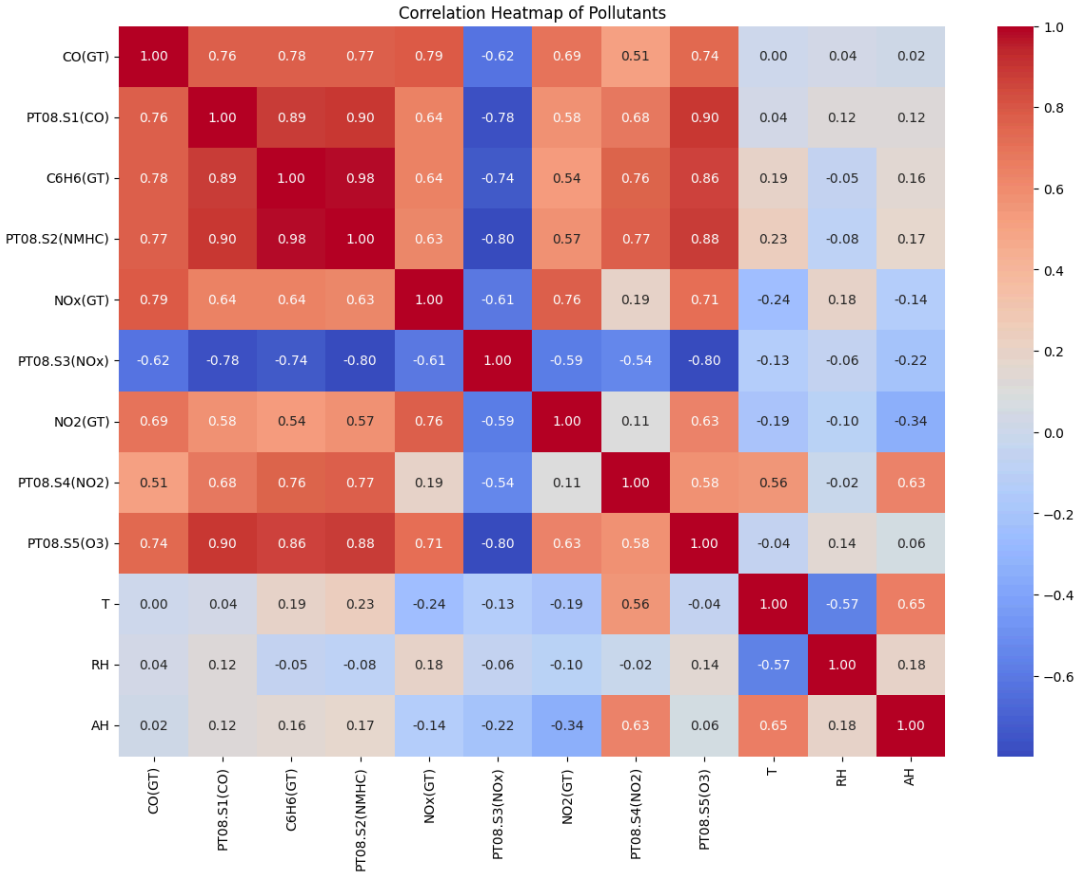
- Strong Positive Correlations:
    - CO(GT), NOx(GT), and C6H6(GT) show strong positive correlations, indicating that these pollutants likely share common sources such as vehicular emissions and industrial activities.
- Sensor vs. Ground-Truth:
    - Sensor readings (e.g., PT08.S1 for CO, PT08.S3 for NOx) closely follow their corresponding ground-truth measurements, confirming sensor reliability.
- Complex Sensor Interactions:
    - Some sensor signals exhibit weaker or even negative correlations with certain pollutants, suggesting differences in sensor response or pollutant dispersion mechanisms.
- Meteorological Influences:
    - While temperature shows a weak to moderate correlation with pollutant levels, humidity variables (RH & AH) have minimal direct impact, highlighting the dominance of emission sources over meteorological effects in this context.

Implications:

- Feature Reduction:
  The strong inter-pollutant correlations suggest that including all variables may lead to multicollinearity. Combining or selecting representative features can simplify the model.

- Data Imputation:
  The high correlation between specific pollutants indicates that lagged values of one pollutant could serve as proxies for another in cases of missing data.
- Modeling Strategy:
  The insights support the use of multivariate time series models that account for interactions among pollutants. Additionally, feature engineering (e.g., aggregating highly correlated variables) may enhance predictive performance.

Although we observed strong correlations among the main pollutants, the collinearity analysis identified redundancy specifically in certain sensor signals. This highlights that the critical pollutant measurements (CO, NOx, Benzene) were retained because they provide unique, valuable information. Some sensor signals, such as PT08.S2(NMHC) and PT08.S5(O3), were dropped due to high redundancy, simplifying the feature set for modeling.



3.3 Advanced graphs

3.3.1 Autocorrelation and Partial Autocorrelation Analysis

Method:

- ACF & PACF Generation:
  The autocorrelation function (ACF) and partial autocorrelation function (PACF) plots for CO(GT) were generated using `plot_acf` and `plot_pacf` (with 50 lags). These plots

help determine how past observations influence current values and assist in identifying significant lag terms.

Observations:

- ACF Plot:
  - A high correlation at lag 1 indicates that the current CO(GT) values are strongly dependent on the immediately preceding observation.
  - Periodic spikes, particularly around lag 24 (and its multiples), reveal a clear daily (diurnal) cycle, which is expected given the nature of urban traffic and routine human activities.
  - The gradual decay in correlation over subsequent lags suggests that while recent values are influential, the impact diminishes over time.
- PACF Plot:
  - The sharp drop after the first lag confirms that an AR(1) process is dominant.
  - The PACF also shows a significant spike around lag 24, reinforcing the importance of including seasonal components (daily patterns) in the model.
  - After these key lags, the partial correlations tend to drop closer to zero, indicating that most of the information is captured by the immediate lags and the seasonal cycle.

Implications:

- Lag Inclusion:
  The significant spikes at lags 1 and 24 support the inclusion of both an autoregressive (AR) term and seasonal lag terms in time series forecasting models.
- Model Parameterization:
  The insights from the ACF and PACF plots not only support the idea of including a 24-hour seasonal component but also inform the feature engineering process for Random Forest and LSTM models.

3.3.2 Time Series Decomposition

Method:

- Seasonal Decomposition:
  The CO(GT) time series was decomposed into its trend, seasonal, and residual components using an additive model (`seasonal_decompose`) with a period of 24. This method helps isolate the underlying patterns in the data.

Observations:

- Trend Component:
  - The trend component illustrates the long-term progression of CO(GT) levels. It may show subtle increases or decreases over the observation period, reflecting gradual changes in baseline air quality.
- Seasonal Component:
  - A pronounced seasonal pattern is evident, with recurring peaks and troughs corresponding to daily cycles. This seasonal component aligns with known patterns such as increased emissions during morning and evening rush hours.
- Residual Component:

    - The residuals (noise) appear relatively random after the trend and seasonal effects are removed, indicating that the decomposition effectively captured the main systematic patterns in the data.

Implications:

- Forecasting Guidance: The clear daily seasonality supports the inclusion of a 24-hour period in model specifications. Although SARIMA models explicitly incorporate seasonal differencing, they were not considered because their performance was ultimately suboptimal for capturing the non-linear dynamics of the dataset.
- Model Design Influence: The decomposition and ACF/PACF analyses provided crucial insights that directly informed the feature engineering strategy. The observed daily cycle led to the creation of lag and rolling statistics, which significantly improved the performance of both the Random Forest and LSTM models.
- Diagnostic Value: Even though SARIMA isn't deployed, the advanced analysis confirmed that seasonal and lagged effects are dominant in the CO(GT) time series, validating the design of the more flexible non-linear models.

4. Modelling approach & results

In this,the predictive models were developed to forecast CO concentrations using feature-engineered time-series data. Two models were implemented: a Random Forest regression model and a Long Short-Term Memory (LSTM) neural network model to capture non-linear patterns and temporal dependencies. Both models were evaluated using error metrics and integrated with Kafka for real-time predictions

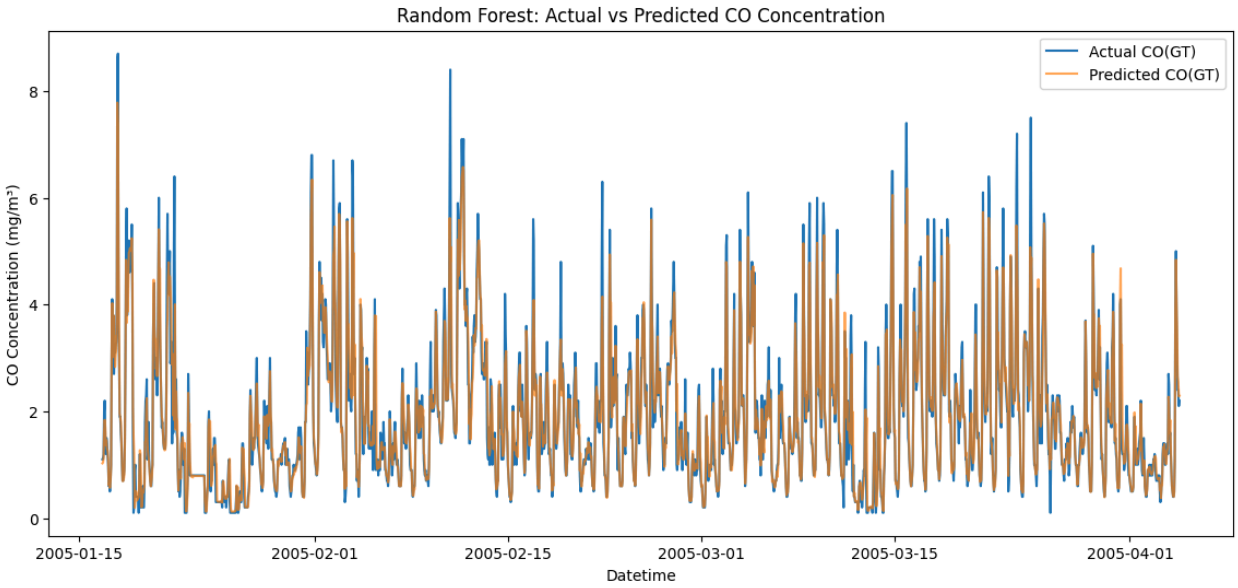
4.1 Model 1: Random Forest Model

Approach: The Random Forest model was developed using the engineered features extracted during the EDA and data cleaning phase. These features included Temporal Features like Hour, DayOfWeek, and Month, Lag Features: E.g., previous hour's CO value and Rolling Features by calculating 3-hour moving average and standard deviation.

Training and Evaluation: The dataset was divided into training and test sets using an 80/20 split. The model was trained on the training set to capture nonlinear interactions between variables. Evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) demonstrated robust performance (e.g., MAE ≈ 0.226, RMSE ≈ 0.400), indicating that the model effectively predicted CO levels.

Advantages: This model has the ability to handle complex relationships and interactions. There is no strict requirement for stationarity, which makes it flexible given the variability in sensor data.

Result: Random forest provided accurate predictions by leveraging multiple engineered features, and it delivered low error rates in the evaluation as seen in above image



4.2 Model 2: LSTM model

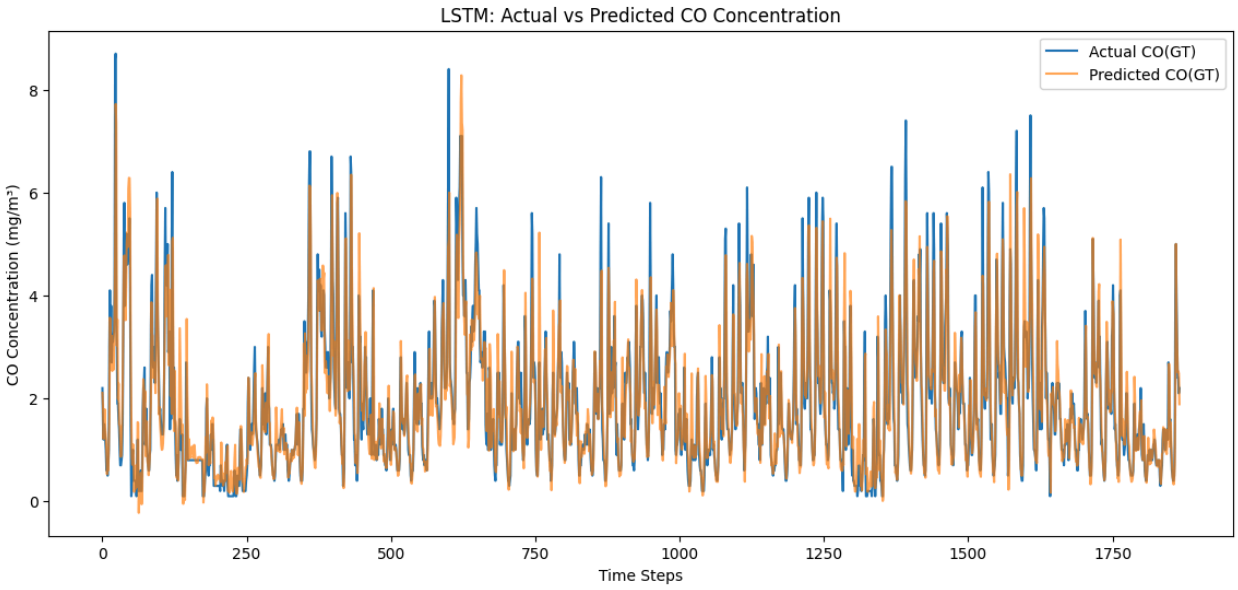
Approach: Given the sequential nature of the air quality data, an LSTM model was developed using Keras. The LSTM model takes advantage of its memory cells to capture long-term dependencies in the time series data.

Data Preparation: The CO time series was normalized using MinMaxScaler. After this, a sequence of 24 consecutive hourly CO values were constructed as inputs, where each sequence is used to predict the next value.

Training and Evaluation: The LSTM network was built with one LSTM layer containing 40 units followed by a Dense output layer. The model was trained on the normalized sequences, with the evaluation performed on a holdout test set. Error metrics showed that the LSTM model produced forecasts with competitive MAE and RMSE values compared to the Random Forest model.

Handling Missing Data in Real-Time: A rolling buffer approach is employed in the Kafka consumer. The buffer stores the most recent 24 values. When an incoming CO value is missing, the consumer uses the last prediction to fill the gap, ensuring that the buffer remains complete for continuous real-time prediction.

Result: LSTM exhibited strong performance in capturing sequential dependencies. Its rolling buffer mechanism for real-time deployment helped maintain prediction accuracy even in the presence of missing values as seen in the below image.



4.3 Real-time deployment

To put the predictive models into use, the trained models were saved for integration into a Kafka-based real-time prediction pipeline. The Random Forest model was saved using Python's `pickle` library (`rf_model.pkl`). The LSTM model was saved in keras format using Keras's `model.save('lstm_model.keras').`

Random Forest Consumer: This consumer extracts time-based and engineered features from each incoming JSON record from the Kafka topic. It then inputs these features into the Random Forest model to predict the CO concentration.

LSTM Consumer: This code maintains a rolling buffer of the most recent 24 CO values. If any record is missing the CO value, the consumer substitutes the last predicted value to ensure the buffer is full. The complete sequence is then reshaped and passed to the LSTM model for prediction.

The integration of these models within the Kafka consumer framework enables continuous, on-the-fly forecasting, which is critical for supporting timely air quality monitoring and decision-making in an operational environment

5. Limitations & Conclusion

In this project, we successfully developed a real-time air quality forecasting system that integrates Apache Kafka with both traditional machine learning and deep learning models. Our experiments showed that while the Random Forest model delivered robust performance using

engineered features, the LSTM model excelled in capturing temporal dependencies present in the sequential sensor data. The LSTM model's ability to learn from sequences of normalized CO values enabled it to produce accurate real-time predictions when deployed via a Kafka consumer with a rolling buffer approach.

However, there exist several limitations. The primary limitation of the LSTM approach is its dependency on having an initial sequence of valid values. If the system starts with null or missing values, the LSTM consumer's rolling buffer will not be fully populated, thereby delaying predictions until sufficient non-null values are received. This delay could be critical in applications requiring immediate responses. Additionally, while we addressed missing data by substituting previous predictions or the last buffered value, such imputation methods may inadvertently propagate errors if persistent data gaps occur.

Other limitations include:

- **Data Quality and Consistency:** Inconsistent sensor readings and improper scaling can impact model performance. Although normalization and robust preprocessing techniques mitigate these issues, variations in sensor calibration or sudden shifts in data quality may still challenge model reliability.

- **Model Generalizability:** Our models are trained on the UCI Air Quality Dataset; their effectiveness in different urban environments or with alternative sensor configurations remains to be validated. Thus, the deployment of these models in other settings may require further tuning and localized data collection.
- **Computational Resource Requirements:** Deep learning models such as the LSTM require significant computational resources for both training and real-time inference. This could limit their deployment in resource-constrained environments or necessitate the use of dedicated hardware accelerators.
- **Real-Time Integration Complexity:** While Kafka provides an effective framework for data streaming and real-time processing, ensuring robust error handling and system scalability in a live production environment will require additional engineering effort.

In summary, while our system provides a promising solution for real-time urban air quality forecasting, further work is needed to optimize model initialization and handling of missing data, enhance model generalizability, and ensure reliable deployment under varying operational conditions

Appendix

1. Kafka not running initially when server start command is used

A. Error - $ bin/windows/kafka-server-start.bat config/kraft/server.properties DEPRECATED: A Log4j 1.x configuration file has been detected, which is no longer recommended. To use a Log4j 2.x configuration, please see https://logging.apache.org/log4j/2.x/migrate-from-log4j1.html#Log4j2ConfigurationFormat for details about Log4j configuration file migration. You can also use the D:\Kafka\kafka_2.13-4.0.0/config/tool-log4j2.yaml file as a starting point. Make sure to remove the Log4j 1.x configuration after completing the migration. [2025-03-21 17:53:13,913] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$) [2025-03-21 17:53:14,145] ERROR Exiting Kafka due to fatal exception (kafka.Kafka$) java.lang.RuntimeException: No readable meta.properties files found. at org.apache.kafka.metadata.properties.MetaPropertiesEnsemble.verify(MetaPropertiesEnsemble.java:480) ~[kafka-metadata-4.0.0.jar:?] at kafka.server.KafkaRaftServer$.initializeLogDirs(KafkaRaftServer.scala:141) ~[kafka_2.13-4.0.0.jar:?] at kafka.server.KafkaRaftServer.<init>(KafkaRaftServer.scala:56) ~[kafka_2.13-4.0.0.jar:?] at kafka.Kafka$.buildServer(Kafka.scala:68) ~[kafka_2.13-4.0.0.jar:?] at kafka.Kafka$.main(Kafka.scala:75) [kafka_2.13-4.0.0.jar:?] at kafka.Kafka.main(Kafka.scala) [kafka_2.13-4.0.0.jar:?]

B. Solution : Completely reset Kafka installation by deleting the contents of the log directory and reformatting it, follow these steps:
    a. First, stop your Kafka server if it's running:
    bin/kafka-server-stop.sh
    b. Delete the contents of your log directory. Based on your error messages, your log directory is /tmp/kraft-combined-logs:
    rm -rf /tmp/kraft-combined-logs/*
    c. Format the storage directory with a new cluster ID:
    bin/kafka-storage.sh format --config config/kraft/server.properties --cluster-id $(bin/kafka-storage.sh random-uuid) --standalone
    d. Start Kafka again:
    bin/kafka-server-start.sh config/kraft/server.properties

C. Reference : Chat gpt was used to understand the error and figure out the solution. The error was copied and pasted and the prompt was “Help me figure out the exact problem and recommend steps to solve it”. And the response given by Chatgpt is pasted in B.

2. Pip install python error

A. Error : Error in installing pip install kafka-python

B. Solution :

a. Step 1: Ensure Python 3 is Installed in WSL
   Check that Python 3 is installed correctly in your WSL environment:
   `python3 --version`
   If Python 3 is not installed, you can install it along with the necessary tools for creating a virtual environment:
   `sudo apt update`
   `sudo apt install python3 python3-venv python3-pip`
b. Step 2: Create a Virtual Environment in WSL
   Once Python 3 is installed, you can create a virtual environment:
   `python3 -m venv ~/myenv`
   Here, `~/myenv` is the location where the virtual environment will be created in your home directory. You can use any directory you prefer.
c. Step 3: Activate the Virtual Environment
   Activate the virtual environment:
   `source ~/myenv/bin/activate`
   You should see the environment's name (e.g., `(myenv)`) in your terminal prompt.
d. Step 4: Install `kafka-python` in the Virtual Environment
   Now that the virtual environment is active, you can install `kafka-python`:
   `pip install kafka-python`
e. Step 5: Install Required Dependencies (If Necessary)
   If you're still encountering errors, there might be some missing dependencies that are required to install packages. You can install the following packages:
   `sudo apt install python3-dev build-essential`

C. Reference : Chat gpt was used to understand the error and figure out the solution. The error was copied and pasted and the prompt was “Help me figure out the exact problem and recommend steps to solve it”. And the response given by Chatgpt is pasted in B