LLM-powered Framework for Automatic Generation of Metacognitive Scaffolding Cues for Introductory Programming in Higher Education

Anushka Durg[1], Can Kultur[1], Adam Zhang[1] and Jaromir Savelka[1]

[1]Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Abstract

Scaffolding cues are instructional prompts designed to guide students through structured reasoning phases – understanding a task, planning a solution, and reflecting on their work – in order to support deeper learning during programming exercises. In this paper, we evaluate the capabilities of large language models (LLMs) to generate scaffolding cues for an introductory Python programming course in higher education. We used GPT-4 and TinyLlama to generate 126 scaffolding cues focused on understanding, planning, and reflecting for 14 programming exercises. We found that LLMs can reliably generate scaffolding cues that align with their intended reasoning type (Understand, Plan, Reflect), with expert annotators confirming the correctness of the reasoning type in 92% of cases overall. Further, we found that LLM-generated cues generally met instructional quality standards for clarity and relevance, though cues involving deeper reasoning (e.g., reflective depth) showed more variation and were harder to evaluate consistently by multiple annotators. The cues generated by GPT-4, in particular, were more likely to meet the quality criteria compared to TinyLlama, especially for Reflect-type cues. Overall, our findings suggest that LLMs could generate scaffolding cues that are clear, relevant, and useful for instruction. This could help reduce the time instructors spend authoring scaffolding cues or potentially enable personalized cues tailored to the needs of each individual student.

Keywords

Scaffolding cues, LLMs, Understand–Plan–Reflect, Intro CS, Rubric evaluation, GPT-4

1. Introduction

Finding the right answer often begins with asking the right question, especially for students learning to solve problems independently. Expert instructors can guide students toward productive lines of inquiry, but their time is scarce compared to the scale of the curriculum and the number of learners. Automatically generated scaffolding cues may help students better understand problems, plan solutions, and reflect on completed work. This may in turn lead to better learning outcomes. Scaffolding cues that encourage students to engage in structured reasoning about a learning activity—such as asking them to explain their plan, reflect on what worked, or connect the task to underlying logic—are well-established pedagogical techniques in computing education. Thinking before coding has been shown to reduce confusion and increase student success in logic-rich domains, while post-coding reflection supports transfer and long-term understanding [1, 2, 3]. In online platforms and other interactive environments, such reasoning can be encouraged by inclusion of scaffolding cues within problem statements to guide students toward deeper and more productive engagement [4, 5]. Our study uses Sail(), an online platform that functions as an interactive textbook—delivering curated instructional content alongside embedded exercises and cues. Similar to online textbooks like OLI[1] and Runestone,[2] Sail() serves as a real-world instantiation of the smart textbook vision.

iTextbooks'25: Sixth Workshop on Intelligent Textbooks, July 26, 2025, Palermo, Italy

adurg@andrew.cmu.edu (A. Durg); ckultur@cs.cmu.edu (C. Kultur); yufanz@andrew.cmu.edu (A. Zhang); jsavelka@cs.cmu.edu (J. Savelka)

0009-0004-3529-8684 (A. Durg); 0000-0002-6427-4161 (C. Kultur); 0000-0002-3674-5456 (J. Savelka)

[1]https://oli.cmu.edu/

[2]https://landing.runestone.academy/

However, manually authoring multiple scaffolding cues for every programming task may be prohibitively expensive. Instructors must not only write the cues for many tasks, but also tailor them across multiple reasoning types -typically centered around *understanding*, *planning*, and *reflecting*. The high cost of such effort has resulted in limited adoption of this useful approach at scale. Recent advances in large language models (LLMs) suggest that AI systems may be capable of generating such scaffolding cues automatically.

In this paper, we present a human-in-the loop framework to automatically generate cues intended to scaffold reasoning in coding tasks. We focus on three reasoning-support types, inspired by the structure of Polya's problem-solving method: "Understand" (interpreting the task), "Plan" (deciding how to proceed), and "Reflect" (evaluating or revising one's thinking) [6]. Our contributions are twofold:

- **We demonstrate that the LLMs are capable of generating scaffolding cues of the intended reasoning type.** In a blinded classification experiment, human experts accurately identified the type of each cue (e.g., Understand vs. Reflect) using only rubric definitions, indicating that LLM-generated reasoning structures are distinct and recognizable.
- **We evaluate LLM-generated scaffolding cues for programming instruction,** focused specifically on reasoning types (Understand, Plan, Reflect). Using a structured 9-point rubric and three reviewers, we show that GPT-4 can reliably produce cues that are clear, relevant, and pedagogically aligned across reasoning types.

To guide our investigations, we pose the following two research questions:

- **RQ1:** To what degree can LLMs generate scaffolding cues that are semantically recognizable as "Understand," "Plan," or "Reflect"?
- **RQ2:** How well do LLM-generated scaffolding cues meet quality standards in terms of clarity, relevance, and reasoning depth (defined in table 2)?

2. Related Work

2.1. Reasoning Before and After Programming Tasks

Many studies in computing education emphasize how important it is for students to reason before, during, and after coding. Planning, which uses natural languages, is essential for coding, which uses formal languages [7]. Explaining code and writing code are processes of transformation [8] that can be aided by careful reflections [9]. VanLehn's analysis showed that self-explanation in tutoring systems can lead to meaningful learning gains in structured problem domains [1]. Grover and Pea also highlighted how reflection and debugging help students build a deeper understanding of how programs work [2]. Koedinger et al. found that encouraging students to think through their approach before coding improved engagement and performance in large-scale online courses [3].

These approaches can be implemented in online platforms by including scaffolding cues to help students engage in the desired types of reasoning. Studies by Roll et al. and Kinnebrew et al. showed that when students are asked to explain their thinking at different points in time, (i.e., before, during, and after coding) they demonstrate better self-regulation and problem-solving skills [4, 5]. Based on these findings, we focus our work on three student reasoning types commonly recognized in curricular design: *Understand*, *Plan*, and *Reflect*.

2.2. Large Language Models for Generating Educational Content

Since the year of 2022, there have been considerable advancements in the educational applications of LLMs. The technology has been used to generate a wide variety of natural language artifacts in general educational context as well as in computing education. Leiker et al. developed a whole course utilizing

an LLM while keeping human experts in the loop to ensure high quality of the generated content [10]. Multiple research groups explored LLMs' potential to support learning by explaining a given code snippet [11, 12, 13]. Researchers have also shown that LLMs can be used to create programming exercises [14, 15]. There is a large body of work focused on LLMs' capabilities in generating model solutions to programming tasks [16, 17, 18]. There is ongoing work exploring the possibilities of generating real-time feedback or answers to student support requests in computing education [19, 20, 21, 22, 23, 24]. Other examples include personalized Parsons puzzles [25], MCQs [26, 27], or learning objectives [28, 29].

There is comparatively little work on using LLMs to generate scaffolding cues that are explicitly categorized by reasoning types. Most studies focus on correctness, explanation, or answer quality, rather than metacognitive structure. Our work provides a novel contribution by assessing whether LLMs can reliably generate scaffolding cues for different types of reasoning (i.e., understanding the task, planning the solution, and reflecting on performance).

3. Dataset

To support the experiments in this paper (see Section 4), we assembled a dataset of 14 coding exercises from the Practical Programming with Python course delivered on the Sail() platform.[3] This course has an interactive introductory Python programming curriculum that emphasizes practical data processing applications. The course is structured into eight instructional units, each containing auto-graded projects, quizzes, and online discussion. We focus on two specific units in this study: Unit 2 (Control Flow and String Manipulation) and Unit 3 (Data Structures).

We chose these two units for a key methodological reason: both include programming exercises in which students implement functions directly in a simple in-browser environment [30]. These exercises are relatively small and focused which makes them suitable for insertion of scaffolding cues and evaluation of the cues' quality.

Unit 2 includes three tasks:

- **Color Game:** Implement `time_color()` and `is_correct()` functions to compute game logic from player input and track correctness.
- **Tabular Reports:** Use iteration and formatting to align numerical output in a structured report.
- **Directory Contents Analysis:** Simulate parsing and analyze file metadata using loops and string conditions.

Unit 3 includes one task:

- **Container Type Agnostic API:** Write reusable logic to update, test, or convert among dicts, lists, tuples, and sets using unified condition checks.

Each task consists of several coding exercises which provide ideal focused points for inserting scaffolding cues. For example, in the *Color Game* task, one coding exercise asking students to "Write a function that formats integers with commas between every three digits, e.g., 1234567 → "1,234,567"" is aligned with the scaffolding cue *"This function isn't about math—it's about display. Why do we care about making large numbers easier to read, and what part of the logic must handle digit grouping?"*. A sample coding exercise from the *Color Game* is shown in Figure 1 in the form it is displayed on the online learning platform. To illustrate the kinds of scaffolding cues generated, Table 1 shows the full set of Understand, Plan, and Reflect prompts generated for the `time_color()` function, which maps time remaining to a visual color display.

[3] https://sailplatform.org/

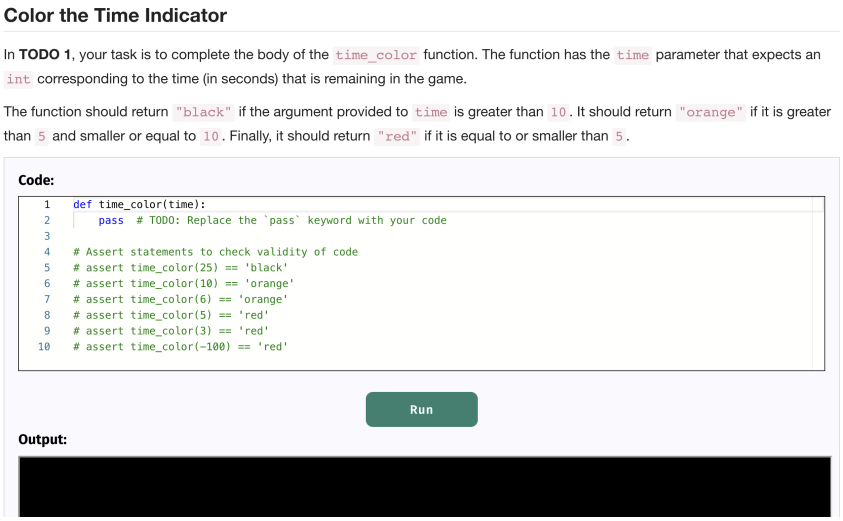

Figure 1: This code example from the Color Game in Unit 2 shows programming exercises. Scaffolding cues were added at each exercise for evaluation.

4. Experiments

In this section, we present our two-part evaluation framework, organized around the two research questions. RQ1 concerns whether LLM-generated scaffolding cues are semantically recognizable as a specific reasoning type (i.e., Understand, Plan, Reflect). RQ2 evaluates the instructional quality of these scaffolding cues using a structured rubric.

Scaffolding Cue Generation

The scaffolding cue generation process is schematically depicted in Figure 2. Each exercise includes embedded "TODO" comments which are ideal anchors for inserting scaffolding cues. For each exercise, we generated scaffolding cues targeting the three reasoning types:

- **Understand:** Helps the student interpret the task and underlying logic.
- **Plan:** Breaks the solution into steps, decisions, or code structure.
- **Reflect:** Encourages the student to analyze, revise, or explain their thinking.

The prompt submitted to an LLM contained:

- A brief task description
- The code context around the exercise
- The target reasoning type (U/P/R)
- A rubric-aligned scaffolding cue template (few-shot for GPT-4)

We used two models in our experiments:

- **GPT-4** (`gpt-4-0613`) via the OpenAI API
- **TinyLlama (1.1B)** running locally on a quantized inference engine

GPT-4 was prompted using a few-shot format that included exemplar scaffolding cues for each reasoning type. In contrast, TinyLlama was prompted using a simplified zero-shot format with only the task description, reasoning type, and instruction, as few-shot examples significantly degraded its output quality due to limited context window size.

Reasoning Type	Scaffolding Cue Prompt
Understand	You're mapping time to a display color — black for high time, red for low. Try to predict the thresholds before programming: what value should trigger red? What range should feel like a warning? Picture a timeline and reason through which time values should fall into each category and why.
	Try to imagine how a player would experience this function: if time is dropping quickly, when should the display change to orange or red? What would happen if your time conditions overlapped or left gaps? Could it return no color — or the wrong one — in those cases?
	If you see a color like 'red', what does that really mean in the game? Can you match that feeling of urgency to a number? Think about what these colors communicate and how the function needs to convert a numeric signal into a visual one.
Plan	Plan your if-elif-else structure: what will be your first condition? Which time values should be caught early? Use a number line if it helps — plot out values like 5, 10, 11 to see what range needs to go where.
	Break the function into small reasoning steps: How do you test for 'less than or equal to'? Will you handle red first or black first? Why? Which values will you use to confirm your code is behaving as expected at boundaries?
	Try programming the condition for red first, then build upward. Just write each rule as a test on time and check what color it returns.
Reflect	Now that you've implemented the logic, go back and ask: which time range was trickiest to classify? Did 5 go where you expected? Did 10?
	How did you decide the order of your conditions — and did it matter? Would changing the order of if/elif affect your outcome?
	If you tested time = 7 and got the wrong color, what would you look at first in your logic to fix it? What did you learn from doing that?

Table 1
Understand, Plan, and Reflect scaffolding prompts for the `time_color()` function (Unit 2 – Color Game, TODO 1).

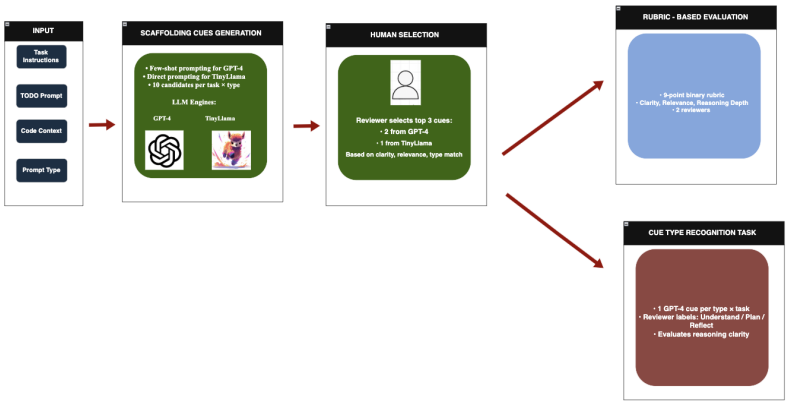

Figure 2: Scaffolding Cue Generation Pipeline for RQ1 and RQ2

For each combination of an exercise and scaffolding cue type, an LLM was prompted to generate **10 candidate scaffolding cues**. One of the authors (the one not included in expert evaluations described below) reviewed these and selected the best **three scaffolding cues** based on clarity, relevance, and conceptual alignment with the exercise. The selection also respected the following constraints:

- **2 scaffolding cues** from GPT-4
- **1 scaffolding cue** from TinyLlama

We repeated this process across the 14 exercises which resulted in:

$$14 \text{ exercises} \times 3 \text{ cue types} \times 3 \text{ scaffolding cues} = \textbf{126 scaffolding cues in total}.$$

The pipeline implements a human-AI interaction by combining automated and manual components:

- Scaffolding cue generation was fully automated via LLMs.
- Scaffolding cue selection was manual: we chose the top 3 of 10 cues for each reasoning type.

This setup combines model-generated content with comparatively inexpensive human supervision to ensure instructional value.

4.1. RQ1: Cue Type Correctness

To evaluate whether the generated scaffolding cues exhibit distinct semantic structures that correspond to given reasoning types, one of the authors performed a simple annotation task. We randomly sampled a single high-quality scaffolding cue of each reasoning type (U/P/R) for each exercise from the GPT-4 outputs. This yielded **42 scaffolding cues**. These scaffolding cues were then randomly shuffled, removing all indicators of reasoning type.

Annotation Procedure. One of the authors classified each scaffolding cue into one of three categories: Understand, Plan, or Reflect. The author was provided the rubric shown in Table 2. The task was to assign each scaffolding cue to the category it most closely matched, using only the rubric definitions.

Evaluation Metrics. We compared the original label followed by the model to the manual annotations. We calculated accuracy for each scaffolding cue type. We also reviewed the mistakes to understand which types were often generated improperly.

4.2. RQ2: Scaffolding Cue Quality

To assess the quality of the generated scaffolding cues, we scored the full set of 126 scaffolding cues using a structured rubric focused on features such as clarity, relevance, and reasoning depth (see Table 2 for details).

Annotator Assignment. Two authors (different from the author that performed the annotations for RQ1) annotated all the 126 scaffolding cues. Each annotator scored the scaffolding cues independently and using the rubric shown in Table 2.

Rubric-Based Evaluation. Each scaffolding cue was scored using a structured 9-item binary rubric shown in Table 2. Each reasoning type had three corresponding rubric criteria. Annotators marked each item as either "Yes" (1) or "No" (0). Scores were aggregated to compute the proportion of criteria met per scaffolding cue.

Evaluation

Each scaffolding cue was scored using the following formula:

$$\text{Rubric Score} = \frac{\text{\# of "Yes" ratings on relevant criteria}}{3}$$

For example, a Plan-type scaffolding cue marked "Yes" on "Clarity" and "Step-Based" but "No" on "Code-Focused" would receive a score of $\frac{2}{3} = 0.67$. Scores were then averaged per scaffolding cue type and per trait to produce the summaries shown in Figure 3.

Scaffolding cue Type	Criterion	Definition
Understand	Clarity	Is the question phrased clearly and without confusing language? A student should be able to rephrase it confidently.
	Relevance	Is the question about this exact task — not a general Python idea or unrelated topic? It should mention key features like time, logic, or structure.
	Conceptual Clarity	Does it help the student understand how the task works — not just what to do? It should bring out a reasoning point.
Plan	Clarity	Is the suggestion or thinking path easy to follow? A student should be able to imagine the steps.
	Step-Based	Does it break the task into logical parts — like what to check first, what to test, or how to structure decisions?
	Code-Focused	Does it relate directly to the kind of code they'll write — like if-statements, range logic, or what values to handle?
Reflect	Clarity	Can the student tell what part of their thinking or code they're supposed to reflect on?
	Relevance	Is it clearly about this task's logic — not how hard it was or how they felt?
	Reflective Depth	Does it ask them to explain what they figured out, fixed, or might reuse — not just if it worked?

Table 2
U/P/R scaffolding cue Evaluation Rubric

5. Results

We report results for both research questions outlined in Section 1, corresponding to the correctness of reasoning types (RQ1) and rubric-based quality assessment (RQ2).

5.1. RQ1: Cue Type Correctness

To assess whether the intended type of scaffolding cues was generated, one of the authors of this paper annotated 42 scaffolding cues as either Understand, Plan, or Reflect. Table 3 shows the results of this experiment. The results suggest that LLMs can reliably generate cues for the prescribed category. The only disagreement in the expert annotations is that several Understand-type cues generated by the LLM have been annotated as either Plan-type or Reflect-type by the human expert. However, classification accuracy was promising. By correct, we mean that the LLM was able to generate a scaffolding cue whose type as determined by the annotator matches the type that is intended by the prompt.

- **Understand:** 20/20 correct (100%)
- **Plan:** 29/34 correct (85.3%) – 5 misclassified as Understand
- **Reflect:** 32/34 correct (94.1%) – 2 misclassified as Understand

Most errors occurred between Plan and Understand, possibly due to overlap in language used to scaffold pre-solution reasoning. Notably, no Reflect scaffolding cues were misclassified as Plan, suggesting that the reflective structure was likely distinctive.

5.2. RQ2: Scaffolding Cue Quality

Figure 3 shows the rubric-based evaluation of scaffolding cues as scored independently by two human annotators—Reviewer A and Reviewer B—on the same dataset. This setup allows us to analyze how differences in reviewer interpretation impact rubric-based scoring.

Table 3
Scaffolding cue-type classification confusion matrix comparing predicted vs. intended reasoning categories.

	Annotated by human		
Generated	**Understand**	**Plan**	**Reflect**
Understand	20	0	0
Plan	5	29	0
Reflect	2	0	32

Note that there was very little agreement between the two annotators which suggests that the rubric needs to be further improved. For this reason, we report the raw results of the annotation process.

- **Understand:** Both reviewers rated Understand-type cues highly, with Reviewer B assigning slightly higher average scores across traits such as "Understand – Clarity" (0.94 vs. 0.85) and "Understand – Conceptual Clarity" (0.83 vs. 0.77).
- **Plan:** Reviewer B again scored consistently higher on planning traits, especially for "Plan – Clarity" (0.91 vs. 0.72), suggesting differences in how the two reviewers interpreted the clarity of planning steps.
- **Reflect:** The largest disagreement appeared in Reflect-type traits. Reviewer B rated "Reflect – Clarity" almost perfectly (0.99) compared to Reviewer A's 0.77. A similar gap was observed for "Reflect – Reflective Depth" (0.84 vs. 0.49).

To better understand these disparities, Figure 3 breaks down rubric scores by individual criterion. Reviewer B tended to assign higher scores across most traits, whereas Reviewer A's scores reflected more variation—particularly on traits involving deeper reasoning such as "Plan – Clarity" and "Reflect – Reflective Depth." These differences highlight not only the difficulty of scoring certain instructional traits consistently, but also suggest that rubric criteria such as "depth" may require clearer anchors or reviewer calibration. Rather than treating variation as noise, we interpret it as insight into how instructors with different pedagogical lenses might differently value the same cue.

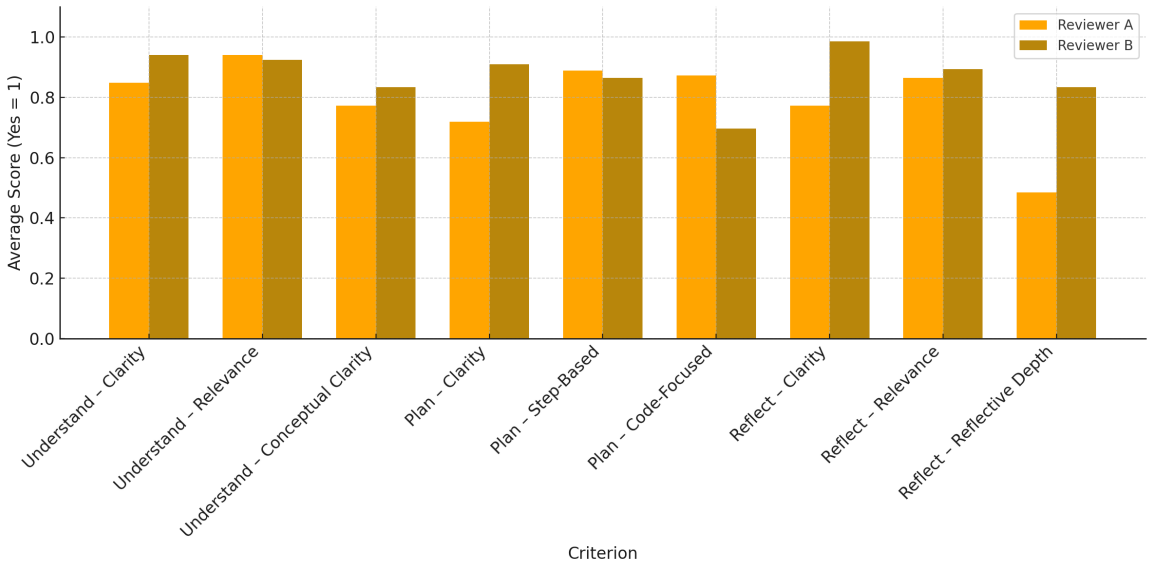

Figure 3: Average rubric score per criterion. Scores reflect proportion of scaffolding cues that met each trait(Yes = 1).

GPT-4 vs TinyLlama

Since each task included two scaffolding cues from GPT-4 and one from TinyLlama, we can compare average rubric scores between the models. Across all reasoning types and rubric traits:

- **GPT-4 scaffolding cues ($n = 84$)** had a mean rubric score of **0.83**.
- **TinyLlama scaffolding cues ($n = 42$)** had a mean rubric score of **0.67**.

This reflects a clear advantage for GPT-4 in producing scaffolding cues that meet human-annotated instructional standards, particularly in reflective depth and task relevance.

6. Discussion

Our results suggest that LLMs—especially a frontier model such as GPT-4—can generate scaffolding cues that are both rubric-aligned and semantically distinct across reasoning types. The cues generated for specific reasoning types (Understand, Plan, Reflect) were annotated by reviewers in alignment with those intended types (RQ1). This supports the conclusion that LLMs are capable of producing reasoning-aligned cues consistent with the type they were instructed to generate. Reviewer agreement with the LLM's intended category serves as evidence that the cues exhibit recognizable structural and semantic features tied to each reasoning type.

GPT-4 consistently outperformed TinyLlama across Understand, Plan, and Reflect categories, with particularly strong performance on Reflect cues (RQ2). These results support prior findings that LLMs are especially well-suited to post-coding reasoning support, particularly for generating reflective cues that help students evaluate and revise their thinking [4, 5, 31]. However, trait-level analysis revealed variability in how cues were rated, particularly in Plan Clarity and Reflective Depth.

During our evaluation, we noticed that the two reviewers often gave different scores for the same scaffolding cues, especially for traits like Reflective Depth and Plan Clarity. This variation was due to the fact that the rubric itself was still being refined. As a result, some criteria left room for interpretation, leading reviewers to apply different expectations or teaching philosophies when making judgments. We chose to report the reviewers' ratings separately as it better reflects real-world teaching, where instructors often bring different perspectives to how student thinking is evaluated. It also acknowledges that some instructional traits—especially deeper reasoning skills—are naturally harder to score consistently without more calibration or a more mature rubric.

From a teaching perspective, these findings highlight both the potential and the limits of using LLMs to generate reasoning-aligned scaffolding cues. On the one hand, GPT-4 shows strong capability in generating cues that support conceptual understanding and post-coding reflection. This can reduce the time instructors spend drafting scaffolding cues from scratch. On the other hand, the variation in reviewer scores emphasizes the importance of human review, especially for traits involving reflection. As one of the reviewers noted in their comments, "the question is often not whether a prompt is technically clear, but whether it reflects the kind of reasoning or level of support the instructor intends to foster." Reviewer B also raised concerns about reasoning structure, stating that a cue "doesn't provide thinking through steps," and pointed out limits in reflective support, noting "it is not clear what to reuse, and fix what for what." Such comments highlight the importance of aligning scaffolding cues not only with rubric traits, but also with instructional intent.

Our findings also have broader implications for instructional design. When two reviewers disagree about whether a cue demonstrates "clarity" or "depth," it often reflects not rubric failure but differences in pedagogical goals. Rather than enforcing universal definitions of these traits, it may be more useful to view LLM-generated cues as adaptable resources where instructors can tune to their own teaching style or course context. LLMs may not eliminate the need for expert judgment, but they can speed up the process of drafting, revising, and contextualizing reasoning supports at scale.

Limitations

While our findings are promising, several limitations remain. We did not test the generated scaffolding cues in live instructional settings. As a result, we cannot make claims about their actual impact on student learning, engagement, or performance. We did not assess whether the scaffolding cues helped students achieve specific learning goals or improved understanding. Future work should pair scaffolding cue exposure with outcome metrics. Although our study involved 126 scaffolding cues and 14 exercises, the number of scaffolding cue types and reviewers remains limited. Including more types of tasks and more reviewers would help test if the findings apply more broadly. Scaffolding cue selection, model completions, and rubric scoring all involve human interpretation. Bias may have influenced both scaffolding cue filtering and annotation.

7. Conclusions

In this paper, we explore whether LLMs can generate high-quality reasoning scaffolding cues for small programming tasks. We focus on three types of reasoning support. Understand, Plan, and Reflect. Using exercises from an introductory Python course on the Sail() platform, we generated a dataset of 126 scaffolding cues using GPT-4 and TinyLlama, and evaluated them across two research questions.

A reviewer was able to confirm the correctness of scaffolding cues with respect to their intended types (U/P/R), showing that the structure of reasoning was clear and distinguishable (RQ1). Two other reviewers rated the scaffolding cues using a structured rubric and found that most scaffolding cues met important criteria for clarity, relevance, and reasoning depth (RQ2). GPT-4 scaffolding cues performed consistently well, especially for Reflect-type reasoning.

These results suggest that LLMs—especially GPT-4—can help instructors generate reasoning-aligned scaffolding cues that reduce manual effort while supporting student thinking in code-based tasks. Even with a small dataset, the scaffolding cues showed strong alignment to pedagogical goals and reflected recognizable reasoning structures.

8. Future Work

Future work will involve testing the scaffolding cues in real classrooms to study their impact on learning outcomes, utilizing scaffolding cues generated by both LLMs and expert instructors in a single-blinded A/B test. We also plan to refine the evaluation process by improving the clarity and granularity of the rubric itself, informed by the discrepancies observed between the two annotators. As seen in our results, reviewer disagreement was especially pronounced for traits like Plan Clarity and Reflective Depth, indicating that rubric refinement may be necessary to support consistent scoring. Beyond evaluation design, we will also explore whether LLMs can be guided to improve their performance on specific weak traits and whether models can be fine-tuned or prompted to generate personalized scaffolding cues based on student-level performance data. Larger-scale evaluations with more tasks and reviewers will help validate the generalizability of these findings and support broader classroom integration.

References

[1] K. VanLehn, The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems, Educational Psychologist 40 (2005) 195–221.

[2] S. Grover, R. Pea, Computational thinking in k–12: A review of the state of the field, Educational Researcher 42 (2015) 38–43.

[3] K. R. Koedinger, J. Kim, J. Jia, E. A. McLaughlin, N. L. Bier, Learning is not a spectator sport: Doing is better than watching for learning from a mooc, ACM Transactions on Computing Education (TOCE) 15 (2015) 1–24.

[4] I. Roll, V. Aleven, B. M. McLaren, K. R. Koedinger, Metacognitive scaffolding for learning programming, in: International Conference on Artificial Intelligence in Education, Springer, 2011, pp. 789–791.

[5] J. S. Kinnebrew, G. Biswas, Planning, reflection, and self-regulation in open-ended learning environments, in: International Conference on Artificial Intelligence in Education, Springer, 2013, pp. 202–211.

[6] G. Polya, How to Solve It: A New Aspect of Mathematical Method, Princeton University Press, 1945.

[7] L. E. Winslow, Programming pedagogy—a psychological overview, SIGCSE Bull. 28 (1996) 17–22. URL: https://doi.org/10.1145/234867.234872. doi:10.1145/234867.234872.

[8] J. F. PANE, C. A. RATANAMAHATANA, B. A. MYERS, Studying the language and structure in non-programmers' solutions to programming problems, International Journal of Human-Computer Studies 54 (2001) 237–264. URL: https://www.sciencedirect.com/science/article/pii/S1071581900904105. doi:https://doi.org/10.1006/ijhc.2000.0410.

[9] A. Naik, J. R. Yin, A. Kamath, Q. Ma, S. T. Wu, C. Murray, C. Bogart, M. Sakr, C. P. Rose, Generating situated reflection triggers about alternative solution paths: A case study of generative ai for computer-supported collaborative learning, in: Artificial Intelligence in Education: 25th International Conference, AIED 2024, Recife, Brazil, July 8–12, 2024, Proceedings, Part I, Springer-Verlag, Berlin, Heidelberg, 2024, p. 46–59. URL: https://doi.org/10.1007/978-3-031-64302-6_4. doi:10.1007/978-3-031-64302-6_4.

[10] D. Leiker, S. Finnigan, A. R. Gyllen, M. Cukurova, Prototyping the use of large language models (llms) for adult learning content creation at scale, in: LLM@AIED, 2023. URL: https://api.semanticscholar.org/CorpusID:259076210.

[11] S. MacNeil, A. Tran, D. Mogil, S. Bernstein, E. Ross, Z. Huang, Generating diverse code explanations using the gpt-3 large language model, ICER '22, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3501709.3544280. doi:10.1145/3501709.3544280.

[12] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, J. Leinonen, Experiences from using code explanations generated by large language models in a web software development e-book, SIGCSE 2023, ACM, New York, NY, USA, 2023, p. 931–937. URL: https://doi.org/10.1145/3545945.3569785. doi:10.1145/3545945.3569785.

[13] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas, Comparing code explanations created by students and large language models, Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (2023). URL: https://api.semanticscholar.org/CorpusID:258049009.

[14] S. Sarsa, P. Denny, A. Hellas, J. Leinonen, Automatic generation of programming exercises and code explanations using large language models, ACM, 2022. URL: https://doi.org/10.1145%2F3501385.3543957. doi:10.1145/3501385.3543957.

[15] A. Del Carpio Gutierrez, P. Denny, A. Luxton-Reilly, Evaluating Automatically Generated Contextualised Programming Exercises, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, ACM, Portland OR USA, 2024, pp. 289–295.

[16] P. Denny, V. Kumar, N. Giacaman, Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 1136–1142. URL: https://doi.org/10.1145/3545945.3569823. doi:10.1145/3545945.3569823.

[17] S. R. Piccolo, P. Denny, A. Luxton-Reilly, S. Payne, P. G. Ridge, Many bioinformatics programming tasks can be automated with chatgpt, arXiv preprint arXiv:2303.13528 (2023).

[18] J. Savelka, A. Agarwal, M. An, C. Bogart, M. Sakr, Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses, in: Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1, ICER '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 78–92.

URL: https://doi.org/10.1145/3568813.3600142. doi:10.1145/3568813.3600142.
[19] M. Liffiton, B. Sheese, J. Savelka, P. Denny, Codehelp: Using large language models with guardrails for scalable support in programming classes, arXiv preprint arXiv:2308.06921 (2023).
[20] B. Sheese, M. Liffiton, J. Savelka, P. Denny, Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant, in: Proceedings of the 26th Australasian Computing Education Conference, ACM, Sydney NSW Australia, 2024, pp. 49–57.
[21] M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, T. Grossman, CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, ACM, Honolulu HI USA, 2024, pp. 1–20.
[22] P. Bassner, E. Frankford, S. Krusche, Iris: An AI-Driven Virtual Tutor For Computer Science Education, 2024. URL: http://arxiv.org/abs/2405.08008, arXiv:2405.08008 [cs].
[23] J. D. Zamfirescu-Pereira, L. Qi, B. Hartmann, J. DeNero, N. Norouzi, 61A-Bot: AI homework assistance in CS1 is fast and cheap – but is it helpful?, 2024. URL: http://arxiv.org/abs/2406.05600, arXiv:2406.05600 [cs].
[24] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, D. J. Malan, Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, ACM, Portland OR USA, 2024, pp. 750–756. URL: https://dl.acm.org/doi/10.1145/3626252.3630938.
[25] X. Hou, Z. Wu, X. Wang, B. J. Ericson, CodeTailor: LLM-Powered Personalized Parsons Puzzles for Engaging Support While Learning Programming, 2024. URL: http://arxiv.org/abs/2401.12125, arXiv:2401.12125 [cs].
[26] J. Doughty, Z. Wan, A. Bompelli, J. Qayum, T. Wang, J. Zhang, Y. Zheng, A. Doyle, P. Sridhar, A. Agarwal, et al., A comparative study of ai-generated (gpt-4) and human-crafted mcqs in programming education, in: Proceedings of the 26th Australasian Computing Education Conference, 2024, pp. 114–123.
[27] A. Tran, K. Angelikas, E. Rama, C. Okechukwu, D. H. Smith, S. MacNeil, Generating multiple choice questions for computing courses using large language models, in: 2023 IEEE Frontiers in Education Conference (FIE), IEEE, 2023, pp. 1–8.
[28] P. Sridhar, A. Doyle, A. Agarwal, C. Bogart, J. Savelka, M. Sakr, Harnessing llms in curricular design: Using gpt-4 to support authoring of learning objectives, arXiv preprint arXiv:2306.17459 (2023).
[29] A. Doyle, P. Sridhar, A. Agarwal, J. Savelka, M. Sakr, A comparative study of ai-generated and human-crafted learning objectives in computing education, Journal of Computer Assisted Learning 41 (2025) e13092.
[30] H. A. Nguyen, C. Bogart, J. Šavelka, A. Zhang, M. Sakr, Examining the trade-offs between simplified and realistic coding environments in an introductory python programming class, in: European Conference on Technology Enhanced Learning, Springer, 2024, pp. 315–329.
[31] A. Naik, J. R. Yin, A. Kamath, Q. Ma, S. T. Wu, C. Murray, C. Bogart, M. Sakr, C. P. Rose, Generating situated reflection triggers about alternative solution paths: A case study of generative ai for computer-supported collaborative learning, in: Artificial Intelligence in Education: 25th International Conference, AIED 2024, Recife, Brazil, July 8–12, 2024, Proceedings, Part I, Springer-Verlag, Berlin, Heidelberg, 2024, pp. 46–59. URL: https://doi.org/10.1007/978-3-031-64302-6_4. doi:10.1007/978-3-031-64302-6_4.