Quantifying the RAG Advantage: A Multi-Metric Benchmark for LLM-based Code Generation

Gabriel Souza Baggio[1], **Gabriel Machado Lunardi**[1], **Guilherme Medeiros Machado**[3], **José Palazzo Moreira de Oliveira**[2]

[1]Centro de Tecnologia - Universidade Federal de Santa Maria - RS, Brasil

[2]Instituto de Informática - Universidade Federal do Rio Grande do Sul - RS, Brasil

[3]LyRIDS lab, ECE Engineering School, Paris, France

gsbaggio@inf.ufsm.br, gabriel.lunardi@ufsm.br,

palazzo@inf.ufrgs.br, gui.medeiros1@gmail.com

Abstract. *The recent advancement of Large Language Models (LLMs) has demonstrated remarkable capabilities in solving programming challenges. However, despite their proficiency, LLMs often suffer from hallucination and limited performance on unfamiliar or complex tasks. Retrieval-Augmented Generation (RAG) has emerged as a promising solution to address these limitations by supplementing prompts with relevant external information. In this paper, we propose a benchmark to assess the efficacy of RAG in solving algorithmic problems by integrating a curated database of 120 LeetCode problems, each paired with corresponding solutions and explanations. An Information Retrieval (IR) system was employed to construct enhanced prompts for solving novel problems.*

1. Introduction

Although current Large Language Models (LLMs) have demonstrated impressive capabilities in code generation tasks [Barbosa et al. 2022, Wang and Chen 2023], they often produce incorrect or inefficient solutions when faced with complex algorithmic problems — as seen in other benchmarks [Xia et al. 2025, Wang et al. 2024, Coignion et al. 2024]. These errors typically arise due to a lack of specific contextual knowledge and a limited understanding of underlying data structures, leading to hallucinations — outputs that are syntactically invalid or logically inconsistent [Liu et al. 2024]. To mitigate this, Retrieval-Augmented Generation (RAG) has been proposed as a mechanism to ground the model's outputs in relevant external knowledge by, with an Information Retrieval (IR) algorithm, retrieving contextually similar information from a structured corpus [Izacard and Grave 2021, Gao et al. 2023].

The primary goal of this study is to evaluate whether RAG improves the performance of LLMs in solving programming problems. To achieve this, we created a database consisting of 120 LeetCode problems, each paired with a verified code solution and a natural language explanation. This dataset serves as the knowledge base for the RAG system. To retrieve these problems, we implemented an IR pipeline based on cosine similarity with a dynamic threshold [Balasubramanian 2011]. The system was evaluated on a separate set of 30 previously unseen problems. For each of these, the IR algorithm selected similar solved examples from the database, which were then used to construct

context-augmented prompts for the LLM. Using the LeetCode judge system, which validates solutions through extensive test case execution, we were able to verify whether or not the retrieved context improved the correctness and performance of the LLMs' outputs.

Even though we leverage insights from earlier studies on LLM-based code generation [Wang and Chen 2023, Xia et al. 2025, Wang et al. 2024, Coignion et al. 2024], this is the first work to apply RAG to retrieve similar examples for solving programming problems. Additionally, the evaluation also analyzes execution metrics such as runtime and memory usage to provide a better assessment of RAG's impact on LLM performance beyond correctness alone. The evaluation also analyzes variations in the difficulty and release years of the problems to test the robustness of the system under different conditions.

2. Related Works

Several works have investigated the ability of LLMs to solve competitive programming problems — albeit without RAG —, particularly in platforms such as LeetCode [Wang et al. 2024, Xia et al. 2025, Huynh and Lin 2025, Tian et al. 2023, Coignion et al. 2024]. Our study draws inspiration from these studies' evaluation strategies and model choices, extending their approaches into the context of RAG.

Other similar work is the study by [Taschetto and Fileto 2024], presented at SBBD in 2024. Their work explores the application of RAG to enhance the performance of LLMs on the Brazilian University Admission Exam (ENEM), which aligns closely with the objectives of our research: mitigate hallucinations and improve answer reliability.

Together, these studies lay important groundwork for our investigation. For instance, incorporating [Wang et al. 2024]'s proposal of using runtime and memory as evaluation metrics, and inspired by the temporal and difficulty-based partitioning of the problems introduced by [Xia et al. 2025], we extend these strategies into the context of RAG.

3. Methodology

To evaluate RAG's impact on our work, we used six well-known LLMs and a set of programming questions, each paired with a previously tested and accepted high-performance solution and explanation. These entries serve as the knowledge base for the RAG. All datasets and code used in the methodology are included in our open-source repository[1].

3.1. Information Retrieval Algorithm

The IR algorithm retrieves similar solved problems from the database. First, each problem description in the database is transformed into a high-dimensional vector embedding using the 'all-MiniLM-L6-v2'[2] model. These embeddings capture the semantic essence of the problems. The collection of embeddings is then indexed using FAISS (Facebook AI Similarity Search), employing an 'IndexFlatL2' structure that computes Euclidean distances. Because the vectors are already normalized, we can compute the cosine similarity directly from the Euclidean distance.

A distinctive aspect of our IR system is its adaptive thresholding mechanism, introduced by [Balasubramanian 2011]. Instead of a fixed cutoff, the threshold for deeming

[1] `https://github.com/gsbaggio/ArtigoRAG`

[2] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

a retrieved problem "similar enough" is dynamically adjusted based on the highest similarity score. The initial threshold used for the cosine similarity is 65%, but if the highest similarity is strong (above 80%), the threshold becomes more stringent; if weaker (less than 70%), it relaxes, but never below the 65% minimum. Similar adaptive strategies are used in RAG to reduce noise [Zhang et al. 2024], and we apply it with the same intent.

3.2. Dataset Overview

The knowledge base of our RAG comprises 120 programming problems, all sourced from LeetCode. This dataset is stored in a JSON file, where each entry includes the problem's title, difficulty level (Easy, Medium or Hard), LeetCode's own algorithmic categories, full question text, a verified and optimized code solution, and a detailed natural language explanation of the solution approach. Each problem's solution chosen was verified to have optimal performance. These problems do not require a specific programming language for implementation, allowing flexibility in the solution approaches.

For evaluation, a distinct set of 30 novel programming problems was utilized, also sourced from LeetCode. These problems were not part of the RAG's knowledge base, ensuring a fair assessment of its generalization capabilities [Izacard and Grave 2021, Gao et al. 2023, Soares et al. 2025, Miranda et al. 2023]. For each of these 30 test questions, we have also retrieved their difficulty level and release year, which allowed a better analysis of the RAG system's performance across different problem characteristics. The difficulties of the problems are well distributed across both datasets. In the knowledge base, we intentionally included a higher proportion of Hard problems to provide more potentially relevant information, while the test set contains more Medium problems to reflect typical algorithmic challenges.

Table 1. Distribution of Problem Difficulty in the Knowledge and Test Datasets

	Knowledge Dataset		Test Dataset	
Difficulty	**Count**	**Proportion (%)**	**Count**	**Proportion (%)**
Easy	27	22.50	5	16.67
Medium	53	44.17	19	63.33
Hard	40	33.33	6	20.00

The problems from both datasets were selected from the same LeetCode's own algorithmic categories. We focused on the most popular categories with the highest number of available problems, such as sorting, dynamic programming, and graph algorithms. Additionally, all test problems were selected to ensure at least one problem was retrieved from the knowledge base, enabling a better evaluation of the system's performance.

3.3. Model Selection

The models utilized were chosen based on their use in prior research: GPT-4-omni [Wang et al. 2024, Tian et al. 2023], Claude 3.7 Sonnet, DeepSeek-R1, and Qwen2.5-max (related to our Qwen2.5-Coder-32B) [Xia et al. 2025], and GPT-3.5 Turbo [Huynh and Lin 2025]. This section provides information about the models used:

- **GPT-4 omni:** Developed by OpenAI, the LLM features a 128K token context window and advanced reasoning. Cutoff date: October 2023.

- **Claude 3.7 Sonnet:** Developed by Anthropic, this model (as shown in [Xia et al. 2025]) is known for strong performance on complex tasks and features a 200K token context window. Cutoff date: November 2024.
- **GPT-3.5 Turbo:** An OpenAI model widely adopted for its balance of speed and capability. Has a 16K token context window. Cutoff date: September 2021.
- **Gemini 2.0 Flash:** A Google's language model, designed for complex reasoning and supporting up to 1 million tokens. Cutoff date: June 2024.
- **Qwen2.5-Coder-32B:** Developed by Alibaba Cloud, this model is specialized for code tasks and supports a 128k tokens of context. Cutoff date: End of 2023.
- **DeepSeek-R1:** From DeepSeek AI, this model demonstrated the highest performance in Xia et al.'s evaluation [Xia et al. 2025] and presents a deeper thinking process. It supports 131k tokens of context. Cutoff date: January 2025.

Another decisive factor in the model selection was the diversity in training focus and knowledge cutoff date: we included a code-specialized model (Qwen2.5-Coder-32B), a model known for advanced reasoning capabilities (DeepSeek-R1), a less powerful model (GPT-3.5 Turbo), as well as general-purpose models with broader training data.

3.4. Experiments

We conducted a series of experiments using the methodology described previously. Each of the 30 test problems was submitted to all six selected LLMs, both with and without the aid of the RAG pipeline. For the RAG-enhanced scenario, each test problem was matched against all entries in the 120-problem knowledge base using the IR system described earlier. Models received additional instructions to use their most reliable programming language (LeetCode accepts multiple languages) for solving the current problem. After generating code solutions, all outputs were submitted to the LeetCode judge system, which verifies correctness by running the solution against a comprehensive set of test cases and returns whether the solution was accepted or rejected (rejection occurs if any test case fails). It also provides execution metrics — runtime and memory usage — indicating the percentage of user submissions that the solution outperformed in each metric.

An important motivation behind applying RAG in this context is the similarity among many problems found on competitive programming platforms. These problems frequently share underlying patterns and solution techniques, often requiring similar approaches or identical algorithms. By retrieving and presenting analogous solved problems from the knowledge database, the RAG-enhanced prompt allows the model to emulate successful solution strategies, improving both correctness and computational efficiency. It also mimics the learning process of human programmers, who often recall and adapt familiar problem-solving techniques to new but similar challenges.

In our implementation, semantic similarity was computed exclusively using the textual content of the questions, excluding metadata such as titles or categories, because only the question text was available for the test problems. However, once the most similar problems were retrieved, all available information of the knowledge questions — including similarity score (normalized between 0 and 1), problem title, categories (e.g., dynamic programming, sort), solution code, and natural language explanation — was incorporated into the final prompt sent to the LLM. This strategy ensured that the augmented prompt provided rich and relevant context to the model. For each test case, we recorded if it was correct and the percentage of other user submissions beaten in runtime and memory.

4. Results and Analysis

The results, included in our repository, demonstrate that RAG improves LLM performance across multiple dimensions. As shown in the analysis in Figure 1, four out of six models exhibited enhanced accuracy when augmented with similar problems. The most substantial improvement was observed in GPT-3.5 Turbo, which increased from 50.0% to 66.7% correctness (a 33.4% relative improvement), supporting previous findings that RAG benefits less capable models by providing structured context to offset limited reasoning [Gao et al. 2023]. The overall increase in correctness across all models was 6.1%.

Notably, Gemini 2.0 and Qwen2.5-Coder maintained identical correctness rates with and without RAG (73.3% each). However, both models demonstrated significant improvements in efficiency metrics, indicating that RAG helps generate optimized solutions even when correctness remains constant. DeepSeek-R1 with RAG achieved the highest overall performance, reaching 96.7% correctness, with only one incorrect answer.

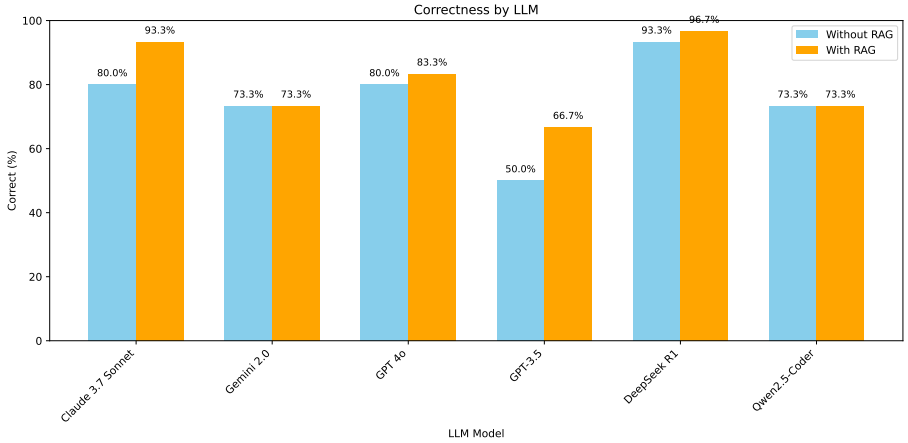

Figure 1. Correctness analysis

Beyond correctness, RAG substantially improved computational efficiency, which can be seen in Figure 2. Speed improvements were observed across all six models, with a median improvement of 6.3%. Memory efficiency gains were even more significant, showing an average increase of 11.0%. Table 2 shows the average performance of all models across different problem characteristics. The RAG approach achieved 100% correctness on Easy problems and showed greater improvements on post-2023 questions.

Table 2. Performance Comparison by Problem Category and Time Period (%)

	Without RAG			With RAG		
Category	**Correct**	**Speed**	**Memory**	**Correct**	**Speed**	**Memory**
Easy	90.0	74.7	45.5	100.0	78.1	57.8
Medium	80.7	68.2	48.0	86.0	74.9	61.5
Hard	44.4	54.8	59.6	50.0	60.9	51.2
Before 2023	80.3	71.7	50.1	84.8	81.0	61.0
2023-2025	60.4	54.2	44.4	70.8	50.3	54.6

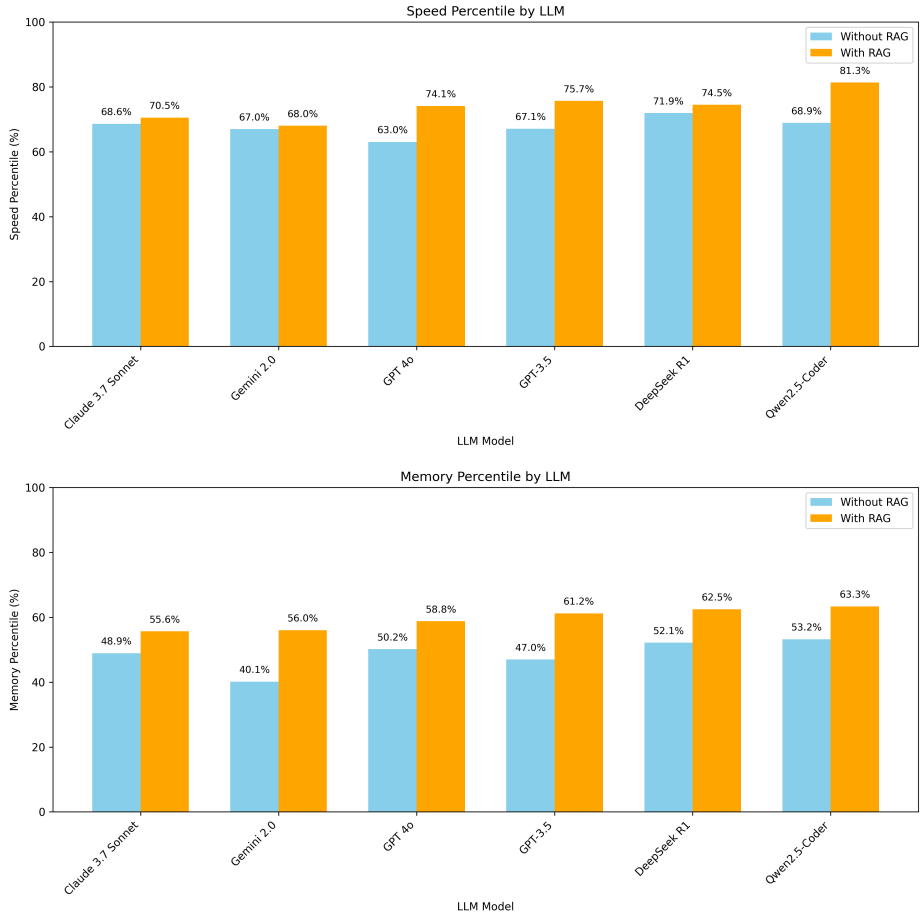

Figure 2. Speed and memory percentile ranking analysis

5. Conclusion and Future Work

This study presents evidence suggesting that Retrieval-Augmented Generation may enhance LLM performance in solving programming problems, with improvements observed across correctness, runtime, and memory. These findings have important implications for education, where students can benefit from more accurate and efficient solution approaches provided by RAG-enhanced LLMs. Additionally, systems could reuse the strategies developed in this study to incorporate LLM-based interactions into data exploration.

Future research should address key limitations and explore several promising directions to extend this work's impact. A key limitation of our study is its reliance on a single dataset source (LeetCode problems), which may limit the generalizability of our findings. Another critical concern is that performance gains may result from longer prompts rather than semantic retrieval quality. To address these issues, future studies should evaluate across diverse programming platforms, and include baselines that retrieve random problems for proper comparison with our RAG strategy. Further investigation into scaling these systems with larger knowledge bases could also enhance retrieval quality.

Acknowledgments

The authors thank CNPq for the support of the Universal Projects 405973/2021-7 and 402086/2023-6, and FAPERGS for the project ARD/ARC – Process 24/2551-0000645-1.

References

Balasubramanian, N. (2011). *Query-dependent selection of retrieval alternatives*. PhD thesis, University of Massachusetts Amherst.

Barbosa, M., Valle, P., Nakamura, W., Guerino, G., Finger, A., Lunardi, G., and Silva, W. (2022). Um estudo exploratório sobre métodos de avaliação de user experience em chatbots. In *VI Escola Regional de Engenharia de Software*, Porto Alegre, RS, Brasil.

Coignion, T., Quinton, C., and Rouvoy, R. (2024). A performance study of llm-generated code on leetcode. In *28th International Conference on Evaluation and Assessment in Software Engineering*, page 79–89, New York, NY, USA. ACM.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., and Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. 2:1.

Huynh, N. and Lin, B. (2025). Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications.

Izacard, G. and Grave, E. (2021). Leveraging passage retrieval with generative models for open domain question answering.

Liu, F., Liu, Y., Shi, L., Huang, H., Wang, R., Yang, Z., Zhang, L., Li, Z., and Ma, Y. (2024). Exploring and evaluating hallucinations in llm-powered code generation.

Miranda, A. L., Garcia, R., Lunardi, G. M., Vilela, R., Vale, P. H., and Silva, W. (2023). Projeto e avaliação de um template de worked examples para o ensino de programação. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1673–1684. SBC.

Soares, T. S., Costa, R. L. H., Soares, E., Calderon, I., Lunardi, G. M., Valle, P. H. D., Guedes, G. T., and Silva, W. (2025). Machine learning-assisted tools for user experience evaluation: A systematic mapping study. *Simpósio Brasileiro de Sistemas de Informaçao (SBSI)*, pages 379–388.

Taschetto, L. and Fileto, R. (2024). Using retrieval-augmented generation to improve performance of large language models on the brazilian university admission exam. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 799–805, Porto Alegre, RS, Brasil. SBC.

Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S.-C., Klein, J., and Bissyandé, T. F. (2023). Is chatgpt the ultimate programming assistant – how far is it?

Wang, J. and Chen, Y. (2023). A review on code generation with llms: Application and evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, pages 284–289. IEEE.

Wang, L., Shi, C., Du, S., Tao, Y., Shen, Y., Zheng, H., and Qiu, X. (2024). Performance review on llm for solving leetcode problems. In *2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM)*, pages 1050–1054.

Xia, Y., Shen, W., Wang, Y., Liu, J. K., Sun, H., Wu, S., Hu, J., and Xu, X. (2025). Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms.

Zhang, Z., Fang, M., and Chen, L. (2024). Retrievalqa: Assessing adaptive retrieval-augmented generation for short-form open-domain question answering.