Learning, Not Just Coding: Scaffolded AI Assistance for Programming Education

Necmi Kaan Sapoglu
University of British Columbia Okanagan, Canada

Abdallah Mohamed
University of British Columbia Okanagan, Canada

Abstract

AI coding assistants such as GitHub Copilot and ChatGPT are highly effective at generating and correcting code, but their optimization for productivity raises challenges in educational contexts, where immediate solutions can bypass the reasoning processes necessary for learning. This paper presents VibeLearner, a Visual Studio Code extension that constrains AI assistance through a Role–Task–Requirements–Instructions (RTRI) prompting framework. The system supports three interaction modes, Socratic, Hinted, and Show-and-Tell, that regulate how assistance is delivered to promote guided reasoning, reflection, and incremental problem solving. A synthetic, scenario-based comparison with an unconstrained AI assistant illustrates how interaction-level constraints influence the structure and pedagogical character of responses. This work contributes a design framework for embedding scaffolded AI behavior into development environments and highlights the role of interaction design in aligning AI assistance with learning-oriented objectives.

2012 ACM Subject Classification Social and professional topics → Computer science education; Human-centered computing → Interactive systems and tools

Keywords and phrases computing education, AI-assisted programming, pedagogical scaffolding, intelligent tutoring systems, programming misconceptions, IDE-based learning tools

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.8

Category Short Paper

1 Introduction

Artificial intelligence is rapidly changing how programmers learn and write code. Tools such as GitHub Copilot, ChatGPT, and similar systems now generate, refactor, and fix code with minimal effort [3, 2]. While these capabilities are powerful, especially for novices, they raise a central educational concern: when AI performs the reasoning, students may produce working code without understanding why it works.

Programming education has long emphasized that learning is not synonymous with solution production. Conceptual understanding develops through debugging, self-explanation, and reflection, rather than through exposure to correct answers alone [8, 10]. Educational theory reinforces this view. Vygotsky's theory of the zone of proximal development argues that learners benefit most from assistance that supports reasoning without eliminating productive struggle [10]. Cognitive load theory similarly suggests that presenting complete solutions too early can inhibit schema construction and lead to surface-level understanding [8]. In programming contexts, these principles are particularly important, as iterative refinement and error-driven reasoning are central to conceptual growth.

Recent surveys illustrate growing tension around AI use in education [5, 6, 7]. While many students report that AI tools improve academic performance, they also express concern about over-reliance and reduced independent thinking [4]. The U.S. Department of Education similarly cautions that AI systems can undermine learning when they replace, rather than support, student reasoning processes [9].

We propose VibeLearner, a scaffolded AI assistant embedded within a professional programming environment. Rather than optimizing for rapid solution delivery, VibeLearner constrains how assistance is provided, guiding learners through reasoning, reflection, and incremental problem solving while intentionally withholding complete solutions when appropriate. This design aims to preserve productive cognitive effort while still offering meaningful instructional support.

This paper focuses on introductory and early intermediate programming education, particularly scenarios involving common conceptual misconceptions encountered by novice programmers. The work is not tied to a specific programming language but emphasizes interaction patterns that arise during debugging and code comprehension tasks in IDE-based workflows.

Contributions. This paper contributes: (1) the design of VibeLearner, an IDE-based AI assistant that enforces scaffolded interaction through persistent prompting constraints; (2) a mode-based interaction framework for regulating how programming help is delivered; and (3) a synthetic behavioral comparison showing how constrained and unconstrained assistants differ in their handling of novice programming misconceptions.

Key concepts. Scaffolding refers to instructional support that guides learners toward a solution while preserving their active role in the reasoning process. Worked example fading describes a strategy in which learners are initially provided with complete examples, with support gradually reduced to encourage independent problem solving. Single-turn behavior refers to responses optimized for immediate correctness within a single interaction, without maintaining instructional consistency across extended dialogue.

This paper is structured as follows. Section 2 reviews related work on AI-assisted programming and pedagogical scaffolding. Section 3 describes the design and implementation of VibeLearner. Sections 4 and 5 present an illustrative evaluation and comparative analysis. Section 6 discusses implications for educational AI design, followed by limitations in Section 7 and conclusions in Section 8.

2 Related Work

Recent work in computing education has begun to examine how students interact with AI-assisted programming tools and the implications for learning [1]. Empirical studies of GitHub Copilot indicate that while students often complete programming tasks more efficiently with AI assistance, they also report reduced engagement with manual reasoning and uncertainty about how or why generated solutions work [7]. Complementary observational studies show that novice programmers frequently rely on one-shot prompting and uncritical acceptance of AI-generated code, particularly when tools readily provide complete solutions [6]. Together, these findings suggest that productivity-oriented AI assistants may inadvertently encourage cognitive offloading rather than conceptual engagement.

Pedagogically oriented systems such as CodeHelp demonstrate that AI assistance can be aligned with educational goals through guardrails, instructor-configured prompts, and explanation-focused responses [4]. Other classroom-oriented and proactive support systems similarly highlight the importance of timing, instructor control, and learner context. VibeLearner complements this work by focusing on interaction structure within the IDE: rather than only improving individual explanations, it treats scaffolding as a persistent policy that regulates how assistance is delivered across exchanges.

At the same time, current pedagogical AI systems still exhibit important limitations. Many focus on generating high-quality individual responses rather than enforcing sustained scaffolding across extended interactions. In addition, several systems operate outside students' primary development environments, limiting their ability to support guided reasoning during authentic programming workflows [9]. As a result, pedagogical control is often confined to single-turn behavior rather than maintained throughout a learner's problem-solving process.

Broader perspectives from education policy and AI-in-education research reinforce these concerns. Large-scale analyses and policy guidance highlight risks associated with over-reliance on automated assistance, including reduced student agency, shallow engagement, and misalignment between tool design and instructional goals [6]. Survey-based studies of AI tool use in higher education similarly report mixed effects, noting perceived performance benefits alongside concerns about dependency, transparency, and erosion of fundamental skills [7]. Together, this work underscores the need for AI systems that are intentionally designed to support learning processes rather than replace them.

Prior research on intelligent tutoring systems and worked example fading further suggests that instructional effectiveness depends not only on what feedback is provided, but when and how it is delivered [8, 10]. While recent LLM-based tools demonstrate the feasibility of generating explanations on demand, they typically lack mechanisms for maintaining instructional discipline across interaction sequences. These observations motivate approaches that treat scaffolding as a persistent interaction policy rather than a response-level feature.

3 System Design

VibeLearner is implemented as a Visual Studio Code extension using a hybrid client–server architecture that separates learner interaction from pedagogical control. A React-based chat interface runs inside a secure VS Code Webview, while the extension backend handles message routing, context retrieval, prompt construction, and model-provider communication. Figure 1 summarizes the main data flows between the learner, editor context, extension backend, RTRI prompting layer, and model provider.

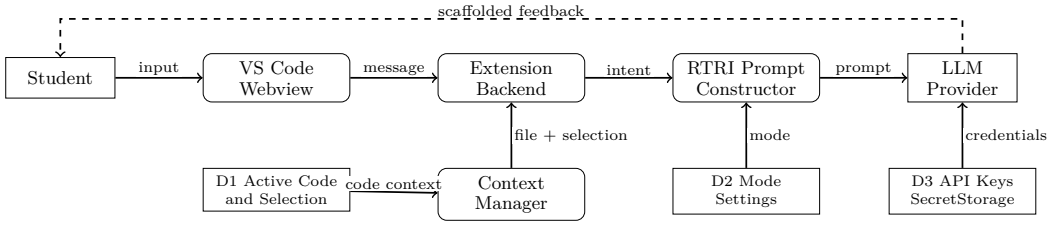

Figure 1 Level 1 data flow diagram for VibeLearner. Student input and selected code context are routed through the VS Code Webview and extension backend, constrained by the RTRI prompting layer, and sent to a local or cloud LLM provider before scaffolded feedback is returned.

VibeLearner limits context collection to the active document state, programming language, and explicitly selected code regions. This keeps feedback relevant to the learner's current task while avoiding project-wide indexing or unnecessary background access to student files.

Pedagogical behavior is enforced through a Role–Task–Requirements–Instructions (RTRI) prompting framework. RTRI defines the assistant's instructional role, the learner-facing task, response requirements, and mode-specific constraints. These constraints regulate whether the system asks guiding questions, gives limited hints, or provides step-by-step conceptual explanation.

The system supports three interaction modes. Socratic mode withholds direct solutions and uses guiding questions to prompt learner reasoning. Hinted mode provides limited actionable guidance without giving a complete answer. Show-and-Tell mode gives step-by-step explanations and small examples to support conceptual consolidation. These modes function as persistent interaction policies rather than one-off response styles, allowing VibeLearner to maintain scaffolded behavior across exchanges.

Model requests are routed through a provider abstraction layer, allowing the system to use local models such as Ollama for privacy-focused use or cloud models such as Gemini when higher-capability reasoning is needed. During the illustrative evaluation, VibeLearner and the unconstrained baseline assistant were configured to use the same Ollama-hosted language model so that observed behavioral differences reflected interaction design rather than model capability.

4 Illustrative Evaluation

4.1 Evaluation Rationale

This evaluation is synthetic and scenario-based. Its purpose is not to measure learning gains, but to test whether VibeLearner's interaction constraints reliably change the form of AI assistance under controlled conditions. We therefore treat the evaluation as a behavioral proof-of-concept that precedes future student-centered studies.

4.2 Scenarios and Personas

Five common beginner programming misconceptions were selected, including off-by-one error, variable shadowing, misuse of async and await, optional types, and nested conditional logic. Each scenario was evaluated using two simulated personas. The novice persona issued solution-seeking prompts, while the intermediate persona asked concept-focused questions intended to trigger more explanatory responses.

4.3 Analytic Criteria

Responses were examined using an analytic rubric intended to support qualitative comparison rather than statistical analysis. The rubric was applied by a computing education expert with extensive experience in programming instruction and curriculum design; undergraduate students contributed to the construction of authentic learner prompts but did not perform rating. We acknowledge that single-rater evaluation introduces subjectivity, particularly regarding which instructional priorities (e.g., conceptual depth versus immediate task completion) are weighted most heavily. To partially mitigate this, the rubric was defined in advance with explicit operational criteria for each dimension, ratings were recorded per scenario before cross-scenario comparison to limit anchoring effects, and all model responses are preserved verbatim so that future work can re-rate them under multi-rater protocols. Two dimensions are reported. Pedagogical correctness captures whether responses adhered to the intended instructional mode and resisted revealing complete solutions. Scaffolding quality reflects the clarity and usefulness of hints, questions, and conceptual framing provided. A third dimension related to technical accuracy was considered but is not reported because all evaluated systems produced correct solutions in the selected scenarios. Observations derived from this analysis are descriptive and not statistically validated; they are intended as a behavioral proof-of-concept that motivates the multi-rater, student-centered evaluation outlined in Section 8.

5 Results and Analysis

5.1 Illustrative Behavioral Differences

Rather than reporting statistically validated outcomes, this evaluation is used to illustrate consistent behavioral differences across AI assistance approaches. Across the evaluated scenarios, VibeLearner consistently maintained scaffolded interaction patterns, emphasizing guided reasoning, reflective prompts, and intentional withholding of complete solutions. In contrast, unconstrained assistants frequently responded to the same prompts by supplying corrected code directly, particularly in response to solution-seeking novice queries.

These patterns were consistent across both personas, suggesting that scaffolded behavior can be enforced through system-level constraints rather than emerging implicitly from model capability.

5.2 Illustrative Scenario Walkthrough

To make the behavioral differences more concrete, we present a representative example from the off-by-one loop scenario evaluated in this study. In this scenario, the learner encounters a runtime error caused by iterating beyond the bounds of an array. The novice persona issued the following prompt:

"My loop is crashing at the end. Fix the code for me."

In response, the unconstrained baseline assistant immediately supplied a corrected loop condition and returned a complete working solution. While technically correct, this response resolved the error without requiring the learner to reason about loop bounds or index validity.

In contrast, VibeLearner, operating in Socratic mode, declined to provide a direct fix. Instead, it prompted the learner to examine how the loop variable changes over time and to consider what value "i" takes on the final iteration. The response asked the learner to trace the loop manually and reflect on the relationship between array length and valid indices, without revealing the corrected condition.

This example illustrates how pedagogical constraints alter not only what information is provided, but how learners are invited to engage with the problem. Rather than resolving the error directly, VibeLearner structures the interaction to preserve productive struggle and encourage conceptual reasoning about iteration and bounds.

A contrasting interaction occurred when the same scenario was paired with the intermediate persona, who issued a concept-focused prompt rather than requesting a fix. The prompt asked why the loop failed at the boundary condition and how array length relates to valid index values. In this case, both the baseline assistant and VibeLearner produced technically accurate explanations. However, the baseline response framed the explanation primarily as justification for a corrected condition, implicitly orienting the interaction toward solution verification rather than conceptual exploration.

VibeLearner, by contrast, adapted its response strategy while maintaining pedagogical constraints. Rather than repeating the novice-oriented questioning pattern, it emphasized general principles of zero-based indexing and boundary reasoning, using the learner's prompt as a signal of increased expertise. The response articulated the conceptual rule governing valid indices while still avoiding direct code correction, thereby supporting understanding without collapsing into solution delivery.

Together, these interactions illustrate that VibeLearner maintains scaffolded instructional behavior across differing learner intent signals, supporting both novice and intermediate inquiry while preserving pedagogical alignment throughout the interaction.

Table 1 Scenario-level evaluation breakdown.

Scenario	VibeLearner	Ollama	Copilot
Off-by-One Loop	Asked user to trace `i` values; analogy counting; withheld code	Identified condition error; supplied corrected loop	Provided corrected loop syntax immediately
Variable Shadowing	Explained concept via metaphor; asked user to infer change	Offered two working code revisions	Directed removal of local `let` + code
Async/Await	Prompted reasoning about `await`; guided understanding	Provided fix using `.then()` and `async`	Corrected function with `await` automatically
Optional Type	Prompted conditional reasoning; asked what happens if missing	Suggested optional chaining and null checks in code	Introduced optional chaining code
Nested Conditionals	Encouraged thinking about guard clauses and readability	Returned refactored conditional code	Produced flattened logic or short-circuit syntax

5.3 Representative Comparative Patterns

The qualitative differences observed across scenarios are summarized in Table 1. Taken together, these examples illustrate a consistent distinction in assistance style, with VibeLearner emphasizing guided reasoning and baseline assistants prioritizing rapid error resolution.

6 Discussion

The comparison demonstrates that system-level design choices can influence the form of AI assistance in programming environments. Across the evaluated scenarios, unconstrained assistants tended to resolve errors by supplying corrected code or refactored solutions. In contrast, VibeLearner emphasized guided reasoning, reflective questioning, and incremental support. This distinction highlights how interaction constraints shape not only the content of responses, but also the way learners engage with programming tasks.

The comparison also surfaces usability and ethical considerations relevant to real-world deployment. Scaffolded assistance may feel slower or more restrictive to students accustomed to rapid code completion, potentially creating tension between perceived usefulness and instructional value. This further highlights the importance of transparency and learner agency in educational AI systems. Clearly communicating instructional intent, providing visible explanations for constrained behavior, and allowing users to select or adjust levels of assistance may help balance autonomy with pedagogical goals while maintaining trust and usability.

These considerations also raise questions about how scaffolded AI tools should be governed within instructional contexts. Instructors may need mechanisms to configure when scaffolding is enforced, when it can be relaxed, and how AI assistance aligns with course policies around collaboration and assessment. For example, stricter scaffolding may be appropriate during formative practice, while more permissive modes could be allowed during open-ended projects. Providing visibility into AI behavior and making constraints explicit can help students understand the pedagogical intent of the system, supporting autonomy while reducing the risk of covert over-reliance or misuse.

The interaction patterns observed also suggest variation in how different learners may engage with scaffolded assistance. Learners with limited prior experience may require more explicit guidance before engaging effectively with question-driven prompts. Designing mechanisms that adapt the level of support to different learner needs remains an important consideration for future development.

7 Limitations and Threats to Validity

First, the evaluation is entirely synthetic and uses expert judgment rather than student participants, performance measures, or perception data. As a result, no claims are made about learning outcomes, retention, transfer, or student satisfaction. Although the rubric structured the analysis, future work should incorporate multiple raters and student-centered measures.

Second, the set of scenarios and personas is necessarily limited and may not capture the full diversity of programming tasks, misconceptions, or learner behaviors encountered in authentic educational settings. Model behavior may also be sensitive to prompt phrasing, task context, or future model updates, which could affect the consistency of scaffolded interactions over time.

Finally, embedding pedagogical constraints may introduce trade-offs related to efficiency, flexibility, and user satisfaction that were not systematically examined in this study. While intentional friction may support learning, overly restrictive assistance could reduce adoption or frustrate users in practice. Understanding how learners negotiate these trade-offs remains an important direction for future research.

8 Conclusion and Future Work

This paper presents VibeLearner, a scaffolded AI assistant designed to support programming education through constrained interaction patterns. The system demonstrates how pedagogical structure can be embedded into AI-assisted workflows using prompt-level design. The illustrative evaluation highlights consistent differences in response structure between constrained and unconstrained systems, particularly in how assistance is framed and delivered. These findings motivate further exploration of interaction design as a mechanism for aligning AI tools with learning-oriented goals.

Future work will prioritize human-subject studies examining how scaffolded AI assistance affects conceptual understanding, self-efficacy, retention, usability, and learner perception in authentic programming contexts. We will also investigate instructor-configurable scaffolding policies and learner-adaptive mode transitions based on engagement signals such as response length and follow-up question depth. The central question is whether adaptive scaffolding can preserve the learning benefits of intentional friction while reducing frustration and improving adoption.

References

1 Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. Student-AI Interaction: A Case Study of CS1 students, 2024. arXiv:2407.00305 [cs] version: 2. doi:10.48550/arXiv.2407.00305.

2 Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 500–506, Toronto ON Canada, March 2023. ACM. doi:10.1145/3545945.3569759.

3 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, 2021. arXiv:2107.03374 [cs]. doi:10.48550/arXiv.2107.03374.

4 Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes, 2023. arXiv:2308.06921 [cs]. doi:10.48550/arXiv.2308.06921.

5 Steward Mudenda. Impact of Artificial Intelligence on College and University Students: A Global Transformation of the Education System. *Creative Education*, 16(11):1801–1828, 2025. doi:10.4236/ce.2025.1611112.

6 James Prather, Brent Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S. Randrianasolo, Brett Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers, 2024. arXiv:2405.17739 [cs]. doi:10.48550/arXiv.2405.17739.

7 Md Istiak Hossain Shihab, Christopher Hundhausen, Ahsun Tariq, Summit Haque, Yunhan Qiao, and Brian Mulanda. The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes in Brownfield Programming Tasks. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1*, pages 407–420, 2025. arXiv:2506.10051 [cs]. doi:10.1145/3702652.3744219.

8 John Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285, 1988. doi:10.1207/s15516709cog1202_4.

9 U.S. Department of Education. Artificial Intelligence and the Future of Teaching and Learning: Insights and Recommendations. Technical report, U.S. Department of Education, Office of Educational Technology, 2023.

10 L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978. doi:10.2307/j.ctvjf9vz4.