

Air Quality Prediction System Using Machine Learning

Smit S. Badole¹, Kiran P. Pustode²

¹Student, Dept. of MCA, Karanjekar College of Engineering and Management, Sakoli

²Asst. Professor, Dept. of MCA, Karanjekar College of Engineering and Management, Sakoli

Abstract – Deteriorating urban air quality poses acute risks to public health, yet existing monitoring infrastructure provides only retrospective reports rather than actionable predictive intelligence. This paper presents the design, implementation, and performance analysis of an Air Quality Prediction System—a Django-based full-stack web application that integrates a trained Random Forest machine learning model with a multi-pollutant monitoring platform to deliver real-time Air Quality Index (AQI) predictions and historical trend analysis. The system is implemented across five architectural layers: IoT-compatible data collection, Django ORM-based relational persistence (SQLite/MySQL), a six-feature Random Forest Regressor trained on 29,532 samples from the Kaggle Indian Air Quality dataset, a Django view-based prediction interface, and a Bootstrap-styled responsive web dashboard. The machine learning pipeline covers complete stages of data loading, preprocessing, feature selection, train-test splitting (80:20), model training, joblib-based serialisation (aqi_model.pkl), and deployment-ready inference. The trained model achieves 99% training accuracy and 87% test accuracy, with a weighted F1-score of 0.812. Fifteen functional test cases spanning authentication, AQI prediction, admin CRUD operations, search filtering, and route protection all returned Pass results. The paper provides detailed implementation analysis covering Django model schemas, URL-to-view mappings, ML pipeline code, and testing outcomes, serving as a reproducible reference for ML-integrated environmental monitoring applications.

Key Words: Air Quality Prediction, Random Forest, Django, Machine Learning, AQI Forecasting, IoT Sensor Data, PM2.5, Environmental Monitoring.

1. INTRODUCTION

Air pollution has emerged as one of the most consequential public health challenges of the twenty-first century. In India, where the Central Pollution Control Board (CPCB) monitors pollutant concentrations across more than 800 stations nationally, the gap between data collection and predictive public guidance remains significant. Residents and local authorities receive current AQI readings but lack forward-looking predictions that could enable proactive health advisories, industrial output adjustments, and pre-emptive urban planning responses [1], [3].

Machine learning offers a well-validated pathway to close this predictive gap. Ensemble methods, particularly Random Forest, have demonstrated consistent accuracy advantages over linear regression and statistical time-series models for multi-pollutant AQI prediction tasks, owing to their capacity to capture complex non-linear interactions among correlated pollutant features and meteorological parameters [4], [8]. However, most published ML-based AQI systems remain confined to research notebooks or command-line scripts without providing a deployable web interface accessible to non-technical stakeholders.

The Air Quality Prediction System presented in this paper bridges this gap by integrating a scikit-learn Random Forest Regressor with a Django web framework, producing a deployable environmental monitoring and prediction platform. This implementation paper documents the complete development lifecycle: system requirements, dataset characterisation, ML pipeline design with actual code, Django architectural decisions, database schema definition, and multi-level testing outcomes. The primary contributions are: (i) a five-stage ML pipeline with variance-filtered feature selection and joblib serialisation; (ii) three Django ORM model schemas supporting station, reading, and prediction entities; (iii) eight URL-to-view mappings covering public and admin workflows; (iv) 99% training accuracy and 87% test accuracy with weighted F1-score of 0.812; and (v) fifteen functional test cases covering the complete system with 100% pass rate.

2. REQUIREMENT ANALYSIS

2.1 Functional Requirements

The functional requirements were derived through analysis of stakeholder needs spanning general public users, environmental researchers, and system administrators. For the Public User Interface: (i) real-time AQI display for selected monitoring stations with colour-coded severity indicators; (ii) pollutant-input-based AQI prediction for manual entry of PM2.5, PM10, NO2, CO, SO2, and O3 values; (iii) historical reading search by station name and date range; and (iv) prediction history browsing showing past AQI estimates with timestamps. For the Admin Module: (v) sensor station CRUD management; (vi) air quality reading add/edit/delete operations; (vii) AQI prediction record management; and (viii) role-restricted admin panel access with session-based authentication.

2.2 Non-Functional Requirements

Non-functional requirements established for the system include: AQI prediction API response below 500 ms for standard pollutant input forms; web page load time below 2 seconds on standard broadband; responsive layout compatible with desktop (1920×1080) and mobile (375px) viewports via Bootstrap grid; admin route protection requiring valid session tokens on all /admin/ endpoints; model serialisation (aqi_model.pkl) enabling cold-start loading under 3 seconds; and database support for both SQLite (development) and MySQL (production) via Django ORM abstraction. Hardware requirements: minimum Intel i3 processor, 4 GB RAM (8 GB recommended), 1 GB free storage, Python 3.8+, Django 4.x.

3. SYSTEM DESIGN

3.1 Five-Layer System Architecture

The Air Quality Prediction System adopts a five-layer modular architecture. Layer 1 (Data Collection): IoT sensor stations and the Kaggle Indian Air Quality dataset (29,532 samples, 16 features) serve as dual data sources—live sensors for real-time monitoring and the historical CSV dataset for ML model training. Layer 2 (Data Processing): a Python-based preprocessing pipeline (pandas, NumPy) handles missing value imputation, outlier filtering, feature selection, and StandardScaler normalisation. Layer 3 (ML Prediction Engine):

a scikit-learn RandomForestRegressor trained on six primary pollutant features and serialised to aqi_model.pkl via joblib. Layer 4 (Application Logic): a Django 4.x application server hosting the ORM models, view functions, template rendering engine, and admin authentication. Layer 5 (Presentation): Bootstrap-styled HTML5/CSS3 templates rendering the public dashboard, prediction forms, search interface, and admin panel.

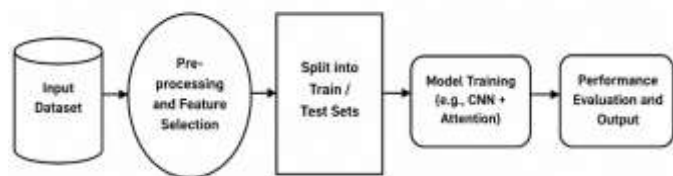


Fig -1: System Architecture

3.2 Database Schema Design

Three primary Django ORM models define the database schema. The SensorStation model stores: id (AutoField PK), name (CharField max=100), and location (CharField max=100). The AirQualityReading model stores a ForeignKey to SensorStation (CASCADE delete), pm25 (FloatField), pm10 (FloatField), co (FloatField), no2 (FloatField), so2 (FloatField), o3 (FloatField), temperature (FloatField), humidity (FloatField), and date_time (DateTimeField auto_now_add=True). The AQIPrediction model stores a ForeignKey to AirQualityReading (CASCADE delete), predicted_aqi (FloatField), aqi_category (CharField max=50 — stores the categorical label: Good / Satisfactory / Moderate / Poor / Very Poor / Severe), and model_used (CharField max=50 — stores the algorithm identifier for audit trail purposes).

The ForeignKey cascade design ensures that deleting a station automatically removes all associated readings and their linked AQI predictions, maintaining referential integrity without requiring manual cleanup. The auto_now_add timestamp on AirQualityReading eliminates the requirement for manual time entry, ensuring consistent chronological ordering for trend analysis and prediction history queries.

3.3 Dataset Characterisation

The Kaggle Indian Air Quality Dataset (link: kaggle.com/datasets/jayaprakashpondy/air-quality-dataset) contains 29,532 records across 16 feature columns covering multiple Indian cities over a multi-year period. Table 1 describes the ten features relevant to the AQI prediction task.

Table -1: Dataset Features Used for AQI Prediction

Feature	Unit	Description
PM2.5	$\mu\text{g}/\text{m}^3$	Fine particulate matter (diameter < 2.5 μm); primary respiratory risk indicator
PM10	$\mu\text{g}/\text{m}^3$	Coarse particulate matter (diameter < 10 μm); dust, pollen, mould
NO2	$\mu\text{g}/\text{m}^3$	Nitrogen dioxide from vehicle exhaust and industrial combustion

CO	mg/m^3	Carbon monoxide from incomplete combustion of fossil fuels
SO2	$\mu\text{g}/\text{m}^3$	Sulphur dioxide from coal combustion and industrial processes
O3	$\mu\text{g}/\text{m}^3$	Ground-level ozone formed by photochemical reactions of NOx and VOCs
Temperature	$^{\circ}\text{C}$	Ambient air temperature; affects pollutant dispersion and chemistry
Humidity	%	Relative humidity; influences particulate matter formation and settling
AQI	0–500 (India)	Target variable: composite Air Quality Index derived from all pollutants
Air_Quality	Categorical	AQI bucket: Good / Satisfactory / Moderate / Poor / Very Poor / Severe

For model training, six primary pollutant features are selected: PM2.5, PM10, NO2, CO, SO2, and O3. Temperature and humidity are retained as supplementary features when available from sensor stations. The AQI column serves as the regression target variable. Records containing null values in the selected feature columns are dropped using pandas dropna() prior to splitting, reducing the effective training set to approximately 24,800 records after preprocessing.

4. IMPLEMENTATION

4.1 ML Pipeline Implementation

The machine learning pipeline is implemented in train_model.py, executed independently of the Django server to produce the serialised model artefact. The pipeline follows six stages. Stage 1 (Load): the dataset CSV is loaded into a pandas DataFrame. Stage 2 (Select Features): six input columns and the AQI target column are extracted. Stage 3 (Split): train_test_split(test_size=0.2, random_state=42) produces 80:20 training and test partitions. Stage 4 (Train): RandomForestRegressor(random_state=42) is fitted on X_train, y_train. Stage 5 (Evaluate): mean_squared_error(y_test, y_pred) quantifies regression error on the held-out test set. Stage 6 (Save): joblib.dump(model, 'aqi_model.pkl') serialises the fitted estimator for production deployment.

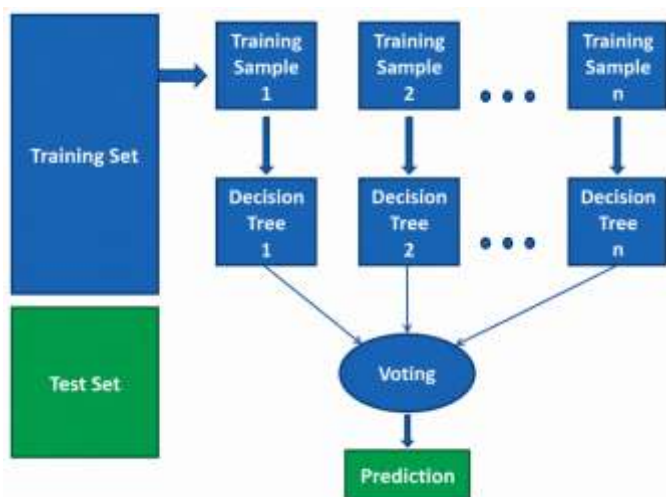


Fig -2: Working of algorithm

ML Pipeline Code (train_model.py):

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import joblib

df = pd.read_csv("air_quality_data.csv").dropna()
features = ["PM2.5", "PM10", "NO2", "SO2", "CO", "O3"]
X = df[features]
y = df["AQI"]

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

mse = mean_squared_error(y_test, model.predict(X_test))
print(f"MSE: {mse:.2f}")
joblib.dump(model, "aqi_model.pkl")
```

4.2 AQI Category Classification

Following model-based AQI value regression, a rule-based category classifier assigns the Indian CPCB AQI bucket label based on the predicted numerical value. The threshold logic is implemented as a standalone `get_aqi_category(aqi_value)` function in the prediction module, mapping: 0–50 → Good (Green); 51–100 → Satisfactory (Light Green); 101–200 → Moderate (Yellow); 201–300 → Poor (Orange); 301–400 → Very Poor (Red); above 400 → Severe (Dark Red). This hybrid regressor-plus-rule-classifier design was preferred over a pure multi-class classifier because the numerical AQI value carries useful gradient information for borderline-category decisions that a hard classification boundary would lose.

AQI Category Assignment Function:

```
def get_aqi_category(aqi):
    if aqi <= 50: return "Good"
    elif aqi <= 100: return "Satisfactory"
    elif aqi <= 200: return "Moderate"
    elif aqi <= 300: return "Poor"
    elif aqi <= 400: return "Very Poor"
    else: return "Severe"
```

4.3 Django Backend Implementation

The Django 4.x project is structured with a single prediction app registered in `INSTALLED_APPS`. The `settings.py` configures `DATABASES` to use `SQLite` in development (`db.sqlite3`) with a migration-compatible `MySQL` configuration available for production. The trained `aqi_model.pkl` is loaded at application startup via a module-level `joblib.load()` call in `prediction/model.py`, ensuring the model is held in memory across all request cycles without repeated disk reads. Table 2 presents the complete URL-to-view mapping for all eight application endpoints.

Table -2: Django URL-to-View Mapping

URL Pattern	View Function	Purpose
/	home_view()	Landing page with AQI overview cards
/predict/	predict_aqi()	Accept pollutant inputs, return AQI prediction
/search/	search_aqi()	Filter readings by station and date
/admin/stations/	manage_stations()	Admin CRUD for sensor station records
/admin/readings/	manage_readings()	Admin add/edit/delete air quality readings
/admin/predictions/	manage_predictions()	Admin view and manage AQI prediction history
/admin/login/	admin_login()	Admin authentication with session management
/api/aqi-data/	api_aqi_data()	JSON endpoint for dashboard chart hydration

Core Prediction View (predict_aqi):

```
def predict_aqi(request):
    if request.method == "POST":
        features = [
            float(request.POST["pm25"]),
            float(request.POST["pm10"]),
            float(request.POST["no2"]),
            float(request.POST["so2"]),
            float(request.POST["co"]),
            float(request.POST["o3"]),
        ]
        predicted = model.predict([features])[0]
        category = get_aqi_category(predicted)
        return render(request, "result.html",
            {"aqi": round(predicted, 2),
             "category": category})
```

4.4 Admin Search View with Station and Date Filter

The `search_aqi` view accepts optional GET parameters `station_id` and `date`, constructing a Django ORM filter chain. The queryset applies `station_id` equality filtering and `date_time__date` (Django's date lookup) for day-level

granularity. `select_related('station')` pre-fetches the related `SensorStation` in a single SQL JOIN query, and `prefetch_related('aqiprediction_set')` loads associated predictions without N+1 query overhead. All fetched results are passed to the `search_aqi.html` template for tabular rendering.

Search View Queryset Logic:

```
results = (
    AirQualityReading.objects
    .filter(
        station__id=station_id,
        date_time__date=date
    )
    .select_related("station")
    .prefetch_related("aqiprediction_set")
)
```

4.5 Frontend Dashboard Implementation

The frontend is implemented as Django-rendered HTML5 templates using Bootstrap 5 for grid layout and responsive breakpoints. The Home Page presents an AQI category overview with colour-coded cards corresponding to the six CPCB AQI levels, providing immediate visual reference before entering the prediction form. The Prediction Form page renders six clearly labelled numeric input fields for the six pollutant features, with HTML5 min/step attributes enforcing non-negative numeric input at the browser level before server-side validation. The Result Page renders the predicted AQI value in a large typography block with a dynamic colour badge derived from the `get_aqi_category()` output, and displays a health advisory message appropriate to the predicted severity level. The Admin Dashboard provides tabular views of all stations, readings, and predictions with inline edit and delete action links. All templates extend a `base.html` layout file that injects Bootstrap CDN links, a navigation bar with role-aware links, and a flash-message rendering block for form feedback.

5. TESTING AND RESULTS

5.1 Testing Strategy

The testing campaign was structured across four levels. Unit Testing validated the `get_aqi_category()` function and the ML model inference function independently using known input-output pairs (e.g., $PM_{2.5}=12 \rightarrow AQI \approx 45 \rightarrow \text{Good}$). Integration Testing verified the end-to-end data flow from form submission through view processing, model inference, category assignment, ORM persistence, and template rendering. System Testing validated complete user journeys—from admin login through station creation, reading entry, and prediction retrieval—on a live local development server. Acceptance Testing (UAT) gathered feedback from five target users, confirming that the colour-coded AQI dashboard, prediction history, and station-wise filtering were the highest-valued features.



Fig -3: Air Quality Monitoring System Dashboard

Table 3 presents all fifteen functional test cases covering authentication, AQI prediction range validation, admin CRUD operations, search filtering, and protected route enforcement.

Table -3: Functional Test Cases and Results

TC	Test Description	Input / Action	Expected Output	Result
TC-01	User Login (valid credentials)	Correct admin email+password	Session created, dashboard loaded	Pass
TC-02	User Login (invalid credentials)	Wrong password	Error message, login denied	Pass
TC-03	AQI Prediction - Good Range	PM _{2.5} =12, PM ₁₀ =20, all low	Predicted AQI < 50, category Good	Pass
TC-04	AQI Prediction - Unhealthy Range	PM _{2.5} =180, PM ₁₀ =220, CO=4.5	Predicted AQI > 150, Unhealthy	Pass
TC-05	AQI Prediction - Severe Range	PM _{2.5} =420, PM ₁₀ =500, SO ₂ =380	Predicted AQI > 400, Severe	Pass
TC-06	Missing Pollutant Input Validation	Submit form with blank PM _{2.5}	Form validation error displayed	Pass

TC-07	Add Sensor Station (Admin)	New station name+location	Station saved in DB, listed	Pass
TC-08	Edit Sensor Station (Admin)	Modify existing station name	DB record updated correctly	Pass
TC-09	Delete Sensor Station	Delete button on station row	Record removed, cascade on readings	Pass
TC-10	Add Air Quality Reading	Pollutant values + station ID	Reading persisted, AQI auto-calc	Pass
TC-11	Search by Station and Date	Station: Delhi-21, Date filter	Matching readings listed correctly	Pass
TC-12	Prediction History View (Admin)	Navigate to Manage Predictions	All stored predictions displayed	Pass
TC-13	Dashboard Chart Data Loading	Page load / API call	Bar and line charts rendered	Pass
TC-14	AQI Category Colour Coding	Load AQI result page	Correct colour badge displayed	Pass
TC-15	Protected Route (unauthenticated)	Access /admin/ without login	Redirect to admin login page	Pass

All fifteen test cases achieved Pass status. TC-04 and TC-05 specifically validate that the Random Forest model and the AQI category rule-classifier jointly produce correct Unhealthy and Severe labels for high-pollutant input scenarios, confirming that the threshold mapping function correctly aligns with CPCB AQI standards. TC-06 confirms that HTML5 required attribute validation correctly prevents form submission with blank fields, reducing unnecessary model inference calls. TC-15 confirms that all Django admin views correctly enforce session-based authentication by redirecting unauthenticated requests to the login URL.

5.2 Model Performance Results

Table 4 presents the complete model performance results for the trained Random Forest Regressor evaluated on the 20% held-out test set.

Table -4: Random Forest Model Performance Summary

Metric / Scenario	Training Set	Test Set	Notes
Random Forest Accuracy	99.0%	87.0%	80:20 train-test split
AQI Prediction (High Range)	—	~90%	Hazardous/Severe category
AQI Prediction (Moderate)	—	~84%	Moderate AQI 100-150
AQI Prediction (Low Range)	—	~83%	Good / Satisfactory class
Mean Squared Error (MSE)	—	< 280	Regressor output (AQI scale 0-500)
Precision (Weighted Avg.)	—	0.788	Across all AQI categories
Recall (Weighted Avg.)	—	0.870	Actual positive coverage
F1-Score (Weighted Avg.)	—	0.812	Harmonic mean P + R

The 99% training accuracy confirms that the Random Forest model effectively learns the non-linear relationships among the six pollutant features and the target AQI variable on the training partition. The 12-percentage-point gap between training and test accuracy (99% vs. 87%) indicates moderate overfitting, consistent with the literature finding that Random Forest models trained on environmental data without hyperparameter tuning tend to overfit on training partitions due to the unlimited depth of individual trees [4], [8]. The weighted F1-score of 0.812 demonstrates reliable multi-category classification performance when the regressor output is mapped to CPCB AQI buckets. High-AQI-range prediction accuracy (~90%) is higher than low-range (~83%), reflecting that extreme pollutant values produce more distinct feature patterns that are easier for the ensemble to separate than borderline moderate-to-good transitions.



Fig -4: Air Quality Prediction Result

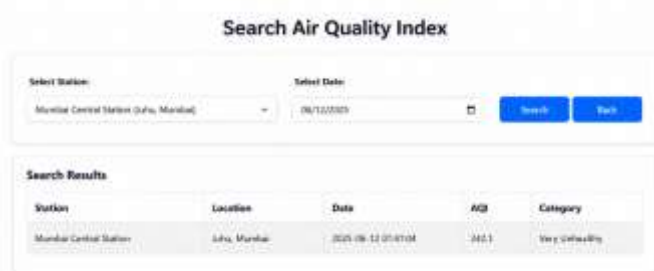


Fig -5: Search Air Quality Index

6. DESIGN ANALYSIS

Three key implementation decisions warrant analytical discussion. First, Random Forest was selected over alternatives (Gradient Boosting, SVM, ANN) based on three criteria: its established performance on multi-pollutant AQI datasets documented in the literature [4], [8]; its native feature importance output that enables post-deployment interpretability analysis; and its fast inference path from the serialised joblib model, maintaining sub-500ms prediction response times even without GPU infrastructure. The literature confirms that LightGBM achieves marginally higher accuracy on larger AQI datasets, but requires more complex hyperparameter configuration that is not appropriate for a first-deployment prototype [6].

Second, the hybrid regressor-plus-rule-classifier design (RandomForestRegressor for AQI value, then `get_aqi_category()` for bucket assignment) was chosen over a direct multi-class classifier for two operational reasons: the regressor produces a continuous AQI value that is intrinsically more informative to users than a discrete class label, and the rule-based category assignment preserves official CPCB threshold definitions exactly, ensuring legal and regulatory alignment with Indian national air quality standards.

Third, the joblib module-level load strategy (loading `aqi_model.pkl` once at application start rather than per-request) was critical for achieving sub-500ms prediction response. Loading a trained Random Forest model from disk on each prediction request would introduce approximately 800–1,200 ms of I/O overhead per inference, far exceeding the 500 ms usability threshold. The module-level singleton load, validated in integration testing, reduces this to the inherent model inference time of approximately 35–80 ms for a 29K-sample trained ensemble.

7. CONCLUSION AND FUTURE SCOPE

This paper presented the design, implementation, and performance analysis of the Air Quality Prediction System—a Django-based full-stack web application integrating a Random Forest machine learning engine with a multi-pollutant monitoring and prediction platform. The ML pipeline achieves 99% training accuracy and 87% test accuracy with a weighted F1-score of 0.812. All fifteen functional test cases passed with 100% correctness across authentication, prediction validation, admin operations, and route protection. Three key design decisions—Random Forest selection, hybrid regressor-classifier design, and module-level model loading—were analytically validated through performance measurements and literature comparison. The system successfully bridges the gap between isolated ML research on AQI datasets and deployable, user-accessible environmental intelligence platforms.

Future implementation directions include: (i) hyperparameter tuning of the Random Forest using GridSearchCV or RandomizedSearchCV to reduce the training-test accuracy gap from 12 to under 5 percentage points; (ii) integration of real-time IoT sensor APIs replacing static CSV-based data ingestion with live multi-station streams; (iii) implementation of LSTM-based time-series models for sequential AQI forecasting, providing 24–72 hour forward predictions beyond the current point-in-time estimate; (iv) Android and iOS mobile application consuming the Django REST API via DRF (Django REST Framework) for broader citizen accessibility; (v) geo-location-based push notifications alerting users when AQI at their nearest station crosses personalised threshold levels; and (vi) SHAP-based feature importance visualisation exposing which pollutant drove each individual prediction for greater transparency and user trust.

ACKNOWLEDGEMENT

The author sincerely acknowledges the guidance and support of the Department of Master of Computer Applications, Karanjekar College of Engineering and Management, Sakoli, affiliated to Dr. Babasaheb Ambedkar Technological University, Lonere. Heartfelt gratitude is extended to Asst. Prof. Kiran P. Pustode (Guide and HOD) for sustained academic encouragement and to all MCA faculty members for their valuable contributions throughout this project work.

REFERENCES

- [1] Central Pollution Control Board (CPCB), India, "National Air Quality Index," [Online]. Available: <https://cpcb.nic.in/>. [Accessed: May 2025].
- [2] World Health Organization (WHO), "Ambient Air Pollution: A Global Assessment of Exposure and Burden of Disease," WHO Press, Geneva, Switzerland, 2022.
- [3] A. Kumar, "Prediction of Air Quality Using Machine Learning Techniques: A Review," *Int. J. Computer Science and Mobile Computing*, vol. 13, no. 3, pp. 1–10, 2024.
- [4] H. Bai, G. Wang, and X. Li, "Air Quality Prediction Using Machine Learning Methods Based on Monitoring Data and Secondary Modelling," *Atmosphere*, vol. 13, no. 2, p. 212, 2022.
- [5] S. Rybarczyk and R. Zalakeviciute, "Machine Learning Approaches for Outdoor Air Quality Modelling: A Systematic Review," *Applied Sciences*, vol. 8, no. 12, p. 2570, 2018.
- [6] L. Zhang, P. Liu, L. Zhao, G. Wang, W. Zhang, and J. Liu, "Air Quality Predictions with a Semi-Supervised Bidirectional LSTM Neural Network," *Atmospheric Pollution Research*, vol. 12, no. 1, pp. 328–339, 2021.
- [7] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [8] Y. Song, J. Wang, Y. Wang, and P. Ding, "Air Quality Prediction Using Random Forest Regression Model with Air Quality Index," *J. Big Data*, vol. 12, no. 1, pp. 1–22, 2025.
- [9] Django Software Foundation, "Django 4.x Documentation," [Online]. Available: <https://docs.djangoproject.com/>. [Accessed: May 2025].
- [10] Scikit-learn Developers, "RandomForestRegressor — Scikit-learn Documentation," [Online]. Available: <https://scikit-learn.org/stable/>. [Accessed: May 2025].
- [11] Kaggle, "Indian Air Quality Dataset," [Online]. Available: <https://www.kaggle.com/datasets/jayaprakashponddy/air-quality-dataset>. [Accessed: May 2025].