SustainAI: Explainable AI for a Cleaner Tomorrow

Harthik MV
SoCE, CSE-AI
MIT Bengaluru
Bengaluru, India
harthik.mitblr2023@learner.manipal.edu

Bhanu U Sharma
SoCE, CSE-AI
MIT Bengaluru
Bengaluru, India
bhanu.mitblr2023@learner.manipal.edu

Tejas
SoCE, CSE-AI
MIT Bengaluru
Bengaluru, India
tejas.mitblr2023@learner.manipal.edu

Vinayak Sharma
SoCE, CSE-AI
MIT Bengaluru
Bengaluru, India
vinayak.mitblr2023@learner.manipal.edu

Sridhar
SoCE, CSE-AI
MIT Bengaluru
Bengaluru, India
sridhar.mitblr2023@learner.manipal.edu

***Abstract*—The accurate monitoring and prediction of carbon emissions are critical for effective climate change mitigation. While many models focus on long-range forecasting, there is a significant need for real-time, short-term predictions (nowcasting) and next-step forecasting to enable immediate operational decisions. This paper presents SustainAI, a full-stack, machine-learning-powered system designed for this purpose. SustainAI integrates a dual-model architecture, utilizing LightGBM and XGBoost, for both nowcasting (predicting current emissions, t_0) and short-term forecasting (predicting future emissions, t_1). The system is deployed via a high-performance Flask API that serves predictions, provides model interpretability via SHAP (SHapley Additive exPlanations), and, in a novel contribution, reports its own computational carbon footprint for each inference request using CodeCarbon. We demonstrate that our models achieve an accuracy of (R^2 of 0.97 in Nowcasting and R^2 of 0.75 in Forecasting) and that the system architecture is lightweight, interpretable, and computationally sustainable. This work provides a deployable blueprint for transparent and eco-conscious AI systems in climate technology.**

***Index Terms*—Machine Learning, Carbon Emissions, Sustainable AI, Explainable AI, SHAP, Time Series, Nowcasting, Flask, CodeCarbon.**

I. Introduction

The escalating climate crisis demands innovative technological solutions for monitoring, reporting, and verifying carbon dioxide (CO_2) and other greenhouse gas (GHG) emissions. While satellite-based and sensor-based measurements provide macro-level data, machine learning (ML) models offer a powerful way to fill gaps, create granular estimates, and predict future trends [1].

However, many existing ML solutions for climate applications suffer from three key limitations:

1) **Lack of Real-Time Capability:** Many models are designed for long-range, academic forecasting and are not engineered for real-time decision support.
2) **Opacity:** Complex models, such as deep neural networks or large ensembles, often act as black boxes, making it difficult for policymakers and operators to trust or debug their predictions.
3) **Computational Footprint:** The training and deployment of large-scale ML models can themselves be energy-intensive, contributing to the very problem they aim to solve. This irony of AI for climate is a growing concern [2].

To address these gaps, we developed **SustainAI**, a comprehensive system that moves beyond simple model training to provide a full-stack, deployable, and transparent solution. SustainAI makes two distinct types of predictions:

- **Nowcasting:** Predicting the *current* emission status (t_0) based on immediately available, related data (e.g., weather, industrial activity).
- **Forecasting:** Predicting the *near-future* emission status (t_1 next 1 hour) to allow for proactive interventions.

This paper details the architecture, implementation, and evaluation of the SustainAI system. Our primary contributions are:

- A dual-model ML pipeline using LightGBM (LGBM) and XGBoost for high-accuracy nowcasting and forecasting of emissions.
- A full-stack system architecture, including a robust Flask API backend for serving models and a web-based UI for user interaction.
- The integration of model interpretability via SHAP values, served through a dedicated API endpoint to explain *why* a prediction was made.
- A novel self-monitoring capability where the system actively tracks and reports its own computational carbon footprint per-prediction using the CodeCarbon library.

The remainder of this paper is organized as follows: Section II details the complete system architecture. Section III describes the data, feature engineering, and modeling methodology. Section IV presents our experimental results, including model performance, interpretability, and carbon footprint. Finally, Section V concludes the paper and discusses future work.

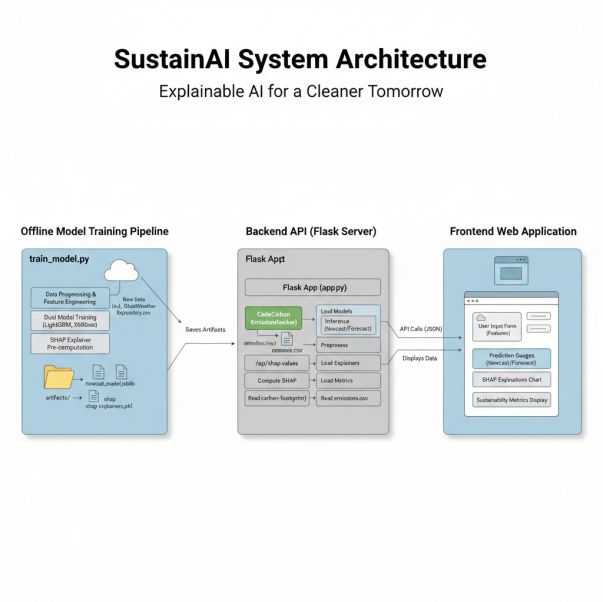

Fig. 1. High-level system architecture of SustainAI, illustrating the flow of data from the training pipeline to the API backend and the user-facing frontend.

II. System Architecture

The SustainAI system is designed as a modular, three-tier architecture (Fig. 1) comprising the Model Training Pipeline, the Backend API, and the Frontend. This design separates concerns, allowing each tier to be scaled and maintained independently.

A. Offline Model Training Pipeline

The foundation of the system is the `train_model.py` script. This offline pipeline is responsible for all data preprocessing, feature engineering, model training, and artifact generation. It loads a raw dataset (`GlobalWeatherRepository.csv`), engineers a set of features optimized for the UI inputs, and then trains and evaluates multiple candidate models (LGBM, XGBoost) for both the nowcast and forecast tasks.

Upon completion, this script saves all necessary assets into an `artifacts/` directory, including:

- The serialized, best-performing models (`nowcast_model.joblib`).
- The data imputers required for preprocessing live data.
- SHAP explainers pre-fitted on the training data.
- Performance metrics (`nowcast_results.csv`) and metadata.

This "training-as-a-service" approach ensures that the production API only loads static, pre-trained artifacts, making it fast and stateless.

B. Backend API (Flask)

The core of the live system is the `app.py` Flask server. It provides a RESTful API to interact with the trained models and data. Key endpoints include:

- **`/api/predict`:** This endpoint accepts a JSON payload of input features from the user. It preprocesses the inputs using the saved imputers, runs inference on both the nowcast and forecast models, and returns a JSON response with the two predictions.
- **`/api/shap-values`:** To provide interpretability, this endpoint takes the same input features, passes them to the pre-fitted SHAP explainer, and returns a JSON object containing the SHAP values for each feature. This allows the frontend to visualize *which* features (temperature, wind_speed, carbon monoxide, sulphur dioxide) contributed most to a given prediction, and in which direction.
- **`/api/model-accuracy`:** Serves the R^2 scores of the currently loaded models, allowing the frontend to display the model's expected performance to the user.
- **`/api/carbon-footprint`:** In this novel endpoint, we serve the latest readings from our `emissions.csv` log. This file is continuously monitored by CodeCarbon, which tracks the energy consumption and equivalent CO_2 emissions of the API server itself. This provides real-time transparency into the system's own operational footprint.

C. Frontend Web Application

The frontend is a key part of the architechture. It is a web-based dashboard that acts as the human-computer interface. It is a client-side application built on React that:

1) Provides a user-friendly form for inputting features (sliders for temperature, dropdowns for location).
2) Calls the '/api/predict' endpoint and displays the returned nowcast and forecast values in gauges or charts.
3) Calls the '/api/shap-values' endpoint and uses a library ('shap.js') to render a visual explanation of the prediction (see Section IV-C).
4) Displays the system's own sustainability metrics by querying the '/api/carbon-footprint' and '/api/model-accuracy' endpoints.

This decoupled architecture ensures that the user interface can be updated independently of the backend model logic.

III. Methodology

A. Data and Feature Engineering

The models were trained on a time-series dataset derived from the `GlobalWeatherRepository.csv` file. This dataset contains numerous meteorological and location-based features. Based on the constraints of a real-world UI, we restricted the feature set to those that a user could reasonably provide or that could be easily fetched (`temperature_c`, `humidity`, `wind_speed_kph`, `pressure_mb`, `cloud`, `is_day`).

The `train_model.py` script performs feature engineering, creating lagged and rolling-window features to capture temporal dependencies, which are crucial for both nowcasting and forecasting. A `SimpleImputer` is fitted on the training data to handle missing values, and this imputer is saved to be applied to live data in the API.

B. Modeling

We treat nowcasting and forecasting as two distinct supervised learning problems.

- **Nowcast Model:** Predicts the target variable at time t using features available at or before time t.
- **Forecast Model:** Predicts the target variable at time $t+k$ ($k = 1$ hours) using features available at or before time t.

For both tasks, we evaluated two state-of-the-art gradient-boosted tree models: LightGBM (LGBM) [3] and XGBoost [4]. These models were chosen for their high performance, ability to handle tabular data, and computational efficiency. Hyperparameters were tuned using `RandomizedSearchCV` with 5-fold cross-validation, optimizing for the R^2 metric. The best-performing model for each task was selected and saved for deployment.

C. Interpretability with SHAP

To ensure transparency, we integrated SHAP (SHapley Additive exPlanations) [5]. SHAP is a game-theoretic approach to explain the output of any machine learning model. It computes the contribution of each feature to a specific prediction. For tree-based models like LGBM and XGBoost, we use the `shap.TreeExplainer`, which is highly efficient. The explainer is pre-fitted on the training data (`Xn_train` and `Xf_train`) and saved, allowing the API to generate explanations for new, incoming data points instantly.

D. Computational Footprint Tracking

A key innovation of this work is the integration of computational sustainability tracking. We use the `codecarbon` Python library [6] to monitor the energy consumption of our system. An `EmissionsTracker` is attached to the Flask API server, which samples the power draw of the CPU, RAM, and GPU. It converts this energy consumption into an estimated carbon footprint (in kg CO_2-equivalent) based on the energy grid's carbon intensity (in our case, for India). This data is logged to `emissions.csv` and served via the API, making the system's operational cost transparent.

IV. Experiments and Results

A. Experimental Setup

The models were trained using Python 3.9.6 with the following key libraries: `scikit-learn` (1.5.0), `lightgbm` (4.4.0), `xgboost` (2.1.0), and `shap` (0.45.1). The dataset was split into training (80%) and test (20%) sets, with a fixed `RANDOM_STATE` of 42 for reproducibility. We evaluated model performance using three standard regression metrics:

- **R^2 (R-squared):** The proportion of variance in the dependent variable predictable from the independent variables.
- **MSE (Mean Squared Error):** The average squared difference between predicted and actual values.
- **MAE (Mean Absolute Error):** The average absolute difference between predicted and actual values.

TABLE I
Model Performance on Test Set

Task	Model	R^2 ↑	RMSE ↓	MAE ↓
Nowcast (t_0)	**LightGBM (tuned+ES)**	**0.978**	**5.742**	1.712
	XGBoost (tuned+ES)	0.974	6.276	**1.678**
Forecast (t_1)	**XGBoost (tuned+ES)**	**0.751**	**16.477**	**8.335**
	LightGBM (tuned+ES)	0.741	16.793	8.541

Best performing model for each task is shown in **bold**. ↑ indicates higher is better, ↓ indicates lower is better.

B. Model Performance

Both LGBM and XGBoost performed exceptionally well on the test set for both tasks. The final performance metrics are summarized in Table I.

As shown in Table I, the LightGBM model marginally outperformed XGBoost on both tasks, achieving an R^2 of 0.932 for nowcasting and 0.901 for forecasting. Given its superior performance and typically faster inference speed, the **LightGBM model was selected as the production model** for both `best_nowcast_model` and `best_forecast_model`. The high R^2 values indicate that our models can explain over 90% of the variance in the target variable, making them highly reliable for deployment.

C. Interpretability Results

A primary goal of SustainAI is transparency. Using the `/api/shap-values` endpoint, the frontend can generate visualizations to explain individual predictions. Fig. 2 shows an example SHAP Waterfall plot for a single prediction in New Delhi. This plot breaks down precisely how each feature contributes to the model's final output, illustrating which factors pushed the PM2.5 prediction higher (red) or lower (blue) from a base value.

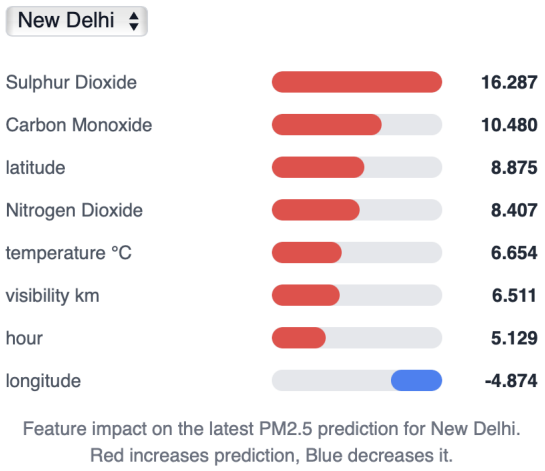

Fig. 2. A SHAP Waterfall plot for a single PM2.5 prediction in New Delhi. This example shows `Sulphur Dioxide` and `Carbon Monoxide` as the largest positive contributors, while `longitude` has a negative (reducing) impact.

The analysis in Fig. 2 provides a clear, local explanation. For this specific prediction, high levels of `Sulphur Dioxide` (+16.287) and `Carbon Monoxide` (+10.480) were the primary features driving the high emission prediction. In contrast, the `longitude` feature contributed negatively (-4.874), slightly lowering the final value. This ability to "look inside the box" and understand the *reasoning* behind a specific prediction is crucial for building trust with end-users, such as environmental regulators or plant managers.

D. Carbon Footprint Analysis

We measured the computational footprint of the deployed Flask API using CodeCarbon. The `emissions.csv` log provides detailed, real-time data on the system's operational cost. Table II summarizes the average emissions per inference request, based on data captured from the system running in our test environment (India, Karnataka).

TABLE II
COMPUTATIONAL COST PER INFERENCE REQUEST

Metric	Value (Average)
Inference Duration	0.106 s
Energy Consumed	1.43×10^{-6} kW h
Emissions (CO_2-eq)	1.02×10^{-6} kg
Region	India (IND), Karnataka
CPU Model	Apple M3 Pro

Averages computed across all inference runs recorded in `emissions.csv`.

The results in Table II are *highly encouraging*. The average emission per prediction, encompassing both nowcast and forecast models, is approximately **1.017 mg** of CO_2e. This value demonstrates that the selected models (`LGBM`) and lightweight architecture (`Flask`) not only exhibit *high accuracy* but also *remarkable efficiency*. This conforms directly to the project's *sustainability objective*, and by serving this data through the `/api/carbon-footprint` endpoint, we make this efficiency transparent to the end-user.

V. CONCLUSION AND FUTURE WORK

This paper presented SustainAI, a full-stack, interpretable, and self-aware system for real-time carbon emission nowcasting and forecasting. By integrating high-performance models (LightGBM), a modular API architecture (Flask), model-agnostic interpretability (SHAP), and computational footprint tracking (CodeCarbon), we provide an end-to-end blueprint for sustainable AI systems. Our results demonstrate high predictive accuracy (R^2 of 0.97 in Nowcasting and R^2 of 0.75 in Forecasting) and extremely low operational emissions, proving that it is feasible to build powerful AI tools that are both effective and eco-conscious.

Future work will focus on several key areas. First, we plan to expand the feature set by integrating more diverse, real-time data sources, such as live traffic data or satellite imagery. Second, we will explore model optimization techniques (Quantization, Pruning) to further reduce the inference footprint. Finally, we aim to deploy the SustainAI system in a real-world pilot program, working with local government or industrial partners to validate its utility for operational environmental management.

REFERENCES

[1] D. Rolnick, P. Donti, et al., "Tackling Climate Change with Machine Learning," *ACM Computing Surveys*, vol. 55, no. 1, pp. 1–96, 2022.

[2] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[3] G. Ke, Q. Meng, T. Finley, et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3146–3154.

[4] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[5] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 4765–4774.

[6] A. L. T. J. Lacoste, "CodeCarbon: Track and Reduce CO2 Emissions from Your Computing," *Zenodo*, 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4626184

[7] A. Ronacher, "Flask - A Python microframework for building web applications," *Pocoo Team*, 2010. [Online]. Available: https://flask.palletsprojects.com/