Large Language Models for Energy-Efficient Code: Emerging Results and Future Directions

Huiyun Peng, Arjun Gupte,
Nicholas John Eliopoulos, Chien Chou Ho,
Rishi Mantri, Leo Deng*,
Wenxin Jiang, Yung-Hsiang Lu
Department of Electrical
and Computer Engineering
*Department of Computer Science**
Purdue University
West Lafayette, Indiana, USA

Konstantin Läufer,
George K. Thiruvathukal
Department of Computer Science
Loyola University Chicago
Chicago, Illinois, USA

James C. Davis
Department of Electrical
and Computer Engineering
Purdue University
West Lafayette, Indiana, USA

arXiv:2410.09241v1 [cs.SE] 11 Oct 2024

***Abstract*—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.**

***Index Terms*—Energy efficiency, Research agenda, Software optimization, Large language models**

I. Introduction

Energy efficiency has become a critical issue across various domains, from mobile devices to large-scale data centers. On mobile devices such as those running Android, energy efficiency directly impacts battery life and accessibility, making it a key concern for user experience [1]. On a larger scale, data centers contribute significantly to climate change, accounting for 4% of electricity generation in the United States [2] and 3% in the European Union [3]. Improving computing energy efficiency is part of addressing environmental sustainability.

Previous approaches have focused primarily on physical hardware, with limited discussion on improving the underlying software for energy efficiency [4]. Efforts to design energy-efficient programs have introduced energy models [5]–[8], energy measurement tools [9], and energy-aware design patterns [10]–[13]. However, the barrier to adopting these energy-efficient practices remains high, often introducing complexity accessible only to systems experts [14]. Software engineers recognize energy efficiency as a desirable property [15]. However, it often loses out to organizational goals like latency and throughput [4], due to the ineffectiveness of existing methods in meeting the dynamic nonfunctional performance requirements. LLMs like ChatGPT are transforming software engineering by aiding in tasks like debugging and code optimization [16], [17]. Although their potential for energy efficiency has yet to be fully explored, LLMs show promise as powerful aids in optimizing code for energy-aware practices.

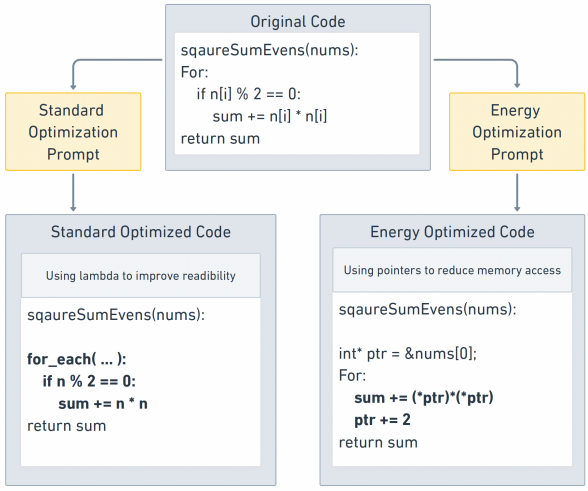

Fig. 1: Standard optimization prompts focus on speed, memory optimization, and code readability. Our proposed energy optimization prompts focus on reducing energy usage. One must then ensure that optimizations *actually* reduce energy use, not just performance metrics such as latency and FLOPS.

In this paper, we explore a novel approach to energy-efficient software development. We ask: *Can LLMs assist developers in optimizing energy efficiency without compromising performance or correctness?* As shown in Figure 1, we propose an automated tool designed to refactor software with a primary focus on optimizing energy efficiency, going beyond conventional performance metrics to directly target energy usage reduction. This tool incorporates energy-aware prompts that are input into a generator LLM alongside the original code to produce more efficient program outputs. The optimized code is then evaluated and refined through Natural Language Feedback from an evaluator LLM, allowing for iterative improvements to the code itself. This approach offers the advantage of being portable — it is easy to set up, and compatible with multiple programming languages without significant modifications. We present preliminary results from evaluating our prototype using the *Energy-Language*

benchmark in §III. In §IV we outline a research agenda for extending these findings.

II. Background and Related Works

A. Energy-efficient Computing and Software

Energy-efficient computing focuses on reducing power consumption, usually while maintaining performance or correctness. Current solutions for energy-efficient software frequently require heavyweight design approaches [18], [19], pattern catalogues [12], [13], specialized programming languages [20], or decision frameworks [21], which makes it difficult for software engineers to adopt, implement, and subsequently maintain energy characteristics after evolution.

Another challenge of energy-efficient software design is that relationships between memory, latency, and energy are counterintuitive. Common metrics for code performance such as FLOPS, latency, and memory usage are not necessarily strongly correlated with each other or with energy efficiency [22]. On certain hardware, even reducing input sizes may increase latency [23]. Furthermore, improving characteristics such as parallelism [24] may increase energy usage [25]. *Thus, creating code that avoids confounding performance metrics with energy efficiency is difficult, but critically important if energy use is the primary metric of interest.*

B. LLM-Driven Code Generation and Optimization

LLMs are transforming software engineering practices [16], [17]. Recent experiments show LLMs assisting with error message interpretation [26], cybersecurity defect repair [27], cloud incident mitigation [28], and requirements elicitation [29]. Moreover, recent studies also evaluate LLMs on efficient code generation and optimization, showing they can achieve both without compromising correctness [30]–[32].

However, existing research primarily focuses on leveraging LLMs to generate fast code, rather than energy-efficient code. When energy consumption is the primary concern, current code-generation LLMs may fall short, as they do not explicitly consider the impact of their code on power consumption. *Nevertheless, building on previous work, we anticipate that with appropriate adaptations, LLMs can be made to optimize for energy efficiency, extending their capabilities beyond just minimizing latency.*

III. Prototype: Design, Implementation, & Eval.

We propose an approach to evaluate LLMs' potential in refactoring software for energy efficiency. As an initial test of this concept, we developed a prototype of an automated LLM-assisted tool for energy optimization. To ensure effectiveness, all optimizations must maintain the original system behavior without introducing semantic changes while improving energy efficiency. This section outlines the design, prototype implementation, and preliminary evaluation of our approach.

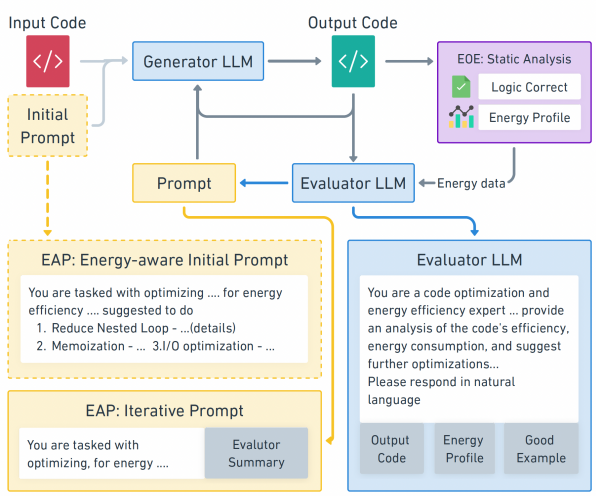

Fig. 2: Overview of our prototype. The core components of the automated feedback loop are Energy-Aware Prompting (EAP) and Energy Optimization Evaluation (EOE).

A. Design

Our system leverages state-of-the-art prompt engineering techniques and LLM feedback loops, applying them uniquely to the domain of energy measurement. The key to our design lies in enabling the LLM to be energy-aware, guiding it to generate code that not only optimizes for speed but also makes significant improvements in energy efficiency. As we described in §II-A, optimizing a program for energy efficiency is non-trivial, since metrics associated with latency and throughput (parallelism) are not necessarily correlated with power consumption. To overcome this challenge, we incorporate energy profiling mechanisms into the LLM feedback loop. The LLM is given power-consumption data alongside latency, which is used to adjust generated-code.

Figure 2 shows the prototype design. Following prior LLM applications, we chose an automated feedback loop structure to allow iterative improvement and repair. Our loop involves two modules: Energy-Aware Prompting (EAP), and Energy Optimization Evaluation (EOE). The EAP module integrates In-Context Learning [33] and Chain-of-Thought Prompting [34] to deliver detailed and informative instructions to the Generator LLM. the EOE module evaluates the correctness and energy efficiency of the LLM-generated code, using an Evaluator LLM to provide feedback for iterative refinement.

B. Prototype Implementation

Automated Feedback Loop: This loop applies the GPT-4o as two agents, building on methods proposed in prior studies [35]–[37]. The AFL consists of a Generator LLM and an Evaluator LLM. As shown in Figure 2, the original code and Energy-Aware Prompting serve as inputs to the Generator LLM. The Generator's output is subsequently evaluated by the Evaluator LLM, which assesses the code based on its correctness and energy profile and provides recommendations for refinement. By decoupling code generation from energy

data analysis, we ensure that each LLM remains focused on its respective task, thereby improving overall performance.

Energy-Aware Prompting (EAP): In the Generator LLM, we employed One-Shot, Chain of Thought (CoT), and self-consistency prompting techniques to enhance the model's performance by providing sufficient contextual information on energy-efficient computing. The prompt consists of three components. First, the task for the Generator to perform is described. Second, a brief code snippet, an optimized example, and a CoT rationale behind the optimization are provided. Third, we integrate self-consistency by prompting the Generator to consider multiple optimization strategies focused on efficiency before selecting a final approach [38]. These prompts encourage the LLM to reason more carefully, and provide an efficient and correct solution.

Energy Optimization Evaluation (EOE): This module has two objectives: (1) ensure the optimized code is compilable and functionally correct, and (2) measure energy efficiency and offer further optimization guidance.

The correctness evaluation involves two key steps. First, the EOE module checks if the optimized code compiles successfully. If it fails, the Generator LLM receives the error message and optimized code for Self-Reflection and correction [39], [40]. Second, the EOE runs a regression test to verify logical correctness. If runtime errors or output mismatches occur, the LLM is iteratively prompted with expected and actual outputs to refine optimization until the regression test passes, ensuring the optimized code matches the original functionality.

Once correctness is confirmed, we assess the energy efficiency and latency of the optimized code. The Evaluator LLM then analyzes the optimized code's energy profile and provides the Generator LLM with Natural Language Feedback (NLF) for further optimization. The Evaluator is provided with (1) the original code, (2) the code with the lowest power consumption from the Optimization History Buffer, and (3) the most recent optimized code. This helps the Evaluator generate more precise feedback by correlating energy usage with optimizations while learning from the best historical example to drive further improvements.

C. Experimental Setup

Software Benchmark: Building on the prior research in Android energy efficiency [41], [42], we selected *Energy-Language* as the benchmark tool for our experiments [43]. It offers comprehensive support for a wide range of testing algorithms across multiple programming languages and includes integrated energy measurement capabilities. Additionally, *Energy-Language* is designed for scalability which facilitates easy incorporation of new codebases and programming languages into the testing suite.

Power Consumption Measurement: We use the *Energy-Language*'s built-in energy measurement system, which reads data from Model-Specific Registers (MSRs) supported by the Running Average Power Limit (RAPL) interface [44]–[46]. To measure energy using MSRs, the RAPL registers accumulate energy usage over time. Energy efficiency is determined by reading the relevant MSR registers before and after a workload and calculating the difference. These values are scaled to Joules using energy units from the `MSR_RAPL_POWER_UNIT` register, allowing precise analysis of power usage for different components.

D. Preliminary Results and Analysis

We evaluated our prototype using the C++ benchmark, consisting of eleven unique programs. We successfully ran six of the programs with the results presented in Table I. The remaining five programs were not tested owing to compilation errors, runtime issues, or `GPT-4o` model token limit constraints. We collected data from three different optimization approaches: the original un-optimized program without the `-O3` flag, the compiler-optimized code with the `-O3` flag, and the LLM-optimized code without the `-O3` flag.

Comparison of Results with GCC –O3: As shown in Table I, the results indicate that our framework reduces energy consumption in 83% of the programs tested and outperforms the compiler optimization baseline in 50% of them. Furthermore, the results from our framework are characterized by a much higher Standard Deviation relative to the compiler optimization baseline. This is most clearly seen in the *fannkuch-redux* test where the Standard Deviation for the measured energy and latency for our framework is 883 and 5441. It is also notable that, in the case of the *spectral-norm* benchmark, compiler optimizations increased energy consumption from 31J to 57J, while our code significantly reduced energy usage, bringing it down from 31J to just 15J.

Observations and Discussion: Our prototype illustrates the potential for LLM-generated energy-aware optimizations. We observed the LLM was proficient at improving memory traffic patterns, leading to reduced energy usage. For instance, in *spectral-norm*, the LLM replaces Streaming SIMD Extensions (SSE) with Advanced Vector Extensions (AVX2), which performs twice as many floating-point computations in parallel. Additionally, the LLM-generated code processed data in larger chunks (size 256 bit vs. size 128 bit), reducing memory traffic and thus energy usage. These enhancements demonstrate the LLM's ability to optimize beyond the scope of individual functions. These results also highlight our framework's ability to reduce energy usage in a deliberate manner.

We highlight two limitations of the prototype. First, there is significant variance in the energy consumption of LLM-optimized software across successive iterations. We conjecture two causes of this property. First, LLM output is sensitive to hyperparameter choices such as Top-K and Temperature; therefore the input-output relationship can be non-deterministic. Though this non-determinism can be controlled (*e.g.,* with Temperature$=0$) using a non-zero Temperature allows for more substantial optimization attempts. Second, we hypothesize that prompting alone is insufficient. Our prototype relies on prompting and learning from feedback, but uses existing model parameters. As a result, the LLM may not be properly conditioned on applying energy-aware optimizations.

Table I: Preliminary results of our prototype. We illustrate the performance of two optimization strategies (compiler, ours) across six algorithms compared with a baseline implementation. LLM improvements over the compiler optimizations are indicated with green, and degradation in red. Values are rounded to the nearest integer.

Program	Size	Original Code		Compiler Optimized		LLM Optimized	
		Latency (ms)	Energy (J)	Latency (ms)	Energy (J)	Latency (ms)	Energy (J)
binary-trees	139 lines (9 funcs)	760	87	538	51	187 – 954	20 – 64
fannkuch-redux	198 lines (5 funcs)	6536	1119	1605	259	5787 – 17918	953 – 2945
n-body	184 lines (6 funcs)	20905	1150	2056	115	11601 – 22575	608 – 1193
pidigits	68 lines (5 funcs)	566	28	592	29	525 – 542	27 – 28
k-nucleotide	154 lines (11 funcs)	3685	476	865	85	3677 – 5318	478 – 789
spectral-norm	135 lines (8 funcs)	176	31	409	57	87 – 6637	15 – 342

As a second limitation, the Generator LLM was effective at resolving compiler errors, but struggled to address run-time errors. Unlike compiler errors, run-time errors often do not yield an explicit error message. As a result, in longer and more complex programs such as *fannkuch-redux*, *n-body*, and *k-nucleotide*, the LLM required more iterations to address this type of error. Additionally, we observed that spending more than three iterations on code-refinement led an increase in energy usage and latency. This indicates that the LLM lacks an understanding of run-time error-handling or early-stopping criteria for code-optimization.

IV. Future Plans

A. Improvements to Evaluation

Baseline Analysis: In our preliminary evaluation, we used the `gcc -O3` optimization flag for C++ benchmarks. We will extend similar analysis to other languages, such as Java with its `JIT` optimizer. Additionally, we will benchmark our approach against the recently released `GPT-o1` model which has native chain-of-thought ability. We will evaluate our pipeline on larger applications including standard data center benchmarks such as the Facebook benchmark of datacenter cloud applications, *DCPerf* [47], and standard HPC benchmarks evaluated in the Green500 work [48].

Extended SOTA Comparison: We plan to evaluate state-of-the-art code-generation models on baseline programs, and compare their performance with our prototype. This will identify shortcomings and potential improvements for energy-aware LLM code generation, paving the way for additional research efforts. It will also be interesting to compare with other approaches for energy efficiency, such as search-based software engineering approaches.

B. Prototype Framework Refinements

LLM Prompting: We will refine our prompts. Building on previous work, we will apply prompt engineering methods such as few-shot learning, structured CoT prompting [49], and prompt chaining. This will yield the first catalog of LLM prompts for energy efficiency.

LLM Fine-tuning: We will fine-tune the LLM on a curated dataset of energy-efficient algorithms and low-power software practices. For this purpose, we will develop a new training dataset for energy-efficient computing, focusing on efficiency-critical data center software and HPC algorithms. Inspired by prior work [50], we will perform large-scale data collection from GitHub repositories, using a language model to analyze commit messages and identify energy-efficiency commits. This dataset would allow us to fine-tune the LLM performance.

Hardware-Aware Code Generation: Our current approach focuses on optimizing specific components of the system or hardware at the code level without considering its deployment scenario (system resources, hardware architecture, etc.). We believe integrating a Retrieval-Augmented-Generation (RAG) framework [51] that relates code-modifications to hardware and system specifications could rectify this issue. In *spectral-norm*, we observed GPT do this without specific prompting.

C. Broader Research Directions

Weighing Costs of LLM-Driven Energy Optimizations: AI systems such as LLMs are power hungry. We will assess both the energy and computational costs of using LLMs for software energy-efficiency optimization, comparing the energy usage of different LLMs for the same tasks. Based on the findings, we will develop adaptive strategies, such as triggering LLM-based optimizations only when significant energy savings are expected.

Multi-Objective Optimization: We will explore whether LLMs can be adapted for multi-objective optimization, balancing energy efficiency with key performance metrics like latency and throughput. Developing tools that optimize across these competing objectives would enhance LLM applicability in real-world scenarios where such trade-offs are critical. LLMs could also generate code across a spectrum of efficiency constraints. In this context, formal verification techniques might be incorporated for stronger correctness guarantees [52].

Second-Class Citizenship is Better than None: After decades of calls for energy-aware computing, it seems clear to us that energy will not be prioritized as much as business-critical metrics such as latency and throughput. We hope, however, that it will not be ignored. We suggest that engineers and researchers might approach energy as a secondary performance metric, one that should still be considered after primary metrics are met. This would require engineering tools and processes that support energy considerations in a lightweight way, and that can be applied after the primary engineering goals are met. Our LLM approach may become one such tool.

V. Conclusion

In this paper, we explored the potential of applying LLMs to energy-efficient software development and introduced an automated approach for energy-aware code optimization. Our initial experiments demonstrate that LLMs can improve energy efficiency while maintaining code correctness. Based on this emerging result, we discuss our immediate and longer-term research plans.

Data availability: Code, prompts, and data are available: https://anonymous.4open.science/r/E2COOL-5CD4.

REFERENCES

[1] C. Wilke, S. Richly *et al.*, "Energy consumption and efficiency in mobile applications: A user feedback study," in *International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 2013.

[2] E. P. R. Institute, "Powering intelligence: Analyzing artificial intelligence and data center energy consumption," Electric Power Research Institute, Brochure Product ID 3002028905, 2024.

[3] European Commission, "Commission adopts EU-wide scheme for rating sustainability of data centres," 2024.

[4] I. Manotas, C. Bird *et al.*, "An empirical study of practitioners' perspectives on green software engineering," in *International Conference on Software Engineering*, 2016.

[5] J. Arjona Aroca, A. Chatzipapas, A. Fernández Anta, and V. Mancuso, "A measurement-based analysis of the energy consumption of data center servers," in *International Conference on Future Energy Systems*, 2014.

[6] J. Gao and R. Jamidar, "Machine learning applications for data center optimization," 2014.

[7] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, 2015.

[8] T. Daradkeh and A. Agarwal, "Cloud workload and data center analytical modeling and optimization using deep machine learning," *Network*, 2022.

[9] F. Almeida, J. Arteaga *et al.*, "Energy measurement tools for ultrascale computing: A survey," *Supercomputing Frontiers and Innovations*, 2015.

[10] C. Sahin, F. Cayci *et al.*, "Initial explorations on design pattern energy usage," in *First International Workshop on Green and Sustainable Software (GREENS)*, 2012.

[11] A. Noureddine and A. Rajan, "Optimising energy consumption of design patterns," in *International Conference on Software Engineering*, 2015.

[12] G. Pinto, K. Liu *et al.*, "A comprehensive study on the energy efficiency of Java's thread-safe collections," in *ICSME*, 2016.

[13] S. Maleki, C. Fu, A. Banotra, and Z. Zong, "Understanding the impact of object oriented programming and design patterns on energy efficiency," in *International Green and Sustainable Computing Conference*, 2017.

[14] L. Papadopoulos, C. Marantos *et al.*, "Interrelations between software quality metrics, performance and energy consumption in embedded applications," in *International Workshop on Software and Compilers for Embedded Systems*, 2018.

[15] L. Karita, B. C. Mourão, and I. Machado, "Software industry awareness on green and sustainable software engineering: a state-of-the-practice survey," in *Brazilian Symposium on Software Engineering*, 2019.

[16] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Software*, 2023.

[17] A. Sarkar, A. D. Gordon *et al.*, "What is it like to program with artificial intelligence?" *arXiv:2208.06213*, 2022.

[18] S. te Brinke, S. Malakuti *et al.*, "A design method for modular energy-aware software," in *ACM Symposium on Applied Computing*, 2013.

[19] S. Te Brinke, S. Malakuti *et al.*, "A tool-supported approach for modular design of energy-aware software," in *ACM Symposium on Applied Computing*, 2014.

[20] M. Couto *et al.*, "Towards a green ranking for programming languages," in *the Brazilian Symposium on Programming Languages*, 2017.

[21] I. Manotas, L. Pollock, and J. Clause, "Seeds: a software engineer's energy-optimization decision support framework," in *International Conference on Software Engineering*, 2014.

[22] M. Dehghani, A. Arnab *et al.*, "The efficiency misnomer," *arXiv: 2110.12894*, 2022.

[23] N. J. Eliopoulos, P. Jajal, J. Davis, G. Liu, G. K. Thiravathukal, and Y.-H. Lu, "Pruning one more token is enough: Leveraging latency-workload non-linearities for vision transformers on the edge," 2024. [Online]. Available: https://arxiv.org/abs/2407.05941

[24] K. Mondal and P. Dutta, "Big data parallelism: Challenges in different computational paradigms," in *International Conference on Computer, Communication, Control and Information Technology*, 2015.

[25] C. Jin, B. R. de Supinski, D. Abramson *et al.*, "A survey on software methods to improve the energy efficiency of parallel computing," *Int. J. High Perform. Comput. Appl.*, 2017.

[26] J. Leinonen, A. Hellas *et al.*, "Using large language models to enhance programming error messages," in *ACM Technical Symposium on Computer Science Education*, 2023.

[27] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[28] T. Ahmed, S. Ghosh *et al.*, "Recommending root-cause and mitigation steps for cloud incidents using large language models," in *International Conference on Software Engineering*, 2023.

[29] J. White *et al.*, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," in *Generative AI for Effective Software Development*, 2024.

[30] J. Liu, S. Xie, J. Wang *et al.*, "Evaluating language models for efficient code generation," *arXiv:2408.06450*, 2024.

[31] A. G. Shypula, A. Madaan, Y. Zeng *et al.*, "Learning performance-improving code edits," in *International Conference on Learning Representations*, 2024.

[32] A. Fan, B. Gokkaya *et al.*, "Large language models for software engineering: Survey and open problems," in *International Conference on Software Engineering: Future of Software Engineering*, 2023.

[33] T. Brown, B. Mann *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020.

[34] J. Wei, X. Wang *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, 2024.

[35] Y. Qu, T. Zhang, N. Garg, and A. Kumar, "Recursive introspection: Teaching LLM agents how to self-improve," in *ICML Workshop on Structured Probabilistic Inference & Generative Modeling*, 2024.

[36] A. Madaan, N. Tandon, P. Gupta *et al.*, "Self-refine: Iterative refinement with self-feedback," 2023.

[37] J. Huang, S. S. Gu, L. Hou *et al.*, "Large language models can self-improve," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[38] D. S. e. a. Xuezhi Wang, Jason Wei, "Self-consistency improves chain of thought reasoning in language models," 2023.

[39] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," 2023.

[40] Y. J. Ma, W. Liang, G. Wang *et al.*, "Eureka: Human-level reward design via coding large language models," in *International Conference on Learning Representations*, 2024.

[41] Q. Xu, J. C. Davis *et al.*, "An empirical study on the impact of deep parameters on mobile app energy usage," in *International Conference on Software Analysis, Evolution and Reengineering*, 2022.

[42] W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of Android app development approaches," in *International Conference on Mining Software Repositories*, 2017.

[43] I. Gouy, "The computer language benchmarks game," accessed: 2024-10-10. [Online]. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame

[44] C. Thorat and V. Inamdar, "Energy measurement of encryption techniques using RAPL," in *International Conference on Computing, Communication, Control and Automation*, 2017.

[45] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using RAPL," *SIGMETRICS Perform. Eval. Rev.*, 2012.

[46] D. Hackenberg, R. Schöne *et al.*, "An energy efficiency feature survey of the Intel Haswell processor," in *International Parallel and Distributed Processing Symposium Workshop*, 2015, pp. 896–904.

[47] F. Research. (2023) Dcperf benchmark suite for hyperscale cloud applications. [Online]. Available: https://github.com/facebookresearch/DCPerf

[48] W.-c. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, 2007.

[49] J. Li, G. Li, Y. Li, and Z. Jin, "Structured chain-of-thought prompting for code generation," *ACM Transactions on Software Engineering and Methodology*, 2023.

[50] I. Moura, G. Pinto, F. Ebert, and F. Castor, "Mining energy-aware commits," in *International Conference on Mining Software Repositories*, 2015.

[51] P. Lewis, E. Perez *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, 2020.

[52] N. Tihanyi, T. Bisztray, R. Jain, M. A. Ferrag, L. C. Cordeiro, and V. Mavroeidis, "The formai dataset: Generative ai in software security through the lens of formal verification," in *International Conference on Predictive Models and Data Analytics in Software Engineering*, 2023.