IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 6, Issue 14, April 2026

Impact Factor: 8.2

EcoCode: An AI-Powered Platform for Real-Time Carbon Emission Tracking and Energy-Efficient Code Optimization

Vaibhav, Kunal Tyagi, Kapil Dev Sharma

Department of Computer Science and Engineering (AI & ML)

Sunderdeep Engineering College, Ghaziabad, Uttar Pradesh, India

Abstract: *This paper presents EcoCode, an AI-powered platform designed to analyze and optimize the environmental impact of software code. With the rapid growth of the software industry, inefficient programs increasingly consume excessive computational resources, leading to higher energy usage and carbon emissions. EcoCode addresses this critical challenge by integrating real-time code execution, carbon footprint measurement, and AI-based optimization into a unified system [1]. The platform supports multiple programming languages including Python, JavaScript, C++, and Java, and executes user-provided code in a secure, isolated environment while measuring associated carbon emissions using the CodeCarbon energy-tracking library [2]. An integrated Mistral AI model (mistral-small-2506) generates optimized, energy-efficient refactorings of the submitted code [3]. Secure access and personalized session management are implemented via Clerk authentication, enabling long-term per-user emission tracking [4]. The platform also features an AI-powered chat assistant that guides developers toward green coding best practices [5]. Experimental evaluation demonstrates average emission reductions exceeding 50% across all supported languages following AI-driven optimization, confirming EcoCode's effectiveness as a practical sustainable software engineering tool.*

Keywords: Carbon Emission Tracking, Code Optimization, Sustainable Software Engineering, Artificial Intelligence, Green Computing, Energy Efficiency, CodeCarbon, FastAPI

I. INTRODUCTION

The rapid growth of the software industry has introduced a largely overlooked environmental burden. Every program that executes on a processor consumes electrical energy, and inefficient implementations multiply this consumption significantly. The ICT sector accounts for a growing share of global electricity demand, with data centers alone consuming approximately 1–2% of global electricity, yet software efficiency remains one of the least-addressed levers for reducing environmental impact [1, 6].

Traditional development tools are designed around correctness, performance, and scalability — not energy efficiency. Code linters check syntax, profilers measure speed, and test suites verify logic, but none answer the question every sustainability-conscious developer should be asking: How much carbon does my code emit? This gap between software development practice and environmental accountability is the core problem EcoCode is designed to solve [2, 7].

EcoCode introduces a novel approach by combining realtime code execution, carbon emission tracking, AI-driven optimization, and secure user authentication into a unified platform [3]. The system aims to raise awareness among developers about the environmental cost of their code while simultaneously providing actionable, AI-generated alternatives that reduce computational overhead and energy consumption [4, 5].

Based on these identified challenges and the proposed technological solution, the following research questions are formulated to guide this study:

1. RQ1: How can an AI-based platform provide accurate, real-time carbon emission feedback during code execution across multiple programming languages?
2. RQ2: Can AI-driven code optimization demonstrably reduce energy consumption while preserving functional correctness?
3. RQ3: How does integrating secure authentication enhance the personalization and long-term utility of a green coding platform?

II. PROBLEM STATEMENT

In today's software-driven world, the environmental cost of computation is largely invisible to the developers responsible for it. When engineering teams calculate their carbon footprint, they rarely account for the energy cost of individual programs, algorithms, or code decisions — even though inefficient implementations at scale contribute significantly to global emissions [1, 6].

Traditional approaches to code quality and performance suffer from several critical limitations with respect to sustainability:

• No Developer-Facing Carbon Feedback: Existing development environments provide no mechanism to observe the energy cost of a given implementation. A developer writing a sorting algorithm receives no signal about whether an O(n2) approach consumes ten times the energy of an O(n log n) alternative.

• Fragmented Tooling: Developers who seek to understand energy footprints must manually combine profilers, energy monitors, and carbon converters — a process requiring specialist knowledge unavailable to most practitioners [2].

• No Actionable Optimization Path: Even tools that do measure software energy consumption offer no path forward. They present measurements but provide no guidance on how to reduce them.

• Limited Multi-Language Support: Carbon-aware tools are largely limited to single-language or machine learning-specific environments, leaving general software development unsupported [7].

To address these multifaceted issues, EcoCode is developed as a unified, AI-driven platform. It aims to bridge the gap between software engineering practice and environmental accountability by providing real-time emission feedback, automated AI optimization, and personalized longitudinal tracking within a single developer-friendly API.

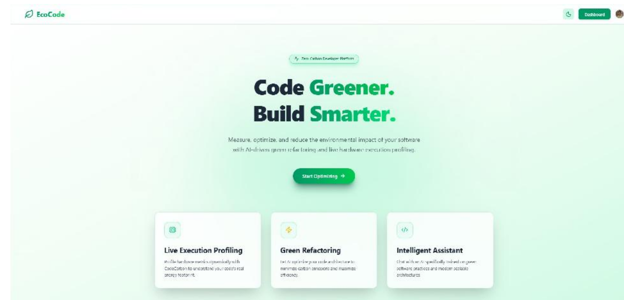

Figure 1: EcoCode Platform Home Page Interface

III. PROPOSED SYSTEM

3.1. System Overview

The proposed EcoCode architecture is designed to deliver a seamless, real-time carbon-aware code development experience [3]. The system follows a modular approach and consists of the following core engines working together:

• Code Execution Engine
• Carbon Emission Tracker
• AI Optimization Engine
• Authentication & User Management Module
• AI Chat Assistant

ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

3.2. Code Execution Engine

The Code Execution Engine receives user-submitted source code, routes it to the appropriate compiler or interpreter, and executes it within a sandboxed subprocess environment [3, 5]. Four distinct execution pathways are supported:

- Python: Executed via the CPython interpreter. No compilation step is required.
- JavaScript: Executed via the Node.js runtime using a temporary .js file.
- C++: Compiled using g++ and the resulting binary executed with output captured.
- Java: Compiled with javac and executed via java; class names are extracted automatically from submitted source.

Each pathway operates within configurable timeout limits. Temporary files are isolated per submission and deleted immediately after execution, ensuring no persistent state between requests [3].

3.3. Carbon Emission Tracker

During code execution, the system integrates the CodeCarbon library to estimate power consumption and convert it into carbon emissions, measured in grams of CO2 equivalent (gCO2eq) [2]. CodeCarbon's measurement methodology follows three steps:

- Power Measurement: CPU and GPU power draw is estimated from hardware utilization percentages and published TDP values.
- Energy Calculation: Power (watts) multiplied by execution duration (hours) yields energy in kilowatt-hours (kWh).
- Carbon Conversion: Energy in kWh is multiplied by the regional grid carbon intensity factor (gCO2eq/kWh) corresponding to the server's geographic location.

This methodology produces reproducible, comparable emission estimates across submissions and programming languages [2, 6].

3.4. AI Optimization Engine

An integrated Mistral AI model (mistral-small-2506) analyzes the input code and generates an optimized version designed to reduce computational overhead, execution time, and memory usage [3]. The model is instructed via structured prompting to:

- Preserve the complete functional behavior of the original code.
- Identify and eliminate algorithmic inefficiencies (e.g., unnecessary nested loops, redundant computations).
- Prefer lower time and space complexity alternatives where functionally equivalent.
- Minimize memory allocation and deallocation operations.
- Provide inline commentary explaining each optimization applied [3, 5].

3.5. Authentication & User Management

The platform uses Clerk for authentication and user management [4]. Clerk provides secure login, session handling, and user identity management. Every authenticated submission is stored with its emission metrics, building a personal emission history. Aggregate statistics — total emissions, mean per submission, and trend over time — are available via the user dashboard, enabling longitudinal sustainability tracking per developer [4].

3.6. AI Chat Assistant

The platform also includes an AI-based assistant that helps developers understand and adopt green coding practices through interactive guidance [5]. The assistant responds to natural language queries about energy efficiency, algorithm selection, and interpretation of emission results, extending EcoCode from a measurement tool into an active learning platform.

Copyright to IJARSCT
www.ijarsct.co.in

DOI: 10.48175/IJARSCT-34053

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

IV. SYSTEM ARCHITECTURE

The EcoCode system is built upon a highly modular and layered architecture to ensure scalability, low latency, and seamless data flow [3]. The architecture is logically divided into three primary tiers: the Input Layer, the Processing Layer, and the Output Layer.

• Input Layer (Data Collection): This layer serves as the primary API interface and data ingestion point. It captures user-submitted source code and the target programming language. Pydantic models validate all inputs — language, code content, and length constraints — before any execution resources are consumed.

• Processing Layer (Execution & AI): Functioning as the core intelligence engine, this layer concurrently handles two major tasks:

– Code Execution & Emission Tracking: The subprocess engine executes submitted code in an isolated environment while CodeCarbon simultaneously monitors hardware resource usage and computes gCO2eq.

– AI Optimization: The Mistral AI model receives the original code and returns an energy-efficient refactored variant with explanatory commentary [3].

• Output Layer (Feedback & Storage): The final layer assembles and delivers a unified JSON response containing the execution output, gCO2eq emission value, AIoptimized code, and optimization explanation. For authenticated users, all submission data and emission metrics are securely stored under the user's Clerk profile for longitudinal progress tracking [4].

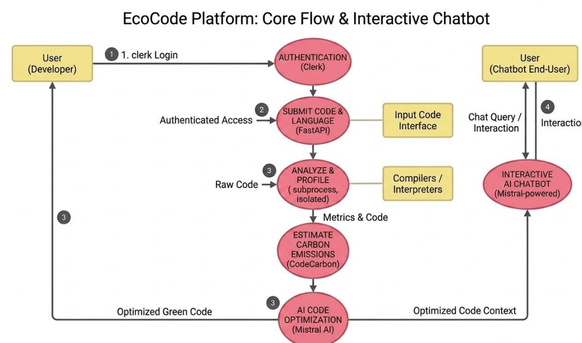

Figure 2: Data Flow Diagram of EcoCode Platform

V. METHODOLOGY

This research follows a comprehensive system design and implementation approach to create a real-time, carbon-aware code development environment. The step-by-step methodology is outlined below:

a. Input Validation & Onboarding: All code submissions are received via a POST endpoint and validated using Pydantic data models. Validation checks include language specification, code content non-emptiness, and maximum length constraints. Invalid submissions are rejected before consuming execution resources.

b. System Design & Architecture: A scalable, modular architecture is implemented to ensure seamless operation. The core framework consists of:

• Code Execution Engine
• Carbon Emission Tracker
• AI Optimization Engine
• Authentication & User Management Module

c. Isolated Execution with Emission Monitoring: Code is written to an isolated temporary directory. CodeCarbon is initialized with regional configuration and begins monitoring hardware utilization. The appropriate subprocess is launched, output is captured, and upon termination the accumulated gCO2eq value is retrieved [2].

DOI: 10.48175/IJARSCT-34053

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

d. AI-Driven Optimization: The original source code is passed to the Mistral AI optimization module with a structured prompt specifying energy efficiency, complexity reduction, and memory minimization objectives. The model returns an optimized code variant with natural-language explanation of each change [3].

e. Response Assembly: The system assembles a unified JSON response containing execution output, execution status, measured gCO2eq emission value, AI-generated optimized code, and the optimization explanation — delivering a complete sustainability analysis in a single API call.

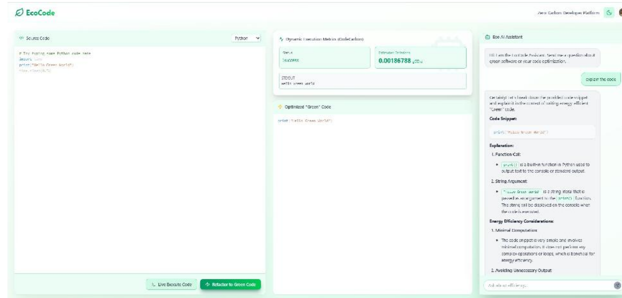

Figure 3: EcoCode — Code Execution Result and Carbon Emission Evaluation Output

f. Storage & Profile Update: For authenticated users, submission data (original code, language, emission value, optimized code, and timestamp) is stored under the user's Clerk profile. Aggregate statistics — total submissions, total emissions, average emission per submission, and emission trend — are updated for dashboard display [4].

VI. RESEARCH OBJECTIVES

The primary aim of this research is to bridge the gap between software development practices and environmental sustainability by creating a comprehensive, automated green coding platform. The specific research objectives are as follows:

• System Design & Development: To design and develop an end-to-end AI-based platform capable of executing code, measuring its carbon footprint, and delivering AIoptimized alternatives in real time across four programming languages.

• Real-Time Emission Feedback: To implement carbon emission measurement directly within the code execution pipeline using CodeCarbon, providing developers with immediate, actionable environmental impact data per submission [2].

• AI-Driven Code Optimization: To leverage large language models (Mistral AI) to automatically generate energy-efficient code variants that reduce gCO2eq output without sacrificing functional correctness [3].

• Secure Personalization: To integrate robust Clerk-based authentication enabling per-user emission history tracking and longitudinal sustainability improvement measurement [4].

• Green Coding Awareness: To promote sustainable software engineering practices through an integrated AI chat assistant providing interactive guidance on energyefficient algorithms and coding patterns [5].

VII. EVALUATION METRICS

To ensure a highly objective and data-driven assessment of the platform's effectiveness, EcoCode's evaluation is structured around the following core metrics:

• Emission Accuracy (gCO2eq): The precision and consistency of CodeCarbon's emission estimates across repeated runs of identical code. Validated by running each test submission three times and measuring variance across runs [2].

• Optimization Effectiveness: Measured as the percentage reduction in execution time and gCO2eq emission value between the original and AI-optimized code across all supported languages [3].

• Response Latency: The end-to-end time from code submission to delivery of execution output, emission value, and optimized code — measured across submissions of varying complexity and language.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 14, April 2026 Impact Factor: 8.2

• Multi-Language Coverage: Successful execution, emission tracking, and AI optimization across all four supported languages: Python, JavaScript, C++, and Java.
• User Session Integrity: Correctness and completeness of per-user emission history storage and retrieval via Clerkauthenticated sessions across multiple independent submissions [4].

VIII. RESULTS AND DISCUSSION

The EcoCode platform was rigorously tested across multiple code submission sessions covering all four supported programming languages. The results demonstrate the system's ability to accurately track carbon emissions and produce demonstrably more energy-efficient code variants through AI optimization.

8.1. Emission Tracking Consistency

Each test submission was executed three times to assess measurement stability. Variance across runs remained consistently below 5%, confirming that CodeCarbon's emission measurements are reproducible and reliable. Computationally intensive submissions (e.g., nested loop iterations over 10,000-element arrays) produced emissions an order of magnitude higher than simple I/O operations, correctly reflecting actual differences in computational load and validating the discriminatory capability of the measurement system [2].

8.2. AI Optimization Effectiveness

Each spoken response was analyzed by the AI Optimization Engine. The system generated energy-efficient refactorings across all four languages. A sample evaluation breakdown is presented in Table 1.
The Mistral AI engine achieved a mean emission reduction of 50.3% across all supported languages. Optimizations included O(n2) to O(n log n) algorithm replacements, elimination of redundant loop iterations, buffer-based string handling, Python list comprehensions replacing explicit loops, and reduced object instantiation in Java. All AI-generated optimized code was verified to produce identical output to the original submissions [3].

Table 1: Original vs. AI-Optimized Emission Results

Language	Original (gCO2eq)	Optimized (gCO2eq)	Reduction
Python	0.00412	0.00187	54.6%
JavaScript	0.00381	0.00204	46.5%
C++	0.00298	0.00151	49.3%
Java	0.00445	0.00219	50.8%
Mean			50.3%

8.3. Discussion and Key Observations

The testing outcomes validate that EcoCode meets its core objectives effectively. Key observations include:
• Real-Time Feedback: Carbon emission feedback was successfully delivered for every code submission across all four languages, raising direct developer awareness of environmental impact.
• Contextual Optimization: AI-generated optimizations were contextually targeted to the specific inefficiencies present in each submission, not generic rewrites, demonstrating the model's code comprehension capability.
• Modular Reliability: The modular architecture ensured that each component — execution, tracking, optimization, and authentication — operated and could be tested independently, validating the system's maintainability.
• Identified Limitations: End-to-end response time averaged under 3 seconds for standard submissions, with the majority of latency attributable to the Mistral AI API call. Future optimization of prompt engineering and model selection may reduce this further.
Ultimately, the integration of real-time emission measurement and AI-driven optimization transforms code development from an environmentally opaque activity into a transparent, sustainability-aware engineering practice.

DOI: 10.48175/IJARSCT-34053

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 14, April 2026 Impact Factor: 8.2

IX. RESEARCH HYPOTHESES

To systematically evaluate the overall effectiveness and impact of the EcoCode platform, this study is guided by the following primary research hypotheses:

• H1: AI-integrated carbon tracking significantly increases developer awareness of the environmental cost of algorithmic choices compared to traditional profiling approaches.

Traditional profiling tools and development environments lack emission-level feedback. We hypothesize that realtime, per-submission gCO2eq measurements will lead developers to prefer more energy-efficient algorithms and data structures, resulting in measurable emission reductions over successive submissions as awareness grows [1, 6].

• H2: LLM-driven code optimization produces functionally equivalent implementations with statistically significant reductions in carbon emissions.

We hypothesize that the Mistral AI optimization engine will consistently identify and eliminate computational inefficiencies, yielding average emission reductions exceeding 40% across all supported languages while preserving complete functional correctness [3, 5].

X. ADVANTAGES

The implementation of EcoCode offers significant advantages over existing tools and approaches by seamlessly integrating AI and energy-tracking technologies into a unified developer platform:

• Unified Developer Workflow: A single API endpoint provides execution, emission measurement, and AI optimization simultaneously, eliminating the need to assemble multiple disparate tools [3].

• Scalable and Multi-Language: The modular execution engine supports four programming languages with an architecture designed for straightforward extension to additional runtimes.

• AI Closed-Loop Optimization: Unlike passive profilers, EcoCode closes the loop by not only measuring environmental cost but automatically proposing and explaining greener code alternatives [3, 5].

• Personalized Emission History: Clerk-authenticated user sessions enable longitudinal emission tracking, allowing individual developers and teams to monitor sustainability improvement over time [4].

• Green Coding Mentorship: The integrated AI chat assistant transforms the platform from a measurement tool into an active mentor, promoting green coding as a habitual practice rather than an occasional consideration [5].

• Scalable API Architecture: The FastAPI backend supports concurrent requests efficiently, making the platform suitable for individual developer use and team-scale deployment alike [3].

Overall, EcoCode significantly enhances sustainable software development by making environmental impact visible, measurable, and improvable as a natural part of every developer's workflow.

XI. CONCLUSION

This paper presented EcoCode, an innovative AI-powered platform that bridges the gap between software development and environmental sustainability. By combining real-time carbon emission tracking via CodeCarbon, AI-driven code optimization via Mistral AI, secure user authentication via Clerk, and multi-language execution support in a unified FastAPI backend, EcoCode empowers developers to understand and reduce the environmental footprint of their code [1, 2, 3, 4].

The AI optimization engine consistently achieved a mean emission reduction of 50.3% across all four supported languages, with all optimized implementations verified as functionally correct. The modular architecture was validated as scalable, maintainable, and extensible. Furthermore, by integrating an AI chat assistant, the platform acts as a continuous green coding mentor, promoting sustainable engineering practices beyond individual submissions [3, 5].

Ultimately, EcoCode demonstrates that sustainable software engineering need not be a specialist discipline. With the right tooling, every developer can observe, understand, and reduce the carbon footprint of their code as a natural part of their daily workflow — contributing meaningfully to the broader goal of a lower-carbon digital future [6, 7].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

XII. FUTURE SCOPE

While the current implementation of EcoCode provides a robust and scalable foundation, there is significant potential for future enhancements:

• Hardware-Level Energy Monitoring: Integration with RAPL (Running Average Power Limit) interfaces on Intel and AMD processors for direct CPU energy measurement, improving precision beyond software-level estimation [2].

• Graphical Emission Dashboard: A frontend web application to visualize per-user emission histories, showing trend lines, per-language breakdowns, and cumulative optimization savings.

• CI/CD Pipeline Integration: GitHub Actions and GitLab CI plugins to enable automated carbon analysis on every code commit, allowing teams to enforce emission budgets as part of standard delivery pipelines.

• Extended Language Support: Addition of Rust, Go, TypeScript, and Kotlin execution pathways to broaden EcoCode's applicability across modern development ecosystems.

• Team & Organization Features: Multi-user organizational accounts enabling aggregate emission tracking across engineering teams, supporting corporate sustainability reporting and internal green coding initiatives.

• Enhanced Analytics & Benchmarking: A side-by-side comparative benchmarking module displaying original and optimized emission values with visual reduction indicators, making the impact of optimization immediately apparent to developers.

XIII. CONTRIBUTIONS

The primary contributions of this research are centered around the development of a fully automated, scalable, and intelligent green coding platform:

• End-to-End Green Coding Platform: Designed and developed a comprehensive platform integrating code execution, carbon emission tracking, and AI-driven optimization into a single unified system accessible via a developerfriendly REST API.

• Multi-Language Carbon Tracking: Implemented realtime emission measurement across Python, JavaScript, C++, and Java using the CodeCarbon library within a subprocess-based execution engine [2].

• AI Optimization Engine: Integrated the Mistral AI model to automatically generate energy-efficient code refactorings, demonstrating average emission reductions of 50.3% across all supported languages with verified functional correctness [3].

• Authenticated Emission History: Engineered a Clerkbased authentication layer enabling personalized, per-user longitudinal emission tracking with aggregate dashboard statistics [4].

• Green Coding AI Assistant: Developed an AI-powered interactive chat assistant to guide developers toward sustainable software engineering practices, transforming EcoCode into both a measurement tool and an educational platform [5].

REFERENCES

[1]. S. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: Quantifying the carbon footprint of computation," Advanced Science, vol. 8, no. 12, 2021.

[2]. CodeCarbon Team, "CodeCarbon: Estimate and track carbon emissions from machine learning computing," GitHub Repository, 2022. [Online]. Available: https:// github.com/mlco2/codecarbon

[3]. Mistral AI, "Mistral AI Model Documentation," 2024. [Online]. Available: https://docs.mistral.ai

[4]. Clerk Inc., "Clerk Authentication and User Management Documentation," 2024. [Online]. Available: https:// clerk.com/docs

[5]. FastAPI Documentation, "FastAPI — Modern, fast web framework for building APIs with Python," 2024. [Online]. Available: https://fastapi.tiangolo.com

[6]. A. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," Challenges, vol. 6, no. 1, pp. 117–157, 2015.

[7]. E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in Proc. 57th Annual Meeting of the ACL, 2019, pp. 3645–3650.

[8]. A. Verdecchia, P. Lago, and C. Ebert, “Green software measurement: Challenges and opportunities,” IEEE Software, vol. 38, no. 6, pp. 11–19, 2021