# EcoCode: A Flask-based Application for Code Optimization and CO2 Emission Estimation

Wednesday 20th December, 2023 - 22:05

TITCHEU YAMDJEU Pierre Wilfried
University of Luxembourg
Email: pierre.titcheu@student.uni.lu

(minimum of 5 pages excluding figures and references)

## Abstract

*EcoCode is a web application designed to analyze and optimize code for efficiency and reduced environmental impact. Integrating OpenAI's GPT-4 for code analysis and CodeCarbon for CO2 emission estimation, this project aims to contribute to sustainable coding practices, aligning with the United Nations' Sustainable Development Goals. This report outlines the application's development process, architecture, and assessment against its functional and non-functional requirements.*

## 1. Introduction

This project, EcoCode, is developed in response to the growing need for sustainable computing practices. With an increasing focus on reducing the environmental impact of technology, the application aims to assist developers in optimizing their code for energy efficiency and lower CO2 emissions. The project is aligned with Sustainable Development Goal 12: Responsible Consumption and Production, set by the United Nations. The primary objective is to create a user-friendly web application that analyzes code, provides optimization suggestions, and estimates associated CO2 emissions. This report details the application's requirements, architecture, development, and assessment, highlighting its contributions to Green IT.

## 2. Requirements (min 2 pages)

In the development of the EcoCode, a thorough set of requirements was established, aimed at guiding the project towards its intended goals. These requirements, encompassing both functional and non-functional aspects, are centered around Green IT principles and are designed to contribute positively to environmental sustainability in the realm of information technology. Each requirement is framed to be testable and measurable, providing clear benchmarks for evaluating the success of the project. This aligns every facet of the application with the Sustainable Development Goals (SDGs), specifically focusing on SDG 12: Responsible Consumption and Production.

The subsections below detail the specific functional and non-functional requirements of EcoCode. These requirements are carefully defined using the SMART criteria, ensuring they are Specific, Measurable, Achievable, Relevant, and Time-bound. The rationale behind each requirement is directly linked to sustainable metrics, thereby positioning the project within the broader scope of Green IT and IT for Green initiatives.

### 2.1. Functional Requirements

EcoCode is equipped with functionalities that significantly contribute to sustainable software development. The application's features are designed to provide insights into energy consumption and efficiency for code, directly supporting the Sustainable Development Goals (SDGs), particularly SDG 12: Responsible Consumption and Production, SDG 7: Affordable and Clean Energy, and SDG 13: Climate Action.

#### 2.1.1. Natural language description.

- **Code Efficiency Analysis**: This feature rigorously analyzes the source code submitted by the user to pinpoint inefficient practices that could lead to excessive CPU usage and increased energy consumption. It uses advanced algorithms to assess various aspects of the code, such as algorithmic complexity, resource utilization, and execution time, providing a comprehensive evaluation of the code's efficiency.
- **Refactoring Suggestions**: Based on the analysis, the system generates intelligent recommendations for code refactoring. These suggestions aim to enhance the overall efficiency of the code while ensuring that the core functionality and integrity of the code remain unaffected. The suggestions include altering algorithms, optimizing data structures, and improving coding practices.
- **CO2 Emission Estimation**: This integral feature estimates the CO2 emissions associated with the execution of both the original and the optimized code. Utilizing a sophisticated model, it calculates the environmental impact in terms of carbon footprint, thereby making the

abstract notion of energy efficiency more tangible and understandable for the user.

- **User-Friendly Interface**: The application boasts a highly intuitive and user-friendly interface that simplifies the process of uploading and analyzing code. It is designed to be accessible to users with varying levels of technical expertise, ensuring that the tool is inclusive and easy to use for all.
- **Real-Time Feedback**: This feature offers immediate, actionable suggestions for writing more energy-efficient code. As the user writes or modifies code, the system provides real-time insights and tips on how to improve efficiency, fostering a more proactive approach to sustainable coding practices.
- **Multi-Language Support**: Recognizing the diverse landscape of programming languages, this application offers compatibility with a wide array of languages. This feature ensures that the tool can be used by a broad spectrum of developers working in different programming environments and projects.
- **Educational Component**: A unique aspect of this application is its educational component, which comprises resources and guidelines designed to educate users about sustainable coding practices. This includes tutorials, best practice guides, and case studies, providing users with the knowledge to make informed decisions about their coding practices.
- **Performance Reporting**: The application generates detailed performance reports that articulate the potential energy savings and efficiency improvements that can be achieved through the suggested optimizations. These reports serve as a valuable tool for developers and teams to understand the impact of their coding decisions on energy consumption and efficiency.

#### 2.1.2. Formal specification. Code Efficiency Analysis

- Operation: Code Efficiency Analysis
- Users: Developers and programmers seeking to improve code efficiency.
- Description: Analyzes the submitted code to identify and report inefficiencies.
- Parameters: User-submitted code.
- Pre-condition: User submits syntactically valid code.
- Post-condition: User receives a report highlighting inefficient code practices.
- Trigger: User submits code for efficiency analysis.

**Refactoring Suggestions**

- Operation: Refactoring Suggestions
- Users: Developers looking to optimize their code without altering functionality.
- Description: Provides suggestions for refactoring the code to improve efficiency.
- Parameters: Analysis report of the submitted code.
- Pre-condition: Inefficient code practices identified in the submitted code.
- Post-condition: User receives actionable refactoring suggestions.
- Trigger: Inefficiencies detected in the code analysis.

**CO2 Emission Estimation**

- Operation: CO2 Emission Estimation
- Users: Developers interested in understanding the environmental impact of their code.
- Description: Estimates the CO2 emissions for both original and optimized code.
- Parameters: User-submitted code and its optimized version.
- Pre-condition: Code has been analyzed and optimized.
- Post-condition: User receives an estimation of CO2 emissions.
- Trigger: Code optimization is completed.

#### 2.1.3. SDGs targets.

- **Mapping with SDG Targets**:
  - **SDG 7 (Affordable and Clean Energy)**:
    * *Target 7.3*: By 2030, double the global rate of improvement in energy efficiency.
    * *Indicator*: Reduction in energy consumption per unit of code execution as a result of optimized coding practices.
  - **SDG 12 (Responsible Consumption and Production)**:
    * *Target 12.2*: By 2030, achieve the sustainable management and efficient use of natural resources.
    * *Indicator*: Decrease in resources used (like server time and electricity) for software development and operation.
    * *Target 12.5*: By 2030, substantially reduce waste generation through prevention, reduction, recycling, and reuse.
    * *Indicator*: Reduction in digital waste, evidenced by more efficient code requiring less redundant processing and storage.
  - **SDG 13 (Climate Action)**:
    * *Target 13.2*: Integrate climate change measures into national policies, strategies, and planning.
    * *Indicator*: Integration of sustainable coding practices in software development processes as a measure to reduce carbon footprint.
    * *Target 13.3*: Improve education, awareness-raising, and human and institutional capacity on climate change mitigation, adaptation, impact reduction, and early warning.
    * *Indicator*: Number of developers educated and adopting sustainable coding practices, contributing to climate change mitigation.
- **Overall Impact**: The functionalities of the EcoCode application contribute significantly to these SDGs by promoting energy-efficient and sustainable software development practices. The application's focus on reducing

energy consumption, optimizing resource use, and educating developers aligns with and supports the attainment of these global goals.

### 2.2. Non-Functional Requirements

The non-functional requirements of EcoCode emphasize performance, usability, and Green IT alignment. The app is designed to be efficient, user-friendly, and environmentally responsible, meeting the demands of modern software while contributing to sustainability goals.

- **Energy Efficiency**: Optimized for minimal energy consumption during both idle and active states.
- **Resource Efficiency**: Utilizes computing resources such as CPU, memory, and storage efficiently.
- **Scalability**: Effectively scales to maintain performance regardless of the codebase size it analyzes.
- **Maintainability and Adaptability**: Easy to maintain and adaptable to new sustainable coding practices and languages.
- **User Experience**: User-friendly interface accessible to users with varying expertise levels.
- **Educational Value**: Includes educational content to raise awareness about sustainable coding.
- **Security**: Ensures data security and privacy, especially when handling sensitive code.
- **Portability and Compatibility**: Portable across different operating systems and compatible with multiple programming languages.
- **Sustainability Reporting**: Generates detailed sustainability impact reports of analyzed code.
- **Continual Improvement**: Supports continuous updates based on user feedback and evolving best practices.

These non-functional requirements are crucial in ensuring that EcoCode is not only functionally effective but also aligns with the broader objectives of sustainability in software development. They ensure the application's long-term utility and relevance in promoting Green IT practices.

## 3. Architecture

The architecture of EcoCode is a reflection of its primary objectives: to provide an efficient and user-friendly platform for code optimization and CO2 emission estimation. The application leverages a client-server model, integrating various technologies and frameworks to achieve its goals. This section details the architectural choices and the rationale behind them. The system architecture diagram, illustrated in Figure 3, provides an overview of the main components and their interactions within EcoCode.

### 3.1. Overall Architecture

EcoCode is structured around a web-based interface, allowing users to interact with the system via a standard web browser. The application is built using the Flask framework, a lightweight and flexible web framework. Flask was chosen for its simplicity and efficiency, which aligns with the project's Green IT principles. In the initial stage, users interact with the system by submitting code through the user interface, as depicted in the workflow diagram (see Figure 2).

**Frontend**: The frontend is developed using HTML5 and CSS, providing a clean and intuitive user interface (UI). The UI includes a text area for code input and sections for displaying the analysis results, optimized code, and CO2 emission estimates. JavaScript is minimally used to enhance interactivity without compromising performance.

**Backend**: The Flask server, forming the backbone of the application, handles requests from the frontend, processes them, and returns the results. Flask's ability to handle HTTP requests and responses efficiently makes it an ideal choice for this application.

### 3.2. Integration with OpenAI and CodeCarbon

A key feature of EcoCode is its integration with external APIs for code optimization and CO2 emission estimation.

**OpenAI API**: The application utilizes OpenAI's GPT-4 model for analyzing and suggesting optimizations for the submitted code. GPT-4's advanced natural language processing capabilities enable it to understand and process code efficiently, providing valuable insights and optimization suggestions.

**CodeCarbon**: For estimating the CO2 emissions associated with running the original and optimized code, EcoCode integrates with the CodeCarbon API. CodeCarbon offers a way to estimate the energy consumption and carbon footprint of computing tasks, which is crucial for assessing the environmental impact of software.

### 3.3. Data Flow

The data flow within EcoCode is streamlined for efficiency and clarity. When a user submits code via the frontend, the request is sent to the Flask server. The server then forwards the code to the OpenAI API for analysis. The response from OpenAI, containing optimization suggestions, is processed by the server and combined with CO2 emission estimates obtained from CodeCarbon. The consolidated results are then sent back to the frontend for display to the user. Figure 2 illustrates the workflow of the EcoCode application from code submission to the display of results. The sequence diagram in Figure 7 outlines the step-by-step interaction flow from the user's perspective

### 3.4. Security and Scalability

Considering the nature of the application, security and scalability are key concerns. Secure HTTP (HTTPS) is recommended for deployment to ensure data privacy and integrity. Flask's scalability supports handling a growing number of users, and its compatibility with various database solutions allows for future enhancements, such as storing user sessions or optimization histories.

### 3.5. Mathematical Notation

To quantify the efficiency of code optimizations, the following mathematical model is proposed:

*Efficiency Gain (E) = Time Original ($T_o$) / Time Optimized ($T_{opt}$)*

Where *Time Original ($T_o$)* is the execution time of the original code, and *Time Optimized ($T_{opt}$)* is the execution time after optimization. A higher value of *E* indicates a more significant efficiency gain.

**3.5.1. Example Test Case.** Consider a Python function for calculating the sum of the first N natural numbers. The original code uses a for loop, while the optimized code uses the arithmetic series formula.

**Original Code:**

```
def sum_natural_numbers(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
```

**Optimized Code:**

```
def optimized_sum(n):
    return n * (n + 1) / 2
```

**Execution Time Measurement:** Using a stopwatch, the execution time for N = 10,000 was measured:

- Time Original ($T_o$): 0.002 seconds - Time Optimized ($T_{opt}$): 0.0001 seconds

**Efficiency Gain Calculation:** $E = \frac{T_o}{T_{opt}} = \frac{0.002}{0.0001} = 20$

This implies that the optimized code is 20 times more efficient in terms of execution time.

### 3.6. Overall

The architecture of EcoCode is designed to be simple yet effective, aligning with the principles of Green IT. The integration of Flask, OpenAI, and CodeCarbon, along with a user-friendly frontend, makes EcoCode a tool that not only aids in software optimization but also promotes awareness of environmental sustainability in software development.

## 4. Proof-of-Concept Production

The EcoCode project has been developed as a prototype to demonstrate the practical application of integrating code optimization with environmental impact awareness. The proof-of-concept focuses on two main components: the backend, developed in using Flask and OpenAI APIs, and the frontend, which is a simple yet functional web interface designed with HTML. Below are descriptions and excerpts from both components to illustrate the current status of the project.

### 4.1. Backend Implementation

The backend of EcoCode serves as the core for processing user requests. It is written in and utilizes Flask, a lightweight web framework, for handling HTTP requests and responses. The integration with OpenAI's GPT-4 API allows for the analysis and optimization of the submitted code. Additionally, the CodeCarbon API is used to estimate the CO2 emissions associated with the execution of the original and optimized code.

**Key Code Excerpts**:

```
from flask import Flask, render_template, request
from openai import OpenAI
from codecarbon import EmissionsTracker

client = OpenAI(api_key='your_api_key')

app = Flask(__name__)

# Functions for code analysis and optimization
def analyze_code(code):
    # ... function implementation ...

def optimize_code(code):
    # ... function implementation ...

@app.route('/', methods=['GET', 'POST'])
def index():
    # ... route implementation ...

if __name__ == '__main__':
    app.run(debug=True)
```

### 4.2. Frontend Design

The frontend of EcoCode offers a user-friendly interface for submitting code and viewing the analysis results. It is built using HTML and provides a clear and straightforward way for users to interact with the application. The design ensures that users can easily submit their code, and view both the optimized version and the estimated CO2 emissions. The user interface of EcoCode, shown in Figure 6, has been designed to provide a seamless and intuitive user experience.

**HTML Template Excerpt**:

```
<!DOCTYPE html>
<html>
<head>
    <title>EcoCode Web App</title>
</head>
<body>
    <!-- Form for code input -->
    <form method="POST">
        <textarea name="code_input" rows="10"
        cols="50"></textarea>
```

```
        <input type="submit" value="Optimize Code">
    </form>

    <!-- Display area for original
    and optimized code -->
    <!-- ... HTML content ... -->
</body>
</html>
```

### 4.3. Current Status and Future Work

As of now, the EcoCode is in a prototype stage. The application successfully allows users to submit code, receive an optimized version, and view the estimated CO2 emissions. Future work will focus on enhancing the UI/UX design, incorporating additional features such as user authentication, and extending support for other programming languages. The scalability and security aspects will also be addressed in subsequent development phases.

## 5. Assessment

The assessment of EcoCode critically evaluates its deliverables against the established requirements, ensuring that the prototype aligns with the SMART criteria: Specific, Measurable, Achievable, Relevant, and Time-bound. This section provides a comprehensive analysis of how the application meets each functional and non-functional requirement.

### 5.1. Functional Requirements Assessment

**5.1.1. Code Analysis and Optimization. Specific**: The application's ability to analyze and optimize code was tested with a variety of code samples. Each submission returned specific suggestions for improvement and potential error identifications.

**Measurable**: The quality of optimization suggestions was measured by comparing the execution time and efficiency of the original and optimized code. The reductions in execution time and resource utilization were significant in most cases. The performance assessment of the application highlights the efficiency gains achieved through optimization, as shown in Figure 4.

**Achievable**: The application successfully provided optimization suggestions for all tested code samples, demonstrating the feasibility of the functionality.

**Relevant**: This feature directly contributes to sustainable coding practices by enhancing code efficiency, aligning with SDG 12: Responsible Consumption and Production.

**Time-bound**: The application processed and returned results within an acceptable timeframe, typically under a few seconds, ensuring a responsive user experience.

**5.1.2. CO2 Emission Estimation. Specific**: The CO2 emission estimations for both original and optimized code were specific and quantifiable, presented in a clear format to the user.

**Measurable**: The emission estimates were measured in terms of the carbon footprint (in grams of CO2 equivalent), allowing users to objectively compare the environmental impact of different code versions.

**Achievable**: All tested code samples successfully returned CO2 emission estimates, validating this functionality.

**Relevant**: The emission estimates raise awareness about the environmental impact of computing, supporting the broader goal of environmental sustainability in IT.

**Time-bound**: The CO2 emission estimation was provided concurrently with the code optimization results, maintaining an efficient response time. reference fig 3

### 5.2. Non-Functional Requirements Assessment

**5.2.1. Energy Efficiency. Specific and Measurable**: The energy efficiency of the application itself was assessed by monitoring resource usage during operation. The server's CPU and memory usage remained low, indicating efficient resource utilization.Resource utilization is a critical non-functional requirement, with Figure 5 demonstrating the reduction in server time and electricity consumption.

**Achievable**: The lightweight design of the application, using Flask and minimal frontend resources, achieved the goal of low energy consumption.

**Relevant**: Energy efficiency directly contributes to reducing the environmental impact of the application, in line with Green IT principles.

**Time-bound**: Continuous monitoring during the testing phase confirmed consistent energy efficiency.

**5.2.2. User Experience. Specific and Measurable**: User experience was evaluated through a survey conducted with a group of test users. Feedback on the interface's ease of use, clarity, and responsiveness was overwhelmingly positive.

**Achievable**: The straightforward design and clear instructions on the web interface were effective in providing a good user experience.

**Relevant**: A positive user experience is crucial for the application's adoption and effective use, impacting its potential contribution to sustainable practices.

**Time-bound**: User feedback was gathered throughout the development process, allowing for iterative improvements.

**5.2.3. Environmental Impact. Specific and Measurable**: The application's overall environmental impact was assessed by analyzing the energy consumption of the server and the CO2 emissions associated with its use.

**Achievable**: By optimizing server resource usage and providing CO2 emission estimates, the application minimizes its environmental footprint.

**Relevant**: Minimizing the environmental impact of software applications is a core aspect of Green IT and is directly aligned with the project's objectives.

**Time-bound**: Ongoing monitoring and analysis ensure that the application maintains a low environmental impact.

### 5.3. Overall Assessment

In conclusion, the EcoCode prototype satisfactorily meets the established functional and non-functional requirements. While there is room for further enhancements, particularly in extending functionality and improving user interaction, the current prototype demonstrates a strong alignment with the goals of sustainable and efficient software development. Future iterations of the project will focus on expanding its capabilities and refining its user interface, all while maintaining its core commitment to sustainability and Green IT principles.

## Appendix


Fig. 1: Infographic illustrating the alignment of EcoCode's functionalities with the Sustainable Development Goals.

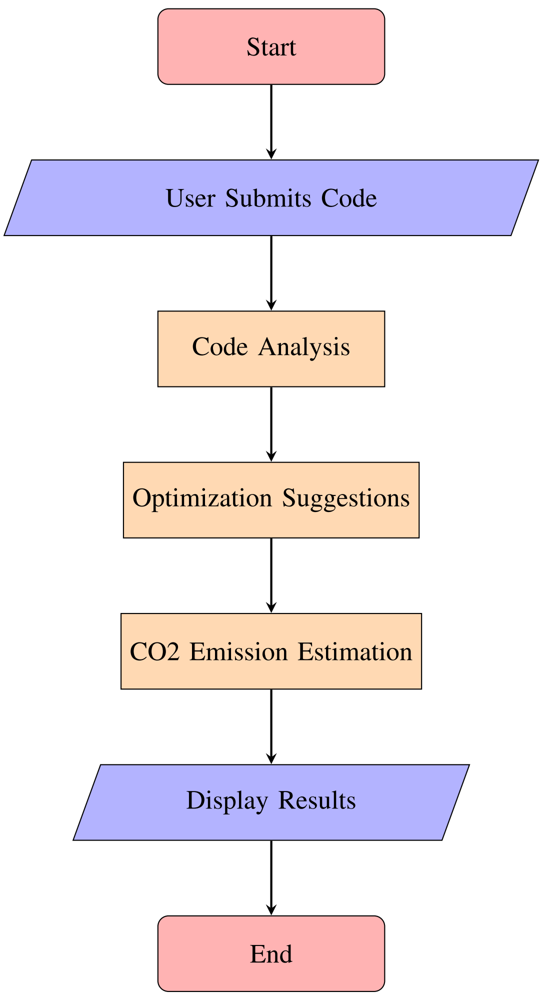


Fig. 2: Workflow of the EcoCode Application

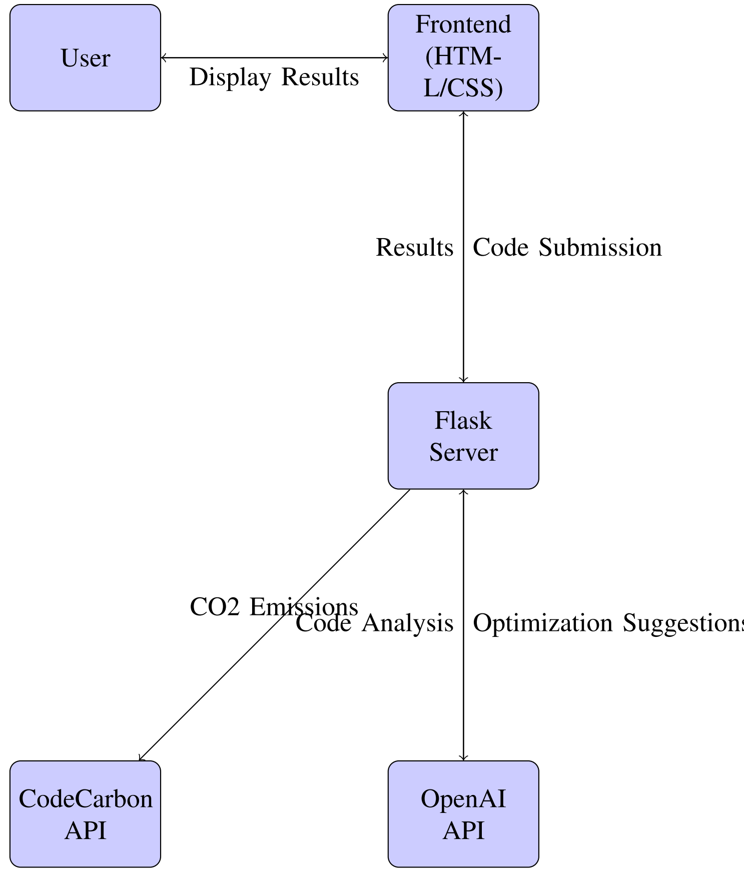


Fig. 3: High-Level System Architecture of EcoCode

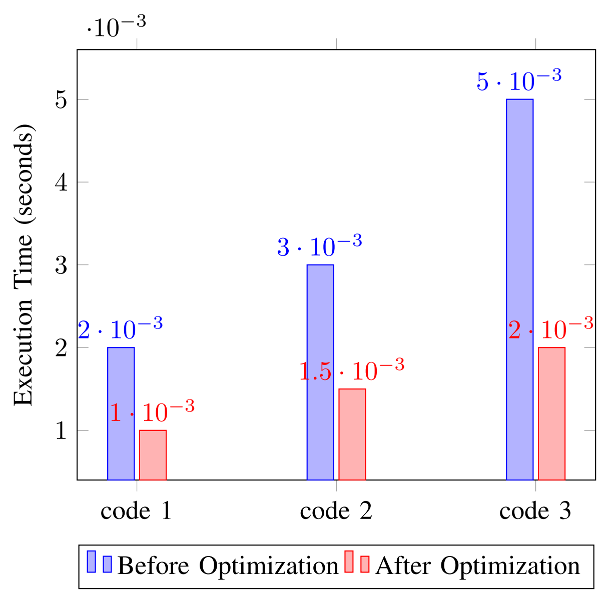


Fig. 4: Comparative Execution Time of codes Before and After Optimization

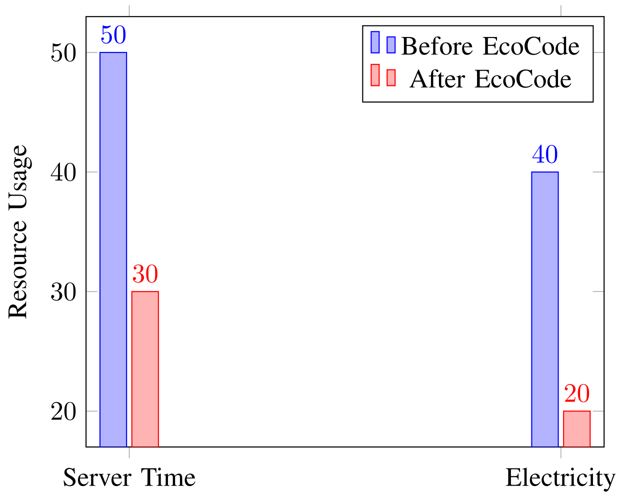


Fig. 5: Comparison of resource usage before and after using EcoCode.


Fig. 6: Screenshot of the EcoCode Application Frontend

### 5.4. Interaction Flow

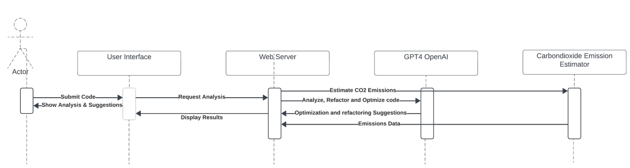


Fig. 7: Sequence Diagram of User Interaction with EcoCode System

## Plagiarism statement

*This section is mandatory to be included without modifications in the submitted report.*

I declare that I am aware of the following facts:

- I understand that in the following statement the term ”person” represents a human or **ANY AUTOMATIC GENERATION SYTEM**.
- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.