1. Energy-efficient LLMs for software developers

Introduction: LLMs are rapidly becoming part of everyday development; developers use them to generate code in IDEs, summarize it, generate test cases, fix bugs, or support code reviews. These tools are used many times a day, even small energy use per request can add up, making sustainability a growing concern. To reduce energy use, prior research has proposed many efficiency methods to build efficient LLMs. Some tasks may work well with smaller models while other may need bigger models. This means that energy-aware decision are task- dependent. Apart from Model size, there can be other factors that may affect energy consumption in practice. For instance, model compression, model choice, serving optimization (batching, caching), context strategies (RAG, code search + reranking), parameter efficient (LoRA vs. fine-tuning), routing models (smaller first and then bigger if needed), workflow design (when to trigger LLM).

Although many such techniques exist in research, their evidence is scattered and making it hard for developers to know what works for which tasks. Therefore, we explore (1) which SE tasks drive energy demand, (2) which efficiency methods reliably reduce energy, and (3) when trade-offs in memory, runtime, and output quality are acceptable so that we can directly inform how we build "green" developer tooling (reduce environmental impact without reducing maintainability and productivity).

Potential Research questions:

1. Which software engineering tasks are often studied for energy-efficient LLM use?
2. What efficiency techniques have been proposed for reducing energy in developer-facing LLM tools?
3. What evidence exists that these techniques reduce energy consumption (or carbon impact) for different SE tasks?
4. How do other performance factors like memory use (how much RAM/VRAM is needed) and speed (how fast the model runs) affect energy use when developers use LLM tools? (for example, energy changes because the model uses less memory, runs faster, or both)
5. What actionable recommendations can you provide to developers for choosing energy-efficient designs for developer tools?

Related Work:

Paula, E., Soni, J., Upadhyay, H. et al. Comparative analysis of model compression techniques for achieving carbon efficient AI. Sci Rep 15, 23461 (2025). https://doi.org/10.1038/s41598-025-07821-w

Cruz, Luís, João Paulo Fernandes, Maja H. Kirkeby, Silverio Martínez-Fernández, June Sallou, Hina Anwar, Enrique Barba Roque et al. "Greening ai-enabled systems with software engineering: A research agenda for environmentally sustainable ai practices." ACM SIGSOFT Software Engineering Notes 50, no. 3 (2025): 14-23.

Niu, Chenxu, Wei Zhang, Jie Li, Yongjian Zhao, Tongyang Wang, Xi Wang, and Yong Chen. "TokenPowerBench: Benchmarking the Power Consumption of LLM Inference." arXiv preprint arXiv:2512.03024 (2025).

Walkowiak, T., 2025, May. Energy Efficiency in Large Language Models: An Empirical Study. In International Conference on Dependability and Complex Systems (pp. 221-228). Cham: Springer Nature Switzerland.
Ilager, S., Florian Briem, L. and Brandic, I., 2025. Green-code: Optimizing energy efficiency in large language models for code generation. arXiv e-prints, pp.arXiv-2501.

2. Energy patterns for Programming Practices (Python or Java or of your choice)

Introduction: The intensive use of Energy-optimized code does not just provide environmental benefits; it can also help optimize the operational cost. However, various programming languages are efficient for various tasks based on performance and energy efficiency. Previous work has identified various energy patterns specific to programming languages, such as improving loops, input output operation. Rani et al identified the energy patterns specific to web applications.

Terminology for energy patterns: programming operations, programming concepts, programming APIs, energy patterns, code smells

Potential Research questions:

- What programming concepts play a role in writing efficient code (energy efficiency, time efficiency)?
- What kind of empirical research methods (controlled experiment, survey) are used to assess the impact of these coding practices?

- What kind of tools (profilers) are used to assess the impact of these coding practices?
- How do these coding practices impact the code's sustainability (time, memory, energy, and task-specific metrics) ?
- What actionable recommendations can you provide to Python/Java developers?

Related Work:

1. Rani, Pooja, et al. "Energy Patterns for Web: An Exploratory Study." arXiv preprint arXiv:2401.06482 (2024).
2. Pinto et al, Mining Questions about Software Energy Consumption, 2014, https://dl.acm.org/doi/pdf/10.1145/2597073.2597110
3. Cruz, Luis, and Rui Abreu. "Catalog of energy patterns for mobile applications." Empirical Software Engineering 24 (2019): 2209-2235.
4. Reya et al., "GreenPy: Evaluating Application-level Energy Efficiency in Python for Green Computing", 2023
5. Geogiou etal., "What are your programming languages' energy delay implications?"2018.

3. Software-only sustainability guidelines for greener software

Introduction:
Software has become an integral part of scientific research workflows, enabling complex data processing and knowledge extraction. While there are many benefits with these advancements, researchers have highlighted the growing energy footprints of these software systems. It is estimated that the energy footprint of the ICT sector to be up to 5.5% of world carbon emissions and 20% of all electricity and is growing at an alarming rate. Despite the recent awareness towards energy consumption, the focus has been on hardware with little attention on Software.

Aim: Conduct a literature review of software-only practices, principles, and guidelines that aim to reduce the environmental impact of software through engineering decisions within the software lifecycle (requirements, architecture/design, implementation, testing, release engineering, maintenance).

Out of scope: hardware design, data center efficiency, cloud region choice, network/infrastructure optimization, procurement, and organizational facilities.

Goal of the review: Identify what guidance exists that developers can apply directly in software artifacts (code, architecture, runtime configuration, release pipelines), how actionable it is, and where research gaps remain.

Potential Research questions:

- Which software lifecycle phases (requirements, design, coding, testing, CI/CD, maintenance) have the strongest evidence-backed guidance for reducing energy/carbon impact?
- What kinds of software practices are recommended (e.g., algorithmic efficiency, I/O reduction, caching strategies, batching, data structure choices, concurrency patterns, resource-aware design patterns)?
- How do papers operationalize “greener” within software boundaries (energy proxies, runtime metrics, performance-per-watt, cost models, profiling approaches)?
- How actionable are the contributions (principles vs. concrete rules/tools), and what tool support exists (linters, profilers, energy estimation tools, refactoring support)?
- What trade-offs are discussed (maintainability, latency, reliability, security) when applying green software practices?

Related Work:

[1] https://sustainability.uzh.ch/en/policy-reporting/sustainability-policy.html
[2] https://greensoftware.foundation/
[3] Lannelongue, Loïc, Jason Grealey, and Michael Inouye. "*Green algorithms: quantifying the carbon footprint of computation*." Advanced science 8.12 (2021): 2100707.