Green AI Comparometer

by

Utkarsh Misra (2448372)
Puspita Biswas (2448348)
Priangshu Paul (2448384)

Under the guidance of
Dr. Jayapriya J

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE | DELHI NCR | PUNE

A Project report submitted in partial fulfillment of the requirements for the award of the degree of Master of Science (Data Science) of CHRIST (Deemed to be University)

September – 2025

CERTIFICATE

This is to certify that the report titled ***Green AI Comparometer*** *is a bonafide record of work done by* ***Puspita Biswas*** *of CHRIST (Deemed to be University), Bengaluru, in partial fulfilment of the requirements of IVth Trimester MSc (Data Science) during the academic year 2025-26.*

Head of the Department **Project Guide**

Valued-by

1. Name :

Register Number :

2. Date of Exam :

CERTIFICATE

This is to certify that the report titled ***Green AI Comparometer*** *is a bonafide record of work done by* ***Utkarsh Misra*** *of CHRIST (Deemed to be University), Bengaluru, in partial fulfilment of the requirements of IVth Trimester MSc (Data Science) during the academic year 2025-26.*

Head of the Department **Project Guide**

Valued-by

1. Name :

Register Number :

2. Date of Exam :

CERTIFICATE

This is to certify that the report titled ***Green AI Comparometer*** *is a bonafide record of work done by* ***Priangshu Paul*** *of CHRIST (Deemed to be University), Bengaluru, in partial fulfilment of the requirements of IVth Trimester MSc (Data Science) during the academic year 2025-26.*

Head of the Department **Project Guide**

Valued-by

1. Name :

Register Number :

2. Date of Exam :

ACKNOWLEDGEMENT

We would like to express our gratitude to **Dr Jayapriya J,** our project guide for her invaluable guidance, continuous support and encouragement throughout the development of this project.

Her expertise and insights have been instrumental in shaping this work. We extend our heartfelt thanks to the faculty members of the **Department of Statistics and Data Science CHRIST (Deemed to be University)** for providing us with the necessary knowledge, foundation and resources to undertake this project.

We also acknowledge the support of our families and friends who have been a constant source of motivation throughout this journey.

ABSTRACT

This report presents a comprehensive solution for measuring and comparing the carbon footprint of machine learning and deep learning code, a critical step toward fostering sustainability in computational practices. The project, titled the **GreenAI Comparometer**, is a user-friendly Streamlit web application that simplifies the environmental impact assessment of computational workloads. The application allows users to seamlessly upload Python scripts or Jupyter notebooks alongside any associated datasets. At its core, the system integrates and orchestrates three distinct and widely-used open-source carbon tracking libraries, **eco2ai, CarbonTracker and CodeCarbon**. This integration provides a standardized, objective summary of key metrics, including **CO_2 emissions (kg)**, **energy consumption (kWh)**, and **execution duration (s)**. The system was specifically engineered to overcome common implementation challenges associated with existing trackers, such as a reliance on static file paths and complex setup procedures. By dynamically managing file uploads in a secure, temporary environment and standardizing output for both libraries, the GreenAI Comparometer offers a reliable and accessible tool. It empowers developers and researchers to gain clear, data-driven insights into the environmental cost of their code, enabling them to make informed decisions and optimize their workflows for a more sustainable future for artificial intelligence.

Table of Contents

List of Figures

CHAPTER 1: INTRODUCTION

1.1 Problem Description

While the proliferation of individual tools for tracking AI's carbon footprint—such as CodeCarbon, eco2AI, and CarbonTracker—is a positive development, it has inadvertently led to a fragmented and often confusing landscape for developers and researchers. As the authors investigated the current state of Green AI tooling, a significant gap became apparent: **there is currently no single, integrated platform where a user can upload their ML/DL code and directly compare the carbon emission results generated by these different tracking libraries simultaneously.**

This absence of a unified comparative framework presents several critical challenges. Firstly, it places the burden of choice and implementation on the user, who may not have the time or expertise to evaluate which tool is best suited for their specific hardware, cloud environment, or geographical location. Each library employs subtle variations in its measurement methodology, data sources for grid intensity, and calculation algorithms, which can lead to notable discrepancies in their final emission reports. A researcher relying on a single tool receives only one perspective, with no straightforward way to validate or contextualize that result.

Secondly, this fragmentation acts as a barrier to the widespread adoption of sustainable practices. To perform a comparative analysis manually, a developer would need to install, configure, and integrate each of these libraries into their workflow—a cumbersome process that detracts from their primary focus on model development. Finally, the lack of a standardized benchmarking environment makes it difficult for the AI community to assess the reliability and consistency of the measurement tools themselves. Without a platform that can execute the same code under identical conditions and report the outputs of multiple trackers, it is challenging to move towards a more standardized and scientifically rigorous approach to carbon accounting in AI.

1.2 Problem Statement

To address this clearly defined gap, the primary objective of this project is to **design and implement a unified web-based tool, the "Green AI Comparometer," which facilitates the seamless measurement and direct comparison of carbon emissions from machine learning and deep learning scripts using multiple, distinct tracking libraries.**

The project aims to create a centralized, user-friendly platform that abstracts away the underlying complexities of individual tool setup and execution. By allowing users to simply upload their code, the Green AI Comparometer will provide a comprehensive and multi-faceted view of its environmental impact. The core mission is to empower users with comparative data, enabling them to make more informed decisions about their computational practices.

To achieve this overarching goal, the project is guided by the following specific objectives:

1. **To Develop a Centralized and Accessible Web Platform:** The primary deliverable will be a web-based application that offers an intuitive user interface for uploading Python or Jupyter Notebook files, managing experiments, and viewing results, thereby eliminating the need for complex local installations.
2. **To Integrate Multiple Carbon Tracking Libraries:** The platform's backend will be engineered to run user-submitted code in a controlled environment while instrumenting it with CodeCarbon, eco2AI, and CarbonTracker simultaneously, ensuring that all measurements are taken under identical hardware and software conditions for a fair comparison.
3. **To Provide Comprehensive Comparative Visualization:** The platform will not merely list the numerical outputs. A key objective is to present the results in a consolidated dashboard featuring comparative charts, graphs, and tables. This will allow users to easily visualize the differences and similarities in the emissions reported by each library, alongside key technical metrics like runtime and energy consumption.
4. **To Foster Transparency and Methodological Insight:** By exposing the potential variations between different measurement tools, the project aims to contribute to greater transparency in the field of computational carbon accounting and encourage a deeper understanding of how these critical environmental metrics are derived.

1.3 Existing Systems and their Limitations

1.3.1 Summary of Existing Systems

Lacoste et al. [1] introduce the Machine Learning Emissions Calculator, a tool designed to help researchers approximate the carbon footprint of training their models. The authors highlight that emissions are heavily influenced by the server's location and energy grid, hardware model, and training duration, and they offer concrete strategies to mitigate this environmental impact. Gu et al. [12] review recent progress in using machine learning to accelerate the discovery of materials essential for renewable energy technologies. Their work summarizes ML applications in key areas like catalysis, batteries, and solar cells, while also analyzing successful real-world discoveries and identifying critical gaps to guide future research. Mittal et al. [22] provides a survey of architecture and system-level techniques for optimizing deep learning applications on GPUs. The review comprehensively covers methods for both inference and training across single-GPU and distributed multi-GPU systems, highlighting the key attributes and differences between various approaches. Esser et al. [24] introduce Learned Step Size Quantization, a novel training method for low-precision deep networks that achieves state-of-the-art accuracy. Their approach treats the quantizer step size as a learnable parameter for each layer, enabling 3-bit models to match the performance of their full-precision counterparts on ImageNet.

Henderson et al. [2] propose a framework to simplify the tracking and reporting of energy consumption and carbon emissions for machine learning research. They demonstrate its utility by creating an energy-efficiency leaderboard for reinforcement learning, aiming to promote sustainable practices and incentivize the development of more efficient algorithms. To address the rising energy consumption of deep learning, Anthony et al. [3] present Carbontracker, a tool designed to track and predict the energy and carbon footprint of training DL models. They propose that environmental impact should be reported as a standard metric alongside model performance, encouraging the ML community to pursue more energy-efficient research and responsible computing practices.

Ligozat and Luccioni [14] offer a guide to help the machine learning community understand and address its environmental footprint. They advocate for a two-step approach where practitioners first quantify the carbon impact of their work and then take active steps to mitigate those emissions to combat climate change. Gholami et al. [19] provide a comprehensive survey on the quantization of deep neural networks, a technique used to convert floating-point values into low-precision integers to reduce memory footprint and latency. Their paper organizes and reviews current approaches, discussing the advantages and disadvantages of various methods to present a useful snapshot of this active research area.

To improve the accuracy of global CO_2 emission forecasts, Meng and Noman [4] developed and compared four SARIMAX machine learning models that explicitly account for the impact of the COVID-19 pandemic. They found that the model trained on the post-pandemic period yielded the most accurate predictions, offering a more effective tool to inform future climate policies and emission reduction strategies. In their perspective, Yao et al. [5] review the application of machine learning (ML) to accelerate advances in renewable energy technologies. They evaluate recent ML-driven progress in energy harvesting, storage, and management, while also outlining current challenges and proposing future research directions for the field. Budennyy et al. [6] introduce eco2AI, an open-source package designed to help researchers track the energy consumption and CO_2 emissions of their AI models. By providing accurate and region-specific emissions data, the tool aims to encourage the development of more computationally efficient architectures and promote the concept of Green AI. Wu et al. [7] provide a holistic, end-to-end analysis of the environmental impact of AI, characterizing its carbon footprint across data, algorithms, and the entire life cycle of system hardware (including manufacturing). Based on industry experience, they identify key challenges and propose hardware-software co-design and optimization as crucial strategies for developing environmentally responsible AI.

Jabed et al. [9] investigated the integration of an AI-powered recommendation engine to optimize Salesforce development and configuration. Their analysis showed that the AI tool significantly reduced development time and defect density while simultaneously improving customization accuracy and overall user satisfaction, thereby demonstrating AI's value in enhancing software customization efficiency. Ali et al. [10] introduce a Green AI framework that

utilizes machine learning to accelerate the transition from a traditional linear economy to a circular one. Their model demonstrates how AI can act as the optimization backbone for efficient, closed-loop supply chains, enabling the large-scale reuse, remanufacturing, and recycling of products and materials. Benti et al. [11] provide a comprehensive review of machine learning (ML) and deep learning (DL) techniques applied to forecasting renewable energy generation. The paper surveys various models, discussing their strengths and limitations, and highlights key challenges such as data variability and model interpretability to guide future research for efficient grid integration. Gaur et al. [15] analyze the dual role of AI in climate change, using a "system of systems" framework to explore its complex relationship with carbon emissions. By empirically calculating and comparing the carbon footprint of six different machine learning models, their study underscores the environmental impact of AI and advocates for the development of more efficient and sustainable models. Tiutiulnikov et al. [16] introduce eco4cast, an open-source package designed to reduce the carbon footprint of machine learning through predictive scheduling. The tool uses a temporal convolutional neural network to accurately forecast periods of low carbon intensity in the electricity grid, enabling it to automatically run cloud computing tasks only during these optimal, low-emission times. Kim et al. [20] propose a green-quantized federated learning (FL) framework to reduce the high energy cost of training on resource-constrained devices. By using low-precision quantized neural networks for both local training and data transmission, their multi-objective optimization approach is shown to cut energy consumption by up to 70% compared to full-precision FL, without harming the convergence rate. Castellanos-Nieves and García-Forte [27] address the high resource consumption of AutoML by proposing a method to make it more sustainable. In a proof-of-concept study, they integrated energy efficiency metrics directly into hyperparameter search strategies, demonstrating that optimizing for energy can significantly improve AutoML's environmental footprint in line with the Green AI paradigm. Rokh et al. [31] present a comprehensive survey of Deep Neural Network (DNN) quantization, a technique for compressing models by reducing the precision of their parameters.[1] Focusing on image classification, the paper reviews various methods, covers training strategies for quantized networks, and analyzes the performance of state-of-the-art approaches on benchmarks like CIFAR-10 and ImageNet to guide future research.

Wei et al. [17] address the conflict between the high computational cost of large AI models and their deployment on resource-constrained devices. They provide an analysis of neural network quantization, reviewing various methods for converting models to low-bit integers, their resulting accuracy, and future challenges in the field. Alizadeh et al. [21] present the first empirical study on the energy efficiency of deep learning runtime infrastructures, comparing frameworks like PyTorch, TensorFlow, MXNet, and the ONNX Runtime. Their findings reveal that performance is highly dependent on the specific model and configuration, but converting models to ONNX and using optimized execution providers like TensorRT often yields significant improvements in both inference speed and energy efficiency. In the context of green deep learning, Tmamna et al. [26] provide a systematic review of recent breakthroughs in

convolutional neural network (CNN) pruning. The paper summarizes current methods, highlights existing challenges and the need for better evaluation metrics, and points to future research directions for creating more energy-efficient models. To broaden the discourse on Green AI, Clemm et al. [29] investigate the environmental assessment and ecodesign of AI systems. They propose a life-cycle-based framework that considers four key elements—model, data, server, and cloud—and introduce the concept of "AI4greenAI" for leveraging AI to mitigate its own environmental impact.

Pathania et al. [8] present a state-of-the-art review of methods and tools for measuring the energy and carbon footprint of software and AI. They introduce a new taxonomy to categorize existing work into monitoring, estimation, or black-box approaches and provide a comparative analysis of tools based on their measurement granularity, highlighting current challenges in the field. Al Nuaimi et al. [13] conduct a systematic literature review on the application of machine learning for estimating carbon emissions, with a distinct focus on the practical, real-world performance of various models. The study evaluates algorithms across sectors like transportation and energy, identifies key research gaps, and provides a comprehensive guidance framework to steer future applications. Reguero et al. [18] developed an autoregressive prediction model to determine the optimal time to apply energy-saving techniques, like quantization and layer freezing, during neural network training. Their approach finds an optimal training path that can reduce energy consumption by 56.5% while simultaneously increasing validation accuracy by 2.38% by avoiding overfitting. Hasan et al. [23] conduct a systematic literature review of over 200 publications (2014–2024) on the environmental impact of machine learning. Their work synthesizes key trends in quantifying and mitigating ML's carbon footprint, identifies challenges such as the lack of standardized metrics, and culminates in a novel framework and research roadmap for sustainable ML practices. Paula et al. [25] investigate the effectiveness of model compression techniques in reducing the energy consumption of various transformer models.[1] Their empirical study shows that methods like pruning and knowledge distillation can cut energy usage by up to 32% while maintaining high accuracy, though they also highlight that certain techniques like quantization can severely degrade performance on already-compact models. Jegadeeswari and Rathipriya [28] propose a Sustainable Hyperparameter Optimization (SHPO) framework that addresses the high carbon footprint of model tuning. Their approach uses a multi-criteria decision-making method (TOPSIS) to simultaneously optimize for both predictive accuracy and low carbon emissions, finding that the Optuna optimization tool consistently provides the best trade-off. This study [30] evaluates state-of-the-art AI models for solar power forecasting using data from two of the world’s largest solar parks. The authors find that a hybrid Convolutional Neural Network–Long Short-Term Memory (CNN-LSTM) model achieves the highest prediction accuracy while maintaining low computational CO_2 emissions, demonstrating its effectiveness for sustainable energy management in smart cities.

Green Algorithms is one of the existing systems in the area; it is a framework created by Lannelongue et al. in 2020 to approximate the carbon footprint of a computational job. Green

Algorithms is mostly provided as an online calculator in which researchers provide information regarding their computation, including what type of hardware they are running on, how many cores, how long it takes, how much memory they use, and efficiency in the datacenter. According to these parameters, the system approximates the amount of energy used in kilowatt-hours and the equivalent carbon dioxide emissions. One of the key advantages of Green Algorithms is its transparent and peer-reviewed approach and its ability to make geographical comparisons, because not all regions have the same energy mixes and thus have equal emissions. It can be used extensively in fields that are not machine learning, and includes life sciences and other fields of computational research. But it is manual based, which limits its accuracy and cannot be combined with code execution in real time, making it more of a planning and reporting tool than a monitoring solution.

In 2019, Lacoste et al. also published another system, the ML CO 2 Impact Calculator, that is specific to machine learning workflows and is part of the MLCO 2 project. Similar to Green Algorithms, it is an online application that requires the user to type the following information: the type of hardware, the number of processors, the time of training, and the location of the cloud provider. It then calculates the energy used and the resultant carbon emissions, but it goes one step further by indicating the findings as relatable comparisons like the number of flights or kilometers covered by car. This renders the tool especially suitable to bring awareness to machine learning practitioners and the wider research fraternity. It is strong in the sense that it is simple and easily communicates the effects of the environment. Simultaneously, it has the same weakness as Green Algorithms, as it is also based completely on the parameters provided by the user and does not have any automated or real-time monitoring.

These two systems combined demonstrate increasing concern with sustainability in computational research. Green Algorithms is a framework applicable generically across fields, whereas the ML CO 2 Impact Calculator is a limited analytical tool with respect to machine learning applications. Both have significant roles in quantification and reporting of environmental impact, but are limited to manual data entry and non-direct integration with execution of code. It is this disconnect that drives the creation of unified tools like the GreenAI Comparator that uses automation alongside comparative analysis of various tools currently in existence.

1.3.2 Project Scope

This project's scope is centered on the design, development, and implementation of the Green AI Comparometer, a platform with a well-defined set of features aimed at promoting environmentally sustainable practices in machine learning. The specific functionalities to be delivered are outlined below.

- **Multi-Library Emission Tracking**: The core technical scope involves the integration of three distinct carbon emission tracking libraries: **eco2AI**, **CodeCarbon**, and **CarbonTracker**. The platform's backend will be engineered to execute user-submitted code within a controlled and instrumented environment. This system will simultaneously capture and record key environmental metrics from each library, including total energy consumed (in kWh), estimated carbon dioxide equivalents (CO2eq), and relevant hardware specifications.
- **Comparative Analysis Dashboard**: A primary objective is to provide users with a clear and intuitive way to compare the results from the different tracking tools. To this end, the project scope includes the creation of a user-facing dashboard. This interface will present the outputs from all three libraries in a side-by-side format, using tables and visualizations. This feature is crucial as it allows users to directly assess the degree of variance or consensus between the tools, providing a more robust and holistic understanding of their code's carbon footprint.

1.3.3 Project Limitations and Boundaries

To ensure the project remains focused and achievable within the available timeframe and resources, it is essential to define its boundaries. The following aspects are explicitly considered out of scope.

- **Language and Framework Dependency**: The platform's functionality is exclusively limited to **Python-based machine learning and deep learning codes**. While other languages like R or C++ are used in the field, Python is the predominant standard. Supporting a multi-language environment would introduce significant architectural complexity, and therefore, the project will focus on serving the largest segment of the target user base.
- **Exclusion of Large-Scale Models and Advanced Optimizations**: Due to significant **computational and resource limitations**, the project will not involve the testing of massive, cloud-scale models (e.g., large language models with billions of parameters). Training or fine-tuning such models requires access to distributed computing clusters and substantial financial investment, which are beyond the project's capacity. Similarly, while numerous model optimization techniques exist, this work will focus solely on **quantization** as a representative example. Other methods, such as **pruning**, knowledge distillation, or neural architecture search, will not be implemented or tested.

1.4 Objectives

The primary objectives of the project are as follows:

1. **To develop an integrated platform** that consolidates the functionality of independent carbon tracking libraries—eco2AI, CodeCarbon, and CarbonTracker—into a single, cohesive tool for analyzing ML/DL scripts.
2. **To enable direct, side-by-side comparison** of carbon emission (CO2eq) reports from the integrated libraries, allowing users to assess the variance and consistency of different measurement methodologies.
3. **To address the gap in ML/DL evaluation** by creating a framework that presents sustainability metrics alongside traditional performance indicators, encouraging a more holistic assessment that balances model accuracy with environmental impact.

CHAPTER 2: SYSTEM ANALYSIS

2.1 Functional Specifications

This section provides a detailed breakdown of the functional requirements for the Green AI Comparometer. These specifications define the system's expected behavior, features, and operations from the perspective of the end-user and system administrators. Each specification is assigned a unique identifier (e.g., FR-1.1) for clear tracking and reference.

2.1.1 User Account and Session Management

This module defines the functionalities related to user authentication, authorization, and personal data management.

- **FR-1.1 User Registration:**
  - The system shall provide a secure registration page where a new user can create an account.
  - The registration form shall require a unique username, a valid email address, and a password.
  - Password entry shall be masked for security. The system shall enforce minimum password complexity rules (e.g., minimum length, use of numbers and special characters).
- **FR-1.2 User Login:**
  - The system shall provide a secure login page for registered users.
  - The system shall authenticate users based on their username/email and password.
  - Upon successful authentication, the system shall create a user session and redirect the user to the main tool dashboard.
  - The system shall provide a clear error message for failed login attempts (e.g., "Invalid credentials").
- **FR-1.3 Session Management:**
  - The system shall maintain the user's session as they navigate between different pages of the application.
  - The system shall provide a "Logout" button that, when clicked, terminates the current session and redirects the user to the login page.
- **FR-1.4 User History:**
  - The system shall automatically associate every analysis performed with the logged-in user's account.
  - The system shall provide a "History" page where a user can view a list of their previously analyzed scripts, including the filename, date of analysis, and a link to the detailed results.

2.1.2 Code Submission and Execution

This module covers the core functionality of uploading and processing user-submitted machine learning scripts.

- **FR-2.1 File Upload Interface:**
  - The system shall provide an intuitive file upload component on the main tool page.
  - The system shall restrict file uploads to valid Python (.py) and Jupyter Notebook (.ipynb) formats.
  - The system shall provide clear visual feedback to the user indicating the status of the file upload (e.g., progress bar, success message).
- **FR-2.2 File Validation:**
  - Upon upload, the system shall perform a basic validation to check for file integrity and correct file extension.
  - The system shall display an error message if the user attempts to upload an unsupported file type.
- **FR-2.3 Backend Execution Environment:**
  - The system shall execute the user's script in a secure, isolated backend environment to prevent interference with other processes.
  - The execution environment shall have Python 3.9+ and all necessary libraries (e.g., TensorFlow, PyTorch, Pandas, Scikit-learn) pre-installed to run common ML/DL tasks.

2.1.3 Emission Tracking and Analysis

This module defines the core analysis engine's functions for measuring and processing carbon emissions.

- **FR-3.1 Simultaneous Tracking:**
  - For each script execution, the system **shall** invoke all three integrated tracking libraries—**eco2AI, CodeCarbon, and CarbonTracker**—to monitor the process concurrently.
- **FR-3.2 Data Capture:**
  - The system **shall** capture the primary output metrics from each library. This must include, at a minimum:
    - Total Energy Consumed (in kWh)
    - Total CO_2 Emissions (in $kgCO_2eq$)
    - Execution Duration
    - Hardware Used (CPU/GPU model)
    - Geographical Location / Power Grid Region
- **FR-3.3 Data Aggregation:**

- The system shall aggregate the captured data from the three libraries into a single, structured data object (e.g., JSON) for each analysis run.
- The system shall store this aggregated result object in the database, linking it to the user's account and the specific script version.

2.1.4 Results Visualization and Reporting

This module specifies how the processed analysis data is presented to the user.

- **FR-4.1 Comparison Dashboard:**
  - The system shall present the results on a dedicated dashboard page.
  - This page shall feature a summary table that displays the key metrics (Energy, CO_2eq, Duration) from eco2AI, CodeCarbon, and CarbonTracker in a side-by-side layout for easy comparison.
- **FR-4.2 Graphical Visualization:**
  - The system shall generate a bar chart to visually compare the total CO_2 emissions reported by the three different tools.
  - All charts and graphs shall be clearly labeled with appropriate titles, axes, and units**.**
- **FR-4.3 Raw Output Log:**
  - The system shall provide an expandable section to display the raw console output generated by the user's script during execution for debugging purposes.
- **FR-4.4 Downloadable Report:**
  - The system shall provide a "Download Report" button.
  - This function shall generate a summary of the analysis in a downloadable format (e.g., CSV or PDF), containing the comparison table and key metadata of the run.

2.2 System Requirements

2.2.1 Hardware Requirements

The hardware requirements are defined to ensure a smooth and responsive experience for the end-user, accounting for the computational load of running both the web application and the machine learning scripts.

- **Internet Connectivity:** A stable **Internet connection** is a mandatory requirement. This is not only for accessing the web-based platform but, more critically, for the carbon tracking libraries to function correctly. Libraries like CodeCarbon rely on real-time data APIs to fetch the carbon intensity of the regional power grid where the code is being executed, which is a crucial variable in the emission calculation.
- **Minimum Client System Specifications:** To ensure the user's browser can render the application and its data visualizations without performance issues, a client machine with a **minimum of a dual-core CPU and 4GB of RAM** is recommended. This baseline specification provides sufficient processing power and memory to handle the frontend

components and ensure a fluid user interaction, especially when analyzing results from computationally intensive ML models.

2.2.2 Software Requirements

The software stack for this project is built entirely on open-source technologies, selected for their robustness, extensive community support, and prevalence within the machine learning and web development ecosystems.

- **Programming Language and Environment:** The core of the platform is developed using **Python version 3.9 or higher**. This version is specified to ensure compatibility with the latest features and security updates of the language, as well as with the dependencies of the required libraries. The platform is also designed to process **Jupyter Notebooks (.ipynb)**, which are the de facto standard for interactive data science and ML experimentation.
- Essential Libraries:
    - **Web Framework (Streamlit):** The user interface and web application are built using **Streamlit**. This Python-based framework was chosen for its ability to rapidly create and deploy data-centric applications, making it ideal for a tool that needs to present complex results and visualizations to the user in an accessible manner.
    - **Emission Tracking (eco2AI, CodeCarbon, CarbonTracker):** These three libraries form the core of the measurement engine. Each must be installed in the backend environment to instrument and monitor the execution of user-submitted code.
    - **Data Handling (Pandas):** The **Pandas** library is a critical dependency for data manipulation. It is used to collect, structure, and process the output from the various tracking libraries into a unified format suitable for display and comparative analysis on the dashboard.
- **Frontend Technologies (HTML, CSS, JS):** While Streamlit manages the primary application structure, standard web technologies including **HTML, CSS, and JavaScript** are utilized for frontend customization. These are necessary for enhancing the user interface, applying custom styling beyond Streamlit's default components, and implementing more interactive client-side functionalities to improve the overall user experience.

2.3 Block Diagram

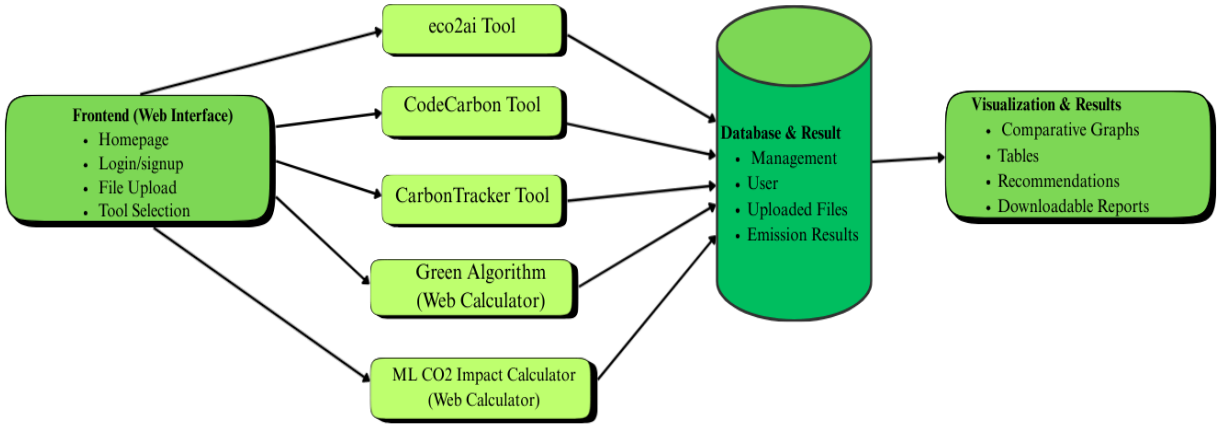

Fig 2.1: Block Diagram

CHAPTER 3: SYSTEM DESIGN

3.1. System Architecture

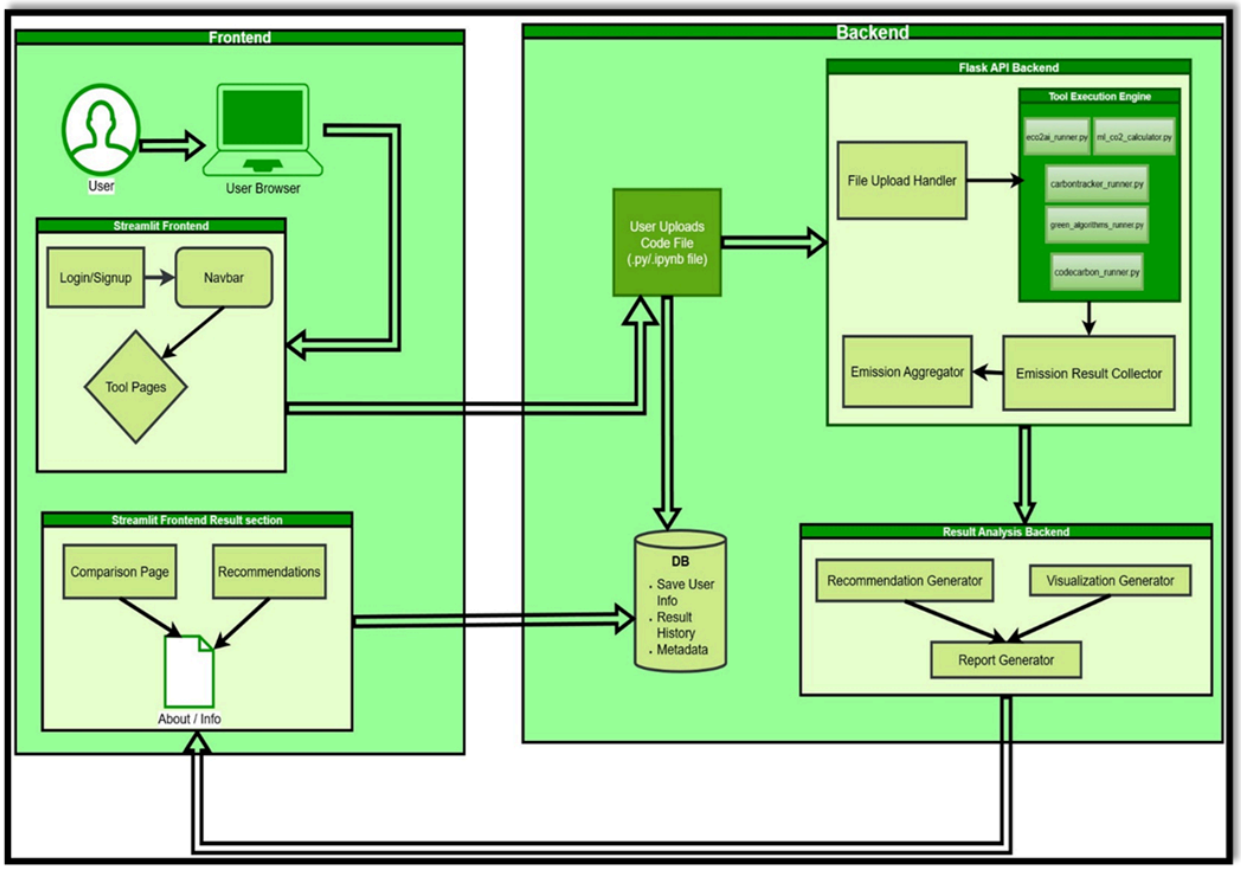

Fig 3.1: System Architecture

3.2. Module Design

The architecture of the Green AI Comparometer is designed as a collection of distinct, interconnected modules. This modular approach ensures a clear separation of concerns, making the system easier to develop, debug, and maintain. Each module is responsible for a specific set of functionalities, from handling user interactions on the frontend to performing complex emission calculations on the backend. The four primary modules are: the User Interface, the Emission Tracking Tools, the Result Management, and the Visualization & Reporting module.

3.2.1. User Interface(UI) Module

The user interface module serves as the primary point of interaction between the user and the platform. It is designed to be intuitive and accessible, guiding the user through the process of analyzing their code.

- **Technology Stack:** The UI is constructed using a combination of technologies. **Streamlit**, a Python-based framework, is used to build the core structure of the web application. Streamlit was specifically chosen for its strength in creating data-centric applications rapidly, allowing for a seamless integration between the frontend display and the backend Python logic. To enhance the user experience and create a unique visual identity, standard web technologies including **HTML, CSS, and JavaScript** are layered on top for custom styling, layout adjustments, and implementing interactive client-side elements.
- **Key Pages:** The UI is organized into several distinct pages:
    - **Homepage:** This is the main landing page, providing an introduction to the Green AI Comparometer, outlining its purpose, and highlighting its key features.
    - **Login/Signup:** This page handles user authentication. It allows new users to register for an account and existing users to log in, which is essential for managing and saving their analysis results.
    - **Tools:** This is the core functional page of the application. Here, users can upload their Python (`.py`) or Jupyter Notebook (`.ipynb`) files. The page contains the interface elements to initiate the analysis, monitor its progress, and view the results once the backend processing is complete.

3.2.2. Emission Tracking Module

This module is the backend engine of the platform, responsible for executing the user's code and measuring its environmental impact. It operates in a controlled server-side environment to ensure consistency and accuracy.

- **Core Functionality:** The primary function of this module is to programmatically integrate and run the three key carbon tracking libraries: **eco2AI, CodeCarbon, and CarbonTracker**.
- **Workflow:** The process is initiated when a user uploads a script via the UI. The workflow is as follows:
    - **Receive Script:** The module receives the `.py` or `.ipynb` file from the Result Management module.
    - **Instrument Code:** It sets up an isolated execution environment where the user's code is "wrapped" by the trackers. Each of the three libraries is invoked to monitor the script's execution from start to finish.
    - **Execute and Monitor:** The script is run. During this time, the libraries track hardware power consumption (CPU, GPU), execution duration, and query external APIs for the carbon intensity of the local power grid.
    - **Capture Emissions Data:** Upon completion, the module captures the final output from each library—typically including total energy consumed in kWh and the estimated carbon emissions in kilograms of CO2 equivalent

(CO2eq). This raw data is then passed to the Result Management module for storage.

3.2.3. Result Management Module

The Result Management module acts as the central data handling layer, connecting the backend processing with the frontend display. It is responsible for the temporary storage of user files and the persistent storage of analysis results.

- **Handling User Uploads:** When a user uploads a script, this module securely handles the file, storing it temporarily in preparation for execution by the Emission Tracking module.
- **Storing Results:** After the Emission Tracking module has finished its analysis, this module receives the processed data (emission figures, runtime, etc.). It then saves this information in a structured format, linking it to the specific user's account and the particular script that was run. This enables users to access a historical log of their previous analyses, track improvements over time, and compare results from different experiments.

3.2.4. Visualizing and Reporting Module

This module is responsible for transforming the raw data stored by the Result Management module into a human-readable and insightful format for the user.

- **Data Presentation:** The module presents the results in multiple ways to cater to different user needs. This includes displaying the raw **console output** from the script's execution, which is useful for debugging purposes. The core feature is the presentation of clear **emission tables** that show the key metrics from eco2AI, CodeCarbon, and CarbonTracker in a side-by-side comparison.
- **Graphical Analysis and Downloads:** To facilitate easier interpretation of the data, this module generates **graphs and charts**—for instance, a bar chart visually comparing the CO2eq reported by each of the three tools. Furthermore, it provides functionality for users to generate and **downloadable reports**. This allows the analysis to be easily shared, archived, or included in other documents, such as academic papers or project summaries.

3.3 Interface Design

3.3.1 User Interface Screen Design

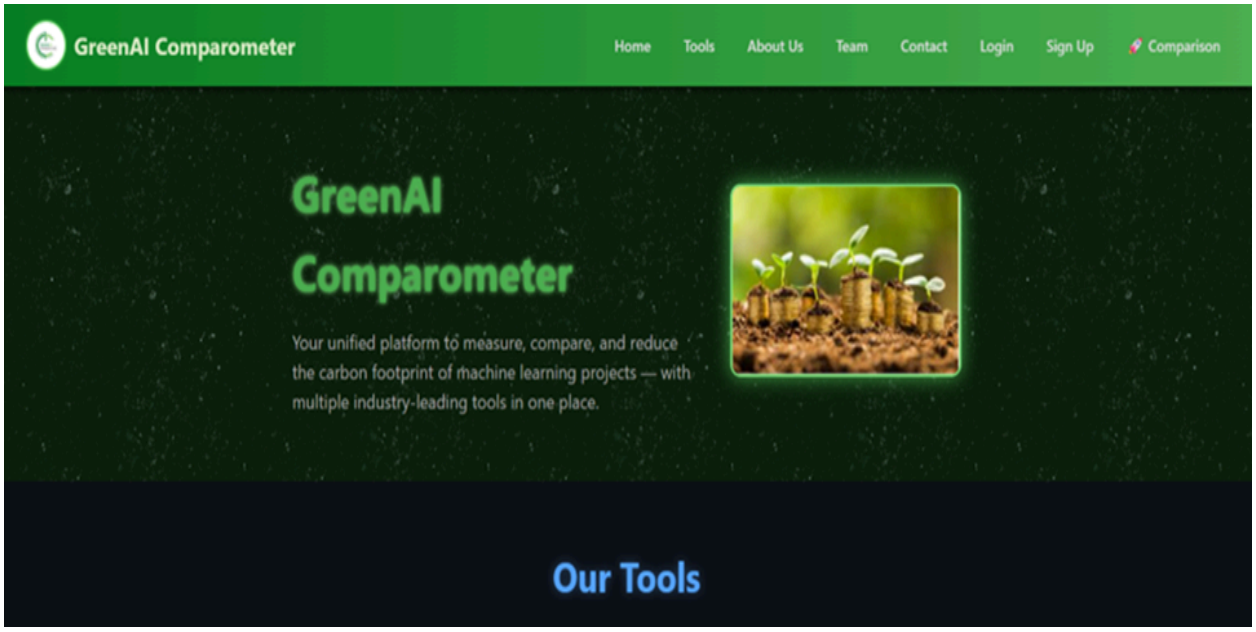

Fig 3.2: Front Design Highlights (Homepage)

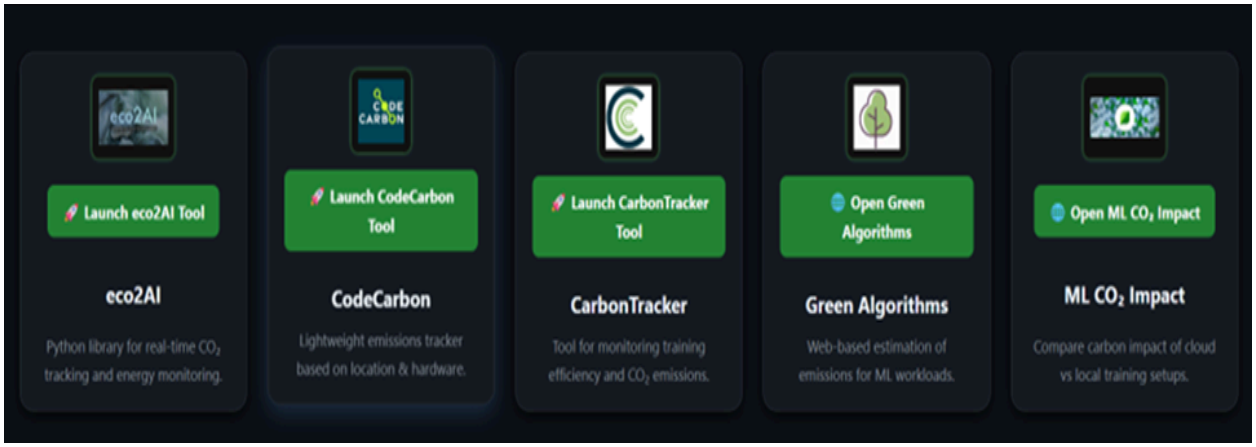

Fig 3.3: Front Design Highlights (Homepage)

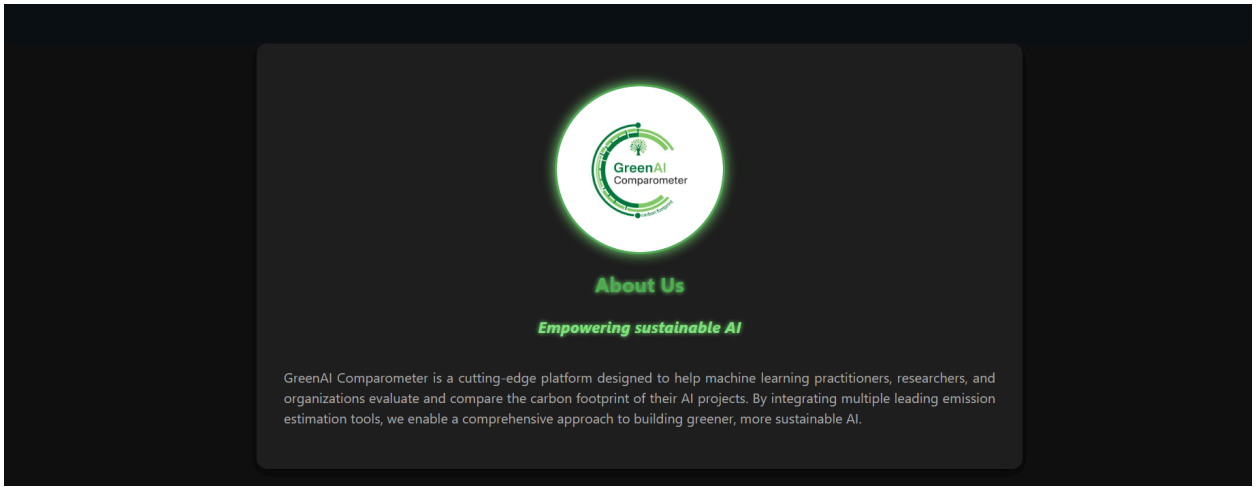

Fig 3.4: Front Design Highlights (Homepage)

.

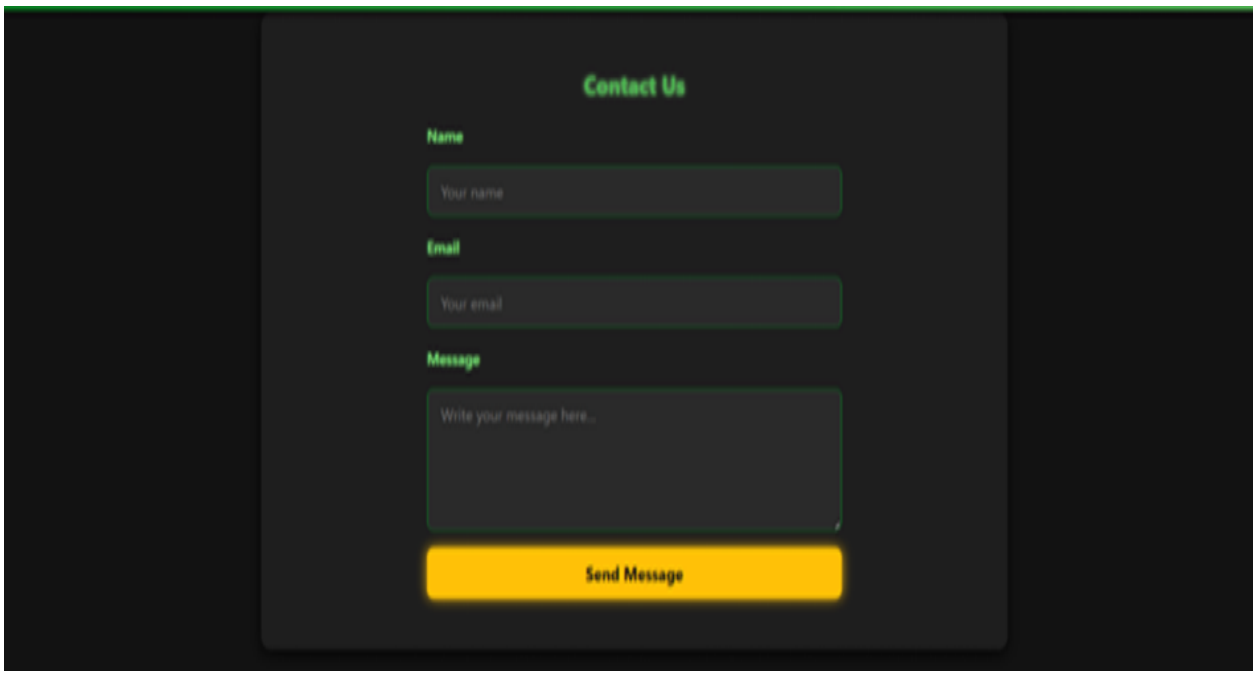

Fig 3.5: Front Design Highlights (Homepage)

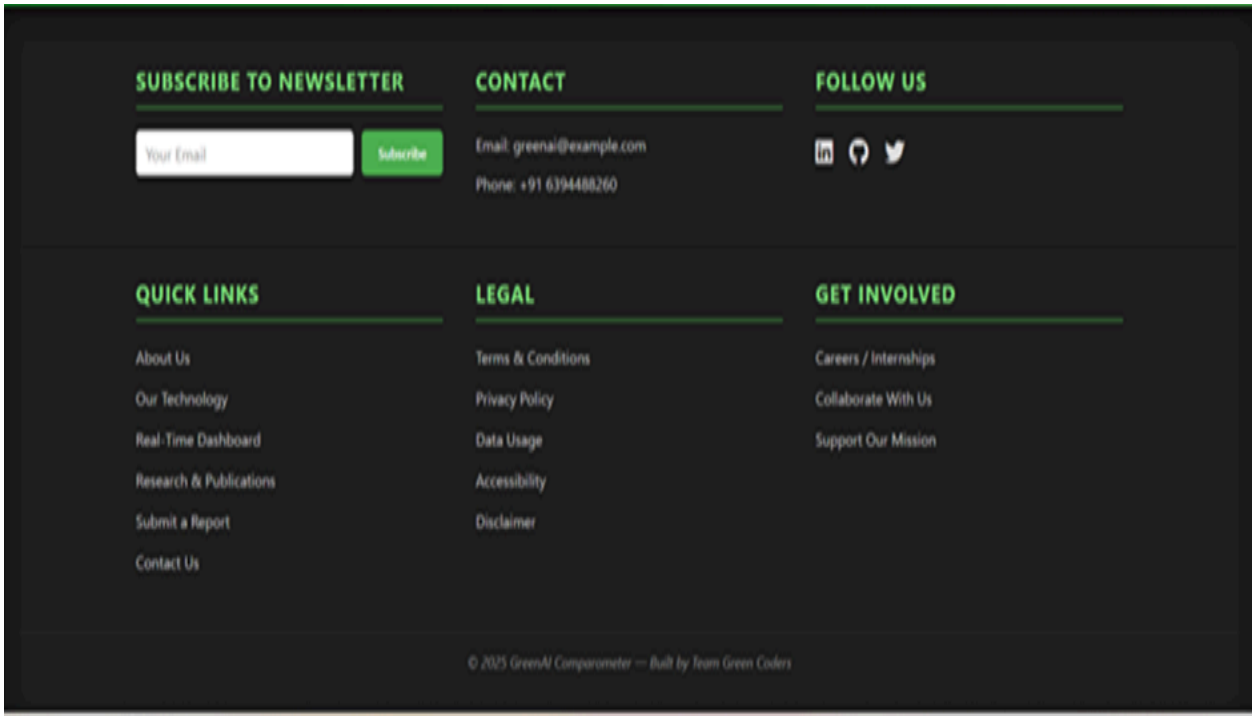

Fig 3.6: Front Design Highlights (Homepage)

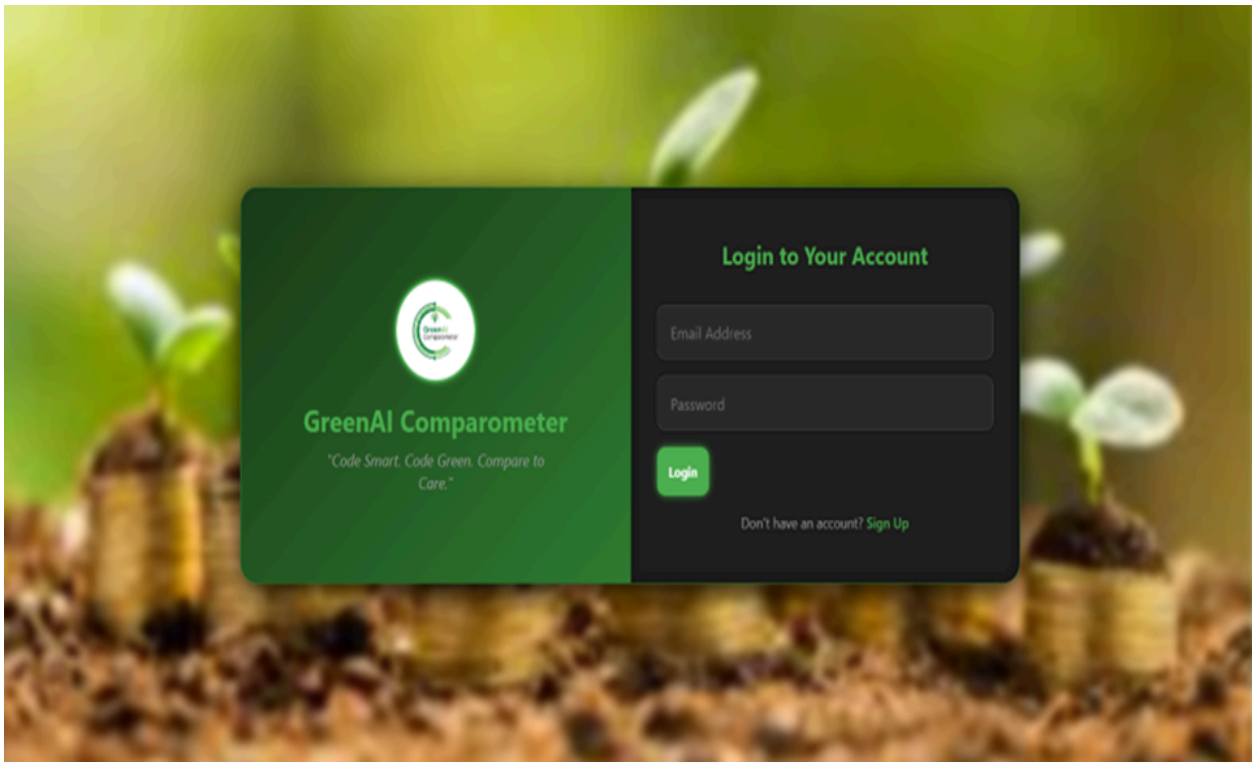

Fig 3.7: Front Design Highlights (Login Page)

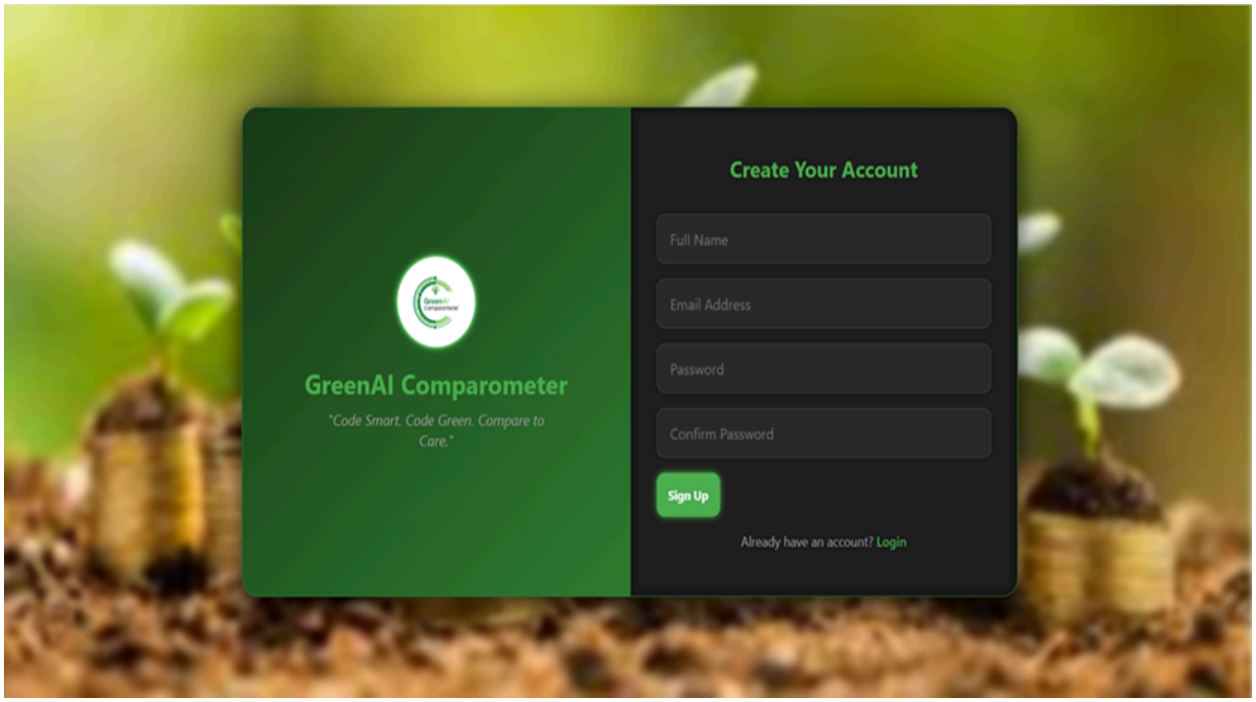

Fig 3.8: Front Design Highlights (Signup Page)

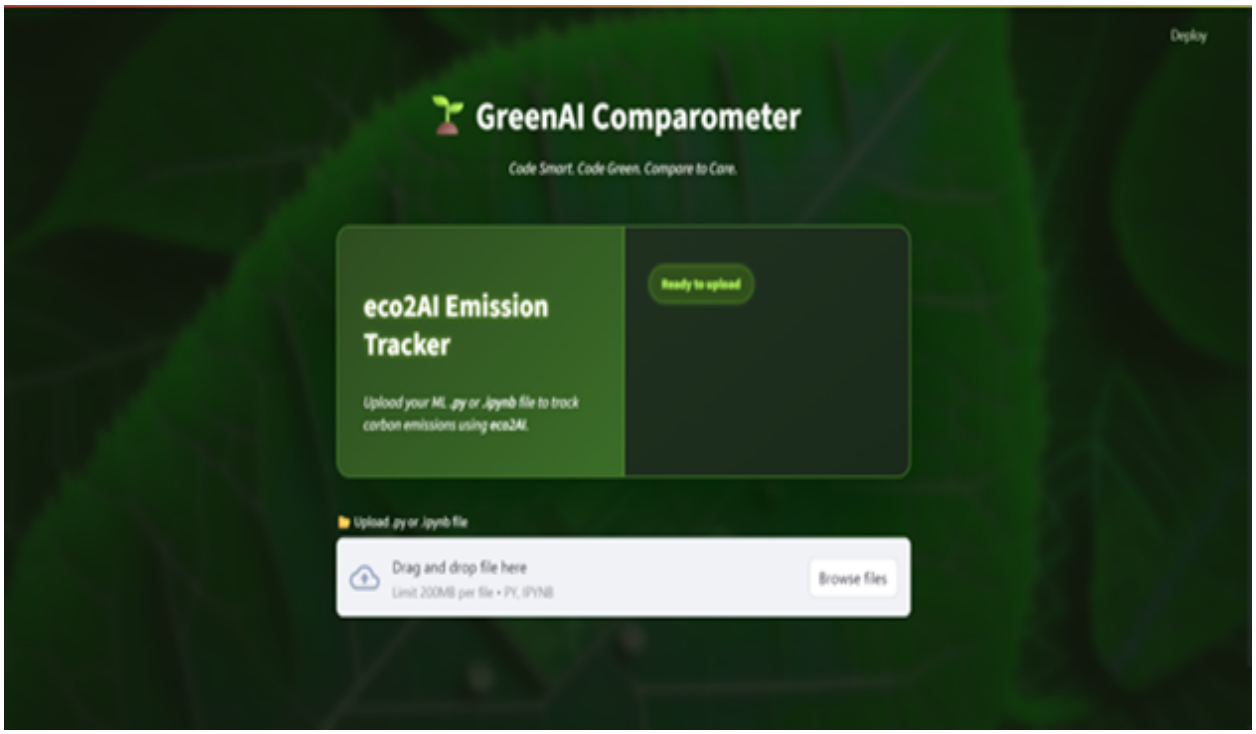

Fig 3.9: eco2AI Front Page

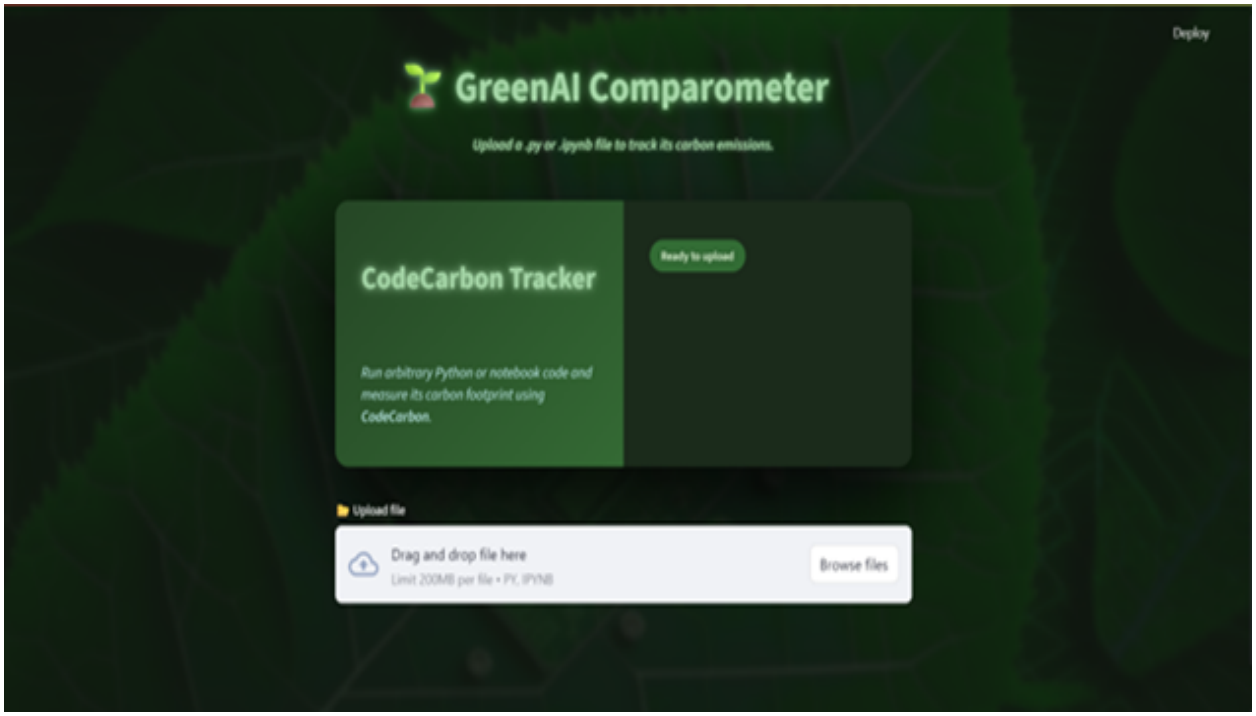

Fig 3.10: CodeCarbon Front Page

Fig 3.11: CarbonTracker Front Page

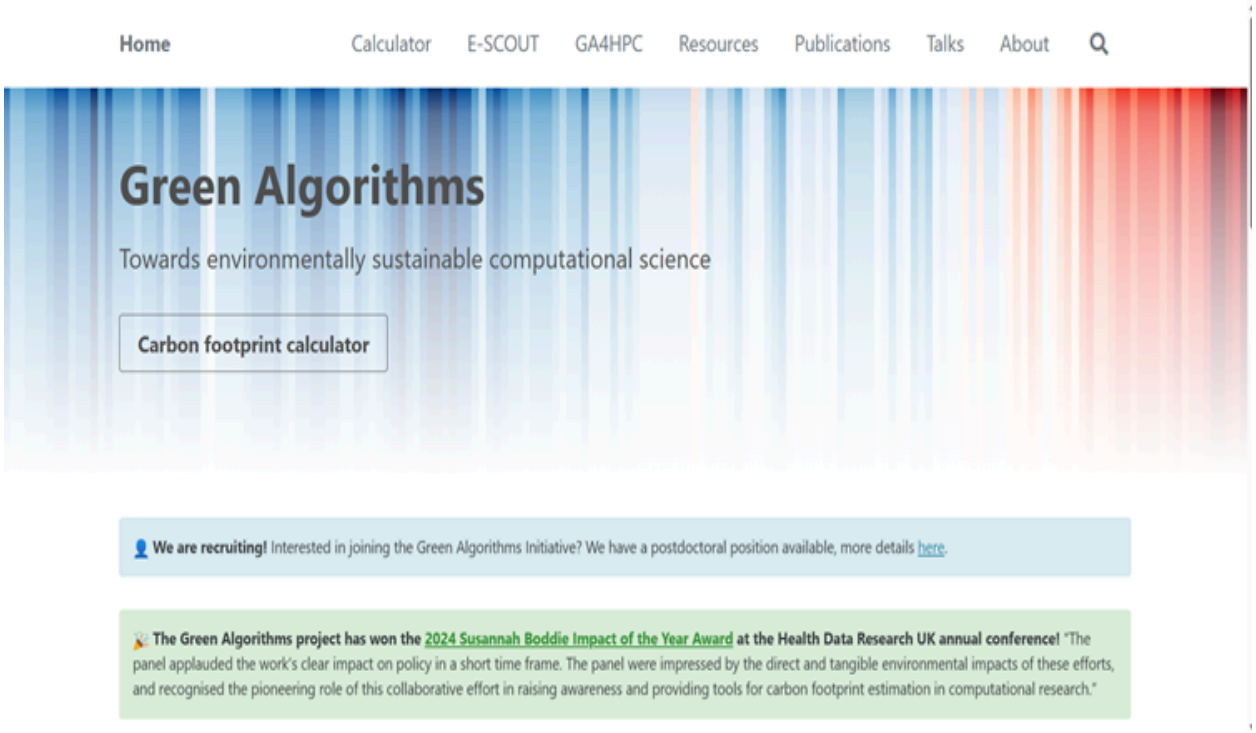

Fig 3.12: Green Algorithms Front Page

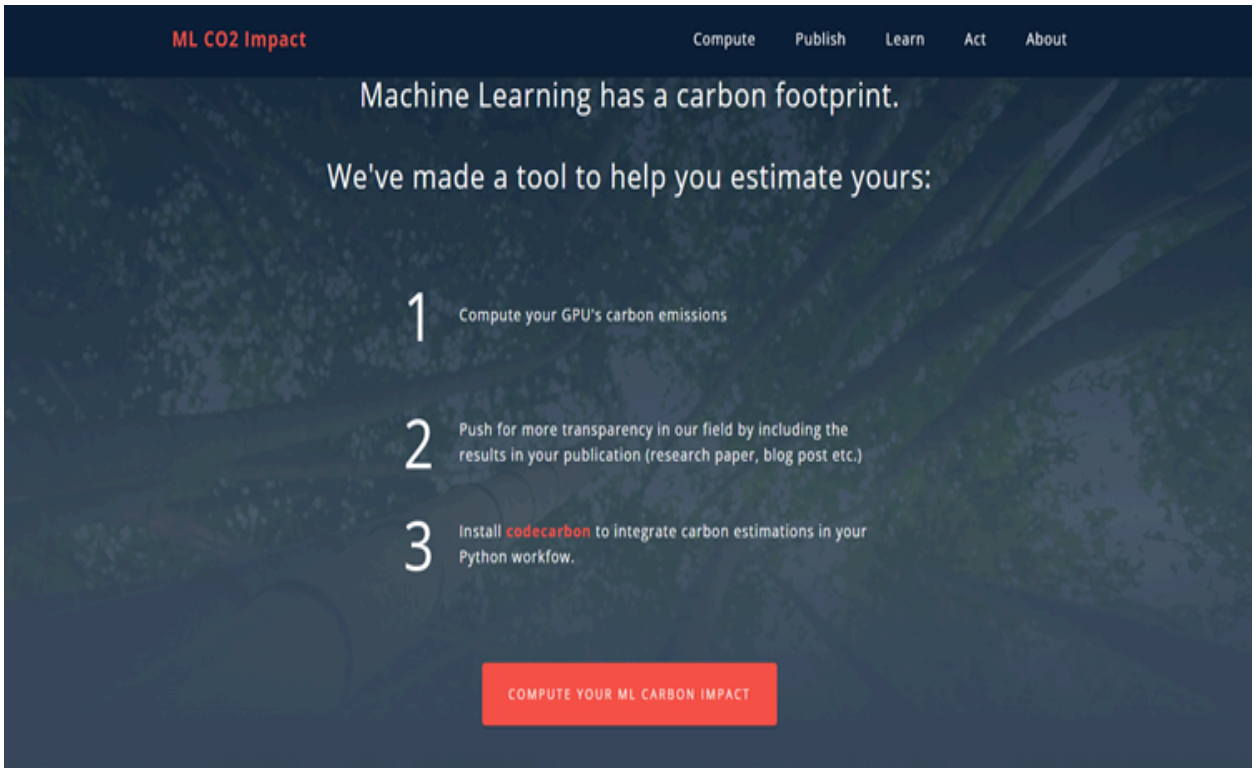

Fig 3.13: ML CO2 Impact Calculator Front Page

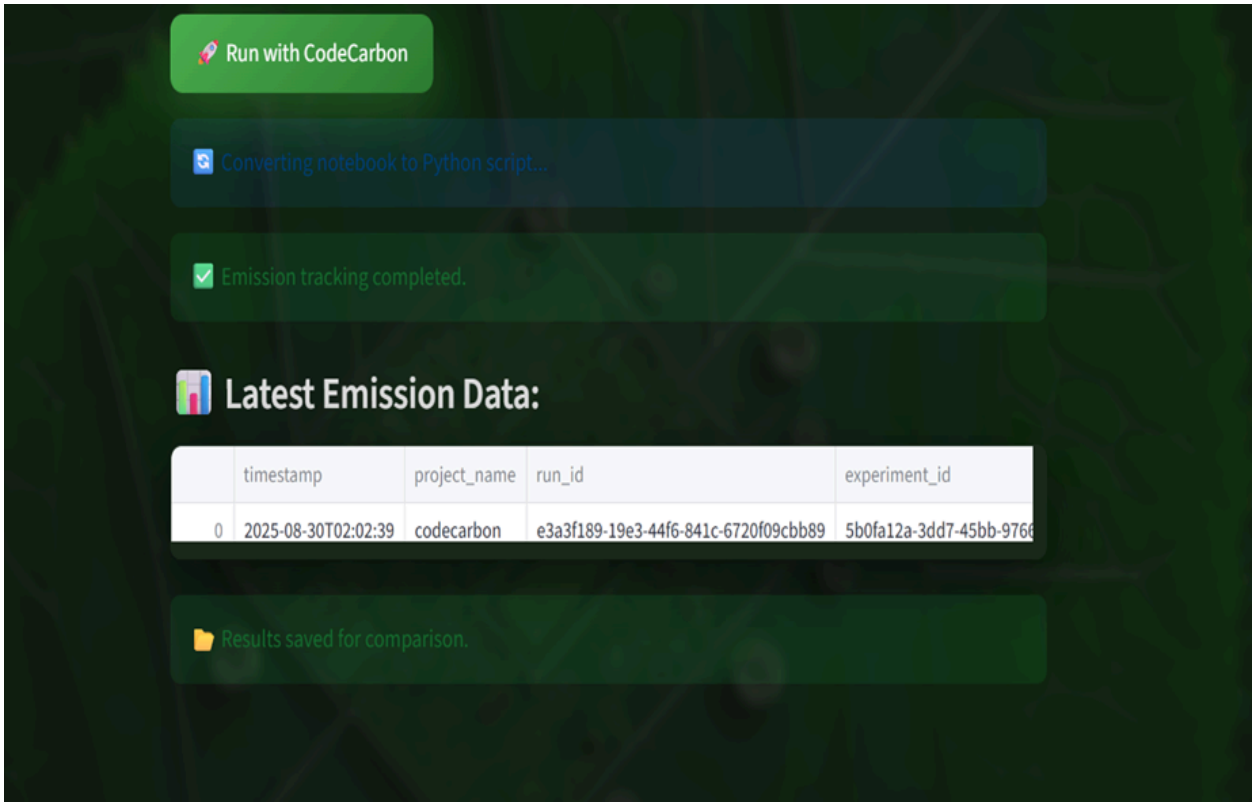

Fig 3.14: CodeCarbon Output

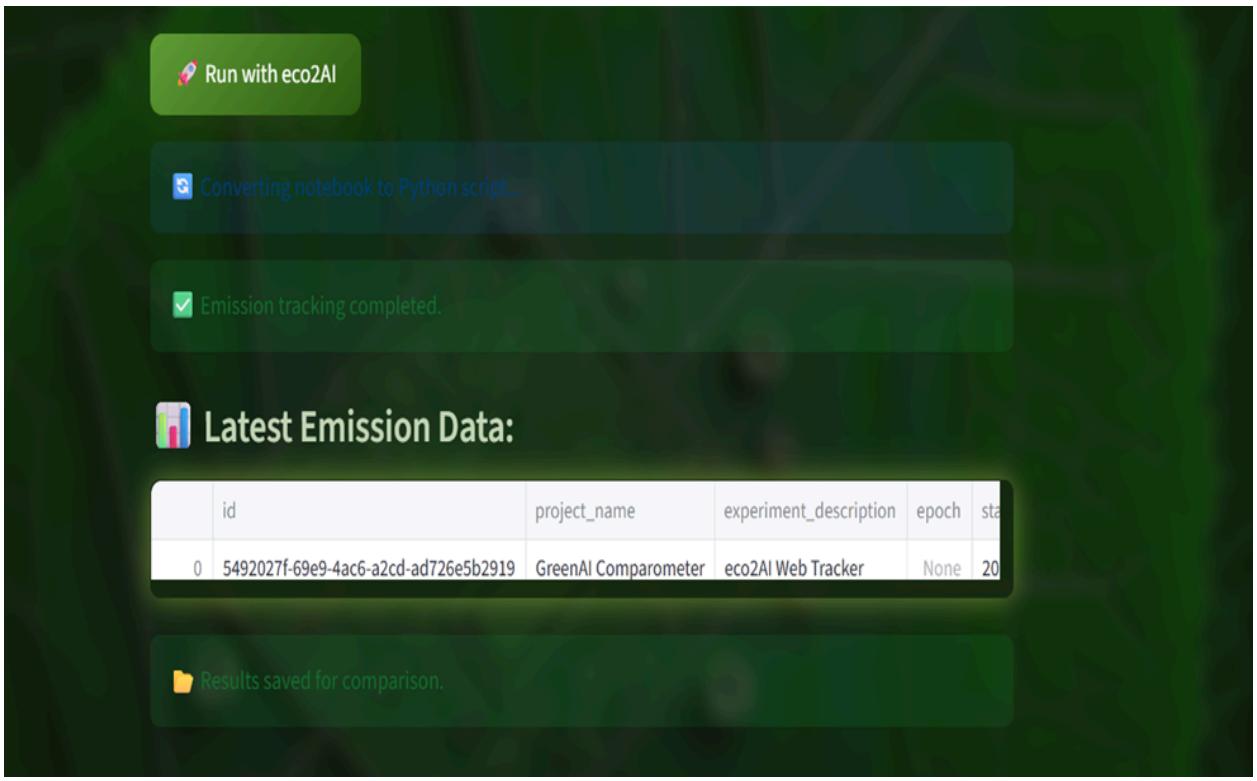

Fig 3.15: Eco2AI Output

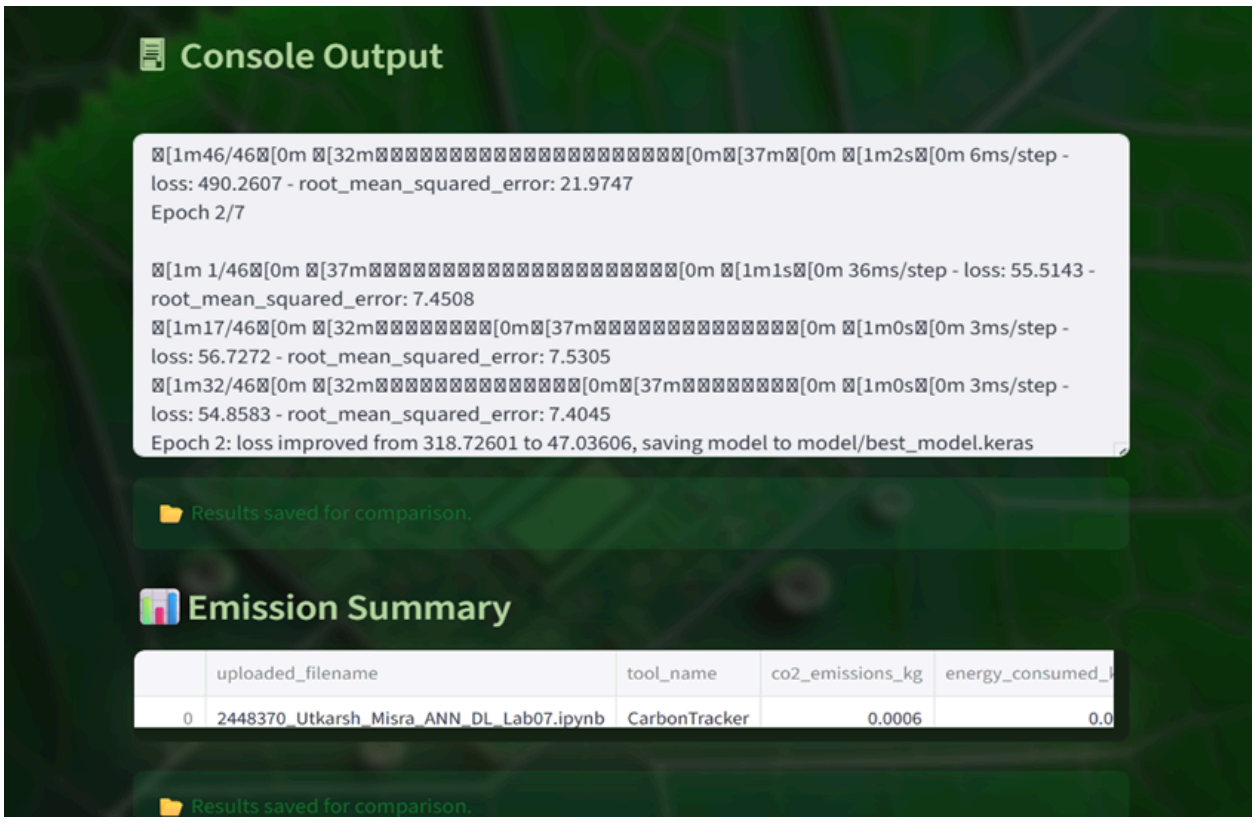

Fig 3.16: CarbonTracker Output

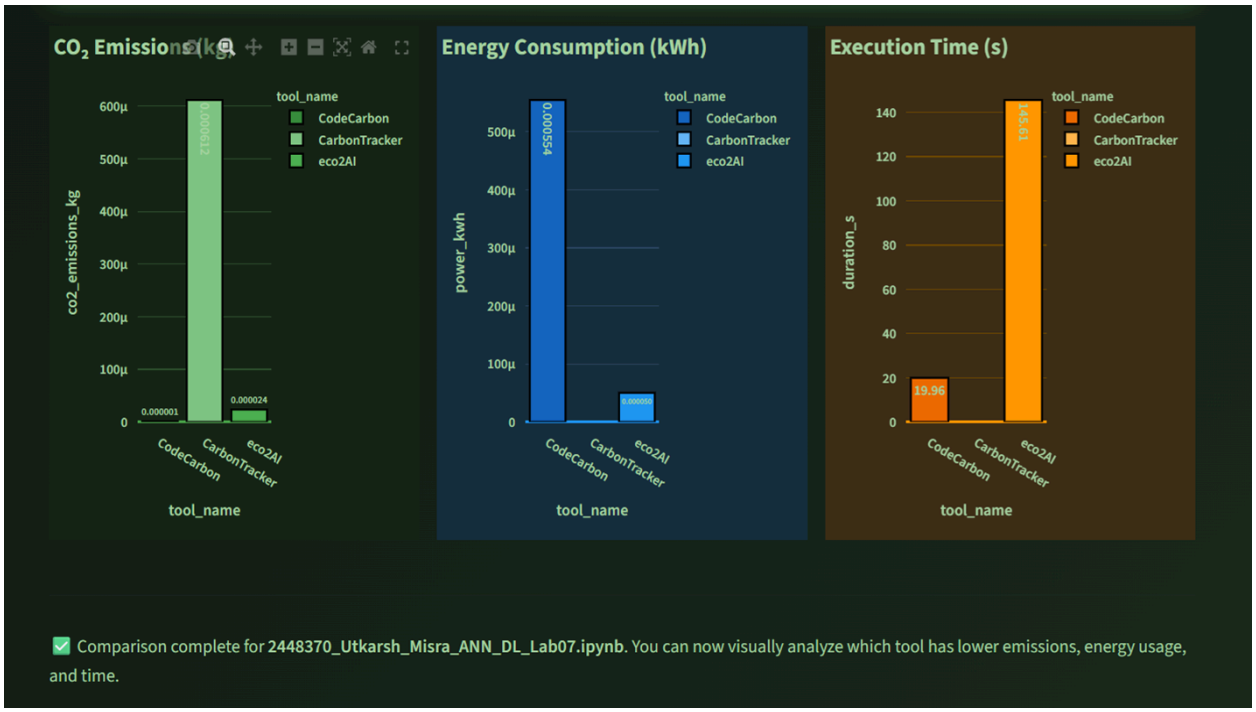

Fig 3.17: Comparison Table Visualization

CHAPTER 4: IMPLEMENTATION

The code folder structure is as follows:

```
Green-AI-Comparometer-main/
│
├── .gitignore
├── README.md
├── requirements.txt
├── runtime.txt
│
├── app.py # Main entry point for the web app.
├── comparison_page.py # The main comparison view.
│
├── carbontracker/
│       ├── __init__.py
│       ├── runner.py # (was carbontracker_runner.py)
│       └── web_tool.py # (was carbontracker_web_tool.py)
│
├── codecarbon/
│       ├── __init__.py
│       ├── runner.py # (was codecarbon_runner.py)
│       └── web_tool.py # (was codecarbon_web_tool.py)
│
└── eco2ai/
        ├── __init__.py
        ├── runner.py # (was eco2ai_runner.py)
        └── web_tool.py # (was eco2ai_web_tool.py)
```

4.1 Coding Standards

4.1.1 Naming Conventions

Consistency in naming is vital for readability. We will follow the standard conventions outlined in **PEP 8**.

- **Modules & Packages**: Use short, all-lowercase names
  - *Good*: src/runners/, src/web/
  - *Bad*: src/Runners/, src/WebUtils/
- **Functions & Variables**: Use snake_case (lowercase with underscores).
  - *Good*: def calculate_carbon_emissions():, total_consumption = 0
  - *Bad*: def CalculateCarbonEmissions():, TotalConsumption = 0
- **Classes**: Use PascalCase (also known as CapWords).
  - *Good*: class CarbonTrackerModel:
  - *Bad*: class carbon_tracker_model:
- **Constants**: Use ALL_CAPS with underscores.
  - *Good*: DEFAULT_REGION = "us-east-1"
  - *Bad*: default_region = "us-east-1"

4.1.2 Code Layout

Proper layout makes code easy to scan and understand.

- **Indentation**: Use **4 spaces** per indentation level. Do not use tabs.
- **Maximum Line Length**: Keep lines under **88 characters**. This is the standard used by the popular black formatter and improves readability, especially on split screens.
- **Blank Lines**:
  - Use **two blank lines** to separate top-level functions and classes.
  - Use **one blank line** to separate methods inside a class.
  - Use blank lines sparingly inside functions to group logical sections.
- **Imports**:
  - Imports should be at the top of the file.
  - Group imports in the following order:
    - Standard library imports (e.g., os, sys).
    - Third-party library imports (e.g., pandas, streamlit).
    - Local application/your own module imports (e.g., from src.runners import codecarbon_runner).

4.1.3 Comments and Docstrings

Code tells you *how*, comments should tell you *why*.

- **Block Comments**: Use comments (#) to explain complex logic, assumptions, or workarounds. Keep them updated as the code changes.

- **Docstrings**: Every public module, function, class, and method should have a docstring. We will use the **Google Style** for its readability.

4.1.4 General Practices and Principles

- **DRY (Don't Repeat Yourself)**: If you find yourself copying and pasting code, it's a sign you should create a function or a class to encapsulate that logic.
- **Type Hinting**: Use modern Python type hints for all function signatures and variables where appropriate. This improves code clarity, allows for static analysis, and helps IDEs provide better support.
    - *Good*: def get_user(user_id: int) -> dict:
    - *Bad*: def get_user(user_id):
- **Error Handling**: Use try...except blocks to handle potential errors gracefully (e.g., file not found, API request fails). Avoid broad except: clauses; always specify the exception you expect to catch (e.g., except FileNotFoundError:).
- **Prefer f-strings**: For formatting strings, use f-strings as they are more readable and faster than other methods.
    - *Good*: emissions = 12.345; print(f"Total emissions: {emissions:.2f} kg CO2")
    - *Bad*: print("Total emissions: %.2f kg CO2" % emissions)

4.1.5 Automated Tooling

To enforce these standards automatically, we will use the following tools. Configure them in a pyproject.toml file in your project's root directory.

- **Formatter (black)**: An opinionated code formatter that automatically reformats your code to be consistent. This eliminates all arguments about style.
    - **Usage**: black .
- **Linter (ruff or flake8)**: A tool that checks your code for style violations (like unused imports, lines that are too long) and potential bugs. `Ruff` is a modern, extremely fast alternative to flake8.
    - **Usage**: `ruff check .`

Using these tools will ensure every file in the project is clean, consistent, and adheres to the standards with minimal manual effort.

4.2 Backend/Output Screenshots

```
# Core UI
streamlit==1.37.0
pandas==1.4.3
plotly==5.23.0

# ML/DL carbon tracking libraries
eco2ai==0.2.5
codecarbon==2.3.5
carbontracker==1.2.4

# Notebook conversion
jupyter==1.0.0
nbconvert==7.16.4

# Extra utilities
psutil==5.9.8
matplotlib==3.7.5
```

Fig 4.1: requirements.txt

```
import sys
import os
import subprocess
from eco2ai import Tracker
import pandas as pd
import time
from pathlib import Path
import shutil
import uuid
```

Fig 4.2: import statements for eco2ai_runner.py

```
import streamlit as st
import os
import subprocess
from eco2ai import Tracker
import pandas as pd
import uuid
import io
import contextlib
import time
from pathlib import Path
import shutil
```

Fig 4.3 import statements for eco2ai_web_tool.py

```python
def run_with_eco2ai(user_code_path):
    original_wd = Path.cwd()
    results_file = original_wd / "comparison_results.csv"

    session_dir = original_wd / "temp_sessions" / str(uuid.uuid4())
    session_dir.mkdir(parents=True, exist_ok=True)

    shutil.copy(user_code_path, session_dir)
    user_code_filename = Path(user_code_path).name

    os.chdir(session_dir)

    print(f"✅ Tracking emissions for: {user_code_filename}")

    if user_code_filename.endswith(".ipynb"):
        base_name = os.path.splitext(user_code_filename)[0]
        subprocess.run(
            ["jupyter", "nbconvert", "--to", "python", user_code_filename, "--output", base_name],
            check=True
        )
        run_file = f"{base_name}.py"
    else:
        run_file = user_code_filename

    tracker = Tracker(
        project_name="GreenAI Comparometer",
        experiment_description=f"Tracking {user_code_filename}"
    )

    try:
        tracker.start()
        exec(Path(run_file).read_text(), {})
        tracker.stop()
        print("✅ Execution complete.")

        # ✅ read results BEFORE cleanup
```

Fig 4.4:code snippet for eco2ai_runner.py

```python
uploaded_code_file = st.file_uploader("📂 Upload .py or .ipynb file", type=["py", "ipynb"])
uploaded_dataset_files = st.file_uploader(
    "📦 Upload dataset files (e.g., .csv, .json, .txt)",
    type=["csv", "json", "txt"],
    accept_multiple_files=True
)

if uploaded_code_file:
    # Create a fresh session dir for this run
    session_dir = temp_root_dir / str(uuid.uuid4())
    session_dir.mkdir(parents=True, exist_ok=True)

    # Save code file
    code_path = session_dir / uploaded_code_file.name
    with open(code_path, "wb") as f:
        f.write(uploaded_code_file.getbuffer())
    st.success(f"✅ Code file '{uploaded_code_file.name}' uploaded.")

    # Save dataset files
    if uploaded_dataset_files:
        for file in uploaded_dataset_files:
            dataset_path = session_dir / file.name
            with open(dataset_path, "wb") as f:
                f.write(file.getbuffer())
        st.success(f"✅ {len(uploaded_dataset_files)} dataset(s) uploaded.")

    if st.button("🚀 Run with eco2AI"):
        try:
            os.chdir(session_dir)

            # Convert notebook if needed
            if uploaded_code_file.name.endswith(".ipynb"):
                st.info("🔄 Converting notebook to Python script...")
                base_name = uploaded_code_file.name.replace(".ipynb", "")
                subprocess.run(
                    ["jupyter", "nbconvert", "--to", "python", uploaded_code_file.name, "--output", base_name],
                    check=True
```

Fig 4.5:code snippet for eco2ai_web_tool.py

tools > codecarbon_runner.py > ...

```python
import sys
import os
import subprocess
import pandas as pd
import time
from pathlib import Path
from codecarbon import EmissionsTracker
import shutil
import uuid
```

Fig 4.6:import statements for codecarbon_runner.py

tools > codecarbon_web_tool.py > ...

```python
import streamlit as st
import os
import subprocess
from codecarbon import EmissionsTracker
import pandas as pd
import uuid
import io
import contextlib
import time
from pathlib import Path
import shutil
```

Fig 4.7: import statements for codecarbon_web_tool.py

Fig 4.8: code snippet for codecarbon_runner.py

```python
        wrapped_file_path = f"carbon_run_{str(uuid.uuid4())[:8]}.py"
        with open(wrapped_file_path, "w", encoding="utf-8") as f:
            f.write(wrapped_code)

        # --- Capture output and execute ---
        output_buffer = io.StringIO()
        with contextlib.redirect_stdout(output_buffer), contextlib.redirect_stderr(output_buffer):
            exec(Path(wrapped_file_path).read_text(), {})

        result_output = output_buffer.getvalue()
        st.subheader("🖥️ Console Output")
        st.text_area("", result_output, height=250)

        # --- Extract Energy & CO₂ ---
        energy_match = re.search(r"Energy:\s*([\d.]+)\s*kWh", result_output)
        co2_match = re.search(r"CO2eq:\s*([\d.]+)\s*g", result_output)

        if energy_match and co2_match:
            energy_kwh = float(energy_match.group(1))
            co2_kg = float(co2_match.group(1)) / 1000.0  # convert g → kg
            duration_match = re.search(r"Time:\s*(\d+):(\d+):(\d+)", result_output)
            if duration_match:
                h, m, s = map(int, duration_match.groups())
                duration_s = h * 3600 + m * 60 + s
            else:
                duration_s = None

            result_row = {
                "uploaded_filename": uploaded_code_file.name,
                "tool_name": "CarbonTracker",
                "co2_emissions_kg": co2_kg,
                "energy_consumed_kwh": energy_kwh,
                "duration_seconds": duration_s,
                "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
            }
```

Fig 4.9: code snippet for codecarbon_web_tool.py

```python
import sys
import os
import subprocess
import pandas as pd
import time
from pathlib import Path
from carbontracker.tracker import CarbonTracker
import glob
import shutil
import uuid
```

Fig 4.10: import statements for carbontracker_runner.py

```python
import streamlit as st
import os
import uuid
import subprocess
import pandas as pd
import io
import contextlib
import time
import re
from pathlib import Path
import shutil
```

Fig 4.11: import statements for carbontracker_web_tool.py

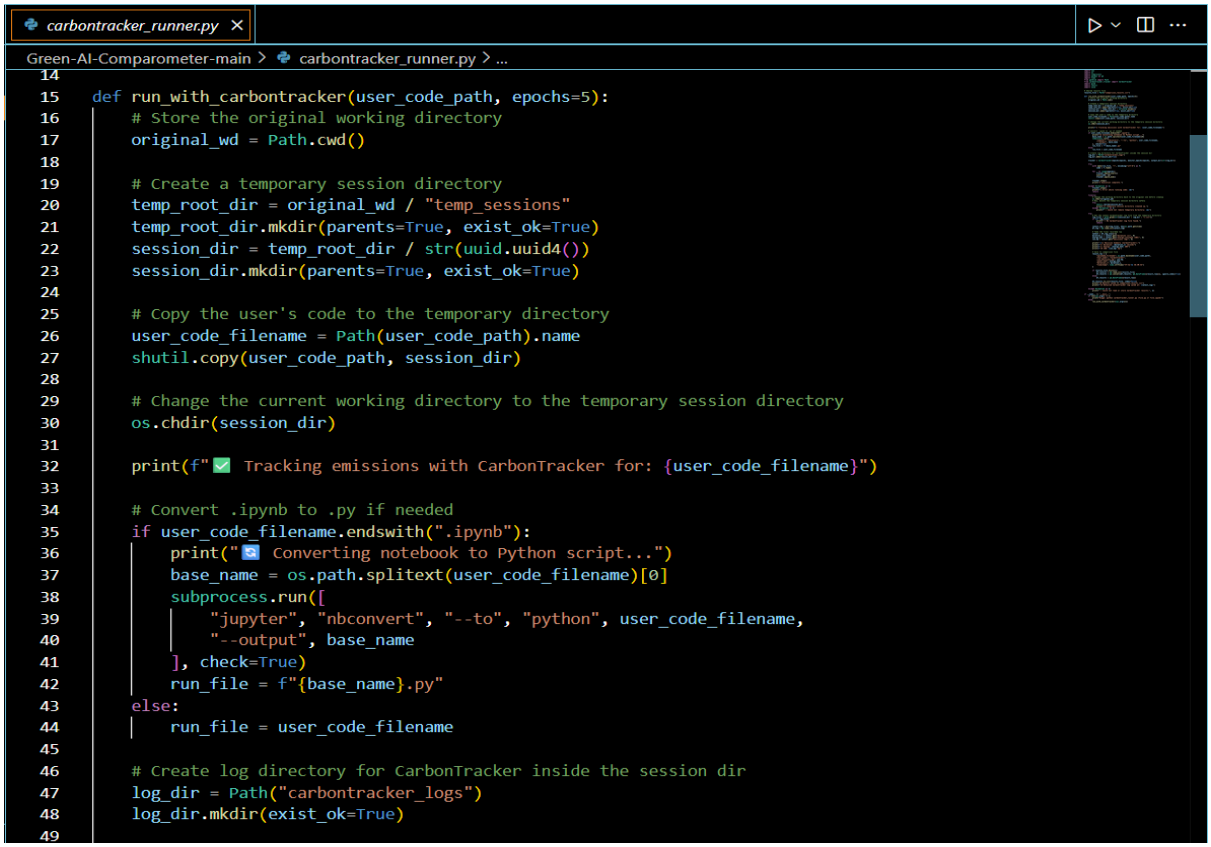

Fig 4.12: code snippet from carbontracker_runner.py

```python
tracker = CarbonTracker(epochs=epochs, monitor_epochs=epochs, output_dir=str(log_dir))

try:
    with open(run_file, "r", encoding="utf-8") as f:
        code = f.read()

    for _ in range(epochs):
        tracker.epoch_start()
        exec(code, {})
        tracker.epoch_end()

    tracker.stop()
    print("✅ Execution complete.")

except Exception as e:
    tracker.stop()
    print(f"❌ Error while running code: {e}")
    return

finally:
    # Change the working directory back to the original one before cleanup
    os.chdir(original_wd)
    # Now, delete the temporary session directory safely
    try:
        shutil.rmtree(session_dir)
        print("🧹 Temporary session directory cleaned up.")
    except Exception as e:
        print(f"⚠️ Could not remove temporary directory: {e}")

try:
    # Get the latest CarbonTracker log file from the temporary directory
    log_files = glob.glob(str(session_dir / log_dir / "*.csv"))
    if not log_files:
        print("⚠️ No CarbonTracker log file found.")
        return
```

Fig 4.13: code snippet from carbontracker_runner.py

```python
import streamlit as st
import pandas as pd
import plotly.express as px
from pathlib import Path
```

Fig 4.14: import statements for comparison_page.py

```python
st.markdown(
    """
    <div class="title-block">
        <h1>📊 GreenAI Comparometer - Emissions Comparison</h1>
        <p>Compare results from <strong>eco2AI</strong>, <strong>CodeCarbon</strong>, and <strong>CarbonTracker</strong>.</p>
    </div>
    """,
    unsafe_allow_html=True,
)

if not results_file.exists():
    st.warning("⚠️ No results found. Please run at least one tracker first.")
    st.stop()

df = pd.read_csv(results_file)

filenames = df["uploaded_filename"].unique().tolist()
selected_file = st.selectbox("Select file to compare:", filenames)

df_filtered = df[df["uploaded_filename"] == selected_file]

if df_filtered.empty:
    st.warning("⚠️ No data available for the selected file.")
    st.stop()

st.subheader("📄 Results Table")
st.dataframe(df_filtered, use_container_width=True)

col1, col2, col3 = st.columns(3)

with col1:
    fig_co2 = px.bar(
        df_filtered,
        x="tool_name",
        y="co2_emissions_kg",
        title="CO₂ Emissions (kg)",
```

Fig 4.15: code snippet from comaprison_page.py

```
tools > comparison_page.py > ...
180             "CodeCarbon": "#1565c0",
181             "CarbonTracker": "#64b5f6"
182         },
183         template="plotly_dark"
184     )
185     fig_power.update_layout(
186         plot_bgcolor="#182f3e",
187         paper_bgcolor="#182f3e",
188         font_color="#a8d5a3",
189         title_font_size=20,
190         title_font_color="#5db7f5",
191         margin=dict(t=50, b=30, l=40, r=40),
192         yaxis=dict(showgrid=True, gridcolor='#2c4660', zerolinecolor='#2196f3'),
193         xaxis=dict(showgrid=False)
194     )
195     fig_power.update_traces(marker_line_color='black', marker_line_width=1.8)
196     st.plotly_chart(fig_power, use_container_width=True)
197
198 with col3:
199     fig_duration = px.bar(
200         df_filtered,
201         x="tool_name",
202         y="duration_s",
203         title="Execution Time (s)",
204         color="tool_name",
205         text_auto=".2f",
206         color_discrete_map={
207             "eco2AI": "#ff9800",
208             "CodeCarbon": "#ef6c00",
209             "CarbonTracker": "#ffb74d"
210         },
211         template="plotly_dark"
212     )
213     fig_duration.update_layout(
214         plot_bgcolor="#3e2e18",
215         paper_bgcolor="#3e2e18",
```

Fig 4.16: code snippet from comaprison_page.py

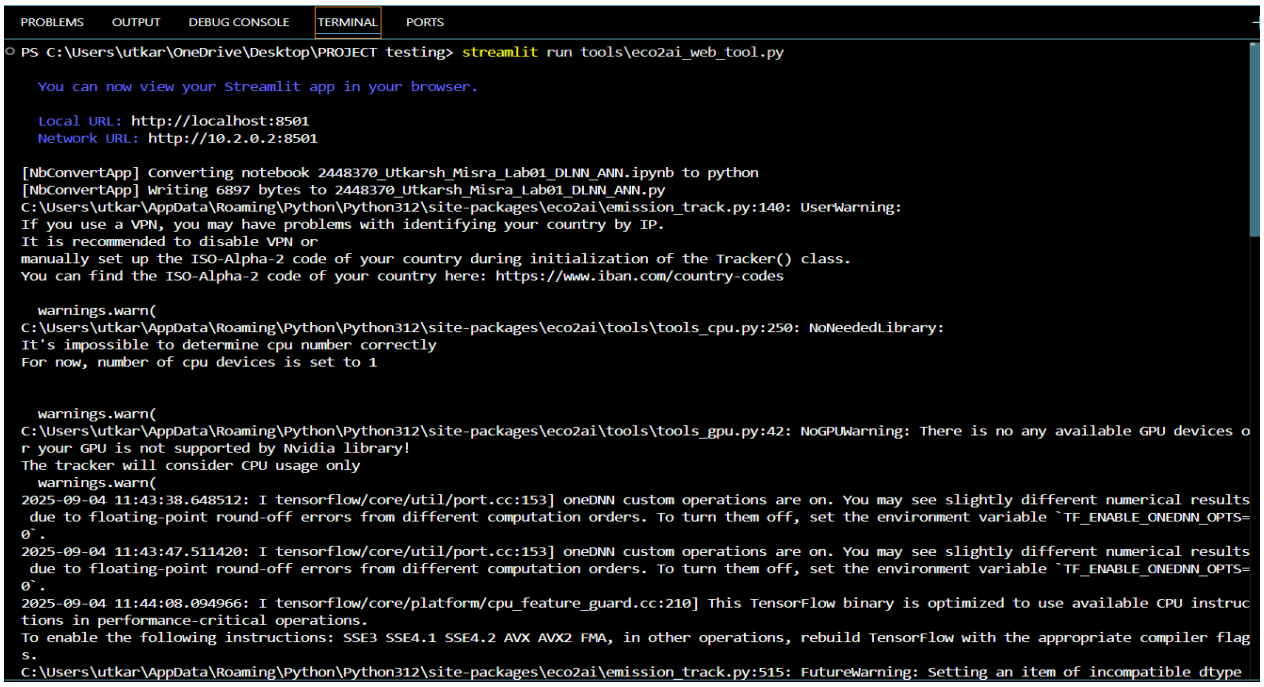

Fig 4.17: Terminal Screenshot for eco2ai_web_tool.py

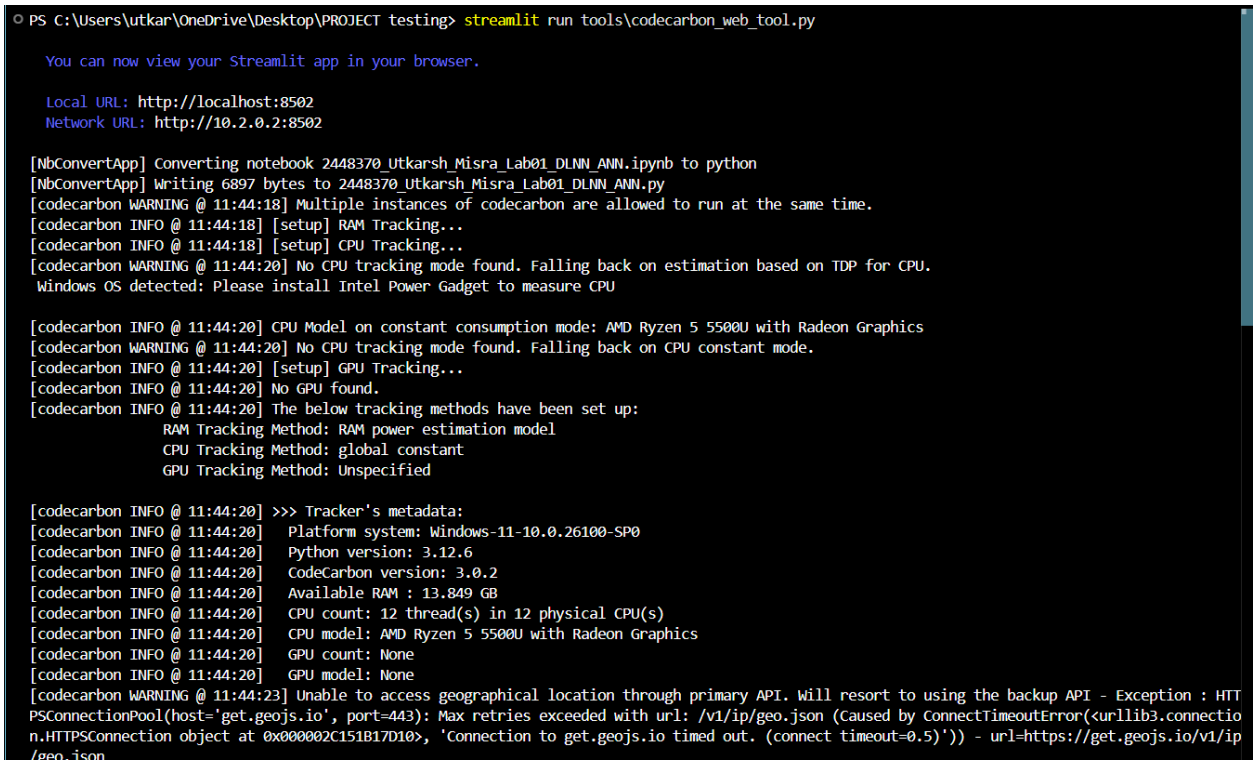

Fig 4.18: Terminal Screenshot for codecarbon_web_tool.py

```
[codecarbon INFO @ 11:44:40] Energy consumed for RAM : 0.000042 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:40] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:40] Energy consumed for All CPU : 0.000375 kWh
[codecarbon INFO @ 11:44:40] 0.000417 kWh of electricity used since the beginning.
[codecarbon INFO @ 11:44:55] Energy consumed for RAM : 0.000083 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:55] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:55] Energy consumed for All CPU : 0.000751 kWh
[codecarbon INFO @ 11:44:55] 0.000834 kWh of electricity used since the beginning.
[codecarbon INFO @ 11:45:06] Energy consumed for RAM : 0.000114 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:45:06] Delta energy consumed for CPU with constant : 0.000273 kWh, power : 90.0 W
[codecarbon INFO @ 11:45:06] Energy consumed for All CPU : 0.001023 kWh
[codecarbon INFO @ 11:45:06] 0.001137 kWh of electricity used since the beginning.

[codecarbon INFO @ 11:44:40] Energy consumed for RAM : 0.000042 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:40] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:40] Energy consumed for All CPU : 0.000375 kWh
[codecarbon INFO @ 11:44:40] 0.000417 kWh of electricity used since the beginning.
[codecarbon INFO @ 11:44:55] Energy consumed for RAM : 0.000083 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:55] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:55] Energy consumed for All CPU : 0.000751 kWh
[codecarbon INFO @ 11:44:55] 0.000834 kWh of electricity used since the beginning.
[codecarbon INFO @ 11:45:06] Energy consumed for RAM : 0.000114 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:45:06] Delta energy consumed for CPU with constant : 0.000273 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:40] Energy consumed for RAM : 0.000042 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:40] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:40] Energy consumed for All CPU : 0.000375 kWh
[codecarbon INFO @ 11:44:40] 0.000417 kWh of electricity used since the beginning.
[codecarbon INFO @ 11:44:55] Energy consumed for RAM : 0.000083 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:55] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:55] Energy consumed for All CPU : 0.000751 kWh
[codecarbon INFO @ 11:44:40] Energy consumed for RAM : 0.000042 kWh. RAM Power : 10.0 W
[codecarbon INFO @ 11:44:40] Delta energy consumed for CPU with constant : 0.000375 kWh, power : 90.0 W
[codecarbon INFO @ 11:44:40] Energy consumed for All CPU : 0.000375 kWh
```

Fig 4.19: Terminal Screenshot for codecarbon_web_tool.py

```
PS C:\Users\utkar\OneDrive\Desktop\PROJECT testing> streamlit run tools\carbontracker_web_tool.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8503
  Network URL: http://10.2.0.2:8503

[NbConvertApp] Converting notebook 2448370_Utkarsh_Misra_Lab01_DLNN_ANN.ipynb to python
[NbConvertApp] Writing 6897 bytes to 2448370_Utkarsh_Misra_Lab01_DLNN_ANN.py
2025-09-04 11:45:16.147431: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results
 due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=
0`.
2025-09-04 11:45:18.193099: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results
 due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=
0`.
2025-09-04 11:45:23.776291: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instruc
tions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flag
s.
2025-09-04 11:45:56.731 `label` got an empty value. This is discouraged for accessibility reasons and may be disallowed in the future by raising
an exception. Please provide a non-empty label and hide it with label_visibility if needed.
Stack (most recent call last):
  File "C:\Program Files\Python312\Lib\threading.py", line 1032, in _bootstrap
    self._bootstrap_inner()
  File "C:\Program Files\Python312\Lib\threading.py", line 1075, in _bootstrap_inner
    self.run()
  File "C:\Program Files\Python312\Lib\threading.py", line 1012, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Users\utkar\AppData\Roaming\Python\Python312\site-packages\streamlit\runtime\scriptrunner\script_runner.py", line 378, in _run_script_
thread
    self._run_script(request.rerun_data)
  File "C:\Users\utkar\AppData\Roaming\Python\Python312\site-packages\streamlit\runtime\scriptrunner\script_runner.py", line 685, in _run_script
    ) = exec_func_with_error_handling(code_to_exec, ctx)
  File "C:\Users\utkar\AppData\Roaming\Python\Python312\site-packages\streamlit\runtime\scriptrunner\exec_code.py", line 128, in exec_func_with_e
rror_handling
    result = func()
  File "C:\Users\utkar\AppData\Roaming\Python\Python312\site-packages\streamlit\runtime\scriptrunner\script_runner.py", line 669, in code_to_exec
    exec(code, module.__dict__)  # noqa: S102
  File "C:\Users\utkar\OneDrive\Desktop\PROJECT testing\tools\carbontracker_web_tool.py", line 323, in <module>
    st.text_area("", result_output, height=250)
```

Fig 4.20: Terminal Screenshot for carbontracker_web_tool.py

```
PS C:\Users\utkar\OneDrive\Desktop\PROJECT testing> streamlit run tools\comparison_page.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8504
  Network URL: http://10.2.0.2:8504

PS C:\Users\utkar\OneDrive\Desktop\PROJECT testing> streamlit run tools\comparison_page.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8504
  Network URL: http://10.2.0.2:8504
```

Fig 4.21: Terminal Screenshot for comparison_page.py

```
tools > eco2ai_web_tool.py > ...
st.markdown(
    """
    <style>
    /* === Background with stronger dark overlay === */
    .stApp {
        background: url('https://www.thedigitalspeaker.com/content/images/2023/02/Sustainable-AI-greener-future.jpg') center/cover fixed no-repeat;
        position: relative;
        font-family: "Inter", system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", Arial, sans-serif;
        color: #c7d9b9;
        padding-top: 0;
        margin: 0;
        min-height: 100vh;
    }
    .stApp::before {
        content: "";
        position: fixed;
        inset: 0;
        background: rgba(20, 30, 15, 0.75);  /* Darker greenish overlay */
        backdrop-filter: blur(6px);
        pointer-events: none;
        z-index: 0;
    }
    /* ensure main content sits above overlay */
    [data-testid="stAppViewContainer"] > .main, .block-container {
        position: relative;
        z-index: 1;
        padding-top: 1rem;
    }

    /* === Header/title === */
    .header-wrapper {
        max-width: 1000px;
        margin: 0 auto 0.5rem;
        display: flex;
        align-items: center;
        gap: 16px;
```

Fig 4.22: Common Styling for all three tools

```
        font-size: 2.4rem;
        color: #a0d468;  /* lighter green */
        line-height: 1.05;
        text-shadow: 0 0 6px #335522aa;
    }
    .title-block .tagline {
        font-size: 1rem;
        font-style: italic;
        color: #8cbf40;
        margin-top: 4px;
        text-shadow: 0 0 4px #233311cc;
    }

    /* === Tracker card === */
    .tracker-card {
        max-width: 1000px;
        margin: 1.75rem auto 2rem;
        display: flex;
        flex-wrap: wrap;
        border-radius: 16px;
        overflow: hidden;
        box-shadow: 0 40px 90px rgba(0, 40, 0, 0.9);
        background: linear-gradient(145deg, #223322 0%, #1b2b1b 100%);
        border: 2px solid #4a7a33;
    }
    .tracker-left, .tracker-right {
        padding: 2.5rem 2rem;
        flex: 1;
        min-width: 320px;
        box-sizing: border-box;
        color: #a9d18e;
    }
    .tracker-left {
        background: linear-gradient(135deg,#2e4d20 0%,#3b6e28 100%);
        display: flex;
        flex-direction: column;
        justify-content: center;
```

Fig 4.23: Common Styling for all three tools

```
        padding: 2.5rem 2rem;
        flex: 1;
        min-width: 320px;
        box-sizing: border-box;
        color: #a9d18e;
    }
    .tracker-left {
        background: linear-gradient(135deg,#2e4d20 0%,#3b6e28 100%);
        display: flex;
        flex-direction: column;
        justify-content: center;
        gap: 0.75rem;
        border-right: 2px solid #5aa236;
    }
    .tracker-left h1 {
        margin: 0;
        font-size: 2rem;
        font-weight: 700;
        color: #c0f06f;
        text-shadow: 0 0 8px #9ecc43cc;
    }
    .tracker-left .sub {
        font-size: 1rem;
        font-style: italic;
        color: #8bc34a;
        line-height: 1.4;
        margin: 0;
    }

    .title-block h1 {
    margin: 0;
    font-size: 2.4rem;
    color: white;  /* Changed from #1f4d0f */
    line-height: 1.05;
}
.title-block .tagline {
    font-size: 1rem;
```

Fig 4.24: Common Styling for all three tools

CHAPTER 5: CONCLUSION

This project set out to address a significant gap in the rapidly evolving field of sustainable Artificial Intelligence: the absence of a single, accessible platform for comparing the carbon footprint of machine learning code across different measurement tools. The successful development of the Green AI Comparometer demonstrates a practical and effective solution to this challenge. By integrating three distinct tracking libraries—eco2AI, CodeCarbon, and CarbonTracker—into a unified web-based interface, this work provides researchers and developers with a powerful tool to foster environmental awareness and promote computational efficiency. The journey from concept to implementation involved navigating several technical challenges, revealing key advantages of this consolidated approach, and illuminating a clear path for future enhancements.

5.1 Design and Implementation Issues

The development process, while ultimately successful, required overcoming several technical hurdles inherent in creating a platform that executes arbitrary user code in a controlled and monitored environment.

- **Diverse Output Parsing:** A primary challenge was handling the heterogeneous output formats of the integrated tracking libraries. Each tool presents its findings differently, ranging from structured JSON files to direct console logs. This necessitated the development of a flexible and robust data extraction layer capable of parsing these varied outputs to standardize key metrics such as energy consumption (kWh), carbon emissions (CO2eq), and execution duration.
- **Environment and File Management:** Executing user-uploaded scripts within the Streamlit application context introduced complexities related to file access and lifecycle management. Ensuring that scripts could seamlessly locate their datasets and that all temporary files were securely and reliably deleted after execution—even if the script encountered an error—was critical. This was effectively mitigated by implementing comprehensive try-except-finally blocks and a disciplined approach to managing temporary directories, guaranteeing system stability and protecting user data.
- **Configuration and Dependencies:** Managing the configurations for each tracking tool, such as handling environment variables for API keys or specifying hardware locations, within a single application required careful architectural planning. An abstraction layer was developed to manage these configurations, ensuring that the tools could be initialized and run correctly without conflicts.

5.2 Advantages and Limitations

The resulting platform offers significant benefits to the AI community but, like any project, also has limitations that must be acknowledged.

- **Advantages:**
    - **Enhanced Usability and Accessibility:** The most significant advantage of the Green AI Comparometer is its simplicity. It abstracts away the complex setup and configuration of multiple tools, lowering the barrier to entry for developers who wish to assess the environmental impact of their work.
    - **Improved Reproducibility and Fair Comparison:** By executing the user's code in a single, consistent environment while monitoring it with all three tools simultaneously, the platform provides a true "apples-to-apples" comparison. This standardized approach is crucial for reproducible research and helps users understand the methodological differences between the trackers.
    - **Automated Workspace Management:** The system's ability to automatically handle the creation and cleanup of temporary files and directories for user scripts and datasets streamlines the user experience and ensures a clean operational environment.
- **Limitations:**
    - **Dependency on External Services:** The accuracy of some tracking tools, particularly CodeCarbon, relies on external APIs to fetch real-time data on the carbon intensity of the local power grid. This introduces a dependency that is outside the control of the platform and could be a potential point of failure.
    - **Inherent Measurement Variability:** The platform intentionally highlights that there is no single "true" value for carbon emissions. The estimates provided can vary based on the specific library's algorithm and the significant fluctuations in the carbon intensity of regional power grids. This variability is not a flaw of the platform itself but reflects the current state of carbon accounting in the field.

5.3 Future Enhancements

The Green AI Comparometer serves as a robust foundation for a more comprehensive suite of Green AI tools. Several avenues for future work have been identified to extend its impact and utility:

- **Expanded Tool Integration:** The modular design of the platform allows for the integration of additional carbon tracking tools as they emerge, providing users with an even broader comparative perspective.
- **User Authentication and Result Persistence:** Implementing a user account system would be a major step forward, allowing users to save their analysis history, track the impact of their optimizations over time, and manage their projects within a personal dashboard.

- **Advanced Visualization and Analytics:** Future versions could include richer dashboards with time-series graphs of power consumption during a script's execution or geographical maps visualizing the emission differences when running code in various cloud data center regions.
- **Real-time Feedback and Recommendations:** A long-term goal is to evolve the platform from a post-analysis tool to a real-time feedback system. This could involve developing plugins for IDEs or providing data-driven recommendations for code optimization based on the analysis of a user's script, guiding them toward more energy-efficient libraries and algorithms.

References

1. A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, “Quantifying the carbon emissions of machine learning,” *arXiv.org*, Oct. 21, 2019. https://arxiv.org/abs/1910.09700
2. P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, “Towards the systematic reporting of the energy and carbon footprints of machine learning,” *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, Jan. 2020, [Online]. Available: https://jmlr.org/papers/volume21/20-312/20-312.pdf
3. L. F. W. Anthony, B. Kanding, and R. Selvan, “CarbonTracker: Tracking and Predicting the carbon footprint of training deep learning models,” *arXiv.org*, Jul. 06, 2020. https://arxiv.org/abs/2007.03051
4. Y. Meng and H. Noman, “Predicting CO2 Emission Footprint Using AI through Machine Learning,” *Atmosphere*, vol. 13, no. 11, p. 1871, Nov. 2022, doi: 10.3390/atmos13111871.
5. Z. Yao *et al.*, “Machine learning for a sustainable energy future,” *Nature Reviews Materials*, vol. 8, no. 3, pp. 202–215, Oct. 2022, doi: 10.1038/s41578-022-00490-5.
6. S. A. Budennyy *et al.*, “ECO2AI: Carbon Emissions Tracking of Machine Learning models as the first step towards Sustainable AI,” *Doklady Mathematics*, vol. 106, no. S1, pp. S118–S128, Dec. 2022, doi: 10.1134/s1064562422060230.
7. C.-J. Wu *et al.*, “Sustainable AI: environmental implications, challenges and opportunities,” Apr. 22, 2022. https://proceedings.mlsys.org/paper_files/paper/2022/hash/462211f67c7d858f663355eff93b745e-Abstract.html
8. P. Pathania *et al.*, ‘Calculating Software’s Energy Use and Carbon Emissions: A Survey of the State of Art, Challenges, and the Way Ahead’, *arXiv [cs.SE]*. 2025.
9. M. M. I. Jabed, “Sustainable AI: Innovations for Energy-Efficient Machine Learning Models,” *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 6, pp. 709–720, Jun. 2023, doi: 10.17762/ijritcc.v11i6.11358.
10. A. H. Ali, “Green AI for Sustainability: Leveraging Machine learning to drive a circular economy,” *Deleted Journal*, vol. 2023, pp. 15–16, Apr. 2023, doi: 10.58496/bjai/2023/004.
11. N. E. Benti, M. D. Chaka, and A. G. Semie, “Forecasting Renewable Energy Generation with Machine Learning and Deep Learning: Current Advances and Future Prospects,” *Sustainability*, vol. 15, no. 9, p. 7087, Apr. 2023, doi: 10.3390/su15097087.
12. G. H. Gu, J. Noh, I. Kim, and Y. Jung, “Machine learning for renewable energy materials,” *Journal of Materials Chemistry A*, vol. 7, no. 29, pp. 17096–17117, Jan. 2019, doi: 10.1039/c9ta02356a.
13. H. S. A. Nuaimi, A. Acquaye, and A. Mayyas, “Machine learning applications for carbon emission estimation,” *Resources Conservation & Recycling Advances*, vol. 27, p. 200263, Jun. 2025, doi: 10.1016/j.rcradv.2025.200263.

14. A.L. Ligozat and S. Luccioni, “A Practical Guide to Quantifying Carbon Emissions for Machine Learning researchers and practitioners,” Jul. 15, 2021. https://hal.science/hal-03376391/
15. L. Gaur, A. Afaq, G. K. Arora, and N. Khan, “Artificial intelligence for carbon emissions using system of systems theory,” *Ecological Informatics*, vol. 76, p. 102165, Jun. 2023, doi: 10.1016/j.ecoinf.2023.102165.
16. M. Tiutiulnikov, V. Lazarev, A. Korovin, N. Zakharenko, I. Doroshchenko, and S. Budennyy, “eco4cast: Bridging Predictive Scheduling and Cloud Computing for Reduction of Carbon Emissions for ML Models Training,” *Doklady Mathematics*, vol. 108, no. S2, pp. S443–S455, Dec. 2023, doi: 10.1134/s1064562423701223.
17. L. Wei, Z. Ma, C. Yang, and Q. Yao, “Advances in the Neural Network Quantization: A Comprehensive review,” *Applied Sciences*, vol. 14, no. 17, p. 7445, Aug. 2024, doi: 10.3390/app14177445.
18. Á. D. Reguero, S. Martínez-Fernández, and R. Verdecchia, “Energy-efficient neural network training through runtime layer freezing, model quantization, and early stopping,” *Computer Standards & Interfaces*, vol. 92, p. 103906, Aug. 2024, doi: 10.1016/j.csi.2024.103906.
19. A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of Quantization Methods for Efficient Neural network Inference,” *arXiv (Cornell University)*, Jan. 2021, doi: 10.48550/arxiv.2103.13630.
20. M. Kim, W. Saad, M. Mozaffari, and M. Debbah, “Green, Quantized Federated Learning over Wireless Networks: an Energy-Efficient Design,” *IEEE Transactions on Wireless Communications*, vol. 23, no. 2, pp. 1386–1402, Jun. 2023, doi: 10.1109/twc.2023.3289177.
21. N. Alizadeh and F. Castor, ‘Green AI: a Preliminary Empirical Study on Energy Consumption in DL Models Across Different Runtime Infrastructures’, in *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, Lisbon, Portugal, 2024, pp. 134–139.
22. S. Mittal and S. Vaishay, ‘A Survey of Techniques for Optimizing Deep Learning on GPUs’, *Journal of Systems Architecture*, vol. 99, 08 2019.
23. “Carbon Emission Quantification of Machine Learning: A review,” *IEEE Journals & Magazine | IEEE Xplore*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=11030805
24. S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, “Learned step size quantization,” *arXiv (Cornell University)*, Jan. 2019, doi: 10.48550/arxiv.1902.08153.
25. E. Paula, J. Soni, H. Upadhyay, and L. Lagos, “Comparative analysis of model compression techniques for achieving carbon efficient AI,” *Scientific Reports*, vol. 15, no. 1, Jul. 2025, doi: 10.1038/s41598-025-07821-w.

26. J. Tmamna *et al.*, “Pruning deep Neural Networks for Green Energy-Efficient Models: A survey,” *Cognitive Computation*, vol. 16, no. 6, pp. 2931–2952, Jul. 2024, doi: 10.1007/s12559-024-10313-0.
27. D. Castellanos-Nieves and L. García-Forte, “Improving Automated Machine-Learning Systems through Green AI,” *Applied Sciences*, vol. 13, no. 20, p. 11583, Oct. 2023, doi: 10.3390/app132011583.
28. K. Jegadeeswari and R. Rathipriya, “Green AI practices in multi-objective hyperparameter optimization for sustainable machine learning,” *International Journal of Information Technology and Computer Science*, vol. 17, no. 2, pp. 1–9, Apr. 2025, doi: 10.5815/ijitcs.2025.02.01.
29. C. Clemm, L. Stobbe, K. Wimalawarne, and J. Druschke, ‘Towards Green AI: Current Status and Future Research’, in *2024 Electronics Goes Green 2024+ (EGG)*, 2024, pp. 1–11.
30. H. Elmousalami, F. K. P. Hui, and A. A. Alnaser, “Enhancing Smart and Zero-Carbon Cities Through a Hybrid CNN-LSTM Algorithm for Sustainable AI-Driven Solar Power Forecasting (SAI-SPF),” *Buildings*, vol. 15, no. 15, p. 2785, Aug. 2025, doi: 10.3390/buildings15152785.
31. B. Rokh, A. Azarpeyvand, and A. Khanteymoori, ‘A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification’, *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, Nov. 2023.
32. L. Wang, Q. Dou, P. T. Fletcher, S. Speidel, and S. Li, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. 2022. doi: 10.1007/978-3-031-16449-1.