CoderBot 2.0: Integrating LLM and Prompt Engineering in the Evolution of an Educational Chatbot

Andre Mendes[1], Arthur Parizotto[1], Renato Garcia[4], Marcelino Garcia[3] Gilleanes Guedes[1,2], Ricardo Vilela[5], Pedro Valle[4], Renato Balancieri[3] Williamson Silva[6,2]

[1]Universidade Federal do Pampa - UNIPAMPA (Alegrete), Alegrete, RS, Brasil

[2]PPGES (UNIPAMPA - Alegrete), Alegrete, RS, Brasil

[3]Universidade Estadual de Maringá (UEM), Maringá, PR, Brazil

[4]Instituto de Matemática e Estatística (IME-USP), São Paulo, SP, Brasil

[5]Universidade de Campinas (FT/Unicamp), Limeira, SP, Brazil

[6]Universidade Federal do Cariri (UFCA), Juazeiro do Norte, CE, Brazil

{[1]andremiranda.aluno,[1]arthurparizotto.aluno}@unipampa.edu.br

[1,2]gilleanesguedes@unipampa.edu.br,[5]rfvilela@unicamp.br

{[4]pedrohenriquevalle,[4]renato.s.garcia}@@usp.br

[3]rbalancieri@uem.br

[6]williamson.silva@ufca.edu.br

Abstract. *Teaching programming to beginner students remains a significant challenge, as it requires the development of complex cognitive skills. Given this, this article presents CoderBot 2.0, an evolution of a pedagogical agent designed for introductory programming instruction based on Example-Based Learning (EBL). The new version integrates Large Language Models (LLMs) and prompt engineering techniques to generate both correct and erroneous examples on demand, as well as code explanations and progressive feedback, without relying on fine-tuning. Aligned with Cognitive Load Theory, its architecture promotes metacognitive strategies such as self-explanation, fading, and contextual variation. The system offers adaptive support, adjusting to the student's level and encouraging engagement, autonomy, and reflection.*

1. Introduction

Teaching programming is widely recognized as a multifaceted, didactic, and cognitive challenge. Beyond syntax, students need to develop skills in mental modeling of execution flow, incremental debugging, and creative problem solving (Robins et al., 2003). Learning strategies as Worked Examples (WE) (Sweller et al., 2011), which make the underlying reasoning behind problem-solving explicit (Sweller et al., 2011); Socratic questioning (Fakour et al., 2025), which encourages self-reflection and critical thinking (Fakour et al., 2025); and gradual feedback (Hattie and Timperley, 2007), which guides learning through progressive hints without directly providing the final solution (Hattie

DOI: 10.5753/sbie.2025.12321

and Timperley, 2007), have proven effective in supporting structured and metacognitive learning.

Example-Based Learning (EBL) consolidates these instructional strategies in light of Cognitive Load Theory (Sweller et al., 2011; Adams et al., 2014), promoting the internalization of strategies through exposure to worked examples—both correct and/or erroneous. These examples are designed to foster meaningful learning by combining structured solution presentations with metacognitive mechanisms such as self-explanations, fading, and context variation (Renkl, 2014a; Kopp et al., 2008). Worked examples are cognitive representations guided by evidence, making expert thinking accessible and understandable to novice learners Renkl (2014b, 2017).

In this context, the advancement of Large Language Models (LLMs), such as GPT-4, introduces new possibilities for generating scalable, personalized, responsive, and interactive instructional content. Recent studies show that, when guided by appropriate *prompt engineering* strategies—such as chaining, persona prompting, and few-shot learning Jury et al. (2024)—LLMs are capable of generating pedagogically plausible *worked examples*, as well as supporting dialogical and contextual interactions with students (Jury et al., 2024). However, despite their potential, there is still a lack of approaches that integrate these resources into cohesive and robust pedagogical frameworks, aligned with learning theories and validated empirically (Pirzado et al. (2024); Puech et al. (2025); Jury et al. (2024)).

This paper introduces CoderBot 2.0, an evolution of the original pedagogical agent, now enhanced with LLMs and pedagogical prompt engineering. The new version enables the dynamic generation of examples tailored to the student's profile, combining contextualized explanations with an integrated educational environment that focuses on personalizing the learning experience.

2. Theoretical Background and Related Work

2.1. Cognitive Load Theory and Example-Based Learning

Cognitive Load Theory provides an essential framework for instructional design in complex domains, such as programming, emphasizing the limits of working memory and the need to balance different types of cognitive load (Baddeley, 1992; Shaffer et al., 2003; Sands, 2019). This theory distinguishes three components: (i) **intrinsic load**, related to the complexity of the content; (ii) **extraneous load**, associated with the way information is presented; and (iii) **germane load**, related to the effort dedicated to meaningful learning. To promote learning, it is necessary to minimize extraneous load and maximize germane load, particularly in the early stages of learning. Strategies such as task segmentation, combined use of visual and verbal resources, and clear instructions are recommended (Sands, 2019). In this scenario, worked examples stand out as one of the most effective approaches, as highlighted by EBL (Renkl, 2014b, 2017). By presenting step-by-step solutions—either correct or erroneous—these examples facilitate the construction of mental schemas, fostering conceptual understanding and critical thinking (Sweller et al., 2011; Atkinson et al., 2000; Beege et al., 2021; Chiarelli et al., 2022).

2.2. Related Work

Technology-assisted programming education has been addressed from different perspectives. Classical tutor systems, such as Cognitive Tutors (Anderson et al., 1995) and student-centered models (Woolf, 2009), have shown consistent gains but face high authoring costs (Keuning et al., 2018). More recent research explores the automatic generation of examples (Barros et al., 2021) and questions (Sychev et al., 2021), but it still relies on explicit rules or restricted statistical models, which limit scalability.

The advancement of LLMs has introduced new possibilities for personalized tutoring. Models such as GPT-3.5 and GPT-4 have already demonstrated potential to provide relevant feedback and mediate instructional interactions (Dai et al., 2023; Zhang et al., 2024). Among the initiatives, WorkedGen (Jury et al., 2024) stands out, utilizing prompt chaining and one-shot learning to generate interactive examples, although with conceptual limitations. Another example is AIIA (Artificial Intelligence-Enabled Intelligent Assistant) (Sajja and Ramesh, 2023), which integrates *quizzes*, flashcards, code execution, and summaries, but lacks alignment with well-established pedagogical theories.

CoderBot 1.0 was developed in this context as an educational agent based on EBL, offering correct and erroneous examples in a web environment, with distinct profiles for students and instructors (Garcia et al., 2025; Mendes et al., 2024). Although it showed initial gains in understanding and self-confidence, it had limitations in terms of flexibility and adaptation to users' proficiency levels (Villa et al., 2024). These findings motivated the creation of CoderBot 2.0, which incorporates LLMs and pedagogical prompt engineering strategies to provide more personalized, responsive, and integrated tutoring.

3. CoderBot 2.0: An Integrated Pedagogical Environment

The **CoderBot 2.0** was conceived as an evolution of its predecessor, overcoming the limitations of agents focused solely on static examples. It aims to position itself as an integrated pedagogical platform, grounded in established cognitive theories and enriched by LLM-based technologies. Unlike the first version, which focused on displaying predefined examples (Garcia et al., 2025; Mendes et al., 2024), CoderBot 2.0 offers an adaptive experience, tailored to the needs of each student.

The innovation in this version is anchored in the incorporation of a **structured pedagogical template for worked examples** (Renkl, 2014b), along with the addition of an **interactive board** for externalizing reasoning. The template organizes essential elements, including problem description, step-by-step solution, correct and incorrect examples, and reflective activities, promoting clarity and instructional consistency. The board, inspired by the principles of distributed cognition (Hutchins, 1995), acts as a visual space that expands the capacity to represent and manipulate concepts. This combination addresses recurring challenges in programming education, such as high cognitive load and the difficulty of abstracting complex structures (Hundhausen et al., 2002).

The new version is designed around three central pedagogical pillars: **worked examples** (the main focus of this research), **Socratic dialogue**, and **feedback in stages**. While all three pillars form the instructional design of the platform, this study primarily focuses on worked examples, investigating how their integration with LLMs and prompt

engineering can reduce cognitive load and foster structured learning in programming education. The pillars do not operate in isolation but are dynamically triggered through a catalog of configurable prompts. This instructional design strikes a balance between demonstration, questioning, and guidance, creating learning conditions that favor both instructional clarity and the development of critical thinking and student autonomy (Fakour et al., 2025; Atkinson et al., 2000).

CoderBot 2.0 represents a pedagogical reconfiguration grounded in Cognitive Load Theory (CLT) (Sweller et al., 2011) and Example-Based Learning (Renkl, 2014b). It is a modular architecture consists of three complementary modules: (i) a conversational intelligent tutor based on **pedagogical prompt engineering**; (ii) an interactive board that functions as a cognitive extension; and (iii) a coding environment, currently in development, integrated with **Continue**, aimed at reducing the effect of split attention.

Figure 1 summarizes this architecture, highlighting how the modules work together to provide a continuous, personalized, and adaptive learning journey. This integration differentiates CoderBot 2.0 from currently available fragmented solutions, positioning it as a systemic approach to programming education.

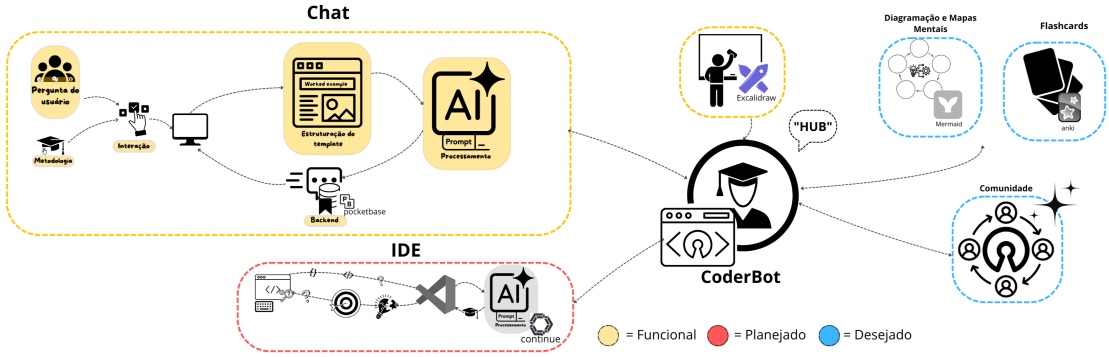

Figure 1. Overview of the CoderBot 2.0 architecture.

3.1. Module 1: Chat-based Tutoring with Pedagogical Prompt Engineering

This module of CoderBot 2.0 introduces an intelligent conversational tutor that overcomes the rule-based approach of the previous version by employing **Prompt Engineering** techniques to generate didactically structured responses via LLMs. Based on the intersection of cognitive science and educational technology, CoderBot 2.0 can be enhanced with **Retrieval-Augmented Generation** (RAG) (Li et al., 2023), connecting prompts to specific knowledge bases such as teaching materials or examples personalized by instructors. This combination ensures that interactions are technically correct, contextually relevant, and pedagogically effective.

3.1.1. Interface, Pedagogical Configuration, and Interaction Flow

The architecture of CoderBot 2.0 combines a pedagogical configuration panel with a responsive interaction flow, providing a learning experience adapted to the cognitive profile and needs of each student. The interface, illustrated in Figure 2, allows for the customization of the tutor based on three central axes: **LLM Model**, where the educator can select

from different language models (e.g., GPT-3.5, GPT-4, or Claude), considering technical and budgetary aspects; **Pedagogical Methodology**, where evidence-based approaches such as worked examples, Socratic dialogue, and feedback in stages can be activated; **Presentation Strategy**, where the system enables the use of didactic analogies to facilitate the connection between prior knowledge and new content.

These options form the core of **adaptive pedagogical personalization**, where not only the content but also its presentation is adjusted according to the cognitive characteristics and learning styles of the students. To illustrate the interface's structure, Figure 2 presents the chat module, highlighting its key functional elements with yellow visual cues. The interaction flow depicted is structured to enhance the pedagogical value of the chatbot's responses. This process unfolds through the following sequence:

1. **Contextual Initialization**, introducing the session with guidance and encouragement for active learning;
2. **Contextualized Query Processing**, considering the conversation history, pedagogical choices, and disciplinary context;
3. **Dynamic Prompt Construction**, integrating all didactic variables to form a command directed to the LLM;
4. **Pedagogically Structured Response Generation**, where the generated content follows the chosen instructional method, promoting progression in learning.

This operational flow is a direct application of the principles of Cognitive Load Theory, seeking to minimize extraneous load and optimize working memory resources. In doing so, it favors the construction of durable mental schemas and meaningful learning.

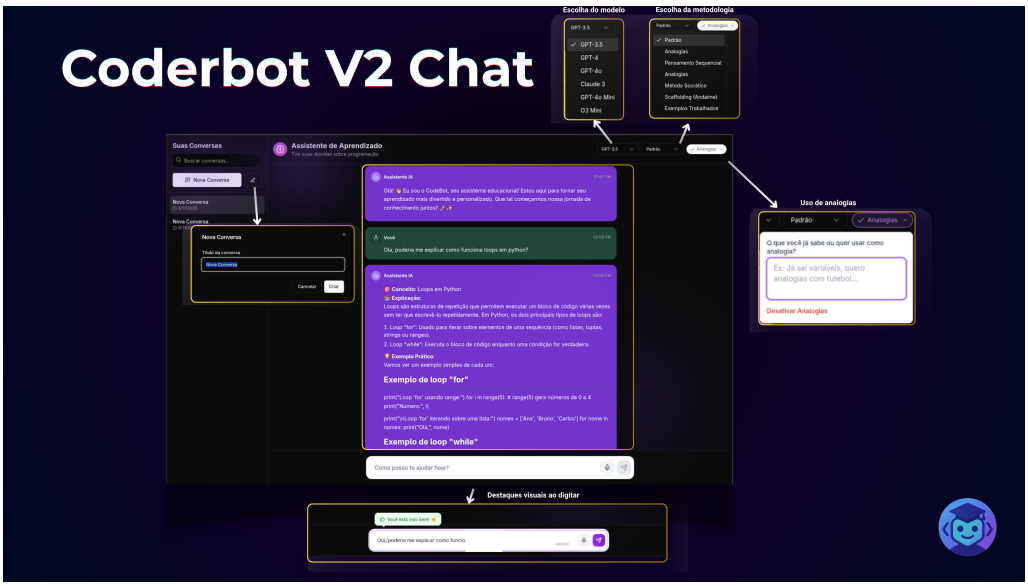

Figure 2. Main interface of the CoderBot 2.0 module: integrated chat area with input field and customizable pedagogical settings in the top header.

3.2. Module 2: Interactive Board for Externalizing Reasoning

The second module of CoderBot 2.0 adds a visual dimension to the learning process with an interactive board, based on the open-source platform Excalidraw. More than just a drawing tool, this board functions as a **cognitive space extension**, allowing the externalization of reasoning and mental structures—a practice known as **cognitive offloading**.

Inspired by the theory of distributed cognition (Hutchins, 1995), the module enhances the ability to manipulate complex concepts through visual representations, facilitating the understanding of data structures, execution flows, and algorithmic logic.

3.3. Module 3: Integration with the Development Environment (In Planning)

The third module proposes direct integration with real development environments such as Visual Studio Code, via the extension **Continue**. This module seeks to shift pedagogical support to the authentic context of professional practice, overcoming the limitations of approaches based solely on simulated environments or decontextualized instructional resources. This approach aims to provide contextualized pedagogical support, fostering learning within the actual environment of practice, in alignment with the principles of Situated Learning (Lave and Wenger, 1991). According to this theory, knowledge acquired in authentic practice contexts tends to be more transferable and meaningful. At the same time, the integration helps eliminate the so-called split-attention effect (Sweller et al., 2011), which occurs when students must constantly switch between tutorials and the code editor. By centralizing support, practice, and feedback within the same environment, we aim to optimize cognitive processing and promote fluency in problem-solving.

4. Pilot Study

This study adopted a qualitative and exploratory approach to evaluate the pedagogical potential of Worked Examples generated by a subject-matter expert and by CoderBot 2.0. The aim was not to statistically validate the tool's effectiveness, but to initially observe how the system can reproduce the structure of instructional examples and what nuances emerge from this comparison. The methodology was structured in four main stages:

- **Definition of Evaluation Criteria:** Based on Cognitive Load Theory and guidelines for effective design of WEs, a set of four criteria adapted from Renkl (2014b) was adopted: (i) clarity and segmentation; (ii) quality of explanations; (iii) support for rule automation (examples that facilitate pattern recognition); and (iv) support for generative activity (stimulating reflection and active explanation by the student).
- **Example Creation by the Instructor:** A PhD-level instructor with experience in programming education manually created a worked example on the Bubble Sort algorithm using a previously validated instructional template. The example included: problem description, solution steps, correct code, erroneous example, and a reflective exercise aimed at consolidating learning.
- **Example Generation by AI:** The textual input from the "Reflective Exercise" field created by the instructor was used as a *prompt* in CoderBot 2.0 (o3-mini model). The agent automatically generated a complete example with a structure similar to the template used by the instructor: introductory explanation, correct code, diagnostic exercise with alternatives, and justifications.
- **Comparative Analysis:** The two examples were organized side by side in an analytical matrix (Table 1), allowing a thorough comparison of each component (problem, steps, code, error, generative activity, and complementary material). The analysis sought to identify similarities and differences in terms of pedagogical structure, language, clarity, interactivity, and potential student engagement.

Although limited to a single instructor and a specific case (*Bubble Sort*), this exploratory pilot provides initial insights into the alignment between content generated by CoderBot 2.0 and best practices in creating pedagogical examples. It also allowed the observation of relevant qualitative aspects, such as the discursive style, conceptual depth, and level of cognitive support offered by each approach. Thus, the results should be understood as preliminary evidence, guiding subsequent validation stages with expanded samples of students and instructors.

5. Preliminary Results

Table 1 presents the main instructional blocks of the example created by a teacher and the response generated by CoderBot 2.0. This organization facilitates a qualitative comparison between the approaches, highlighting differences and convergences in terms of instructional structure, clarity, and engagement strategies.

Table 1. Instructor's example and the one provided by CoderBot 2.0

Section/Block	Instructor's Template	AI's Response
Problem Description	Implement a bubble sort algorithm to sort the values of a fixed-size integer array of 10 elements.	Explanation of the problem, algorithm context, and the goal of sorting a 10-element array.
Solution Steps	Detailed listing of the steps (numbered) to implement the algorithm, focusing on clarity and segmentation.	Step-by-step explanation with accessible language, bullet points, and emojis for enhanced engagement.
Correct Code	Complete Java code, structured according to best practices, with comments focusing on clarity.	Correct Java code, presented in a block, with explanation of reasoning and suggestions for variation.
Erroneous Example	Java code with an intentional error (lack of an auxiliary variable), followed by an open question for the student.	Java code with a common error, accompanied by response alternatives for diagnosis and reflection.
Generative Activity	Open-ended question: "Can you identify the error? Indicate the line where it occurs."	Multiple-choice quiz: alternatives for the student to identify the error, promoting self-explanation.
Supplementary Material	Recommendation of an illustrative video (bubble sort dance) and additional context.	Suggestion for a video and encouragement to experiment with different inputs.

The comparative analysis between the example created by the instructor and the response generated by CoderBot 2.0 follows a qualitative approach that seeks to identify differences and similarities in the pedagogical dimensions that impact student learning. The central goal is to evaluate how the WEs generated by a subject-matter expert compare to those produced by our agent, in terms of clarity, explanatory quality, support for rule automation, and stimulus for generative activity. The analysis structure is based on key dimensions for the pedagogical effectiveness of worked examples, as proposed by Renkl (2014b). The qualitative comparison was conducted in the following categories:

Regarding **Clarity and Segmentation**, the instructor's example was structured with a clear and sequential organization, facilitating the student's understanding of the algorithm's steps. Information segmentation is crucial to avoid overloading the student's cognitive load. The example generated by the AI, while accessible and well-structured, incorporated engagement elements such as emojis and a more informal style, which may have both positive and negative impacts on clarity depending on the student's profile. In terms of **Quality of Explanation**, the instructor provided a more detailed explanation, focusing on the rules and underlying concepts of the algorithm, to promote a deep understanding. The AI provided a similar explanation but with less emphasis on conceptual connections, instead focusing more on the immediate execution of the code. This approach may be useful for a more superficial or pragmatic understanding of the problem, but may not foster a deep understanding.

Regarding **Support for Rule Automation**, the instructor integrated comments directly into the code, explaining the reasons and principles behind the choices made in the algorithm. This provides a solid foundation for automating rules and internalizing the process. On the other hand, the AI provided a valid explanation, but with less emphasis on the rationale behind the code's steps, which limits support for more robust rule automation. Finally, regarding **Generative Activity**, the instructor employed an open-ended approach, asking the student to identify a specific error in the code, thereby promoting active learning and problem-solving. The AI generated a multiple-choice quiz to test the student's understanding, which may be effective for quick assessments but may not stimulate as much deep reflection and self-explanation as the open task proposed by the instructor.

We observe that while the AI can provide quick and functional worked examples, the pedagogical interaction provided by the instructor offers a richer experience in terms of conceptual explanations and stimulation for deep reflection. While the AI can replicate the basic structure of a worked example, the pedagogical skills and fine-tuning necessary for more effective teaching are still dominated by human presence. This comparison highlights the potential and limitations of using AI in programming education environments. AI can be an effective tool to support teaching, but it does not replace the instructor's ability to adapt instruction to engage students and promote a deep understanding of concepts.

6. Conclusion

This work presented CoderBot 2.0, a significant evolution of a pedagogical agent for teaching programming, now incorporating the potential of LLMs and advanced pedagogical *prompt engineering* techniques. Unlike the previous version, which focused on static examples, CoderBot 2.0 offers a responsive and adaptive approach, capable of generating on-demand correct and erroneous examples, code explanations, and recommendations aligned with best development practices. The new system architecture enables the adaptation of content to meet the needs and proficiency levels of each student, promoting a more personalized, reflective, and continuous learning experience. Grounded in Example-Based Learning (EBL) and Cognitive Load Theory, CoderBot 2.0 explores metacognitive strategies such as *self-explanation*, *fading*, and contextual variation, establishing itself as a dynamic educational environment that goes beyond simple automated tutoring.

The comparative analysis conducted in this exploratory study indicates that the system is capable of producing well-structured and functionally useful instructional artifacts, approaching the quality of examples developed by human experts in some aspects. However, it was observed that the conceptual depth and pedagogical richness remain superior in the examples crafted by instructors, reinforcing the irreplaceable role of the teacher as the curator and configurator of interactions with the AI. Thus, CoderBot 2.0 should be understood as a supportive technology, capable of scaling and personalizing teaching, but still dependent on teacher mediation to ensure pedagogical consistency and depth. As future perspectives, the need to expand empirical validation with students in real learning situations is highlighted, exploring multiple content areas and contexts, as well as comparing different pedagogical strategies (worked examples, Socratic dialogue, gradual feedback) mediated by the system. Additionally, the improvement of the modules still under development aims to consolidate CoderBot 2.0 as a central part of an intelligent learning *hub*, focused on fostering critical, autonomous, and practical training for new programmers.

References

Adams, D. M., McLaren, B. M., Durkin, K., Mayer, R. E., Rittle-Johnson, B., Isotani, S., and Van Velsen, M. (2014). Using erroneous examples to improve mathematics learning with a web-based tutoring system. *Computers in Human Behavior*, 36:401–411.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. (1995). Cognitive tutors: Lessons learned. In *The Journal of the Learning Sciences*, volume 4, pages 167–207.

Atkinson, R. K., Derry, S. J., Renkl, A., and Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research*, 70(2):181–214.

Baddeley, A. (1992). Working memory. *Science*, 255(5044):556–559.

Barros, T., Macedo, L., and Mendes, A. (2021). Automatic generation of worked examples for programming tutors. In *International Conference on Artificial Intelligence in Education*, pages 48–60. Springer.

Beege, M., Schneider, S., Nebel, S., Zimm, J., Windisch, S., and Rey, G. D. (2021). Learning programming from erroneous worked-examples. which type of error is beneficial for learning? *Learning and Instruction*, 75:101497.

Chiarelli, V., Lonati, V., Malacaria, A., Monti, M., and Taverna, A. (2022). A review of worked examples in programming activities. *ACM Transactions on Computing Education (TOCE)*, 23(1):1–28.

Dai, W., Lin, J., Jin, F., and *et al.* (2023). Can generative artificial intelligence outperform self-instructional learning in computer programming? *IEEE Transactions on Learning Technologies*, 16(6):918–929.

Fakour, R., Shahnazari, M., and Alemi, M. (2025). The effectiveness of socratic questioning and concept mapping techniques in developing university students' critical thinking skills. *Teaching in Higher Education*, 30(1):1–20.

Garcia, R. D. S., Villa, J. E. A., Miranda, A. L. M., Guedes, G. T. A., Oran, A. C., De Souza, P. S. S., Vilela, R. F., Valle, P. H. D., and Silva, W. (2025). Theory inspires, but examples engage: A mixed-methods analysis of worked examples from coderbot in programming education. *IEEE Access*.

Hattie, J. and Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1):81–112.

Hundhausen, C. D., Douglas, S. A., and Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290.

Hutchins, E. (1995). *Cognition in the Wild*. MIT Press, Cambridge, MA.

Jury, B., Lorusso, A., Leinonen, J., Denny, P., and Luxton-Reilly, A. (2024). Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian computing education conference*, pages 77–86.

Keuning, H., Jeuring, J., and Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, 19(1):1–43.

Kopp, V., Stark, R., and Fischer, M. R. (2008). Fostering diagnostic knowledge through computer-supported, case-based worked examples: effects of erroneous examples and feedback. *Medical education*, 42(8):823–829.

Lave, J. and Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, UK.

Li, Y., Chen, J., and Wang, X. (2023). Prompt engineering for large language models: A survey.

Mendes, A., Garcia, R., Villa, J., Oran, A., Santana, B. S., Guedes, G. T., Silva, D. G., Valle, P., and Silva, W. (2024). Avaliando a autoeficácia e a aceitação do coderbot em cursos introdutórios de programação: um estudo exploratório. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 3264–3273. SBC.

Pirzado, F. A., Ahmed, A., Mendoza-Urdiales, R. A., and Terashima-Marin, H. (2024). Navigating the pitfalls: Analyzing the behavior of llms as a coding assistant for computer science students-a systematic review of the literature. *IEEE Access*.

Puech, R., Macina, J., Chatain, J., Sachan, M., and Kapur, M. (2025). Towards the pedagogical steering of large language models for tutoring: A case study with modeling productive failure.

Renkl, A. (2014a). Toward an instructionally oriented theory of example-based learning. *Cognitive science*, 38(1):1–37.

Renkl, A. (2014b). The worked examples principle in multimedia learning.

Renkl, A. (2017). Learning from worked-examples in mathematics: Students relate procedures to principles. *ZDM*, 49(4):571–584.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

Sajja, G. S. and Ramesh, S. (2023). Artificial intelligence-enabled intelligent assistant for personalized and adaptive learning in higher education. *arXiv preprint arXiv:2309.10892*.

Sands, P. (2019). Addressing cognitive load in the computer science classroom. In *The Journal of Computing Sciences in Colleges*, volume 34, pages 55–62. Consortium for Computing Sciences in Colleges.

Shaffer, D., Doubé, W., and Tuovinen, J. (2003). Applying cognitive load theory to computer science education. *Annual Workshop of the Psychology of Programming Interest Group*.

Sweller, J., Ayres, P., and Kalyuga, S. (2011). *Cognitive Load Theory*. Springer.

Sychev, O., Anikin, A., Denisov, M., and *et al.* (2021). Improving automated program repair using question asking tutoring strategy. In *International Conference on Artificial Intelligence in Education*, pages 301–307. Springer.

Villa, J. E. A., Garcia, R., Miranda, A. L., Oran, A., Guedes, G. T., Santana, B. S., Silva, D. G., Valle, P., and Silva, W. (2024). Perspectiva dos estudantes sobre um agente pedagógico baseado em exemplos para a aprendizagem de programação: uma análise qualitativa. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 459–473. SBC.

Woolf, B. P. (2009). *Building Intelligent Interactive Tutors: Student-Centered Strategies for Revolutionizing E-Learning*. Morgan Kaufmann.

Zhang, Y., Feng, X., Zhang, J., and *et al.* (2024). Spl: A socratic playground for learning powered by large language models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, pages 1381–1387.