**Related work**

Search Terms: **AI code generation + energy efficiency + optimization + LLM**

*[* [*https://codecarbon.io/*](https://codecarbon.io/) *can be used to assess carbon footprint]*

|  |  |  |  |
| --- | --- | --- | --- |
| Title | Link/Doi | Abstract | Comments |
| AI-Powered, But Power-Hungry? Energy Efficiency of LLM-Generated Code | <https://doi.org/10.1109/Forge66646.2025.00012> | Large language models (LLMs) are used in software development to assist in various tasks, e.g., code generation and code completion, but empirical evaluations of the quality of the results produced by these models focus on correctness and ignore other relevant aspects, such as their performance and energy efficiency. Studying the performance of LLM-produced programs is essential to understand how well LLMs can support the construction of performance- and energy-critical software, such as operating systems, servers, and mobile applications. This paper presents the first study analyzing the energy efficiency and performance of LLM-generated code for three programming languages Python, Java, and C++, on two platforms, a Mac and a PC, leveraging three frontier LLMs, Github Copilot, GPT-4o, and the recently-released OpenAI o1-mini, and targeting "hard" programming problems from LeetCode. Our results show that the models are much more successful in generating Python and Java than C++ code. Also, LLM-generated code sometimes surpasses an efficient human-written solution, although that is language-dependent and the language with the best results, Python, is the one where application performance and energy consumption tend to matter the least in practice. Furthermore, the performance of generated code is highly correlated across the two platforms, hinting at potential for results to be portable across platforms. |  |
| Adaptive Multi-Agent AI Framework for Real-Time Energy Optimization and Context-Aware Code Review in Software Development | <https://doi.org/10.1109/ISCTIS65944.2025.11066037> | Energy efficiency has become a critical concern in modern software development, particularly for applications deployed in resource-constrained environments such as IoT, mobile devices, and cloud infrastructure. Traditional static analysis tools and single-model code review solutions fail to provide timely, adaptive, and context aware recommendations, leading to suboptimal energy performance and increased technical debt. This paper introduces a novel adaptive multi-agent AI framework that delivers real-time, personalized energy optimization feedback directly within the software development workflow. The proposed system leverages transformer-based code embeddings for semantic analysis, a FAISS-powered contextual memory for historical learning, and a collaborative multi-agent system consisting of specialized AI agents for energy profiling, compliance monitoring, resource allocation, and maintainability assessment. Adaptive reinforcement learning dynamically refines energy recommendations based on developer interactions, ensuring continuous improvement over time. Empirical evaluations demonstrate that the framework effectively reduces redundant feedback, improves system-wide energy efficiency, and enhances developer trust through explainable AI techniques. The results highlight the potential of integrating intelligent, energy-efficient coding practices into modern software engineering workflows, fostering sustainability without compromising performance. |  |
| Learn to Code Sustainably: An Empirical Study on LLM-based Green Code Generation | https://doi.org/10.48550/arXiv.2403.03344 | The increasing use of information technology has led to a significant share of energy consumption and carbon emissions from data centers. These contributions are expected to rise with the growing demand for big data analytics, increasing digitization, and the development of large artificial intelligence (AI) models. The need to address the environmental impact of software development has led to increased interest in green (sustainable) coding and claims that the use of AI models can lead to energy efficiency gains. Here, we provide an empirical study on green code and an overview of green coding practices, as well as metrics used to quantify the sustainability awareness of AI models. In this framework, we evaluate the sustainability of auto-generated code. The auto-generate codes considered in this study are produced by generative commercial AI language models, GitHub Copilot, OpenAI ChatGPT-3, and Amazon CodeWhisperer. Within our methodology, in order to quantify the sustainability awareness of these AI models, we propose a definition of the code's "green capacity", based on certain sustainability metrics. We compare the performance and green capacity of human-generated code and code generated by the three AI language models in response to easy-to-hard problem statements. Our findings shed light on the current capacity of AI models to contribute to sustainable software development. |  |
| Carbon Footprint Evaluation of Code Generation through LLM as a Service | <https://link.springer.com/chapter/10.1007/978-3-658-45010-6_15> | Due to increased computing use, data centers consume and emit a lot of energy and carbon. These contributions are expected to rise as big data analytics, digitization, and large AI models grow and become major components of daily working routines. To reduce the environmental impact of software development, green (sustainable) coding and claims that AI models can improve energy efficiency have grown in popularity. Furthermore, in the automotive industry, where software increasingly governs vehicle performance, safety, and user experience, the principles of green coding and AI-driven efficiency could significantly contribute to reducing the sector’s environmental footprint. We present an overview of green coding and metrics to measure AI model sustainability awareness. This study introduces LLM as a service and uses a generative commercial AI language model, GitHub Copilot, to auto-generate code. Using sustainability metrics to quantify these AI models’ sustainability awareness, we define the code’s embodied and operational carbon. |  |
| Green-Code: Learning to Optimize Energy Efficiency in Llm-Based Code Generation | <https://doi.org/10.1109/CCGRID64434.2025.00068> | Large Language Models (LLMs) are becoming integral to daily life, showcasing their vast potential across various Natural Language Processing (NLP) tasks. Beyond NLP, LLMs are increasingly used in software development tasks, such as code completion, modification, bug fixing, and code translation. Software engineers widely use tools like GitHub Copilot and Amazon Q, streamlining workflows and automating tasks with high accuracy. While the resource and energy intensity of LLM training is often highlighted, inference can be even more resourceintensive over time, as it's a continuous process with a high number of invocations. Therefore, developing resource-efficient alternatives for LLM inference is crucial for sustainability. This work proposes GREEN-CODE, a framework for energy-aware code generation in LLMs. GREEN-CODE performs dynamic early exit during LLM inference. We train a Reinforcement Learning (RL) agent that learns to balance the trade-offs between accuracy, latency, and energy consumption. Our approach is evaluated on two open-source LLMs, Llama 3.2 3B and OPT 2.7 B, using the JavaCorpus and PY150 datasets. Results show that our method reduces the energy consumption between 2350 % on average for code generation tasks without significantly affecting accuracy. |  |
| Large Language Models for Energy-Efficient Code: Emerging Results and Future Directions | <https://doi.org/10.48550/arXiv.2410.09241> | Energy-efficient software helps improve mobile device experiences and reduce the carbon footprint of data centers. However, energy goals are often de-prioritized in order to meet other requirements. We take inspiration from recent work exploring the use of large language models (LLMs) for different software engineering activities. We propose a novel application of LLMs: as code optimizers for energy efficiency. We describe and evaluate a prototype, finding that over 6 small programs our system can improve energy efficiency in 3 of them, up to 2x better than compiler optimizations alone. From our experience, we identify some of the challenges of energy-efficient LLM code optimization and propose a research agenda. |  |
| Generating Energy-efficient code with LLMs | <https://doi.org/10.48550/arXiv.2411.10599> | The increasing electricity demands of personal computers, communication networks, and data centers contribute to higher atmospheric greenhouse gas emissions, which in turn lead to global warming and climate change. Therefore the energy consumption of code must be minimized. Code can be generated by large language models. We look at the influence of prompt modification on the energy consumption of the code generated. We use three different Python code problems of varying difficulty levels. Prompt modification is done by adding the sentence ``Give me an energy-optimized solution for this problem'' or by using two Python coding best practices. The large language models used are CodeLlama-70b, CodeLlama-70b-Instruct, CodeLlama-70b-Python, DeepSeek-Coder-33b-base, and DeepSeek-Coder-33b-instruct. We find a decrease in energy consumption for a specific combination of prompt optimization, LLM, and Python code problem. However, no single optimization prompt consistently decreases energy consumption for the same LLM across the different Python code problems. |  |
| Walking the Tightrope: Balancing Energy Efficiency and Accuract in LLM-Driven Code Generation | <https://studenttheses.uu.nl/handle/20.500.12932/48350> | Large Language Models (LLMs) consume significant amounts of energy during inference, espe cially for computationally expensive tasks like code generation, which leads to environmental con cerns. This work aims to reduce the energy consumption during inference without compromising model performance. The energy consumption of Qwen2.5-Coder-7B-Instruct, Meta-LLaMA 3.1 8B-Instruct, and DeepSeekCoder-V2-Instruct-16B was evaluated on BigCodeBench, a benchmark that consists of 1,140 diverse coding tasks, using a software-based energy measuring approach. The relations between task nature, batch size, model size, fine-tuning, Activation-Aware Weight Quan tization (AWQ), and GPTQ with 8-bit and 4-bit precision were investigated for a variety of models including the Qwen2.5 models. Results indicate that task nature significantly affects energy con sumption across all tested models, while batch size has a minor effect. Notably, the Meta-LLaMA model consumed 130.77% more energy than the DeepSeekCoder model while achieving lower ac curacy. Fine-tuning, AWQ, GPTQ-INT8, andGPTQ-INT4quantizations reducing energy consump tion by up to 19%, 67%, 40%and67%,respectively. GPTQ-INT8models achieved these reductions without significantly reduced accuracy, whereas GPTQ-INT4 models showed slight decreases and AWQshowedsubstantially lower pass@1 scores. This work demonstrates that energy consumption of LLMs can effectively be reduced without significant performance loss, which demonstrates the importance and contributions of innovative research for sustainable AI practices. |  |
| When Faster Isn't Greener: The Hidden Costs of LLM-Based Code Optimization | <https://hal.science/hal-05227453/> | Large Language Models (LLMs) are increasingly adopted to optimize source code, offering the promise of faster, more efficient programs without manual tuning. This capability is particularly appealing in the context of sustainable computing, where enhanced performance is often assumed to correspond to reduced energy consumption. However, LLMs themselves are energy-and resource-intensive, raising critical questions about whether their use for code optimization is energetically justified. Prior work mainly focused on runtime performance gains, leaving a gap in our understanding of the broader energy implications of LLM-based code optimization. In this paper, we report on a systematic, energy-focused evaluation of LLM-based code optimization methods. Relying on 118 tasks from the EvalPerf benchmark, we assess the trade-offs between code performance, correctness, and energy consumption of multiple optimization methods across multiple families of LLMs. We introduce the Break-Even Point (BEP) as a key metric to quantify the number of executions required for an optimized program to outweigh the energy consumed when generating the optimization itself. Our results show that, while certain configurations achieve substantial speedups and energy reductions, these benefits often demand from hundreds to hundreds of thousands of executions to become energetically profitable. Moreover, the optimization process often yields incorrect or less efficient code. Importantly, we identify a weak negative correlation between performance gains and actual energy savings, challenging assumptions that faster code automatically equates to a smaller energy footprint. This work underscores the necessity of energy-aware optimization strategies. Practitioners should carefully target LLM-based optimization efforts to high-frequency, high-impact workloads, while monitoring energy consumption across the entire life-cycle of development and deployment. |  |
| Can We Make Code Green? Understanding Trade-Offs in LLMs vs. Human Code Optimizations | <https://doi.org/10.48550/arXiv.2503.20126> | The rapid technological evolution has accelerated software development for various domains and use cases, contributing to a growing share of global carbon emissions. While recent large language models (LLMs) claim to assist developers in optimizing code for performance and energy efficiency, their efficacy in real-world scenarios remains under exploration. In this work, we explore the effectiveness of LLMs in reducing the environmental footprint of real-world projects, focusing on software written in Matlab-widely used in both academia and industry for scientific and engineering applications. We analyze energy-focused optimization on 400 scripts across 100 top GitHub repositories. We examine potential 2,176 optimizations recommended by leading LLMs, such as GPT-3, GPT-4, Llama, and Mixtral, and a senior Matlab developer, on energy consumption, memory usage, execution time consumption, and code correctness. The developer serves as a real-world baseline for comparing typical human and LLM-generated optimizations.  Mapping these optimizations to 13 high-level themes, we found that LLMs propose a broad spectrum of improvements--beyond energy efficiency--including improving code readability and maintainability, memory management, error handling while the developer overlooked some parallel processing, error handling etc. However, our statistical tests reveal that the energy-focused optimizations unexpectedly negatively impacted memory usage, with no clear benefits regarding execution time or energy consumption. Our qualitative analysis of energy-time trade-offs revealed that some themes, such as vectorization preallocation, were among the common themes shaping these trade-offs. With LLMs becoming ubiquitous in modern software development, our study serves as a call to action: prioritizing the evaluation of common coding practices to identify the green ones. |  |
| An exploration of prompting LLMs to generate energy-efficient code | <https://doi.ieeecomputersociety.org/10.1109/GREENS66463.2025.00010> | The increasing electricity demands of personal computers, communication networks, and data centers contribute to higher atmospheric greenhouse gas emissions, which in turn lead to global warming and climate change. Therefore the energy consumption of code must be minimised. Large language models can generate code, so we study the influence of prompting for energy-efficient code by examining the energy consumption of the generated code. We use three different Python code problems of varying difficulty levels. Prompt modification is done by adding the sentence "Give me an energy-optimised solution for this problem" or by providing two Python coding best practices. The large language models used are Code Llama-70b, Code Llama-70b-Instruct, Code Llama-70b-Python, DeepSeek-Coder-33b-base, and DeepSeek-Coder-33b-instruct. We find a decrease in energy consumption for a specific combination of prompt optimisation, LLM, and Python code problem. However, no single optimisation prompt consistently decreases energy consumption for the same LLM across the different Python code problems. |  |
| Evaluating the Energy-Efficiency of the Code Generated by LLMs | <https://doi.org/10.48550/arXiv.2505.20324> | As the quality of code generated by Large Language Models (LLMs) improves, their adoption in the software industry for automated code generation continues to grow. Researchers primarily focus on enhancing the functional correctness of the generated code while commonly overlooking its energy efficiency and environmental impact. This paper investigates the energy efficiency of the code generated by 20 popular LLMs for 878 programming problems of varying difficulty levels and diverse algorithmic categories selected from the LeetCode platform by comparing them against canonical human-written solutions. Although LLMs can produce functionally correct results in most cases, our findings show that the performance and energy efficiency of LLM-produced solutions are often far below those of human-written solutions. Among the studied LLMs, DeepSeek-v3 and GPT-4o generate the most energy-efficient code, whereas Grok-2 and Gemini-1.5-Pro are among the least energy-efficient models. On average, human-generated canonical solutions are approximately 1.17 times more energy efficient than DeepSeek-v3, 1.21 times more energy efficient than GPT-4o, and over 2 times more energy efficient than Grok-2 and Gemini-1.5-Pro. For specific algorithmic groups such as dynamic programming, backtracking, and bit manipulation, LLM-generated code can consume up to 450 times more energy than human-generated canonical solutions. |  |
| Energy-Aware Code Generation with LLMs: Benchmarking Small vs. Large Language Models for Sustainable AI Programming | <https://doi.org/10.48550/arXiv.2508.08332> | Large Language Models (LLMs) are widely used for code generation. However, commercial models like ChatGPT require significant computing power, which leads to high energy use and carbon emissions. This has raised concerns about their environmental impact. In this study, we evaluate open-source Small Language Models (SLMs) trained explicitly for code generation and compare their performance and energy efficiency against large LLMs and efficient human-written Python code. The goal is to investigate whether SLMs can match the performance of LLMs on certain types of programming problems while producing more energy-efficient code. We evaluate 150 coding problems from LeetCode, evenly distributed across three difficulty levels: easy, medium, and hard. Our comparison includes three small open-source models, StableCode-3B, StarCoderBase-3B, and Qwen2.5-Coder-3B-Instruct, and two large commercial models, GPT-4.0 and DeepSeek-Reasoner. The generated code is evaluated using four key metrics: run-time, memory usage, energy consumption, and correctness. We use human-written solutions as a baseline to assess the quality and efficiency of the model-generated code. Results indicate that LLMs achieve the highest correctness across all difficulty levels, but SLMs are often more energy-efficient when their outputs are correct. In over 52% of the evaluated problems, SLMs consumed the same or less energy than LLMs. |  |
|  |  |  |  |
|  |  |  |  |