GREEN CODE ANALYZER: STATIC AND DYNAMIC PROFILING OF SOFTWARE FOR ENERGY EFFICIENCY AND CARBON FOOTPRINT ESTIMATION

Naisha Khan (22BEC048)
Zoheen Shahzad (22BEC062)
Under the Supervision of Dr. Mainuddin
Department of Electronics & Communication Engineering
F/O Engineering & Technology, Jamia Millia Islamia
New Delhi - 110025

PROBLEM STATEMENT

- Software systems significantly contribute to global energy consumption.
- Focus in sustainability has largely been on hardware (low-power chips, green data centers).
- However, inefficient code (redundant computations, nested loops, poor memory usage) also increases energy demand.
- Existing tools either work at hardware level or are not developer-friendly.

Need:

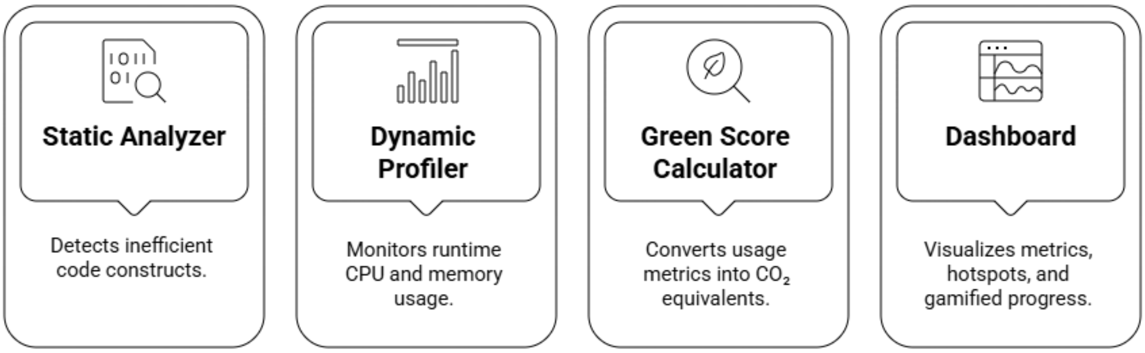
A tool that provides developers with real-time insights on how their code impacts energy and carbon footprint.

LITERATURE REVIEW

- **Couto et al., 2017**: Introduced *Energy-Aware Software Engineering* as a quality attribute.
- **Liqat et al., 2014**: Proposed *EACOF Framework* linking energy counters to software constructs.
- **Laine, 2023**: Built *SonarQube Plugin* for energy-related code smells in Java.
- **Vasconcelos et al., 2025**: Demonstrated up to **30% energy reduction** through refactoring.
- **Lannelongue et al., 2020**: Developed *Green Algorithms* for computing carbon footprint of algorithms.

Gap Identified:
No integrated, developer-centric tool combining **static + dynamic energy profiling** and **CO_2 estimation.**

PROPOSED SYSTEM ARCHITECTURE



Static Analyzer
Detects inefficient code constructs.

Dynamic Profiler
Monitors runtime CPU and memory usage.

Green Score Calculator
Converts usage metrics into CO_2 equivalents.

Dashboard
Visualizes metrics, hotspots, and gamified progress.

Data Flow:
Code → Static Analyzer → Profiler → Green Score Calculator → Dashboard

TOOLS AND TECHNOLOGIES

Category	Tools/Framework
Programming Language	Python
Libraries	psutil, ast, pandas, matplotlib
Frontend	VS Code Extension + Dashboard (Streamlit / Flask)
Version Control	Github

WORK DONE SO FAR

01

Focused on building a static code analyzer as the first module.

02

Analyzed code without running it to detect energy-inefficient patterns.

03

Selected python as programming language. Used AST to inspect code structure.

04

Features Extracted: Loops, nested loops, function calls, recursion, conditionals, I/O operations, data structures, line count.

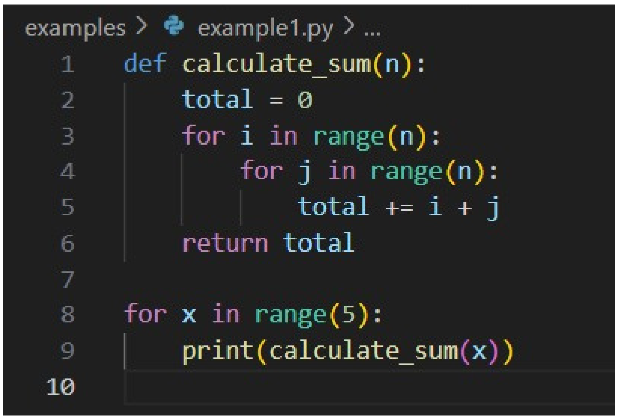
05

Energy Scoring: Heuristic score based on feature weights to indicate green/non-green code.

06

Set up Github Repository and uploaded files.

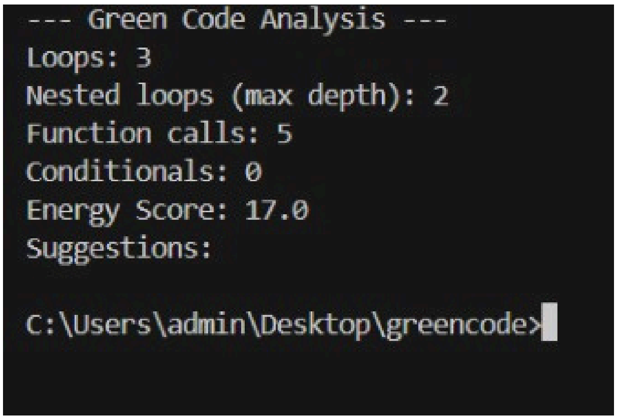
OUTCOME



```python
def calculate_sum(n):
    total = 0
    for i in range(n):
        for j in range(n):
            total += i + j
    return total

for x in range(5):
    print(calculate_sum(x))
```

Input



```
--- Green Code Analysis ---
Loops: 3
Nested loops (max depth): 2
Function calls: 5
Conditionals: 0
Energy Score: 17.0
Suggestions:

C:\Users\admin\Desktop\greencode>
```

Output

REFERENCES

- Couto, M., Cunha, J., & Fernandes, J. (2017). *Energy-aware software engineering: A systematic review.*
- Liqat, U., et al. (2014). *EACOF: Energy Aware Computing Framework.*
- Laine, A. (2023). *Static code smell detection for energy efficiency.*
- Vasconcelos, A., et al. (2025). *Refactoring code smells to reduce energy consumption.*
- Lannelongue, L., et al. (2020). *Green Algorithms: Quantifying the carbon footprint of computation.*
- Loureiro, R., et al. (2025). *Tools for measuring software energy and carbon emissions.*