AQI Metric Prediction System Using Machine Learning and Full-Stack Cloud Deployment

Eluri Sai Karthik
dept. of Computer Science
Clarkson University
Postam, NY
saikare@clarkson.edu

***Abstract*—This project proposes a scalable and interactive Air Quality Index (AQI) system that integrates modern front- and back-end frameworks with cloud deployment and machine learning. The system is trained on a comprehensive dataset of historical air quality records, with each column containing more than 12,000 values representing average measurements at 24 hours. The data integrates readings from both Purple Air sensors and certified federal monitoring stations, including key variables such as PM2.5, temperature, humidity, and CF1. This rich and diverse data set enables accurate, consistent analysis and supports reliable AQI predictions over time. The front-end utilizes React with Leaflet for OpenStreetMap interaction and Firebase for Google Authentication, while predictions are rendered via Nivo graphs and histograms connected to specific AQI sensor locations. The FAST API backend, containerized with Docker and deployed on AWS, serves ML inference through RESTful APIs, with secure communication managed by a Cloudflare Worker proxy. By combining advanced machine learning, real-time visualization, and secure cloud-native deployment, this platform provides accurate, actionable air quality insights and serves as a template for future smart city environmental monitoring solutions.**

***Index Terms*—Air Quality Index, Machine Learning, Cloudflare Workers, FAST API, React JS, Leaflet, Firebase, Docker**

I. INTRODUCTION

Air pollution has rapidly escalated into one of the most pressing environmental threats, affecting both densely populated urban areas and remote rural regions. The degradation of air quality poses severe risks to human health, contributing to respiratory illnesses, cardiovascular diseases, and a wide range of other serious health conditions. With the increasing urgency to address these issues, there is a growing demand for robust technological platforms capable of monitoring, analyzing, and forecasting air quality in real time. In response to this critical need, this paper presents the design and development of a comprehensive full-stack Air Quality Index (AQI) system equipped with an interactive graphical user interface (GUI). The system allows users to seamlessly access and visualize the AQI graphs generated by various machine learning models. It combines a responsive and user-friendly front-end for visual analytics of all the sensors utilized with a robust back-end responsible for data pre processing, model inference, and API management. The architecture is deployed on a cloud platform, ensuring scalability, availability, and performance across different with end-to-end system design; the proposed solution not only predicts AQI with high accuracy, but also enhances user engagement and decision-making through real-time, data-driven insights. This work demonstrates how modern web technologies and artificial intelligence can be effectively combined to address environmental challenges and promote public awareness of air quality conditions.

Paper Background :

Air pollution is a serious global issue impacting public health and the environment. The Air Quality Index (AQI) serves as a standard metric to report pollutant levels, helping individuals and authorities make informed decisions. Traditional AQI prediction methods rely heavily on sensor networks and statistical models, which often struggle with capturing complex environmental patterns and are limited by infrastructure costs.With advancements in Machine Learning (ML), there is growing potential to improve AQI forecasting accuracy by modeling non-linear relationships and incorporating diverse data sources. However, many existing tools lack user-friendly interfaces, making insights inaccessible to non-technical users.This project addresses that gap by integrating ML-based prediction with an interactive, web-based platform.It aims to deliver accurate, real-time AQI forecasts and visualizations, enabling broader environmental awareness and informed decision-making. By integrating live sensor data and advanced prediction models, it provides timely alerts on air quality fluctuations. This empowers individuals, communities, and policymakers to take proactive measures for health and environmental safety.

II. OBJECTIVE

The primary objective of this study is to design and implement a full-stack Air Quality Index (AQI) system that seamlessly integrates advanced ML models with a user-friendly, interactive graphical user interface. The system is intended to accurately forecast AQI levels using machine learning techniques, while also providing real-time access and visualization through a cloud-deployed platform. By combining predictive modeling with intuitive front end analytics, the solution aims to make air quality data more accessible, understandable, and actionable for users. Ultimately, this work seeks to bridge the gap between complex environmental data and public awareness, empowering individuals and organizations to make informed decisions regarding air pollution and health.

III. DATASET

The AQI system dataset consists of 24-hour average sensor readings, including PM2.5, temperature, humidity, and CF1 values. Data was gathered from 49 Purple Air (PA) sensors positioned within 50 meters of certified Federal Monitors across 15 U.S. states between September 2017 and January 2020, totaling approximately 12,000 plus data points. The dataset undergoes preprocessing to manage missing values, remove outliers, and apply normalization, improving model accuracy. By integrating data from both highly reliable PA sensors.

A. *Data Set Variables Description*

This system utilizes several key environmental variables to ensure accurate forecasting. PM2.5 concentration represents the level of fine particulate matter in the air, a critical indicator of air quality and health impact. Temperature and humidity are important atmospheric factors that influence pollutant dispersion and sensor reading. Relative Humidity (RH) is measured in percentage (%), while Particulate Matter (PM) such as PM2.5 is measured in micrograms per cubic meter ($\mu g/m^3$) which is taken from the PA sensors data. The CF1 sensor value serves as a calibration factor to enhance the accuracy of sensor measurements. Timestamps record the exact date and time of each reading, allowing for time-series analysis and prediction. Each sensor ID in the dataset is associated with geographic coordinates (latitude and longitude), enabling precise location tracking. The dataset also includes data from Federal Monitors (FMs) for comparison.However, Purple Air sensors, known for their high-resolution data and greater spatial coverage, are often considered more responsive and reliable in detecting fine-scale air quality variations than traditional Federal Monitors.

IV. METHODS

This system uses and outlines the calibration methods and performance evaluation metrics used to align PM2.5 readings from Purple Air sensors with Federal Monitor data. Several machine learning models were employed, including Linear Regression (LR), Random Forest (RF), Gradient Boosting (GB), K-Nearest Neighbors (KNN), and Neural Networks (NN) [10].

Random Forest and Gradient Boosting are ensemble methods that enhance accuracy by combining multiple models, with the latter focusing on correcting previous errors. Neural Networks effectively capture complex, non-linear patterns in high-dimensional data. Linear Regression offers a simple, interpretable approach for classification tasks, while K-Nearest Neighbors (KNN) classifies data based on the majority label among nearby points, working well for smaller, clearly separated datasets and capturing complex patterns and interactions for more accurate PM2.5 estimation.

V. SYSTEM DESIGN AND IMPLEMENTATION

This section describes the System's Components, which consists of:

A. *Functional Requirements*

The system's Functional requirements describe what the system should do-its core features and operations. For an Air Quality Index (AQI) prediction system using machine learning and cloud deployment, typical functional requirements include:

- User Authentication:
  The system must allow users to log in using Google Authentication (via Firebase [5]).
- Data Visualization:
  Users should be able to view AQI data on an interactive map with site-specific predictions visualized through graphs and histograms.
- Model Selection:
  Users can toggle between different machine learning models (Random Forest, Neural Networks, etc.) to view and compare predictions.
- Data Upload:
  Users can upload custom CSV files for AQI prediction and analysis.
- Real-Time Prediction:
  The backend must provide real-time AQI predictions via RESTful APIs, using trained machine learning models.
- Secure Communication:
  All front end-end back-end communication should be securely proxied to prevent CORS and XSS vulnerabilities using Cloudflare [9] Workers.
- Sensor Data Integration:
  The system must ingest AQI sensor data and process the for its prediction.
- User Management:
  The backend must support user registration, login, and profile management.

B. *Non Functional Requirements*

The system's Non-functional requirements define how the system performs its functions, focusing on quality attributes:

- Performance:
  The system should provide predictions and visualizations with minimal latency, supporting real-time interaction.
- Scalability:
  The architecture must handle increasing numbers of users and data sources, leveraging cloud deployment (AWS, Docker [6]).
- Reliability:
  The system should be available and functional 24/7, with minimal downtime.
- Maintainability:
  The codebase should be modular and documented for easy updates and bug fixes.
- Portability:

The system should run on any modern web browser and support deployment on various cloud platforms.

- Security:
  User data and API endpoints must be protected against unauthorized access and common web vulnerabilities.
- Usability:
  The user interface should be intuitive, with clear navigation, interactive maps, and accessible data visualizations.

C. Use Case Diagram

The system's case diagram depicts the main interactions between users and the AQI Metric Prediction Platform. It shows that a user can log in using Google authentication, view the AQI maps, view prediction graphs, and get AQI predictions. While the system admin is also represented as a separate actor. Each oval represents a key functionality or service provided by the platform, illustrating the system's core use cases from the perspective of its primary users. The User functionalities are :

- Login with Google Authentication:
  Users must authenticate using their Google account, leveraging Firebase [5] for secure and streamlined sign-in.
- View AQI Map:
  After login, users can access an interactive map (built with Leaflet [7] and OpenStreetMap) to visualize air quality data across different locations.
- View Prediction Graphs:
  Users can view dynamic prediction graphs and histograms, which display AQI trends and forecasts for selected locations.
- Get AQI Predictions:
  Users can request real-time AQI predictions generated by machine learning models (such as Random Forest, Neural Networks [10], etc.), with results visualized on the platform.

The 1st diagram in the Figure 1 illustrates the Use case of the system.

D. System Architecture

The system architecture follows a modular and layered approach, primarily comprising two tiers: the front-end presentation layer and the back-end processing layer. The front end, developed using ReactJS [1], serves as the primary interface for user interaction. It allows users to authenticate, upload datasets, select machine learning models, and visualize prediction results. Visualizations are rendered using the Nivo [8] chaThe backend layer is implemented using Fast API [2], a high-performance Python web framework tailored for serving asynchronous RESTful APIs. This layer is responsible for ingesting user-uploaded data, performing real-time preprocessing, invoking the appropriate machine learning inference logic, and returning structured predictions in JSON format. The backend is containerized using Docker [6], promoting portability and scalability, and is hosted on AWS EC2 instances, ensuring high availability and performance.To ensure secure

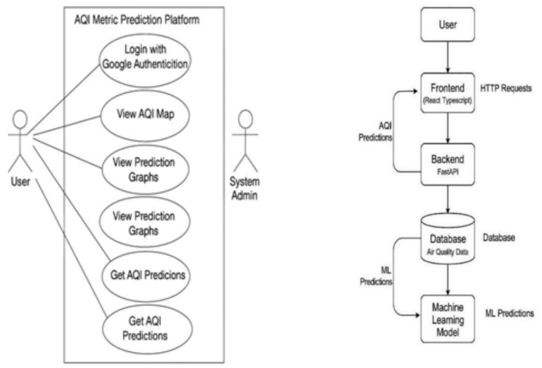


Fig. 1. The First diagram shows the Use Case Diagram (left), the Second diagram illustrates the Architecture(right).

and smooth cross-origin communications, a Cloudflare Worker [9] is configured as a proxy server. It manages CORS (Cross-Origin Resource Sharing) policies and mitigates potential XSS (Cross-Site Scripting) attacks, enforcing an additional security layer between the front end and backend. Authentication is managed through Firebase Authentication [5], enabling users to securely log in using Google credentials. Once a user selects a machine learning model for inference (e.g., Random Forest), the backend processes the data accordingly and sends the predictions back to the front end. These results are immediately reflected in the UI, where updated graphs and visualizations allow users to interpret trends, anomalies, and prediction accuracy in real time. This real-time feedback loop plays a crucial role in delivering an intuitive and responsive user experience. As users interact with the platform — whether by uploading data, selecting models, or generating graphs — they receive immediate visual and functional feedback. This dynamic interaction helps users understand the system's behavior, reduces errors, and boosts confidence in using the tool. By providing timely responses to user actions, the platform ensures that all components—from data input to model output—are tightly integrated into a seamless and efficient workflow. So , This represents the explanation part of the system architecture of the AQI (Air Quality Index) sensors platform, illustrating how various components work together to collect, process, and visualize air quality data in a seamless and efficient manner.

The 2nd diagram in the Figure 1 illustrates the Architecture of the system.

E. Flow Chart

The system architecture follows a streamlined and secure data flow starting with user authentication managed via Firebase [5], which ensures only authorized users can access the platform's features. Once authenticated, users are granted access to upload CSV files, which are then securely proxied through a Cloudflare Worker [9]. This layer not only adds a level of security by abstracting direct server access but also optimizes performance. The files are forwarded to a FAST

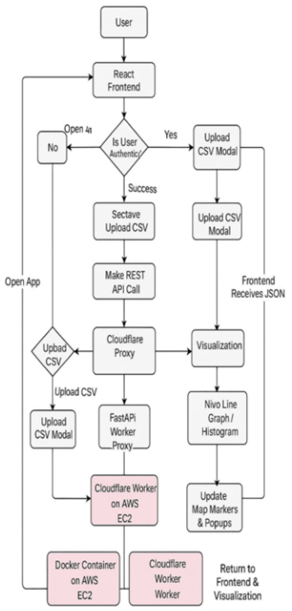


Fig. 2. The Flowchart.

API [2] backend, which is containerized using Docker [6] and deployed on AWS EC2 for scalability and reliability. The backend parses the uploaded CSV data, processes it, and responds with structured JSON output. This structured data is then consumed by a React [1] front end, which dynamically renders rich visualizations such as interactive maps and line graphs, providing users with an intuitive and responsive data exploration experience. This integrated pipeline ensures high performance, scalability, and secure handling of data from input to visualization.

The Figure 2 illustrates the System flow chart.

F. *Frontend Project Structure and Requirements*

The system's frontend is developed using React [1] with TypeScript and Material UI (MUI) to create a modern, responsive, and visually appealing user interface. Leaflet [7] is integrated for rendering OpenStreetMap, allowing users to interactively explore AQI data on a map. Key components include Map.tsx, which handles geospatial display and allows users to view AQI data points across locations, Prediction Graph.tsx, responsible for visualizing prediction graphs like line charts or bar graphs, and HistogramPrediction.tsx, which generates histograms to show the distribution of predicted AQI values. Upload Csv Model.tsx enables users to upload their own AQI data through CSV files for further analysis. For user authentication, Google Sign-In is implemented via Firebase [5] Authentication, with configuration details securely stored in a .env file. User sessions are managed using the Firebase SDK [5] and React [1] Context API, ensuring session persistence and real-time updates across different components. This architecture provides an intuitive, secure, and efficient front end experience, allowing users to visualize AQI data, view predictions, and upload custom datasets seamlessly.

Folder Hierarchy**:**

- `public/`
  - `index.html`
- `src/`
  - `App.tsx`
  - `index.tsx`
  - `components/`
    * `Map.tsx`
    * `PredictionGraph.tsx`
    * `HistogramPrediction.tsx`
    * `UploadCsvModal.tsx`
  - `context/`
    * `AuthContext.tsx`
  - `utils/`
    * `api.ts`
  - `services/`
    * `firebaseConfig.ts`
  - `styles/`
    * `App.css`
- `.env`
- `package.json`
- `tsconfig.json`

Frontend Requirements and Dependencies:

- React [1] - UI library for building web interfaces

• TypeScript - Strongly typed JavaScript
• Material UI (MUI) - Component library for design
• Leaflet [7] - Interactive maps for OpenStreetMap integration
• Firebase [5]- Google authentication
• Axios - HTTP client for making API calls
• Nivo [8] - Visualization library for graphs and charts

G. *Backend Project Structure and Requirements*

The system's backend is developed using FAST API [2], a high-performance web framework for building APIs with Python 3.7+ based on standard Python type hints. This framework allows for the rapid creation of secure, efficient, and scalable RESTful APIs. The backend is orchestrated through Docker [6] containers, ensuring that the application runs in isolated, reproducible environments, facilitating consistent deployments across different stages and reducing potential configuration onflicts. SQLAlchemy, a powerful Object-Relational Mapping (ORM) library, is used to interact with the database, abstracting SQL queries and enabling seamless database operations. The system's database models and relationships are defined through SQLAlchemy, ensuring smooth and efficient database management. Additionally, Pydantic models are utilized to handle request validation. Pydantic allows the backend to validate incoming data, ensuring that the payloads conform to the expected structure and types before processing. This adds a layer of data integrity, preventing potential errors and vulnerabilities. The backend exposes several endpoints, including those for user management (such as user registration, authentication, and profile management), data ingestion (allowing users to upload AQI data in CSV format), and ML prediction (serving machine learning model predictions for air quality forecasting). The backend also integrates with machine learning models to generate real-time predictions, which are accessible via the API. By leveraging FAST API's [2] asynchronous capabilities, the backend can handle multiple requests efficiently, ensuring high throughput and low latency even under heavy load. The Docker [6] and Pydantic, the backend is scalable, maintainable, and well-structured to meet both current and future requirements, making it a reliable component of the overall system.

Folder Hierarchy:

- `Dockerfile.aip`
- `app/`
  - `__init__.py`
  - `main.py`
  - `api/routes/`
    * `__init__.py`
    * `air_quality_sites.py`
    * `auth.py`
    * `cors.py`
    * `prediction.py`
    * `users.py`
  - `core/config.py`
  - `db/`
    * `air_quality_sites.py`
    * `base.py`
    * `users.py`
  - `schemas/`
  - `services/prediction.py`
- `models/`
  - `RF_model_for_website.joblib`
  - `NN_model_for_website.h5`
- `config/config.json`
- `requirements.txt`

Backend Requirements and Dependencies:

• FastAPI[2] - Web framework for creating APIs
• SQLAlchemy - ORM for database operations
• Pydantic - Data validation
• Uvicorn - ASGI server
• Tensorflow [4] - For neural network models (h5)
• Joblib - To load Scikit-learn [3] models
• Scikit-learn [3]- ML model framework

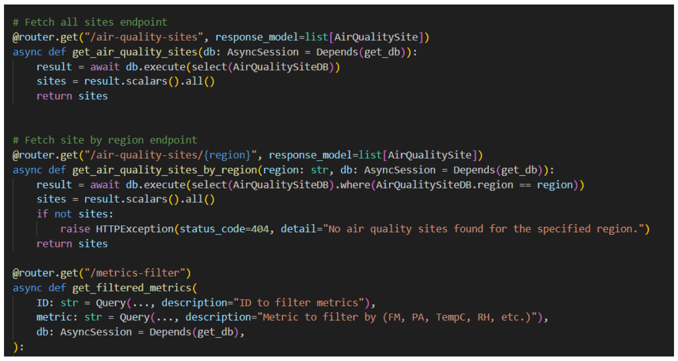
H. API Endpoints for Line Graphs Generation Project

These API endpoints are responsible for generating visual graphs (Line Graphs) based on user-defined parameters such as date range or air quality site and others.

Here is a detailed list:

1. POST/air-quality sites:

The POST /air-quality-sites endpoint is designed to upload new air quality data when an authenticated user submits a CSV file from the front end. The data flows through the React front end [1], is securely routed via the Cloudflare Proxy [9], processed by the Fast API backend [2], and ultimately handled on AWS EC2 instances using Docker. This endpoint parses and stores the AQI data for use in future predictions and visualizations.



```python
# Fetch all sites endpoint
@router.get("/air-quality-sites", response_model=list[AirQualitySite])
async def get_air_quality_sites(db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(AirQualitySiteDB))
    sites = result.scalars().all()
    return sites


# Fetch site by region endpoint
@router.get("/air-quality-sites/{region}", response_model=list[AirQualitySite])
async def get_air_quality_sites_by_region(region: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(AirQualitySiteDB).where(AirQualitySiteDB.region == region))
    sites = result.scalars().all()
    if not sites:
        raise HTTPException(status_code=404, detail="No air quality sites found for the specified region.")
    return sites

@router.get("/metrics-filter")
async def get_filtered_metrics(
    ID: str = Query(..., description="ID to filter metrics"),
    metric: str = Query(..., description="Metric to filter by (FM, PA, TempC, RH, etc.)"),
    db: AsyncSession = Depends(get_db),
):
```

Fig. 3. API Endpoints for the AQI Filters

2. GET /air-quality-sites:

The GET /air-quality-sites endpoint is used to retrieve all stored air quality data entries from the system. Once the data has been ingested and stored, this endpoint allows the front end to access the complete dataset for rendering graphs, histograms, and map-based visualizations. It supports comprehensive analysis by providing users with access to the full range of AQI records.

3. GET /air-quality-sites/region:

The GET /air-quality-sites/region endpoint enables the front end to request AQI data filtered by a specific region. This supports region-specific analysis and dynamic updates on the interactive map, allowing users to focus on localized air quality trends.

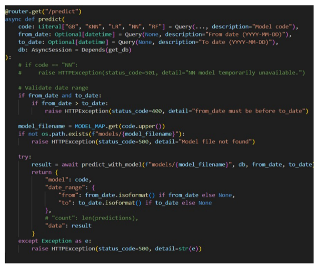
4. GET /metrics-filter:

The GET /metrics-filter endpoint allows users to retrieve only selected air quality metrics such as FM, cf1, temperature (Temp C), and relative humidity (RH). This helps filter and refine visualizations on the front end by focusing on relevant parameters, thereby improving usability and data clarity during analysis.

5. GET /all-regions:

The GET /all-regions endpoint returns a list of all regions available in the stored AQI data. It optionally accepts latitude and longitude filters to refine the geographic scope of the results. This is primarily used for populating region selectors, updating map markers, and enhancing geo-spatial data interaction on the platform.

From Figure 3, This Fast API [2] code snippet defines three endpoints to manage and retrieve air quality monitoring data. The /air-quality-sites endpoint fetches all available monitoring sites, while /air-quality- sites/region} filters sites based on a specific region and returns a 404 error if no matching data is found. The /metrics-filter endpoint is designed to filter specific metrics (like PM, temperature, humidity) based on site ID, though its implementation is not shown.

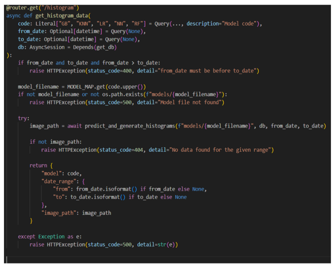


```python
@router.get("/predict")
async def predict(
    code: Literal["GB", "KNN", "LR", "NN", "RF"] = Query(..., description="Model code"),
    from_date: Optional[datetime] = Query(None, description="From date (YYYY-MM-DD)"),
    to_date: Optional[datetime] = Query(None, description="To date (YYYY-MM-DD)"),
    db: AsyncSession = Depends(get_db)
):
    # if code == "NN":
    #     raise HTTPException(status_code=501, detail="NN model temporarily unavailable.")

    # Validate date range
    if from_date and to_date:
        if from_date > to_date:
            raise HTTPException(status_code=400, detail="from_date must be before to_date")

    model_filename = MODEL_MAP.get(code.upper())
    if not os.path.exists(f"models/{model_filename}"):
        raise HTTPException(status_code=500, detail="Model file not found")

    try:
        result = await predict_with_model(f"models/{model_filename}", db, from_date, to_date
        return {
            "model": code,
            "date_range": {
                "from": from_date.isoformat() if from_date else None,
                "to": to_date.isoformat() if to_date else None
            },
            # "count": len(predictions),
            "data": result
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```



```python
@router.get("/histogram")
async def get_histogram_data(
    code: Literal["GB", "KNN", "LR", "NN", "RF"] = Query(..., description="Model code"),
    from_date: Optional[datetime] = Query(None),
    to_date: Optional[datetime] = Query(None),
    db: AsyncSession = Depends(get_db)
):
    if from_date and to_date and from_date > to_date:
        raise HTTPException(status_code=400, detail="from_date must be before to_date")

    model_filename = MODEL_MAP.get(code.upper())
    if not model_filename or not os.path.exists(f"models/{model_filename}"):
        raise HTTPException(status_code=500, detail="Model file not found")

    try:
        image_path = await predict_and_generate_histograms(f"models/{model_filename}", db, from_date, to_date)

        if not image_path:
            raise HTTPException(status_code=404, detail="No data found for the given range")

        return {
            "model": code,
            "date_range": {
                "from": from_date.isoformat() if from_date else None,
                "to": to_date.isoformat() if to_date else None
            },
            "image_path": image_path
        }

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Fig. 4. API Endpoints for ML Model Integration(left) to Generate Histogram Graphs (right)

All endpoints utilize asynchronous database sessions for efficient data access.

I. *Machine Learning Models*

The system utilizes multiple machine learning models: Random Forest, Neural Networks, Linear Regression, Gradient Boosting, and K-Nearest Neighbors[10]. Models are serialized using Joblib (.joblib) and Tensorflow [4] (h5) formats. These models are trained on historical AQI datasets containing temperature, humidity, and CF1 sensor data. Models are serialized using Joblib (. joblib) and Tensorflow [4] (h5) formats. These models are trained on historical AQI datasets containing temperature, humidity, and CF1 sensor data.

List of API Endpoints Specification for Machine Learning Integration Graph Generation:

These API endpoints are responsible for generating visual graphs (Histogram) based on the selected machine learning models, enabling users to analyze predictions and model performance interactively.

6. GET / predictors /predict:

The GET /predict endpoint generates predictions using a specified machine learning model, such as Gradient Boosting (GB), K-Nearest Neighbors (KNN), Linear Regression (LR), Neural Networks (NN), or Random Forest (RF) [10], along with a defined date range.

7. Histogram API [get(”/histogram”)]:

The GET /histogram endpoint generates a histogram graph for the chosen machine learning model's predictions. This endpoint is used when the user wants to visualize the distribution of the predicted air quality metrics. The backend calculates the necessary data and returns the histogram, which is then displayed in the front end for a clear visual representation of the model's prediction spreads.

From Figure 4, The snippet shows the endpoint can be invoked when a user requests a forecast or future scenario, and the backend processes the input data to provide predictions. The results are then visualized on the front end as graphs or overlays, helping users to understand potential future air quality trends and it also shows that the Fast API [2] endpoint /histogram generates and returns a histogram image path for AQI data predictions based on a selected ML model and optional date range. It also validates dates, loads the appropriate model, and uses a helper function to generate the histogram. If no data is found, it returns to a 404 error, and also shows that this Fast API [2] code defines a /predict endpoint that allows users to select a machine learning model (e.g., RF, KNN, LR, etc.) and optionally provides a date range to get AQI predictions. It validates input, checks model file availability, also it uses an async function to generate predictions. The response includes the model used, date range, prediction count, and data. Neural Network (NN) model is also included, and proper error handling is implemented throughout the API End Point.

J. *Visualization and Nivo Graph Integration*

The system uses Nivo, which is a powerful, open-source data visualization library built specifically for React [1] applications, and it plays a central role in the AQI Metric Prediction System's front end. By leveraging Nivo [8], the platform delivers highly interactive and customizable visualizations, allowing users to toggle between different machine learning models and instantly view results through dynamic line graphs and histograms directly on the map interface. These graphs are not static; they update in real time based on the AQI site IDs selected by the user, which are highlighted on the OpenStreetMap component, providing immediate visual feedback and supporting in-depth exploration of air quality trends.

Use of Nivo's [8] responsive design ensures that visualizations render seamlessly across devices, while its Extensive customization options enable the system to present complex AQI data in a clear and engaging manner. This integration of Nivo [8] enhances user engagement, making it easier for both technical and non-technical users to interpret model outputs, compare predictions, and gain actionable insights from the system's rich environmental datasets.

K. *Deployment and Cloud Infrastructure*

The system's backend is containerized using Docker [6], which allows applications and them dependencies to be packaged together in lightweight, portable containers. This ensures consistent behavior across development, testing, and production environments. Each backend service—including the Fast API [2] application, machine learning inference modules, and data processors—is encapsulated in its own Docker [6] container. These containers are orchestrated and deployed on AWS EC2 instances, offering a flexible and scalable infrastructure. Amazon EC2 (Elastic Compute Cloud) provides virtual servers that can be scaled up or down depending on demand, ensuring high availability and performance for real-time AQI predictions and data processing tasks. The front end is hosted using Cloudflare Pages [9], a JAM stack platform that supports direct deployment from Git repositories. Cloudflare Pages delivers the React-based front end globally via a Content Delivery Network (CDN), minimizing latency for end users. To address security and cross-origin communicationchallenges, a Cloudflare Worker [9] acts as a reverse proxy between the front end and backend. Cloudflare Workers [9] are lightweight serverless functions that intercept and manage HTTP requests at the

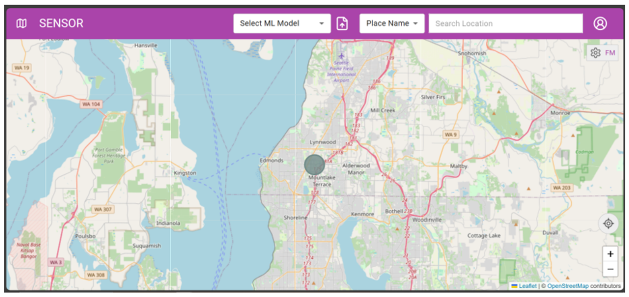


Fig. 5. The Landing Page of the website

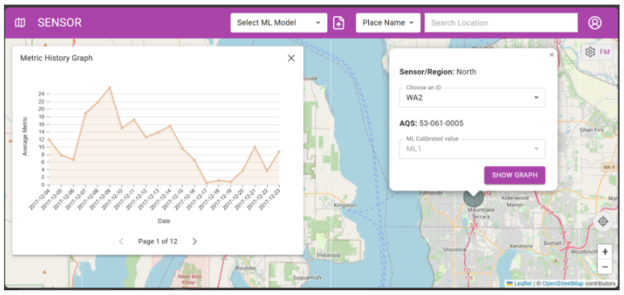


Fig. 6. Line Graph Showing AQI Sensor Readings

edge, enabling features like CORS handling. XSS protection, input validation, and HTTPS enforcement. This approach improves performance by reducing backend load and enhances security by filtering and controlling access to sensitive API endpoints. Together, Docker, AWS, and Cloudflare form a robust, cloud-native architecture that ensures the system is scalable, secure, and efficient.

VI. RESULTS

This section shows that the system effectively delivers accurate AQI predictions by processing data from multiple sensor inputs. The website has been successfully deployed, providing users with real-time access to predictions and interactive visualizations.

A. *AQI Sensors*

The results section of the AQI Metric System demonstrates a seamless and interactive user experience, beginning with secure Google authentication via Firebase[5], which ensures that only authorized users can access the platform's data upload and prediction features.

Once logged in, users can upload datasets through a dedicated file upload option. The system immediately validates the uploaded CSV, checking for the presence of all required fields-PM2.5, temperature, relative humidity, and CF1-in every row. If any values are missing, the upload is rejected, and a pop-up notification informs the user of the issue, halting further processing and graph generation. For valid datasets, the backend processes the data, stores it, and enables the generation of relevant graphs for analysis.

Users can explore air quality data through an interactive map interface, selecting regions or specific sensor IDs to view detailed metrics and visualizations. The platform's filter options allow users to apply up to four different options to the user and to check the different AQI of the sensor types and drill down to individual sensor data, with the ability to search for locations using an autocomplete feature that centers and highlights the chosen site on the map. For each selected filter or sensor, dynamic line graphs are generated, displaying time-series trends and supporting comprehensive analysis within the specified date range.

The following steps outline the workflow for generating an AQI line graph:

When the user accesses the website, they are greeted with a landing page that highlights the platform's main functionalities in a clear and user-friendly layout.

The Figure 5 presents a detailed view of the landing page and the user interface layout. It highlights the key components available to the user upon accessing the application, including navigation options, model selection menus, upload functionality, and graph generation controls. This interface serves as the primary point of interaction, designed to guide users through the workflow in an intuitive and user-friendly manner.

Users can upload a custom dataset using the **"Upload CSV File"** option to generate corresponding visualizations and graphs. The interface displays the CSV upload process, confirming once the file has been successfully uploaded. Upon upload, both a line graph and a histogram graph are generated based on the provided data. A message is displayed during the upload process, stating: *"Any row with blank values for FM, Temp C, RH, or Cf1 columns will be rejected. Please check before uploading."* The expected visualizations will only be generated if the uploaded CSV file includes valid entries with all four parameters present and properly formatted. Once the data is validated, the platform successfully generates the line graph, and similarly, equivalent graphs are produced for other filters applied.

The filtering functionality that allows users to apply up to four distinct filters—FM, Cf1, RH, and Temp C—to efficiently narrow down sensor data based on specific criteria. This dynamic filtering system, accessible through the Dropbox section, enables users to explore subsets of sensor information with precision and ease. After applying the desired filters, users can further refine their analysis by selecting a specific sensor ID from the filtered list.

From Figure 6, upon selecting a sensor from the platform's map , users are presented with detailed insights related to that specific AQI id of the sensor. This includes a comprehensive display of data readings, key operational metrics, and interactive visualizations such as Line graphs, as well as the sensor region and its AQS ID. When a user selects a particular sensor ID from the **'Choose an ID'** dropdown menu, a line graph is generated that displays individual data

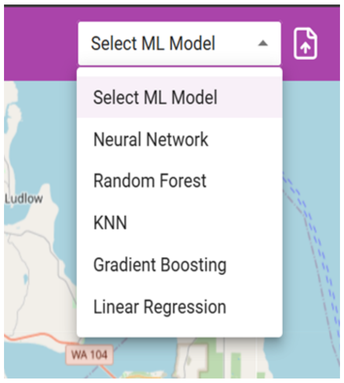


Fig. 7. Drop down Selection for Machine Learning Models

points with corresponding dates and times related to the sensor. Similarly, when other filters are applied, the platform generates comparable line graphs based on the selected criteria.

Finally, the required line graph is generated and displayed on the dashboard, visualizing the historical metric data for the selected sensor and sensor ID within the specified date range. These features collectively provide users with a deeper understanding of the sensor's behavior over time, including its environmental readings and overall performance within the monitored system across the past few years.

B. ML Model Selection

The system also supports advanced machine learning model selection, allowing users to choose from five different models-Random Forest, Neural Networks, Logistic Regression, Gradient Boosting, and K-Nearest Neighbors [10] -to generate calibrated prediction graphs. Upon model selection, a confirmation dialog appears, and once confirmed, the chosen model runs in the background to generate a histogram comparing original and calibrated values. The resulting graph is presented as a downloadable image for further analysis, it is displayed as a downloadable image. The other models also generate similar graphs.

The following steps outline the workflow for generating a histogram graph based on the selected machine learning model from the available options.

The Figure 7 illustrates a user-friendly dropdown menu interface that enhances interactivity within the application by allowing users to choose from five distinct predictive models. This dropdown acts as a control mechanism in the front end, enabling users to seamlessly select a specific model based on their preference or analytical needs. Once a model is selected, the application sends a request to the backend, typically including the model identifier and relevant data inputs. The backend processes this request—running the selected model's prediction logic—and returns the output in the form of structured data. This data is then used to dynamically render a prediction graph on the front end, updating the visualization in real time without requiring a full page reload. This design helps to make the UI more practical and better for the user

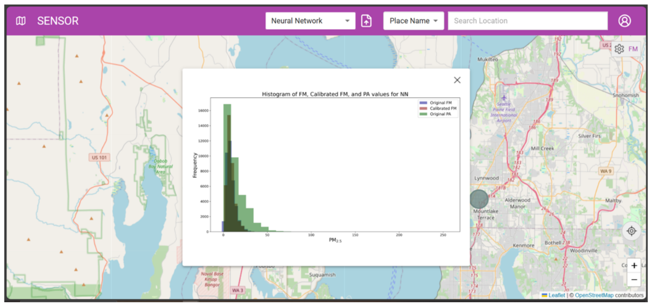


Fig. 8. Histogram Graph of the Neural Network model

and Improves user engagement and flexibility but also supports comparative analysis across different models.

When the Neural Network (NN) model is selected, the user is prompted with the message: **'You selected NN. Would you like to proceed?'** This confirmation step ensures the user intentionally opts to use the NN model for analysis. Upon confirming and proceeding, the system enables the option to generate a graph. Once the user initiates the graph generation, the system processes the input using the NN model and generates the corresponding histogram graph.

The Figure 8 shows a histogram graph for the selected Neural Network model, comparing the original values to the calibrated values. The graph includes dataset values, specifically the original **Fm** values and the calibrated **Cf1** values. It visually compares how the calibrated outputs (FM) align with the original data (Fm), helping users assess the effectiveness of the model calibration.

Finally, a histogram graph is generated based on the model selected by the user, illustrating the comparison between original and calibrated values. This visualization highlights the effectiveness of the calibration process by showing how the model adjusts raw data to improve accuracy and consistency. The distribution of values before and after calibration provides insights into model performance, error correction, and the degree of alignment with expected outcomes. The results can be further enhanced by using top-performing models such as Neural Networks (NN), Random Forest (RF), and Gradient Boosting (GB) [10], which are known for their high accuracy in prediction tasks. This allows users to see how the model's predictions have been adjusted through calibration, making it easier to understand the accuracy and reliability of the model.

The Other Functionalities are the system allows the Users to efficiently search for locations on the map using the autocomplete suggestion feature, which provides instant place suggestions as they type. Before logging in, users are prompted to sign in with their Google account, ensuring secure access to the platform; after successful authentication, the full dashboard functionality becomes available, including data upload and visualization tools.When a location is selected from the auto complete suggestions, a pointer or marker appears on the map at the corresponding coordinates, visually highlighting the Searched location for easy reference.

The project source code is available at the following GitHub repository:

https://github.com/KarthikElur/MASTERS-CAPSTONE-PROJECT

VII. CONCLUSION

This project shows a comprehensive and resilient Air Quality Index (AQI) monitoring and prediction platform, seamlessly integrating real-time geospatial mapping technologies, advanced machine learning algorithms for predictive analytics, and secure cloud-based deployment to ensure high availability, scalability, and reliability. The platform supports real-time data ingestion from Federal monitor and purple air environmental sensors, implementing data pre-processing and anomaly detection techniques, and deliver insightful visualizations through interactive dashboards with different types of graph-based data with also integrating different types of ML model that shows calibrated data to build with modular microservices architecture, it enables easy scaling and maintenance, supports multi-region deployments and ensures data privacy and integrity through robust security practices.

Designed with future proofing in mind, the system is highly extensible to incorporate additional smart city applications such as traffic flow optimization, energy management, disaster response, and public health.

REFERENCES

[1] React JS Documentation. https://reactjs.org
[2] FAST API Documentation.https://fastapi.tiangolo.com
[3] Scikit-learn Documentation. https://scikit-learn.org
[4] TensorFlow Documentation. https://www.tensorflow.org
[5] Firebase Documentation.https://firebase.google.com
[6] Docker Documentation. https://docs.docker.com
[7] Leaflet.js Documentation.https://leafletjs.com
[8] Nivo Charts. https://nivo.rocks
[9] Cloudflare Workers Documentation. https://developers.cloudflare.com/workers
[10] ML Models Integrated in the System :

- Neural Network
- Random Forest
- Linear Regression
- Gradient Boosting
- K-Nearest Neighbors