AlgoAce: Retrieval-Augmented Generation for Assistance in Competitive Programming

Anav Agrawal[1,*], Jill-Jênn Vie[1,*]

[1]Soda team, Inria Saclay, 91120 Palaiseau, France

Abstract

Competitive programming platforms like Codeforces are widely used for self-directed learning and assessment in computer science education. However, students often struggle to identify relevant practice problems or receive targeted help without personalized guidance. In this paper we present AlgoAce, an AI-powered assistant that integrates with Codeforces and uses retrieval-augmented generation (RAG) to provide support while attending at two key sources of information: user submission history, and problem metadata. The system leverages large language models to fetch relevant problems and past user submissions, grounding user queries in retrieved content before generating responses. AlgoAce adapts to user preferences—such as programming language or problem tags—and supports reflective learning through natural language interactions. We describe the tool's architecture and discuss its potential for integration into the competitive programming learning ecosystem.

Keywords

Retrieval-augmented generation, Competitive programming, Intelligent tutoring systems, Codeforces

1. Introduction

Competitive programming platforms, such as Codeforces[1], HackerRank, or Kattis, provide thousands of problem-solving challenges. While they primarily rank programmers through contests, participants also use them for self-directed learning. However, it is always easier to rank students than to upskill them: surprisingly, little effort has been directed towards building tools to improve the skills of competitive programming contestants. Students frequently struggle to identify exercises that align with their current skill levels. There remains a gap in providing personalized, actionable feedback on what students should learn or practice next.

Recently, Large Language Models (LLMs) have emerged as powerful tools in educational applications, but they often face challenges related to hallucinations—generating incorrect or misleading information. To mitigate this issue, Retrieval-Augmented Generation (RAG) was introduced [1]. RAG enhances LLM outputs by incorporating relevant context retrieved through a maximum inner product search in a vector database, thereby improving the accuracy and relevance of the generated responses.

In this paper, we introduce AlgoAce, an LLM-powered assistant designed to recommend relevant Codeforces problems to users based on their submission history, using a RAG architecture. A demo video is available[2], as well as the code[3].

2. Related Work

LLMs have been widely applied to educational tasks, including student performance prediction, personalized practice generation, and automated feedback [2, 3, 4, 5]. RAG, in particular, has shown promise in improving the quality and relevance of educational outputs by grounding LLM responses in external knowledge [6, 7, 8, 9].

CSEDM'25: Workshop, July 20, 2025, Palermo, Sicily, Italy

anavagrawal2309@gmail.com (A. Agrawal); jill-jenn.vie@inria.fr (J. Vie)

https://jjv.ie (J. Vie)

0009-0005-8283-9110 (A. Agrawal); 0000-0002-9304-2220 (J. Vie)

[1]https://codeforces.com

[2]Demo video: https://drive.google.com/file/d/1M2dBUxHcES4RTFKhKwvP2BJCigYMvdfw/view?usp=sharing

[3]Source code: https://github.com/AnavAgrawal/AlgoAce

Recent studies have explored programming exercise recommendation systems [10, 11, 12, 13] and AI support in competitive programming environments [14, 15], yet few approaches have leveraged user submission history for fine-grained personalization. While course recommendation using LLMs has seen substantial progress—such as RAMO [16], which uses RAG to recommend MOOCs based on course metadata—there remains a gap in personalized exercise recommendation, particularly in competitive programming where detailed skill modeling is essential.

AlgoAce addresses this gap by incorporating users' historical problem attempts and performance data into its recommendation pipeline. This enables the system to dynamically adapt to a learner's evolving skill set, offering targeted practice.

3. Our RAG solution AlgoAce

On the open source platform AlgoAce, users input a query, e.g. "What should I learn next?". They can provide their Codeforces handle, which allows the agent to retrieve their submission history—notably successful and unsuccessful problem attempts—via the Codeforces API. If no handle is provided, generic recommendations are made based on the query alone. Sample outputs are provided in Appendix A.

Each Codeforces problem is represented by the following metadata: its title, associated tags (e.g., "shortest paths"), Elo rating, and URL, see an example in Table 1 in Appendix C. The difference in Elo rating between a user and a problem is proportional to the logit of the probability that the user can solve the problem. AlgoAce embeds this available metadata into a dense vector space to enable semantic retrieval. The user submission history contains metadata of the attempted problem and the result of the submission (OK, Wrong Answer, or Time Limit Exceeded), see an example in Table 2 in Appendix C. It is similar to the data encountered in knowledge tracing scenarios.

AlgoAce runs two parallel retrievers, one over user's past submissions and another over the Codeforces problem set, both using vector similarity search. The retrieved results are merged and then fed into a structured prompt to guide the LLM's generation. The system is designed with modularity in mind, currently supporting both API-based large language models such as OpenAI's GPT models and locally hosted open-source alternatives, so that the data never leaves the device. The complete prompt templates are provided in Appendix B. A diagram of the whole pipeline is also provided in Figure 2 in Appendix D.

To evaluate the retrieval of submission history, we prioritized ensuring the relevance of retrieved documents. Our initial observations revealed a critical need for query reformulation: queries that didn't explicitly define terms like "OK" or "time limit exceeded" consistently yielded irrelevant submissions. This led us to implement a two-way prompting strategy, where the first prompt is specifically designed to reformulate the user's query, thereby improving the relevance of the retrieved submission history.

Implementation details AlgoAce is implemented using the Pathway framework, which provides efficient Retrieval-Augmented Generation (RAG) pipelines. For approximate nearest neighbor search, we use Pathway's `KNNIndex` class, which applies locality-sensitive hashing to perform fast similarity search across embedded problems. The user interface is built using Streamlit, with a Pathway RESTful API connecting the front-end to the backend services. All code from the platform is released on GitHub.

4. Conclusion

As future work, we plan to incorporate problem solutions into the retrieval and recommendation process, and conduct user studies to evaluate our tool and see the common queries made by users. By analyzing official or community-provided solutions, we can better estimate the prerequisite skills required for each exercise, leading to more accurate and pedagogically meaningful recommendations. Additionally, we aim to align the objective of the LLM more closely with learning outcomes, ensuring that the system prioritizes educational progress rather than optimizing user satisfaction. This could involve fine-tuning prompts to emphasize skill development and problem-solving strategies.

Declaration on Generative AI

During the preparation of this work, the authors used LLMs like ChatGPT or Mistral in order to perform grammar, spelling check, and help with formatting. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, Advances in Neural Information Processing Systems 33 (2020) 9459–9474.

[2] L. Zhang, J. Lin, C. Borchers, J. Sabatini, J. Hollander, M. Cao, X. Hu, Predicting Learning Performance with Large Language Models: A Study in Adult Literacy, 2024. URL: https://arxiv.org/abs/2403.14668.

[3] N. Kiesler, D. Schiffner, Large language models in introductory programming education: Chatgpt's performance and implications for assessments, 2023. URL: https://arxiv.org/abs/2308.08572. arXiv:2308.08572.

[4] S. P. Neshaei, R. L. Davis, A. Hazimeh, B. Lazarevski, P. Dillenbourg, T. Käser, Towards modeling learner performance with large language models, 2024. URL: https://arxiv.org/abs/2403.14661. arXiv:2403.14661.

[5] F. Ai, Y. Chen, Y. Guo, Y. Zhao, Z. Wang, G. Fu, G. Wang, Concept-aware deep knowledge tracing and exercise recommendation in an online learning system, in: EDM, International Educational Data Mining Society (IEDMS), 2019.

[6] S. Jacobs, S. Jaschke, Leveraging lecture content for improved feedback: Explorations with GPT-4 and retrieval augmented generation, in: 2024 36th International Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2024, pp. 1–5.

[7] O. Henkel, Z. Levonian, C. Li, M. Postle, Retrieval-augmented generation to improve math question-answering: Trade-offs between groundedness and human preference, in: Proceedings of the 17th International Conference on Educational Data Mining, 2024, pp. 315–320.

[8] S. S. Manathunga, Y. A. Illangasekara, Retrieval augmented generation and representative vector summarization for large unstructured textual data in medical education, 2023. URL: https://arxiv.org/abs/2308.00479. arXiv:2308.00479.

[9] Z. F. Han, J. Lin, A. Gurung, D. R. Thomas, E. Chen, C. Borchers, S. Gupta, K. R. Koedinger, Improving assessment of tutoring practices using retrieval-augmented generation, 2024. URL: https://arxiv.org/abs/2402.14594. arXiv:2402.14594.

[10] Y. Liu, R. Zhu, M. Gao, Personalized Programming Guidance based on Deep Programming Learning Style Capturing, 2024. URL: https://arxiv.org/abs/2403.14638.

[11] S. Branchi, C. D. Francescomarino, C. Ghidini, D. Massimo, F. Ricci, M. Ronzani, Learning to act: a reinforcement learning approach to recommend the best next activities, 2022. URL: https://arxiv.org/abs/2203.15398. arXiv:2203.15398.

[12] P. Agarwal, A. Gupta, R. Sindhgatta, S. Dechu, Goal-oriented next best activity recommendation using reinforcement learning, 2022. URL: https://arxiv.org/abs/2205.03219. arXiv:2205.03219.

[13] J. Vassoyan, J.-J. Vie, P. Lemberger, Towards scalable adaptive learning with graph neural networks and reinforcement learning, 2023. URL: https://arxiv.org/abs/2305.06398. arXiv:2305.06398.

[14] P. Veličković, A. Vitvitskyi, L. Markeeva, B. Ibarz, L. Buesing, M. Balog, A. Novikov, Amplifying human performance in combinatorial competitive programming, 2024. URL: https://arxiv.org/abs/2411.19744.

[15] J. Frej, N. Shah, M. Knezevic, T. Nazaretsky, T. Käser, Finding paths for explainable mooc recommendation: A learner perspective, in: Proceedings of the 14th Learning Analytics and Knowledge Conference, LAK '24, ACM, 2024, p. 426–437. URL: http://dx.doi.org/10.1145/3636555.3636898. doi:10.1145/3636555.3636898.

[16] J. Rao, J. Lin, RAMO: Retrieval-Augmented Generation for Enhancing MOOCs Recommendations, 2024. URL: https://arxiv.org/abs/2407.04925.

A. Sample Outputs

Examples of the output generated by AlgoAce are provided in Figure 1. We observe that the LLM is explaining the recommendations made by looking at user submission history.

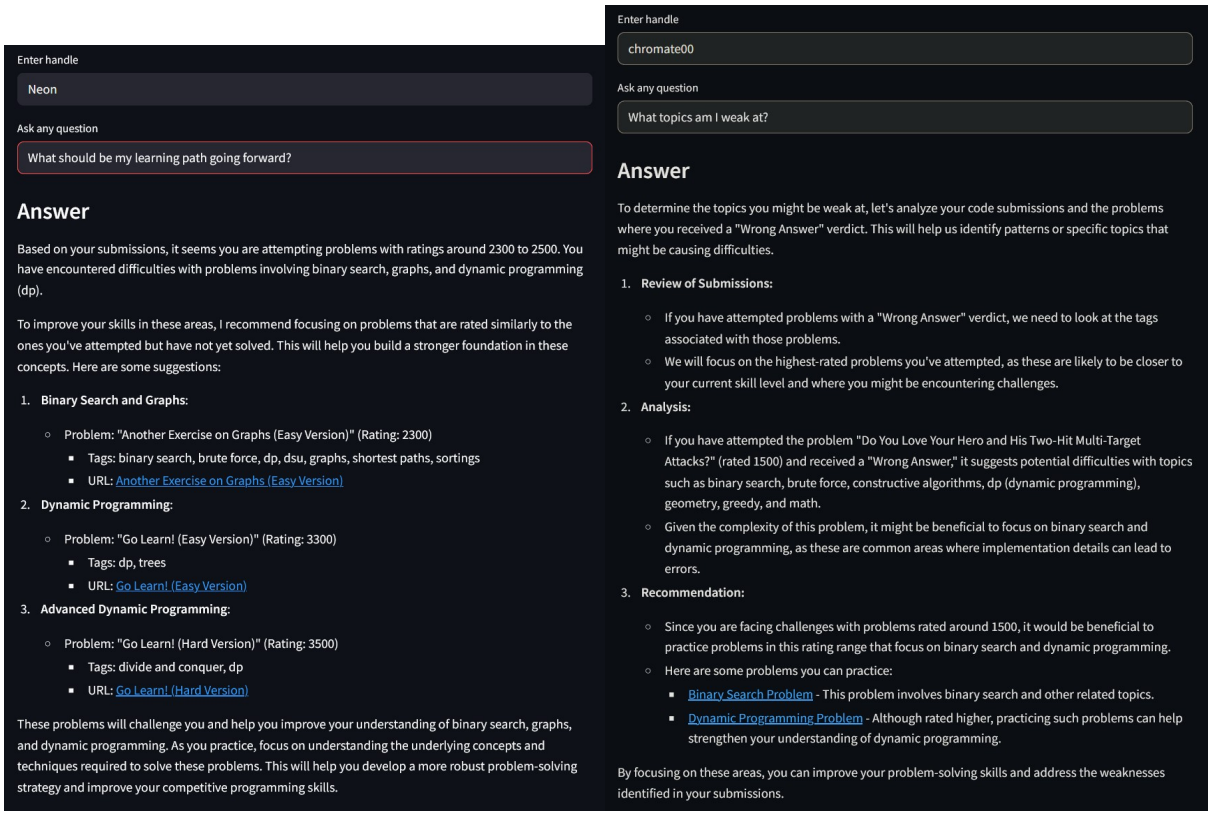

Figure 1: Examples of AlgoAce's output for problem recommendations and strategies.

B. Prompt Template

B.1. Query Rewriting Prompt

The query rewriting prompt is used to reformulate user queries with explicit metadata terms, improving retrieval accuracy:

```
You are a query rewriting assistant for a competitive programming tool.
The goal is to rewrite user queries so that they include concrete terms that match
problem metadata stored in a database. The database retrieves problems based on word
overlap with the query - so include exact terms like problem 'tags' (e.g. 'greedy',
'dp', 'graphs'), verdicts (e.g. 'WRONG_ANSWER', 'OK', 'RUNTIME_ERROR'), difficulty
'ratings' (e.g. 800,1000, 1700), or specific problem names if known.
You do NOT have access to the user's past submissions, but you must guess the user's
intent and include as many relevant keywords as possible to help the system match the
query to relevant problems. Your output should be in the following format:
Retrieval terms: <insert keywords for retrieval>
```

Table 1
Example of Codeforces problems

Rating	Name (URL)	Tags
1700	Mathematical Problem	brute force, constructive algorithms, geometry, math
1200	Training Before the Olympiad	constructive algorithms, games, greedy, implementation, math
1000	Two Divisors	constructive algorithms, math, number theory
800	2023	constructive algorithms, implementation, math, number theory
1800	Bicycles	graphs, greedy, implementation, shortest paths, sortings

Table 2
Example of user submission history

Rating	Verdict	Name (URL)	Tags
2300	OK	Farm Game	combinatorics, games
2100	OK	Learning to Paint	binary search, data structures, dfs and similar, dp, greedy, implementation, sortings
1900	Timeout	Yet Another Permutation Constructive	*special, constructive algorithms
1900	OK	Yet Another Permutation Constructive	*special, constructive algorithms
2000	Wrong	Narrow Paths	*special, combinatorics
2000	OK	Narrow Paths	*special, combinatorics

```
User query: <natural language query reformulated for the LLM>

Make sure the retrieval terms are rich in specific tokens, and the user query is clear
and helpful for the LLM.
```

B.2. Response Generation Prompt

The response generation prompt guides the LLM in analyzing user submissions and generating tailored recommendations or explanations:

```
You are a competitive programming coach.
I will give you the user code submissions and a list of all available problems after
"Data:". The data lines with verdict are the user code submissions. The other lines are
the available problems on codeforces. Answer the user query after looking at the user
code submissions and the questions he got a wrong answer in. The problems you analyze
and suggest should only be from the given data lines without verdict and should be
similar in rating to the highest rated problems that the user has attempted.
The answer should also explain how you arrived at the answer looking at the user code
submissions if necessary. Keep your thinking short.
Example query : "What problems should I practice next?"
Ideal Answer : "Looking at your submissions, you solve questions rated around 1500 and
you are facing difficulty in implementing binary search. So here are some binary
search problems you can practice: [Links to Binary search
problems]."
Example query : "How to implement dynamic programming?"
Ideal Answer : "Here's an implementation of dynamic programming. [Dynamic Programming
Code].\n
Data: \n {context} \n{query} /think.
```

C. Sample Codeforces data

Examples of Codeforces problems are provided in Table 1. An example of user submission history is provided in Table 2.

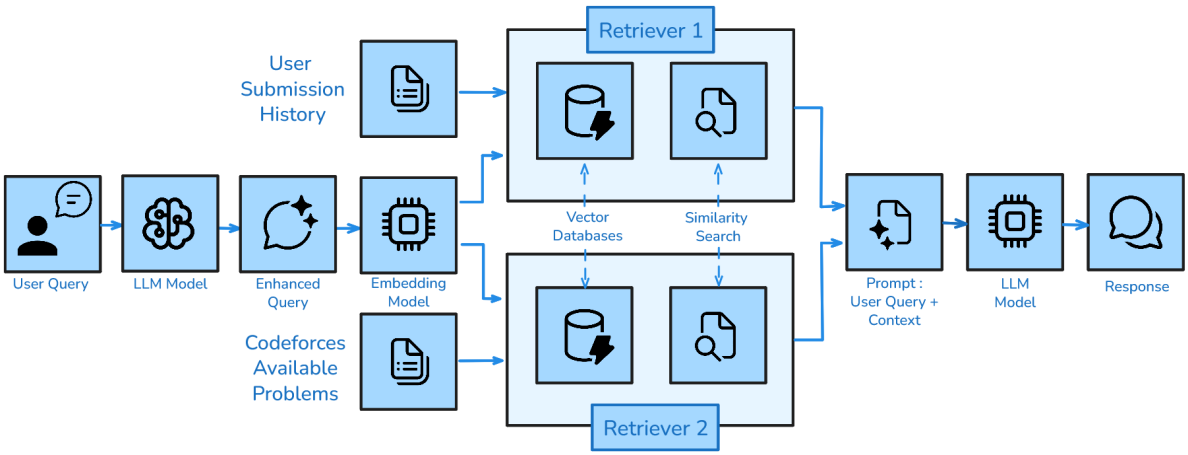

Figure 2: RAG pipeline used by AlgoAce

D. RAG pipeline

Our pipeline is provided in Figure 2.