



Delft University of Technology

Document Version

Final published version

Licence

CC BY

Citation (APA)

Cunha, S., Ribeiro, F., Cruz, L., & Saraiva, J. (2026). Not All Local LLMs Are Equal: A Benchmark of Energy and Performance. In *Proceedings - 2026 IEEE/ACM 10th International Workshop on Green and Sustainable Software, GREENS 2026* (pp. 99-106). (Proceedings - 2026 IEEE/ACM 10th International Workshop on Green and Sustainable Software, GREENS 2026). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3786148.3788630>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

In case the licence states "Dutch Copyright Act (Article 25fa)", this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Not All Local LLMs Are Equal: A Benchmark of Energy and Performance

Simão Cunha

University of Minho & HASLab, INESC TEC
Braga, Portugal
simao.s.cunha@inesctec.pt

Luís Cruz

Delft University of Technology
Delft, Netherlands
L.Cruz@tudelft.nl

Francisco Ribeiro

New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
francisco.ribeiro@nyu.edu

João Saraiva

University of Minho & HASLab, INESC TEC
Braga, Portugal
saraiva@di.uminho.pt

Abstract

The rapid adoption of Large Language Models (LLMs) is transforming research, education, software development and everyday life. As their use grows, so does the diversity of available models, from general-purpose to code-oriented LLMs that can run both in data centers and on edge devices. Several benchmarks have emerged to evaluate their performance in code generation and completion tasks, yet their energy and time efficiency remain underexplored.

This paper evaluates five local LLMs on HumanEval-X and MBPP+ to analyze their accuracy, runtime and energy consumption under CPU-only inference, reflecting realistic on-device deployment scenarios where GPUs are unavailable. The results reveal clear trade-offs between effectiveness and efficiency: while some models achieve higher accuracy, others deliver comparable results with substantially lower energy use. In particular, 3-shot prompting consistently improves runtime and energy efficiency compared to 0-shot, without sacrificing code quality. These findings emphasize that prompt design and model selection must be considered together when deploying LLMs for coding tasks and call for the creation of practical prompt-efficiency guidelines to support more sustainable and efficient use of local LLMs.

CCS Concepts

• **Hardware** → Power estimation and optimization; • **Computing methodologies** → Natural language processing; • **Software and its engineering** → Software creation and management.

Keywords

Large Language Models, Energy Consumption, Green Software, Programming Languages

ACM Reference Format:

Simão Cunha, Francisco Ribeiro, Luís Cruz, and João Saraiva. 2026. Not All Local LLMs Are Equal: A Benchmark of Energy and Performance. In *10th International Workshop on Green and Sustainable Software (GREENS '26)*, April 12–18, 2026, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3786148.3788630>



This work is licensed under a Creative Commons Attribution 4.0 International License. *GREENS '26, Rio de Janeiro, Brazil*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2381-0/26/04
<https://doi.org/10.1145/3786148.3788630>

1 Introduction

Large language models (LLMs), such as GPT-4 [56], are transforming software development. Initially created for natural language processing [21, 45], LLMs now serve as AI-driven coding assistants, supporting code generation [10], debugging [70] and optimization [49]. Tools like ChatGPT [55] and GitHub Copilot [19], as well as models such as CodeBERT [23] and Code Llama [63], illustrate their growing role in software engineering, assisting in many tasks such as code completion [35], test generation [44] and program repair [64].

The widespread use of AI coding assistants shows their growing importance in software development. A 2024 GitHub survey [26] found that over 97% of developers have used such tools and most rely on them both professionally and personally. As their use expands, so do the computational and environmental costs. It is therefore essential to assess not only their effectiveness but also their efficiency in energy consumption and runtime.

Despite their benefits, LLMs demand significant computational resources, raising concerns about energy consumption [54]. For example, Microsoft reported a 30% increase in greenhouse gas emissions in 2023, largely due to AI expansion [7]. Privacy and security are also critical, especially for organizations handling sensitive data [65]. Running LLMs locally in offline mode offers greater control and data protection, particularly in sectors like finance, healthcare and defense [40]. As LLMs become more influential, the range of available models continues to expand. Between October 2019 and December 2023, more than fifty LLMs were released [54], covering a wide range of tasks and running on increasingly diverse hardware. This diversity motivates comparing local LLMs not only in terms of effectiveness - their ability to generate correct code - but also in terms of efficiency, as both dimensions influence their practical usability in real-world software development.

This paper evaluates five widely used local LLMs on two standard benchmarks for code generation and code completion, focusing on the following key aspects: (1) energy efficiency and runtime (considering only the LLMs' consumption during inference, *not* that of the generated code); (2) code generation effectiveness, measured as task accuracy and analyzed together with runtime and energy to highlight performance-efficiency trade-offs.

Prompting strategies such as *zero-shot* and *few-shot* prompting, where models receive no examples or only a few examples to guide their responses, directly influence output quality [5, 30,

31, 69, 81, 82]. While prior work has primarily examined accuracy, little is known about how prompting configurations affect energy consumption. To address this gap, we systematically evaluate the energy consumption of five LLMs under different n -shot prompting setups. Furthermore, we examine how the number of examples (*shots*) affects energy use and runtime efficiency across models and tasks, helping practitioners optimize deployments for greater sustainability.

Thus, our work addresses the following research questions:

RQ1: How do state-of-the-art local LLMs for code compare in terms of energy consumption and runtime efficiency?

To answer this question, we benchmark five widely used local LLMs (using quantized versions) including *Meta-Llama-3*, *deepseek-coder*, *codellama*, *codegeex4* and *starling-lm*. We use two established benchmarks, HumanEval-X [83] and MBPP+ [47], to evaluate the quality and functionality of the generated code. However, traditional evaluations often ignore energy consumption, a key aspect of Green AI [16]. We therefore incorporate energy consumption as a sustainability metric alongside accuracy and runtime.

RQ2: How does n -shot prompting affect energy consumption and runtime efficiency across models and tasks?

We ran a large-scale empirical study using a batch-based framework to execute LLMs on coding tasks with the HumanEval-X and MBPP+ benchmarks, tracking both energy consumption and runtime. The study consumed about 304 million joules over 53 days. We tested $n = 0, 3$, treating $n = 3$ as a *few-shot* setting to see its effect on energy consumption and runtime during inference. Results show that energy consumption and runtime vary widely with the model, benchmark, programming language and prompting style, with 3-shot prompting being generally more efficient than 0-shot.

This paper is organized as follows: Section 2 reviews related work. Section 3 describes our methodology, including models, benchmarks, energy measurement framework and experimental setup. Section 4 presents and discusses results. Section 5 addresses threats to validity. Section 6 summarizes our conclusions, followed by a description of our replication package and data availability.

2 Related Work

Several approaches have targeted energy efficiency in software. These range from code-level techniques, such as removing obsolete code [11], to managing software lifecycles to avoid energy-intensive processes [58], optimizing hardware utilization [38], and, in distributed settings, improving data-transfer protocols [14]. All these efforts align with the Green AI agenda of minimizing computational carbon footprints [77]. Moreover, research shows that programming language choice significantly affects energy consumption [57]. Tools like Intel RAPL [34] and CodeCarbon [15] enable real-time energy analysis, allowing developers to make informed decisions during software optimization.

More recently, LLMs have been applied in software development through specialized models such as CodeGPT, which focuses on automating tasks like bug fixing and code repair [60], and Code Llama, which is designed for efficient and secure code generation [63]. Other efforts, such as DeepDev-PERF [24] and Mentat [61], focus on performance enhancement and debugging. Additionally, multilingual approaches to program repair have been demonstrated

by tools like AlphaRepair, which addresses software defects across various languages [80]. LLMs trained on large datasets support code generation, debugging, and optimization [28], with fine-tuning and prompt engineering enhancing performance [9, 37].

Structured benchmarks like HumanEval-X and MBPP+ provide a comprehensive framework to assess the quality and efficiency of LLM-generated code [2, 10, 13, 47, 52]. At the same time, the field of Green AI has focused on sustainability by monitoring and benchmarking computational footprints [77].

Several researchers have investigated key aspects of LLMs for Code (LLM4Code). A significant body of work has focused on the energy consumption of these models, particularly during the inference phase for code generation tasks [1, 22, 48, 66], as well as during the training phase [68, 76, 78].

Our work explores similar topics by leveraging quantized models and evaluating LLM benchmarks, but focuses on energy consumption and CPU execution time. We chose CPUs instead of GPUs or TPUs because they are widely available, from laptops to servers, whereas GPUs and TPUs are not always accessible or practical. Building on prior studies of LLM energy usage, we evaluate five LLMs across quantized configurations on CPUs. Many scenarios require smaller models typically run on CPUs and practitioners often want to run LLMs locally on laptops or edge devices without GPU access, which is not a niche use case [42, 53, 67]. Industry initiatives also support CPU inference: HuggingFace provides Optimum and BetterTransformer [32]. Intel has published Llama 3 CPU optimizations [33] and Neural Magic explores sparsity-based efficiency techniques [41]. Studying CPU-based inference is realistic and valuable, while assuming GPU-only execution oversimplifies real-world AI on laptops, desktops and edge devices.

3 Methodology

This section presents the empirical methodology used to address the research questions (RQs) from Section 1. Rather than aligning methods with each RQ separately, we use a unified experimental design for our analysis. Guided by [79] (see Part II - Steps in the Experiment Process), our study systematically benchmarks and compares multiple LLMs under identical conditions using standardized benchmarks and quantitative metrics. The following subsections detail the evaluated LLMs, energy measurement, benchmarks, metrics, experimental setup and hardware specifications. This integrated approach ensures all RQs are fully addressed.

3.1 LLMs Under Analysis

This study explores running LLMs locally on typical machines, specifically a CPU-only setup with 32 GB of RAM (more information in Section 3.5), reflecting common hardware constraints. We prioritized models that run efficiently under these conditions, focusing on quantized GGUF models, which reduce resource usage while maintaining reasonable performance [51]. To ensure diversity, we selected models from various families, avoiding bias toward any single architecture or training method.

Since our focus is on code generation, we selected models that are either specialized or fine-tuned for programming tasks and have been evaluated on public code-related benchmarks [20, 27, 43, 50, 63, 71, 75, 83]. These include *deepseek-coder*, *codellama* and

codegeex4. Additionally, we included general-purpose models outside the LLM4Code scope, namely *Meta-Llama-3* and *starling-lm*, to assess their performance in code generation tasks.

All selected models are open-source or, at a minimum, publicly available and runnable locally, ensuring transparency, reproducibility and practical feasibility for offline experimentation.

The models and their GGUF variants, referenced by short names throughout this paper, are *Meta-Llama-3* (*Meta-Llama-3-8B-Instruct-Q6_K*) [4], *deepseek-coder* (*deepseek-coder-6.7b-instruct.Q5_K_M*) [73], *codellama* (*codellama-7b-instruct.Q5_K_M*) [72], *codegeex4* (*codegeex4-all-9b-Q6_K_L*) [3] and *starling-lm* (*starling-lm-7b-alpha.Q5_K_S*) [74].

3.2 Energy Measurements

We measured CPU energy consumption using CodeCarbon on a Linux system equipped with an Intel CPU. CodeCarbon leverages Intel RAPL, providing accurate energy estimates with minimal overhead [29, 39, 59, 62]. Unless otherwise specified, all reported energy values correspond to CPU energy consumption measured by CodeCarbon through RAPL.

3.3 LLM Benchmarks

To evaluate the code generation and completion capabilities of the LLMs in this study, we use two established benchmarks focused on functional correctness: HumanEval-X [17], which assesses code completion in C++, Go, Java, JavaScript and Python [12] and MBPP+ [18], which evaluates Python code generation [47]. The original HumanEval [10] and MBPP [2] - simplified versions of these - include mainly synthetic tasks to test LLM code synthesis. Although not fully representative of real-world scenarios, they remain standard benchmarks [6, 10, 47].

The benchmarks themselves specify the evaluation setup for each LLM. The pass@k metric measures model performance by counting correct solutions that pass unit tests, using an unbiased estimator to select the best result among k generated samples [10]. In our experiments, we used HumanEval-X and MBPP+ with $k \in \{1, 10\}$ for pass@k.

3.4 Experimental Setup

We conducted experiments using the HumanEval-X and MBPP+ test suites, each following the methodologies and configurations detailed in Section 3.4.2. All LLMs were executed using LLaMa.cpp [25], a C/C++ library optimized for efficient inference across a range of hardware platforms, both local and cloud-based.

3.4.1 System Architecture. To ensure a modular and reproducible experimental setup, we designed our system architecture as observed in Figure 1. The framework consists of a command-line interface (CLI) tool developed in Python to analyze the energy consumption and runtime of LLMs while generating code for the benchmarks described in Section 3.3. The tool also records the metrics obtained from each generation.

3.4.2 Benchmark Execution. Each LLM was evaluated using both 0-shot and 3-shot prompting across all prompts in the test suites described in Section 3.3. To minimize the impact of outliers in energy consumption measurements (collected via CodeCarbon),

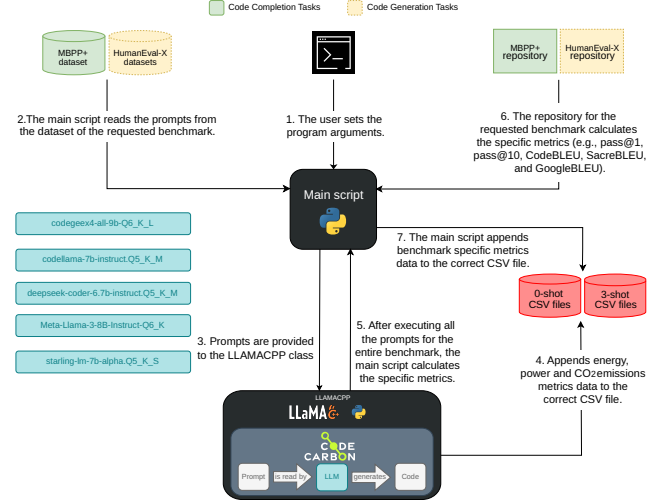


Figure 1: Architecture for Execution and Measurement

each prompt was executed 10 times per LLM. This repetition reduces the influence of environmental variability, leading to more reliable energy consumption estimates, which is in line with established recommendations [8]. For the pass@1 and pass@10 metrics, the 10 runs using the same model, prompt and parameters were grouped together to compute the final results. For energy and runtime analysis, each run was considered individually. Outliers in energy data were removed using the Interquartile Range (IQR) method on groups defined by prompt and programming language.

We computed pass@1 and pass@10 for all LLMs with unchanged LLaMa.cpp parameters: $max_tokens = 512$, $n_ctx = 4096$, $seed = 42$, $top_p = 0.95$ and $temperature = 0.6$ (the last two follow [63]).

For 3-shot prompting, we used the first three prompts of each programming language in HumanEval-X and the first three prompts from the MBPP+ test suite as examples. These specific prompts were not included in the actual 3-shot evaluation to avoid overlap. This ensured consistency in the examples used across all evaluations.

3.4.3 Statistical Significance. We performed a statistical analysis to determine if there were significant differences in energy consumption and execution time between 0-shot and 3-shot prompting. First, we assessed the normality of the data using the Shapiro-Wilk test. Based on the results, we applied the appropriate statistical test: the paired samples t-test for normally distributed data, or the Mann-Whitney U test for non-normally distributed data. For each comparison, we calculated the corresponding p-value to evaluate statistical significance. To account for multiple comparisons and reduce the risk of Type I errors, we adjusted the p-values using the Holm-Bonferroni method.

3.5 Experimental Equipment

The experiments were performed on a GNU/Linux computer, featuring an x86_64 Intel Core i7-8700 CPU with 6 cores and 12 threads, supported by 32 GB of DDR4 2666 MHz RAM. To minimize overhead and ensure consistent performance, only the minimal necessary

processes were running on the operating system during the experiments.

4 Results and Analysis

This section presents and analyzes the statistical tests, the relationship between energy consumption and execution time and the results from the benchmark problems HumanEval-X and MBPP+.

We analyze energy consumption and runtime, ranking LLMs based on results for HumanEval-X and MBPP+ tasks under 0-shot and 3-shot prompting. Our analysis shows a perfect positive correlation (coefficient = 1) between execution time and energy consumption for all benchmarks, meaning higher energy always implies longer runtime. Therefore, we show only energy plots here; execution time plots are in the online appendix.

4.1 Statistical Significance

We analyzed whether the prompting strategy (0-shot vs 3-shot) leads to statistically significant differences in execution time and energy consumption. This analysis was carried out across all combinations of LLM, programming language and benchmark.

In total, we conducted 120 hypothesis tests. For the HumanEval-X benchmark, we evaluated five LLMs across five programming languages, yielding 25 combinations. Considering two efficiency metrics - energy consumption and execution time - and comparing two prompting strategies, this results in 50 tests per metric, for a total of 100 tests. For the MBPP+ benchmark, which includes only one programming language, we performed 10 tests per metric, totaling 20 additional tests.

For each comparison, we tested whether the choice of prompting strategy (0-shot vs 3-shot) leads to a statistically significant difference in the distribution in a given metric. The null hypothesis (H_0) assumes no difference between the two prompting strategies, while the alternative hypothesis (H_1) suggests the distributions differ significantly.

We first tested the normality of each sample with the Shapiro-Wilk test. Because most samples were not normally distributed, we applied the Mann-Whitney U test to compare the metric values between the 0-shot and 3-shot prompting conditions.

Due to the large number of comparisons, we applied the Holm-Bonferroni correction to control the family-wise error rate. The correction was performed separately for each metric (Energy Consumption and Execution Time), considering all combinations of LLM, language and prompting condition (0-shot vs 3-shot).

After correction, all results remained statistically significant ($p_{\text{adjusted}} < 0.05$), indicating that prompting strategy can meaningfully impact both energy consumption and execution time. Complete tables with raw and adjusted p-values are included in the online appendix.

4.2 Code Completion: HumanEval-X

To estimate error rates in HumanEval-X, we applied the Wilson score interval for each LLM across programming languages, with detailed results provided in the online appendix. In the 0-shot setting, error rates ranged from 0.02% to 1.13% (4-74 errors out of 8200 entries), indicating low and consistent values across models and languages. In the 3-shot setting, the range slightly widened to

0.02%-1.22% (up to 79 errors out of 8050 entries), confirming that erroneous entries remain negligible.

Figure 2 compares the energy consumption of five LLMs on HumanEval-X, showing a clear reduction under 3-shot prompting, likely due to fewer generated tokens (see online appendix for details).

Among the LLMs, *starling-lm* demonstrates the highest average energy consumption (5523 J), while *codellama* exhibits the lowest (2238 J) in 0-shot prompting. In contrast, during 3-shot prompting, *codegeex4* consumes the most energy on average (2007 J), whereas the other four LLMs show similar consumption patterns.

Regarding programming languages, in 0-shot prompting, C++ code generation consumes the most energy on average (4773 J), while Python consumes the least (3654 J). Conversely, during 3-shot prompting, generating Python code requires the least energy (915 J), whereas C++ requires the most (2046 J).

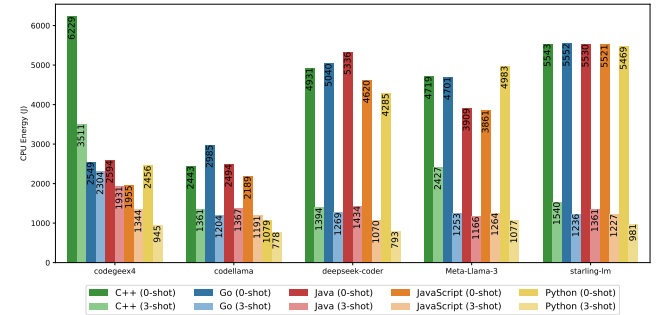


Figure 2: Energy for HumanEval-X in 0-Shot and 3-Shot

Focusing on the functional correctness metrics, such as pass@1 and pass@10, calculated for all the LLMs using the HumanEval-X test suites, we provide bar plots for both metrics using 0-shot and 3-shot prompting. Figure 3 refers to the pass@1 and pass@10 metrics, respectively. We observe that the pass@1 metric increases significantly from 0-shot to 3-shot prompting, especially in code generation for C++, Go, Java and JavaScript. For instance, *Meta-Llama-3* improves from 0.04 in pass@1 for Java in 0-shot to 0.50 in 3-shot. However, for Python, not all LLMs show improvement; for example, *codegeex4* decreases from 0.70 in 0-shot to 0.64 in 3-shot. This appears to be an exception, as most LLMs increase their pass@1 scores across other languages.

Another noteworthy observation is the increase in the pass@10 metric compared to pass@1 for both 0-shot and 3-shot prompting across all LLMs and programming languages. This can be explained by the higher likelihood of passing more tests when generating multiple outputs (10 outputs in pass@10) rather than relying on a single output, as in pass@1. For example, *deepseek-coder*'s Python executions improve from 0.58 in pass@1 in 0-shot to 0.89 in 3-shot. Similarly, *codellama*'s Java generations improve from 0.36 to 0.37.

4.3 Code Generation: MBPP+

To examine the energy consumption of the same LLMs on a code generation task, we used the MBPP+ benchmark, as shown in Figure 4. This complementary analysis provides insights into energy

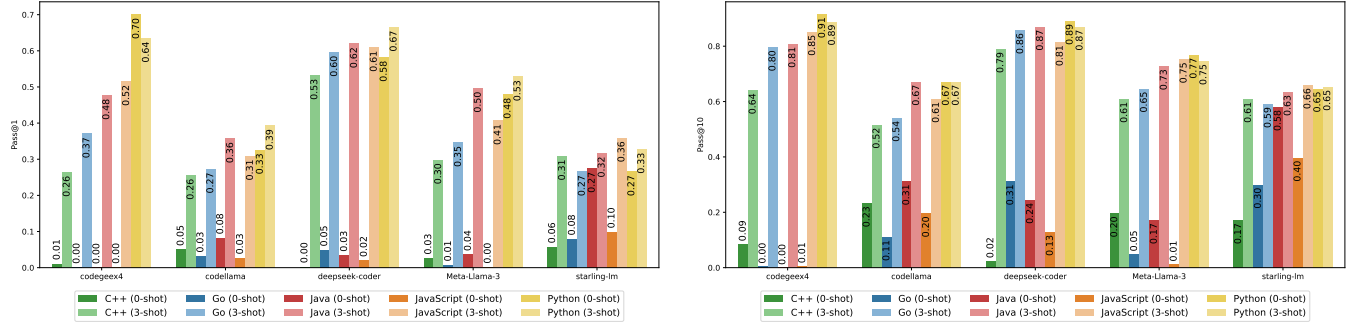


Figure 3: Pass@1 (left) and Pass@10 (right) for HumanEval-X in 0-Shot and 3-Shot settings.

efficiency in code generation, supplementing the code completion results from HumanEval-X. In the 0-shot setting, *starling-lm* exhibits the highest average energy consumption (5201 J), while *codellama* consumes the least (598 J). In the 3-shot setting, energy consumption generally decreases across all LLMs, except for *codellama*, which increases to 678 J - a 13.38% rise. *starling-lm* shows the largest reduction, decreasing to 582 J (an 88.81% drop), making it the most energy-efficient model. In contrast, *codegeex4* records the highest average energy consumption per task (698 J) in the MBPP+ test suite.

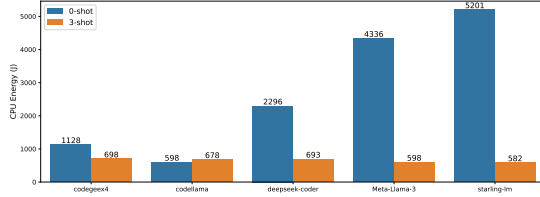


Figure 4: Energy for MBPP+ in 0-Shot and 3-Shot

Figure 5 shows the pass@1 and pass@10 results for both the original MBPP and the enhanced MBPP+ test suites. The MBPP results are included for completeness, as the MBPP+ evaluation automatically reports both. Since the two benchmarks use the same tasks but MBPP+ adds more test cases per task, we compare them briefly to show the impact of stricter validation.

Among the models, *codegeex4* shows the lowest overall performance, with average pass@1 scores of 0.12 (0-shot) and 0.27 (3-shot), and pass@10 scores of 0.13 (0-shot) and 0.28 (3-shot) on MBPP and MBPP+. In contrast, *Meta-Llama-3* achieves the highest scores across all models, with pass@1 averaging 0.63 (0-shot) and 0.37 (3-shot), and pass@10 averaging 0.78 (0-shot) and 0.73 (3-shot). As expected, scores are generally lower on MBPP+ due to the larger number of test cases. Also, pass@10 consistently outperforms pass@1, since generating multiple outputs increases the likelihood of a correct solution.

All LLMs show improved pass@1 and pass@10 scores on both MBPP and MBPP+ after code sanitization, which removes extraneous code and isolates the function of interest. The drop in performance from MBPP to MBPP+ remains consistent post-sanitization.

In the 0-shot setting with sanitized code, *codellama* and *deepseek-coder* achieve the highest scores (pass@1: 0.52, pass@10: 0.77), while *codegeex4* scores lowest (pass@1: 0.21, pass@10: 0.49). In the 3-shot setting, *starling-lm* performs best (pass@1: 0.52, pass@10: 0.77), whereas *deepseek-coder* has the lowest scores (pass@1: 0.21, pass@10: 0.49).

RQ1: How do state-of-the-art local LLMs for code compare in terms of energy consumption and runtime efficiency?

According to Figures 2 and 4, there are significant variations in energy consumption across different models. The results highlight that energy usage is influenced by factors such as the specific architecture of each model, the programming languages used and the prompting methods adopted.

4.4 LLMs' Energy-Based Ranking

This section ranks LLMs on the HumanEval-X and MBPP+ benchmarks for both 0-shot and 3-shot settings, as shown in Table 1. Models are sorted by average energy consumption (ascending). For each LLM, we report runtime, mean pass@1 and pass@10 (averaged over HumanEval-X and MBPP+).

Table 1: Ranking of LLMs with Energy and Time, as well as Pass@1 and Pass@10 in Each Shot Setting

	Rank	LLM	Energy (J)	Time (s)	Pass@1	Pass@10
0-Shot	1	codellama	1742	26.5	0.101	0.299
	2	codegeex4	2543 _{1.46×}	38.9 _{1.47×}	0.138	0.197
	3	deepseek-coder	3973 _{2.28×}	60.8 _{2.30×}	0.143	0.326
	4	Meta-Llama-3	4408 _{2.53×}	67.3 _{2.54×}	0.113	0.245
	5	starling-lm	5417 _{3.11×}	82.9 _{3.14×}	0.156	0.420
3-Shot	1	codellama	1009	14.9	0.317	0.600
	2	deepseek-coder	1025 _{1.02×}	15.4 _{1.03×}	0.605	0.840
	3	starling-lm	1047 _{1.04×}	15.7 _{1.05×}	0.317	0.629
	4	Meta-Llama-3	1186 _{1.18×}	17.7 _{1.19×}	0.413	0.694
	5	codegeex4	1617 _{1.60×}	24.5 _{1.64×}	0.448	0.792

For models ranked second through fifth, additional energy and time relative to the most efficient model are shown as subscripts. For example, in the 0-shot ranking, *codellama* consumes an average of 1742 J and 26.5 s per task, achieving pass@1 = 0.10 and pass@10 = 0.30. The next model, *codegeex4*, requires 2543 J and 38.9 s (1.46× more energy and 1.47× more time than the most efficient model)

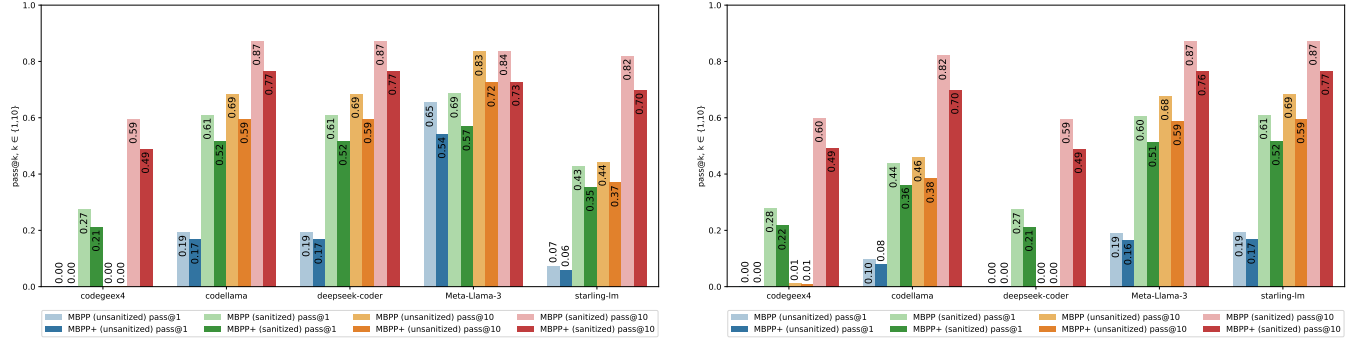


Figure 5: Pass@1 and Pass@10 Unsanitized and Sanitized for MBPP and MBPP+ in 0-Shot (left) and 3-Shot (right) settings

achieving a slightly higher pass@1 = 0.14 but a lower pass@10 = 0.20.

In the 0-shot setting, the most efficient model, *codellama*, achieves moderate correctness (pass@1 = 0.10, pass@10 = 0.30). By contrast, *starling-lm*, the most energy-demanding model, consumes about three times more energy (5417 J) and time (82.9 s), reaching higher correctness (pass@1 = 0.16, pass@10 = 0.42). In the 3-shot case, *codellama* again exhibits the lowest energy consumption (1009 J) and fastest runtime (14.9 s), with pass@1 = 0.32 and pass@10 = 0.60. The higher-energy model, *codegeex4* (1617 J, 24.5 s), achieves higher correctness (pass@1 = 0.45, pass@10 = 0.79).

These results show that higher correctness often comes at the cost of more energy and longer runtime. Users should balance accuracy and efficiency based on their needs.

RQ2: How does n-shot prompting affect energy consumption and runtime efficiency across models and tasks?

Table 1 shows that prompting type clearly affects LLM’s energy consumption. 3-shot prompting generally performs better than 0-shot and is therefore recommended. Guidelines are needed for writing prompts that balance performance and energy use, as the best prompt can vary across models. We encourage LLM providers to study these differences and offer best practices, ideally designing tools to make energy-efficient prompting easier. In summary, n-shot prompting can improve both performance and energy efficiency in code generation.

4.4.1 Detailed Analysis. Next, we illustrate key differences for each benchmark.

HumanEval-X: *codellama* is the most energy-efficient model in both n-shot settings. Although it does not lead in pass@1 and pass@10 (0-shot: pass@1 = 0.10, pass@10 = 0.30; 3-shot: pass@1 = 0.32, pass@10 = 0.60), its energy and time savings are notable. For example, in 0-shot, it uses about half the energy (2238 J) compared to *deepseek-coder* (4842 J). For Python tasks, it achieves good accuracy (pass@1 = 0.33, pass@10 = 0.67) at a much lower energy cost. However, in 3-shot, *deepseek-coder* is a strong alternative: it uses only about 1% more energy (1192 J) than *codellama* (1180 J) but delivers nearly double the pass@1 (0.61 vs 0.32 from *codellama*).

MBPP+: Again, *codellama* is the most energy-efficient model but does not achieve the highest pass@1 and pass@10 scores. In the

0-shot setting, *Meta-Llama-3* has the highest (sanitized) scores (pass@1 = 0.57, pass@10 = 0.73) but uses 2.47 times more energy and takes 2.48 times longer than *codellama*. In 3-shot, *Meta-Llama-3* (pass@1 = 0.51, pass@10 = 0.76) and *starling-lm* (pass@1 = 0.52, pass@10 = 0.77) reach similar scores but use 1.17 and 1.09 times more energy, respectively, than *codellama*. Therefore, *Meta-Llama-3* is preferred for code completion tasks because it is more energy-efficient and faster than *starling-lm* while delivering similar results.

5 Threats to Validity

Our study compares the energy use and performance of local LLMs on coding tasks, focusing on efficiency and accuracy, and discusses construct, internal, and external threats to validity.

Construct Validity: Benchmarking programming tasks does not always capture how developers actually use LLMs, since most benchmarks ignore the interactive and contextual aspects of real development. The benchmarks we used are mainly algorithmic and well-established, so LLMs often perform well on them, which may not reflect broader programming challenges. In addition, CodeCarbon measures total chip energy use rather than isolating LLM-specific consumption, meaning background processes can affect accuracy. To address this, our study minimized background noise through a controlled environment and limited concurrent processes. We used these standard benchmarks as a reference to enable consistent and widely accepted cross-model comparisons.

Internal Validity: Results may be influenced by factors beyond model differences, such as quantization methods or execution environments. Although all benchmarks ran under consistent conditions, some system-level variability is unavoidable. Still, CodeCarbon has shown reliability for comparative energy analysis on Linux, due to RAPL.

External Validity: We used quantized versions of all LLMs, which may reduce output quality. Additionally, we evaluated a fixed set of LLMs, but other models - with different architectures or quantization levels - could have been selected. Newer versions are now available, and using them might yield different results and rankings.

6 Conclusions

This study benchmarked five local LLMs on code completion and code generation tasks using HumanEval-X and MBPP+. We jointly

analyzed energy consumption, execution time and accuracy to uncover trade-offs between effectiveness and efficiency. The results show that model architecture and prompting style substantially affect both runtime and energy consumption, with 3-shot prompting generally offering better energy-time efficiency than 0-shot without compromising accuracy. These findings reinforce that prompt design and model selection must be considered together when deploying local LLMs. Beyond performance metrics, we highlight the need for clearer prompt-efficiency guidelines and model-specific best practices to help practitioners and LLM providers reduce computational waste. Our work thus contributes empirical evidence and practical recommendations toward more sustainable and efficient use of local LLMs for coding tasks.

Future Work: Our study suggests several directions for future research. Evaluating LLMs on broader benchmarks like RepoCod [46] and LiveCodeBench [36] could better capture real-world coding complexity. Developing methods to isolate LLM-specific energy use would improve sustainability assessments. Testing newer LLMs, different hardware and more quantization levels could generalize findings and track progress in energy-efficient code generation. Finally, examining GPU energy use and runtime during inference is important, as our study focused mainly on CPUs.

Replication Package

Repository: <https://doi.org/10.6084/m9.figshare.30777077>

Online Appendix: <https://sites.google.com/view/energy-local-llms-for-code/>

Acknowledgments

This work is funded by national funds through FCT - Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (<https://doi.org/10.54499/UID/50014/2025>). Simão Cunha is also funded by FCT grant 2025.04596.BDANA.

References

- [1] Negar Alizadeh, Boris Belchev, Nishant Saurabh, et al. 2024. Analyzing the Energy and Accuracy of LLMs in Software Development. arXiv:2412.00329 [cs.SE] <https://arxiv.org/abs/2412.00329>
- [2] Jacob Austin, Augustus Odena, Maxwell Nye, et al. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.PL]
- [3] Bartowski. 2024. codegeex4-all-9b-Q6_K_L. https://huggingface.co/bartowski/codegeex4-all-9b-GGUF/blob/main/codegeex4-all-9b-Q6_K_L.gguf
- [4] Bartowski. 2024. Meta-Llama-3-8B-Instruct-Q6_K. https://huggingface.co/bartowski/Meta-Llama-3-8B-Instruct-GGUF/blob/main/Meta-Llama-3-8B-Instruct-Q6_K.gguf
- [5] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] <https://arxiv.org/abs/2005.14165>
- [6] Tuan-Dung Bui, Thanh Trong Vu, Thu-Trang Nguyen, et al. 2025. Correctness Assessment of Code Generated by Large Language Models Using Internal Representations. arXiv:2501.12934 [cs.SE] <https://arxiv.org/abs/2501.12934>
- [7] Justine Calma. 2024. Microsoft's AI obsession is jeopardizing its climate ambitions. <https://www.theverge.com/2024/5/15/24157496/microsoft-ai-carbon-footprint-greenhouse-gas-emissions-grow-climate-pledge>. Accessed: January 24, 2025.
- [8] Fernando Castor. 2024. Estimating the Energy Footprint of Software Systems: a Primer. arXiv:2407.11611 [cs.SE] <https://arxiv.org/abs/2407.11611>
- [9] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, et al. 2023. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. arXiv:2310.14735 [cs.CL]
- [10] Mark Chen et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] <https://arxiv.org/abs/2107.03374>
- [11] Minyu Chen, Guoqiang Li, Ling-I Wu, et al. 2025. DCE-LLM: Dead Code Elimination with Large Language Models. arXiv:2506.11076 [cs.SE] <https://arxiv.org/abs/2506.11076>
- [12] CodeFuse. 2023. HumanEval-X. <https://github.com/codefuse-ai/codefuse-evaluation>. Accessed: January 10, 2025.
- [13] CodeParrot. 2024. InstructHumanEval. <https://huggingface.co/datasets/codeparrot/instructhumaneval>. Accessed: January 26, 2025.
- [14] Georges Da Costa and Jean-Marc Pierson. 2015. DVFS Governor for HPC: Higher, Faster, Greener. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 533–540. doi:10.1109/PDP.2015.73
- [15] Benoit Courty, Victor Schmidt, Sasha Luccioni, et al. 2024. *mlco2/codecarbon*: v2.4.1. doi:10.5281/zenodo.11171501
- [16] Luis Cruz, João Paulo Fernandes, Maja H. Kirkeby, et al. 2025. Greening AI-enabled Systems with Software Engineering: A Research Agenda for Environmentally Sustainable AI Practices. CoRR abs/2506.01774 (2025). doi:10.48550/ARXIV.2506.01774 arXiv:2506.01774
- [17] Simão Cunha. 2024. Adapted HumanEval-X Benchmark. <https://github.com/simaocunha71/codefuse-evaluation>. Forked from original HumanEval-X repository. Last commit SHA: 90b0148256c254caff6abc61dcada5d0c7555400.
- [18] Simão Cunha. 2024. Adapted MBPP+ Benchmark. <https://github.com/simaocunha71/evalplus>. Forked from original MBPP+ repository. Last commit SHA: f86db47fd3150b480376fe149c3ad047b7ff0b0.
- [19] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, et al. 2023. GitHub Copilot AI pair programmer: Asset or Liability? arXiv:2206.15331 [cs.SE] <https://arxiv.org/abs/2206.15331>
- [20] DeepSeek-AI, Qihao Zhu, Daya Guo, et al. 2024. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. arXiv:2406.11931 [cs.SE] <https://arxiv.org/abs/2406.11931>
- [21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] <https://arxiv.org/abs/1810.04805>
- [22] Francisco Durán, Matias Martinez, Patricia Lago, et al. 2024. Energy consumption of code small language models serving with runtime engines and execution providers. arXiv:2412.15441 [cs.SE] <https://arxiv.org/abs/2412.15441>
- [23] Zhangyin Feng, Daya Guo, Duyu Tang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv:2002.08155 [cs.CL]
- [24] Spandan Garg, Roshanak Zilouchian Moghaddam, Colin B. Clement, et al. 2022. DeepDev-PERF: a deep learning-based approach for improving software performance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Singapore, Singapore) (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 948–958. doi:10.1145/3540250.3549096
- [25] G. Gerganov. 2023. LLaMa.cpp: LLM Inference in C/C++. <https://github.com/ggerganov/llama.cpp>. Accessed: January 16, 2024.
- [26] GitHub. 2024. Survey: The AI wave continues to grow on software development teams. <https://github.blog/news-insights/research/survey-ai-wave-grows/>. Accessed: October 11, 2025.
- [27] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] <https://arxiv.org/abs/2407.21783>
- [28] Muhammad Usman Hadi, Qasem Al Tashi, Rizwan Qureshi, et al. 2023. A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage. *TechRxiv* (July 2023). doi:10.36227/techrxiv.23589741.v1
- [29] Marcus Hähnel, Björn Döbel, Marcus Völpl, et al. 2012. Measuring Energy Consumption for Short Code Paths Using RAPL. *SIGMETRICS Performance Evaluation Review* 40, 3 (Jan 2012), 13–17.
- [30] Dan Hendrycks, Collin Burns, Steven Basart, et al. 2020. Measuring Massive Multitask Language Understanding. CoRR abs/2009.03300 (2020). arXiv:2009.03300 <https://arxiv.org/abs/2009.03300>
- [31] Dan Hendrycks, Collin Burns, Saurav Kadavath, et al. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS* (2021).
- [32] Hugging Face. 2023. Efficient Inference on CPU. https://huggingface.co/docs/transformers/en/perf_infer_cpu. Accessed: October 11, 2025.
- [33] Intel. 2024. Learn How to Reduce Model Latency When Deploying Meta* Llama 3 on CPUs. <https://www.intel.com/content/www/us/en/developer/articles/technical/meta-llama-3-optimized-cpu-inference.html>. Accessed: October 11, 2025.
- [34] Intel Corporation. 2022. Running Average Power Limit (RAPL) Energy Reporting. <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html>
- [35] Maliheh Izadi, Jonathan Katzy, Tim van Dam, et al. 2024. Language Models for Code Completion: A Practical Evaluation. arXiv:2402.16197 [cs.SE] <https://arxiv.org/abs/2402.16197>
- [36] Naman Jain, King Han, Alex Gu, et al. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974 [cs.SE] <https://arxiv.org/abs/2403.07974>
- [37] Jean Kaddour, Joshua Harris, Maximilian Mozes, et al. 2023. Challenges and Applications of Large Language Models. arXiv:2307.10169 [cs.CL]
- [38] Kostis Kafes, Dragos Sbirlea, Yiyan Lin, et al. 2020. Leveraging application classes to save power in highly-utilized data centers. In *Proceedings of the 11th ACM Symposium on Cloud Computing (Virtual Event, USA) (SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 134–149. doi:10.1145/3419111.3421274

- [39] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, et al. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3, 2, Article 9 (mar 2018).
- [40] B. Kumar and Md Shaheer Ahmed. 2024. Beyond Clouds: Locally Runnable LLMs as a Secure Solution for AI Applications. *Digital Society* 3 (10 2024). doi:10.1007/s44206-024-00141-y
- [41] Eldar Kurtic, Denis Kuznetsov, Elias Frantar, Michael Goin, and Dan Alistarh. 2023. Sparse Fine-tuning for Inference Acceleration of Large Language Models. arXiv:2310.06927 [cs.CL] <https://arxiv.org/abs/2310.06927>
- [42] Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, et al. 2024. MELting Point: Mobile Evaluation of Language Transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking* (Washington D.C., DC, USA) (*ACM MobiCom '24*). Association for Computing Machinery, New York, NY, USA, 890–907. doi:10.1145/3636534.3690668
- [43] Bowen Li, Wenhan Wu, Ziwei Tang, et al. 2024. Prompting Large Language Models to Tackle the Full Software Development Lifecycle: A Case Study. arXiv:2403.08604 [cs.CL] <https://arxiv.org/abs/2403.08604>
- [44] Kefan Li and Yuan Yuan. 2024. Large Language Models as Test Case Generators: Performance Evaluation and Enhancement. arXiv:2404.13340 [cs.SE] <https://arxiv.org/abs/2404.13340>
- [45] Linyang Li, Pengyu Wang, Ke Ren, et al. 2023. Origin Tracing and Detecting of LLMs. arXiv:2304.14072 [cs.CL] <https://arxiv.org/abs/2304.14072>
- [46] Shanchao Liang, Yiran Hu, Nan Jiang, et al. 2025. Can Language Models Replace Programmers for Coding? REPOCOD Says 'Not Yet'. arXiv:2410.21647 [cs.SE] <https://arxiv.org/abs/2410.21647>
- [47] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, et al. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Thirty-seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=1qv610Cu7>
- [48] Sasha Luccioni, Yacine Jernite, and Emma Strubell. 2024. Power Hungry Processing: Watts Driving the Cost of AI Deployment?. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24)*. ACM, 85–99. doi:10.1145/3630106.3658542
- [49] Aman Madaan, Niket Tandon, Prakhar Gupta, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL] <https://arxiv.org/abs/2303.17651>
- [50] Samuel Miller, Daking Rai, and Ziyu Yao. 2025. Mechanistic Understanding of Language Models in Syntactic Code Completion. arXiv:2502.18499 [cs.SE] <https://arxiv.org/abs/2502.18499>
- [51] Tim Mucci. 2024. GGUF versus GGML. <https://www.ibm.com/think/topics/gguf-versus-ggml>
- [52] Niklas Muennighoff, Qian Liu, Armel Zebaze, et al. 2023. OctoPack: Instruction Tuning Code Large Language Models. arXiv preprint arXiv:2308.07124 (2023).
- [53] Seonjin Na, Geonhwa Jeong, Byung Hoon Ahn, et al. 2024. Understanding Performance Implications of LLM Inference on CPUs. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. 169–180. doi:10.1109/IISWC63097.2024.00024
- [54] Humza Naveed, Asad Ullah Khan, Shi Qiu, et al. 2023. A Comprehensive Overview of Large Language Models. arXiv:2307.06435 [cs.CL]
- [55] OpenAI. 2025. ChatGPT. <https://chat.openai.com>
- [56] OpenAI et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] <https://arxiv.org/abs/2303.08774>
- [57] Rui Pereira, Marco Couto, Francisco Ribeiro, et al. 2021. Ranking programming languages by energy efficiency. *Science of Computer Programming* 205 (2021), 102609. doi:10.1016/j.scico.2021.102609
- [58] Tobias Leonhard Joschka Peslitz and Bastian Katz. 2025. Educated Energy Efficiency Optimization of Distributed Software: Measure, Monitor, Mitigate. In *2025 IEEE/ACM 9th International Workshop on Green and Sustainable Software (GREENS)*. 20–27. doi:10.1109/GREENS66463.2025.00008
- [59] Guillaume Raffin and Denis Trystram. 2024. Dissecting the software-based measurement of CPU energy consumption: a comparative analysis. arXiv:2401.15985 [cs.DC] <https://arxiv.org/abs/2401.15985>
- [60] Francisco Ribeiro, Rui Abreu, and João Saraiva. 2022. Framing Program Repair as Code Completion. In *2022 IEEE/ACM International Workshop on Automated Program Repair (APR)*. 38–45. doi:10.1145/3524459.3527347
- [61] Francisco Ribeiro, José Nuno Castro de Macedo, Kanae Tsushima, et al. 2023. GPT-3-Powered Type Error Debugging: Investigating the Use of Large Language Models for Code Repair. In *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering (Cascas, Portugal) (SLE 2023)*. Association for Computing Machinery, New York, NY, USA, 111–124. doi:10.1145/3623476.3623522
- [62] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, et al. 2012. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 32, 2 (2012), 20–27.
- [63] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, et al. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL] <https://arxiv.org/abs/2308.12950>
- [64] Sofia Santos, João Saraiva, and Francisco Ribeiro. 2024. Large Language Models in Automated Repair of Haskell Type Errors. In *Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair (Lisbon, Portugal) (APR '24)*. Association for Computing Machinery, New York, NY, USA, 42–45. doi:10.1145/3643788.3648012
- [65] Sebastian Scholz, Alexander Lawall, and Kristina Schaaff. 2024. The Impact of Large Language Models on IT Security in the Corporate Environment. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. 109–115. doi:10.1109/FLLM63129.2024.10852476
- [66] Jieke Shi, Zhou Yang, Hong Jin Kang, et al. 2024. Greening Large Language Models of Code. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS'24)*. ACM, 142–153. doi:10.1145/3639475.3640097
- [67] Jieke Shi, Zhou Yang, Hong Jin Kang, et al. 2024. Greening Large Language Models of Code. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society (Lisbon, Portugal) (ICSE-SEIS'24)*. Association for Computing Machinery, New York, NY, USA, 142–153. doi:10.1145/3639475.3640097
- [68] Jieke Shi, Zhou Yang, and David Lo. 2025. Efficient and Green Large Language Models for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 137 (May 2025), 22 pages. doi:10.1145/3708525
- [69] Karan Singhal, Shekoofeh Azizi, Tao Tu, et al. 2022. Large Language Models Encode Clinical Knowledge. arXiv:2212.13138 [cs.CL] <https://arxiv.org/abs/2212.13138>
- [70] Dominik Sobania, Martin Briesch, Carol Hanna, et al. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. arXiv:2301.08653 [cs.SE] <https://arxiv.org/abs/2301.08653>
- [71] Balázs Szalontai, Gergő Szalay, Tamás Márton, et al. 2024. Large Language Models for Code Summarization. arXiv:2405.19032 [cs.AI] <https://arxiv.org/abs/2405.19032>
- [72] Tom Jobbins. 2023. codellama-7b-instruct.Q5_K_M. https://huggingface.co/TheBloke/Codellama-7B-Instruct-GGUF/blob/main/codellama-7b-instruct.Q5_K_M.gguf
- [73] Tom Jobbins. 2023. deepseek-coder-6.7b-instruct.Q5_K_M. https://huggingface.co/TheBloke/deepseek-coder-6.7b-instruct-GGUF/blob/main/deepseek-coder-6.7b-instruct.Q5_K_M.gguf
- [74] Tom Jobbins. 2023. starling-lm-7b-alpha.Q5_K_S. https://huggingface.co/TheBloke/Starling-LM-7B-alpha-GGUF/blob/main/starling-lm-7b-alpha.Q5_K_S.gguf
- [75] Weixi Tong and Tianyi Zhang. 2024. CodeJudge: Evaluating Code Generation with Large Language Models. arXiv:2410.02184 [cs.LG] <https://arxiv.org/abs/2410.02184>
- [76] Tina Vartziotis, Ippolyti Dellatolas, George Dasoulas, et al. 2024. Learn to Code Sustainably: An Empirical Study on Green Code Generation. In *Proceedings of the 1st International Workshop on Large Language Models for Code (Lisbon, Portugal) (LLM4Code '24)*. Association for Computing Machinery, New York, NY, USA, 30–37. doi:10.1145/3643795.3648394
- [77] Roberto Verdecchia, June Sallou, and Luis Cruz. 2023. A Systematic Review of Green AI. arXiv:2301.11047 [cs.AI]
- [78] Zhongwei Wan, Xin Wang, Che Liu, et al. 2024. Efficient Large Language Models: A Survey. arXiv:2312.03863 [cs.CL] <https://arxiv.org/abs/2312.03863>
- [79] Claes Wohlin, Per Runeson, Martin Hst, et al. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.
- [80] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Singapore, Singapore) (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 959–971. doi:10.1145/3540250.3549101
- [81] Lui Yoshida. 2024. *The Impact of Example Selection in Few-Shot Prompting on Automated Essay Scoring Using GPT Models*. Springer Nature Switzerland, 61–73. doi:10.1007/978-3-031-64315-6_5
- [82] Wayne Xin Zhao, Kun Zhou, Junyi Li, et al. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] <https://arxiv.org/abs/2303.18223>
- [83] Qinkai Zheng, Xiao Xia, Xu Zou, et al. 2023. CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Benchmarking on HumanEval-X. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (Long Beach, CA, USA) (KDD '23)*. Association for Computing Machinery, New York, NY, USA, 5673–5684. doi:10.1145/3580305.3599790