Green Code Analyzer: Static and Dynamic Profiling of Software for Energy Efficiency and Carbon Footprint Estimation

Minor Project Synopsis submitted in partial fulfilment of the requirements for the Degree of

Bachelor of Technology

Electronics and Communication Engineering

By

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

AUGUST 2025

Literature Review

The relationship between software and energy consumption has received growing attention in the past decade. Traditional sustainability efforts in computing have focused primarily on hardware efficiency (low-power chips, cooling systems, renewable-powered data centers). However, software design and coding practices can also significantly influence energy consumption.

Energy-Aware Software Engineering

- **Couto et al. (2017)** highlighted that *inefficient algorithms, poor data structures, and redundant computations* increase CPU cycles, memory usage, and energy consumption.
- They proposed **energy-aware software engineering** — treating energy efficiency as a quality metric, just like performance or scalability.
- This laid the foundation for what we now call **Green Software Engineering**.

Key Point: Software itself—not just hardware—can make systems significantly greener.

Frameworks for Energy Transparency

- **Liqat et al. (2014)** introduced **EACOF (Energy Aware Computing Framework)**, which links hardware-level energy counters to software constructs (e.g., functions, loops).
- Such frameworks provide transparency but require **complex setup** or **root access**, making them impractical for everyday developers.

Gap: Developers lack **easy-to-use tools** that embed energy insights directly into IDEs and workflows.

Static Code Analysis for Energy Efficiency

- **Cruz et al. (2025)** studied the energy impact of **static analysis tools** themselves (like PMD) and found rule complexity affects energy usage.
- **Laine (2023)** built a **SonarQube plugin for Java** to detect energy code smells in loops. Developers rated the warnings useful (~3.85/5).

Key Insights: Static analysis is scalable. But tools are **language-specific** and lack multi-language + practical developer integration.

Dynamic Profiling and Hotspot Detection

- Static analysis is preventive, but **dynamic profiling** shows *real execution hotspots*.
- **Pinto et al. (2025)** showed that **<10% of functions account for >80% of energy usage** (Pareto principle).
- Tools like **Pypen (2025, ScienceDirect)** can estimate energy at a *function-call level*.

Challenge: Most profilers output **raw data** making developers struggle to interpret or act on it.

Refactoring for Energy Efficiency

- **Vasconcelos et al. (2025)**: Refactoring inefficient patterns (e.g., replacing nested loops with hash lookups, caching expensive calls) reduced energy use by **~29%** in workloads.
- **IBM Research (2023)**: Refactoring enterprise workloads reduced energy by **~13%** and CO_2 emissions by **~5%**, *without hurting performance*.

Lesson: Refactoring has measurable benefits — but we need **automated suggestions** during coding.

Carbon Footprint Estimation Models

- **Lannelongue et al. (2020)** introduced **Green Algorithms** which translates runtime + hardware use into CO_2 equivalents (e.g., "charging 100 smartphones").
- **Loureiro et al. (2025)** surveyed tools and classified them into:
  1. Monitoring tools (hardware meters)
  2. Estimation models (FLOPs, device specs)
  3. Black-box methods (runtime × avg. power)

Takeaway: Hybrid models are needed, accurate and usable by developers.

Research Gap & Opportunity

From the literature:

- Frameworks (Liqat, 2014) exist but are too low-level.
- Static analysis (Laine, 2023; Cruz, 2025) works but is limited.
- Profilers (Pinto, 2025; Pypen) are powerful but not developer-friendly.
- Refactoring studies (Vasconcelos, 2025; IBM, 2023) prove benefits.
- Estimation models (Lannelongue, 2020) contextualize emissions.

But no single tool integrates:

- - Static + dynamic analysis
- - Energy + CO_2 estimation
- - Real-time IDE & CI/CD integration
- - Gamification to influence developer behavior

Opportunity: The **Green Code Analyze**r will bridge this gap by providing a **developer-friendly, end-to-end tool** that makes coding sustainable without adding friction.

Proposed Workplan

WP1 — Static Analysis Core (Weeks 1–2).
- Implement Python code parser to detect inefficient constructs (nested loops, inefficient data structures).
- Link inefficiencies to estimated energy consumption and CO_2 emissions.

WP2 — Dynamic Profiling & Hotspot Detection (Weeks 3–5).
- Integrate runtime profiler using psutil to measure CPU/memory.
- Identify functions with high resource usage.

WP3 — Developer Tooling (Weeks 6–8).
- Create IDE plugin (VS Code) for real-time feedback.
- CI/CD integration to evaluate Green Score on commits.

WP4 — Dashboard & Gamification (Weeks 9–11).
- Visual dashboard for repo-level metrics.
- Introduce Green Score, badges, and historical trends.

WP5 — Evaluation & Reporting (Weeks 12).
- Test on open-source repositories.
- Compare energy estimates before/after refactoring.
- Prepare results and documentation.

Expected Outcomes

The project will deliver a **Green Code Analyzer** that combines both static and dynamic analysis to highlight energy-inefficient code patterns and runtime hotspots. Developers will be able to see not just where inefficiencies exist, but also how they translate into **energy use and CO_2 emissions**.

Key outcomes include:
- A **VS Code plugin** offering real-time feedback and suggestions during coding.
- **CI/CD pipeline** integration with a Green Score metric to track sustainability across commits.
- An **interactive dashboard** showing hotspots, emission equivalents, and gamified badges to encourage improvement.

By applying refactoring suggestions, the analyzer is expected to enable a **10–30% reduction in estimated energy consumption** on benchmark repositories, aligning with prior research findings.

References

- Couto, M., Cunha, J., & Fernandes, J. (2017). Energy-aware software engineering: A systematic review. Information and Software Technology.
- Liqat, U., Kerrison, S., Eder, K., & Hermenegildo, M. (2014). EACOF: Energy Aware Computing Framework. arXiv preprint arXiv:1406.0117.
- Cruz, L., et al. (2025). Energy consumption of static analysis tools: a case study on PMD. Sustainable Software Engineering Course Paper.
- Laine, A. (2023). Static code smell detection for energy efficiency: A SonarQube plugin for Java. LUT University Thesis.
- Vasconcelos, A., et al. (2025). Refactoring code smells to reduce energy consumption: An empirical study. arXiv preprint arXiv:2506.09370.
- IBM Research. (2023). Towards sustainable cloud software systems through energy-aware code smell refactoring.
- Lannelongue, L., Grealey, J., Inouye, M. (2020). Green Algorithms: Quantifying the carbon footprint of computation. arXiv:2007.07610.
- Loureiro, R., et al. (2025). Tools for measuring software energy and carbon emissions: A survey. arXiv preprint arXiv:2506.09683.
- Pinto, G., Castor, F., et al. (2025). Lightweight function-level profiling for energy-aware computing. Conference Proceedings.
- Pypen (2025). Dynamic profiling for energy hotspots in Python. ScienceDirect, Journal of Systems Architecture.