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

ELURI SAI KARTHIK

Clarkson University

Abstract :

This project proposes a scalable and interactive AQI prediction system that integrates modern frontend and backend frameworks with cloud deployment and machine learning. The frontend utilizes React with Leaflet for OpenStreetMap interaction and Firebase for Google Authentication. Predictions are rendered via Nivo graphs and histograms, connected to specific AQ sensor locations. The FastAPI backend, hosted by using Docker on AWS, serves ML inference through RESTful APIs, leveraging models such as Random Forest and Neural Networks. To handle CORS and XSS issues, a Cloudflare Worker is implemented as a proxy layer for secure front-end-backend communication.

Keywords— Air Quality Index, Machine Learning, Cloudflare Workers, FastAPI, React, Leaflet, Firebase, Docker

Introduction

Air pollution has rapidly escalated into one of the most pressing environmental threats, impacting 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) prediction system. The system seamlessly integrates multiple components, including an intuitive and interactive frontend for visual analytics, a powerful backend for data processing and management, cloud-based deployment for scalability and accessibility, and sophisticated machine learning (ML) models for accurate prediction of air quality trends. By leveraging real-time data streams, advanced predictive algorithms, and user-friendly interfaces, the proposed platform not only enables continuous monitoring but also empowers users and policymakers with actionable insights to make informed decisions aimed at improving public health and environmental sustainability

1. Background

The Air Quality Index (AQI) monitoring and prediction has become increasingly critical due to rapid urbanization and industrialization. Pollution not only impacts human health but also the environment, making accurate AQI prediction vital. Traditional AQI models have relied heavily on deterministic and statistical approaches, which often lack flexibility and accuracy with real-world noisy data. With the advent of machine learning and cloud-based infrastructures, real-time, scalable, and highly accurate AQI prediction systems can now be developed.

2. Objective

The objective of this project is to build an accurate, scalable, and secure AQI prediction system leveraging machine learning models and full-stack cloud deployment. The system aims to provide real-time AQI predictions through a user-friendly frontend, reliable backend APIs, and robust cloud infrastructure while ensuring security and high performance.

3. Dataset

The dataset for AQI prediction includes historical records of air quality sensor data such as PM2.5, temperature, humidity, and CF1 sensor readings. Data is preprocessed to handle missing values, outliers, and normalization to enhance model performance. The system integrates data from highly reliable Purple Air sensors and certified federal monitoring stations.

This combination ensures enhanced accuracy, consistency, and credibility in air quality measurements.

By leveraging both sources, the platform delivers robust real-time monitoring and precise AQI predictions.

4. Variables Description

- PM2.5 Concentration: Fine particulate matter concentration.
- Temperature: Ambient temperature at the sensor location.
- Humidity: Relative humidity percentage.
- CF1 Sensor Value: Calibration factor for sensor adjustment.
- Timestamps: Date and time stamps for each data reading

- FM: Federal Monitors

-Tensor flow: TensorFlow is an open-source platform for machine learning and deep learning that enables easy model building, training, and deployment.

5. Methods

5.1 Data Collection and Preprocessing

- Missing values are filled using mean imputation.
- Outliers are capped using interquartile range (IQR) methods.
- Features are normalized using z-score standardization.
- Duplicate entries are removed to maintain data integrity.

5.2 Feature Selection

- Correlation analysis to remove redundant variables.
- Feature importance analysis using Random Forest to retain significant predictors.
- Principal Component Analysis (PCA) was tested but final models used raw features for better explainability.

5.3 Model Selection

- Random Forest: Chosen for its robustness.
- Neural Networks: Used for capturing non-linear relationships.
- Logistic Regression, Gradient Boosting, and K-Nearest Neighbors models were also tested.

5.4 Model Training

- 80:20 training/testing data split.
- 5-fold cross-validation for hyperparameter tuning.
- Evaluation metrics recorded: RMSE, Accuracy.

5.5 Model Evaluation and Validation

- Metrics Used: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-squared.
- Confusion matrix for classification models (where applicable).

5.6 Interpretation and Visualization

- Feature importance plots for tree-based models.
- Nivo graphs for dynamic visualization on the frontend.
- SHAP values were explored for model explainability but not fully deployed.

5.7 Deployment

- Backend containerized with Docker.
- Deployed on AWS EC2 instances.
- Cloudflare Worker acts as a reverse proxy for security and CORS handling.
- Frontend deployed on Cloudflare Pages.

5.8 Variable Distributions

- PM2.5 readings skewed toward high values in urban areas.
- Temperature and humidity show seasonal patterns affecting AQI readings.

6. System Components

6.1. **System Architecture**: The system architecture is divided into two layers: a React-based frontend for visualization and interaction, and a Fast API backend for data ingestion, ML inference, and secure API serving. Communication occurs via RESTful APIs. Firebase Authentication handles user login, and Docker containers deployed on AWS host the backend services. A Cloudflare Worker acts as a proxy to address CORS and XSS concerns.

Nivo is used for interactive data visualization. Users can toggle between different models and view results via line graphs and histograms on the map. Graphs dynamically updated based on selected AQ site IDs highlighted on the OpenStreetMap.

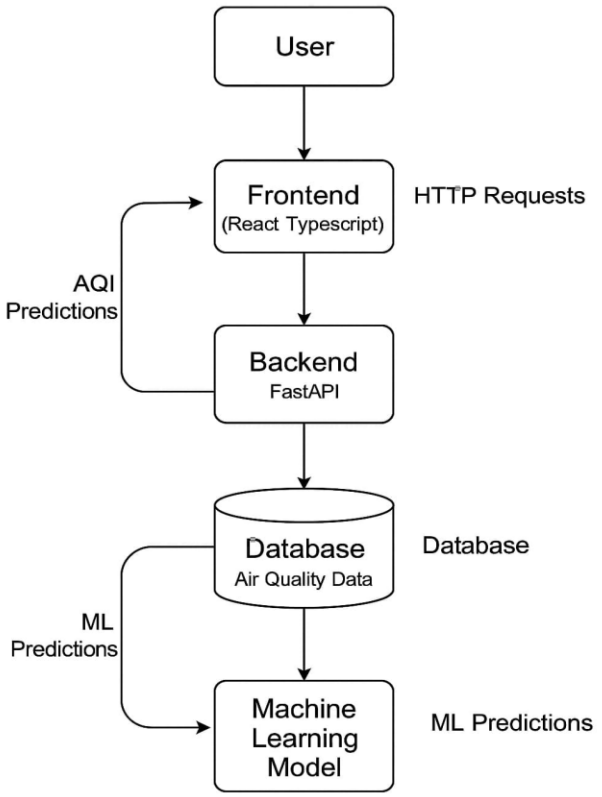


FIG 6.1: SYSTEM ARCHITECTURE

6.2. **Flow Chart**: The flow chart shows user authentication before allowing CSV uploads. Uploaded files are processed via Cloudflare proxy and backend services on AWS EC2, then returned as JSON. The frontend uses this data for visualizations like line graphs and map updates.

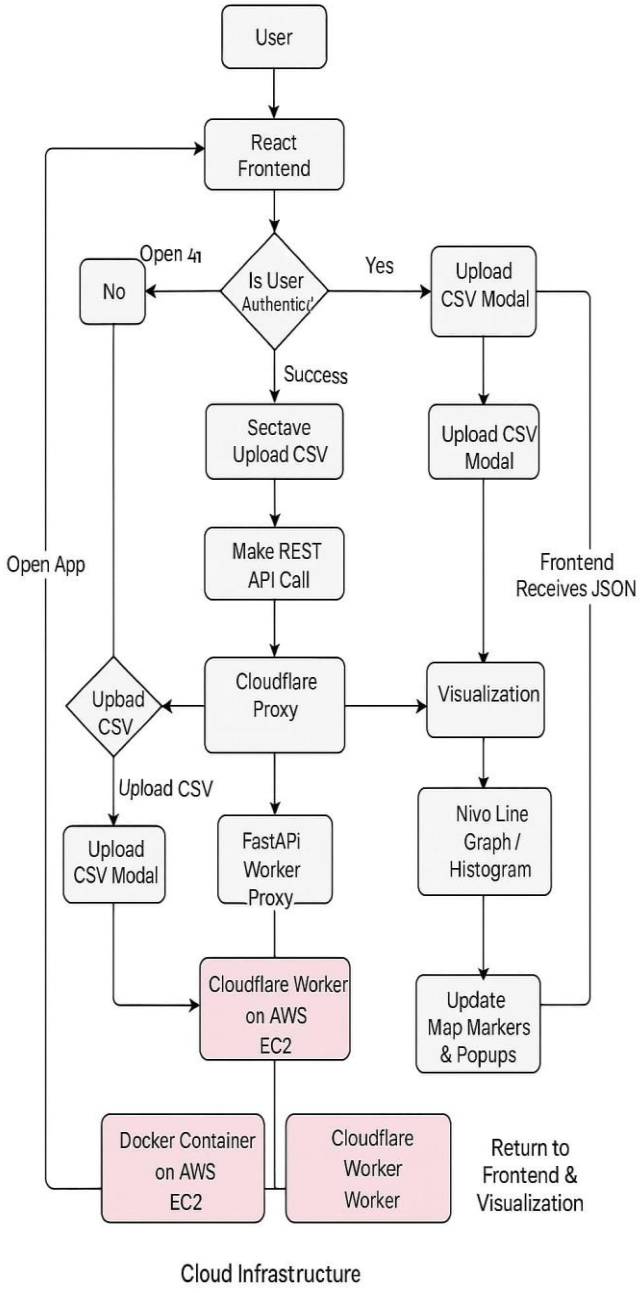


FIG 6.2: Flow chart

7. Functional and Non-Functional Requirements

7.1 Functional Requirements:

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).
- 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 frontend-backend communication should be securely proxied to prevent CORS and XSS vulnerabilities (using Cloudflare Workers).
- Sensor Data Integration: The system must ingest AQI sensor and process it for prediction.
- User Management: The backend must support user registration, login, and profile management.

7.2 Non-Functional Requirements:

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).
- 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.
- Compatibility: The backend and frontend should be compatible with major operating systems and browsers,

8. USE CASE DIAGRAM

The use 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 map, view prediction graphs, and get AQI predictions, while a 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.

8.1 User Functionalities

- Login with Google Authentication:
  Users must authenticate using their Google account, leveraging Firebase for secure and streamlined sign-in.

- View AQI Map:
  After login, users can access an interactive map (built with Leaflet 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, etc.), with results visualized on the platform.

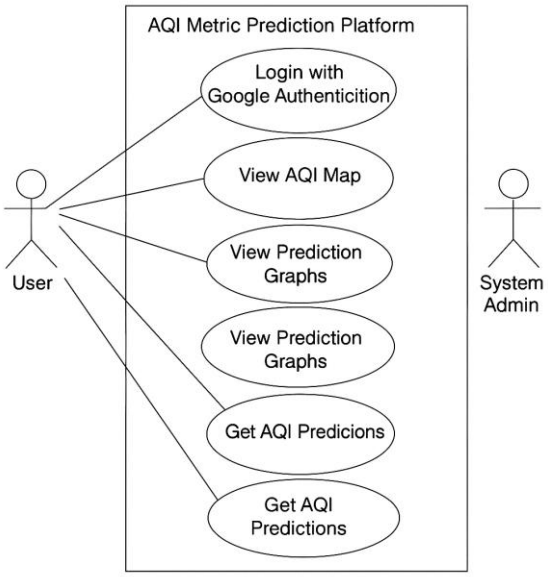


FIG 8. Use Case Diagram

9. Detailed Project Structure

9.1 Frontend Project Structure and Requirements

The frontend is developed using React with TypeScript and Material UI (MUI). Leaflet is used for rendering OpenStreetMap, enabling users to explore AQI data visually. Key components include Map.tsx for geospatial display, PredictionGraph.tsx and HistogramPrediction.tsx for visualizing predictions, and UploadCsvModal.tsx for uploading custom data.

Google Sign-In is implemented through Firebase Authentication. Firebase configuration is stored in a secure .env file. User sessions are managed using the Firebase SDK and React Context API.

9.1.1 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
```

9.1.2 Frontend Requirements and Dependencies

- React - UI library for building web interfaces
- TypeScript - Strongly typed JavaScript
- Material UI (MUI) - Component library for design
- Leaflet - Interactive maps for OpenStreetMap integration
- Firebase - Google authentication
- Axios - HTTP client for making API calls
- Nivo - Visualization library for graphs and charts
- React Context API - For managing authentication state

9.2 Backend Project Structure and Requirements

The backend is developed using FastAPI and orchestrated through Docker containers. SQLAlchemy is used for ORM and Pydantic models handle request validation. The backend serves multiple endpoints including user management, data ingestion, and ML prediction.

9.2.1 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
```

9.2.2 Backend Requirements and Dependencies

- FastAPI - Web framework for creating APIs
- SQLAlchemy - ORM for database operations
- Pydantic - Data validation
- Uvicorn - ASGI server
- TensorFlow - For neural network models (h5)
- Joblib - To load scikit-learn models
- Scikit-learn - ML model framework
- Pandas - Data manipulation

10. API Endpoints for Graphs Generation Project

The system leverages a set of well-defined API endpoints to power both data visualization (graphs) and machine learning (ML) model predictions, as depicted in the flowchart. Here's a more refined explanation of how these endpoints is integrated into the backend and frontend workflow:

10.1 List of API Endpoints Specification for AQI Graphs Generation System

- **POST /air-quality-sites**: Used to upload new air quality data (typically from CSV uploads by authenticated users). This endpoint is called from the frontend after the user successfully uploads a CSV, passing the data through the backend infrastructure (React frontend → Cloudflare Proxy → Fast API Worker → AWS EC2/Docker).

```python
# Insert endpoint
@router.post("/air-quality-sites")
async def add_air_quality_data(req: AirQualityDataReq, email: str = Header(...),
                               db: AsyncSession = Depends(get_db),):
    try:
        user = await get_user(email, db)
        pass
    except UserNotFoundError as e:
        raise HTTPException(status_code=401, detail=str(e))

    if not user:
        raise HTTPException(status_code=401, detail="Unauthorized: User not found")

    if user.role != 'admin':
        raise HTTPException(status_code=401, detail="Unauthorized: Invalid user credentials")

    # Check if the list size exceeds 1000
    if len(req.data) > 1000:
        raise HTTPException(status_code=400, detail="Data list exceeds the limit of 1000 records.")
```

FIG (10.1.1): This figure shows the POST **/air-quality-sites** endpoint

- **GET /air-quality-sites**: Retrieves all stored air quality data entries, enabling the frontend to fetch comprehensive datasets for visualization.
- **GET /air-quality-sites/region:** Fetches air quality data filtered by a specific region, supporting region-based visualizations and map updates.
- **GET /metrics-filter**: Allows the frontend to request specific metrics (such as FM, cf1, Temp C, RH) based on filters, ensuring that only relevant data is visualized.
- **GET /all-regions**: Retrieves data for all regions, optionally filtered by latitude and longitude, supporting geospatial visualizations and updating map markers/popups.

FIG (10.1.2): the below figures show the GET endpoints of the AQI that generates the graph

```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 specif
    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, cf1, TempC, RH, etc.)"),
    db: AsyncSession = Depends(get_db),
):
```

10.2 Machine Learning Models

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

10.2.1 List of API Endpoints Specification for Prediction and Machine Learning Integration

- **GET / predictors /predict**: Generates predictions using the specified ML model (GB, KNN, LR, NN, RF) and date range. This endpoint is called when the user requests forecast or future scenario, and the predicted results are visualized as graphs or overlays in the front end.

```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))
```

FIG (10.2.1.1): This figure shows the GET/predictors/ predict endpoint is called for getting and fetching the predicted results are visualized as graphs

```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)
):
```

FIG (10.2.1.2): This figure shows the GET. HISTORAM endpoint is called for getting and fetching the histogram graphs

11. Visualization and Nivo Graph Integration

Nivo is used for interactive data visualization. Users can toggle between different models and view results via line graphs and histograms on the map. Graphs dynamically updated based on selected AQ site IDs highlighted on the OpenStreetMap.

12. Deployment and Cloud Infrastructure

The backend is containerized using Docker and deployed on AWS EC2 instances. The front end is hosted on Cloudflare Pages. To handle CORS and XSS issues, a Cloudflare Worker acts as a reverse proxy, securely routing requests to the backend.

13. Expected Results

The system successfully delivers accurate AQI predictions across multiple sensor inputs. The website is deployed.

13.1 AQI SENSORS

- The dashboard offers an interactive map where users can select regions or specific sensor IDs, providing detailed metrics and clear geographic context for the data.
- Core website functions include uploading datasets, visualizing sensor data, generating calibrated predictions, and a search bar for easy navigation.
- The process begins with users uploading a dataset via the file upload option.
- Upon upload, the website automatically checks that all four required values are present in every row; if any value is missing, the upload is immediately rejected, and a pop-up message notifies the user of the issue. The upload is canceled upon confirmation, and no graphs are generated.
- If the dataset passes validation, the system proceeds to generate relevant graphs for analysis.
- Users can select any sensor filter (such as FM), which automatically generates a line graph for that sensor. They can further refine their view by choosing individual sensor IDs, with details about the sensor such as region and AQS ID displayed for deeper analysis is displayed.
- The search bar allows users to quickly find sensor locations by entering a region name (e.g., "Alaska"); the map centers and highlights the exact sensor location, enabling fast and precise navigation.
- Line graphs are generated for each selected sensor filter, displaying all relevant data for the chosen sensor within the specified date range, supporting comprehensive trend analysis and comparison**.**

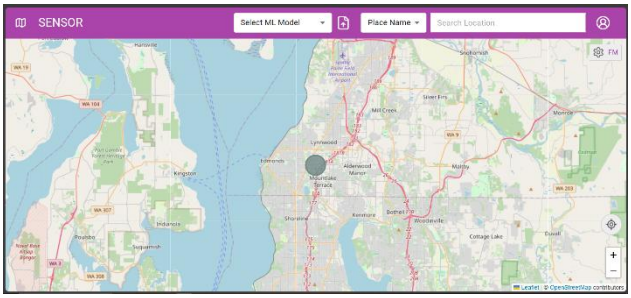


Fig. A. Landing page

Users see this landing page with main functionalities.

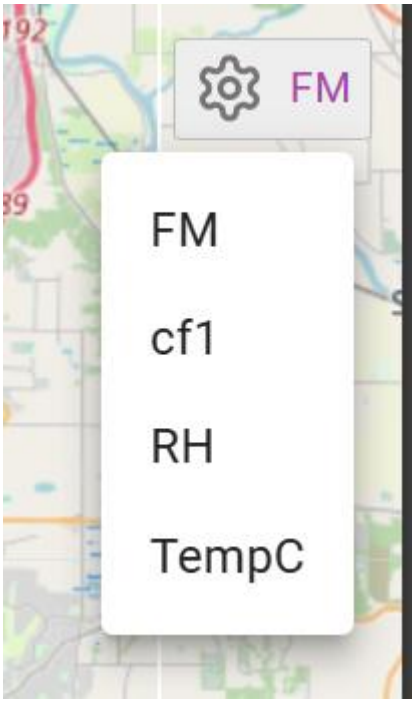


Fig. B. Filter option (FM, cf1, RH, Temp C)

Users can apply up to four different filters to explore various sensor types and their details, and then select a specific sensor ID to view comprehensive data and visualizations for that individual sensor.

Expected Graph of Filter FM:

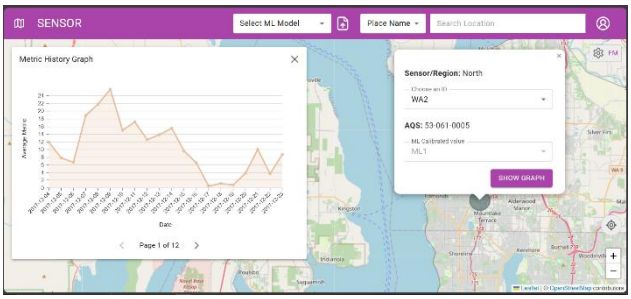


Fig. C. Graph with Filter FM

The user can upload the dataset for generating the graph in the UPLOAD CSV FILE option.

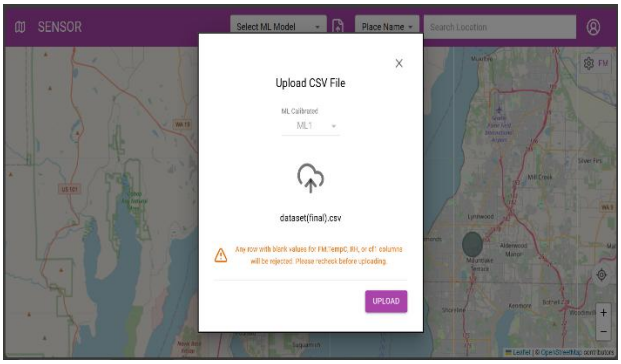


Fig D: File acceptance for correct data

13.2 ML MODEL SELECTION:

The user can select any 5 models from the drop box to get any model calibration values graph

- The platform allows users to choose from multiple machine learning models, which then generate histograms comparing original and calibrated values.
- Upon model selection, a confirmation popup appears (e.g., "You selected NN, would you like to proceed?").
- When the user confirms, the chosen model runs in the background and generates a histogram comparing original and calibrated values, which is displayed as a downloadable image.
- The other models also generate similar graphs

FIG 13.2.1. Selecting NEURAL NETWORK from the drop-down box below

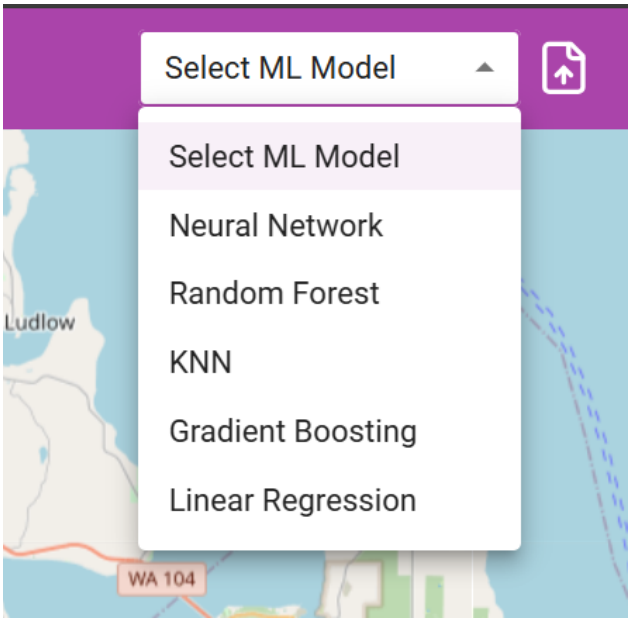


FIG 13.2.1.1: Confirm the selection,

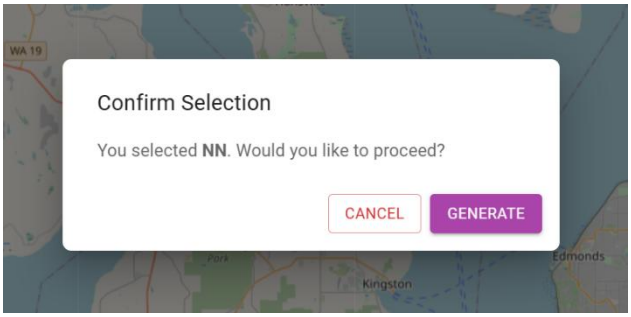
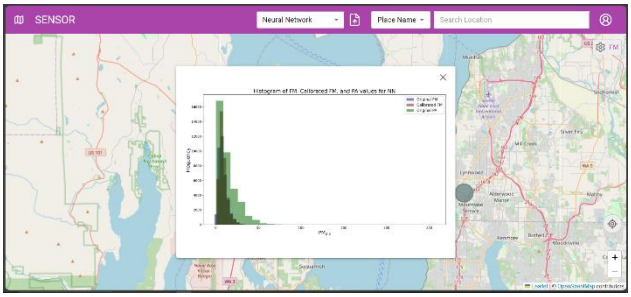


FIG 13.2.1.2: Generating the graph,



13.3 Other Functionalities:

- Users can 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 autocomplete suggestions, a pointer or marker appears on the map at the corresponding coordinates, visually highlighting the searched location for easy reference.
  highlighting the searched location for easy reference.

14. Conclusion

This project showcases 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, implements data preprocessing and anomaly detection techniques, and delivers 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. Built 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 futureproofing 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 monitoring initiatives etc.

15. References

[1] ReactJS Documentation, https://reactjs.org
[2] FastAPI Documentation, https://fastapi.tiangolo.com
[3] Scikit-learn Documentation, https://scikit-learn.org
[4] TensorFlow Documentation, https://www.tensorflow.org
[5] Firebase Docs, https://firebase.google.com
[6] Docker Docs, https://docs.docker.com
[7] Leaflet.js Docs, https://leafletjs.com
[8] Nivo Charts, https://nivo.rocks
[9] Cloudflare Workers Docs, https://developers.cloudflare.com/workers