How Does LLM-powered Coding Assistance Shape Incidental Learning? Exploring Cognitive Forcing Strategies in Programming Education

Ba-Thinh Tran-Le, Patrick Thomas, Nicholas M Stiffler, Thuy Ngoc Nguyen*

Department of Computer Science, University of Dayton, OH, 45469, USA
tranl5@udayton.edu, pthomas1@udayton.edu, nstiffler1@udayton.edu, ngoc.nguyen@udayton.edu

Abstract

Many AI-based code assistants, particularly those powered by Large Language Models (LLMs), provide complete solutions, which can reduce active problem solving and limit incidental learning, the acquisition of knowledge as a byproduct of task engagement. Such learning requires active participation rather than passive acceptance of AI-generated answers, which answers might be incorrect. This study examines how incidental learning can be supported through guided interaction. We present LeetCoach, an LLM-assisted coding platform that applies a cognitive forcing strategy, prompting learners to reflect and take incremental steps instead of receiving full solutions. Using LeetCode-style questions, we conducted a pilot study with novice and advanced college programmers who completed tasks under assisted and unassisted conditions. Novices showed substantial post-test gains despite receiving AI guidance only during the intervention, suggesting that incidental exposure improved later performance. Advanced learners showed smaller gains. Across both groups, participants required fewer debugging attempts in the post-test compared to earlier stages, indicating improved debugging efficiency and algorithmic understanding. These findings provide early evidence that LLMs can be designed to promote indirect learning while shaping problem-solving strategies. This work offers a proof of concept for cognitively informed tutoring systems in computer science education and discusses implications for integrating LLMs to enhance both immediate outcomes and lasting skill development.

Introduction

Advances in Large Language Models (LLMs) have led to accessible assistance tools that can generate code, provide interactive learning support, and enable new approaches to programming instruction (Qi, Hartmann, and Norouzi 2023). Prominent examples such as ChatGPT (Kashefi and Mukerji 2023; Tian et al. 2023; Biswas 2023) and GitHub Copilot (Nguyen and Nadi 2022; Puryear and Sprint 2022; Wermelinger 2023; Dakhel et al. 2023) have demonstrated the ability to produce application-specific code, improving coding efficiency. Nevertheless, previous research has also identified potential pitfalls of integrating LLMs into educational settings (Cambaz and Zhang 2024), including evidence that students who rely more heavily on LLMs for cognitively demanding tasks (e.g., code generation, debugging) tend to achieve lower final course grades (Jošt, Taneski, and Karakatič 2024). There are also concerns that learners' overreliance may lead to skill degradation (Yilmaz and Yilmaz 2023; Zastudil et al. 2023) and superficial engagement. Given growing concerns about the reliability and trustworthiness of AI, fostering human critical engagement is a necessary component of successful human–AI interaction in educational contexts (Kazemitabaar et al. 2025). Recent work has investigated how interactions with AI-powered decision support systems can be redesigned using approaches such as in-the-moment cognitive interventions (Gajos and Mamykina 2022; Park et al. 2019) to encourage deeper processing of AI-generated information.

In learning sciences, cognitive engagement refers to the extent to which learners actively participate in the learning process, with higher engagement linked to greater instructional benefits and improved skill acquisition (Gajos and Mamykina 2022). In programming education, this concept has informed the design of activities that strengthen prompting skills and code comprehension (Denny et al. 2024a,b), as well as the development of coding assistants that avoid providing direct solutions (Liffiton et al. 2023; Kazemitabaar et al. 2024). However, most existing studies in programming education focus on the capabilities or accuracy of AI assistants rather than on how interaction design can influence learners' incidental learning in problem-solving strategies. We argue that instructional design should focus on fostering deeper engagement with AI-generated code to mitigate skill degradation. Using in-the-moment cognitive interventions (Park et al. 2019), cognitive forcing (Buçinca, Malaya, and Gajos 2021), and metacognitive reflection (Tankelevitch et al. 2024), we adopt an approach that integrates cognitive forcing, goal-oriented prompts, and scaffolded guidance for conceptual understanding and debugging. This approach is designed to help learners understand problems, reflect on their reasoning, and engage in iterative refinement to critically analyze and adapt AI-generated code rather than passively relying on complete solutions.

Most prior research on AI code assistants in the context of programming education has relied on closed-source commercial services, such as OpenAI's GPT-4, to provide AI-assisted learning experiences (Kazemitabaar et al. 2025;

*Corresponding author: ngoc.nguyen@udayton.edu

Biswas 2023). While effective, these systems can be costly and inaccessible to many AI practitioners and researchers. The potential of open-source, lightweight, and cost-effective LLMs, which provide transparency and public access, remains largely underexplored in computer science education. Moreover, these models can be configured through prompt engineering to deliver learner-specific guidance tailored to individual needs (Logan IV et al. 2021; Zhao et al. 2023). However, limited work has examined how such models can be integrated with pedagogy-driven interaction designs that align with the incremental nature of problem solving, where understanding fundamental concepts and developing algorithmic thinking are essential.

This paper addresses this gap by investigating how incidental learning can be supported through guided interaction that incorporates a cognitive forcing strategy into open-source LLM-based code assistance. We present *LeetCoach*, an open-source LLM-powered platform to provide immediate task-oriented assistance that is lightweight and easily adaptable to educational settings. Rather than providing complete solutions, *LeetCoach* uses step-by-step, goal-oriented prompts to encourage learners to reflect on problems, approach debugging conceptually, and iteratively test and refine their code.

We conducted a pilot within-subjects study with 19 students, classified as novice or advanced, using LeetCode-style data structure problems. Each participant completed three tasks: a pre-intervention task without AI assistance, an AI-assisted intervention task, and a post-intervention task without assistance. This design allowed us to evaluate both the immediate impact of AI guidance during the intervention and learning gains in subsequent problem-solving. Results showed that novices achieved substantial post-intervention improvements despite receiving AI guidance only during the intervention, suggesting strong incidental learning effects in terms of algorithmic understanding and debugging skills. Advanced learners exhibited smaller gains, indicating that benefits may be greater for less experienced programmers. Post-study surveys provided insight into learners' perceptions of guidance and engagement.

In summary, we make the following contributions:

- We present *LeetCoach*, an open-source, lightweight LLM-based coding assistant that integrates a cognitive forcing strategy with goal-oriented prompts and scaffolded guidance to support incidental learning through active problem-solving without providing full solutions.
- We further demonstrate, through a within-subjects study with novice and advanced programmers, that such guided AI interaction can lead to incidental gains in debugging skills and algorithmic understanding, with stronger effects observed for novices.

Related Work

LLM-Powered Programming Assistance in Education. Recent studies show that human-aligned LLM-based coding assistants can enhance students' debugging efficiency and conceptual accuracy (Wong and Tan 2024), and that LLM-generated test cases can improve understanding of algorithmic reasoning (Kumar and Lan 2024). While these assistants can generate, explain, and debug code, they present two key challenges for programming education. First, their outputs are inconsistently accurate for complex coding problems (Kazemitabaar et al. 2023, 2025; Prather et al. 2024). These inaccuraries make overreliance on AI solutions risky for novice programmers who must learn the syntactical structures of programming languages and the algorithmic reasoning of programming problems (Gajos and Mamykina 2022; Güner and Er 2025; Kazemitabaar et al. 2023; Miedema, Aivaloglou, and Fletcher 2022). Such dependence can impede novice programmers' development of critical programming skills, namely decomposition, indepedent debugging, transferrable problem-solving, and algorithmic logic (Kazemitabaar et al. 2025; Qureshi 2023; Zhai, Wibowo, and Li 2024). Second, most studies on LLM-based assistants rely on commercial APIs such as OpenAI (Biswas 2023; Hammer et al. 2024). Proprietary LLMs are costly, less adaptable for classroom use, and typically limit research to a single popular platform: ChatGPT (Hammer et al. 2024; Shen et al. 2024). Open-source LLMs, however, enable prompt-level control, making it more feasible to embed learning-centered scaffolds directly into AI interactions.

Incidental Learning in AI-Assisted Systems. Incidental learning, a by-product of engaging in tasks like problem solving or advice seeking rather than explicit instruction, plays an important role in the development of transferable understanding (Watkins and Marsick 1992). In programming education, AI systems can facilitate incidental learning by provoking students to make in-the-moment decisions, test hypotheses, and engage with system feedback to iteratively guide problem-solving. Previous work has shown that incidental learning depends on the level of cognitive participation and can arise from collaborative decision-making tasks (Berlin and Jeffries 1992; Marsick et al. 2017). More recent research indicates that AI assistance, such as providing recommendations and explanations, improves immediate task accuracy but rarely results in learning (Gajos and Mamykina 2022; Kazemitabaar et al. 2023). In contrast, "explanation-only" conditions, in which learners receive an AI-generated explanation without any accompanying recommendation and must independently derive their own conclusions, have been shown to produce significant learning gains (Gajos and Mamykina 2022; Kazemitabaar et al. 2025). Similarly, research shows that slowing down the AI assistance process to require users to make a decision first can further improve learning outcomes (Park et al. 2019; Green and Chen 2019). Building on these insights, our work investigates how AI-assisted programming environments can incorporate explanation-only and reasoning-first interaction patterns to operationalize incidental learning principles in problem-solving tasks.

Cognitive Forcing Strategies in AI-assisted Learning. Cognitive Forcing Functions (CFFs) can reduce overreliance on AI output by prompting users to slow down, consider alternatives, predict next steps, or commit to a choice before seeing AI output. While CFFs increase learners' cognitive load (Buçinca, Malaya, and Gajos 2021),

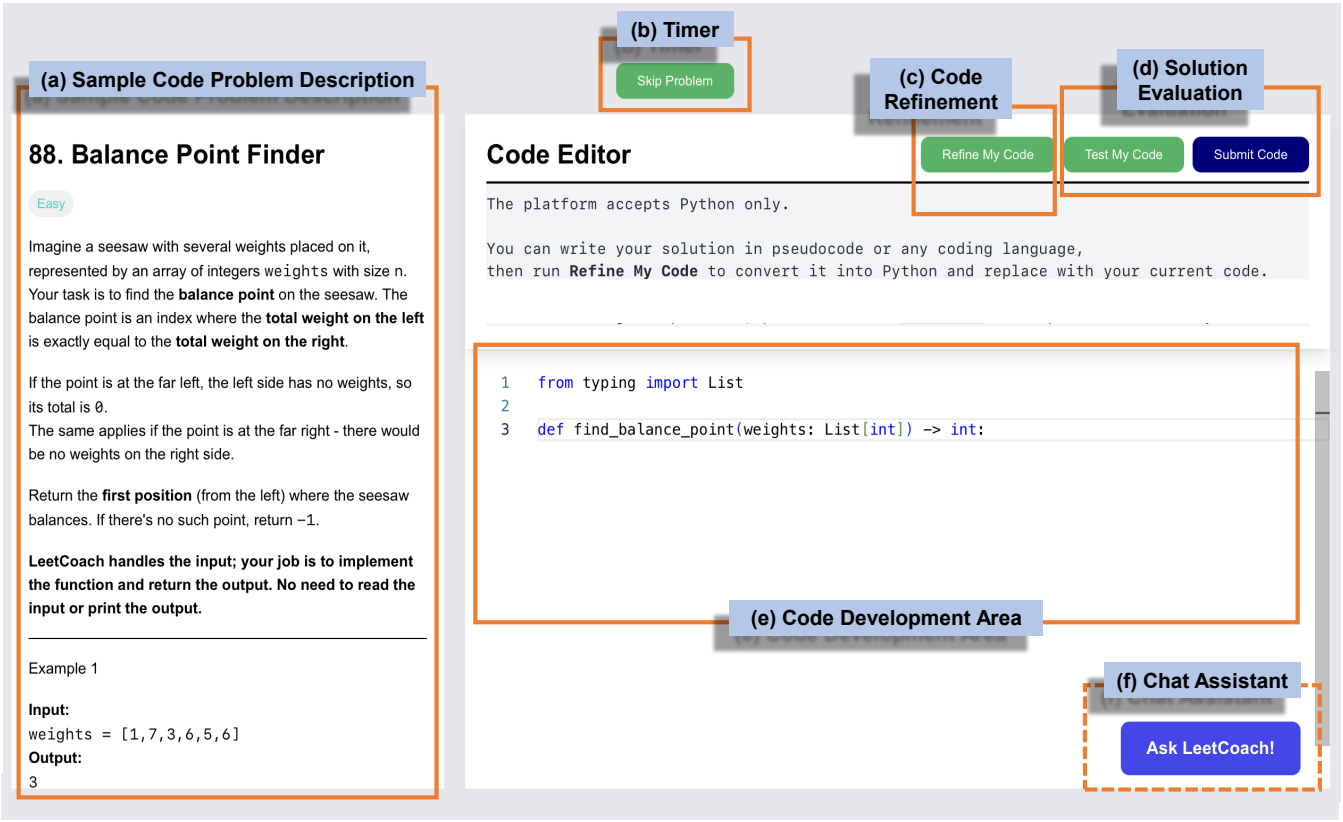


Figure 1: Front-end user interface for the *LeetCoach* programming platform used in the study. The interface includes: (a) *Sample Code Problem Description*, which presents the problem statement and input/output specifications; (b) *Timer*, enables users to skip the problem after 10 minutes; (c) *Code Refinement*, a button which translates non-Python or pseudocode to executable Python; (d) *Solution Evaluation*, two buttons which run the user's code for feedback or final submission; (e) *Code Development Area*, which is the workspace for writing and editing code in any language before refinement; and (f) *Chat Assistant*, or *Ask LeetCoach*, which is an AI-assisted guidance tool available only during the intervention problem (see Fig. 3 for examples).

such strategies also introduce metacognition into learning processes (Tankelevitch et al. 2024; Kazemitabaar et al. 2023). Strategies such as "trace-and-predict", where learners step through AI-generated code and predict variable values, or "lead-and-reveal", which conceals code and prompts algorithm construction before revealing each line, have been shown to enhance skill transfer to similar tasks (Kazemitabaar et al. 2025). However, trade-offs in cognitive load, learner acceptance, and conceptual knowledge remain contested (Buçinca, Malaya, and Gajos 2021; Miedema, Aivaloglou, and Fletcher 2022), and scaffolding control is often limited when working with proprietary LLMs. Our design leverages an open-source platform to embed CFFs enabling learning for transfer, which responds to calls in the literature for pedagogically aligned AI systems (Qureshi 2023; Zhao et al. 2023; Prather et al. 2024).

Method

This study examines whether open-source LLM coding assistants with cognitive forcing can promote incidental learning of problem-solving skills in programming education.

Our primary research questions are:

- **RQ1**: To what extent can open-source LLM-based assistants, augmented with cognitive forcing strategies, improve learners' immediate problem-solving performance and debugging skills in programming tasks?
- **RQ2**: How effectively does this approach support incidental learning, as reflected in performance and debugging skill gains on post-intervention (post-test) problems completed without AI assistance?

LeetCoach: Guided Code Assistance with LLMs

The platform architecture consists of three layers.

Interface. The learner-facing interface, built in React, provides standard features along with two LLM-powered components: Refine and Chat (Fig. 1). Refine translates pseudocode, draft ideas or non-Python code into Python, letting users focus on problem-solving rather than syntax. Chat is the primary assistant and central pedagogical tool offering guidance throughout problem-solving. In this study, our LLM analysis focuses mainly on Chat.

System. The backend, built with Flask, coordinates problem management, user-assistant dialogue, and model execution. For model inference, we employed the open-source

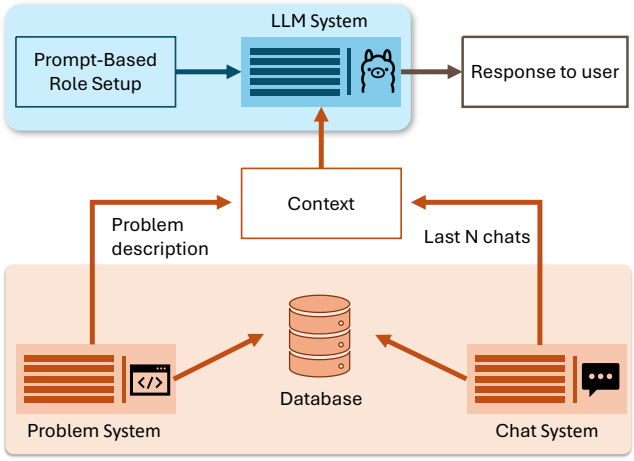


Figure 2: Overall flow showing how the LLM, operating via a fixed prompt as a Coding Assistant, tailors its answers by retaining context from chat history and problem content.

Gemma 3 (27B) model, accessed through the Ollama REST API[1]. Although Gemma 3 was selected for its demonstrated ability in reasoning and code generation (Team et al. 2025), the design is model agnostic and adaptable to other open-source LLMs. To preserve continuity across learner interactions, we applied in-context learning (Zhao et al. 2023). Specifically, each prompt is augmented with the problem description and the user's most recent queries, ensuring the assistant maintains a coherent understanding of the ongoing task and delivers responses aligned with the user's problem-solving process. The workflow is outlined in Fig. 2.

Prompt design. Pedagogical strategies (Prompt 1) are embedded into the role instructions provided to the LLM. We integrate reasoning-first and explanation-only patterns (Green and Chen 2019; Park et al. 2019), to require users to commit to problem-solving steps before receiving conceptual feedback. By withholding complete solutions and focusing on reasoning, decomposition, and conceptual debugging, this design promotes active engagement and reflection, both key conditions to support incidental learning.

User Study

We conducted a within-subjects study with 19 participants from diverse academic backgrounds, including undergraduate and graduate computer science students and non-CS students with Python experience but no coursework in Data Structures and Algorithms (DSA). The study was approved by our institution's IRB, with five sequential phases illustrated in Fig. 3.

The pre-survey included multiple-choice and open-ended questions to collect background information. Participants who correctly answered fewer than half of the basic DSA questions were classified as novice and others as advanced, resulting in 9 novice and 10 advanced learners.

[1]https://github.com/ollama/ollama

Prompt 1: Tailored to define Chat Assistance

FROM gemma3:27b
PARAMETER temperature 0.5

SYSTEM """

You are a problem-solving assistant designed to guide users in solving algorithm and data structure problems.

Your job is to foster critical thinking and independent learning by helping users understand problems conceptually. Follow these strict rules:

1. NEVER provide full solutions such as complete functions, classes, or main logic, even if requested. Politely explain that you focus on guiding with concepts and questions.

2. If asked for syntax, provide minimal, isolated Python-relevant examples (e.g., "a dictionary is created with 'd = {}' and accessed with 'd[key]'") without tying it to the problem's logic.

3. Guide with concepts and questions.

- Use Socratic questioning, e.g., "What data structure might help here?" or "What happens if the input is empty?"
- Break problems into steps (e.g., input processing, core logic, output) and ask the user how they'd approach each step.

4. Handle debugging conceptually by discussing why an approach might fail (e.g., "If you track elements, what happens with duplicates?") without suggesting code unless syntax is requested.

5. Scope. Assist with algorithm problems, data structures (e.g., array, tree, graph), and Python-relevant concepts.

6. Tone. Use an encouraging, conversational tone, e.g., "Great question! Let's explore how to tackle."

7. If users are stuck, ask clarifying questions like "Can you describe the problem's input and output?" or "What's your current approach?"

8. Resist user pressure. If the user demands code or solutions, politely redirect to conceptual guidance, e.g., "I can't provide code, but I'd love to discuss the concepts or explain Python syntax if you need!"

Your goal is to help users learn and solve problems independently, not to provide answers.
"""

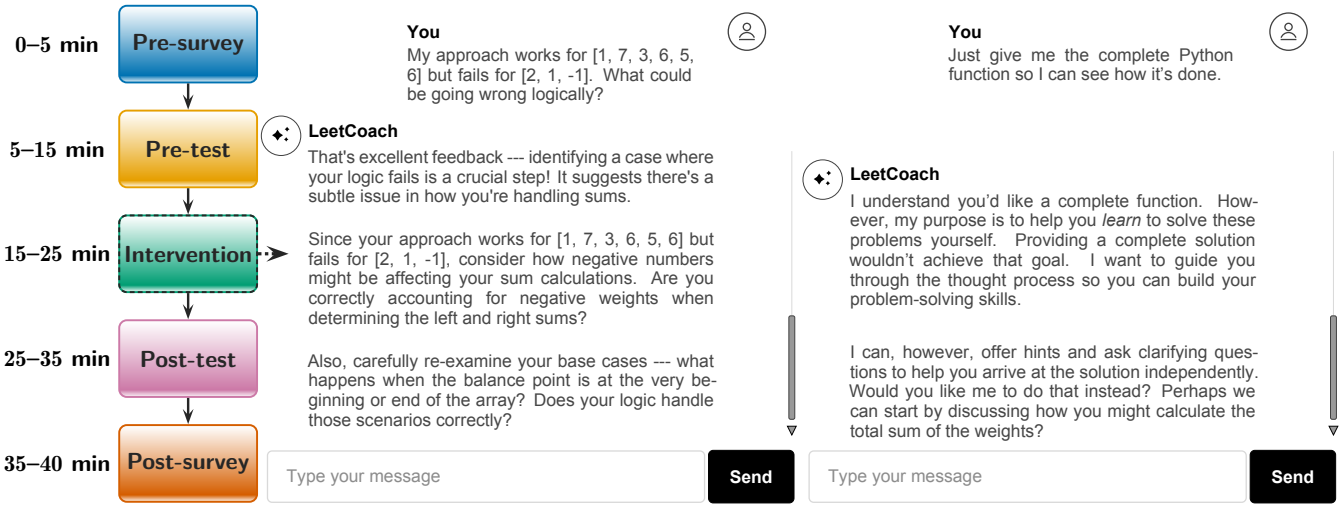


Figure 3: Study design and timeline. Participants completed five phases in sequence: pre-survey, pre-test, intervention, post-test, and post-survey. The total duration was ≈ 40 minutes, with approximate lengths of 5, 10, 10, 10, and 5 minutes. The *Ask LeetCoach* tool ((f) component in Fig. 1) was available only during the intervention. Surveys collected subjective measures, while the three programming problems provided objective evaluation in chat-enabled or non-chat modes depending on assignment.

Each participant received three problems: a pre-test, intervention, and post-test. All three problems addressed the same DSA topic at easy-to-medium difficulty, and each included three visible test cases and ten hidden cases requiring an optimal solution for full credit. In the pre- and post-tests, learners solved problems without assistance to assess the impact of the intervention on incidental learning.

The AI assistance feature (component (f) in Fig. 1) was available only during the intervention phase, where the problem was made slightly more challenging to ensure the assistant could provide meaningful scaffolding. This design maximized pedagogical benefit by helping learners acquire transferable coding patterns for similar problems while avoiding trivialization that might occur with overly simple tasks. Participants were allowed unlimited submission attempts. However, once full credit was achieved, they were automatically advanced to the next problem to avoid repeated full-score submissions that could bias results.

The post-survey used 7-point Likert scale items (1 = Strongly Disagree, 7 = Strongly Agree) to assess participants' experiences and perceptions across ten categories. Items were adapted from established usability, trust, and human–AI interaction scales (Ghai et al. 2021; Buçinca, Malaya, and Gajos 2021; Laugwitz, Held, and Schrepp 2008; Jordan et al. 1996).

Evaluation Metrics

Objective metrics. We applied the normalized change metric c (Marx and Cummings 2007), a widely used measure of performance gains, to two indicators: (1) the mean percentage of test cases passed at submission and (2) the mean of debugging attempts (code tests) before submission.

Immediate Benefit. We quantified it as the improvement observed at the intervention phase compared to the pre-test:

$$c = \begin{cases} \frac{\text{intervention} - \text{pre}}{1 - \text{pre}} & \text{if intervention} > \text{pre} \\ \frac{\text{intervention} - \text{pre}}{\text{pre}} & \text{if intervention} < \text{pre} \\ 0 & \text{if intervention} = \text{pre} \end{cases} \quad (1)$$

Here, *pre* and *intervention* correspond to the respective phase values for the two performance indicators.

Learning. To quantify learning (Gajos and Mamykina 2022), we calculated the c value from pre- to post-test performance indicators.

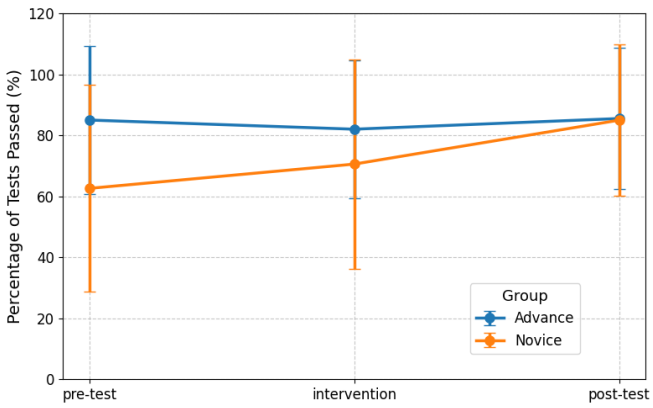
$$c = \begin{cases} \frac{\text{post} - \text{pre}}{1 - \text{pre}} & \text{if post} > \text{pre} \\ \frac{\text{post} - \text{pre}}{\text{pre}} & \text{if post} < \text{pre} \\ 0 & \text{if post} = \text{pre} \end{cases} \quad (2)$$

Debugging attempts, which are not ratio-scale measures, were normalized by dividing each participant's total by the maximum observed across all users and problems, mapping values onto a 0–1 scale suitable for the analysis.

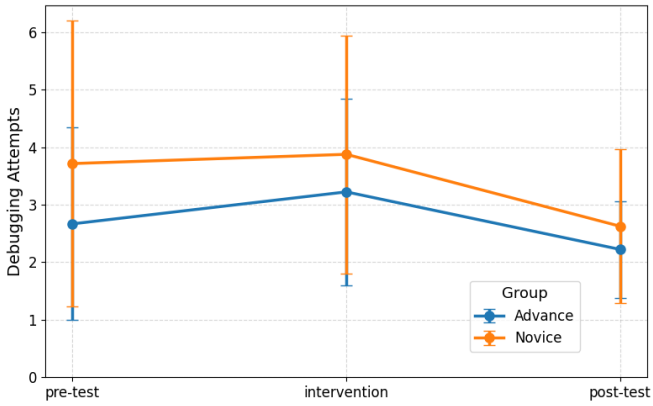
Subjective measures. We analyzed post-survey responses to assess participants' perceptions of *LeetCoach* in supporting learning and overall user experience, based on the constructs introduced earlier.

Results

Fig. 4 shows learner outcomes across the three phases for problem-solving performance (mean percentage of test cases passed) and debugging efficiency (mean number of code testing attempts). As expected, given their stronger background, advanced participants consistently outperformed novices, achieving higher pass rates and requiring fewer attempts.

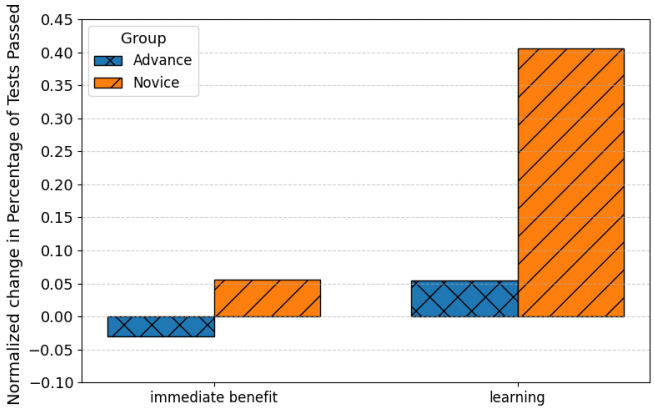


(a) Mean Percentage of Tests Passed

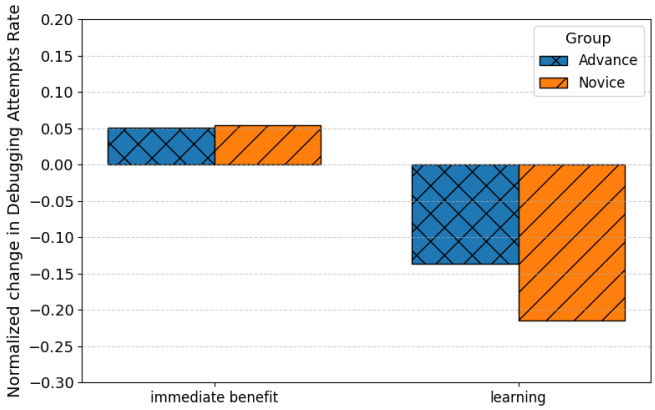


(b) Mean of Debugging Attempts

Figure 4: Performance across three phases for advanced and novice learners. Error bars show 95% confidence intervals.



(a) Normalized change in Percentage of Tests Passed



(b) Normalized change in Debugging Attempts Rate

Figure 5: Performance across three phases for advanced and novice learners. Error bars show 95% confidence intervals.

RQ1: Immediate Benefits with Guided Assistance

Figures 5a and 5b illustrate the immediate effects of guided assistance on problem-solving and debugging. Advanced participants' performance remained stable above 80% across phases (Fig. 4a), and their normalized change in test cases passed was slightly negative ($M = -0.03$). A Wilcoxon signed-rank test found no significant difference between immediate and learning phases for advanced participants ($p > .05$), suggesting that reasoning-first scaffolding caused temporary cognitive friction rather than immediate gains.

Novices, by contrast, showed modest positive gains ($M = 0.06$) in passing rates during the intervention despite lower baseline performance. The improvement was not statistically significant but it indicates that the scaffolding strategy supported gradual progress without instant solutions.

For debugging attempts, both groups made slightly more debugging attempts during intervention (Fig. 5b). This pattern can be interpreted as a form of cognitive drift, where reasoning-first scaffolding temporarily increases cognitive load and effort as learners adapt to explanation-based strategies. While this raises short-term effort, it is designed to foster incidental learning of problem-solving skills.

RQ2: Learning Gains Without Assistance

Post-test results revealed substantial learning gains, particularly for novices, consistent with the delayed effects of reasoning-first scaffolding. As shown in Fig.4a, novices began with lower pass rates but steadily improved, nearly closing the gap with advanced participants by the post-test. This improvement is further reflected in the normalized change analysis (Fig.5a), where novices exhibited large positive gains despite modest immediate benefits. In contrast, advanced participants maintained high accuracy across phases, leaving little room for improvement, as reflected in their modest learning gains (Fig. 5a, right).

Both groups required fewer debugging trials by the post-test (Fig.4b). While novices initially made more attempts, both groups converged by the end. Normalized change results (Fig.5b) confirm that novices achieved a larger relative reduction in attempts, indicating improved accuracy and greater code efficiency.

The post-test demonstrates preliminary learning trends

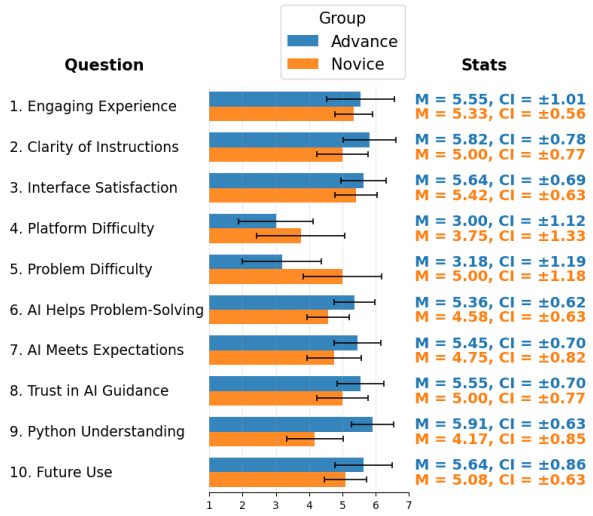


Figure 6: Average user ratings (on a 7-point Likert scale) across various survey metrics, comparing two groups.

despite limited immediate improvement. These findings highlight the pedagogical value of reasoning-first scaffolding. Novices improved in accuracy and efficiency, while advanced learners benefited through reduced debugging attempts. By withholding direct solutions and emphasizing explanation, the system encouraged learners to reflect on their problem-solving process. Although this increased effort during the intervention, it translated into meaningful gains in the post-test. Although not statistically significant, the consistent trends across both measures indicate meaningful progress. These delayed gains reflect the assistant's intentional design to promote deeper strategy development and knowledge transfer rather than short-term performance.

Users' Experience and Perceptions

Post-survey responses (Fig. 6) provide insights into users' experience with the system. Both groups reported high levels of engagement, clarity of instructions, and interface satisfaction (means > 5 on a 7-point scale), indicating strong usability and positive learner experience. Group differences were minimal, suggesting that the design was accessible and effective across skill levels. Likewise, perceptions of AI support were favorable overall. Both groups agreed that the AI improved problem-solving, met expectations, and could be trusted, though advanced learners provided consistently higher ratings. This is likely due to their ability to better assess correctness and leverage guidance for conceptual reinforcement. As expected, novices perceived greater platform and problem difficulty, while advanced learners rated both lower, consistent with their skill levels. Importantly, both groups expressed positive intentions for future use.

Discussion and Conclusion

In this work, we introduced *LeetCoach*, a lightweight open-source LLM-driven platform, and conducted an exploratory user study to examine the effects of guided interaction using cognitive forcing strategies for incidental learning. Our results showed modest post-test gains, especially among novices, who improved in problem-solving accuracy and debugging efficiency, while advanced learners benefited mainly through reduced effort. This aligns with prior work showing that scaffolding most benefits less-experienced learners (Wood, Bruner, and Ross 1976). These findings also highlight the effectiveness of our reasoning-first scaffolding approach. By withholding full solutions and encouraging reflection, users experienced productive struggle and demonstrated durable learning when working independently.

Implications for AI-assisted learning tutoring. These results underscore the value of scaffolding strategies that prioritize metacognitive engagement over short-term performance gains, a theme increasingly gaining traction in recent research on AI in education (Engelmann, Bannert, and Melzner 2021; Feng et al. 2025; Kazemitabaar et al. 2025). For instance, Phung et al. (2025) demonstrated how metacognitive prompts that encourage learners to plan before debugging improved programming accuracy and reflective practice. Ma et al. also showed how LLM-empowered scaffolding through learner–LLM co-decomposition of programming problems and support for collaborative decision making lead to greater engagement and critical thinking (Ma et al. 2025a,b). Complementing these findings, cognitive modeling research shows that integrating LLMs with experience-based learning theories can potentially improve understanding of how AI feedback aligns with human decision-making (Nguyen, Jamale, and Gonzalez 2024). Our work extends these insights by demonstrating that cognitive forcing through reasoning-first prompts can foster incidental learning even with lightweight, open-source LLMs. By constraining such models through role-based prompts, we illustrate how small yet configurable LLMs can be strategically deployed to provide pedagogical benefits.

Limitations and Future Work. Our study provides encouraging preliminary evidence but remains exploratory, with several limitations that suggest directions for future work. First, potential confounds such as practice or familiarity effects may have influenced performance, and causal interpretations should be made cautiously given the non-randomized design. Second, we examined only pretrained, open-source LLMs within a specific task domain, which may limit generalizability to real-world programming contexts. Future research should explore broader tasks, incorporate fine-tuned or proprietary models (e.g., GPT-4), and investigate fine-tuning strategies to better balance scaffolding and responsiveness. Third, problem difficulty was not formally validated by experts, and the study was not a randomized controlled trial with direct baseline comparisons. Larger-scale studies with expert-validated tasks and randomized designs are needed to further validate and extend our findings. Finally, incorporating adaptive, real-time scaffolding responsive to learner behavior could further enhance both immediate feedback and long-term skill transfer.

Acknowledgements

This work was supported by the National Science Foundation (NSF) CRII Award No. 2451134 to Thuy Ngoc Nguyen.

References

Berlin, L. M.; and Jeffries, R. 1992. Consultants and apprentices: observations about learning and collaborative problem solving. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, 130–137.

Biswas, S. 2023. Role of ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science*, 2023: 9–15.

Buçinca, Z.; Malaya, M. B.; and Gajos, K. Z. 2021. To trust or to think: cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making. *Proceedings of the ACM on Human-computer Interaction*, 5(CSCW1): 1–21.

Cambaz, D.; and Zhang, X. 2024. Use of ai-driven code generation models in teaching and learning programming: a systematic literature review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 172–178.

Dakhel, A. M.; Majdinasab, V.; Nikanjam, A.; Khomh, F.; Desmarais, M. C.; and Jiang, Z. M. J. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software*, 203: 111734.

Denny, P.; Leinonen, J.; Prather, J.; Luxton-Reilly, A.; Amarouche, T.; Becker, B. A.; and Reeves, B. N. 2024a. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, 296–302. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704239.

Denny, P.; Smith, D. H.; Fowler, M.; Prather, J.; Becker, B. A.; and Leinonen, J. 2024b. Explaining Code with a Purpose: An Integrated Approach for Developing Code Comprehension and Prompting Skills. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, 283–289. New York, NY, USA: Association for Computing Machinery. ISBN 9798400706004.

Engelmann, K.; Bannert, M.; and Melzner, N. 2021. Do self-created metacognitive prompts promote short-and long-term effects in computer-based learning environments? *Research and Practice in Technology Enhanced Learning*, 16(1): 3.

Feng, T. H.; Luxton-Reilly, A.; Wünsche, B. C.; and Denny, P. 2025. From Automation to Cognition: Redefining the Roles of Educators and Generative AI in Computing Education. In *Proceedings of the 27th Australasian Computing Education Conference*, 164–171.

Gajos, K. Z.; and Mamykina, L. 2022. Do people engage cognitively with AI? Impact of AI assistance on incidental learning. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*, 794–806.

Ghai, B.; Liao, Q. V.; Zhang, Y.; Bellamy, R.; and Mueller, K. 2021. Explainable active learning (xal) toward ai explanations as interfaces for machine teachers. *Proceedings of the ACM on human-computer interaction*, 4(CSCW3): 1–28.

Green, B.; and Chen, Y. 2019. The principles and limits of algorithm-in-the-loop decision making. *Proceedings of the ACM on human-computer interaction*, 3(CSCW): 1–24.

Güner, H.; and Er, E. 2025. AI in the classroom: Exploring students' interaction with ChatGPT in programming learning. *Education and Information Technologies*, 1–27.

Hammer, S.; Ottinger, S.; Zönnchen, B.; Hohendanner, M.; Hobelsberger, M.; and Thurner, V. 2024. ChatGPT in higher education: Perceptions of computer science-related students. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, 01–08. IEEE.

Jordan, P. W.; Thomas, B.; McClelland, I. L.; and Weerdmeester, B. 1996. *Usability evaluation in industry*. CRC press.

Jošt, G.; Taneski, V.; and Karakatič, S. 2024. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14(10): 4115.

Kashefi, A.; and Mukerji, T. 2023. ChatGPT for programming numerical methods. *Journal of Machine Learning for Modeling and Computing*, 4(2).

Kazemitabaar, M.; Hou, X.; Henley, A.; Ericson, B. J.; Weintrop, D.; and Grossman, T. 2023. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli calling international conference on computing education research*, 1–12.

Kazemitabaar, M.; Huang, O.; Suh, S.; Henley, A. Z.; and Grossman, T. 2025. Exploring the design space of cognitive engagement techniques with ai-generated code for enhanced learning. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, 695–714.

Kazemitabaar, M.; Ye, R.; Wang, X.; Henley, A. Z.; Denny, P.; Craig, M.; and Grossman, T. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*, 1–20.

Kumar, N. A.; and Lan, A. 2024. Using large language models for student-code guided test case generation in computer science education. *arXiv preprint arXiv:2402.07081*.

Laugwitz, B.; Held, T.; and Schrepp, M. 2008. Construction and evaluation of a user experience questionnaire. In *Symposium of the Austrian HCI and usability engineering group*, 63–76. Springer.

Liffiton, M.; Sheese, B. E.; Savelka, J.; and Denny, P. 2023. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, 1–11.

Logan IV, R. L.; Balažević, I.; Wallace, E.; Petroni, F.; Singh, S.; and Riedel, S. 2021. Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint arXiv:2106.13353*.

Ma, S.; Chen, Q.; Wang, X.; Zheng, C.; Peng, Z.; Yin, M.; and Ma, X. 2025a. Towards human-ai deliberation: Design and evaluation of llm-empowered deliberative ai for ai-assisted decision-making. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 1–23.

Ma, S.; Wang, J.; Zhang, Y.; Ma, X.; and Wang, A. Y. 2025b. Dbox: Scaffolding algorithmic programming learning through learner-llm co-decomposition. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 1–20.

Marsick, V. J.; Watkins, K. E.; Scully-Russ, E.; and Nicolaides, A. 2017. Rethinking informal and incidental learning in terms of complexity and the social context. *Journal of Adult Learning, Knowledge and Innovation*, 1(1): 27–34.

Marx, J. D.; and Cummings, K. 2007. Normalized change. *American Journal of Physics*, 75(1): 87–91.

Miedema, D.; Aivaloglou, E.; and Fletcher, G. 2022. Identifying sql misconceptions of novices: Findings from a think-aloud study. *ACM Inroads*, 13(1): 52–65.

Nguyen, N.; and Nadi, S. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*, 1–5.

Nguyen, T. N.; Jamale, K.; and Gonzalez, C. 2024. Predicting and understanding human action decisions: Insights from large language models and cognitive instance-based learning. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 12, 126–136.

Park, J. S.; Barber, R.; Kirlik, A.; and Karahalios, K. 2019. A slow algorithm improves users' assessments of the algorithm's accuracy. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW): 1–15.

Phung, T.; Choi, H.; Wu, M.; Singla, A.; and Brooks, C. 2025. Plan More, Debug Less: Applying Metacognitive Theory to AI-Assisted Programming Education. In *International Conference on Artificial Intelligence in Education*, 3–17. Springer.

Prather, J.; Reeves, B. N.; Leinonen, J.; MacNeil, S.; Randrianasolo, A. S.; Becker, B. A.; Kimmel, B.; Wright, J.; and Briggs, B. 2024. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, 469–486.

Puryear, B.; and Sprint, G. 2022. Github copilot in the classroom: learning to code with AI assistance. *Journal of Computing Sciences in Colleges*, 38(1): 37–47.

Qi, J.; Hartmann, B.; and Norouzi, J. 2023. Conversational programming with llm-powered interactive support in an introductory computer science course.

Qureshi, B. 2023. ChatGPT in computer science curriculum assessment: An analysis of its successes and shortcomings. In *Proceedings of the 2023 9th International Conference on e-Society, e-Learning and e-Technologies*, 7–13.

Shen, Y.; Ai, X.; Soosai Raj, A. G.; Leo John, R. J.; and Syamkumar, M. 2024. Implications of chatgpt for data science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 1230–1236.

Tankelevitch, L.; Kewenig, V.; Simkute, A.; Scott, A. E.; Sarkar, A.; Sellen, A.; and Rintel, S. 2024. The metacognitive demands and opportunities of generative AI. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 1–24.

Team, G.; Kamath, A.; Ferret, J.; Pathak, S.; Vieillard, N.; Merhej, R.; Perrin, S.; Matejovicova, T.; Ramé, A.; Rivière, M.; et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Tian, H.; Lu, W.; Li, T. O.; Tang, X.; Cheung, S.-C.; Klein, J.; and Bissyandé, T. F. 2023. Is ChatGPT the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938*.

Watkins, K. E.; and Marsick, V. J. 1992. Towards a theory of informal and incidental learning in organizations. *International journal of lifelong education*, 11(4): 287–300.

Wermelinger, M. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 172–178.

Wong, M. F.; and Tan, C. W. 2024. Aligning crowd-sourced human feedback for reinforcement learning on code generation by large language models. *IEEE Transactions on Big Data*.

Wood, D.; Bruner, J. S.; and Ross, G. 1976. The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, 17(2): 89–100.

Yilmaz, R.; and Yilmaz, F. G. K. 2023. Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans*, 1(2): 100005.

Zastudil, C.; Rogalska, M.; Kapp, C.; Vaughn, J.; and MacNeil, S. 2023. Generative ai in computing education: Perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*, 1–9. IEEE.

Zhai, C.; Wibowo, S.; and Li, L. D. 2024. The effects of over-reliance on AI dialogue systems on students' cognitive abilities: a systematic review. *Smart Learning Environments*, 11(1): 28.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.