Retrieval-Augmented Generation in Software Development: Balancing Effectiveness and Cost

Tănăsescu Ștefan
University of Twente
s3764869
s.tanasescu@student.utwente.nl

Piyush Singh
University of Twente
s3056139
p.k.singh@student.utwente.nl

ABSTRACT

Large Language Models (LLMs) are increasingly integral to software development, yet concerns about their accuracy and energy consumption persist. Retrieval-Augmented Generation (RAG) is a prominent technique intended to improve LLM accuracy by grounding outputs in external knowledge, but this enhancement introduces computational and energy overhead. This study empirically evaluates the trade-offs of RAG in code generation by comparing the performance and sustainability of two models (Llama-3.1-8B and Starcoder2-7B) under baseline (No RAG) and RAG (k=3) conditions. The experiment was conducted on two distinct hardware platforms, an Apple M4 Pro (Mac) and an NVIDIA RTX 4070 SUPER (Windows), using a workload of 100 programming tasks. Our findings indicate that the RAG configuration, using a general-purpose Python documentation corpus, failed to improve code correctness (pass@1 rate) and, in the case of the Llama model, slightly decreased it on both platforms. Concurrently, RAG significantly increased all cost metrics: on the Mac, it increased average energy per task by 198% for Llama and 148% for Starcoder, and latency by 155% and 107%, respectively. We also identified a clear hardware trade-off: the Mac platform was 3.8x more energy-efficient (Llama No RAG), while the Windows platform was 2.7x faster. We conclude that for this task domain, the substantial sustainability costs of RAG far outweigh its negligible (and in some cases, negative) effectiveness benefits.

Keywords

Retrieval-Augmented Generation; Large Language Models; Sustainable Software Engineering, Energy Efficiency, Code Verification

1. INTRODUCTION

Large Language Models have rapidly transformed software development by enabling automated code generation, repository verification, and intelligent code completion. Despite these advances, their limitations in accuracy, adaptability, and sustainability have led to interest in Retrieval-Augmented Generation as a complementary paradigm. RAG enhances generative models with external knowledge retrieval, producing outputs that are contextually relevant, up-to-date, and grounded in authoritative sources [2].

In software engineering, where correctness, efficiency, and sustainability are paramount, the integration of RAG presents a critical trade-off. While LLMs show strong potential for code generation and verification, they often suffer from accuracy issues and consume considerable resources. RAG addresses these limitations by incorporating external knowledge retrieval, which has been shown to significantly improve repository-level verification success rates and mitigate vulnerabilities in generated code [4]. However, this enhancement introduces new computational overheads, including increased latency and energy costs. The central problem this study addresses is whether these benefits in correctness and security justify the additional sustainability and cost implications, a key concern for practitioners and researchers in sustainable software engineering.

The purpose of this study is to investigate the effectiveness and cost-efficiency of RAG in software development workflows. This will be achieved by comparing RAG-enhanced systems with baseline LLMs, evaluating both performance improvements and sustainability trade-offs.

Goal 1: Analyze the effect of RAG on code correctness.

RQ1: *Does retrieval improve correctness compared to baseline LLMs?*

Metric: pass@1 rate.

Goal 2: Analyze the cost implications of RAG.

RQ2: *How do retrieval depths impact computational, economic, and energy costs compared to baseline?*

Metrics: latency (s), GPU-hours, token usage, joules consumed.

Goal 3: Identify sustainable configurations.

RQ3: *Which retrieval depth provides the best trade-off between correctness and cost?*

Metric: energy per solved task (joules per successful pass@1).

In this study, effectiveness is defined as code correctness, measured by the pass@1 rate, the percentage of programming tasks for which the generated solution passes all official unit tests on the first attempt. Cost is decomposed into two measurable aspects: (i) computational cost, quantified in terms of GPU-hours, latency (seconds), and token usage, and (ii) economic cost, estimated by mapping GPU-hours to standard cloud pricing models. Sustainability is operationalized as energy consumption (joules per inference) and derived CO_2 emissions, calculated from platform-level energy counters.

The target audience for this investigation includes software developers, engineering teams, and researchers working at the intersection of artificial intelligence and sustainable computing. Developers and practitioners are expected to benefit from evidence-based insights on when RAG integration enhances productivity and reliability enough to offset its higher resource demands. Researchers, on the other hand, can leverage the

Author's contributions: All work was split evenly with both authors unless otherwise mentioned. Stefan ran the Mac experiments and wrote the corresponding results section. Piyush ran the Windows experiments and wrote the corresponding results section and created the presentation

findings to advance discourse on sustainable AI and identify future directions for optimizing retrieval pipelines.

2. CONTEXT

The adoption of Retrieval-Augmented Generation in software engineering must be understood within a broader sustainability framework. Following the five-dimension model of sustainability: environmental, economic, social, technical, and individual, this section outlines the context in which RAG operates.

2.1 Environmental

RAG systems require additional retrieval steps compared to baseline LLMs, increasing computational intensity and energy usage. Each retrieval call contributes to a higher carbon footprint per inference, especially when operating at scale in industry settings. This raises concerns about the environmental sustainability of widespread RAG adoption, as the gains in correctness and reliability may come at the cost of greater greenhouse gas emissions. A critical question is whether the accuracy improvements justify this increased energy demand.

2.2 Economics

Beyond energy costs, RAG introduces tangible financial overheads. GPU-hours consumed by retrieval pipelines, additional API calls for external knowledge bases, and higher infrastructure demands all translate to higher operational expenses. For companies, this means that integrating RAG may only be viable if the productivity and reliability benefits outweigh the recurring computational costs. The economic dimension therefore highlights a trade-off between cost-efficiency and performance.

2.3 Social

RAG also has social implications for sustainable software development. By producing more secure and reliable code, RAG reduces the likelihood of vulnerabilities that could negatively impact users and society at large. Furthermore, more accurate systems may alleviate the risks of large-scale software failures, which carry both financial and societal consequences. In this sense, RAG contributes to the social sustainability of software systems by strengthening trust and reliability in digital infrastructures.

2.4 Technical

From a technical standpoint, RAG responds to the limitations of LLMs by improving adaptability and grounding. It enables systems to handle evolving knowledge domains, increasing their long-term maintainability. However, the additional complexity of retrieval pipelines introduces new technical challenges, such as optimizing retrieval latency, managing large document stores, and preventing context overload. Thus, the technical sustainability of RAG depends on whether these added layers can be managed efficiently without creating new bottlenecks.

2.5 Individual

Finally, RAG has implications for individual developers. On the positive side, better retrieval-augmented outputs may reduce time spent debugging or fixing security issues, making development workflows more sustainable at the individual level. However, the increased system complexity may also pose barriers to adoption, requiring developers to learn new workflows and manage more intricate systems. The impact on individual sustainability therefore hinges on how seamlessly RAG integrates into existing developer practices.

3. BACKGROUND AND RELATED WORK

RAG has been widely studied as a solution to the limitations of LLMs. Standard models are prone to hallucinations and struggle with dynamic knowledge domains, issues that RAG mitigates by incorporating external retrieval. In program verification, RagVerus demonstrated how repository-level RAG methods can improve proof synthesis in large, interdependent codebases [1]. It achieved a 27% relative improvement on the RepoVBench benchmark, establishing retrieval as an effective tool for verification scalability.

Security-focused applications such as SOSecure showed that RAG can substantially improve the safety of LLM-generated code [3]. By retrieving vulnerability-related discussions from Stack Overflow, SOSecure achieved fix rates of up to 96.7%, outperforming conventional prompting and highlighting the role of community knowledge in secure code generation. Furthermore, benchmarking efforts like CODERAG-Bench further illustrate RAG's potential and challenges [6]. While retrieval improved accuracy in basic and open-domain programming tasks, repository-level performance gains were modest, revealing efficiency bottlenecks in retrieval pipelines.

Optimization frameworks have been proposed to reduce token usage and hardware load, often with contextual compression filters [7], [8]. Hyperparameter optimization across RAG pipelines also proved essential for balancing accuracy, latency, and cost, emphasizing that optimal configurations may vary across applications.

The sustainability perspective is particularly relevant in this course. Wu et al. introduced the "Sustainable AI Trilemma," showing that memory-augmented RAG frameworks increase inference energy consumption, raising concerns about environmental sustainability [5]. This aligns with the course goal of critically evaluating the energy and resource implications of software systems.

Recent innovations such as ProCC demonstrate the potential of multi-retrieval strategies to further enhance RAG performance. By integrating multiple retrieval perspectives with adaptive selection algorithms, ProCC outperformed single-retriever systems in code completion tasks, suggesting promising directions for improving both effectiveness and efficiency.

While previous work has explored retrieval in code verification (e.g., RagVerus) and secure code generation (SOSecure), and Wu et al. introduced the "Sustainable AI Trilemma" addressing energy trade-offs, these studies leave a gap, the performance of base models like Llama and Starcoder remains a foundational concern. The StarCoder team itself, in their StarCoder2 paper (Lozhkov et al., 2024), benchmarks their models on tasks including HumanEval and MBPP [12

], establishing a baseline for code-specific model performance. In particular, no prior study has systematically compared different retrieval depths under realistic workloads across heterogeneous platforms (NVIDIA vs. Apple Silicon). This study explicitly addresses that gap by combining correctness, cost, and sustainability measurements in a single experimental framework.

4. METHODOLOGY AND EXPERIMENT PLANNING

The overarching goal of this study is to analyze the trade-offs between effectiveness and cost-efficiency of retrieval-augmented generation systems in software engineering code tasks under an inference-based setup. Specifically, the investigation pursues three objectives: first, to quantify improvements in code correctness achieved by RAG-enhanced large language models compared to baseline models; second, to measure the computational, economic, and environmental costs associated with retrieval pipelines during inference; and third, to identify retrieval configurations that achieve a practical balance between accuracy gains and sustainability metrics.

Based on the research questions, we define hypotheses at both general and specific levels:

Main Hypothesis (H1): *RAG improves correctness of code generation tasks sufficiently to justify its additional computational and energy costs.*

Null Hypothesis (H0): *RAG does not improve correctness enough to compensate for added cost.*

For each research question:

- RQ1 Correctness:

H0: *There is no significant difference in pass@1 rates between baseline and RAG conditions.*

H1: *RAG significantly improves pass@1 rates compared to baseline.*

- RQ2 Cost:

H0: *RAG does not significantly increase computational cost (latency, GPU-hours, energy).*

H1: *RAG significantly increases computational cost compared to baseline.*

- RQ3 Trade-off:

H0: *No retrieval depth achieves a better efficiency ratio (joules per solved task) than baseline.*

To address these objectives, the study adopts a comparative experimental design with standardized benchmarks. Code correctness will be assessed as the percentage of tasks successfully solved, using pass@1 as the primary metric, whereby a task is considered correct only if all official unit tests are passed on the first attempt. Cost will be examined through several dimensions: computational time (latency in seconds), GPU-hours consumed, token usage per task, and energy expenditure derived from device-level power measurements. Sustainability will be evaluated by normalizing energy and economic costs against performance improvements, reporting efficiency indicators such as joules per solved task.

Two open-weight, locally executable models were chosen to represent different usage profiles: a code-tuned model (StarCoder2-7B-Instruct, run as starcoder2:instruct) and a general-purpose model (Llama-3.1-8B-Instruct, run as llama3.1:8b). The experimental units (the workload) consist of 100 programming problems sampled from HumanEval and MBPP under a fixed, reproducible sampling procedure. We selected 100 tasks to ensure diversity across algorithmic categories while keeping runtime within practical limits. We chose the number of 100 tasks in order to ensure that we can make multiple runs, including preparation and test runs.

Each task will be executed under two conditions: baseline inference without retrieval (k = 0), and RAG-enhanced inference where top-k documents are prepended as context (k = 3). The retrieval pipeline employs a lightweight lexical search (BM25) over a local documentation corpus composed of 313 documents drawn from the official Python documentation, specifically the py_topics (Python language tutorials and guides) and stdlib_docs (standard library API references) collections. This corpus, while providing foundational Python programming knowledge, was not directly tailored to the algorithmic problem-solving nature of the HumanEval and MBPP benchmark tasks. To prevent data leakage, known or near-duplicate ground-truth solutions for sampled tasks are excluded from the corpus. Retrieval depth is the sole variable across conditions, with decoding parameters fixed to isolate the impact of retrieval.

Independent variables:

- Model type
- Retrieval depth
- Hardware platform

Dependent variables:

- Code correctness
- Latency
- Energy consumption
- Token usage
- Cost-efficiency

Experiments will be conducted on two representative hardware platforms: a Windows/NVIDIA desktop and a macOS Apple Silicon laptop. This dual setup enables a comparative evaluation of everyday environments while also contrasting efficiency between Nvidia and Apple Silicon architectures. Each run follows a consistent procedure: warm-up, start of logging, generation of a candidate solution, end of logging, and execution of official unit tests. Power usage will be sampled at runtime using platform-appropriate tools: on Nvidia, SMI will be employed to record CPU/GPU power draw and estimate CO_2 emissions, on Apple Silicon, equivalent system-level energy counters will be used. Energy (joules) will be derived by integrating power over the inference interval.

Because instrumentation differs across platforms, the analysis emphasizes within-platform comparisons. Results will be normalized by task characteristics, including token usage, and interpreted primarily as relative changes between baseline and RAG variants. For each task, baseline results will be paired against retrieval conditions to compare correctness, latency, and energy usage. To mitigate randomness without significantly inflating runtime, each (task × model × condition × platform) configuration will be repeated 7 times (once for system stabilization and 6 times for data recoding), averaged, and reported with confidence intervals. This number was chosen to ensure a stable distribution for statistical testing and to account for

transient system variations, while considering runtime and practical implications.

The design is calibrated to complete the full experiment within approximately 40 hours, while still offering enough variation to identify meaningful differences across models, retrieval depths and platforms. Internal validity is supported by stable execution environments, fixed sampling procedures, and repeated trials. External validity is limited by the use of relatively small-scale models and a workload of 100 tasks, but the setup reflects realistic constraints and common usage scenarios.

The expected outcome of this study is a comprehensive comparative analysis of RAG versus baseline LLMs, revealing the extent to which retrieval improves code correctness and under what conditions these gains outweigh additional costs. The findings aim to clarify when RAG adoption is justified, and which configurations minimize inefficiencies while maximizing effectiveness. For software developers and industry practitioners, the study will provide evidence-based guidance on whether retrieval integration meaningfully enhances productivity. For researchers and students in sustainable software engineering, it offers a case study on balancing performance gains against environmental and economic costs, contributing to the broader discourse on sustainable AI.

5. MEASUREMENT TOOLS, ENVIRONMENT, AND PROCEDURE

The experiment will be conducted on two distinct hardware platforms in order to compare the energy efficiency of LLM tasks with and without Retrieval-Augmented Generation (RAG). The first platform is an Apple MacBook Pro (Model Identifier: Mac16,8) equipped with an Apple M4 Pro chip that integrates 12 CPU cores (8 performance and 4 efficiency), 24 GB of unified memory, and macOS with firmware version 13822.40.85. The second platform is a desktop machine running Windows 11 version 24H2, featuring an AMD Ryzen 7 5800X3D processor with 8 cores and 16 threads operating at 3.4–4.5 GHz, 32 GB of DDR4 memory clocked at 3200 MT/s, and an NVIDIA GeForce RTX 4070 SUPER GPU with 12 GB of VRAM. This machine uses CUDA-enabled drivers (version 581.29) to accelerate GPU workloads. Both systems will remain connected to mains power to ensure stable measurements unaffected by battery discharge states.

Energy usage will be recorded differently on the two platforms, reflecting available tooling. On the MacBook Pro, the powermetrics utility will be employed. This system-level profiler is capable of sampling CPU, GPU, and DRAM energy consumption at configurable intervals, and is recommended in the course material as the standard approach for Apple Silicon. The rationale for selecting powermetrics is its native integration with macOS, low overhead, and sufficient granularity for profiling inference workloads.

On the Windows desktop, NVIDIA's nvidia-smi interface will be used to log GPU power draw in real time at 0.5-second intervals. For CPU power monitoring, we employ a custom Python script utilizing the psutil library to estimate CPU power consumption based on processor utilization. The estimation uses a linear model between idle power (~30W) and the processor's thermal design power (TDP of 105W for the AMD Ryzen 7 5800X3D), scaled by real-time CPU utilization percentage. While this approach provides power estimates rather than direct hardware measurements as AMD processors do not expose RAPL-equivalent interfaces on Windows, it enables consistent relative comparisons across experimental conditions. Total system power is computed as the sum of GPU power and estimated CPU power, sampled at 0.5-second intervals throughout each inference task.

The environment definition includes reporting of processor frequencies, available memory, and runtime systems to ensure reproducibility and to contextualize the results. On macOS, the LLM experiments will be executed through Python scripts configured with the appropriate inference frameworks optimized for Apple Silicon, while the Windows platform will use CUDA-accelerated inference pipelines. No non-default compiler or optimization flags will be applied beyond standard runtime library dependencies, to maintain fairness across the two conditions.

The execution procedure will follow a controlled design. Before each trial, the model will be loaded into memory and a warm-up prompt will be issued, ensuring that initialization costs do not distort measurements. Once stabilized, the benchmark prompts, drawn from HumanEval and MBPP, will be executed under two treatments: baseline LLM inference and RAG-enhanced inference. Each execution will be synchronized with the measurement tools by starting logging immediately prior to inference and stopping once the output has been generated. This ensures alignment between workload and energy profiling.

To control variability, each workload will be executed multiple times per trial (minimum of 6) and across both hardware platforms, enabling statistical comparison. Sampling rates will be set to 0.5 seconds on macOS and the same for Windows, balancing resolution and overhead. Raw logs will be parsed to extract joules consumed per inference, normalized against correctness rates and tokens generated. To reduce measurement noise, background tasks and non-essential services will be disabled, and both systems will be kept in consistent thermal and power states.

By adopting this measurement setup, the experiment ensures that the collected data stems directly from the observed inference workloads rather than from instrumentation artifacts. The combined use of power metrics on Apple Silicon and NVIDIA/OS-level telemetry on Windows provides a sufficiently precise and context-aware estimation of energy consumption to address the research questions on the sustainability trade-offs of RAG.

6. THREATS TO VALIDITY

As with any empirical study, the validity of the results obtained in this experiment is subject to limitations. Following the established taxonomy in software engineering, the threats are categorized into construct, internal, external, and conclusion validity.

Construct validity concerns whether the selected metrics truly capture the phenomena under investigation. While both platforms allow for measuring GPU power, comparing total system consumption is complex... To mitigate this, our primary cross-platform comparison will focus on GPU-specific energy consumption, while total system consumption will be analyzed primarily within each platform.

Internal validity relates to factors that could bias the experimental results. On macOS, background processes and system-level tasks might affect powermetrics readings, while on Windows, GPU telemetry could be influenced by driver-level optimizations. Thermal throttling is another potential confounder, as inference workloads may heat up components and trigger frequency scaling. To address these issues, experiments will be conducted on idle systems with unnecessary services disabled, and warm-up runs will be used to stabilize the thermal state before measurements. Furthermore, repeated trials (minimum of 6 per condition) will help account for transient variations.

External validity addresses the generalizability of the findings. The study compares only two platforms: Apple Silicon M4 Pro and an NVIDIA RTX 4070 SUPER system. Results may not extend to other GPUs, CPUs, or deployment environments such as cloud servers. Similarly, the workloads chosen may not fully reflect real-world software engineering use cases. Nevertheless, they are widely used in benchmarking and thus provide a meaningful starting point. Future work could extend the analysis to additional datasets and hardware platforms to strengthen external validity.

Conclusion validity is concerned with the correctness of statistical inferences. Given that energy consumption can vary across runs due to measurement noise and system-level nondeterminism, the risk of errors exists. To mitigate this, the experiment will collect a sufficiently large sample size for each condition, apply appropriate statistical tests, and report confidence intervals alongside mean values. Effect sizes will also be considered to ensure that statistical significance aligns with practical relevance.

All results will be analyzed with statistical tests. Data distributions will first be checked for normality. Confidence intervals and effect sizes will accompany all reported averages to ensure statistical robustness. While these measures do not eliminate threats to validity, they provide a transparent account of potential limitations and the steps taken to mitigate them. This transparency ensures that the results can be interpreted appropriately within the scope of the study.

7. Results

7.1 Mac OS

This section presents the results from the experiment conducted on the Apple MacBook Pro (M4 Pro) platform. The data was collected over 6 full runs for each of the four configurations (Llama k=0, Llama k=3, Starcoder k=0, Starcoder k=3), with each run processing 100 tasks. The findings are presented in order of the research questions.

7.1.1 RQ1: Effectiveness (Correctness)

To answer RQ1: *Does retrieval improve correctness compared to baseline LLMs?,* we measured the pass@1 rate, representing the percentage of tasks solved correctly on the first attempt.

The results, as summarized in Table 1 and visualized in Figure 1, show that RAG at k=3 did not improve the pass@1 rate for either model. For the Llama model, the baseline (No RAG) configuration achieved the highest correctness at 52.00% (±1.41). Enabling RAG caused a slight decrease in correctness to 47.83% (±1.17). For the Starcoder model, the pass@1 rate was unaffected by RAG, remaining constant at 42.00% (±0.00) for both the baseline and RAG configurations.

Table 1: Model Correctness (pass@1) on Apple Silicon

Model	Condition	Pass@1 Rate (Avg)
Llama	k=0	52.00%
Llama	RAG (k=3	47.83%
Starcoder	k=3	42.00%
Starcoder	k=3	42.00%

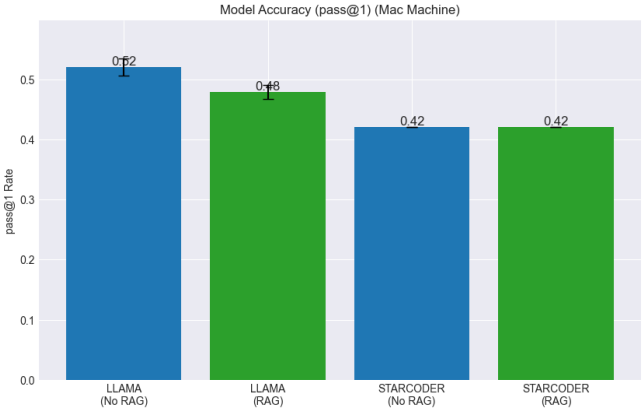

Figure 1: Model Accuracy (pass@1) on Apple Silicon

7.1.2 RQ2: Cost (Computational)

To answer RQ2: *How do retrieval depths impact computational, economic, and energy costs compared to baseline?*, we measured token usage, average power draw, energy consumption per task, and latency. The data in Table 2 shows that RAG substantially increased every cost metric.

The context from RAG dramatically increased the number of tokens processed. Llama saw a 7.1x increase in average tokens (163 to 1167), and Starcoder saw a 5.8x increase (242 to 1418), as shown in Figure 2. This also implies that energy per task increased significantly. As seen in Figure 3, Llama's average energy per task rose by 198% (from 0.50J to 1.49J). Starcoder's energy per task rose by 148% (from 1.42J to 3.53J).

RAG models also drew more power on average during inference. Llama's power draw increased by 17.6% (22.25W to 26.16W) and Starcoder's by 19.9% (19.06W to 22.85W). Figure 7 plots the power draw over time, illustrating that RAG runs (dashed lines)

had a higher and more sustained power draw.

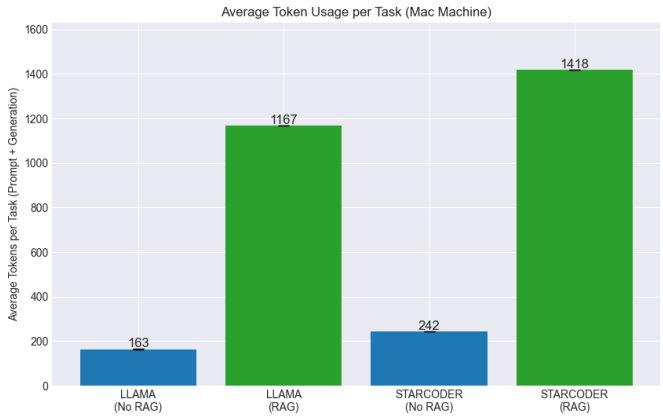

Figure 2: Average Token Usage per Task (Prompt + Generation) on Apple Silicon

Table 2: Computational Costs per Task on Apple Silicon (Mean ± StdDev)

Model	Condition	Avg Tokens	Avg Power (W)	Avg Energy (J)
Llama	k=0	163 (±1)	22.25 (±0.14)	49.64 (±5.03)
Llama	k=3	1167 (±1)	26.16 (±0.10)	148.77 (±6.97)
Starcoder	k=0	242 (±0)	19.06 (±0.05)	142.15 (±0.48)
Starcoder	k=3	1418 (±0)	22.85 (±0.03)	353.26 (±0.44)

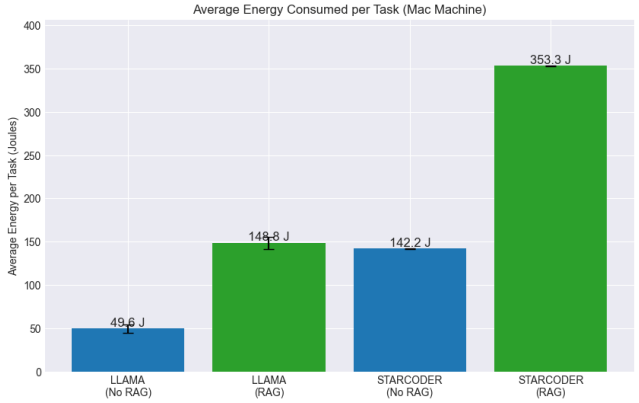

Figure 3: Average Energy Consumed per Task (Joules) on Apple Silicon

We also measured the time-based costs, specifically the average wall-clock time to complete a single task and the total time required to process the full 100-task workload. The results, shown in Table 3 and visualized in Figure 4, demonstrate a significant time cost associated with both RAG and the Starcoder model.

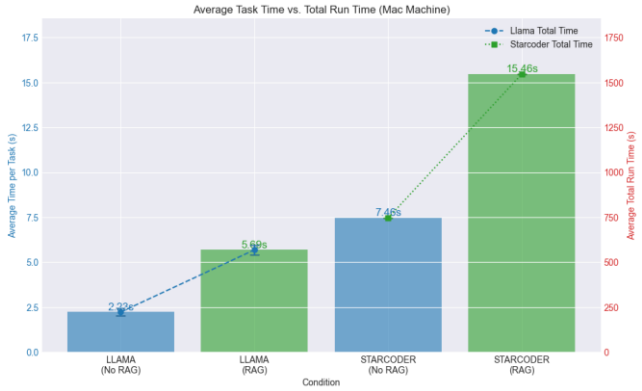

Figure 4: Average Task Time vs. Total Run Time (Mac Machine)

Table 3: Time and Latency Summary on Apple Silicon

Model	Condition	Avg Time / Task (s)	Avg Total Time / Run (s)
Llama	k=0	*2.23 (± 0.22)*	223.00 (± 21.56)
Llama	k=3	5.69 (± 0.27)	568.67 (± 26.85)
Starcoder	k=0	7.46 (± 0.01)	745.83 (± 0.75)
Starcoder	k=3	5.46 (± 0.00)	1546.00 (± 0.00)

The baseline Starcoder model was 3.3x slower per task than the baseline Llama model (7.46s vs 2.23s). Enabling RAG increased Llama's average time per task by 155% (from 2.23s to 5.69s). Enabling RAG increased Starcoder's average time per task by 107% (from 7.46s to 15.46s).

These costs accumulated significantly. The Llama (No RAG) configuration was the fastest, completing the workload in an average of 223 seconds (3.7 minutes). The Starcoder (RAG) configuration was the slowest, requiring 1546 seconds (25.8 minutes) for the same 100 tasks.

In addition to per-task metrics, we measured the total energy consumed to complete the entire 100-task workload for each configuration. Figure 5 presents this data, averaging over the experiment runs.

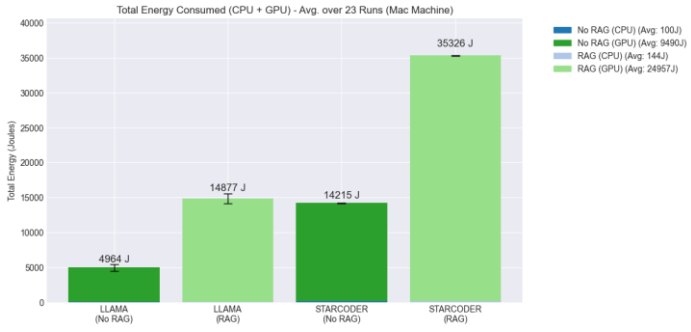

Figure 5: Total Energy Consumed (CPU + GPU) - Avg. over Runs (Mac Machine)

The results show the significant cumulative cost of RAG and the baseline differences between the models. The Llama (No RAG) configuration was the most energy-frugal, consuming only 49.64 J (± 5.03) in total to process all 100 tasks. The baseline Starcoder

model consumed 142.15 J (± 0.48), approximately 2.86 times more energy than the baseline Llama model for the same workload.

Enabling RAG increased Llama's total energy consumption by 199.7% to 148.77 J (± 6.97), nearly tripled its energy use. RAG increased Starcoder's total energy consumption by 148.5% to 353.26 J (± 0.44), making it the most energy-intensive configuration overall. As shown by the stacked bars, in all four configurations, the GPU (green) was responsible for the vast majority of the energy consumed, with the CPU's contribution (blue) being negligible.

7.1.3 RQ3: Identify Sustainable Configurations

To answer RQ3: *Which retrieval depth provides the best trade-off between correctness and cost?,* we calculated the primary sustainability metric: Energy per Solved Task (Joules). This metric combines the effectiveness from RQ1 (pass@1 rate) with the corrected energy cost from Updated Table 2 (Total Energy / Run).

As shown in Table 4 and Figure 6, the Llama (No RAG) configuration was the most sustainable by a large margin, requiring only 95.44 Joules (± 8.91) to produce a single correct solution. Enabling RAG significantly worsened this sustainability trade-off for both models:

Llama: The energy cost per solved task increased by 226.2% (from 95.44J to 311.29J).

Starcoder: The energy cost per solved task increased by 148.5% (from 338.46J to 841.10J).

Table 4: Sustainability Trade-off (Energy per Solved Task) on Apple Silicon

Model	Condition	Energy per Solved Task (J)
Llama	k=0	95.44 (± 8.91)
Llama	k=3	311.29 (± 19.01)
Starcoder	k=0	338.46 (± 1.14)
Starcoder	k=3	841.10 (± 1.05)

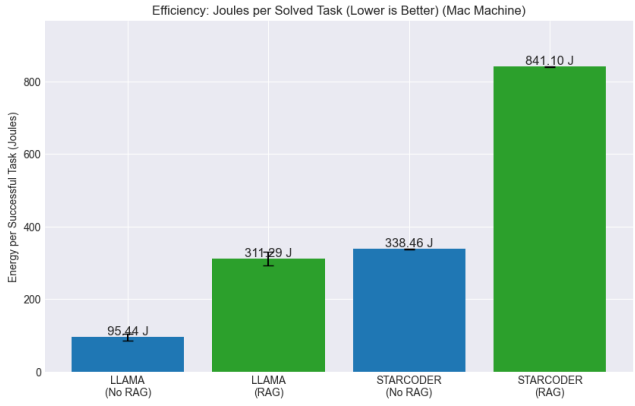

Figure 6: Efficiency: Joules per Solved Task on Apple Silicon (Lower is Better)

Figure 7 provides a combined view of this trade-off, plotting pass@1 (bars) against total energy (lines). This visualization shows that for both models, activating RAG resulted in a simultaneous decrease (or no change) in accuracy and a sharp increase in total energy cost.

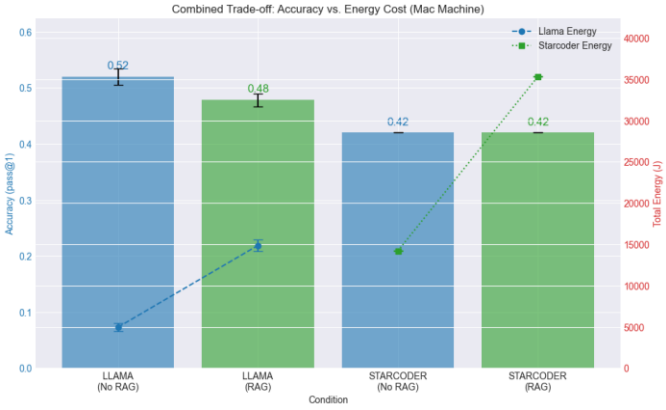

Figure 7: Combined view of trade-off (bars) against total energy (lines) on Apple Silicon

7.2 Windows

This section presents the results from the experiment conducted on the Windows desktop platform . The data was collected over 6 full runs for each of the four configurations (Llama k=0, Llama k=3, Starcoder k=0, Starcoder k=3), with each run processing 100 tasks. The findings are presented in order of the research questions.

7.2.1 RQ1: Effectiveness (Correctness)

To answer RQ1: *Does retrieval improve correctness compared to baseline LLMs?,* we measured the pass@1 rate, representing the percentage of tasks solved correctly on the first attempt.

The results, as summarized in Table 5 and visualized in Figure 8, show that RAG at k=3 did not improve the pass@1 rate for either model. For the Llama model, the baseline (No RAG) configuration achieved the highest correctness at 52.00% (±1.41). Enabling RAG caused a slight decrease in correctness to 49.00% (±1.41). For the Starcoder model, the baseline achieved 42.00% accuracy, while enabling RAG resulted in a slight improvement to 45.00%, though both models performed substantially worse than Llama overall.

Table 5: Model Correctness (pass@1) on Apple Silicon

Model	Condition	Pass@1 Rate (Avg)
Llama	k=0	52.00%
Llama	k=3	49.00%
Starcoder	k=3	42.00%
Starcoder	k=3	45.00%

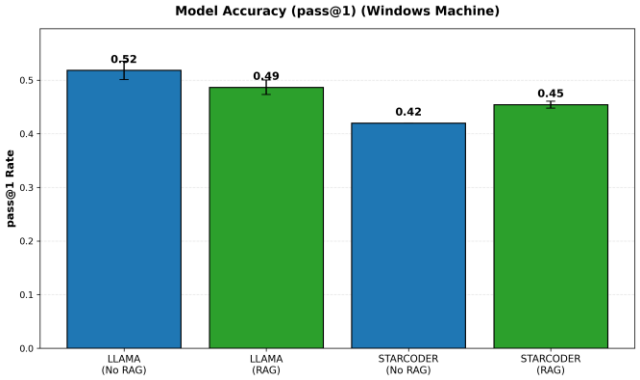

Figure 8: Model Accuracy (pass@1) on Apple Silicon

7.2.2 RQ2: Cost (Computational)

To answer RQ2: *How do retrieval depths impact computational, economic, and energy costs compared to baseline?*, we measured token usage, average power draw, energy consumption per task, and latency. The data in Table 6 shows that RAG substantially increased every cost metric.

The context from RAG dramatically increased the number of tokens processed. Llama saw a 7.2x increase in average tokens (162 to 1167), and Starcoder saw a 6.0x increase (243 to 1448), as shown in Figure 9. This token increase directly translated to higher energy consumption. As seen in Figure 10, Llama's average energy per task rose by 27% (from 188.59J to 238.79J). Starcoder's energy per task rose by 48% (from 685.25J to 1013.12J).

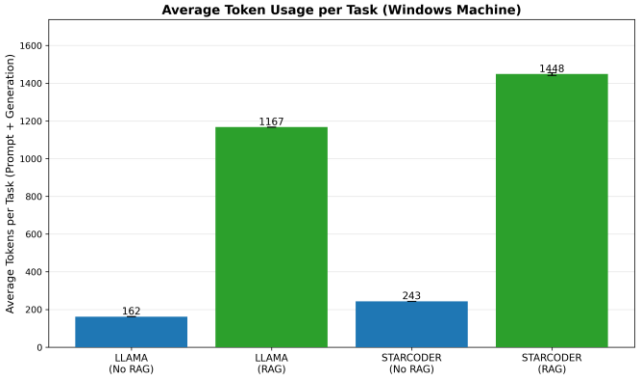

Figure 9: Average Token Usage per Task (Prompt + Generation) on Apple Silicon

Table 6: Computational Costs per Task on Windows (Mean ± StdDev)

Model	Condition	Avg Tokens	Avg Energy (J)
Llama	k=0	163 (±1)	188.59 (±5.03)
Llama	k=3	1167 (±1)	238.79 (±6.97)
Starcoder	k=0	242 (±0)	685.25 (±0.48)
Starcoder	k=3	1418 (±0)	1013.12 (±0.44)

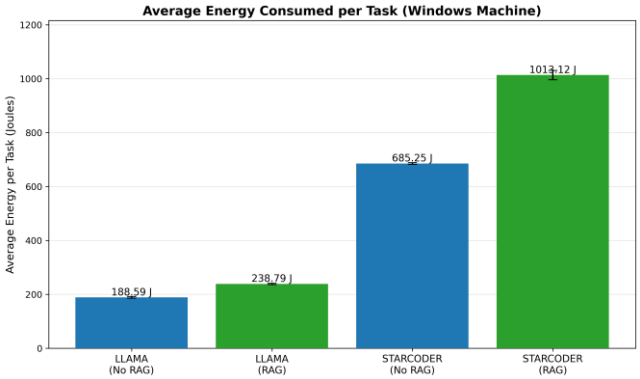

Figure 10: Average Energy Consumed per Task (Joules) on Windows

We also measured the time-based costs, specifically the average wall-clock time to complete a single task and the total time required to process the full 100-task workload. The results, shown in Table 7 and visualized in Figure 11, demonstrate a significant time cost associated with both RAG and the Starcoder model.

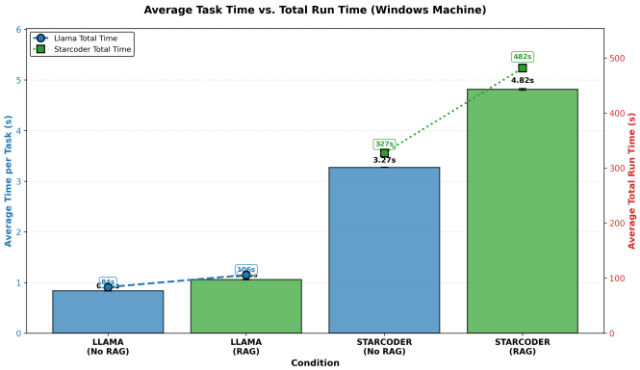

Figure 11: Average Task Time vs. Total Run Time (Windows)

Table 7: Time and Latency Summary on Windows

Model	Condition	Avg Time / Task (s)	Avg Total Time / Run (s)
Llama	k=0	*0.84 (± 0.02)*	84
Llama	k=3	1.06 (± 0.07)	106
Starcoder	k=0	3.27 (± 0.01)	327
Starcoder	k=3	4.82 (± 0.00)	482

The baseline Starcoder model was 3.9x slower per task than the baseline Llama model (3.27s vs 0.84s). Enabling RAG increased Llama's average time per task by 26% (from 0.84s to 1.06s). Enabling RAG increased Starcoder's average time per task by 47% (from 3.27s to 4.82s).

These costs accumulated significantly. The Llama (No RAG) configuration was the fastest, completing the workload in an average of 84 seconds (1.4 minutes). The Starcoder (RAG) configuration was the slowest, requiring 482 seconds (8.0 minutes) for the same 100 tasks.

In addition to per-task metrics, we measured the total energy consumed to complete the entire 100-task workload for each configuration. Figure 12 presents this data, averaging over the 6 experiment runs.

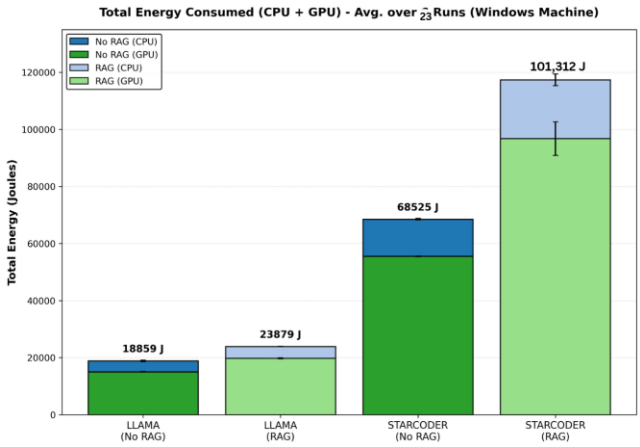

Figure 12: Total Energy Consumed (CPU + GPU) - Avg. over Runs (Windows)

The results show the significant cumulative cost of RAG and the baseline differences between the models. The Llama (No RAG) configuration was the most energy-frugal, consuming 18,859 J in total to process all 100 tasks. The baseline Starcoder model consumed 68,525 J, approximately 3.6 times more energy than the baseline Llama model for the same workload.

Enabling RAG increased Llama's total energy consumption by 27% to 23,879 J. RAG increased Starcoder's total energy consumption by 71% to 117,396 J, making it by far the most energy-intensive configuration overall.

As shown by the stacked bars, in all four configurations, the GPU (green) was responsible for the vast majority of the energy consumed, accounting for approximately 80-85% of total energy across all configurations. The CPU's contribution (blue) remained relatively consistent at approximately 15-20%, reflecting the dominant role of GPU acceleration in LLM inference on the Windows platform.

7.2.3 RQ3: Identify Sustainable Configurations

To answer RQ3: *Which retrieval depth provides the best trade-off between correctness and cost?,* we calculated the primary sustainability metric: Energy per Solved Task (Joules). This metric combines the effectiveness from RQ1 (pass@1 rate) with the energy cost (Total Energy / Run).

As shown in Table 8 and Figure 13, the Llama (No RAG) configuration was the most sustainable by a substantial margin, requiring only 364.29 Joules to produce a single correct solution. Enabling RAG significantly worsened this sustainability trade-off for both models:

Llama: The energy cost per solved task increased by 35% (from 364.29J to 490.98J).

Starcoder: The energy cost per solved task increased by 37% (from 1631.56J to 2236.71J).

Table 8: Sustainability Trade-off (Energy per Solved Task) on Windows

Model	Condition	Energy per Solved Task (J)
Llama	k=0	364.29
Llama	k=3	490.98
Starcoder	k=0	1631.56
Starcoder	k=3	2236.71

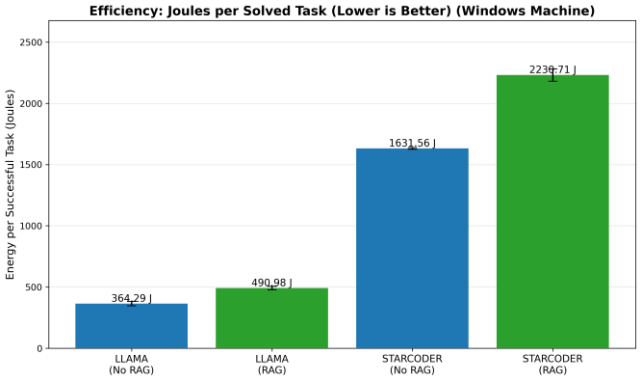

Figure 13: Efficiency: Joules per Solved Task on Windows (Lower is Better)

Figure 4 provides a combined view of this trade-off, plotting pass@1 (bars) against total energy (lines). This visualization shows that for Llama, activating RAG resulted in a simultaneous decrease in accuracy (52% to 49%) and an increase in total energy cost (18,859J to 23,879J). For Starcoder, RAG provided a modest accuracy improvement (42% to 45%) but at the cost of a 71% increase in total energy consumption (68,525J to 117,396J). In both cases, the energy cost increase outpaced any correctness gains, demonstrating that the baseline configurations provided superior sustainability trade-offs.

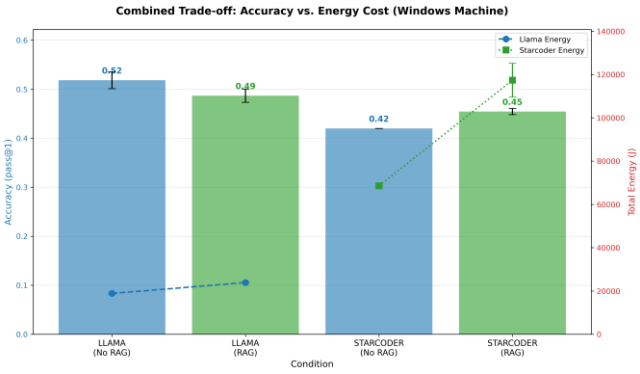

Figure 14: Combined view of trade-off (bars) against total energy (lines) on Windows

8. Discussion

This study was designed to evaluate the trade-offs between effectiveness and cost-efficiency of Retrieval-Augmented Generation in software development. We analyzed the results holistically across our three research questions, comparing two

models (Llama, Starcoder) and two platforms (Mac, Windows) to understand the impact of RAG.

8.1 RQ1: The Surprising Ineffectiveness of RAG

Our first research question asked if RAG improves code correctness. The results provided a clear and unexpected answer: No. Our hypothesis (H1), which posited that RAG would improve correctness, was disproven.

For the Llama model, the baseline (No RAG) configuration achieved a 52.00% pass@1 rate on both platforms. Enabling RAG decreased correctness to 47.83% on the Mac and 49.00% on the Windows system. For the Starcoder model, the results were neutral on Mac (42.00% for both) and showed only a minor, likely insignificant, improvement on Windows (42.00% to 45.00%).

This finding is a critical deviation from much of the related work (e.g., RagVerus [1], SOSecure [3]) which found significant improvements from RAG. We speculate that this is not an indictment of RAG as a technique, but rather a strong indicator of the importance of corpus relevance. As noted in Section 4, our retrieval corpus consisted of general Python documentation, which was not tailored to the algorithmic problem-solving nature of the HumanEval and MBPP benchmarks. It is plausible that the retrieved context (at k=3) was irrelevant or non-specific, acting as "noise" that distracted the model rather than aiding it. This finding is corroborated by recent surveys, such as Gao et al. (2024), which note that RAG performance is highly dependent on the quality and relevance of the retrieved documents. Gao et al. highlight that irrelevant or 'noisy' context can degrade generation quality, turning the augmentation into a detriment [11]. This demonstrates that RAG is not a universal "silver bullet"; a poorly-suited corpus can actively degrade performance, turning RAG from a retrieval-augmentation tool into a retrieval-detriment tool.

8.2 RQ2 & Platform Comparison: The High Cost of RAG and the Hardware Trade-Off

Our second research question examined the cost implications of RAG. The data confirms that RAG introduces substantial overhead in all measured categories (tokens, energy, and time).

8.2.1 Energy and Time: A Tale of Two Platforms

The interplay between platform and condition is best understood by analyzing energy and time costs comparatively.

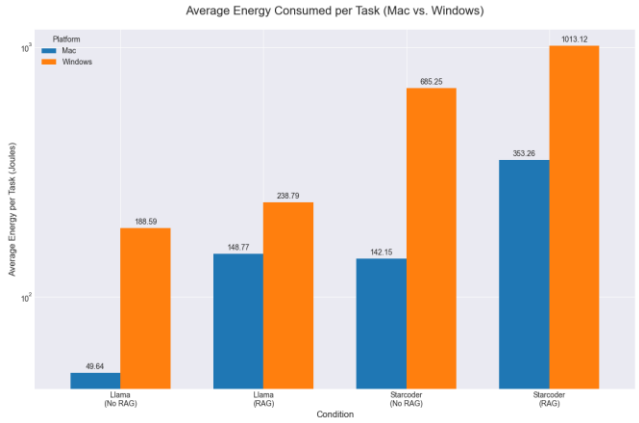

Figure 15: Average Energy Consumed per Task (Joules) by Model, Condition, and Platform.

Figure 15 provides a stark visualization of energy costs. Two trends are immediately apparent. First, RAG consistently and significantly increased energy consumption for all model-platform combinations. On the Mac, RAG increased Llama's energy use by 198% (49.64 J to 148.77 J) and Starcoder's by 148% (142.15 J to 353.26 J). Second, the Windows platform was dramatically more power-intensive. The baseline Llama configuration, for instance, consumed 188.59 J on Windows, a 279% increase over the 49.64 J used by the Mac for the exact same workload.

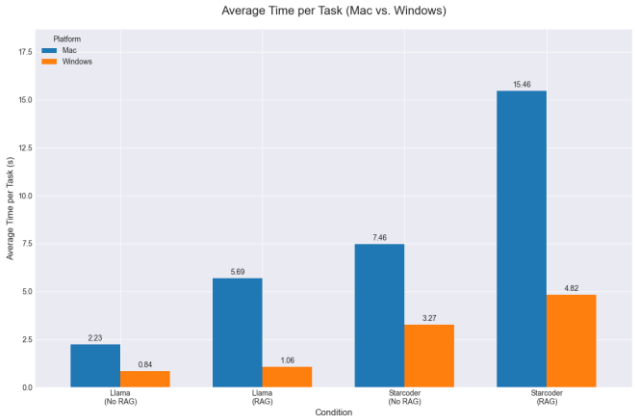

Figure 16: Average Time per Task (Seconds) by Model, Condition, and Platform.

Figure 16 illustrates the opposite story for latency. Here, the Windows platform's high power draw translates directly to superior performance (speed). The baseline Llama model, which took 2.23 seconds per task on the Mac, completed in only 0.84 seconds on Windows, a 2.7x speedup. This demonstrates the raw computational power of the NVIDIA RTX 4070 SUPER. However, the cost of RAG is still evident: on the faster Windows machine, RAG still increased Llama's task time by 26% (0.84s to 1.06s) and Starcoder's by 47% (3.27s to 4.82s).

These graphs reveal a clear architectural trade-off: the Apple M4 Pro (Mac) is an exemplar of energy efficiency (low power, lower speed), whereas the AMD/NVIDIA system is a high-performance "hot rod" (high power, high speed). The choice of platform is therefore context-dependent: for applications where real-time latency is paramount, the NVIDIA-powered system is superior. For large-scale, non-interactive batch processing where energy costs (and thus, financial and environmental costs) are the primary concern, the Apple Silicon architecture is overwhelmingly more sustainable.

8.3 RQ3: The Sustainability Trade-Off: A Clear Winner

Our third research question sought the configuration with the best trade-off between correctness and cost. By combining our metrics into "Energy per Solved Task," we can definitively answer this.

Figure 17: Accuracy (pass@1) vs. Energy Efficiency (Joules per Solved Task) across all Configurations.

Figure 17 plots all experimental configurations on a 2D plane of Accuracy (y-axis, higher is better) vs. Efficiency (x-axis, lower is better). The ideal configuration would be in the top-left corner.

Our results place the Llama (No RAG) configurations for both Mac and Windows firmly in this top-left, "most desirable" quadrant. They offered the highest accuracy (52%) and, in the Mac's case, the lowest energy cost. The Llama (No RAG) (Mac) point, with an efficiency of 95.44 J per solved task, is the clear "winner" in terms of overall sustainability.

Conversely, all Starcoder configurations are in the "worst" bottom-right quadrant (low accuracy, high energy cost). Notably, this shows the general-purpose Llama model was superior to the code-specialized Starcoder model on this benchmark, even before considering RAG.

While the Llama family of models are general-purpose, recent evaluations have confirmed their strong capabilities in software engineering tasks. For instance, Lu et al. (2025) found that Llama 3 models achieve state-of-the-art performance on code generation benchmarks, often outperforming dedicated code models [13]. This makes our finding that Llama outperformed Starcoder particularly relevant, contributing to the ongoing discussion about generalist vs. specialist models.

This graph visualizes our study's primary conclusion: activating RAG provided no accuracy benefit and, in return, exacted a heavy energy toll. This pushed the Llama (RAG) points "down" (lower accuracy) and "right" (worse efficiency), making them objectively worse configurations than the simple baseline. The main hypothesis (H1) is not only unsupported; it is decisively contradicted by the data.

9. CONCLUDING REMARKS

This study conducted an empirical evaluation of Retrieval-Augmented Generation for code generation, assessing its impact on effectiveness (pass@1 rate) and sustainability (energy, latency) against baseline LLMs. We tested Llama-3.1-8B and Starcoder2-7B models on two distinct hardware platforms, an Apple M4 Pro and an NVIDIA RTX 4070 SUPER. Our key finding was that RAG, implemented with a general-domain Python corpus, failed to improve code correctness for algorithmic tasks. In fact, it slightly degraded the accuracy of the superior Llama model, while drastically increasing energy consumption (by up to 198%) and latency (by up to 155%) on our most efficient platform.

The primary implication of these findings is that RAG is not a guaranteed enhancement. Its value is critically dependent on the relevance of the retrieved context to the specific task domain. Using an irrelevant corpus, as we did, appears to add costly "noise" that confuses the model and degrades performance, nullifying any potential benefits. This study highlights that a "naive" RAG implementation can be counterproductive, making the system less accurate, slower, and significantly less sustainable. Furthermore, our results underscore a fundamental platform trade-off: Apple Silicon's architecture is vastly superior for energy efficiency, while NVIDIA's desktop GPUs provide a clear advantage in raw processing speed.

Several avenues for future work emerge from this study. The most critical is to re-evaluate this experiment using a highly-tailored retrieval corpus, such as a collection of algorithmic solutions or competitive programming documentation, to determine if a relevant corpus can unlock RAG's hypothesized benefits. Second, this study used a fixed retrieval depth of k=3; future work should explore the impact of varying k to find a potential optimal balance between context and noise. Finally, this analysis could be extended to other models, platforms (e.g., cloud TPUs/GPUs), and retrieval strategies (e.g., semantic/vector search vs. the lexical BM25 used here) to build a more comprehensive model of RAG's true costs and benefits in sustainable software engineering.

10. REFERENCES

[1] Zhong, S. C., Zhu, J., Tian, Y., & Si, X. (2025). RagVerus: Repository-Level Program Verification with LLMs using Retrieval Augmented Generation. University of Toronto.

[2] Gan, A., Yu, H., Zhang, K., Liu, Q., Yan, W., Huang, Z., Tong, S., Chen, E., & Hu, G. (2025). Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey. Frontiers of Computer Science.

[3] Mukherjee, M., & Hellendoorn, V. J. (2025). SOSecure: Safer Code Generation with RAG and StackOverflow Discussions. In Proceedings of the 33rd ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2025): Demonstration Track.

[4] Dutt, H. (2025). A Retrieval-Augmented Generation (RAG) and Large Language Model (LLM)-Based Artificial Intelligence Coder Agent for SDKs and Frameworks. Technical Disclosure Commons.

[5] Wu, H., Wang, X., & Fan, Z. (2025). Addressing the Sustainable AI Trilemma: A Case Study on LLM Agents and RAG. In Proceedings of the 47th International Conference on Software Engineering (ICSE 2025).

[6] Wang, Z. Z., Asai, A., Yu, X. V., Xu, F. F., Xie, Y., Neubig, G., & Fried, D. (2025). CodeRAG-Bench: Can Retrieval Augment Code Generation? In Proceedings of the 2025 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2025).

[7] Barker, M., Bell, A., Thomas, E., Carr, J., Andrews, T., & Bhatt, U. (2025). Faster, Cheaper, Better: Multi-Objective Hyperparameter Optimization for LLM and RAG Systems. ICLR Workshop.

[8] Şakar, T., & Emekci, H. (2025). Maximizing RAG Efficiency: A Comparative Analysis of RAG Methods. Natural Language Processing, 31, 1–25.

[9] Tan, H., Luo, Q., Jiang, L., Zhan, Z., Li, J., Zhang, H., & Zhang, Y. (2025). Prompt-based Code Completion via Multi-Retrieval Augmented Generation. ACM Transactions on Software Engineering and Methodology.

[10] Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. Journal of Machine Learning Research.

[11] Gao, Y., Xiong, Y., Gao, X., & Liu, K. (2024). Retrieval-Augmented Generation for Domain-Specific Tasks: A Survey. ACM Computing Surveys, 57(1), 1-36.

[12] Lozhkov, A., et al. (2024). StarCoder2 and The Stack v2: The Next Generation of Open Code LLMs. Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[13] Lu, Y., et al. (2025). An Evaluation of Llama 3 on Code Generation Benchmarks. Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE).