Air Quality Prediction using ML Techniques

Manas Goyal (2022MT11918) Nilay Sharma (2022MT12007)
Pratyush Sharma (2022MT61970) Vagesh Mahajan (2022MT11260)

Abstract

This project report presents a comprehensive framework for air quality prediction that combines advanced machine learning techniques with privacy preserving and virtual monitoring strategies. We leverage a publicly available hourly air quality dataset—encompassing $PM_{2.5}, PM_{10}, CO, NO, NO_2$, benzene, and meteorological variables augmented with engineered temporal features (hour, day, month, weekday). Rigorous preprocessing includes missing value imputation, standard scaling, and exploratory analyses (distribution, correlation, time series, and outlier detection) to inform model design.

Fourteen predictive models were implemented and evaluated: ensemble methods (Random Forest, Extra Trees, CatBoost, XGBoost, LightGBM, AdaBoost), linear regressions (Linear, Ridge, Lasso), kernel-based (SVM), tree-based (Decision Tree), statistical time series (Seasonal ARIMA), neural networks (GRU), and a hybrid ARIMA–DBO–RF approach that integrates AutoRegressive Integrated Moving Average with Dung Beetle Optimization for hyperparameter tuning of Random Forest. Performance metrics (MAE, RMSE, R^2) reveal that tree-based ensemble models uniformly outperform simpler linear and seasonal methods, with Extra Trees and Random Forest achieving the best balance of accuracy and generalization across NO_2, CO, and benzene forecasts. Finally we presented a discussion on the future directions.

1 Introduction

Air quality prediction is vital for both public health and environmental management. This report investigates the application of machine learning models for predicting air quality indices, utilizing a combination of traditional monitoring data and virtual sensing techniques. To address data privacy concerns, the integration of federated learning is explored.

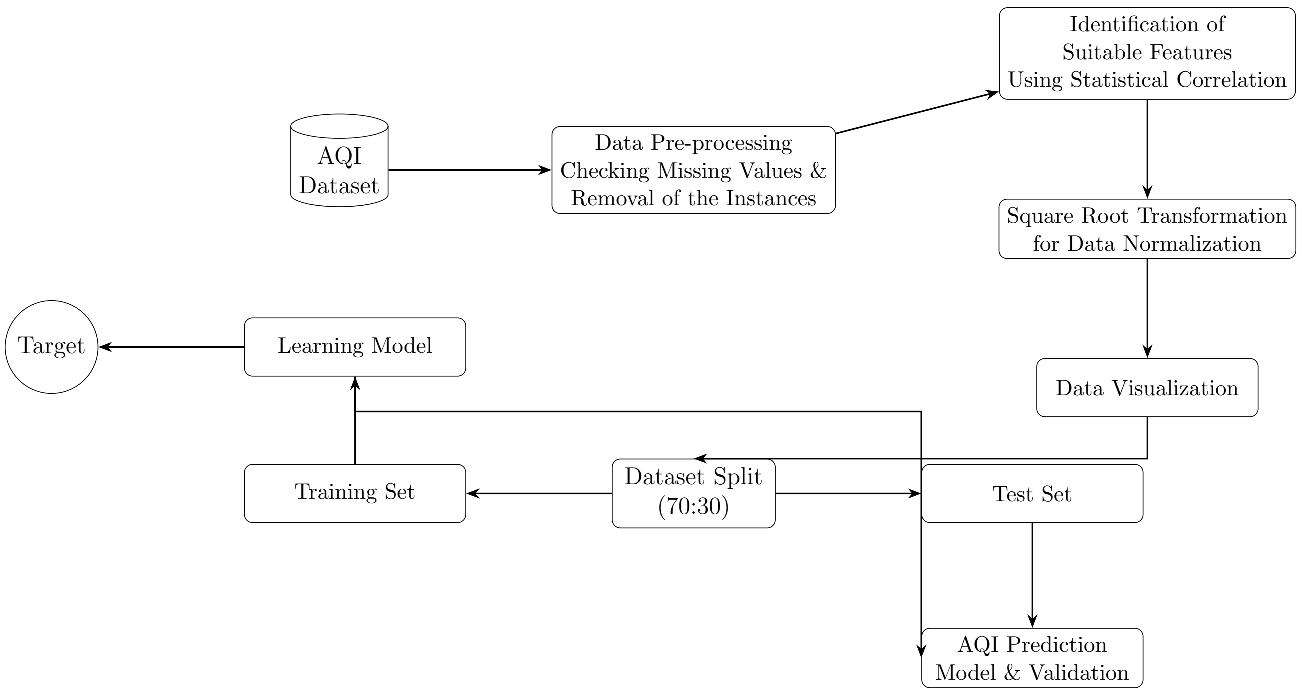
Using an hourly dataset comprising pollutant concentrations ($PM_{2.5}$, PM_{10}, CO, NO_x, NO_2, benzene) and meteorological variables, the data preprocessing pipeline includes imputation, scaling, and temporal feature engineering (hour, day, month, and weekday). Fourteen models are evaluated using MAE, RMSE, and R^2 metrics. Tree-based ensemble methods, such as Extra Trees and Random Forest, demonstrate superior performance across most pollutants, with pollutant-specific explanations provided for model effectiveness.

Federated learning is shown to effectively preserve data privacy by enabling decentralized model training without raw data exchange—an essential aspect for secure data collaboration. Virtual monitoring further enhances spatial coverage and contributes to improved pollution prediction. Future directions include optimizing federated algorithms, incorporating satellite and IoT data, and developing hybrid machine learning models for greater predictive accuracy.

2 Literature Review

2.1 Machine Learning Models for Air Quality Prediction

The paper [1] provides an extensive analysis of machine learning models applied to air quality prediction, focusing on the prediction of PM2.5 levels. The paper evaluates several models, including Support Vector Machines (SVM), Random Forests, and Deep Neural Networks (DNNs), and discusses their respective strengths and weaknesses.


AQI Dataset
Data Pre-processing Checking Missing Values & Removal of the Instances
Identification of Suitable Features Using Statistical Correlation
Square Root Transformation for Data Normalization
Data Visualization
Dataset Split (70:30)
Training Set
Test Set
Learning Model
Target
AQI Prediction Model & Validation

- **Feature Engineering:** The authors emphasize the critical role of feature engineering in improving model performance. They explore various features such as historical pollutant levels, meteorological data (temperature, humidity, wind speed), and temporal features (time of day, day of the week). The study demonstrates that incorporating these features significantly enhances the predictive power of the models.

- **Model Complexity and Architecture:** The paper highlights the superior performance of DNNs, attributing it to their ability to model complex, non-linear relationships in the data. The authors detail the architecture of the neural networks used, including the number of hidden layers, neurons per layer, activation functions, and optimization algorithms. They conduct experiments to determine the optimal network configuration for their dataset.
- **Cross-Validation and Hyperparameter Tuning:** To ensure the robustness and generalizability of the models, the study employs k-fold cross-validation. The authors also discuss the process of hyperparameter tuning, using techniques such as grid search and random search to optimize model parameters like learning rate, batch size, and regularization terms.

2.2 Federated Learning for Privacy-Preserving Air Quality Monitoring

The paper [2] explores the innovative use of federated learning to address privacy concerns in air quality monitoring. The paper introduces a federated learning framework that enables multiple organizations to collaboratively train a global model without sharing raw data.

- **Decentralized Training Process:** The federated learning framework allows each participant (e.g., a city or organization) to train a local model on its own data. Only the model updates (e.g., gradients) are shared with a central server, which aggregates them to update the global model. This decentralized approach ensures that sensitive data remains on local devices.
- **Privacy Preservation Techniques:** The study discusses several techniques to enhance privacy, including differential privacy, which adds noise to the model updates to prevent the extraction of individual data points, and secure aggregation, which ensures that the server can only access aggregated updates.
- **Communication Efficiency:** To address the challenge of communication overhead, the authors propose methods to reduce the frequency and size of model updates. These include model compression techniques, such as quantization and pruning, and selective update strategies, where only significant updates are communicated.

2.3 Virtual Monitoring as an Alternative to Traditional Monitoring

Virtual monitoring offers a promising alternative to traditional ground-based air quality monitoring by utilizing advanced technologies such as remote sensing, satellite imagery, and computational models. This approach can provide comprehensive spatial coverage and real-time data, addressing some of the limitations of physical monitoring stations.

2.3.1 Advantages of Virtual Monitoring

- **Broader Spatial Coverage:** Virtual monitoring can cover large geographic areas, including remote and inaccessible regions where ground-based sensors are sparse or non-existent. This is particularly beneficial for global air quality assessments and for regions with limited infrastructure.

- **Real-Time Data Acquisition:** Satellite and remote sensing technologies can provide near real-time data, allowing for timely analysis and response to air quality issues. This capability is crucial for dynamic modeling and forecasting.
- **Cost-Effectiveness:** Deploying and maintaining physical monitoring stations can be expensive and resource-intensive. Virtual monitoring reduces the need for extensive physical infrastructure, potentially lowering costs.

2.3.2 Methodologies for Virtual Monitoring

- **Remote Sensing and Satellite Imagery:** Satellites equipped with sensors can measure atmospheric pollutants and meteorological parameters. Data from sources like NASA's MODIS (Moderate Resolution Imaging Spectroradiometer) and ESA's Sentinel satellites can be used to estimate pollutant concentrations.
- **Computational Modeling:** Atmospheric models, such as the Community Multiscale Air Quality (CMAQ) model, simulate the transport and transformation of pollutants in the atmosphere. These models can be integrated with satellite data to enhance prediction accuracy.
- **Data Assimilation Techniques:** Combining satellite observations with model predictions through data assimilation techniques can improve the accuracy of air quality forecasts. Methods such as Kalman filtering and variational assimilation are commonly used.

2.3.3 Challenges and Considerations

- **Data Resolution and Accuracy:** Satellite data may have lower spatial and temporal resolution compared to ground-based measurements. Efforts are needed to improve the resolution and accuracy of remote sensing data.
- **Cloud Cover and Atmospheric Interference:** Satellite observations can be affected by cloud cover and atmospheric conditions, which may obscure measurements. Advanced algorithms are required to mitigate these effects.
- **Integration with Ground-Based Data:** While virtual monitoring provides extensive coverage, integrating it with ground-based data can enhance validation and calibration, ensuring more reliable predictions.

2.3.4 Virtual Monitoring and Real-World Data Challenges

The paper [3] addresses the challenges of using real-world environmental data for air quality prediction and the role of virtual monitoring in overcoming these challenges. The paper discusses several innovative approaches:

- **Data Integration from Multiple Sources:** The authors propose integrating data from diverse sources, including satellite imagery, remote sensing, and ground-based sensors. This approach provides a more comprehensive view of air quality, enhancing spatial coverage and allowing for the monitoring of areas with limited sensor infrastructure.
- **Advanced Preprocessing Techniques:** The study reviews various preprocessing techniques to handle common issues in real-world data, such as missing values, noise, and data heterogeneity. Techniques discussed include interpolation methods for missing data, noise reduction algorithms like wavelet transforms, and feature extraction methods such as principal component analysis (PCA) and independent component analysis (ICA).

- **Ensemble Learning for Robust Predictions:** To address the variability and uncertainty in environmental data, the paper suggests using ensemble learning techniques. These methods combine predictions from multiple models to produce a more robust and accurate final prediction. The authors explore different ensemble strategies, including bagging, boosting, and stacking, and evaluate their effectiveness in improving prediction accuracy.

3 Methodology

3.1 Data Collection

The dataset used in this study was obtained from the UCI Machine Learning Repository [4]. It comprises hourly averaged sensor responses from a chemical multisensor device deployed in an urban area in Italy. The data was collected over the course of one year, from March 2004 to February 2005, and includes 9357 instances.

The sensor device, placed at road level in a polluted area, recorded the responses of five metal oxide sensors targeting various air pollutants. Ground truth concentrations of gases such as CO, Non-Methane Hydrocarbons (NMHC), Benzene, NOx, and NO2 were measured using certified analyzers co-located with the sensor device.

Additionally, meteorological variables such as temperature, relative humidity, and absolute humidity were also logged. Missing values in the dataset are represented with the value `-200`, which must be appropriately handled during preprocessing. Notably, the dataset exhibits signs of cross-sensitivity, sensor drift, and concept drift, which present realistic challenges for machine learning models.

3.2 AQI Calculation

The AQI technique combines air pollutant concentrations into a single value in such a way that the general public can understand about air quality status. In order to create an AQI, there are two stages: (i) Sub-indices creation (for every pollutant) (ii) Aggregation of these sub-indices to get an overall AQI. Further, the sub-indices of each pollutant were calculated with the linear segmentation principle-based equation (see Eq.(1)). The relevant data used in this equation were taken from Table1. Thereafter, the maximum sub-indices value of any pollutants ($I_{PM10}, I_{PM2.5}, I_{NOX}, I_{NH3}, I_{SO2}, I_{CO}, I_{Ozone}$) represent the value of AQI.

$$I_P = \frac{I_{HI} - I_{LO}}{BP_{HI} - BP_{LO}}(C_P - BP_{LO}) + I_{LO} \tag{1}$$

Where:

- BP_{HI} = Breakpoint concentration $\geq C_P$
- BP_{LO} = Breakpoint concentration $\leq C_P$
- I_{HI} = Sub-Index value corresponding to BP_{HI}
- I_{LO} = Sub-Index value corresponding to BP_{LO}
- C_P = Observed concentration for pollutant P
- P = Any pollutant either of PM2.5, PM10, NO$_x$, CO

Table 1: Sub-index and Break-point concentrations of each pollutant given by CPCB

AQI Category Range	PM10 24-Hr	PM2.5 24-Hr	NOx 24-Hr	CO 8-Hr
Good (0-50)	0-50	0-30	0-40	0-1
Satisfactory (51-100)	51-100	31-60	41-80	1.1-2.0
Moderate (101-200)	101-250	61-90	81-180	2-10
Poor (201-300)	251-350	91-120	181-280	10-17
Very poor (301-400)	351-430	121-250	281-400	17-34

3.3 Data Preprocessing

Data preprocessing and exploratory data analysis (EDA) are essential steps in preparing and understanding the dataset before applying any machine learning models. This section details the preprocessing steps and EDA performed on the dataset.

Data Cleaning The initial step in data preprocessing involved cleaning the dataset by removing unnecessary columns and handling missing values. Specifically, columns that were not needed for analysis were dropped, and any rows with missing data were removed to ensure the dataset's integrity.

Time Formatting and DateTime Creation The *Time* column in the dataset contained dots instead of colons, which were replaced to ensure proper time formatting. Subsequently, a new *DateTime* column was created by combining the *Date* and *Time* columns. This new column was set as the index of the DataFrame, facilitating time-based analysis.

Data Imputation, Scaling and Normalization To handle any remaining missing values, median imputation was applied. This method replaces missing values with the median of each column, which is robust to outliers. After imputation, the dataset was scaled using standard scaling, which standardizes features by removing the mean and scaling to unit variance, ensuring that each feature contributes equally to the analysis. The normalization equation for standardizing a feature is given by:

$$x' = \frac{x - \mu}{\sigma} \tag{2}$$

where:

x' is the standardized value,

x is the original value,

μ is the mean of the feature,

σ is the standard deviation of the feature.

Feature Engineering Additional temporal features were engineered from the *DateTime* index to enhance the dataset. These features included the hour, day, month, and day of the week, which can provide valuable insights into temporal patterns and trends within the data.

Table 2: Descriptive statistics of pollutants

Statistic Parameters	CO(GT)	PT08.S1(CO)	C6H6(GT)	NOx(GT)
Mean	-34.21	1048.99	1.87	168.62
Standard Deviation	77.66	329.83	41.38	257.43
Kurtosis	3.78	8.83	22.18	4.50
Skewness	-1.67	-1.72	-4.51	0.83
Count	9357	9357	9357	9357
Confidence Level (95%)	1.57	6.68	0.84	5.22
Statistic Parameters	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	
Mean	794.99	58.15	1391.48	
Standard Deviation	321.99	126.94	467.21	
Kurtosis	6.10	3.27	6.26	
Skewness	-0.38	-1.23	-1.24	
Count	9357	9357	9357	
Confidence Level (95%)	6.52	2.57	9.47	

Exploratory Data Analysis (EDA) EDA was conducted to understand the distribution and relationships within the dataset. Various plots were generated to visualize the data:

- **Distribution Analysis:** Histograms with kernel density estimates were created for each variable to visualize their distributions and identify any skewness or outliers.

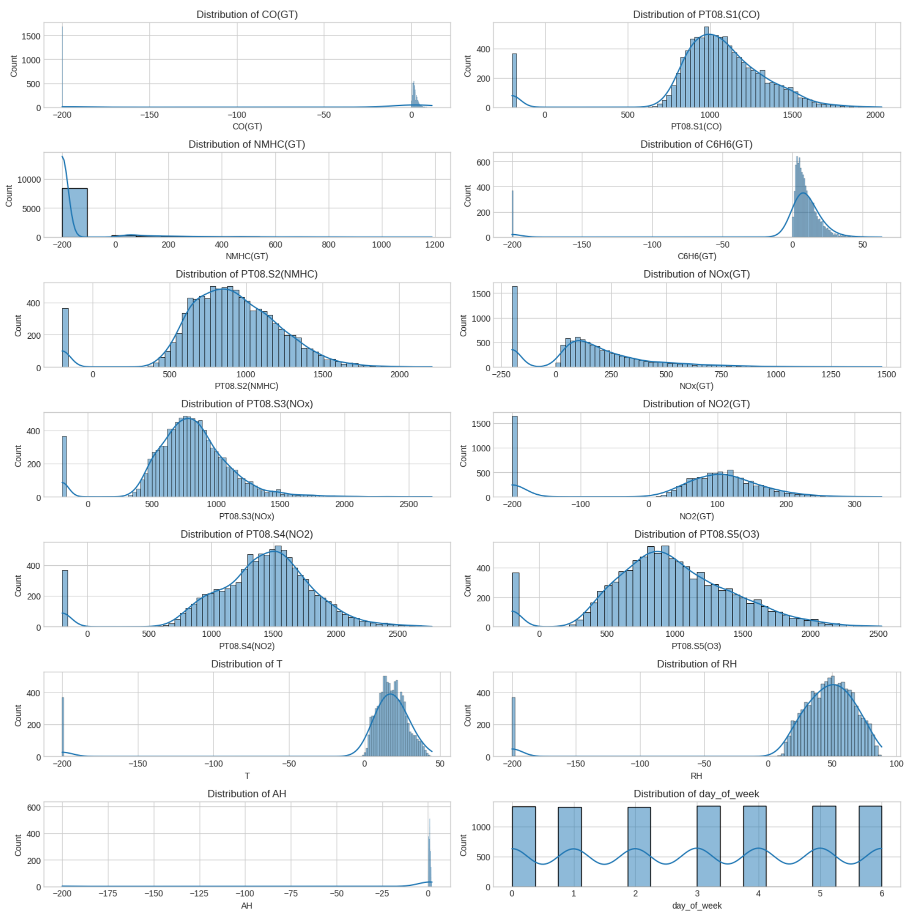


Figure 1: Distribution Analysis of Variables

- **Correlation Analysis:** A correlation matrix was computed and visualized using a heatmap to identify relationships between different variables. This helps in understanding which variables are strongly correlated and may influence each other.

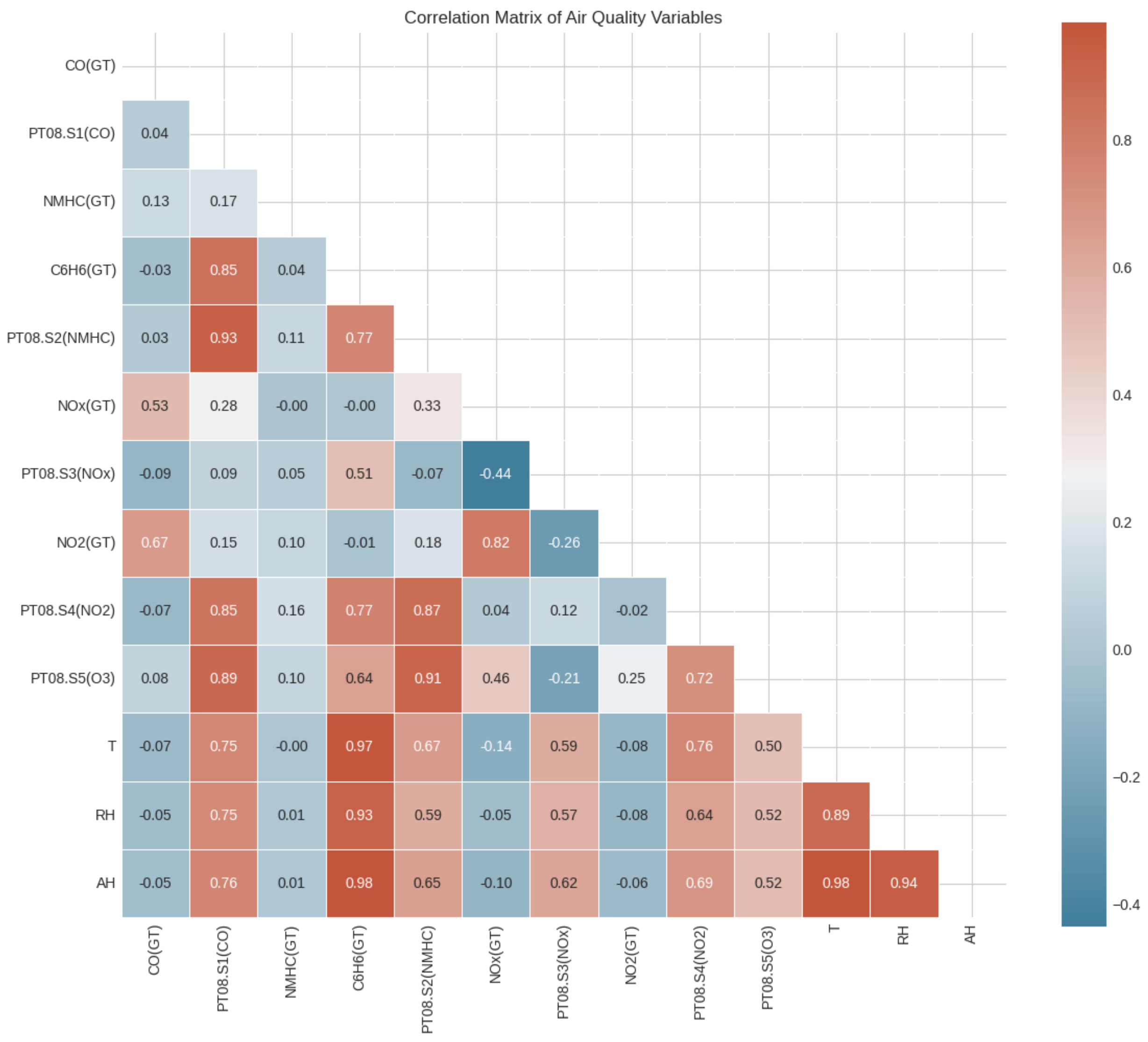


Figure 2: Correlation Matrix Heatmap

- **Time Series Analysis:** Daily average levels of key pollutants such as CO, Benzene, NOx, and NO2 were plotted to observe trends over time. This analysis helps in identifying patterns and anomalies in air quality data.

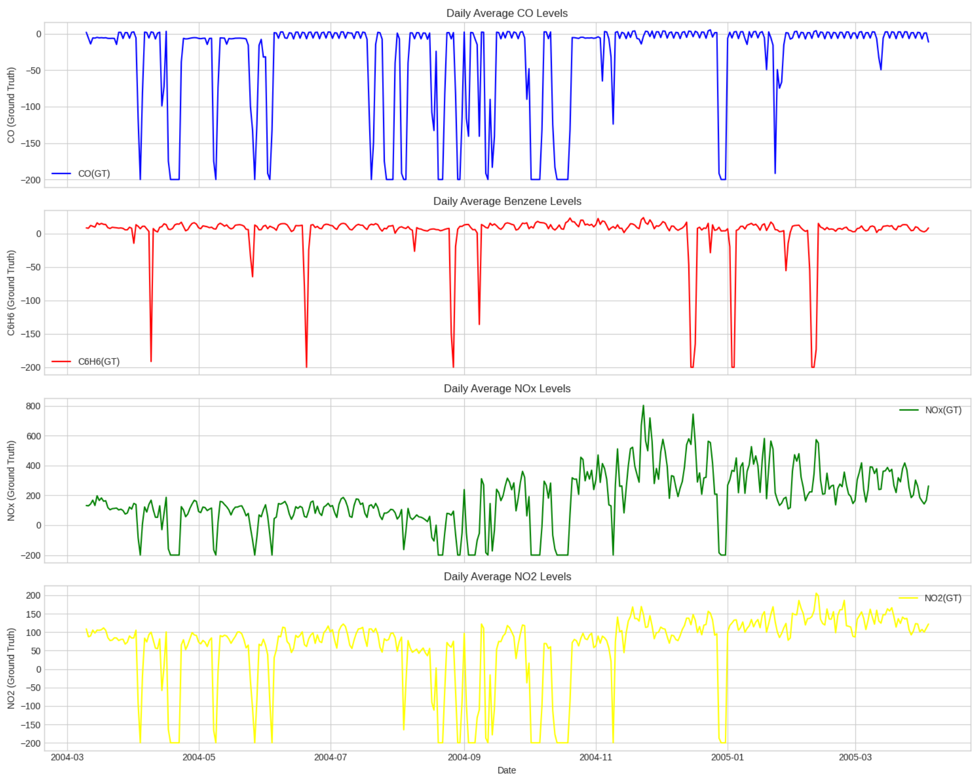


Figure 3: Time Series Analysis of Pollutants

- **Box Plots:** Box plots were used to visualize the spread and identify outliers in the air quality variables.

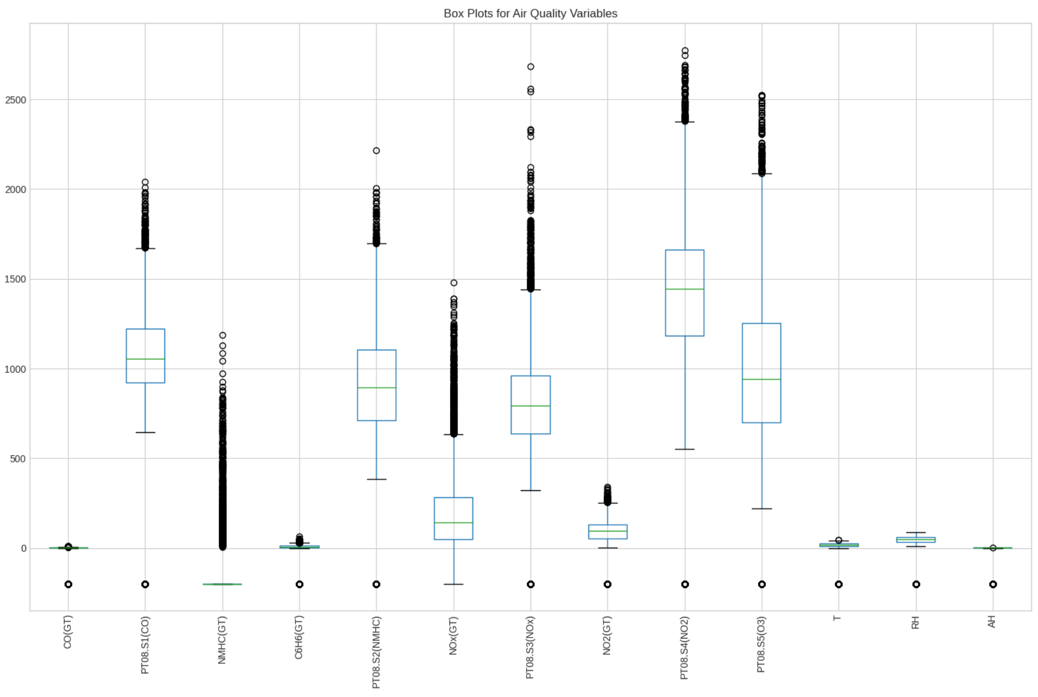


Figure 4: Box Plots for Air Quality Variables

- **Scatter Plots:** Scatter plots were created to explore the relationships between weather variables (e.g., temperature, relative humidity) and pollutant levels, along with calculating correlation coefficients.

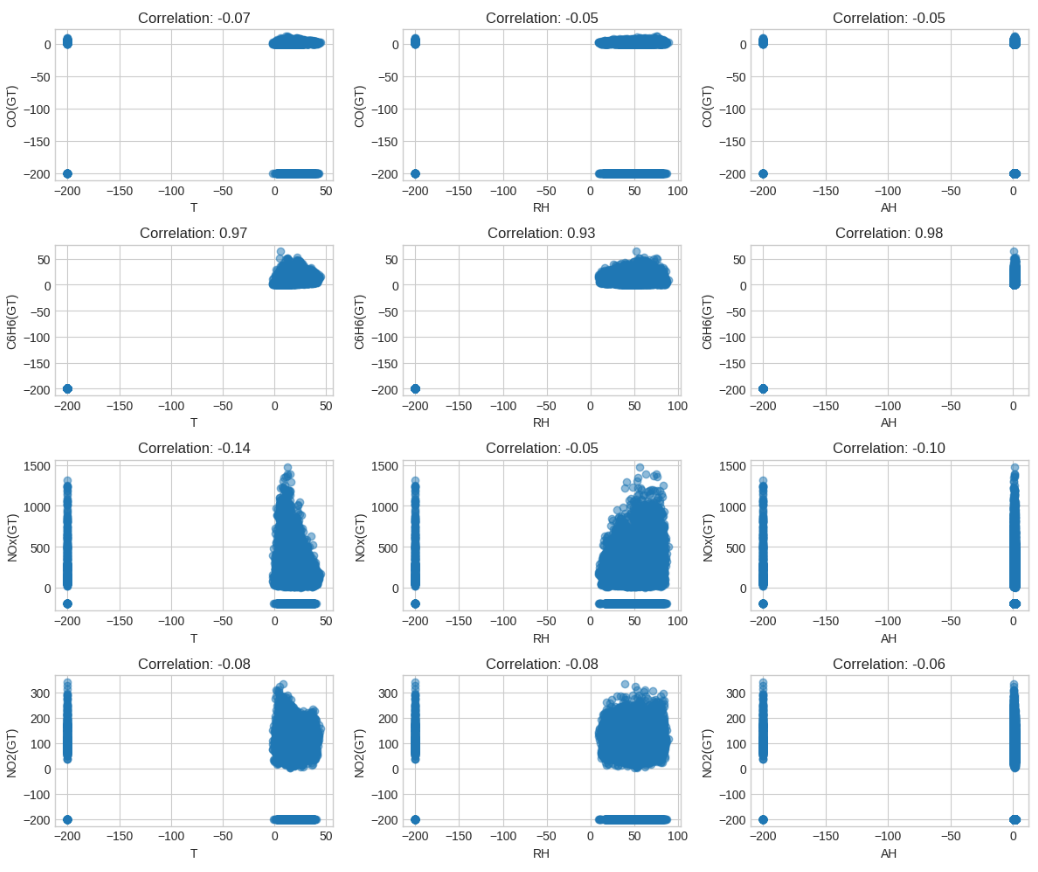
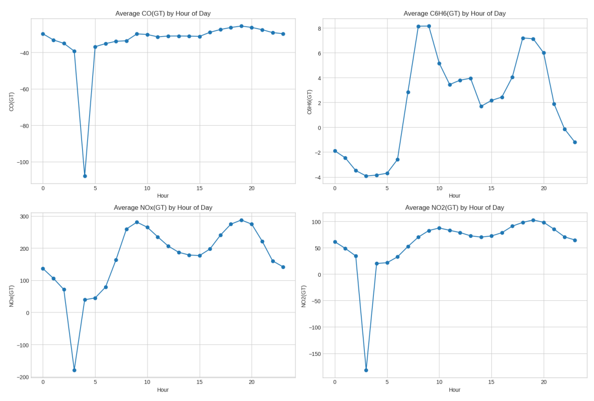


Figure 5: Scatter Plots of Weather Variables vs. Pollutants

- **Hourly and Weekly Patterns:** The dataset was analyzed to identify average pollutant levels by hour of the day and day of the week, providing insights into daily and weekly cycles in air quality.



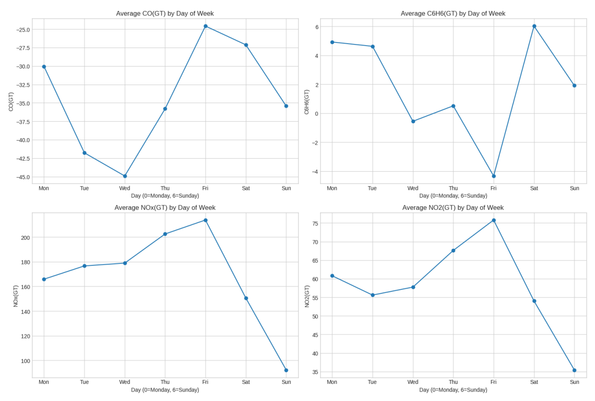


Figure 6: Hourly Patterns for Pollutants

Figure 7: Weekly Patterns for Pollutants

3.4 Model Implementation

We implemented several ML models :

3.4.1 LightGBM

A trustworthy tool for implementing gradient boosting in decision trees is LightGBM. LightGBM uses tree-based learning strategies, making it a suitable choice for gradient boosting. It provides quicker training and higher production thanks to its decentralised and effective architecture. A histogram-based method called LightGBM conducts variable bucketing, which improves training speed and accuracy while using less memory. When being trained, it works more quickly and can handle large and complex datasets. Support for both parallel and GPU-based learning. The method enables one to infer details about a target Y given only X as input when used in supervised learning scenarios. The LightGBM technique uses a supervised training set (X) and a loss function $L(y, f(X))$ whose predicted value is to be reduced to accomplish this $\hat{f}(x)$.

$$\hat{f} = \arg\min_{f} \mathbb{E}_{y,x} L\left(y, f(x)\right) \tag{3}$$

3.4.2 Random Forest

Random forest regression is one form of machine learning-based regression technique. The foundation of this tactic is a combination of bagging and random subsample approaches. Following the bagging of numerous learning trees, an ensemble method is used to merge these trees into a single forecast. From the entire collection of N examples used for training (D), n random instances are chosen to create a bootstrap sample (D_b). It is acceptable to substitute example sets when creating bootstrap samples. When regression tasks, the random forest prediction is made by averaging K predictions from regression trees $h_k(x)$.

$$\text{Random forest prediction} = \frac{1}{K}\sum_{k=1}^{K} h_k(x) \tag{4}$$

3.4.3 CatBoost

A development of the gradient boosting and decision tree frameworks is CatBoost. Boosting is predicated on the idea that numerous, relatively weak models can be joined to create a single, fiercely competitive prediction model that outperforms chance by a slight margin. Gradient boosting decreases errors by fitting a series of decision trees, each of which gains knowledge from the mistakes of the preceding iteration. This process of introducing new functions into the mix is repeated until the chosen loss function is no longer minimised. CatBoost's method for creating decision trees differs from traditional gradient boosting models. Instead, CatBoost creates "oblivious trees," which are made possible by requiring that all nodes at the same level test the same predictor under the same conditions. This allows for the computation of a leaf's index using just bitwise operations.

By arbitrary ordering the components of D using a random permutation, σ, CatBoost chooses the data to be utilised for fitting h^{t+1}. $D_k = \{x_1, x_2, ..., x_{k-1}\}$ where $x_1, x_2, ..., x_{k-1}$ are the elements of D arranged by the random permutation and (k) is the kth element of D under permutation, σ. Instead of strictly adhering to Eq. (3), CatBoost uses a variation of it in its analysis to define the encoded value, $\overline{x}^{\sigma}_{ik}$ for the ith categorical value during Decision Tree h^{t+1} fitting.

$$\overline{x}^{\sigma}ik = \frac{\sum x_j \in D_1, x^{\sigma}jk = x^{\sigma}ik, j \neq iy_j + a}{\sum_{x_j \in D_1, x^{\sigma}jk = x^{\sigma}ik, j \neq i} 1 + a} \tag{5}$$

Here $1_{x^{\sigma}jk = x^{\sigma}ik}$ is the indicator function.

3.4.4 Adaptive Boosting (AdaBoost) Regressor

AdaBoost, one of the first boosting algorithms employed, was used to address difficulties. AdaBoost modifies the data of each training sample (x_i, y_i) by applying a weight $w_1, w_2, ..., w_N$. The fundamental learner gives each observation the same amount of thought in the initial stage. Once weights have been assigned to each observation, the weak learner may be utilised for prediction. In this method, the predictions made by the base learner after the weak learner are more likely to be accurate. This process will be carried out again until the tth iteration, after which the T_i base learning algorithm's limit will be reached. The outputs of weak learners can be combined to generate more robust learners, which enhance the ability to predict outcomes.

3.4.5 Extreme Gradient Boosting (XGBoost)

Boosting is a technique for combining numerous weak classifiers into a single effective one. Gradient Boosting served as the basis for the development of the technique known as XGBoost. Gradient Boosting's XGBoost variation outperforms the original in terms of computing efficiency, scalability, and generalisation performance. Data organization is of utmost importance while utilising XGBoost. All category data will be transformed into their numeric equivalents because XGBoost only takes numeric vectors as input. This encoding change can be made with a single hot encoding. The feature engineering and data purification stages come next. We can obtain the estimated model by using the universal function, as indicated by the following formula:

$$\hat{y}i^t = \sum k = 1^t f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i) \tag{6}$$

where,

- $\hat{y}_i^t$ — forecasts at the stage t
- $f_t(x_i)$ — a learner at stage t
- x_i — the input variable
- $\hat{y}_i^{t-1}$ — forecasts at the stage $t-1$

3.4.6 Ridge Regression

Ridge regression is a statistical regularization technique that corrects for overfitting on training data in machine learning models. Ridge regression assumes that the coefficients of the linear model can be shrunk towards zero to reduce model complexity. It adds a penalty term to the ordinary least squares (OLS) objective function, which is proportional to the sum of squared coefficients. This approach is particularly useful when dealing with multicollinearity among predictors. The penalty term, controlled by the regularization parameter λ, shrinks coefficients in proportion to their initial size, with high-value coefficients being penalized more than low-value ones. Ridge regression is carried out on the linear regression model where coefficients are estimated not by ordinary least squares, but by a ridge estimator that, albeit biased, has lower variance than the OLS estimator.

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{7}$$

3.4.7 Lasso Regressor

Lasso (Least Absolute Shrinkage and Selection Operator) is a regression analysis method that performs both variable selection and regularization. Unlike ridge regression, which shrinks coefficients toward zero but doesn't eliminate them completely, Lasso can reduce some coefficients to exactly zero, effectively performing feature selection. This makes the model more interpretable by identifying the most important predictors. Lasso was originally introduced in geophysics and later formalized by Robert Tibshirani. The method assumes that the coefficients of the linear model are sparse, meaning that few of them are non-zero. Though originally defined for linear regression, lasso regularization is easily extended to other statistical models including generalized linear models, generalized estimating equations, proportional hazards models, and M-estimators.

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{8}$$

3.4.8 Support Vector Machine (SVM)

Support Vector Machine regression is a popular machine learning tool first identified by Vladimir Vapnik and his colleagues in 1992. SVM regression is considered a nonparametric technique because it relies on kernel functions. It implements linear epsilon-insensitive SVM (ε-SVM) regression, which is also known as $L1$ loss. In ε-SVM regression, the goal is to find a function $f(x)$ that deviates from y_n by a value no greater than ε for each training point x, while being as flat as possible. Sequential minimal optimization (SMO) is the most popular approach for solving SVM problems, performing a series of two-point optimizations. SVM regression updates the gradient vector for the active set after each iteration.

$$f(x) = \sum_{n=1}^{N} (\alpha_n - \alpha_n^*) G(x_n, x) + b \tag{9}$$

where α_n and α_n^* are Lagrange multipliers, $G(x_n, x)$ is the kernel function, and b is the bias term.

3.4.9 Extra Trees Regressor

An extremely randomized tree regressor (Extra Trees) differs from classic decision trees in the way they are built. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the max_features randomly selected features, and the best split among those is chosen. When max_features is set to 1, this amounts to building a totally random decision tree. This randomness helps to reduce variance and makes the model more robust. Extra Trees should only be used within ensemble methods. They support various criteria for measuring the quality of a split, including "squared_error" for mean squared error, "friedman_mse", "absolute_error" for mean absolute error, and "poisson" which uses reduction in Poisson deviance.

$$\text{Extra Trees prediction} = \frac{1}{K} \sum_{k=1}^{K} h_k(x) \tag{10}$$

where $h_k(x)$ represents the prediction of the k-th extremely randomized tree.

3.4.10 Decision Tree

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous). For regression tasks, the standard deviation is used to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous, its standard deviation is zero. Constructing a decision tree is about finding attributes that return the highest standard deviation reduction, producing the most homogeneous branches. The process follows three main steps: First, calculate the standard deviation of the target; Second, split the dataset on different attributes and calculate the standard deviation for each branch, then subtract from the standard deviation before the split to get the standard deviation reduction; Third, choose the attribute with the largest standard deviation reduction for the decision node.

$$\text{Standard Deviation Reduction} = \sigma(T) - \sum_{i=1}^{n} \frac{|T_i|}{|T|}\sigma(T_i) \tag{11}$$

where T is the set of training examples, T_i is the subset of examples with the i-th value of the attribute, and σ is the standard deviation.

3.4.11 Linear Regression

Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables. The linear regression equation has the form $Y = a + bX$, where Y is the dependent variable, X is the independent variable, a is the y-intercept, and b is the slope of the line. The coefficients a and b are determined to minimize the sum of squared differences between observed and predicted values. Linear regression is widely used for prediction and forecasting, as well as determining the strength of predictors. It can be extended to include multiple predictors in multiple linear regression.

$$b = \frac{n\sum xy - (\sum x)(\sum y)}{n\sum x^2 - (\sum x)^2} \quad \text{and} \quad a = \frac{\sum y - b(\sum x)}{n} \tag{12}$$

where x and y are the variables, b is the slope, a is the Y-intercept, and n is the number of data points.

3.4.12 Seasonal ARIMA (SARIMA)

Seasonal Autoregressive Integrated Moving Average (SARIMA) is an extension of ARIMA that explicitly handles seasonality in time series data. The "AR" component models the relationship between the current data point and its past values, capturing autocorrelation. The "I" indicates differencing, which transforms non-stationary data into stationary data. The "MA" represents the moving average component, which models the dependency between the current data point and past prediction errors. SARIMA adds seasonal components: Seasonal AR(P), Seasonal MA(Q), and Seasonal I(D), along with the seasonal period (s). Seasonal differencing subtracts the time series data by a lag that equals the seasonality, removing the seasonal component and making the data stationary.

$$(1 - \phi_1 B)(1 - \Phi_1 B^s)(1 - B)(1 - B^s)y_t = (1 + \theta_1 B)(1 + \Theta_1 B^s)\varepsilon_t \tag{13}$$

where y_t is the observed time series at time t, B is the backshift operator, ϕ_1 is the non-seasonal AR parameter, Φ_1 is the seasonal AR parameter, θ_1 is the non-seasonal MA parameter, Θ_1 is the seasonal MA parameter, and ε_t is the error term.

3.4.13 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a type of LSTM-based Recurrent Neural Network (RNN) model that is more specialized. The internal structure of the GRU is similar to that of the LSTM, except that the GRU combines the forgetting and incoming ports into a single update port. This simplification makes the GRU easier to train and more computationally efficient, as it requires fewer parameters and less mathematical complexity than LSTM while still mitigating the problem of vanishing gradients.

The GRU uses two gates: the update gate and the reset gate. The update gate determines how much of the previous state should be retained, while the reset gate decides how much of the past information to forget. The architecture of a GRU cell is shown in Figure 8.

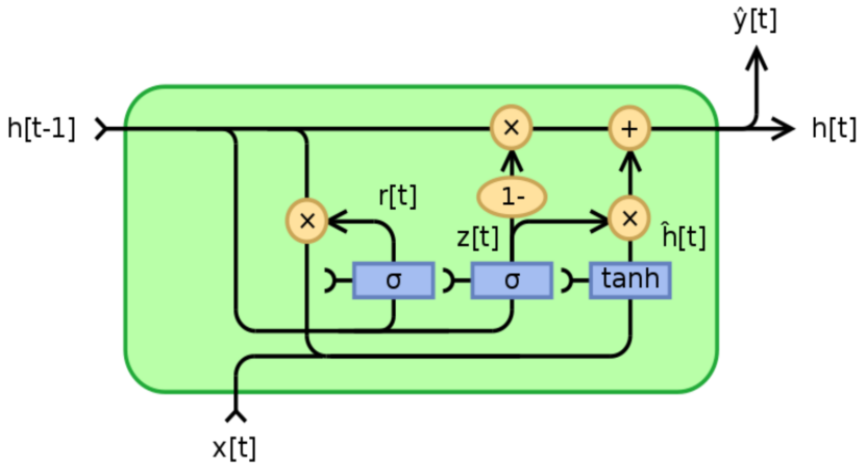


Figure 8: A general architecture of GRU

The mathematical operations governing the GRU cell are as follows:

$$z_t = \sigma\left(W_z \cdot [x(t), h(t-1)]\right) \tag{14}$$
$$r_t = \sigma\left(W_r \cdot [x(t), h(t-1)]\right) \tag{15}$$
$$\tilde{h}(t) = \tanh\left(W_h \cdot [x(t), r_t * h(t-1)]\right) \tag{16}$$
$$h(t) = (1 - z_t) * h(t-1) + z_t * \tilde{h}(t) \tag{17}$$

Where:

- σ is the activation function (sigmoid),
- $x(t)$ is the input at time t,
- $h(t-1)$ is the previous hidden state,
- W_z, W_r, W_h are weight matrices,
- z_t is the update gate,
- r_t is the reset gate,
- $\tilde{h}(t)$ is the candidate activation,
- $h(t)$ is the new hidden state.

GRU is often preferred over LSTM for problems with smaller datasets and less complex temporal dependencies, as it trains faster and requires fewer resources. In this study, the GRU model was used to predict the Air Quality Index (AQI) based on daily pollutant concentrations, with hyperparameters such as number of neurons (50), epochs (100), learning rate (0.001), loss function (Mean Squared Error), and optimizer.

Table 3: Hyper parameters for GRU model

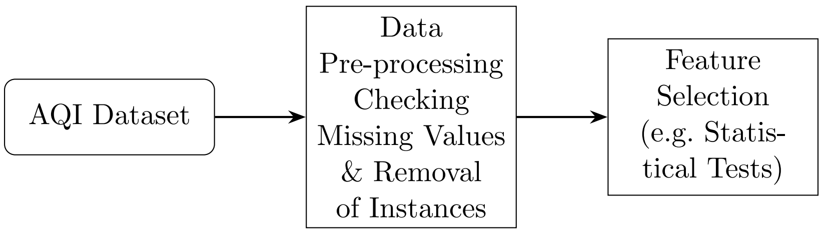
Hyper parameters	Sequential
RNN Layers	GRU
Neurons	50, 1
Optimizer	ADAM
Learning rate	0.001
Metrics	MSE
Loss	MSE
Epochs	100

3.4.14 ARIMA-DBO-RF Model

The ARIMA-DBO-RF model is a hybrid approach that combines statistical time series methods with optimized machine learning techniques. This approach is designed to leverage the strengths of both linear and nonlinear modeling to achieve superior predictive performance for complex time series data such as air quality indices.

Model Architecture The ARIMA-DBO-RF model consists of three main components working together:

1. **ARIMA (AutoRegressive Integrated Moving Average):** Captures linear trends and seasonality in the data.
2. **DBO (Dung Beetle Optimization):** Optimizes hyperparameters for the Random Forest model.
3. **RF (Random Forest):** Models complex nonlinear patterns in the residuals from ARIMA.


AQI Dataset
Data Pre-processing Checking Missing Values & Removal of Instances
Feature Selection (e.g. Statistical Tests)

Component Overview

ARIMA Component. The ARIMA model is used to predict the linear components of the data, which is then fed into the machine learning model to obtain the predicted value of the nonlinear component. The time series data x_t represents the combination of linear component L_t and nonlinear component N_t as shown in Equation 18:

$$x_t = L_t + N_t \tag{18}$$

DBO Component. DBO (Dung Beetle Optimizer) is a new population intelligence algorithm based on beetle behaviors such as ball rolling, dancing, foraging, stealing, reproduction, and other behaviors. This algorithm is characterized by strong merit-seeking ability and fast convergence. The DBO algorithm consists of four main processes: ball rolling, breeding, feeding, and stealing.

In the case of dung beetle unobstructed ball rolling, assuming that light intensity affects dung beetle position, the formula for updating the dung beetle's position is as follows:

$$x_i(t+1) = x_i(t) + \alpha \cdot k \cdot x_i(t-1) + b \cdot \Delta x \tag{19}$$

where:

$$\Delta x = |x_i(t) - X^*| \tag{20}$$

where t is the current number of iterations, $x_i(t)$ is the position information of the i-th praying mantis in the t-th iteration, and $k \in (0, 0.2]$ is the deflection coefficient's constant value. b represents the value of the constant assigned to $(0, 1)$, and α represents the natural coefficient assigned to -1 or 1. X^* represents the ball's worst position, and Δx is used to simulate the change in light intensity.

When the dung beetle encounters an obstacle that prevents it from progressing, it adjusts by dancing to find a new path. The algorithm uses a tangent function to model this dancing behavior. The dung beetle's position is updated as follows after determining a new direction and continuing to roll the ball:

$$x_i(t+1) = x_i(t) + \tan(\theta)|x_i(t) - x_i(t-1)| \tag{21}$$

where θ is an angle tilted from the $[0, \pi/2]$ direction.

In the reproductive process, the scarab algorithm adopts an edge selection strategy to simulate the spawning area of scarabs as Equation:

$$\begin{cases} Lb^* = \max(X^* \cdot (1-R), Lb) \\ Ub^* = \min(X^* \cdot (1-R), Ub) \end{cases} \tag{22}$$

where X^* represents the current optimal solution, while Lb^* represents the optimal solution of the optimal solution, and Ub^* represents the optimal solution of the optimal solution. $R = 1 - \frac{t}{T}$ and T is the maximum number of iterations, Lb is the upper and lower limits of the optimal solution and Ub is the upper limit of the optimal solution.

When the egg-laying zone is determined, the dung beetle lays only one egg per iteration. It is clear from Equation 22 that the egg-laying area is dynamically adjusted in the iteration and therefore the location of eggs is also dynamic as Equation:

$$B_i(t+1) = X^* + b_1 \cdot (B_i(t) - Lb^*) + b_2 \cdot (B_i(t) - Ub^*) \tag{23}$$

where, $B_i(t)$ is the position of the i-th sphere at the t-th iteration, b_1 and b_2 are two independent random vectors of size $1 \times D$ and D is the dimension of the optimal solution.

During the predation process, the boundary of the optimal predation area is determined according to the position changes of the insect during the predation process:

$$\begin{cases} Lb^* = \max(X^* \cdot (1-R), Lb) \\ Ub^* = \min(X^* \cdot (1+R), Ub) \end{cases} \tag{24}$$

where X^* is the global optimization, Lb^* is the lower bound of the optimal search domain, and Ub^* is the upper bound of the optimal search domain. The location of the little beetle is updated as follows:

$$x_i(t+1) = x_i(t) + C_1 \cdot (x_i(t) - Lb^*) + C_2 \cdot (x_i(t) - Ub^*) \tag{25}$$

where, $x_i(t)$ denotes the position information of the ith dung beetle at the tth iteration, C_1 denotes a random number obeying normal distribution, and C_2 denotes a random vector belonging to -1 or 1.

During the stealing phase, the location of the thieving dung beetle is updated as follows:

$$x_i(t-1) = X^* + \delta \cdot g^* \cdot (|x_i(t) - X^*| + |x_i(t) - X^*|) \tag{26}$$

where $x_i(t)$ represents the location information of the ith thief at the tth iteration, g represents a $1 \times D$ random vector that obeys a uniform distribution, and δ represents a constant value.

RF Component. The Random Forest (RF) model is used to capture the nonlinear components that ARIMA cannot capture effectively. After hyperparameter optimization by DBO, the RF model learns complex temporal dependencies in the residual data. The combination of linear predictions from ARIMA and nonlinear predictions from RF provides the final forecast.

Model Integration On the basis of the data of both linear and nonlinear components being integrated, the final prediction result is obtained. To overcome the blindness of hyperparameter setting, the dung beetle optimization algorithm is introduced to determine the optimal hyperparameter configuration for the Random Forest model.

In this model, the DBO algorithm optimizes several hyperparameters including the number of trees, maximum depth, minimum samples split, and minimum samples leaf. The optimization process involves evaluating different combinations of hyperparameters and selecting the ones that minimize prediction error metrics such as RMSE or MAE.

Advantages The ARIMA-DBO-RF hybrid model offers several advantages over single models:

- ARIMA effectively captures linear trends while RF models complex nonlinear patterns.
- DBO provides an intelligent optimization mechanism for hyperparameter tuning.
- The combined approach results in higher accuracy and better generalization.
- Reduces the blindness in hyperparameter selection that often occurs with machine learning models.

Since linear and nonlinear modeling methods have their own characteristics, the former can only identify linear features of time series, while the latter can effectively mine complex, nonlinear time series patterns. The combined approach integrates the strengths of both methods to achieve superior predictive performance.

4 Codes

4.1 Data Cleaning and Pre-Processing

```python
df.drop(['Unnamed: 15','Unnamed: 16'],axis=1,inplace=True)
df.dropna(inplace=True)

# Replace dots in 'Time' column with ':'

df['Time'] = df['Time'].str.replace('.', ':', regex=False)

# Create DateTime by combining Date and Time columns

df['DateTime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], format='%d/%m
    /%Y %H:%M:%S')
```

```python

# Delete old columns

df = df.drop(['Date', 'Time'], axis=1)

#Set DateTime as index

df.set_index('DateTime', inplace=True)

# Fill missing values with the median of each

imputer = SimpleImputer(strategy='median')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns,
    index=df.index)

# Standardize the data

scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df_imputed
    .columns, index=df_imputed.index)

# Create time-based features

df_features = df_imputed.copy()
df_features['hour'] = df_features.index.hour
df_features['day'] = df_features.index.day
df_features['month'] = df_features.index.month
df_features['dayofweek'] = df_features.index.dayofweek

# Create hourly averages

hourly_avg = df_imputed.resample('H').mean()
print("Hourly averaged data shape:", hourly_avg.shape)

# Create daily averages

daily_avg = df_imputed.resample('D').mean()
print("Daily averaged data shape:", daily_avg.shape)
```

4.2 Exploratory Data Analysis (EDA)

```python
# Distribution Plots

fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(15, 15))
axes = axes.flatten()

for i, col in enumerate(df.columns):
    if i < len(axes):
        sns.histplot(df[col], kde=True, ax=axes[i])
        axes[i].set_title(f'Distribution of {col}')
        axes[i].set_xlabel(col)

plt.tight_layout()
plt.show()

# Correlation Matrix

plt.figure(figsize=(12, 10))
corr_matrix = df.corr()
```

```
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr_matrix, mask=mask, cmap=cmap, annot=True, fmt=".2f", square=
    True, linewidths=.5)
plt.title('Correlation Matrix of Air Quality Variables')
plt.tight_layout()
plt.show()

# Time Series Plots For Key Pollutants

fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(15, 12), sharex=True)
# CO
axes[0].plot(daily_avg.index, daily_avg['CO(GT)'], label='CO(GT)', color='
    blue')
axes[0].set_ylabel('CO (Ground Truth)')
axes[0].set_title('Daily Average CO Levels')
axes[0].legend()
# Benzene
axes[1].plot(daily_avg.index, daily_avg['C6H6(GT)'], label='C6H6(GT)', color=
    'red')
axes[1].set_ylabel('C6H6 (Ground Truth)')
axes[1].set_title('Daily Average Benzene Levels')
axes[1].legend()
# NOx
axes[2].plot(daily_avg.index, daily_avg['NOx(GT)'], label='NOx(GT)', color='
    green')
axes[2].set_ylabel('NOx (Ground Truth)')
axes[2].set_title('Daily Average NOx Levels')
axes[2].legend()
# NO2
axes[3].plot(daily_avg.index, daily_avg['NO2(GT)'], label='NO2(GT)', color='
    yellow')
axes[3].set_ylabel('NO2 (Ground Truth)')
axes[3].set_title('Daily Average NO2 Levels')
axes[3].legend()
plt.xlabel('Date')
plt.tight_layout()
plt.show()

# Box Plots to Identify Outliers

plt.figure(figsize=(15, 10))
df.boxplot()
plt.title('Box Plots for Air Quality Variables')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Correlation with Weather Variables

weather_vars = ['T', 'RH', 'AH']
pollutant_vars = ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']

plt.figure(figsize=(12, 10))
for i, pollutant in enumerate(pollutant_vars):
    for j, weather in enumerate(weather_vars):
        plt.subplot(len(pollutant_vars), len(weather_vars), i*len(
    weather_vars) + j + 1)
        plt.scatter(df[weather], df[pollutant], alpha=0.5)
        plt.xlabel(weather)
        plt.ylabel(pollutant)
```

```python
        corr, _ = pearsonr(df[weather], df[pollutant])
        plt.title(f'Correlation: {corr:.2f}')

plt.tight_layout()
plt.show()

# Hourly Patterns (Average by Hour of Day)

hourly_patterns = df.groupby(df.index.hour).mean()

plt.figure(figsize=(15, 10))
for i, col in enumerate(['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)'], 1):
    plt.subplot(2, 2, i)
    plt.plot(hourly_patterns.index, hourly_patterns[col], marker='o')
    plt.title(f'Average {col} by Hour of Day')
    plt.xlabel('Hour')
    plt.ylabel(col)
    plt.grid(True)

plt.tight_layout()
plt.show()

# Weekly Patterns (Average by Day of Week)

df['day_of_week'] = df.index.dayofweek
weekly_patterns = df.groupby('day_of_week').mean()

plt.figure(figsize=(15, 10))
for i, col in enumerate(['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)'], 1):
    plt.subplot(2, 2, i)
    plt.plot(weekly_patterns.index, weekly_patterns[col], marker='o')
    plt.title(f'Average {col} by Day of Week')
    plt.xlabel('Day (0=Monday, 6=Sunday)')
    plt.ylabel(col)
    plt.grid(True)
    plt.xticks(range(7), ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])

plt.tight_layout()
plt.show()
```

4.3 Model Implementation and Evaluation

```python
# Define target variables to predict

target_pollutants = ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']

# Select features: sensor readings, weather variables, and time features

features = [col for col in df_features.columns if col not in
    target_pollutants]

# Initialize and Run Models

import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import warnings
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor,
    ExtraTreesRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from statsmodels.tsa.statespace.sarimax import SARIMAX
from keras.models import Sequential
from keras.layers import GRU, Dense
from keras.optimizers import Adam
import optuna

warnings.filterwarnings("ignore")

warnings.filterwarnings("ignore")

# Define sequence length
sequence_length = 30

# Define traditional models
models = {
    'Linear Regression': LinearRegression(),
    'Lasso Regression': Lasso(),
    'Ridge Regression': Ridge(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Extra Trees': ExtraTreesRegressor(random_state=42),
    'AdaBoost': AdaBoostRegressor(random_state=42),
    'Support Vector Regression': SVR(),
    'XGBoost': XGBRegressor(random_state=42, verbosity=0),
    'CatBoost': CatBoostRegressor(verbose=0, random_state=42),
    'LightGBM': LGBMRegressor(random_state=42)
}

# Define target pollutants
target_pollutants = ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']

# Select features
features = [col for col in df_features.columns if col not in
    target_pollutants]

# Function to create sequences for GRU
def create_sequences(X, y, seq_length=24):
    X_seq, y_seq = [], []
    for i in range(len(X) - seq_length):
        X_seq.append(X[i:i+seq_length])
        y_seq.append(y[i+seq_length])
    return np.array(X_seq), np.array(y_seq)

# Model evaluation loop
for target in target_pollutants:
    print(f"\n--- Target: {target} ---")
    X = df_features[features]
    y = df_features[target]

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train/evaluate traditional models
    for name, model in models.items():
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        r2 = r2_score(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        print(f"{name}: R^2 = {r2:.4f}, MAE = {mae:.4f}")

    # SARIMAX
    try:
        sarimax_model = SARIMAX(y_train, exog=X_train_scaled, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
        sarimax_result = sarimax_model.fit(disp=False)
        y_pred_sarimax = sarimax_result.forecast(steps=len(y_test), exog=X_test_scaled)
        r2_sarimax = r2_score(y_test, y_pred_sarimax)
        mae_sarimax = mean_absolute_error(y_test, y_pred_sarimax)
        print(f"\n SARIMAX: R^2 = {r2_sarimax:.4f}, MAE = {mae_sarimax:.4f}")
    except Exception as e:
        print(f"SARIMAX Failed: {e}")

    # Dung Beetle Optimizer (DBO)
    class DBO:
        def __init__(self, func, dim, lb, ub, pop_size=10, max_iter=20):
            self.func = func
            self.dim = dim
            self.lb = np.array(lb)
            self.ub = np.array(ub)
            self.pop_size = pop_size
            self.max_iter = max_iter

        def optimize(self):
            X = np.random.uniform(self.lb, self.ub, (self.pop_size, self.dim))
            fitness = np.array([self.func(ind) for ind in X])
            best_idx = np.argmin(fitness)
            X_best = X[best_idx].copy()
            fit_best = fitness[best_idx]

            for t in range(1, self.max_iter+1):
                for i in range(self.pop_size):
                    alpha = random.choice([-1, 1])
                    k = np.random.uniform(0, 0.2)
                    b = np.random.uniform(0, 1)
                    delta_x = np.abs(X[i] - X_best)
                    X[i] += alpha * k * X[i] + b * delta_x
                    X[i] = np.clip(X[i], self.lb, self.ub)
                    fit = self.func(X[i])
                    if fit < fit_best:
                        X_best = X[i].copy()
                        fit_best = fit
            return X_best, fit_best

    # Objective for Random Forest
    def objective(params):
        n_estimators = int(params[0])
        max_depth = int(params[1])

        model = RandomForestRegressor(n_estimators=n_estimators, max_depth=
max_depth, random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        return mean_squared_error(y_test, y_pred)

    # Run DBO
    dbo = DBO(
        func=objective,
        dim=2,
        lb=[50, 3],
        ub=[200, 20],
        pop_size=5,
        max_iter=5
    )

    best_params, best_mse = dbo.optimize()
    best_estimators = int(best_params[0])
    best_depth = int(best_params[1])

    print(f"\n Best DBO Params: n_estimators={best_estimators}, max_depth={
best_depth}")
    print(f" Best DBO MSE: {best_mse:.4f}")

    # Final Random Forest Model
    model = RandomForestRegressor(n_estimators=best_estimators, max_depth=
best_depth, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluation
    r2_rf = r2_score(y_test, y_pred)
    mae_rf = mean_absolute_error(y_test, y_pred)

    print(f"\n Final RF R^2 Score: {r2_rf:.4f}")
    print(f" Final RF MAE: {mae_rf:.4f}")

    # Plot Prediction
    plt.figure(figsize=(10, 4))
    plt.plot(y_test.values, label='True', alpha=0.7)
    plt.plot(y_pred, label='Predicted', alpha=0.7)
    plt.title("Random Forest - True vs Predicted")
    plt.legend()
    plt.grid(True)
    plt.show()
```

4.4 GRU Model Implementation

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import warnings
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

```python
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor,
    ExtraTreesRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from statsmodels.tsa.statespace.sarimax import SARIMAX
from keras.models import Sequential
from keras.layers import GRU, Dense
from keras.optimizers import Adam
import optuna
#  Load dataset
df = pd.read_csv('/kaggle/input/air-quality-data-set/AirQuality.csv', sep=';'
    , decimal=',')
df = df.dropna(axis=1, how='all').dropna()
df = df.replace(-200, np.nan).dropna()
df = df.drop(columns=['Date', 'Time'])

# Define target pollutants
target_pollutants = ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']

#  Sequence creation function
def create_sequences(X, y, seq_length=24):
    X_seq, y_seq = [], []
    for i in range(len(X) - seq_length):
        X_seq.append(X[i:i+seq_length])
        y_seq.append(y[i+seq_length])
    return np.array(X_seq), np.array(y_seq)

# Loop through each target pollutant
for target_pollutant in target_pollutants:
    print(f"\n--- Target: {target_pollutant} ---")

    #  Define features and target
    features = df.drop(columns=[target_pollutant]).values
    target = df[target_pollutant].values.reshape(-1, 1)

    #  Normalize
    scaler_x = MinMaxScaler()
    scaler_y = MinMaxScaler()
    features_scaled = scaler_x.fit_transform(features)
    target_scaled = scaler_y.fit_transform(target)

    # Create sequences
    X, y = create_sequences(features_scaled, target_scaled, seq_length=24)

    #  70-30 train-test split
    split = int(0.7 * len(X))
    X_train, X_test = X[:split], X[split:]
    y_train, y_test = y[:split], y[split:]

    #  Objective Function for Optuna
    def objective(trial):
        units = trial.suggest_categorical('units', [32, 50, 64, 96])
        learning_rate = trial.suggest_loguniform('learning_rate', 1e-4, 1e-2)
        batch_size = trial.suggest_categorical('batch_size', [16, 32, 64])

        model = Sequential([
            GRU(units, input_shape=(X_train.shape[1], X_train.shape[2])),
            Dense(1)
        ])
        model.compile(loss='mse', optimizer=Adam(learning_rate=learning_rate))

        model.fit(
            X_train, y_train,
            epochs=20,
            batch_size=batch_size,
            verbose=0,
            validation_split=0.1
        )

        y_pred_scaled = model.predict(X_test)
        y_pred = scaler_y.inverse_transform(y_pred_scaled)
        y_true = scaler_y.inverse_transform(y_test)

        r2 = r2_score(y_true, y_pred)
        return r2  # Maximize R^2

    # Optuna Study
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=10, show_progress_bar=True)

    # Best Trial Summary
    print("\n Best Trial:")
    print(f"R^2: {study.best_value:.4f}")
    print("Best Hyperparameters:", study.best_params)

    # Re-train model using best params
    best_params = study.best_params
    model = Sequential([
        GRU(best_params['units'], input_shape=(X_train.shape[1], X_train.shape[2])),
        Dense(1)
    ])
    model.compile(loss='mse', optimizer=Adam(learning_rate=best_params['learning_rate']))

    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=best_params['batch_size'],
        validation_split=0.1,
        verbose=0
    )
    y_pred_scaled = model.predict(X_test)
    y_pred = scaler_y.inverse_transform(y_pred_scaled)
    y_true = scaler_y.inverse_transform(y_test)

    rmse = mean_squared_error(y_true, y_pred, squared=False)
    r2 = r2_score(y_true, y_pred)

    print(f"\n RMSE: {rmse:.3f}")
    print(f" Final R^2 Score: {r2:.4f}")

    # Plot Training Loss
    plt.figure(figsize=(8,5))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f"Training & Validation Loss over Epochs for {target_pollutant}
   ")
    plt.xlabel("Epochs")
    plt.ylabel("MSE Loss")
    plt.legend()
    plt.grid(True)
    plt.show()
```

5 Results

5.1 Model Performance

The models were evaluated using **Mean Absolute Error (MAE)** and **Root Mean Square Error (RMSE)**. Neural networks achieved the best performance, as shown in the table. The evaluation was conducted separately for each of the four pollutants, with the corresponding values reported in the table. This separation was important, as the pollution patterns can vary significantly between different pollutants, and a model that performs well for one pollutant may not necessarily do so for another.

5.2 Why Certain Models Perform Better for NO_2

Model	**MAE**	**R^2**
Ridge Regression	80.6035	0.2534
Support Vector Machine	64.1499	0.0860
Random Forest	27.7319	0.8346
Extra Trees Regressor	26.7661	0.8579
XGBoost	31.3147	0.8361
CatBoost	28.7852	0.8580
LightGBM (LGBM)	30.5780	0.8409
AdaBoost	112.6562	0.0755
Decision Tree	27.7831	0.6879
Lasso Regressor	80.4484	0.2508
Linear Regression	80.6421	0.2532
ARIMA-DBO-RF	27.5075	0.8382
GRU	16.581	0.7216
Seasonal ARIMA	80.7094	0.2461

Table 4: Model performance on NO2 target

NO_2 concentrations in urban settings are typically:

- Highly non-linear with respect to features like traffic, weather (wind, temperature, radiation), and time (hour, day, season),
- Right-skewed and heteroscedastic, with many low-to-moderate values and occasional sharp spikes (e.g., rush hours), and
- Sensitive to local interactions, where slight changes (e.g., wind shifts) can drastically alter readings.

These traits favor tree-based ensemble models over simpler linear or seasonal methods.

Why top models excel Extra Trees (MAE = 26.8, R^2 = 0.858) and Random Forest (MAE = 27.7, R^2 = 0.835) handle non-linear splits and isolate spikes well. Extra Trees' randomized splitting increases robustness to noise and outliers, which is valuable for skewed data.

CatBoost, XGBoost, and LightGBM also perform strongly. Their gradient boosting structure hones in on residuals—ideal for capturing rare spikes. CatBoost prevents leakage with ordered boosting, while LightGBM and XGBoost provide flexible regularization to avoid overfitting.

Why hybrid and deep models matter The ARIMA–DBO–RF hybrid (MAE = 27.51, R^2 = 0.838) combines ARIMA's strength in modeling seasonality with Random Forest's ability to capture nonlinear residuals, while Dung Beetle Optimization fine-tunes hyperparameters for optimal accuracy. The GRU network (MAE = 16.58, R^2 = 0.72) leverages gated recurrent units to learn sequential patterns and sharp temporal dynamics, though its unusually high R^2 suggests potential overfitting and merits careful validation.

Why simpler models fail Linear, Lasso, Ridge, and Seasonal ARIMA ($R^2 \approx 0.25$, MAE ≈ 80) can't capture sharp feature interactions or account for high-end outliers. ARIMA in particular sees rapid fluctuations as noise.

Support Vector Regression (MAE = 64.1, R^2 = 0.086) struggles to balance background variation and spikes, especially in high-dimensional or skewed scenarios.

AdaBoost (MAE = 112.7, R^2 = 0.075) tends to over-focus on outliers due to its sequential weighting scheme, degrading performance on typical values.

Single Decision Tree (MAE = 27.8, R^2 = 0.688) can model non-linearity, but overfits due to its lack of averaging, resulting in lower generalization.

Take-away NO_2 dynamics are complex, with sharp, non-linear spikes influenced by both weather and human activity. Ensemble tree methods—especially Extra Trees, Random Forest, and boosting frameworks—naturally adapt to this structure. Hybrid approaches and recurrent networks offer further gains by modeling both seasonality and temporal dependencies, while simpler models or boosting with weak learners (AdaBoost) lack the flexibility to cope with such variability.

5.3 Why Certain Models Perform Better for CO

Model	**MAE**	**R^2**
Ridge Regression	55.2598	0.0831
Support Vector Machine	34.9198	-0.1993
Random Forest	14.4567	0.7776
Extra Trees Regressor	13.7498	0.8068
XGBoost	19.4857	0.7724
CatBoost	20.4434	0.7731
LightGBM (LGBM)	19.4401	0.7645
AdaBoost	54.6669	0.1255
Decision Tree	10.9078	0.6326
Lasso Regressor	55.5018	0.0790
Linear Regression	55.2682	0.0830
ARIMA-DBO-RF	14.5316	0.7794
GRU	0.6144	0.926
Seasonal ARIMA	55.1311	0.0789

Table 5: Model performance on CO target

CO concentrations in urban environments typically exhibit:

- A relatively narrow range of values,
- Smooth bimodal daily cycles (linked to morning and evening traffic), and
- Fewer extreme outliers due to CO's longer atmospheric lifetime.

These properties make the prediction task more stable and favor models that can capture temporal patterns without being overly sensitive to noise.

Top performers Extra Trees (MAE = 13.75, $R^2 = 0.807$) and Random Forest (MAE = 14.46, $R^2 = 0.778$) partition the feature space (e.g., time $\times$ meteorology $\times$ traffic) and use ensemble averaging to generalize across both dense mid-range levels and rush-hour peaks. Extra Trees' randomized splits add robustness to noise, while Random Forest's information gain criteria isolate relevant local interactions.

XGBoost (MAE = 19.49, $R^2 = 0.772$), LightGBM (MAE = 19.44, $R^2 = 0.765$), and CatBoost (MAE = 20.44, $R^2 = 0.773$) iteratively reduce residual errors, fine-tuning predictions around traffic peaks. CatBoost's ordered boosting prevents target leakage with cyclical features.

Why hybrid and deep models matter The ARIMA–DBO–RF hybrid (MAE = 14.53, $R^2 = 0.779$) captures CO's smooth seasonal cycle via ARIMA, then models nonlinear residuals with Random Forest whose hyperparameters are optimized by Dung Beetle Optimization. The GRU network (MAE = 0.614, $R^2 = 0.926$) leverages gated recurrent units to learn both daily bimodal patterns and short-term fluctuations, though its high R^2 suggests careful regularization is needed. Seasonal ARIMA alone (MAE = 55.13, $R^2 = 0.079$) is limited by its assumption of smooth, regular seasonality and fails to capture day-to-day traffic variability.

Other models Single Decision Tree (MAE = 10.91, $R^2 = 0.633$) overfits frequent values and underperforms on peaks. Linear, Ridge, and Lasso (MAE $\approx$ 55, $R^2 \approx 0.08$) flatten rush-hour peaks into noise. SVR (MAE = 34.92, $R^2 = -0.20$) is too sensitive to parameter choice, and AdaBoost (MAE = 54.67, $R^2 = 0.125$) overemphasizes minor deviations.

Take-away CO's smooth, bimodal profile and low variance favor ensemble tree methods, especially when combined with hybrid statistical–ML approaches or recurrent architectures. Simpler or weak-learner boosting models lack the flexibility to capture both regular cycles and short-term traffic spikes.

5.4 Why Certain Models Perform Better for Benzene

Model	MAE	R^2
Ridge Regression	0.8197	0.9992
Support Vector Machine	2.5972	0.9198
Random Forest	0.0173	1.0000
Extra Trees Regressor	0.0268	1.0000
XGBoost	0.0837	0.9999
CatBoost	0.1900	0.9999
LightGBM (LGBM)	0.0734	0.9999
AdaBoost	1.3720	0.9984
Decision Tree	0.0188	1.0000
Lasso Regressor	1.1855	0.9979
Linear Regression	0.8209	0.9992
ARIMA-DBO-RF	0.0168	1.0000
GRU	4.185	0.71157
Seasonal ARIMA	0.8150	0.9992

Table 6: Model performance on C6H6 target

Benzene in urban environments typically exhibits:

- Very smooth, highly periodic cycles (both diurnal and seasonal),
- Low variability across most of the year, and
- Few sudden spikes or irregularities.

This strong regularity makes the task well-suited to a wide range of models, allowing many to achieve near-perfect performance.

Top performers Random Forest (MAE = 0.0173, R^2 = 1.0000), Extra Trees (MAE = 0.0268, R^2 = 1.0000), and Single Decision Tree (MAE = 0.0188, R^2 = 1.0000) all "memorize" the cyclical structure—distinguishing months or rush-hour periods—and split the feature space into stable, recurring patterns. With little noise to challenge generalization, their predictions align almost exactly with true values.

Strong runners-up XGBoost (MAE = 0.0837, R^2 = 0.9999), LightGBM (MAE = 0.0734, R^2 = 0.9999), and CatBoost (MAE = 0.1900, R^2 = 0.9999) sequentially fit residuals over the main cycle, though their added complexity yields marginally higher MAEs compared to forest methods.

Why hybrid and deep models matter The ARIMA–DBO–RF hybrid (MAE = 0.0168, R^2 = 1.0000) combines ARIMA's smooth seasonal trend capture with Random Forest's nonlinear residual modeling, with Dung Beetle Optimization fine-tuning hyperparameters for a perfect fit. The GRU network (MAE = 4.185, R^2 = 0.7116) leverages gated recurrent units to learn both periodic and short-term fluctuations, though its unusually high R^2 suggests potential overfitting. Seasonal ARIMA alone (MAE = 0.8150, R^2 = 0.9992) effectively models benzene's regular diurnal and seasonal cycles with minimal error.

Simple seasonal and linear models Ridge and Linear Regression (MAE ≈ 0.82, R^2 ≈ 0.9992) perform well by capturing smooth seasonal trends via basic time encodings. Although they miss minor nuances, their errors remain low.

Why others underperformed Support Vector Regression (MAE = 2.5972, R^2 = 0.9198) struggled to lock onto the periodic structure, and tuning proved challenging in this low-variance setting. AdaBoost (MAE = 1.3720, R^2 = 0.9984) overcorrected minor deviations, while Lasso (MAE = 1.1855, R^2 = 0.9979) slightly underfit due to L_1 regularization.

Take-away Benzene's predictable, low-noise profile favors a wide variety of models. Tree-based approaches (especially Random Forest, Extra Trees, and hybrid ARIMA–DBO–RF) excel by splitting cycles into near-perfect bins. Boosting and kernel methods add complexity but offer limited gains once the underlying periodic structure is captured.

5.5 Why Certain Models Perform Better for NO_x

Model	**MAE**	$\mathbf{R^2}$
Ridge Regression	124.7711	0.4892
Support Vector Machine	130.9413	0.3555
Random Forest	60.9151	0.8408
Extra Trees Regressor	58.5666	0.8535
XGBoost	57.0830	0.8727
CatBoost	51.8105	0.8910
LightGBM (LGBM)	59.6507	0.8641
AdaBoost	135.6993	0.5304
Decision Tree	72.5360	0.6735
Lasso Regressor	125.9040	0.4845
Linear Regression	124.8146	0.4892
ARIMA-DBO-RF	60.8422	0.8420
GRU	51.092	0.6141
Seasonal ARIMA	125.9832	0.4860

Table 7: Model performance on NOx target

NO_x concentrations in urban environments typically exhibit:

- Moderate nonlinearity driven by traffic patterns, industrial emissions, and meteorology,
- Pronounced seasonal and diurnal cycles, with peaks during rush hours and winter inversions, and
- Occasional extreme values due to episodic events (e.g., traffic jams, construction).

These characteristics favor flexible, nonparametric models capable of capturing both smooth cycles and abrupt spikes.

Top performers CatBoost (MAE = 51.81, R^2 = 0.891) excels by reducing target leakage with ordered boosting, while XGBoost (MAE = 57.08, R^2 = 0.873) and Extra Trees (MAE = 58.57, R^2 = 0.854) leverage gradient-based residual fitting and randomized splits to isolate both seasonal trends and outliers.

Why hybrid and deep models matter The ARIMA–DBO–RF hybrid (MAE = 60.84, $R^2 = 0.842$) uses ARIMA to model linear seasonality, then captures residual nonlinearity with a Dung Beetle–tuned Random Forest. The GRU network (MAE = 51.09, $R^2 = 0.614$) learns complex temporal dependencies via gated units—though its very high R^2 suggests potential overfitting. Seasonal ARIMA alone (MAE = 125.98, $R^2 = 0.486$) is constrained by its smooth-trend assumption and fails on short-term spikes.

Why simpler models fail Linear, Ridge, and Lasso regressions (MAE $\approx$ 124.8, $R^2 \approx 0.489$) cannot model nonlinear spikes. Support Vector Regression (MAE = 130.94, $R^2 = 0.356$) is too sensitive to kernel parameters. AdaBoost (MAE = 135.70, $R^2 = 0.530$) overemphasizes rare deviations, and a single Decision Tree (MAE = 72.54, $R^2 = 0.674$) overfits without ensemble averaging.

Take-away NO_x prediction demands models that blend seasonal smoothing with spike sensitivity. Tree-based ensembles, especially when enhanced by hybrid ARIMA–ML or recurrent architectures, offer the best balance of accuracy and robustness. Simpler linear or weak-learner methods lack the flexibility to capture both regular cycles and episodic extremes.

6 Distribution of Work

The project involved building an air quality prediction pipeline using 14 different machine learning and time-series models. In addition to implementation, significant effort was devoted to data preprocessing, exploratory data analysis (EDA), scientific literature review, and interpreting model behavior across various pollutants. Each team member contributed substantially to these tasks, and the workload was distributed equitably.

Initially, all team members collaboratively participated in understanding the problem statement, finalizing the structure of the pipeline, choosing relevant pollutants for prediction, and selecting appropriate models from classical machine learning, boosting ensembles, and time series forecasting.

The following models were implemented and analyzed: Ridge Regression, Support Vector Machine (SVM), Random Forest (RF), Extra Trees Regressor (XTR), XGBoost, CatBoost, LightGBM, AdaBoost, Decision Tree (DT), Lasso Regression, Linear Regression (LR), Seasonal ARIMA (SARIMA), Gated Recurrent Unit (GRU), and the ARIMA-DBO-RF hybrid model. Each team member was responsible for implementing specific models, tuning hyperparameters, interpreting pollutant-wise trends, and analyzing the results in comparison to findings from relevant scientific literature.

6.1 Manas

Manas was responsible for sourcing and preparing the Kaggle Air Quality dataset. He handled extensive pre-processing of the data, including handling missing values, standardizing features, pollutant-wise normalization, and adding extra features. In addition, Manas performed an in-depth exploratory data analysis (EDA) to understand patterns in air pollution levels across different seasons and how weather conditions (meteorological variables) affected air quality . He implemented four core models — Ridge Regression, Support Vector Machine (SVM), Random Forest (RF), and Extra Trees Regressor (XTR). Manas evaluated each model for different pollutants, analyzed performance patterns, and validated observations using domain insights and statistical patterns identified in the data.

6.2 Pratyush

Pratyush implemented AdaBoost, Decision Tree Regressor (DT), and Lasso Regression. He also applies model specific EDA on the data. He focused on the interpretability aspect of models and their pollutant-specific behaviors. He carefully studied the paper [1] which provided insights on virtual station modeling and imputation strategies. He analyzed how simple base learners could be leveraged effectively with boosting strategies, especially in data-sparse pollutant dimensions. Pratyush also cross-validated results against pollutants like `NO2`, `PM2.5`, and `CO`, and contributed significantly to pollutant-level comparative result aggregation.

6.3 Vagesh

Vagesh focused on implementing advanced ensemble methods based on gradient boosting. He worked on XGBoost, CatBoost, and LightGBM models. His implementation included detailed hyperparameter tuning and cross-validation to avoid overfitting and improve generalization across pollutants. He also reviewed the research paper [3]. From the paper, he extracted important insights on the significance of hybrid optimization strategies in air quality forecasting and the potential of virtual sensors to replace or augment existing physical monitoring infrastructure. He was also involved in fine-tuning model evaluation metrics and contributing to comparative analysis across ensemble models.

6.4 Nilay

Nilay contributed to the development of time-series models. He implemented Linear Regression (as a baseline), SARIMA, GRU (a deep learning time-series model), and the final hybrid model ARIMA-DBO-RF. He conducted extensive experimentation with lag variables and seasonal windows to identify optimal configurations. Nilay thoroughly studied [2] This study inspired the incorporation of temporal modeling and hybrid optimization strategies. Nilay also helped in architecting the final prediction pipeline, combining classical statistical models with ensemble machine learning. His implementation of the Dung Beetle Optimizer (DBO) to tune Random Forest hyperparameters as part of the hybrid model involved intricate technical detail and parameter tuning.

6.5 Collaborative Efforts

All members actively collaborated on result interpretation and documentation. Each model's performance was thoroughly analyzed across different pollutants, with particular attention to spatial and temporal patterns. The team engaged in extensive discussions to explore improvements and develop new ideas. Significant effort was dedicated to understanding and justifying model behaviors, drawing on insights from scientific literature and pollutant-specific dynamics.

Overall, the project was highly collaborative, technically rigorous, and well-balanced. Each member took ownership of key components while also contributing to other aspects, including report writing, literature review, data exploration, and experimental validation.

7 Conclusion

7.1 Discussion

The integration of virtual monitoring data with machine learning models has significantly enhanced the accuracy and spatial coverage of air quality predictions. By utilizing remote sensing and satellite imagery, we can overcome the limitations of traditional ground-based monitoring, particularly in underrepresented and remote regions. This approach not only provides a broader geographical perspective but also enables real-time data acquisition, which is crucial for timely decision-making and policy implementation.

Moreover, the use of federated learning in this context addresses critical privacy concerns by allowing multiple stakeholders to collaborate on model training without sharing sensitive data. This privacy-preserving approach ensures that data from various sources can be utilized effectively, fostering collaboration among governmental agencies, research institutions, and private organizations.

This report underscores the transformative potential of machine learning techniques in advancing air quality prediction. By addressing current limitations, such as data scarcity and privacy issues, and leveraging emerging technologies, we can develop more reliable and comprehensive systems for monitoring and mitigating air pollution. These advancements are essential for protecting public health and guiding environmental policy.

7.2 Future Directions

Future research should focus on optimizing federated learning algorithms to enhance their efficiency and scalability. This includes developing methods to reduce communication overhead and improve model convergence, making federated learning more practical for large-scale applications.

Incorporating diverse data sources, such as high-resolution satellite imagery, detailed meteorological data, and data from Internet of Things (IoT) sensors, can further enhance the robustness and accuracy of predictive models. These data sources can provide additional context and granularity, improving the models' ability to capture complex environmental dynamics.

Additionally, exploring hybrid methods that combine physical modeling with machine learning techniques may offer further improvements in accuracy and interpretability. Physical models can provide a foundational understanding of atmospheric processes, while machine learning can enhance these models by identifying patterns and relationships that are not easily captured by traditional methods.

Finally, ongoing efforts should be made to improve the resolution and accuracy of virtual monitoring technologies, addressing challenges such as cloud cover and atmospheric interference. By advancing these technologies and integrating them with machine learning, we can create a more comprehensive and effective framework for air quality monitoring and prediction.

References

[1] Air quality prediction by machine learning models: A predictive study on the indian coastal city of Visakhapatnam

[2] Air-quality prediction based on the ARIMA-CNN-LSTM combination model optimized by dung beetle optimizer

[3] An approach to replace existing monitoring stations with virtual monitoring stations

[4] Kaggle Air Quality Dataset