Understanding LLM Responses in Programming Tasks: A Study on Prompt Quality, Personalization, and RAG Limitation

Meilyna Hutajulu[1], Ioka Purba[2], Mulyadi Siahaan[3], Samuel Situmeang[4*], Mario Simaremare[5]

[1, 2, 3, 4, 5] *Information Systems Study Program, Institut Teknologi Del, Toba Regency, Indonesia*

Keywords

Generative AI; Personalized Learning; Automated Prompt-Refinement; Retrieval-Augmented Generation

***Correspondence Email:**
samuel.situmeang@del.ac.id*

Abstract

This position paper evaluates the challenges and opportunities of integrating Generative Artificial Intelligence (GenAI) into programming education, where student understanding varies significantly between senior and junior learners. While senior students tend to possess a stronger comprehension of code structure and logic, junior students often struggle to identify well-structured code or articulate technical problems. Current learning processes remain largely general and are not sufficiently tailored to these individual needs, as existing personalization approaches are often shallow and lack deep integration with a student's specific learning context. The emergence of Large Language Models (LLMs) such as GPT, Claude, and Gemini offers new opportunities to support adaptive education by acting as interactive assistants capable of providing direct feedback. However, their use presents significant risks, including model hallucinations and a heavy dependency on the quality of user-crafted prompts. Furthermore, evidence suggests that Retrieval-Augmented Generation (RAG) has not yet performed optimally in debugging contexts, as external documents are often not effectively utilized by the model's reasoning engine. These systemic issues highlight the urgent need for more effective, contextual, and safe approaches to leveraging GenAI in delivering personalized programming instruction.

1. Introduction

Programming has become one of the core competencies in modern technology and education [1]. In today's digital era, the ability to write and understand code is not only essential for students in computer science related fields but has also become part of broader technological literacy. However, the level of programming comprehension among students varies significantly. Senior or advanced-level students generally possess stronger abilities in understanding code structure, program logic, and syntactic patterns compared to junior or beginner-level students [2].

These differences indicate a clear gap in programming education. Traditional teaching methods that rely on generalized and uniform instructional approaches often fail to accommodate the individual needs of students [3]. As a result, some students struggle to follow the material, while others do not receive sufficient challenge to further develop their skills. This highlights the importance of adopting more personalized and adaptive learning approaches tailored to each student's capabilities. Several studies show that personalization in programming education remains limited to shallow personalization, where learning is adapted only superficially without considering individual interests, capabilities, or specific learning difficulties [4], [5], [6]. Such approaches are not sufficient to help students deeply understand core concepts or navigate through code structures effectively.

Recent advancements in Generative Artificial Intelligence (GenAI), particularly Large Language Models (LLMs) such as GPT, Gemini, and Claude, open promising opportunities for more adaptive and personalized learning support. LLMs can understand user queries, interpret context, and produce relevant responses, making them potential interactive assistants for learning concepts, writing code, and performing debugging tasks. However, these models still face significant challenges [7]. One major limitation is hallucination, where the model generates plausible-sounding but incorrect information [8]. This poses risks for novice learners, who may adopt incorrect concepts or misunderstand code semantics.

Another challenge lies in students' ability to formulate effective prompts. Many students, especially beginners, tend to produce unclear or incomplete prompts, resulting in irrelevant or unhelpful responses [9]. Previous studies also found that the integration of Retrieval-Augmented Generation (RAG) within programming education has not been fully effective, especially for debugging tasks, where external documents provided by the system are often not utilized optimally by the model [10].

These challenges collectively demonstrate that leveraging GenAI for programming education is not straightforward. Despite its potential, significant gaps remain in how LLMs interpret student intent, adapt explanations to different skill levels, and deliver contextually appropriate guidance while minimizing hallucination. Therefore, this study focuses on identifying and analyzing the underlying issues related to the use of GenAI, particularly LLMs, in supporting programming education. The goal is to develop a deeper understanding of how these systems respond to student questions, how contextual alignment can be improved, and how personalized learning experiences can be enhanced for students across varying levels of programming proficiency.

2. Literature Review

2.1 Programming Education and Personalization

Learning programming requires connecting theoretical concepts with practical syntax, a struggle for early-stage students. Proficiency develops through exposure; senior students recognize organized structures better than juniors [2]. However, uniform teaching methods cause early-stage students to fall behind. While personalization aims to adapt pacing and methods, digital education often relies on superficial adjustments (e.g., changing narrative themes) rather than analyzing learning behaviors or problem-solving strategies [6], [11].

2.3. GenAI and Technical Barriers

LLMs can explain concepts and generate code, but they are prone to hallucinations that novices may fail to detect. The effectiveness of these tools relies heavily on the quality of user prompts; ambiguous instructions yield suboptimal results [12].

To mitigate inaccuracy, RAG combines LLMs with external document retrieval. However, research indicates that RAG is inconsistent in programming contexts [8]. In debugging, retrieved documents are often not effectively incorporated into the model's reasoning, leading to minimal quality improvements. Additionally, human factors play a role; students often struggle to craft specific, contextual prompts, creating a barrier to effective adoption [9].

2.4. Retrieval-Augmented Generation (RAG)

To mitigate hallucination and improve accuracy, some studies have applied RAG. RAG combines the generative capability of LLMs with an external retrieval mechanism that fetches relevant documents as grounding sources before the model generates its response [13].

Although RAG theoretically improves factual accuracy, its use in programming education especially for debugging tasks remains challenging. Prior research indicates that during debugging, RAG does not consistently contribute useful external context [14]. Retrieved documents are often not effectively incorporated into the model's reasoning, resulting in minimal improvement in feedback quality. This suggests a need for further analysis of how LLMs interpret and utilize retrieved context when analyzing and correcting source code.

2.5. Barriers in Student-AI Interaction and Prompt Formulation

Beyond technological considerations, human factors play an important role in the effectiveness of GenAI-assisted learning. Many students struggle to craft effective prompts. They often provide instructions that are too general, non-contextual, or insufficiently specific to the problem at hand. Since prompt clarity directly influences the quality of responses, poorly constructed prompts frequently lead to irrelevant or suboptimal answers from the model [9].

This limitation in prompt formulation poses a key obstacle to the adoption of GenAI in programming education. Therefore, it is important to investigate how prompt characteristics influence the quality of responses students receive, and how guidance on prompt construction can improve learning outcomes.

3. Research Method

This paper adopts a qualitative analytical approach to evaluate the efficacy of Large Language Models (LLMs) in the context of computer science pedagogy. The methodology focuses on synthesizing existing literature to identify systemic gaps between AI capabilities and learner needs. By examining the intersection of generative technology and educational psychology, the study establishes a foundation for more adaptive learning environments.

- **Thematic Synthesis:** We analyzed traditional instructional methods to identify "shallow personalization" as a primary obstacle to skill mastery.
- **Comparative Analysis:** We examined interaction patterns, contrasting how advanced learners construct contextual prompts against novices who struggle to articulate technical problems.
- **Technical Evaluation:** We analyzed case studies regarding RAG performance to understand why external documentation is poorly integrated during debugging.
- **Framework Conceptualization:** We developed the "Dynamic Proficiency Modeling" framework. This concept proposes tracking parameters like *historical error frequency* and *syntactic complexity* to autonomously adjust AI response depth.

4. Result and Discussion

The qualitative synthesis of existing literature reveals that the effectiveness of LLM-based systems in programming education is not uniform across all user groups. While these models possess significant potential as interactive assistants, their impact is heavily moderated by the clarity of the user's prompt and their baseline technical proficiency. This variability suggests that simply deploying Generative AI tools in a classroom setting is insufficient if the tools are not specifically designed to account for the diverse pedagogical backgrounds of the students. The value of these assistants depends as much on the student's ability to communicate the problem as it does on the model's underlying computational capabilities.

One of the most prominent obstacles identified in this study is the "prompting gap" that separates novice and senior learners. Junior students often lack the foundational technical vocabulary required to articulate their logic errors, resulting in vague or underspecified prompts that yield generic AI responses. In contrast, advanced students leverage their prior knowledge to provide high-level contextual detail, enabling the model to generate technically accurate and relevant feedback. Addressing this challenge requires moving beyond static interactions and integrating automated prompt-refinement mechanisms that can bridge the gap between user intent and model output.

Furthermore, the technical constraints of current implementations, such as hallucination and the inefficiency of Retrieval-Augmented Generation (RAG), present substantial risks to learning safety. Novice students may fail to detect subtle syntax or logic errors in AI-generated code, which risks reinforcing fundamental conceptual misunderstandings. Even when RAG provides external technical documentation as grounding, models often fail to integrate this context into their internal reasoning during complex debugging tasks. This failure is particularly evident when the model's internal logic for common syntax errors overrides the provided external context, leading to inaccurate or irrelevant feedback.

To overcome these limitations, the "Dynamic Proficiency Modeling" framework is presented here as the essential solution for truly adaptive learning systems. This approach proposes that AI assistants should autonomously adjust the complexity and technical depth of their responses based on a continuous analysis of a student's **historical error frequency** and **syntactic complexity**. By tracking how often a student repeats specific logic errors and measuring their command of advanced language constructs, the system can calibrate its guidance to the learner's actual capabilities. Integrating such dynamic modeling with robust verification mechanisms is necessary to transform GenAI from a simple code generator into a reliable pedagogical partner that ensures the provided guidance is contextually appropriate and pedagogically sound.

5. Conclusions

This position paper has evaluated the systemic challenges and opportunities of integrating Large Language Models (LLMs) into programming education, concluding that the efficacy of Generative AI is deeply contingent upon the learner's existing proficiency and the model's contextual alignment. The thematic analysis confirms that a one-size-fits-all model fails to address the unique needs of diverse student cohorts, particularly due to the "prompting gap" where beginners struggle to articulate technical problems. This lack of clarity often results in generic feedback that hinders rather than helps the learning process, leaving novice learners vulnerable to reinforcing conceptual misunderstandings through unverified AI outputs.

The technical evaluation of current implementations reveals that existing safeguards, such as Retrieval-Augmented Generation (RAG), are currently insufficient for complex tasks like debugging. We have identified that models often fail to integrate retrieved external context into their internal reasoning, allowing predefined syntactic patterns to override accurate technical documentation. To be effective, future systems must bridge this gap by ensuring that retrieved information is meaningfully synthesized during the model's inference process to prevent the generation of plausible but incorrect code.

To address these limitations, this paper proposes a transition toward **Dynamic Proficiency Modeling** as a core architectural requirement for educational AI. By autonomously adjusting response complexity based on a student's **historical error frequency** and **syntactic complexity**, AI assistants can provide truly tailored support that scales with the learner's development. Coupled with automated prompt-refinement mechanisms, these advancements will enable a safer and more individualized experience that empowers students across the full spectrum of proficiency. Moving forward, the integration of these adaptive frameworks will be essential in ensuring that AI tools serve as effective pedagogical partners rather than merely acting as black-box code generators.

6. References

[1] N. Kiesler and C. Thorbrügge, "A Comparative Study of Programming Competencies in Vocational Training and Higher Education," in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, New York, NY, USA: ACM, Jul. 2022, pp. 214–220. doi: 10.1145/3502718.3524818.

[2] S. Nurollahian, A. N. Rafferty, N. Brown, and E. Wiese, “Growth in Knowledge of Programming Patterns: A Comparison Study of CS1 vs. CS2 Students,” in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, New York, NY, USA: ACM, Mar. 2024, pp. 979–985. doi: 10.1145/3626252.3630865.

[3] C. Li and J. Long, “Comparing AI-Assisted and Traditional Teaching in College English: Pedagogical Benefits and Learning Behaviors,” *Information*, vol. 16, no. 10, p. 895, Oct. 2025, doi: 10.3390/info16100895.

[4] A. Rojas-López and F. J. García-Peñalvo, “Personalized Education for a Programming Course in Higher Education,” in *Research Anthology on Computational Thinking, Programming, and Robotics in the Classroom*, IGI Global, 2022, pp. 344–367. doi: 10.4018/978-1-6684-2411-7.ch017.

[5] C.-H. Tseng, H.-C. K. Lin, A. C.-W. Huang, and J.-R. Lin, “Personalized Programming Education: Using Machine Learning to Boost Learning Performance Based on Students’ Personality Traits,” *Cogent Education*, vol. 10, no. 2, Dec. 2023, doi: 10.1080/2331186X.2023.2245637.

[6] E. Logacheva, A. Hellas, J. Prather, S. Sarsa, and J. Leinonen, “Evaluating Contextually Personalized Programming Exercises Created with Generative AI,” in *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, New York, NY, USA: ACM, Aug. 2024, pp. 95–113. doi: 10.1145/3632620.3671103.

[7] N. Annuš, “Investigation of Generative AI Adoption in IT-Focused Vocational Secondary School Programming Education,” *Education Sciences*, vol. 15, no. 9, p. 1152, Sep. 2025, doi: 10.3390/educsci15091152.

[8] E. Berberette, J. Hutchins, and A. Sadovnik, “Redefining ‘Hallucination’ in LLMs: Towards a Psychology-Informed Framework for Mitigating Misinformation,” Feb. 01, 2024. [Online]. Available: http://arxiv.org/abs/2402.01769

[9] M. Simaremare, C. Pardede, I. Tampubolon, D. Simangunsong, and P. Manurung, “Pair Programming in Programming Courses in the Era of Generative AI : Students ’ Perspective,” *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*, pp. 507–511, 2024, doi: 10.1109/APSEC65559.2024.00069.

[10] C. C. Y. Yang, G.-J. Liu, and C.-J. Yu, “Optimizing RAG in Programming Education: A Comparative Study of Models and Strategies,” *IEEE Access*, vol. 13, pp. 190890–190903, 2025, doi: 10.1109/ACCESS.2025.3629843.

[11] H. Fake and N. Dabbagh, *Designing Personalized Learning Experiences*. 2023. doi: 10.4324/9781003121008.

[12] S. R. Solanki and D. K. Khublani, *Generative Artificial Intelligence: Exploring the Power and Potential of Generative AI*. Apress, 2024.

[13] A. Gheorghiu, *Building Data-Driven Applications with LlamaIndex: A Practical Guide to Retrieval-Augmented Generation (RAG) to Enhance LLM Applications*. Packt Publishing, 2024.

[14] P. Feldman, J. R. Foulds, and S. Pan, “RAGged Edges: The Double-Edged Sword of Retrieval-Augmented Chatbots,” Jun. 12, 2024. [Online]. Available: http://arxiv.org/abs/2403.01193