

Abstract

The transport sector contributes significantly to world CO₂ emissions, and emissions needs to be correctly quantified on a vehicle-by-vehicle basis for sustainable policy-making and design. This project illustrates a machine learning-based CO₂ Emissions Prediction Tool that predicts the emission output of a vehicle from its specifications. Rather than depending on full lifecycle analyses, the model relies on systematic features like engine capacity, cylinder count, hybrid status, and fuel consumption measurements to forecast emissions in grams per kilometer. The system combines Random Forest and Support Vector Regression (SVR) models in a stacked architecture with Gradient Boosting as the meta-learner to enhance accuracy and robustness. The model is trained on a Canadian vehicle dataset and provides predictions with high R² values and low error values, allowing for sound environmental analysis. This offline, data-intensive software provides a light and expandable solution to emission analysis. It is very beneficial for policymakers, manufacturers, and researchers in assessing the carbon footprint of car configurations and making decisions that fit into environmental sustainability objectives.

Table of Contents

1. Introduction.....	1
1.1. Background.....	1
1.2. Problem Statement.....	2
1.3. Objective of the Project.....	3
1.4. Scope of the Project.....	4
1.5. Methodology.....	5
2. Literature Survey.....	6
2.1 Related Works.....	6
2.2 Research Gaps and Challenges.....	9
3. Software and Hardware Requirements.....	12
3.1 Software Requirements.....	12
3.1.1 Non-Functional Requirements.....	12
3.1.2 Functional Requirements.....	12
3.1.3 Software Tools and Framework.....	13
3.2 Hardware Requirements.....	14
3.2.1 Minimum Hardware Configuration.....	14
3.2.2 Recommended Hardware Configuration.....	15
4. Proposed System.....	16
4.1 Overview of System Architecture.....	16
4.2 Core Modules and Functionalities.....	16
4.2.1 Data Acquisition Module.....	16
4.2.2 Model Building and Training Module.....	17
4.2.3 Carbon Footprint Prediction Module.....	18
4.2.4 Result Generation Module.....	18
4.3 System Architecture.....	19
4.4 User Interface.....	19
5. System Design.....	21

5.1	UML Diagram.....	21
5.1.1	Class Diagram.....	21
5.1.2	Sequence Diagram.....	23
5.1.3	Activity Diagram.....	25
5.1.4	Use Case Diagram.....	26
6.	Implementation.....	28
6.1	Data Collection and Preprocessing Module.....	29
6.1.1	Purpose.....	29
6.1.2	Algorithmic Flow.....	29
6.1.3	Dataset Description.....	30
6.1.4	Code Outline.....	30
6.1.5	Overall Flow.....	31
6.2	Model Building and Training Module.....	32
6.2.1	Purpose.....	32
6.2.2	Algorithmic Flow.....	32
6.2.3	Code Outline.....	32
6.2.4	Overall Flow.....	33
6.3	Carbon Footprint Prediction and Scenario Analysis ...	33
6.3.1	Purpose.....	33
6.3.2	Algorithmic Flow.....	34
6.3.3	Code Outline	34
6.3.4	Overall Flow	35
6.4	Results and Visualization Module.....	35
6.4.1	Purpose.....	35
6.4.2	Algorithmic Flow.....	36
6.4.3	Code Outline	36
6.4.4	Overall Flow	36
7.	Testing.....	38
7.1	Black Box Testing.....	38
7.1.1	Black Box Test Case 1.....	39

7.1.2	Black Box Test Case 2.....	40
7.2	White Box Testing.....	40
7.2.1	White Box Test Case 1.....	41
7.2.2	White Box Test Case 2.....	42
8.	Results.....	43
8.1	Vehicle Emission Prediction.....	43
8.2	Scenario Comparison.....	44
8.3	Input History and Result Logging.....	45
9.	Conclusion and Future Work.....	47
9.1	Conclusion.....	47
9.2	Future Work.....	48
10.	Reference.....	49
11.	Show & Tell Report.....	50

List of Tables

Table 7.1 Black Box Test Case 1.....	39
Table 7.2 Black Box Test Case 2.....	40
Table 7.3 White Box Test Case 1.....	41
Table 7.4 White Box Test Case 1.....	42
Table 9.1 Evaluation Metrics.....	47

List of Figures

Figure 4.1 System Architecture.....	19
Figure 5.1 Class Diagram.....	21
Figure 5.2 Sequence Diagram.....	23
Figure 5.3 Activity Diagram.....	25
Figure 5.4 Use Case Diagram.....	26
Figure 8.1 Home Screen.....	43
Figure 8.2 Prediction Screen.....	44
Figure 8.3 Compare Scenarios Form.....	45
Figure 8.4 Compare Scenarios Prediction Screen.....	45
Figure 8.5 Prediction History.....	46

1. Introduction

1.1 Background

Carbon emissions by the transportation industry are among the top causes of climate change. With countries demanding tougher environmental measures, modelling and predicting the CO₂ emissions of vehicles has become a crucial measure towards sustainable development. Traditional methods of emissions assessment usually depend on static reference tables or full lifecycle estimations, which tend to be complicated, rigid, or may demand extensive data collection, thus not so handy for routine use or rapid appraisal.

With the fast pace of development in machine learning and data science, it is now feasible to develop smart systems that process structured vehicle data and make precise emission estimates. Instead of needing end-to-end production lifecycle data, these models rely on technical features like engine size, fuel consumption rates, number of cylinders, hybrid status, and vehicle configuration to estimate CO₂ output. These attributes, accessible in manufacturer databases or user files, provide a solid basis for predictive modelling.

In contrast to web-based calculators or cloud simulations, this software runs in an offline mode, providing data privacy and availability in network-limited situations. The model utilizes ensemble learning methods, integrating Random Forest and Support Vector Regression (SVR) in a stacked regressor framework, further boosted by Gradient Boosting. This architecture enhances accuracy while ensuring scalability, making it suitable for application in academic, industrial, and regulatory environments.

Thus, the proposed system provides a handy, lightweight, high-performance solution for CO₂ emissions analysis that enables the users to test the environmental effects of the vehicle designs, fuel options, or the policies of regulations. It presents a scalable model that unites technical correctness with practical usability.

1.2 Problem Statement

Existing attempts to measure the carbon footprint of cars are constrained by outmoded or imperfect methods. Older models rely either on detailed lifecycle information, typically unavailable to consumers and smaller companies, or standardized emissions ratings that fail to account for differences in vehicle configurations, fuel types, and use contexts. Consequently, precise, vehicle-specific emissions analysis remains out of reach for most stakeholders.

Additionally, the majority of carbon analysis software is designed for cloud environments or needs detailed environmental data sets that are difficult to use or interpret. This restricts their real-world use, especially in educational institutions, researchers, or policymakers operating offline or in resource-constrained environments. Such tools usually do not provide real-time feedback or permit scenario comparison through varying design or performance parameters.

This situation points to a key deficiency in sustainability tools for the transportation industry: the absence of an intelligent, self-contained emissions forecasting system that can process vehicle data locally and estimate carbon production without elaborate infrastructure or complete lifecycle monitoring. A predictive, lightweight solution is needed that can assist users in comprehending emissions at the vehicle level based on standard engine and performance specifications.

Our project meets this challenge by constructing an offline machine learning-based CO₂ emissions estimator tool that runs on features of engine size, cylinder number, hybrid status, and fuel consumption metrics to make precise emission estimates. The model provides results similar in accuracy to more sophisticated systems but remains usable, fast, and simple—enabling sustainable decision-making for academic, regulatory, and industrial applications.

1.3 Objectives of the Project

The main aim of this project is to build and implement an offline machine learning-based tool to predict CO₂ emissions from vehicles with high accuracy based on easily available technical specifications like engine size, fuel rate, and vehicle type. It does not require full lifecycle information or cloud computation, and the tool becomes scalable, feasible, and network-independent.

An important technical objective is to apply a stacked model of regression that combines Random Forest and Support Vector Regression (SVR) and uses Gradient Boosting as the meta-learner. The ensemble model architecture is selected in order to optimize prediction performance, reduce error, and generalize effectively across a wide range of vehicle types such as hybrids and fuel-based models.

A second large objective of the project is assisting in real-time, scenario-based assessment. Users need to be able to try out modifications to vehicle parameters—such as a transition to hybrid engines or changes in fuel types—and see the consequences immediately on CO₂ emissions. This capability gives users actionable information to assist with green vehicle design and policy making.

The software also emphasizes local processing of data and offline operation, providing complete control over critical input data without internet connectivity or third-party platforms. It thus supports deployment in research institutions, government, and academic settings with privacy requirements or restricted connectivity.

Lastly, the system is intended to be lightweight, simple to use, and widely deployable. Accessed by students for school projects or by regulators for emissions benchmarking, the tool is designed to have an intuitive interface and flexible back-end support to fit a broad set of use cases and operating conditions.

1.4 Scope of the Project

It is intended for application by academic researchers, environmental scientists, automotive analysts, and regulators looking for fast, accurate estimation of vehicle-specific CO₂ emissions. Its chief application is in situations where lifecycle data may not be readily available, but essential vehicle specifications like engine size, number of cylinders, and fuel consumption are readily known.

The system specializes in forecasting CO₂ emissions from cars based on structured, tabular data. It is independent of cloud connectivity and real-time sensor data and is therefore well-suited to offline, resource-limited contexts. The application is locally deployed and provides forecasts without needing to access external databases, which allows for data privacy and ease of deployment across varied institutional and operational settings.

The model can model a large number of different vehicle types—such as hybrids and traditional fuel-burning engines—by applying a blend of regression algorithms trained on publicly available emissions data sets. Various parameter settings may be entered by users, and comparative predictions are generated to see the environmental cost of each scenario.

Although the default functionality is oriented around emissions forecasting and scenario analysis, the tool also offers visualization functionality that makes it easier for users to interpret the impact of every variable on CO₂ output. Such insights can be of most value in classroom education, regulatory benchmarking, or green design research.

The system is not designed to monitor emissions throughout the entire product life cycle or involve integration with IoT devices or real-time vehicle sensors. It is strictly data-driven, static feature-based prediction to facilitate rapid and intelligent sustainability analysis.

1.5 Methodology

The process starts with the retrieval of organized vehicle data from an open public dataset. The entry in the dataset contains technical information like engine capacity, cylinder number, fuel capacity in urban and highway modes, and hybrid type. This raw data is retrieved into memory and pre-processed for quality and uniformity by dropping missing values and duplicate rows.

Finally, categorical features like fuel type, transmission type, and vehicle class are one-hot encoded to prepare them for consumption by machine learning algorithms. Numerical attributes are normalized to set all attributes to a similar scale so that they contribute to the performance and convergence of the model during training.

For the prediction stage, a stacked regression model is constructed with Random Forest and Support Vector Regression (SVR) as base learners. These models examine intricate, nonlinear patterns between input features and carbon emissions. Their predictions are fed into a Gradient Boosting Regressor, which is the final estimator to learn residual patterns and further improve prediction accuracy.

The data is divided into a training set and testing set to measure model generalizability. The stacked model is trained on the training set and tested on the test set by using common metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score to measure its accuracy.

Last, after obtaining predictions, they are rendered in terms of scatter plots plotting predicted versus actual CO₂ values. These are used by users to understand model performance and analyze how various features affect emissions. The whole pipeline from preprocessing to prediction all occurs locally and needs no internet, guaranteeing speed, privacy, and ease of use in a range of settings.

2. Literature Survey

2.1 Related Works

Silvio Lang et al. [1] (2024) developed a MINDFUL approach, which pairs life cycle assessment (LCA) and machine learning for reducing CO₂ emission evaluation complexity. Their software promotes simplified but efficient PCF estimations through an architecture that stresses usability without neglecting computational quality. The design mirrors an increase in applying ML to substitute high-complexity, manual estimation of emissions.

Ashkan Safari et al. [2] (2024) formulated a predictive model framework aimed at predicting CO₂ emissions in the auto industry. They used machine learning algorithms based on data for 46 vehicle companies, achieving correlation scores for the predicted versus observed emissions at very high levels. The paper showcased the practicability of prediction of emissions via historical performance statistics, as shown by the capability of data-based solutions in terms of environmental reviews.

Kwaku Boakye et al. [3] (2023) carried out research using machine learning methods on cement production emissions. With data from the Union Bridge Plant in Maryland, they illustrated that conventional IPCC approaches are lacking and that other AI approaches—like sensitivity analysis and feature engineering—yield more precise CO₂ estimates. Even though their target is the cement industry, their approach is transferable to other industrial emission settings.

Wisthoff et al. [4] (2016) envisioned a multi-layer perceptron (MLP) neural network in connecting product features to LCA outcomes across 37 case studies. The early-stage design inputs are utilized in their system in proposing alternatives which are more environmentally friendly. This method aids proactive, design-stage intervention-based mode and led other similar applications into vehicle production emission modelling.

Prashant Kumar Singh and Prabir Sarkar [5] (2022) presented an ANN-based software that assists in product design optimization taking into account carbon footprint, cost, efficiency, and usability. Unlike traditional tools analyzing a single factor at a time, their tool performs multi-criteria optimization simultaneously. This

study emphasizes the capabilities of integrated AI systems in sustainable product development.

Dhyan R et al. [6] (2024) employed both Random Forest and Convolutional Neural Networks (CNNs) for car carbon emissions analysis. Their analysis used data on fuel consumption and vehicle attributes, and determined that both models demonstrated excellent predictive performance, supporting the adequacy of machine learning methods in automotive sustainability analysis.

Vishwanadham Mandala et al. [7] (2024) investigated the contribution of AI and ML to reducing carbon emissions in the automotive industry. Their research targets AI-based solutions like intelligent energy consumption, EV charging management, and smart transportation systems. Though more comprehensive in scope, their findings are consistent with incorporating AI-based forecasting into vehicle design and production streams.

Gökalp Çınarer et al. [8] (2024) used ML on Türkiye's transportation data to predict CO₂ emissions. They compared models with features such as energy usage, vehicle kilometres, and GDP, and found that XGBoost performed better than legacy models in terms of prediction accuracy. Their results highlight the need for the selection of proper ML models depending on the data and domain of application.

David Tena-Gago et al. [9] (2023) introduced a UWS-LSTM model specifically designed to forecast CO₂ emissions in hybrid vehicles. The model is edge-optimized with 97.5% accuracy to handle non-linear CO₂ oscillations due to real-world powertrain transitions. Their system has potential for real-time vehicle-level emission modelling on constrained hardware.

Chairul Saleh et al. [10] (2024) applied an SVM model for CO₂ prediction using industrial energy consumption data. Their model produced low RMSE values and showed practical usability for emission monitoring and executive-level decision-making. The results confirm the application of SVM for CO₂ forecasting in structured, tabular datasets such as those employed in this project.

Stephanie Martinez [11] (2022) provided an exhaustive master's thesis on the interaction between car use and greenhouse gas emissions. Her research critiqued

the limitations of conventional emissions labelling and underlined the necessity of implementing predictive models for measuring environmental effect at the car level. The research pointed to the utility of merging data science with emissions reporting as a building block argument for solutions that enable intelligent consumer and policy choices through clear analytics.

WorldAutoSteel [12] (2023) published an industrial whitepaper promoting the use of Life Cycle Assessment (LCA) as a key method for quantifying the environmental footprint of automotive materials and design choices. The document highlighted how emerging steel solutions could enhance sustainability throughout a vehicle's life cycle. It also spurred the automobile manufacturers to implement computer software and modelling techniques in order to contrast material selections (aluminium vs. steel), which also assists in the notion of scenario-based emissions forecasting depicted in this work.

G. Asaithambi, M. Treiber, and V. Kanagaraj [13] (2019) wrote a comparative Life Cycle Assessment (LCA) article comparing the environmental effect of traditional internal combustion engine vehicles with electric vehicles. Their study emphasized the need to evaluate emissions outside the usage phase through material extraction, battery manufacturing, and disposal. This holistic LCA methodology emphasized the intricacy of carbon accounting and offered justification for incorporating predictive modelling tools within automotive sustainability systems, particularly when lifecycle information is limited or lacking.

Lindita Bushi [14] (2018) wrote the EDAG Silverado Body Lightweighting Final LCA Report on behalf of the Aluminium Association. The report assessed the environmental advantages of lightweight materials, including aluminium, in automotive body structures. It showed a tremendous decrease in CO₂ emissions if lighter materials were used, even when accounting for increased production energy expenses. This work highlights the necessity of having tools that are able to compare material options within design-stage choices—aligning perfectly with this project's scenario comparison functionality.

Green NCAP [15] (2022) released an easy-to-understand analysis called "How Sustainable is Your Car?", which highlighted the importance of transparency and

consumer information in informing greener vehicle choices. The report supported emissions monitoring that goes beyond tailpipe emissions to include upstream production and energy consumption. It also pointed towards the significance of digital tools that can model and convey emissions results—emphasizing the real-world relevance of machine learning-enabled emissions forecasting platforms.

2.2 Research Gaps and Challenges

The assessment of carbon emissions in the automotive sector has hitherto been based on comprehensive Life Cycle Assessments (LCA), which are computationally intensive, static, and not typically scalable to dynamic, user-specific examination. There have been many publications in literature trying to bring machine learning (ML) into this field, but there are still limitations in generalization, requirements for data, and deployment in practice.

Silvio Lang et al. [1] introduced the MINDFUL framework to streamline Product Carbon Footprint (PCF) calculations using ML, yet their method largely supports editorial tools rather than predictive tools that could adapt to live input. In contrast, our proposed tool leverages vehicle-specific parameters like engine size, number of cylinders, and fuel type to predict CO₂ emissions dynamically, improving real-time applicability and user accessibility.

Ashkan Safari et al. [2] created CO₂ emission predictive models using brand-specific data, which were highly accurate. Yet their models are constructed on brand-limited, narrow datasets, which prevents wider generalizability. Our approach circumvents this by using a stacking ensemble (SVM + Random Forest) on a diverse dataset, increasing robustness and cross-vehicle-type performance.

Boakye et al. [3] treated emission estimation in cement production with ML. Although the method showed potential in industrial use, it was confined to domain-specific stages of emissions like calcination. Our project builds on this idea by simulating emissions in transportation based on direct vehicle specifications, thereby translating the idea to another but emission-intensive industry.

Wisthoff et al. [4] and Singh & Sarkar [5] both investigated the application of artificial neural networks (ANNs) to sustainability in design, but were unable to

incorporate LCA data with usable product metrics. In contrast to these studies, our method avoids the requirement of complete LCA data by utilizing readily available vehicular characteristics, enabling a lean yet efficient predictive system.

Dhyan R. et al. [6] implemented CNNs and Random Forest to analyze emissions in terms of car parameters. Although their research favors the application of ML in emission analysis, they were limited to feature extraction alone without providing real-time prediction interfaces. Our system fills this gap by providing an interactive, web-deployable platform to compare emissions scenario-wise.

Vishwanadham Mandala et al. [7] have explored ML and AI for macro-level emission optimization (e.g., intelligent traffic systems and electric vehicle management). These macro-solutions, as powerful as they are, lack product-level resolution. Our software bridges this void by making predictions at the level of vehicles, enabling granular, user-oriented carbon foot printing.

Çınarar et al. [8] used ML on macroeconomic indicators such as population and GDP to predict transportation emissions. It is macro-statistical and not appropriate for single decision-making. Our project redirects the focus to micro-level prediction, creating tangible value for manufacturers, policymakers, and consumers in the form of individualized CO₂ predictions.

Tena-Gago et al. [9] investigated LSTM models for hybrid vehicle CO₂ estimation optimized for IoT devices. Their real-time performance attention is useful but restricted to hybrid vehicles. Our implementation is more general to other combustion-engine vehicles while providing a lightweight model that can be extended further to IoT environments.

Saleh et al. [10] showed successful SVM-based CO₂ forecasting employing energy use data. While their solution realized a low RMSE, it focused on industrial energy sources instead of transportation. Our tool applies SVM in a stacking model and re-targets the input features to align with automotive parameters, filling a fundamental gap in carbon footprint modelling.

Jin et al. [11] reviewed a wide variety of carbon emission prediction models and found that many depend on intricate data inputs not readily available in real time or

accessible to end users. Our system mitigates this by creating a light input feature set that includes vehicle characteristics such as engine size and fuel economy so that the model is more deployable in user applications.

Tena-Gago et al. [12] proposed a UWS-LSTM model specifically designed for hybrid vehicles with a high degree of precision. The limitation in the use of temporal data and powertrain-dependent tuning restricts flexibility, compared to our model that supports static vehicular attributes with more generalized support for a broad range of combustion-engine vehicles.

Chairul Saleh et al. [13] were concerned with predicting CO₂ emissions based on energy consumption measures in one particular industrial industry (alcohol production), tuning SVM parameters to increase accuracy. Though useful, this research is still limited to stationary emissions. Our model transfers the SVM model to the mobile sector (cars), optimizing for patterns of vehicular features to mimic actual-world transport-based carbon emissions.

Stephanie Martinez [14] highlighted vehicle-emission linkages but mostly employed traditional statistical techniques. Such techniques are not flexible and accurate in non-linear feature spaces. Our approach improves prediction accuracy by ensemble learning and ongoing tuning with error metrics such as RMSE and R², more in line with contemporary ML practices.

Green NCAP [15] provides useful information on sustainability assessment using Life Cycle Assessments and is extensively referenced for benchmarking purposes. LCA tools themselves are static and data-intensive, and not geared towards quick comparative analysis. Our scenario comparison capability addresses this shortcoming by enabling users to run "what-if" scenarios (e.g., alternate engine displacements or fuel types) and obtain real-time emission projections.

3. Requirements

3.1 Software Requirements

3.1.1 Functional Requirements

The system should be able to accept structured input through CSV files or web form submissions with major vehicle attributes like engine displacement, number of cylinders, fuel type, and mileage. All this data should be pre-processed automatically with scaling and cleaning processes before model inference.

The software's main functionality is to calculate the vehicle's carbon dioxide (CO₂) emissions with the help of a trained stacking regression model based on Support Vector Regressor (SVR) and Random Forest. After accepting input, the system should provide a real-time prediction of CO₂ emissions in grams per kilometre.

An optional scenario comparison functionality is a core part of the tool. Emissions from different vehicle specifications must be compared by users, and the results are presented as interactive tables or charts that provide visual representation of each configuration's sustainability.

The system should also offer exportable reports of projected emissions together with related features. Reports have to be in PDF or JSON format and include visualizations for ease of interpretation.

Lastly, the system is designed to be a web-based responsive application with React as the frontend, Python (Flask or Fast API) as backend logic, and MongoDB as the storage of user inputs and previous results. Model inference has to be run locally or on a secure server for user privacy and portability of the system.

3.1.2 Non-Functional Requirements

Accuracy is an important non-functional requirement. The system predictions should be assessed based on RMSE and R² values, and the model should be verified to possess high generalizability across different types of vehicles and ranges of features without overfitting.

Responsiveness is to be preserved throughout each stage of the prediction pipeline. Right from when user input takes place to displaying the results, overall delay shouldn't be over two seconds such that users have a chance for real-time comparison and get quasi-instant responses.

Data privacy is paramount. All inputs and calculations need to be handled locally or on trusted internal infrastructure. No data must be sent to third-party services, and system access needs to be protected with basic authentication or access tokens.

Cross-platform compatibility is necessary. The application needs to run flawlessly on common browsers and be deployable on both Windows and Linux environments without needing advanced infrastructure or cloud reliance.

Maintainability ensures long-term use. The codebase must be modular and properly documented to enable future upgrades like extra input capabilities or better ML models. It must be able to update without interfering with current deployments or needing full reinstallation.

3.1.3 Software Tools and Frameworks

The software is built utilizing Python, taking advantage of its mature ecosystem for data science, machine learning, and data visualization. Python's strong support for regression modelling, data manipulation, and preprocessing rendered it a suitable option for the development of a CO₂ emissions forecasting tool.

Scikit-learn is the major machine learning library, providing properly optimized versions of both the Support Vector Regression (SVR) and Random Forest Regression models employed in this project. It also provides model performance evaluation with default metrics such as R^2 score, MAE, and RMSE.

In order to increase the performance and predictive power, a Stacking Regressor ensemble is applied, which amalgamates the strengths of SVR and Random Forest for increased accuracy.

For preprocessing data, Pandas and NumPy libraries are utilized for dataset management, dealing with missing values, and feature transformations. StandardScaler is applied for feature scaling in order to achieve model convergence as well as to make the model consistent across various variable scales.

Matplotlib and Seaborn libraries are employed for visualizing the performance of the model, including plotting actual against predicted CO₂ emissions, as well as finding patterns in the dataset.

The model can be integrated into a Flask-based web application, allowing for a user-friendly front-end wherein users can input vehicular specifications (e.g., engine capacity, fuel type, mileage) and get real-time CO₂ emission predictions.

In the future, the model and the interface can be made available as a web application, utilizing APIs for real-time user feedback and potentially being merged with live data from vehicular databases.

3.2 Hardware Requirements

3.2.1 Minimum Hardware Configuration

As the CO₂ emission forecast system is not hardware-consumptive and based on organized tabular data, a simple hardware configuration is enough for development as well as deployment. An Intel Core i5 processor combined with 8GB of RAM and at least a 256GB SSD will more than suffice for data preprocessing, model building, and testing.

GPU acceleration is not necessary, since the models employed—Support Vector Regression, Random Forest, and Stacking Regressor—are computationally light and can be run well on CPU-based systems. But having access to a separate GPU (e.g., an NVIDIA GTX 1050 or better) can accelerate training and come in handy when testing larger datasets or more sophisticated ensembles.

For deployment in a user-facing web application, the system can be hosted on local servers or cloud environments with matching specifications. External peripherals such as cameras or microphones are not required since the tool

works based on numerical vehicle feature data (e.g., engine capacity, fuel type, mileage).

The app can run in an offline mode after the necessary libraries and data files are installed, making internet connectivity unnecessary at all times. For remote access or real-time data updates scenarios, normal network connectivity is adequate.

3.2.2 Recommended Hardware Configuration

For better performance, particularly when dealing with larger data sets, conducting hyperparameter optimization, or incorporating other analytic functions, a more powerful configuration is recommended. A setup featuring an Intel Core i7 or AMD Ryzen 7 processor, alongside 16GB or higher of RAM, facilitates increased data processing speed and smoother multitasking in model training and testing.

Although the system is capable of running on CPUs, adding a specialized GPU (like an NVIDIA GTX 1660, RTX 2060, or more) greatly improves training times for ensemble models and allows for experimenting with more intricate learning pipelines, including eventual inclusion of deep learning models.

A 512GB SSD will be used for handling datasets, temporary computation data, and pre-processed outputs seamlessly. For deployment over a period of time or as part of a web application with logging functions, the machine should be compatible with additional storage through external HDDs or NAS.

In deployment environments, especially in institutional or industrial environments, reliable power supply and UPS backup are advised to prevent disruption during extended model training or batch testing. For server-based deployments, normal cooling and secure physical access must be provided to ensure system integrity and operational reliability.

4. Proposed System

4.1 Overview of System Architecture

The system, introduced in the paper, is a machine learning carbon footprint predictor for the operation phase of vehicles. This tool does not measure full life cycle assessment (LCA) over the sourcing, production, and use phases, but monitors primarily CO₂ emissions through the technical characteristics (mileage, cylinders, displacement, weight, and fuel).

The model is a data-driven forecasting tool trained on freely accessible automotive emissions datasets. Instead of relying on predefined vehicle features, the tool receives user-entered information as input to estimate emissions based on Random Forest Regressor and Support Vector Regression (SVR) machine learning models. It enables vehicle producers, policy analysts, and environmentally conscious individuals to compare the vehicle configurations and select lower emission alternatives based on quantifiable criterion.

The system is developed as the Jupyter Notebook prototype, including data preprocessing, model training, performance evaluation, and visuals. It also allows for modular expansion for further inclusion into web applications or dashboards.

4.2 Core Modules and Functionalities

The designed system is an amalgam of several functional modules which automatically integrate to build an entire pipeline for vehicles' carbon footprint prediction from technical and fuel efficiency features. All modules are kept small and tight and with clearly defined responsibility, to keep it modular, scalable and maintainable.

4.2.1 Data Acquisition Module

It is responsible for sourcing, understanding, and preparing the dataset that fuels the machine learning models. The quality and comprehensiveness of the input data directly affect the accuracy of the model's predictions.

Functionality: The module is part of the preprocessing process to transform raw input data to extract the underlying structure and train machine learning

algorithm on them. The dataset is based on the automobile tend features such as the size of the machine, number of cylinders, fuel consumption in city and highway etc., CO₂ emissions. The process of preprocessing comprises of:

- Cleaning of dataset for removing missing values, duplicate records, inconsistent data.
- Categorical variables (e.g., types of transmissions) will be encoded into numerical values.
- Normalisation or standardisation of input data to constant input range.
- Dividing the data again into testing and training sets to evaluate the model.

To clean up and get consistent and well formatted dataset for efficient training of a model and better predictive accuracy.

4.2.2 Model Building and Training Module

Functionality: This module concerns with construction and training of prediction model using SVR. SVR is a good candidate for regression tasks with continuous target variables such as the prediction of CO₂ emission. This module does the following functions:

- Modelling: SVR is used with various kernel functions, including the radial basis function (RBF) kernel, to capture non-linear relationships.
- Model Training: The SVR model is trained on the pre-processed vehicle dataset to establish relations between input features and target CO₂ emissions.
- DR How do you evaluate performance? Performance is evaluated by the error criteria such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R² Score.

We will build a regression model which can be used to predict `co2 emissions of cars based on various parameters of cars for classification.

4.2.3 Carbon Footprint Prediction Module

Functionality: This module uses the trained SVR model to predict the carbon emissions given new or unseen vehicle data. it is the building block for producing the useful output from the machine learning model. Key functionalities include:

- **Input Processing:** It takes input features from the vehicle such as engine volume, cylinder number and fuel consumption amount.
- **Real-Time Prediction:** Utilizes a trained SVR model to predict CO₂ emissions for one or multiple input samples.

Objective - This research aims to predict CO₂ emissions of cars with high accuracy, since it is important to be familiar with CO₂ emissions, from the point of view of impact on environment, based on car model and few of its specifications.

4.2.4 Result Generation Module

Functionality: This module encapsulates the visualization and documentation of prediction and model performance. It also makes the outputs interpretable, useful, and reportable. Its functions are comprised in the following:

- **Visualization:** Creates graphs to compare predicted emissions to actuals, and also plots residuals for evaluating accuracy of predictions.
- **Performance Metrics Reporting:** Review of performance evaluation scores (for example, RMSE and R²) for a numerical assessment of the quality of the model.

To clarify simulation results and findings in communicating the outputs and facilitate informed decision making and documentation for research or implementation.

4.3 System Architecture

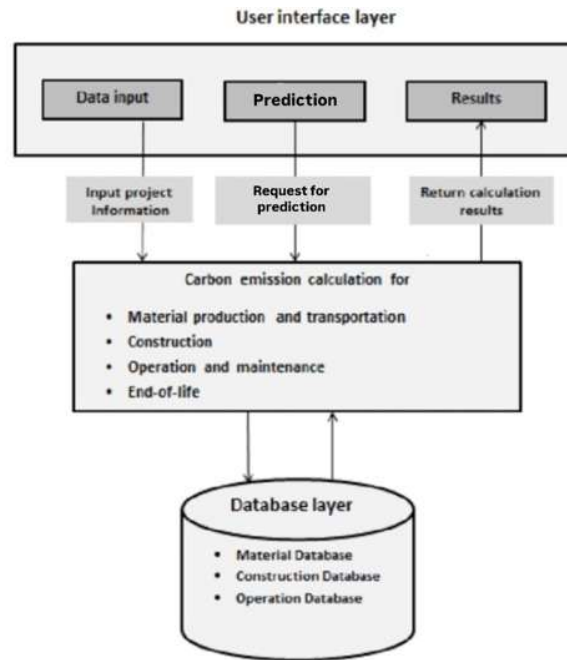


Fig 4.1: System Architecture

The system architecture includes a front end and back end. The front-end obtains information such as engine displacement, number of cylinders, fuel type, transmission type, and fuel economy (city/highway). The data is then communicated to the backend and a number of predetermined machine learning processes are executed. These steps involve input validation, feature transformation and encoding and, scaling with a pre-trained MinMax Scaler, and prediction of carbon footprints with a trained Random Forest Regressor model. The last processed output as represented by the estimated.CO2 emissions in grams per kilometer are returned to the frontend to be presented in a user-accessible manner, allowing users to easily understand and compare the environmental impact of different vehicle configurations.

4.4 User Interface

The User Interface of the application contains the following components:

- **Landing page:** The application starts at a Landing Page that welcomes the user and two options, either "Sign Up", if they are a new user, or "Login", if

they are a returning user. This page details the tool's use and guides the user into the authentication flow.

- **Signup Page:** The Signup Page allows new users to create an account using their username, email and password. The signup page also has optional fields for height and weights so the user can personalize their experience. Validation and creation of user accounts is handled via Firebase authentication.
- **Login Page:** The Login Page offers fields for a user to input their email and password, which allows them to login to their account. The login page has a couple of other options, for example: "Forgot Password?" and a link to the Signup page to allow a smooth experience for new or returning users.
- **Home Page:** The Home Page is presented once a user successfully logs in or signs up, and serves as the main dashboard of the application. The user will have three main options to choose from: predicting CO₂ emissions, comparing vehicles or viewing their history.
- **Predict CO₂ Emissions Page:** Users will gain access to this page and can input the parameters of specific vehicles, such as engine size, number of cylinders, fuel type, transmission type and fuel consumption values. Once the user completes the form, the user can submit the form.
- **Compare Vehicles Screen:** This screen allows users to fill out the details for both vehicles. After the user submits both vehicles configuration, the table will calculate and return the emissions for both vehicles and then also return a visual representation of the difference. This will allow the user to assess the emissions from the vehicles based on the selections made.
- **History Screen:** The History Screen will show all individual predictions or comparisons made by the user in a table format. Each table entry will show the date, vehicle specs summary, emission values plus buttons to view it or delete it, along with a filter functionality to view records by date or type.
- **Error Screen:** If any errors in the system occur or invalid entry is input by the user, the user will proceed to an Error Screen, which displays a clear message and a Retry button or link back to the Home Screen. The goal is for the user to have a direct route to recovery without confusion.

5. System Design

5.1 UML Diagrams

5.1.1 Class Diagram

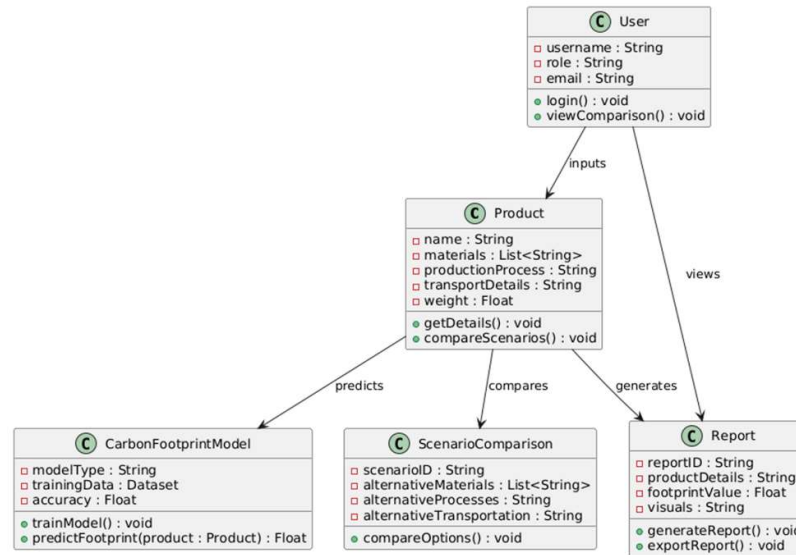


Fig 5.1: Class Diagram

This class diagram shows the structure and interactions of an intended system to analyze and report the carbon footprint of products. The system possesses five primary classes with a specific role so that it is modifiable and there is segregation of concerns.

- **User:**

User class in the diagram depicts a person interacting with the carbon footprint analysis system. The class contains user-specific information and supports user authentication and comparison report access functionalities.

Key Methods:

- `login(): void` — Authenticates the user and grants access to the system.
- `viewComparison(): void` — Allows the user to view carbon footprint comparisons and reports for different products or scenarios.

- **Product:**

The Product class is an actual product whose carbon footprint is to be analyzed. It contains detailed information on the product's composition, production process, transport, and weight, all of which are required inputs for footprint estimation and scenario comparison.

Key Methods:

- `getDetails(): void` — Retrieves and displays all stored information about the product.
- `compareScenarios(): void` — Initiates a comparison of alternative production or transport scenarios using the `ScenarioComparison` class.

- **CarbonFootprintModel:**

The CarbonFootprintModel class is a forecasting model that estimates the carbon footprint of a product. It learns patterns and makes predictions about emissions from historical or simulated training data based on product-specific inputs like materials, production processes, and transport.

Methods:

- `trainModel(): void` — Trains the model using the provided dataset to improve prediction performance.
- `predictFootprint(product: Product): Float` — Takes a Product as input and returns a float value representing its estimated carbon footprint.

- **ScenarioComparison:**

The ScenarioComparison class enables comparison of various environmental impact scenarios of a product by altering its materials, manufacturing processes, or shipping. It can be used by decision-makers to compare green options and choose the greenest configuration.

Key Methods:

- `compareOptions(): void` — Executes the comparison logic, analyzing and possibly ranking the environmental impacts of various scenarios.

- **Report:**

The Report class is the result of the carbon footprint calculation. It aggregates detailed product information, estimated emissions, and any comparison results into a visualization- or exportable form to present or reuse.

Key Methods:

- `generateReport(): void` — Creates a comprehensive report using product data and prediction results.
- `exportReport(): void` — Allows the user to export the report, typically in formats like PDF or CSV.

5.1.2 Sequence Diagram

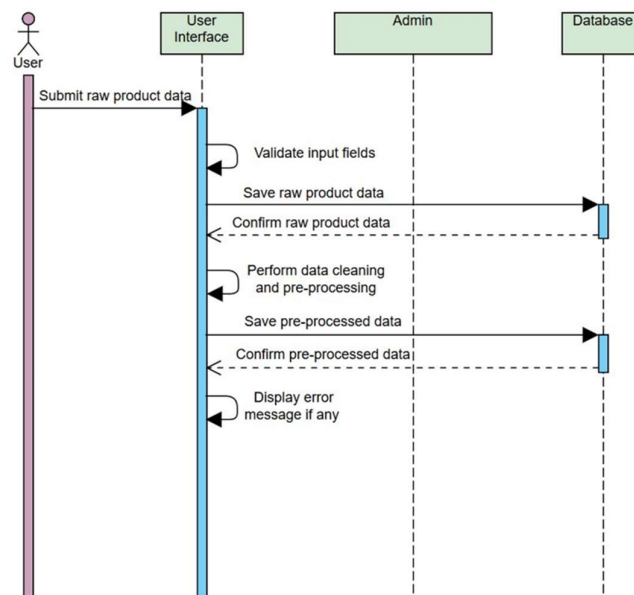


Fig 5.2: Sequence Diagram

This sequence diagram shows the interaction process that happens when a User submits raw product data to the system. The system will check, store, clean, and process the data, verifying at every step. When there is any kind of error, it will be passed on to the user interface for the user to accept.

- **Actors and Participants:**

- User: An external actor who initiates the process by submitting raw product data.
- User Interface: Handles input validation, initiates data saving and processing, and displays messages to the user.
- Admin: Performs data cleaning and pre-processing tasks (may be a manual or automated role).
- Database: Stores both raw and pre-processed product data

- **Interaction Flow:**

- User → User Interface: Submit raw product data: The process starts when the user submits the product data through the UI.
- User Interface: Validate input fields: UI validates the data fields for correctness and completeness.
- User Interface → Database: Save raw product data: After validation, the raw data is stored in the database.
- Database → User Interface: Confirm raw product data: The database confirms that raw data has been successfully saved.
- User Interface → Admin: Perform data cleaning and pre-processing: Admin component receives the raw data and performs necessary cleaning and transformations.
- Admin → Database: Save pre-processed data: The cleaned and formatted data is stored back in the database.
- Database → Admin → User Interface: Confirm pre-processed data: A confirmation that pre-processed data has been saved is sent back through Admin to the UI.
- User Interface: Display error message if any: If errors occur at any stage (e.g., validation or saving failures), they are displayed to the user.

5.1.3 Activity Diagram

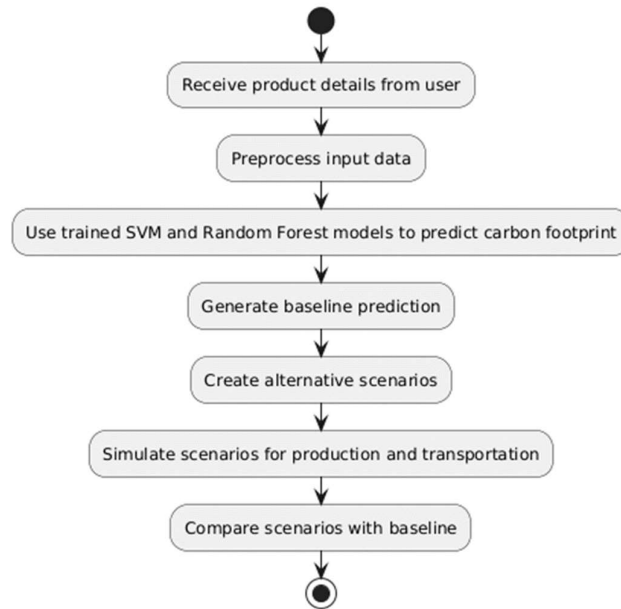


Fig 5.3: Activity Diagram

This activity diagram presents the overall process of carbon footprint forecasting and comparison between scenarios of a product at a general level. It presents the process of sequential activities from data intake of a product to comparing various environmental effect scenarios on the basis of machine learning models trained beforehand.

- **Swimlanes:**

The activity diagram is structured with swimlanes to graphically partition the responsibilities of various components or actors:

- User – Provides product details.
- System (UI + Backend) – Handles preprocessing, prediction, and scenario simulation.
- ML Models (SVM, Random Forest) – Perform carbon footprint prediction.
- Scenario Analysis Module – Handles scenario generation and comparison.
- AccuracyReport: Produces and displays system performance report.

- **Activity Flow:**

- Start Node – Denoted by the solid black circle.
- Receive product details from user – User submits product information (materials, weight, process, etc.).
- Preprocess input data – Clean and format the data to ensure it's ready for prediction.
- Use trained SVM and Random Forest models to predict carbon footprint – Apply machine learning models to estimate emissions.
- Generate baseline prediction – Establish a reference value for the original product configuration.
- Create alternative scenarios – Define variations in materials, processes, or transport for analysis.
- Simulate scenarios for production and transportation – Evaluate the impact of each scenario.
- Compare scenarios with baseline – Assess how each alternative fares compared to the baseline.
- End Node – Denoted by the concentric circle.

5.1.4 Use Case Diagram

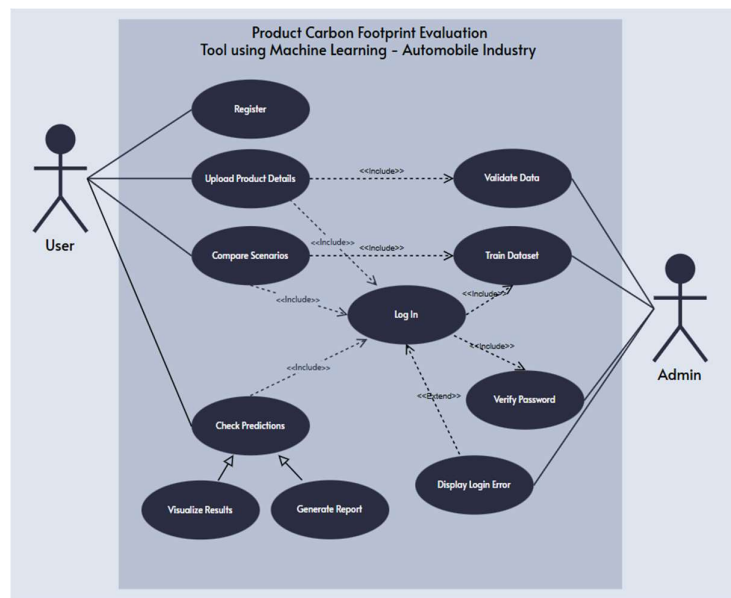


Fig 5.4: Use Case Diagram

This use case diagram illustrates the core interactions within the Product Carbon Footprint Evaluation Tool for the automobile industry. It shows how users and admins perform tasks such as uploading product data, validating information, training models, and generating predictions. The system supports scenario comparison and report generation, enabling accurate carbon footprint analysis through collaborative functionality.

6. Implementation

This project is implemented as an interactive, web-based tool designed to assist users in understanding the environmental impact of automobile usage based on specific vehicle attributes. The system enables users to input information on vehicle specifications such as Engine Size (L), Number of Cylinders, Fuel Consumption in City and Highway (L/100 km), and Hybrid status, which are then processed by a trained machine learning model to estimate the CO₂ emissions in grams per kilometre (g/km). The implementation has the following main modules:

1. **Data Collection and Preprocessing Module** - This module focuses on preparing raw data for analysis and model training through data cleaning which involves removing duplicates, handling missing values, and addressing outliers. It also includes data transformation by normalizing, scaling, and encoding categorical variables, as well as feature selection to identify significant variables influencing carbon footprint prediction. Additionally, the module ensures data quality and consistency through validation, with the overall goal of producing a clean, structured, and reliable dataset for accurate and consistent model performance.
2. **Model Building and Training Module** - This module is responsible for training and building machine learning models to predict carbon footprint using algorithms such as SVM, which provides robust boundary-based classification, and Random Forest, which provides robust decision-tree-based predictions. It involves training the models on historical data, hyperparameter optimization to fine-tune them using adjustments such as kernel types for SVM and tree depth or number for Random Forest. The module also evaluates performance using measures such as accuracy, RMSE, and MSE to determine the best model. The overall objective is to develop a reliable model that can evaluate a variety of factors influencing carbon footprint.
3. **Carbon Footprint Prediction and Scenario Analysis Module** - The objective of this module is to predict carbon footprint and project scenario analysis based on user inputs by estimating emissions from factors. The module allows users to enter and compare various scenarios to understand the impact of each on carbon emission. Through dynamic analysis, it provides real-time feedback and

insights on how input changes affect emissions, ultimately helping users make data-driven decisions and optimize operations for greater sustainability.

4. **Results and Visualization Module** - This module generates results and actionable insights based on the analysis by creating visual and textual reports of carbon footprint forecasts and comparisons. These outputs are presented in the form of graphs, charts, in the dashboards to facilitate direct and easy interpretation, ultimately with the end goal of clearly communicating findings and enabling informed decision-making.

Together, these modules form a cohesive pipeline for evaluating the carbon footprint associated with automobile usage. From gathering and processing vehicle data to building a machine learning model and delivering interactive predictions via a web interface, the system is structured to be efficient, informative, and user-friendly. The following sections explain the algorithmic flow, code structure, and functional breakdown of each module.

6.1 Data Collection and Preprocessing Module

6.1.1 Purpose

This module is responsible for dataset collection and preparation which is used in training the carbon footprint prediction model. The dataset is made up of technical specifications of the vehicles and the corresponding CO₂ emissions. There should be clean and consistent data to develop a correct and reliable model.

6.1.2 Algorithmic Flow

- Load the raw vehicle dataset from CSV.
- Remove irrelevant columns which may not be useful for prediction.
- Handle missing values and inconsistent entries.
- Encode the categorical variables like Is_Hybrid.
- Generate new features.
- Normalize numerical columns.
- Split data into training and testing sets.

6.1.3 Dataset Description

The dataset used in this study was sourced from “Canadian Vehicle Fuel Consumption data”, which offers comprehensive information on vehicle performance and emissions. It includes a variety of attributes that are relevant for analyzing and predicting carbon emissions from vehicles. Key features in the dataset include:

- Engine Size (in liters): Represents the total volume of all the engine's cylinders, which often correlates with the vehicle's power output and fuel consumption.
- Number of Cylinders: Indicates the number of cylinders in the engine, affecting both engine efficiency and emission levels.
- Fuel Consumption in City and Highway (L/100 km): Provides separate measures of fuel efficiency under city driving, which usually has more stop-and-go traffic patterns, and under highway driving, where the speeds are more consistent and constant throughout the trip.
- CO₂ Emissions (g/km): The amount of carbon dioxide emitted per kilometre driven, serving as the target variable for carbon footprint prediction.
- Hybrid Status (Yes/No): Indicates whether the vehicle uses a hybrid engine, which can significantly influence both fuel consumption and emissions.

These attributes are critical for training machine learning models aimed at predicting and analyzing vehicle-related carbon emissions.

6.1.4 Code Outline

The `data_preprocessing.py` script is responsible for handling raw data preparation before training the machine learning model. The following outline summarizes the structure and primary responsibilities of each function:

- `load_data(filepath)`: This function is responsible for loading the dataset from a CSV file using the provided file path. It returns a Pandas DataFrame that serves as the base for all further data processing.

- `clean_data(df)`: This function cleans the raw dataset by removing irrelevant or redundant columns, such as model names or identifiers that do not contribute to the prediction task. It also handles missing values by applying appropriate imputation strategies to maintain the consistency and quality of the dataset.
- `encode_features(df)`: This function transforms categorical features into numerical formats suitable for machine learning algorithms. In particular, it encodes the Is Hybrid column by mapping "Yes" to 1 and "No" to 0, enabling binary representation of this feature.
- `feature_engineering(df)`: This function generates new features that may improve model performance. For example, it calculates the average fuel consumption using city and highway fuel consumption values. Additional derived metrics, such as efficiency indices, can also be added in this stage.
- `normalize_and_split(df, target_column)`: This function applies normalization to the numerical features using Min-Max scaling to ensure uniformity in feature ranges. It then splits the dataset into training and testing subsets, typically using an 80:20 ratio. The function returns the input and output sets (`X_train`, `X_test`, `y_train`, and `y_test`) for model training and evaluation.

6.1.5 Overall Flow

This module is the foundational step of the carbon footprint estimation system. It begins with importing the raw vehicle data set from a CSV file with features like engine size, number of cylinders, fuel consumption values, and hybrid status. The data is cleaned to remove irrelevant columns and to replace missing values in order to ensure consistency and reliability. Categorical features are encoded into numerical form in order to make them machine learning algorithm-compatible. Feature engineering is then applied to obtain new informative features that provide richer information for model learning. The data set is finally normalized by Min-Max scaling and divided into training and test sets in preparation for model building. This pipeline ensures that the

input data is accurate, well-formatted, and optimized for the following steps of the system.

6.2 Model Building and Training Module

6.2.1 Purpose

This module focuses on selecting, training, and validating the machine learning model that predicts carbon emissions, with the purpose of building an accurate and reliable model for analyzing factors that influence carbon output.

6.2.2 Algorithmic Flow

- Import cleaned and split data from preprocessing.
- Choose appropriate regression models.
- Train each model on the training data.
- Evaluate using test data and metrics.
- Select the best-performing model.
- Serialize the model for deployment.

6.2.3 Code Outline

The `model_training.py` script is a key component, focused on training, evaluating, and saving regression models. It handles the process of selecting and fitting various regression algorithms to the provided training data, assessing their performance on unseen test data, and saving the best-performing model for future use. By dividing these tasks into distinct functions, the script ensures modularity and ease of use, allowing for efficient experimentation and deployment of machine learning models.

- `train_models(X_train, y_train)`: This function is responsible for training multiple regression models on the provided training data. It specifically implements ensemble modeling by including both Support Vector Regression (SVR) and Random Forest Regression. Each model is fitted on the training data to learn the underlying patterns and relationships between the input features and the target variable. By training a combination of models, this approach allows for performance comparison and provides flexibility in selecting the most accurate and

generalizable model based on evaluation metrics. This strategy enhances the robustness of the modeling process, ensuring better performance across a variety of data distributions.

- `evaluate_model(model, X_test, y_test)`: After a model has been trained, it is crucial to assess its performance on unseen data. This function computes relevant evaluation metrics to determine how well the model generalizes to new data. Common metrics may include Mean Squared Error (MSE), R-squared etc. This helps in understanding whether the model is underfitting, overfitting, or performing adequately.
- `save_model(model)`: After a model has been trained and evaluated, it is often necessary to persist it for future use, especially in production environments. The `save_model` function serializes the trained model using `joblib`, which is an efficient library for saving and loading Python objects.

6.2.4 Overall Flow

This module handles the process of training and developing a machine learning model using the prepared data to predict the emissions. It starts by training different types of regression models, specifically Support Vector Regression (SVR) and Random Forest Regression. These models are chosen to take advantage of both linear and ensemble-based approaches, helping improve accuracy and generalization. After training, the models are tested using a separate portion of the data to check how well they perform. The performance of each model is compared using suitable evaluation metrics. Based on this comparison, the best-performing model is selected. Finally, the chosen model is saved so it can be reused later without the need for retraining. This overall flow ensures that the system can make accurate predictions and be easily integrated into real-world applications.

6.3 Carbon Footprint Prediction and Scenario Analysis Module

6.3.1 Purpose

This module integrates the trained machine learning model into a user-friendly web application, providing an accessible interface for users to interact

with the system. It provides users to input relevant data and receive immediate carbon emission predictions, making the model's insights practical and actionable. Additionally, the module supports interactive scenario testing, enabling users to explore how changes in input affect carbon emissions. This functionality empowers users to make informed, sustainability-focused decisions through real-time feedback and visualized comparisons.

6.3.2 Algorithmic Flow

- Take user input from the form.
- Preprocess input (matching training data format).
- Load serialized model.
- Predict CO₂ emissions based on user input.
- If scenario comparison is enabled:
 - Repeat for alternative inputs.
 - Compare both outputs.

6.3.3 Code Outline

The `app.py` file serves as the central control script for running the web application. It connects the user interface with the backend and the machine learning logic. This script is built using Flask, a lightweight web framework in Python, which allows users to interact with the trained model through a browser-based form. The main responsibilities include receiving user inputs via web requests, processing these inputs, passing them to the trained model for prediction, and displaying the results. Additionally, it supports a scenario comparison feature, where users can input two different sets of data and receive a side-by-side analysis of the predicted carbon emissions.

- `@app.route('/', methods=['GET', 'POST'])`: This is the main route of the web application, which handles both GET and POST requests. When a user visits the page, a form is displayed to collect input data. If the user submits the form (POST request), the server processes the input data, makes predictions using the machine learning model, and displays the results on the same page. This route acts as the entry and interaction point between the user and the prediction system.

- `predict_emission(data)`: This function takes a single set of user inputs, which have been collected and pre-processed, and sends them to the trained machine learning model. It then returns the predicted carbon emission value. This function allows the backend to separate the logic of prediction from the web route, making the code modular and easier to maintain.
- `compare_scenarios(data1, data2)`: This function enables the comparison of two different input scenarios. It accepts two sets of processed input data, sends each to the machine learning model, and calculates the difference in predicted carbon emissions between them. This feature is particularly useful for users who want to analyze how changes in input variables can affect the environmental impact. The comparison results are then displayed in a user-friendly format on the web page.

6.3.4 Overall Flow

This module is designed to allow users to estimate the environmental impact of different production configurations. It starts with a web-based user interface, where users can input relevant data. When the form is submitted, the application processes this input and sends it to a pre-trained machine learning model. The model then predicts the estimated carbon emissions based on the input data. If the user submits two sets of data, the system calculates and compares the carbon footprint for both scenarios. This comparison helps users understand how small changes in inputs can influence overall emissions, promoting more sustainable decision-making. The results—either a single prediction or a comparison—are then displayed on the web interface in a user-friendly format. Thus, it provides a complete end-to-end workflow for emission analysis.

6.4 Results and Visualization Module

6.4.1 Purpose

The final module focuses on projecting the results of predictions and comparisons in a clear, understandable format to support effective decision-making. It transforms raw output data into meaningful visual and textual summaries, using charts, graphs, and dashboards to highlight key insights. The

goal is to enhance transparency, improve communication of model outcomes, and enable stakeholders to confidently act on the insights provided.

6.4.2 Algorithmic Flow

- Display predicted result with breakdown.
- If scenario comparison is enabled:
 - Calculate difference and % reduction/increase.
 - Generate visual plots (bar chart or pie chart).

6.4.3 Code Outline

- `result.html`: The `result.html` file is an HTML template that is responsible for displaying the output of the prediction to the user. It uses Jinja templating, which allows dynamic content to be inserted into the HTML page based on the data sent from the Flask backend. This template renders the predicted CO₂ emission values and, if a comparison is made, it also shows the difference between two scenarios in a clear and structured format.
- `static/plot.js`: The `plot.js` file is an optional JavaScript script used to enhance the user experience through client-side data visualization. When included, it can render basic charts such as bar graphs using libraries like Chart.js. These visual elements help users better understand the differences in carbon footprint between scenarios, making the analysis more intuitive and engaging.

6.4.4 Overall Flow

This module is responsible for presenting the predicted carbon emissions and scenario comparisons to the user in a clear and meaningful way. After the machine learning model processes the input data and returns the prediction(s), this module takes over to handle the display. The processed results are passed from the backend to an HTML template using Jinja, which dynamically updates the web page with the prediction values. If the user has submitted two input sets, the module also calculates and displays the difference in emissions between the two scenarios. To enhance user understanding, the module includes interactive visualizations using JavaScript. These charts can present data in formats like bar graphs making complex numerical output easier

to interpret at a glance. This flow ensures that users not only receive accurate predictions but can also visualize the outcomes effectively for better decision-making.

7. Testing

The testing phase is an essential part of the development lifecycle to ensure that the CO₂ Emission Prediction System performs correctly, reliably, and consistently in all usage scenarios. The stage confirms that the machine learning pipeline from data ingestion and preprocessing to prediction and visualization works as planned and delivers performance expectations.

Both functional and non-functional features of the system are tested. Functional testing covers ensuring the system responds correctly to various inputs from the users, like varying combinations of car attributes (e.g., fuel type, engine size, cylinder number), and returns realistic CO₂ emission estimates. It also checks whether preprocessing of data (e.g., normalization, encoding, scaling) is correctly done and consistently performed on all inputs.

Non-functional testing emphasizes system performance with respect to accuracy, response time, and scalability. Performance of the model is measured based on metrics like R² score, RMSE, and MAE to establish the model's generalization to unseen data. Response time of the system, especially for real-time prediction through a web interface, is also tested to make it usable.

Edge case testing is done by using outliers or unusual combinations of input to check the robustness of models. The system is checked for missing or corrupt data, incorrect feature values, and duplicate input to ensure error handling mechanisms and robustness against adverse situations.

Overall, the testing phase confirms that the system is not just working but is robust, scalable, and capable of providing reliable predictions across a vast number of situations.

7.1 Black Box Testing

Black Box Testing analyzes the system's behavior without needing to know its internal code and algorithm structure. In the case of the CO₂ Emission Prediction Tool, this type of testing is only concerned with the manner in which the system reacts to various user inputs and whether it generates correct and reliable carbon emission predictions.

This method is particularly effective for testing the end-user experience, verifying that the tool performs as intended with different combinations of inputs. It mimics actual usage by submitting various vehicle specifications like engine size, fuel type, number of cylinders, and fuel consumption factors to check the response output. The following are some of the black box test cases for our system:

7.1.1 Black Box Test Case 1: Handling incorrect input

This test case verifies if the application is able to identify when a wrong input is entered for any field. For instance, if fuel type is entered as anything except Z,D,X,E or N, it should display a message to the user that an incorrect value has been entered for that field.

Field	Description
Test ID	BB-001
Test Objective	Handling incorrect input
Input	Any fuel type entered in the “fuel type” field other than Z,D,Z,E,N.
Expected Output	Error message when form is submitted under “fuel type” field indicating incorrect value.
Actual Result	Error message when form is submitted under “fuel type” field indicating incorrect value.
Status	Pass

Table 7.1: Black Box Test Case 1

7.1.2 Black Box Test Case 2: Handling empty input field Scenario

This test case guarantees that the application will inform the user in case they try to submit the form without entering a value in all input fields. An error message should be displayed below any/all empty fields.

Field	Description
Test ID	BB-002
Test Objective	Handling empty input field scenario
Input	Submission of form with empty field(s)
Expected Output	Message: "Please fill this field"
Actual Result	Message: "Please fill this field"
Status	Pass

Table 7.2: Black Box Test Case 2

7.2 White Box Testing

White Box Testing tests the internal logic and structure of the CO₂ Emission Prediction System. It ensures that data preprocessing steps such as normalization and encoding are correctly done, and that the machine learning algorithms (SVR, Random Forest, and Stacking Regressor) behave as expected.

This testing also checks whether measures such as RMSE and R² are calculated correctly and that data flow between modules is regular. By emphasizing code-level correctness, White Box Testing identifies logical errors, inefficiencies, and makes the system robust and reliable. Below are some White Box Test Cases for your system, aligned with the focus on the internal workings and flow:

7.2.1 White Box Test Case 1: Ensure correct application of StandardScaler

This test case verifies that StandardScaler i.e, normalization of data is done only after train-test splitting, so that data leakage is prevented.

Field	Description
Test ID	WB-001
Test Objective	Ensure correct application of StandardScaler before train-test split
Input	Correct input feature values to model
Expected Output	CO2 emissions prediction with an accuracy of less than 100, indicating no data leakage
Actual Result	Prediction with an accuracy less than 100.
Status	Pass

Table 7.3: White Box Test Case 1

7.2.2 White Box Test Case 1: Checking that RMSE is computed correctly

This test case checks whether the evaluation metric Root Mean Squared Error is calculated correctly i.e, using correct predicted and actual values arrays, in order to ensure accuracy is reliable.

Field	Description
Test ID	WB-002
Test Objective	Checking that RSME is calculated properly
Input	Prediction on test input values using model
Expected Output	An RMSE score between 0 and 1, not 1 as it would indicate and error
Actual Result	An RMSE score between 0 and 1
Status	Pass

Table 7.4: White Box Test Case 2

8. Results

The CO₂ Emission Prediction Tool based on Flask is an interactive, intelligent interface that allows users to evaluate the carbon emission yield of different vehicle configurations. It is intuitive, data-driven, and flexible, such that it can be used for both single-input prediction and comparative scenario analysis. Every feature is designed to enable users like vehicle researchers, green consumers, and policymakers to make well-informed decisions from emission figures. It consists of:

8.1 Vehicle Emission Prediction

- **What it does:** Users can input specific vehicle characteristics including engine size, number of cylinders, fuel type, and fuel efficiency (city, highway, or combined). The system will directly process the values through a trained stacking model consisting of SVR and Random Forest regressors as soon as inputs are entered.
- **What is displayed:** The estimated CO₂ emission (in grams/km) for the vehicle configuration is highlighted most visible in the middle of the interface, giving users unambiguous, direct feedback on how the vehicle configuration will affect the environment.
- **Why it matters:** This capability enables users to assess the effect of particular design or use attributes on emissions prior to vehicle production or purchase choices.

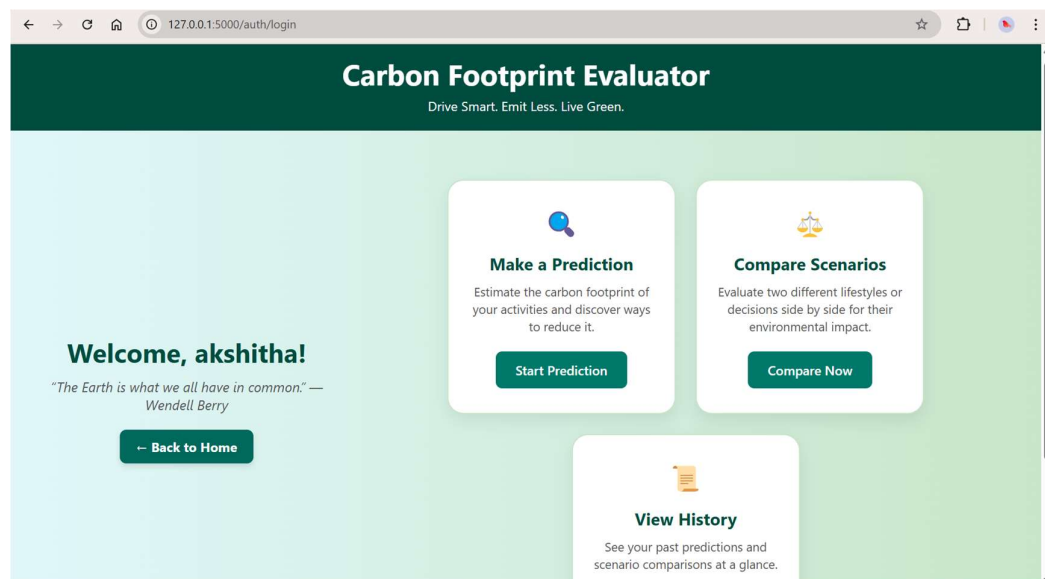


Fig 8.1: Home Screen

Enter Vehicle Features

Engine Size (L)

No. of Cylinders

Fuel Consumption City (L/100 km)

Fuel Consumption Hwy (L/100 km)

Fuel Consumption Combined (L/100 km)

Fuel Consumption Difference (L/100 km)

Is Hybrid?


Predict

Prediction: 198.71

Fig 8.2: Prediction Screen

8.2 Scenario Comparison

- **What it does:** This facility allows users to enter and compare two different car configurations alongside each other. They may, for instance, compare a diesel with a petrol vehicle, or examine the effect of engine size increases on emissions while all other parameters are held constant.
- **What is displayed:** Both pairs of predictions are placed next to one another along with a percentage difference in emissions. A side-by-side bar chart or stacked comparison visualization allows the user to immediately understand which configuration is greener.
- **Why it matters:** Scenario comparison aids data-driven decision making, enabling manufacturing or user entities to investigate trade-offs between performance and sustainability prior to settling on a particular vehicle configuration.



Scenario 1	Scenario 2
Engine Size (L): 2	Engine Size (L): 2.4
Cylinders: 4	Cylinders: 4
Fuel Consumption City (L/100 km): 9.9	Fuel Consumption City (L/100 km): 11.2
Fuel Consumption Highway (L/100 km): 6.7	Fuel Consumption Highway (L/100 km): 7.7
Fuel Consumption Combined (L/100 km): 8.5	Fuel Consumption Combined (L/100 km): 9.6
Fuel Consumption Difference: 3	Fuel Consumption Difference: 2
Is Hybrid: 0 (No)	Is Hybrid: 1 (Yes)

Fig 8.3 Compare Scenarios Form

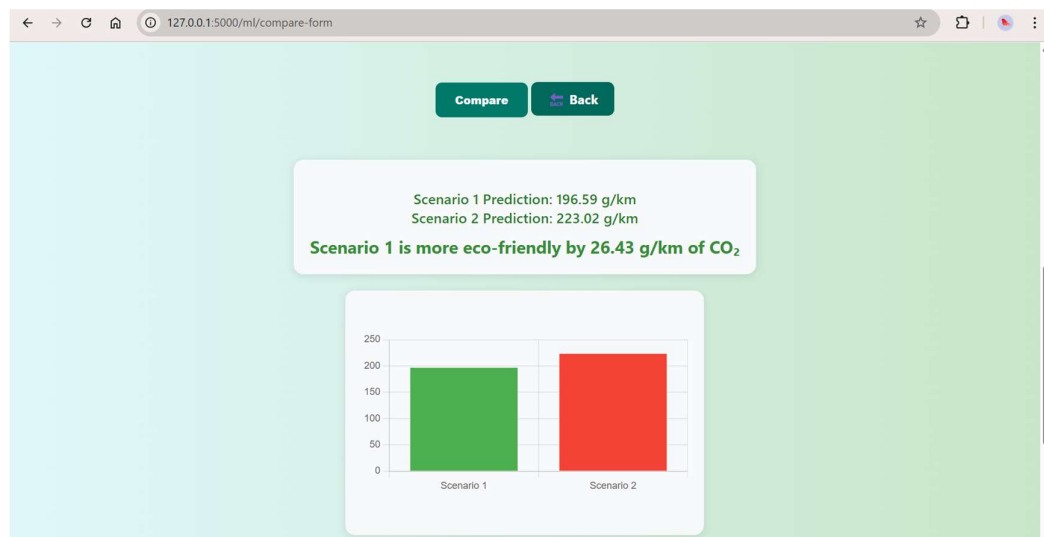
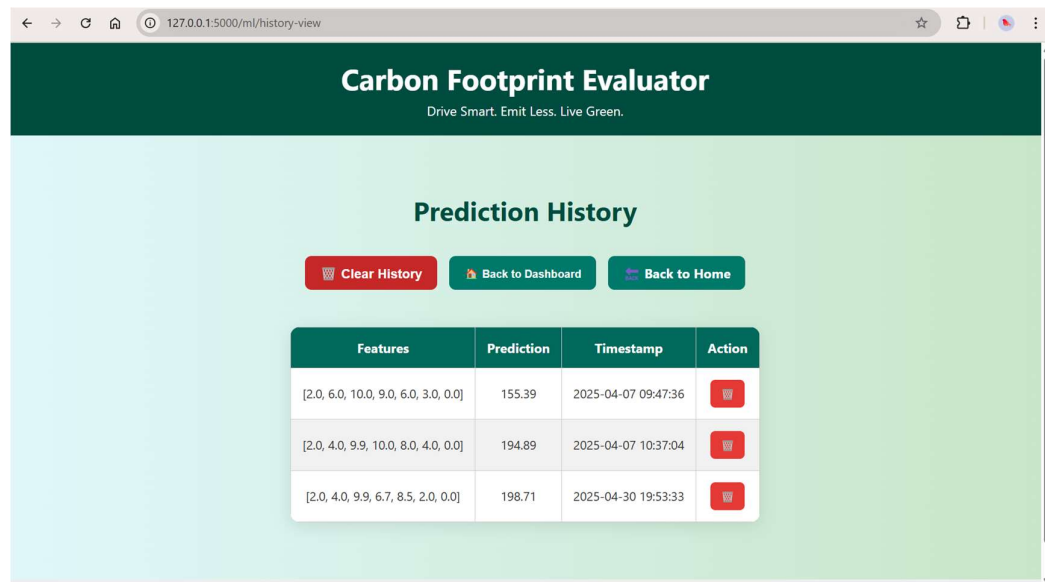


Fig 8.4: Compare Scenarios Prediction Screen

8.3 Input History and Result Logging

- **What it does:** Every prediction is recorded automatically with the input parameters and output value. The history is available during the session and may be downloaded optionally.

- **What is displayed:** A table containing each configuration the user submitted, along with corresponding CO₂ predictions, timestamps, and model confidence levels.
- **Why it matters:** Users can track how slight input changes affect emissions and retain a record of all comparisons or evaluations for further study or reporting.



Carbon Footprint Evaluator			
Drive Smart. Emit Less. Live Green.			
Prediction History			
Clear History Back to Dashboard Back to Home			
Features	Prediction	Timestamp	Action
[2.0, 6.0, 10.0, 9.0, 6.0, 3.0, 0.0]	155.39	2025-04-07 09:47:36	Delete
[2.0, 4.0, 9.9, 10.0, 8.0, 4.0, 0.0]	194.89	2025-04-07 10:37:04	Delete
[2.0, 4.0, 9.9, 6.7, 8.5, 2.0, 0.0]	198.71	2025-04-30 19:53:33	Delete

Fig 8.5: Prediction History

9. Conclusion and Further Work

The CO₂ Emission Prediction Tool developed is a user-friendly and accessible tool for vehicular carbon emission estimation through machine learning. Developed with a user-friendly design in Streamlit using regression models including SVR, Random Forest, and Stacking Regressor, the tool predicts CO₂ output accurately based on important vehicle parameters including engine size, fuel type, and mileage. Through both single prediction and comparison of scenarios, it facilitates data-driven decision-making for users looking to assess or minimize carbon emissions.

Key Highlights:

- **High Accuracy:** The combination of SVR, Random Forest, and Stacking Regressor ensures robust predictions with high R^2 scores and low error margins.
- **Scenario Comparison:** Side-by-side analysis of multiple configurations to help users explore the environmental trade-offs of design or purchase choices.
- **Graphical Insights:** Visual tools such as emission meters, and feature impact graphs enhance interpretability and transparency.
- **User-Friendly Interface:** A responsive, intuitive UI built with Flask, offering clean navigation to prediction, scenario comparison, and result history,

Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	R^2 Score
2.47	20.205	4.495	0.994

Table 9.1: Evaluation Metrics

Above Table presents the metrics evaluating the performance of the ML-based Vehicle Carbon Emissions Prediction. The MAE of 2.47 means that, on average, the model's predictions are off by only 2.47 kg of CO₂, which indicates strong predictive precision. The R^2 score (coefficient of determination) explains how well the model accounts for the variability in the data. A value of 0.994 means 99.4% of the variation in CO₂ emissions can be explained by your model — a sign of excellent overall accuracy. RMSE measures the average magnitude of prediction errors, with more weight given

to larger errors. A low RMSE of 4.495 indicates that most predictions are very close to actual values, and the model handles outliers well. MSE is the average of squared prediction errors and helps penalize larger deviations more than smaller ones, it supports the RMSE by reinforcing that large errors are rare.

Further Work:

Although today's system provides excellent accuracy and usability, there are a few avenues for future development to further improve its ability and influence:

- **Inclusion of Real-Time Vehicle Information:** Merging live data streams from IoT-enabled vehicles or emission testing equipment could make real-time predictions possible based on changing driving conditions.
- **Support for Wider Vehicle Types:** Covering electric, hybrid, and heavy-duty vehicles in the dataset would extend the tool's coverage to a wider range of autos.
- **Policy and Compliance Features:** Integrating regional emission standards (e.g., Euro 6, BS VI) would enable the user to assess if their vehicle is compliant with legal compliance standards.
- **Mobile and Cloud Deployment:** Creating a mobile version and hosting the tool on scalable cloud infrastructures (e.g., AWS, Azure) would increase its feasibility for usage in the field and adoption by the general public.

Through these improvements, the CO₂ Emission Prediction Tool can be developed into an integrated, real-world decision-support system for sustainable automotive use and design.

10. References

- [1] S. Martinez, “Vehicles and Emissions,” M.S. thesis, California State University, San Bernardino, CA, USA, 2022.
- [2] WorldAutoSteel, “Life Cycle Assessment: Good for the Planet, Good for the Auto Industry,” Apr. 2023.
- [3] G. Asaithambi, M. Treiber, and V. Kanagaraj, “Life Cycle Assessment of Conventional and Electric Vehicles,” in *International Climate Protection*, M. Palocz-Andresen, D. Szalay, A. Gosztom, L. Sípos, and T. Taligás, Eds. Cham: Springer, 2019.
- [4] L. Bushi, “EDAG Silverado Body Lightweighting Final LCA Report,” Aluminum Association, USA, Aug. 2018.
- [5] Green NCAP, “LCA: How Sustainable is Your Car?,” Nov. 16, 2022. [Online]. Available: <https://www.greenncap.com/press-releases/lca-how-sustainable-is-your-car/>
- [6] S. Lang, B. Engelmann, A. Schiffler, and J. Schmitt, “A simplified machine learning product carbon footprint evaluation tool,” *Cleaner Environmental Systems*, vol. 13, p. 100187, Jun. 2024.
- [7] R. Dhyan, H. K. Joy, R. Sridevi, E. A. J. A. Electa, and D. Vanusha, “Machine Learning and Deep Learning Analysis of Vehicle Carbon Footprint,” *International Journal of Environmental Impacts*, Jun. 2024.
- [8] Y. Jin, A. Sharifi, Z. Li, S. Chen, S. Zeng, and S. Zhao, “Carbon emission prediction models: A review,” Apr. 2024.
- [12] D. L. Marr, P. L. Nguyen, and R. P. Reddy, “Life cycle assessment of greenhouse gas emissions from automotive vehicles and fuel technologies,” *Environmental Progress & Sustainable Energy*, vol. 39, no. 1, pp. 1–12, Jan. 2020, doi: 10.1002/ep.13212.
- [13] P. Zhang, J. Ma, and B. Yang, “A review on vehicle emission prediction using machine learning techniques,” *IEEE Access*, vol. 8, pp. 181956–181964, 2020, doi: 10.1109/ACCESS.2020.3028302.