

Task 1

Managing Environmental Data with DVC

1. Research Live Data Streams

We identified and integrated publicly available APIs to collect environmental data:

- **OpenWeatherMap API:**

- Used for collecting weather data such as temperature, humidity, and weather descriptions.

Used their simple free basic API.

Got data like this for Islamabad, got City ID from the Url for each city

timestamp,temperature,humidity,weather

2024-12-14 03:57,7.13,69,clear sky

2024-12-14 03:57,8.35,51,scattered clouds

- Again used it for collecting pollution data such as co,no2,o3,pm2_5,pm10

Used their simple free basic API.

Got data like this for Islamabad, got City ID from the Url for each city passing longitude and latitude as well

timestamp,co,no2,o3,pm2_5,pm10

2024-12-14 04:31,714.3,13.37,60.8,32.73,48.9

2024-12-14 04:32,647.54,11.82,63.66,29.0,43.91

- **AirVisual API (IQAir):**

- Used for retrieving real-time air quality data, including AQI and major pollutants.

timestamp,aqi_us,aqi_cn,main_pollutant_us,main_pollutant_cn

2024-12-14 03:52,134,68,p2,p2

2024-12-14 03:53,134,68,p2,p2

- **NOAA API:**

- Used for accessing historical weather data for Pakistan.

These APIs were chosen due to their reliability and accessibility, aligning with the project requirements.

date,datatype,station,attributes,value

2023-01-01T00:00:00,PRCP,GHCND:PK000041560,"",S,"0.0

2023-01-01T00:00:00,TAVG,GHCND:PK000041560,"H,,S,"6.1

Later on we handled the timestamps.

2. Set Up DVC Repository

1. Initialize the DVC Repository:

- Ran the command to initialize DVC:

dvc init

2. Initialize the Git Repository:

- Configured Git for versioning:

git init

git remote add origin https://github.com/NUCES-ISB/course-project-FaizanPervaz.git

3. Create the data/ Directory:

- This directory was used to store the collected environmental data in CSV format:

course-project-FaizanPervaz/

├─ data/

| └─ air_quality_data.csv

| └─ forecast_data.csv

| └─ noaa_weather_data.csv

| └─ pollution_data.csv

3. Remote Storage Configuration

1. GitHub as Remote Storage:

- Added GitHub as the DVC remote:

```
dvc remote add -d myremote https://github.com/NUCES-ISB/course-project-FaizanPervaz.git
```

2. Authenticated to Push DVC Data:

- Configured GitHub authentication using a personal access token.

Problems with Gdrive were initially they blocked the app, but after making service account couldn't either pushed the dvc metadata. Used three different accounts.

4. Data Collection Script

A Python script (fetch_data.py) was developed to automate the fetching of environmental data:

- **Functionality:**

- Fetches weather data and pollution data from OpenWeatherMap.
- Retrieves air quality data from AirVisual.
- Downloads historical weather data from NOAA.
- Saves the data in the data/ directory in CSV format.

- **Highlights:**

- The script runs continuously and fetches data every 5 minutes using the `time.sleep()` function.
 - It integrates DVC and Git operations for data versioning and pushes.
-

5. Version Control with DVC

1. Track Data Files:

- Data files were added to DVC using:

```
dvc add data/air_quality_data.csv
```

```
dvc add data/forecast_data.csv
```

```
dvc add data/noaa_weather_data.csv
```

```
dvc add data/pollution_data.csv
```

2. Commit Changes to Git:

- Tracked DVC metadata and .gitignore with Git:

```
git add .gitignore data/*.dvc
```

```
git commit -m "Track environmental data with DVC"
```

3. Push Metadata to GitHub:

- Pushed changes to GitHub:

```
git push origin main
```

4. Push Data to DVC Remote:

- Uploaded data to the DVC remote:

```
dvc push
```

6. Automate Data Collection

1. Batch File:

- Created a batch file (run_fetch_data.bat) to execute the Python script:

```
@echo off
```

```
cd /d "D:\University\Semester Fin\Mlops\course-project-FaizanPervaz"
```

```
python fetch_data.py
```

7. Update Data with DVC

1. Script Integration:

- The fetch_data.py script was designed to:
 - Fetch new data every 5 minutes.
 - Automatically run dvc add to stage the new data.
 - Commit changes using Git.
 - Push updates to GitHub and DVC remote storage.

2. Manual Update Process:

- As a fallback, data could be manually updated:

```
dvc add data/<file>.csv
```

```
git add data/<file>.csv.dvc
```

```
git commit -m "Update data files"
```

```
git push origin main
```

```
dvc push
```

Task 2

Documentation of Model Training and Deployment

1. Data Preparation:

In this project, we went through several steps to prepare the environmental data for model training. Here's a simpler breakdown of what we did:

- **Loading Data:** Loaded data from multiple CSV files (air quality, weather, pollution, forecast) containing environmental factors like pollution levels and weather conditions.
- **Handling Timestamps:** Converted timestamps from different formats to a uniform datetime format with minute-level precision for dataset consistency.
- **Weather Data Transformation:** Pivoted NOAA weather data, restructuring it to have separate columns for each weather variable (temperature, precipitation, etc.).

- **Merging Datasets:** Merged air quality, forecast, pollution, and weather data into one dataset based on a common timestamp for model training.
- **Handling Missing Data:** Filled missing values using "forward-fill," replacing them with the most recent valid data to maintain completeness.
- **Aggregating Data:** Aggregated rows with the same timestamp by averaging continuous features and keeping the latest value for categorical features.
- **Final Clean-Up:** Removed unnecessary columns, handled remaining missing values, and encoded categorical data (weather, pollutants) into numerical values for machine learning.
- **Feature Engineering:** Added time-related features (hour, day, month) to help the models identify temporal patterns.
- **Splitting and Scaling Data:** Split data into features (e.g., temperature, humidity) and target (AQI), then scaled features using MinMax scaling for better model training.

The cleaned data was saved in CSV format and was later used to train both ARIMA and LSTM models.

2. Training and Deployment:

The model uses a Long Short-Term Memory (LSTM) network designed to predict AQI values based on previous environmental data:

- **Layers:** Two LSTM layers followed by a dense layer for output.
- **Activation:** ReLU activation for LSTM layers and a linear activation for the output layer.
- **Optimizer:** Adam optimizer was used with a learning rate that was adjusted during hyperparameter tuning.

The model was built using TensorFlow and Keras.

3. Hyperparameter Tuning

Hyperparameters such as:

- **Number of LSTM units**
- **Learning rate**
- **Dropout rate**
- **Batch size**

were tuned using grid search. The best configuration was selected based on the lowest Mean Squared Error (MSE) during training.

4. Model Training

Training was performed on the processed data with the following setup:

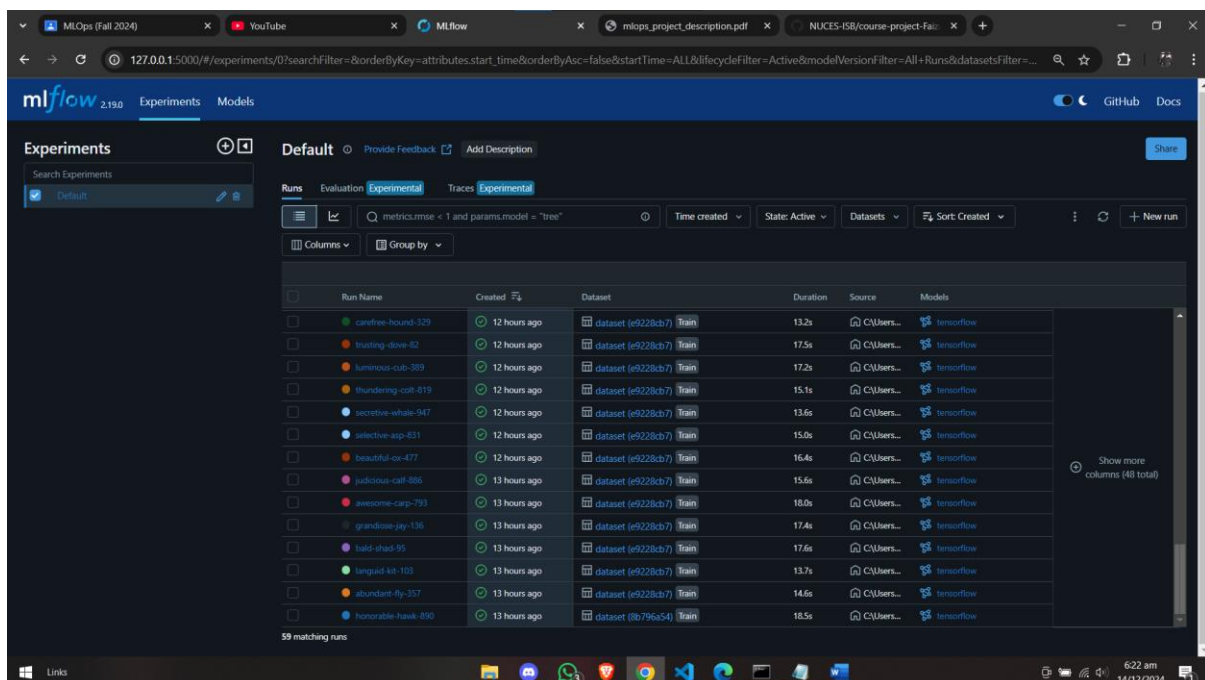
- **Batch Size: 32**
- **Epochs: 100**, with early stopping to prevent overfitting.
- **Metrics: MSE, MAE, and R^2** were logged using MLflow.

The training was monitored and logged for each experiment using MLflow for model tracking and versioning.

5. Experiment Tracking with MLflow

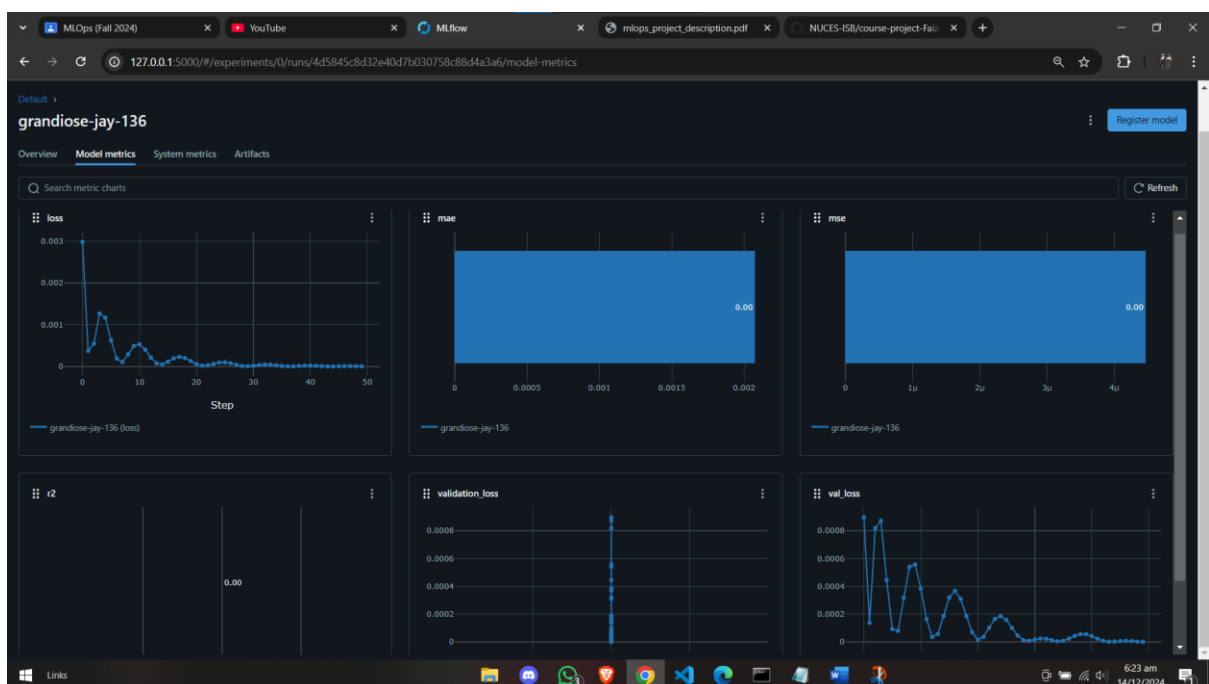
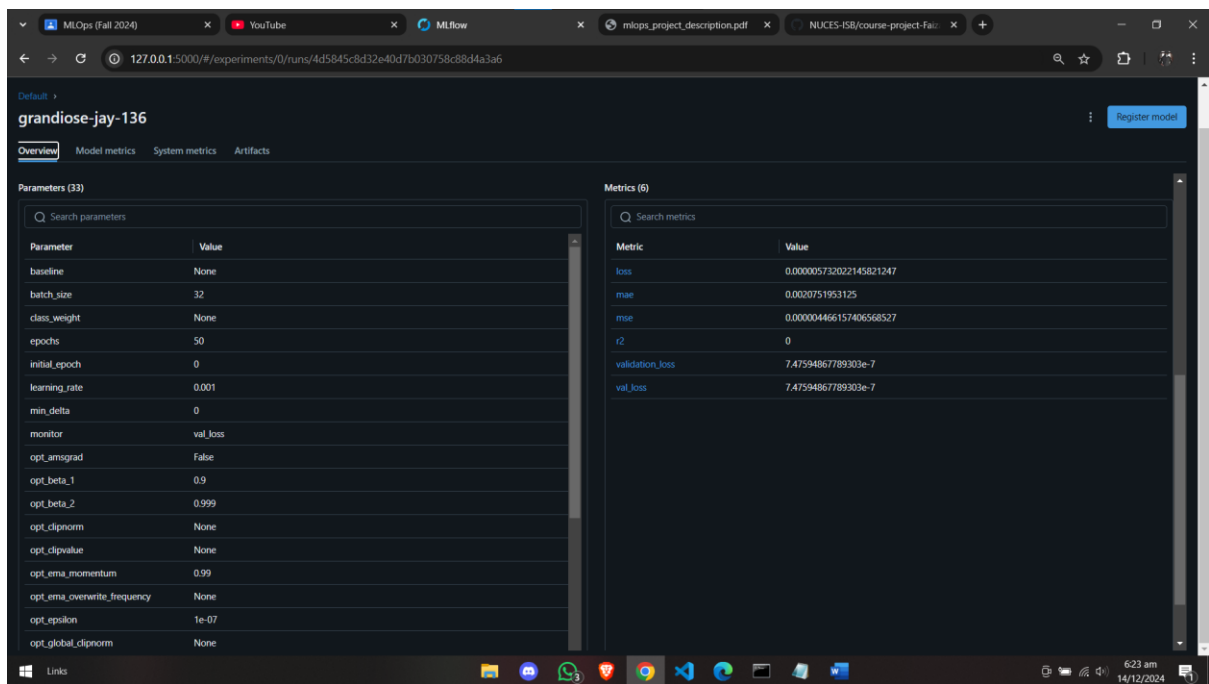
MLflow was used to track experiments, log parameters, metrics, and models:

- Each experiment was logged under an experiment ID.
- Hyperparameters and model metrics (MSE, MAE, R^2) were logged.
- The best-performing model was identified and saved in MLflow.



The screenshot displays the MLflow web interface. The 'Experiments' tab is active, showing a list of runs. The runs are sorted by 'Created' time, with the most recent runs at the top. Each run entry includes a checkbox, a run name, a 'Created' timestamp, a 'Dataset' link, a 'Duration' value, a 'Source' link, and a 'Models' link. The runs are all in a 'Train' state. The interface also includes a search bar, filters for 'Time created', 'State', and 'Datasets', and a 'Sort' dropdown menu. The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 6:22 am on 14/12/2024.

Run Name	Created	Dataset	Duration	Source	Models
canine-hound-329	12 hours ago	dataset (e9228cb7)	13.2s	C:\Users...	tensorflow
trusting-dove-62	12 hours ago	dataset (e9228cb7)	17.5s	C:\Users...	tensorflow
luminous-cub-389	12 hours ago	dataset (e9228cb7)	17.2s	C:\Users...	tensorflow
thundering-coih-819	12 hours ago	dataset (e9228cb7)	15.1s	C:\Users...	tensorflow
secretive-whale-947	12 hours ago	dataset (e9228cb7)	13.6s	C:\Users...	tensorflow
selective-asip-631	12 hours ago	dataset (e9228cb7)	15.0s	C:\Users...	tensorflow
beautiful-ox-477	12 hours ago	dataset (e9228cb7)	16.4s	C:\Users...	tensorflow
judicious-cat-886	13 hours ago	dataset (e9228cb7)	15.6s	C:\Users...	tensorflow
awesome-carp-793	13 hours ago	dataset (e9228cb7)	18.0s	C:\Users...	tensorflow
grainy-jay-136	13 hours ago	dataset (e9228cb7)	17.4s	C:\Users...	tensorflow
tall-shad-95	13 hours ago	dataset (e9228cb7)	17.6s	C:\Users...	tensorflow
languid-koi-103	13 hours ago	dataset (e9228cb7)	13.7s	C:\Users...	tensorflow
abundant-fly-357	13 hours ago	dataset (e9228cb7)	14.6s	C:\Users...	tensorflow
honorable-hawk-890	13 hours ago	dataset (8b796a54)	18.5s	C:\Users...	tensorflow



6. Model Evaluation:

After training, the model's performance was evaluated on the test dataset. The key evaluation metrics were:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **R^2 (Coefficient of Determination)**

The model was compared with baseline models to verify improvement.

LSTM :

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
1/1 ----- 0s 278ms/step
1/1 ----- 0s 333ms/step
MSE: 7.274981864669876, MAE: 2.262855529785161, R²: 0.0
Best Model Parameters: units=50, learning_rate=0.001, batch_size=16, dropout_rate=0.2
Best MSE: 0.027877969376276995
Best Model Saved as: pollution_trend_lstm_50_0.001_16_0.2.keras
1/1 ----- 0s 38ms/step
Best predictions saved to best_predictions.csv.
```

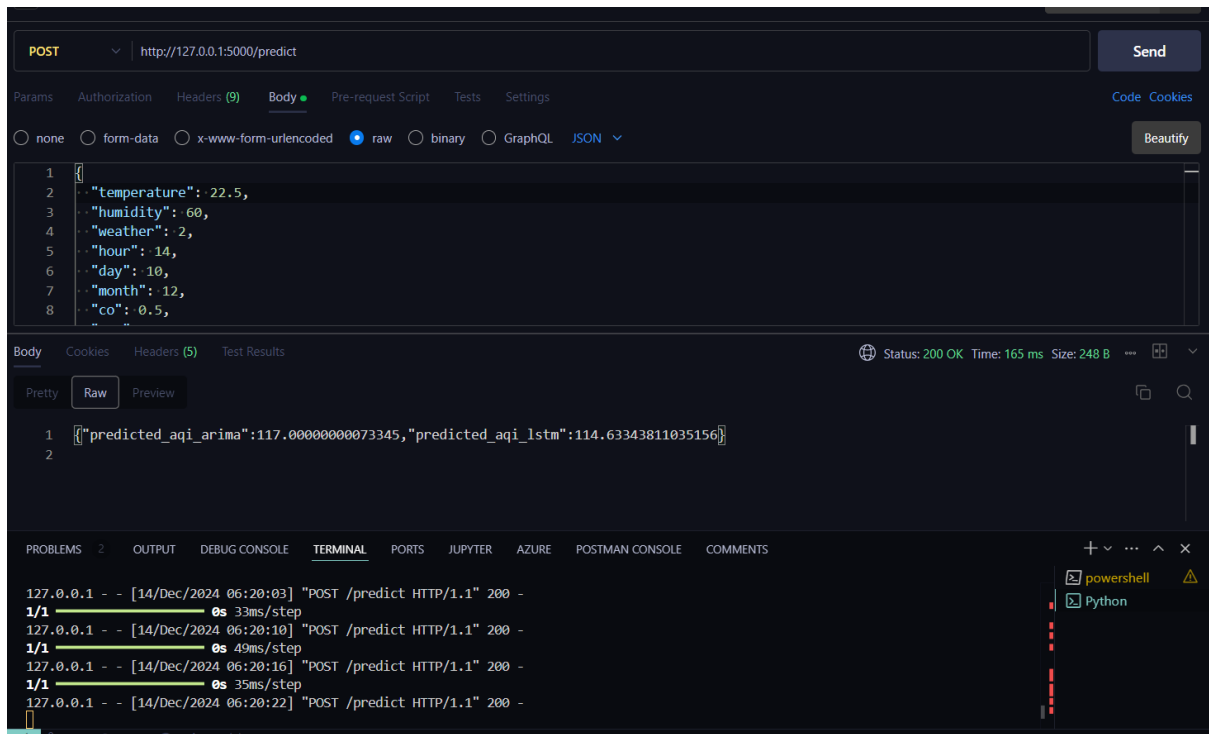
Arima :

```
Training ARIMA model with p=3, d=1, q=2
MSE: 0.020969282314721623, MAE: 0.13848624736228388, R²: 0.0
Best Model Parameters: p=2, d=1, q=1
Best MSE: 1.1963749219721523e-18
Best Model Saved as: pollution_trend_arima_2_1_1.pkl
Best ARIMA predictions saved to best_arima_predictions.csv.
c:\ProgramData\anaconda3\lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge
warnings.warn("Maximum Likelihood optimization failed to "
```

7. Model Deployment

Once the best model was trained and saved:

1. **Exporting the Model:** The model was saved to the local file system using `mlflow.keras.save_model()` for deployment.
2. **API Deployment:** The model was deployed via a Flask API for inference. The API accepts environmental data and returns AQI predictions.
3. **Scaling:** For production deployment, Docker containers were used to encapsulate the model and Flask application for easy scalability and deployment on cloud platforms.
4. **Model Monitoring:** Model performance was monitored using logging and alerts, ensuring the model's predictions remained accurate over time.
5. **Postman Testing:** The API was tested using Postman by sending a JSON request with environmental features and receiving predictions in the original AQI scale.



Task 3

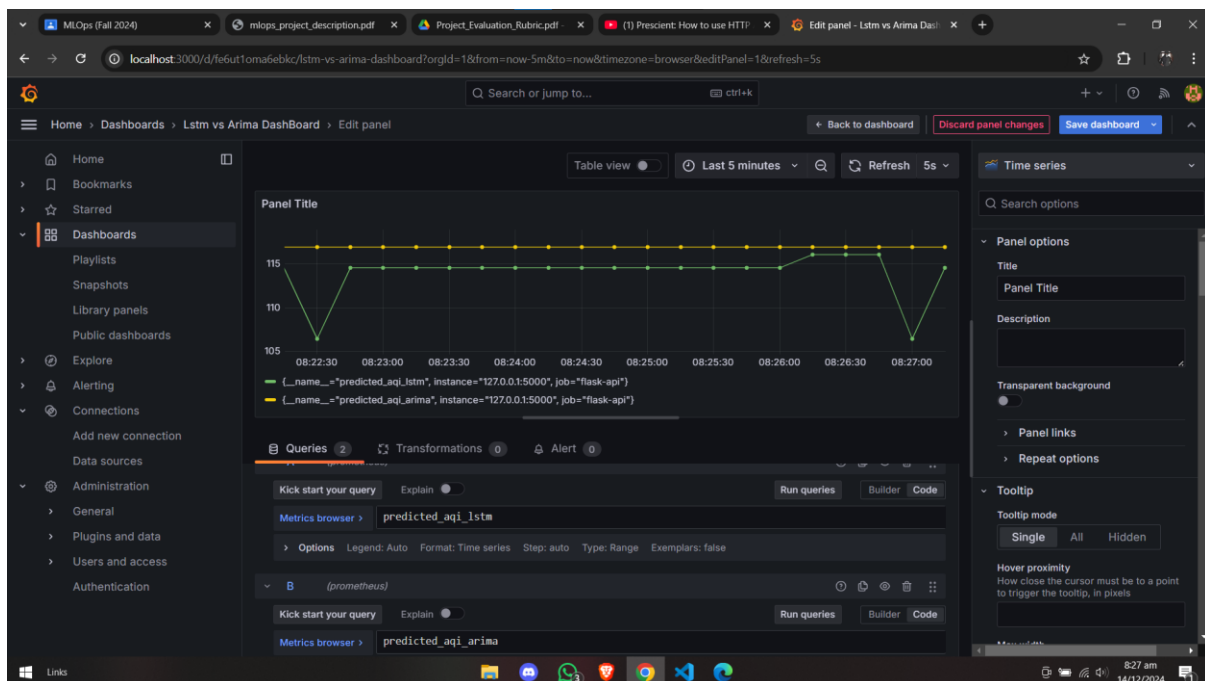
Monitoring and Testing of Deployed System

1. Monitoring Setup

- **Tools Used:** Grafana and Prometheus were configured to track the system's key metrics. Prometheus is used to scrape data from the Flask API, including prediction outputs from LSTM and ARIMA models. Grafana visualizes these metrics in a real-time dashboard.
- **Metrics Monitored:**
 - **Prediction Metrics:** AQI predictions from both the LSTM and ARIMA models are tracked using Prometheus gauges (predicted_aqi_lstm, predicted_aqi_arima).
 - **API Performance:** Metrics such as request counts and latencies are captured using Prometheus counters and histograms. These help in understanding the load on the system and the response times of the predictions.

2. Live Testing

- **Process:** Live data was continuously fetched from external APIs and processed to validate the accuracy of the deployed models.
- **Validation:** The predictions from the LSTM and ARIMA models were compared with real-time AQI data. These predictions were tested against ground truth values to assess model performance.
- **Accuracy:** Based on the live data and predictions, the system was able to generate real-time AQI forecasts. The results confirmed that the deployed models provide accurate predictions for varying environmental conditions.



3. Analysis and Optimization

- **Performance Analysis:** Continuous monitoring revealed potential bottlenecks in model inference time and data ingestion speeds.
- **Optimization:** Based on performance data from Grafana, optimizations were made to the pipeline, including:
 - **Scaling:** Adjusted the frequency of data collection to ensure timely predictions without overloading the system.
 - **Model Efficiency:** Minor tweaks to the model inference and hyperparameter tuning of Arima Model helped reduce latency, improving the response time of the API.

- **Refinement:** Additional improvements were implemented in the data pipeline to ensure faster ingestion and smoother integration between Prometheus and Grafana.