WIP: MENTOR: Machine-Enhanced Tutor for ORientation in Programming

Gabriel Bonetti, Guilherme Brandão, Liana Duenha
Faculty of Computing - Federal University of Mato Grosso do Sul
Campo Grande, Brazil
{gabriel.bonetti,guilherme_brandao,liana.duenha}@ufms.br

***Abstract*—This innovative practice WIP paper describes MENTOR, a Machine-Enhanced Tutor for ORientation in Programming designed to support students in introductory programming courses. The system interacts with students through natural language, using conceptual explanations, step-by-step hints, and feedback to guide users in solutions, while intentionally avoiding the direct generation of complete solutions. The system has been developed for use in an introductory programming course at the Federal University of Mato Grosso do Sul (UFMS), focusing on Python. MENTOR was evaluated with 130 undergraduate computing students through a structured protocol combining a profile questionnaire, a Python-based technical assessment, and a multidimensional user experience survey. Preliminary results indicate that MENTOR acts as a catalyst for active learning, enabling students to transition from passive code consumption to structured problem-solving thinking.**

***Index Terms*—AI in Education, Intelligent Tutoring Systems, Introductory Programming, Computing Education.**

I. Introduction

Introductory programming courses are known for their learning challenges, with failure rates often reaching 50% [1]. Large classes make things harder by reducing chances for individual help during labs and practical work. This highlights the need for extra tools that give ongoing, immediate support while still letting instructors lead the teaching. Unlike traditional laboratory sessions where instructors become bottlenecks for debugging help, this automated approach provides instantaneous, individualized pedagogical scaffolding, ensuring students remain engaged in active problem-solving rather than waiting for assistance.

Recent progress in large language models (LLMs) has opened up new ways to support students. With the right teaching guidelines, these systems can act as virtual tutors, helping students understand ideas and think about their solutions instead of just giving them full answers.

This paper introduces MENTOR (**M**achine-**En**hanced **T**utor for **OR**ientation in Programming), an AI-based tutoring system for beginner programming courses. MENTOR talks with students in natural language and uses both the Socratic Method and Computational Thinking ideas to help them break down problems, spot patterns, and use logic, without giving away full solutions. The main goals of MENTOR are:

- Helping students better understand programming concepts by giving guidance that adapts to the difficulty of each problem;
- Raising course success rates and lowering dropout rates, which are important for the university;
- Making teaching more efficient by offering automated, round-the-clock tutoring that supports instructors in large classes;
- Achieving high user satisfaction. The system received an average rating of 3.90/5 for usability, learning support, response quality, and interaction.

MENTOR was created for the introductory programming course at the Federal University of Mato Grosso do Sul (UFMS), focusing on Python. The evaluation used an iterative approach, where 130 undergraduate students from five computing programs tried the system. Data was collected in three steps: a profile questionnaire, a technical test with 15 Python questions at different levels, and a user experience survey. Based on student feedback, the platform was improved; subsequently, a second round of testing measured the effects of those changes. This paper focuses exclusively on the findings from the first phase of this evaluation, presenting the initial deployment results and baseline student perceptions to establish the tool's foundational efficacy.

The rest of this paper is organized as follows. Section II covers background and related work. Section III explains the system's design and teaching approach. Section IV details how the system was tested and the results. Section V wraps up and suggests future directions.

II. Background and Related Work

Artificial intelligence has made educational technologies better, moving them from simple content delivery to more interactive and personalized learning [2]. Effective learning environments require guidance, support, and opportunities for reflection, not just question-and-answer exchanges [3]. MENTOR is built on these principles, encouraging active learning through interactive engagement rather than acting as a standard chatbot.

A. Intelligent Tutoring Systems for Programming

Researchers have studied Intelligent Tutoring Systems (ITS) for programming for many years. Early examples like Lisp Tutor [4] and ProPL [5] compared student answers to expert solutions. These systems work well for structured exercises,

TABLE I: Comparison of AI Tutoring Platforms for Programming

Feature	MENTOR	CodeHelp	CS50Duck	ChatGPT
Scaffolding policy	✓	✓	✓	-
Code execution	✓	-	-	✓*
RAG grounding	✓	-	-	-
Content filtering	✓	Partial	Partial	-
Learning analytics	✓	-	-	-
Multilanguage	✓	-	-	✓ *

(*)Via interpreter, not self-hosted, whereas MENTOR is deployed within the institution's own infrastructure.

but they usually cannot have natural conversations with students. They can spot mistakes, but often have trouble explaining concepts or giving deeper help.

B. LLM-Based Educational Tools

Large language models (LLMs) have enabled a new generation of educational tools, but with important distinctions. Code-generation tools like Copilot [6] assist with writing code but are not designed for teaching — they optimize for code completion rather than conceptual understanding, and may reinforce errors without proper pedagogical guidance [7].

Some tools, like CodeHelp˜ [8] and CS50's "Duck" debugger, try to guide students without giving them the answers directly. However, these are usually separate chatbots that do not connect well with course materials, coding environments, or school systems. To solve these problems, researchers have looked into hybrid approaches. Retrieval-Augmented Generation (RAG) [9] combines LLMs with outside knowledge sources, which helps reduce mistakes and makes answers more reliable.

C. Positioning of MENTOR

MENTOR goes beyond single-purpose tools by integrating four capabilities in a single, self-hosted platform: (1) LLM-based tutoring with pedagogical constraints that adapt to problem complexity, (2) RAG grounded in the instructor's course materials, (3) secure code execution with real compiler feedback, and (4) learning analytics for instructors. This combination allows students to receive contextually relevant guidance while writing and testing code in the same environment. Table I summarizes how MENTOR compares to existing tools.

III. System Architecture and Pedagogical Design

MENTOR uses a microservices architecture managed by Docker Compose. It includes six containerized services: `Nginx` as a reverse proxy with secure socket layer (SSL) and rate limiting, a `Next.js` 14 frontend, and a FastAPI [10] backend. PostgreSQL 16 with `pgvector` handles both relational and vector storage. Redis is used for caching and task queues. Isolated Docker sandboxes provide secure code execution. The entire system is self-hosted within the institution.

The frontend provides a chat-based interface with a built-in code editor. Specifically, the editor supports syntax highlighting and line numbers for Python, JavaScript, C, and C++. In addition, a terminal panel shows execution output and supports `stdin`. Furthermore, the system allows file uploads for PDF, DOCX, and images. Finally, students can manage several sessions and choose between light and dark themes.

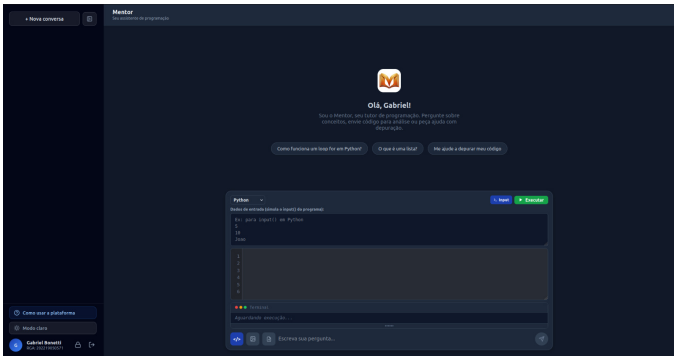


Fig. 1: MENTOR main interface.

Figure 1 shows the main interface of the MENTOR platform, which has three main sections. On the left, a sidebar lets users access past conversations, platform help, settings like light or dark mode, and user information. The center is the main area, where students see a welcome message and example prompts to help them get started, such as asking questions or getting debugging help. At the bottom, there is a workspace for code, with a field for program input, a code editor that supports multiple languages, and a terminal to show results. Students can also use a chat box to talk with the tutor in natural language. This setup lets students easily switch between asking questions, writing code, and running it all in one place.

A. Pedagogical Strategy

MENTOR blends the Socratic Method with Computational Thinking, based on Vygotsky's Zone of Proximal Development [11] and scaffolding [12]. The system adjusts the depth of its responses as needed. It aims to build four main skills: decomposition, pattern recognition, abstraction, and mental testing. The system handles interactions in five ways. Simple questions get direct corrections. Medium problems come with similar examples and guiding prompts. Complex problems are explored using Socratic questions instead of giving full solutions right away. Code with errors gets targeted feedback. The correct code is checked with explanations and optional challenges. All examples use real, executable code instead of pseudocode.

A three-layer filtering system keeps the focus on education. It detects prompt injection, including encoded inputs. The system blocks offensive or off-topic content and checks if messages are relevant to programming. Any remaining messages are further limited by the LLM system prompt.

B. Retrieval-Augmented Generation

MENTOR uses a RAG pipeline [9] to match responses with course materials. Content is split into chunks of 500 tokens with a 50-token overlap and embedded using `text-embedding-3-small`. The embeddings are stored

in `pgvector`. When a query is made, the system retrieves the top three relevant chunks (cosine similarity ≤ 0.75) and adds them to the model context.

C. Secure Code Execution

The system supports code execution for Python 3.12, JavaScript (Node.js 20), C (GCC 14), and C++ (G++ 14) in isolated Docker containers. Each execution uses a temporary container with no network access and a read-only filesystem. Each container has 128MB of memory, 50% CPU, and a 10-second timeout. Containers run as non-root users without Linux capabilities. Code is sent using `base64` encoding. Any compilation errors are reported with detailed feedback.

IV. Experimental Evaluation

A. Evaluation Methodology

The evaluation of MENTOR was conducted with undergraduate students from five computing-related programs, who self-classified their programming proficiency as beginner, basic, intermediate, or advanced. The study was carried out within a dedicated virtual learning environment developed for the experiment. Students from the introductory programming classes were invited to participate voluntarily, integrating the tool usage as an optional complementary activity to their standard coursework. The evaluation protocol consisted of three sequential steps:

1) **Student Profile Questionnaire:** Participants first completed a questionnaire to collect demographic and academic information, allowing characterization of the sample.
2) **Technical Assessment:** Students answered a set of 15 programming questions covering core topics of an introductory programming course. The questions were divided into three levels (five per level), following the course syllabus. Level I included questions requiring the use of conditional and repetition structures, Level II focused on one-dimensional data structures (arrays), and Level III explored two-dimensional structures (matrices).
   Testing Levels II and III early in the semester was a deliberate choice to evaluate MENTOR's ability to scaffold concepts not yet covered in lectures, simulating independent self-learning scenarios.
   All responses required Python code and were automatically graded by the platform. Each question allowed up to two submissions, with only the final submission considered for evaluation.
3) **User Experience Survey:** Finally, participants completed a 23-question survey evaluating their experience with MENTOR. The survey was structured around five key dimensions: *Usability, Learning Support, Response Quality, Human–Computer Interaction,* and open-ended *Feedback*. The first 20 items were objective statements rated on a five-point Likert scale (1 = strongly disagree to 5 = strongly agree). The final three items were open-ended questions, enabling participants to provide qualitative feedback and suggestions for improving the tool.

B. Results

A total of 181 students initially enrolled in the study. However, the results presented are based on the responses of the 130 students who completed the study stages, including the profile, at least one complete technical questionnaire, and the user experience survey.

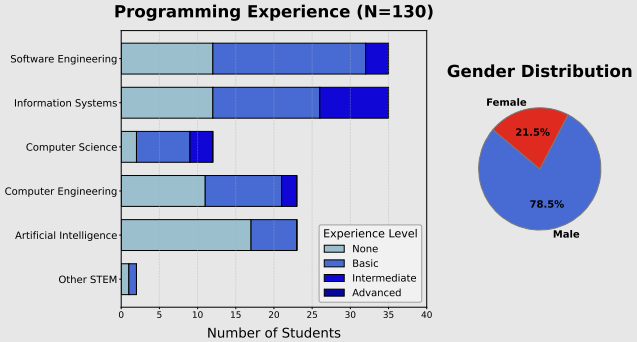


Fig. 2: Profile data for the 130 participants, including gender and programming skills, grouped by undergraduate program.

Figure 2 shows the profile data for the participants, including gender and self-reported programming skills, grouped by undergraduate program. The sample includes students from various computing programs, with most coming from Software Engineering and Information Systems. Since the semester has just started, most students have only been learning programming for two weeks. Most students had no prior experience or basic programming skills, fewer reported intermediate skills, and only a small number had advanced skills. This means the evaluation mainly represents beginners, which matches the setting of introductory programming courses. In terms of gender, 78.5% of the group is male, and 21.5% female. This gender imbalance is common in computing programs.

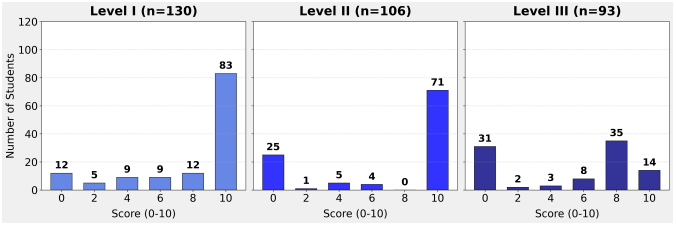


Fig. 3: Distribution of student performance across difficulty levels.

Figure 3 shows the student performance distributed across the three levels of the technical questionnaire. There is a strong concentration of high scores (8–10) in Levels I and II, with 73% and 67% of students, respectively, achieving results in this top range. This indicates that most participants were able to successfully solve problems of lower and intermediate difficulty. In contrast, Level III displays a more dispersed score distribution, with only 53% of students reaching the highest

range and a noticeable increase in lower score intervals. It is important to note that the majority of participants were in the first weeks of an introductory programming course and reported low levels of prior experience. In this context, the fact that most students were able to complete the assessment, even with some difficulty, provides evidence of the effectiveness and practical usefulness of MENTOR in supporting early-stage learning.

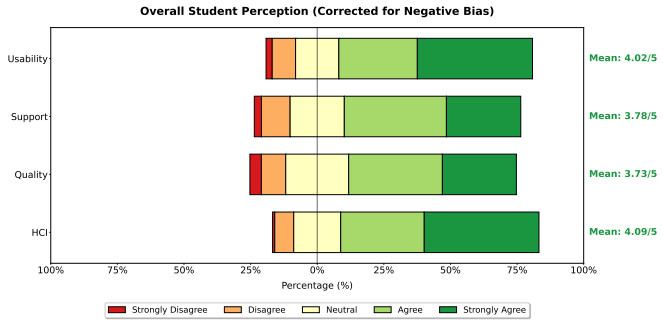
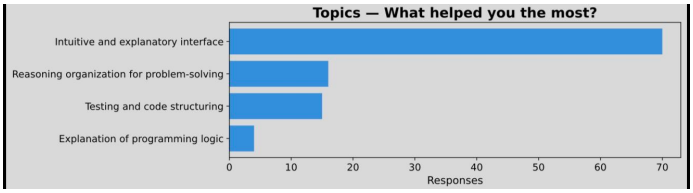


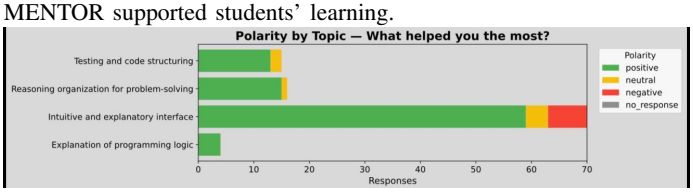
Fig. 4: MENTOR students' perception based on the user experience questionnaire.

Figure 4 presents the overall student perception of MENTOR based on the user experience questionnaire, organized into dimensions: Usability, Learning Support, Response Quality, and Human-Computer Interaction (HCI). The results indicate a predominantly positive evaluation across all dimensions, with a clear concentration of responses in the "agree" and "strongly agree" categories. HCI achieved the highest average score (4.09/5), followed closely by usability (4.02/5), suggesting that students found the system intuitive, accessible, and easy to interact with. Learning Support (3.78/5) and Response Quality (3.73/5), while slightly lower, still demonstrate favorable perceptions, indicating that MENTOR effectively contributes to the learning process and provides generally positive feedback.

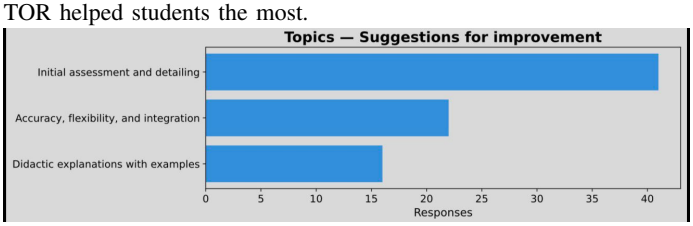
A semi-automated qualitative analysis was conducted to examine students' open-ended responses. Topic extraction techniques were applied to identify recurring themes, followed by manual refinement to ensure semantic consistency. For each question, the frequency of topics, their associated sentiment polarity, and the overall polarity distribution were computed. The sentiment analysis reveals a predominantly positive perception across all categories, especially regarding interface and interaction aspects, as shown in Figures 5a and 5b. The remaining topics also exhibit largely positive sentiment, indicating that, although less frequently mentioned, they were consistently well evaluated. Overall, these results suggest that MENTOR's main contribution lies in providing clear and intuitive interaction while supporting structured reasoning and coding practices.



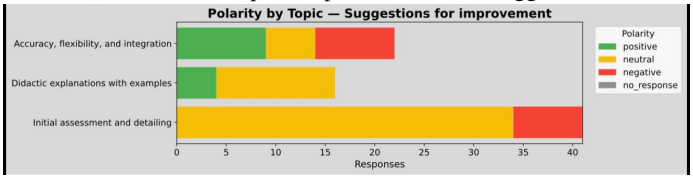
(a) Distribution of the most frequently mentioned topics on how MENTOR supported students' learning.



(b) Sentiment polarity distribution across topics related to how MENTOR helped students the most.



(c) The most frequent topics in students' suggestions.



(d) Sentiment polarity for the main topics identified in improvement suggestions.

Fig. 5: Overview of topic frequency and sentiment polarity.

Complementarily, Figures 5c and 5d show the analysis of improvement suggestions, indicating the most frequent aspect concerns initial assessment and problem detailing, followed by accuracy, flexibility, integration, and requests for more didactic explanations. In contrast to the previous results, the sentiment distribution for these suggestions is more balanced, with a predominance of neutral polarity and some negative mentions, particularly regarding system precision and contextual understanding. Together, these findings indicate that, although MENTOR is positively perceived, improvements are mainly centered on enhancing contextual interpretation, response accuracy, and adaptability to students' needs.

V. Conclusion

This study presents MENTOR, a self-hosted artificial intelligence tutor designed for introductory programming courses. Results indicate that the system supports student progression and reduces initial learning barriers, with positive perceptions across usability, interaction, and learning support. Additionally, the findings demonstrate the feasibility of integrating pedagogically guided LLM interaction, curriculum-aligned responses, and secure code execution within a unified educational platform.

A limitation of the current study is its focus on a single institution and a predominantly beginner demographic. This context limits the immediate generalizability of the findings, indicating the need for further research to examine how MENTOR performs in more advanced computing courses and across diverse institutional settings.

Based on student feedback, future work will pursue four primary directions:

- Conducting longitudinal studies to evaluate the impact of MENTOR on student performance, approval rates, and dropout reduction over an academic semester;
- Performing comparative experimental studies between MENTOR and other dedicated or general-purpose AI tutors to benchmark its pedagogical effectiveness and scaffold quality;
- Assessing the instructor-facing analytics dashboard to identify common student challenges and inform data-driven teaching interventions; and
- Implementing lightweight, continuous feedback mechanisms to capture real-time student perceptions of response quality and further refine the system.

VI. References

[1] Lauren E Margulieux, Briana B Morrison, and Adrienne Decker. Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, 7(1):19, 2020.

[2] Wayne Holmes, Maya Bialik, and Charles Fadel. *Artificial Intelligence in Education: Promises and Implications for Teaching and Learning*. Center for Curriculum Redesign, Boston, USA, 2019.

[3] R. Keith Sawyer, editor. *The Cambridge Handbook of the Learning Sciences*. Cambridge University Press, Cambridge, UK, 2 edition, 2014.

[4] John R. Anderson, C. Franklin Boyle, and Brian J. Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.

[5] H. Chad Lane and Kurt VanLehn. Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education*, 15(3):183–201, 2005.

[6] GitHub. GitHub Copilot: Your AI pair programmer. ht tps://github.com/features/copilot, 2021.

[7] James Prather et al. The robots are here: Navigating the generative AI revolution in computing education. In *Proceedings of the ACM Conference on Global Computing Education (CompEd)*, pages 108–116. ACM, 2023.

[8] Mark Liffiton, Brett E. Sheese, Jaromir E. Savelka, and Paul Denny. CodeHelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*, pages 394–400. ACM, 2023.

[9] Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474, 2020.

[10] Sebastián Ramírez. FastAPI: Modern, fast web framework for building APIs with Python. https://fastapi.tiangolo.com, 2018.

[11] Lev S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.

[12] David Wood, Jerome S. Bruner, and Gail Ross. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17(2):89–100, 1976.