International Society of the Learning Sciences

Enhance Code Understanding through Prompted Self-Explanation with Mixed-Up Code Puzzles: Novice Preferences and Outcomes

Xinying Hou, University of Michigan, xyhou@umich.edu
Evie Katmanivong, University of Michigan, ekatmani@umich.edu
Xu Wang, University of Michigan, xwanghci@umich.edu
Barbara J. Ericson, University of Michigan, barbarer@umich.edu

Abstract: As AI enters computer science (CS) education, novices need early opportunities to strengthen code understanding. This work responds to this need by investigating pairing mixed-up code puzzles (Parsons puzzles) with self-explanation (SE) prompts. Drawing on learning sciences work, we designed two SE prompts: Select, where students engage in SE by choosing correct options, and Fill, where they engage in SE by typing answers. In an in-depth within-subjects study (N=10), all novices valued SE prompts for understanding code after solving puzzles. Select was strongly preferred to Fill for its efficiency and scaffolding. Fill was removed due to frustration from likely non-learning mental effort. A between-subjects study (N=110) in an intro-level CS classroom compared practice with puzzles-only to puzzles plus Select SEs. The group solving puzzles with Select showed significantly greater pre-to-posttest learning gains with similar perceived in-practice mental effort.

Introduction

Investment in computer science (CS) education is expanding as computing becomes increasingly intertwined with society. Computer programming is a complex domain requiring strong cognitive and procedural skills, especially from novices (Robins, 2019). This makes the ability to comprehend code a foundational skill (Fowler et al., 2022). Given a piece of code, students must understand why and how a procedure works to generalize it to a broader range of problems (Izu et al., 2019). Due to the rise of AI programming assistants, many instructors have emphasized devoting more time in intro-level courses to code program understanding and critical evaluation (Mahon et al., 2024). However, as AI can produce code that appears correct, students may even be unaware of the need to understand code, not to mention evaluating it (Prather et al., 2024).

Therefore, to enhance novices' code understanding, deliberate practice in code comprehension should be integrated with problem-solving from the beginning of CS learning. Researchers in the learning sciences have examined prompted self-explanation (SE) activities as an effective way to foster self-explanation (Bisra et al., 2018). SE prompts can be incorporated into problem-solving tasks (Chiu & Chi, 2014) to prompt students to engage in self-explanation (Chi et al., 1989; Zhang & Fiorella, 2024). Within CS education, it holds promise for improving students' comprehension of code. One problem-solving activity in intro CS classes is mixed-up code puzzles (*Parsons puzzles*), where students select and place mixed-up code blocks in the correct order to solve a problem (Parsons & Haden, 2006). In a Parsons puzzle, students complete code solutions without the burden of writing code from scratch, which delivers a simplified yet authentic problem-solving experience (Haynes & Ericson, 2021; Weinman et al., 2021). Integrating prompted SE activities with Parsons puzzles can further enhance solution understanding, thereby advancing our goal of strengthening novices' code comprehension in the early stages of CS learning.

In this work, drawing from prior learning sciences research on SE prompts (Price et al., 2020; Wylie & Chi, 2014), we first designed and employed two types of SE prompts: *Select* and *Fill*. We then conducted a within-subjects study with 10 undergraduate programming novices, using both types to understand their perceived values, type preferences and reasons. Students were asked if the SE prompts helped them understand the code, their preferred SE prompt type, and the reasons for that preference. The first two RQs were:

***RQ1*:** Do students value SE prompts with Parsons puzzles for understanding code, and why?

***RQ2*:** Which SE prompt type (Select or Fill) do students prefer after a Parsons puzzle, and why?

Next, we conducted a between-subjects classroom study in an undergraduate intro-level CS course with two conditions: 1) solving a Parsons puzzle followed by a SE prompt vs 2) solving only a Parsons puzzle. This study measured learning gains and students' perceived mental effort during practice:

***RQ3*:** Do students' learning gains from pretest to posttest differ by condition?

***RQ4*:** Do students' perceived in-practice mental effort differ by condition?

Related Work

Self-explanation (SE) in learning sciences and different types of SE prompts

Self-explanation (SE) is a process in which students make inferences about relationships to absorb concepts better (Bisra et al., 2018). Research in the learning sciences has shown SE to be one of the most robust learning strategies across different domains and educational settings (Bisra et al., 2018; Kwon et al., 2011). However, students often face difficulties with SE. Ideally, students should initiate this process themselves, but many struggle to conduct in-depth SE independently (McCarthy, 2025; Renkl, 2002).

One approach is to trigger SE with deliberate activities. SE prompts encourage active reflection and reasoning (Kwon et al., 2011; Su et al., 2025). Learning sciences scholars proposed a *prompted SE format spectrum* that varies by the support level of a self-explanation prompt (Wylie & Chi, 2014). At one end, open-ended prompts ask leading questions without immediate feedback; at the other, menu-based prompts have students select from predetermined options (Wylie & Chi, 2014). In the middle is scaffolded SE prompts, where students complete explanations by filling in blanks (Wylie & Chi, 2014). However, even when students engaged in SE, they can still produce incorrect explanations (Renkl, 2002), leading to misconceptions. To address this, scholars recommended hints and immediate feedback so students can correct their mental models (Aleven & Koedinger, 2002), which our design also incorporated.

Additionally, some SE prompts may impede learning when they lead to cognitive overload (Joo et al., 2020). Cognitive load theory is a framework used to determine effective instructional design based on students' limited cognitive processing capacity (Sweller, 2020). This framework highlights two main types of cognitive load: intrinsic cognitive load refers to the built-in complexity of the materials and is hard to adjust; instructional materials influence extraneous cognitive load and can be reduced through better instructional design (Sweller, 2020). The target audience for this study was programming novices, meaning they often lack a robust conceptual understanding and can easily become cognitively overloaded. Therefore, less-scaffolded SE prompts, such as those that are open-ended, could lead students to spend excessive time generating a self-explanation that is not correct or complete (Lehtinen et al., 2021). With this in mind, we only considered SE prompt types that provide at least a medium level of assistance (scaffolded and menu-based) for this work.

Parsons puzzles in intro-level CS education

Parsons puzzles are code construction exercises where students arrange mixed-up code blocks in the correct order to solve a problem (Parsons & Haden, 2006). Puzzles can vary based on the block types, order directions, and practice adaptivity. In a two-dimensional Parsons puzzle, blocks need to be sequenced vertically and appropriately indented horizontally (Ihantola & Karavirta, 2011). Adaptive Parsons puzzles can dynamically make the puzzle easier by either removing distractor blocks or combining blocks (Ericson et al., 2019). Distractor blocks are unnecessary code blocks with syntax or semantic errors.

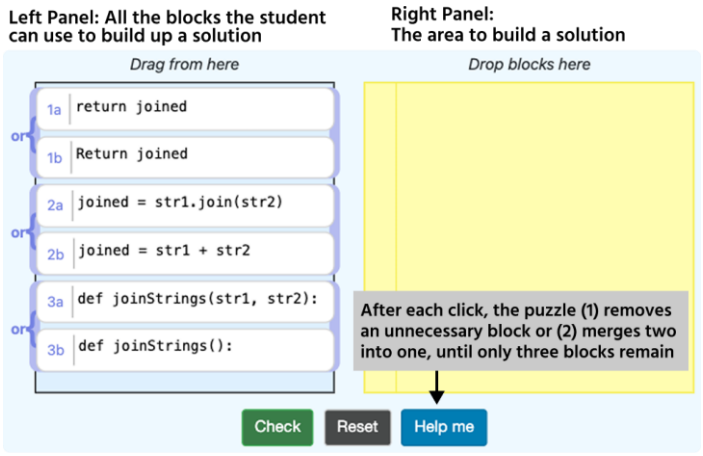
Parsons puzzles are an appropriate activity type to explore pairing with retrospective SE questions in intro CS courses. Working with Parsons puzzles provides a structured way for students to gain problem-solving experience without getting overwhelmed by syntax details (Parsons & Haden, 2006). In adaptive Parsons puzzles, students can click "Help me" to remove an unnecessary block or merge two blocks, until three blocks remain (Ericson et al., 2019). While Parsons puzzles offer structural support, they do not guarantee that students understand the solution (Denny et al., 2008). However, code comprehension is a fundamental programming skill that often precedes and supports the development of code-writing abilities (Fowler et al., 2022). This has taken on greater importance in AI-era CS education, serving as a prerequisite to evaluating AI-generated code. Therefore, a retrospective SE prompt can complement a puzzle-solving by encouraging students to cognitively engage in explaining the solution. Retrospective here means the prompt was given after engaging with the learning materials (Bisra et al., 2018). This bridges the gap between doing and understanding, helping novices' code comprehension through deliberate practice.

The existing use of SE prompts in intro-level CS education

In intro-level CS education, SE prompts have been used to engage students with worked examples (Vieira et al., 2019) and instructional videos (Chiarelli et al., 2023). In the context of problem-solving, SE prompts have been added to student submissions in program writing tasks (Lehtinen et al., 2021) and debugging tasks (Kwon et al., 2011), encouraging students to justify their own program submissions. For instance, Lehtinen et al. (2021) asked students to explain the structure and execution of their program after submitting their programs. Results showed that 1/3 of the students struggled to explain their code with such open-ended SE prompt formats.

Figure 1

Example of an adaptive Parsons puzzle (Step 1)



Regarding SE prompts with Parsons puzzles, one research group incorporated SE prompts with faded Parsons puzzles in a mobile learning environment (Fabic et al., 2017, 2019). They had students fill in incomplete code lines within Parsons blocks and then complete a SE prompt about their code entries. However, our work differs from theirs in the timing of SE prompts within the puzzles and the final output. Specifically, their SE prompts appeared midway through puzzle-solving, whereas ours appeared after each puzzle was solved, allowing students to focus without interruptions and reflect more comprehensively afterward. In addition, Price et al. (2020) had students complete a code-writing task, then provided structured prompts guiding them to explain an instructor solution by answering three multiple-choice questions. Our work builds on this but differs in three ways: (1) our SE prompts target Parsons puzzles, which are more beginner-friendly than writing tasks from scratch; (2) we provide immediate, elaborated feedback with hints to further scaffold the SE process (Aleven & Koedinger, 2002); and (3) we adapted the number of active areas to code complexity rather than using a fixed count. Together, these features make our SE prompt design novel and valuable for supporting code comprehension in intro-level CS education.

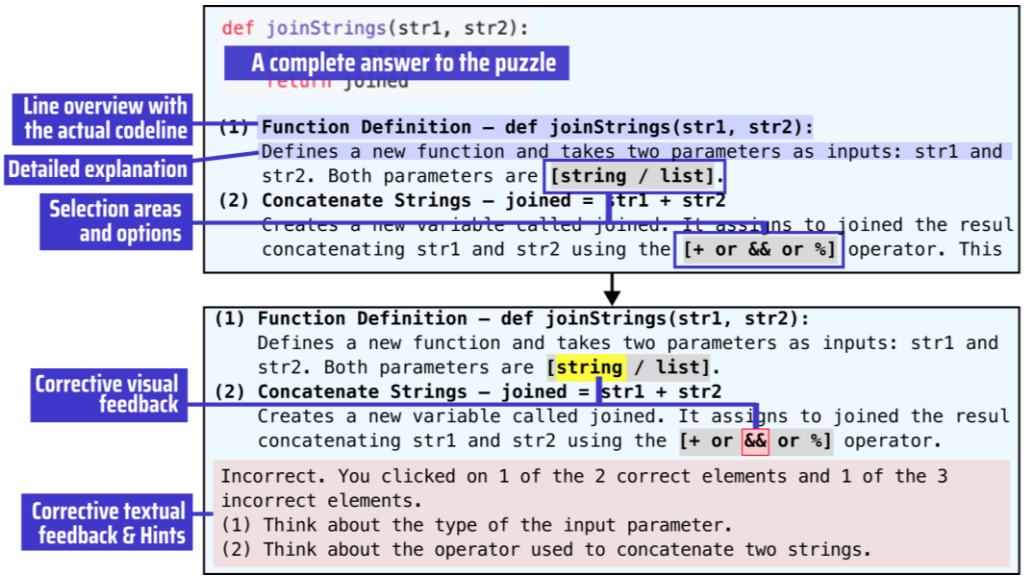
Design of two SE prompt types with Parsons puzzles: Select and Fill

Two types of SE prompts: Select (see Figure 2) and Fill (see Figure 3), were designed and deployed at an online CS learning platform. The two types were inspired by menu-based and scaffolded SE prompts in learning sciences. In Select, the student chooses from a set of choices (similar to menu-based SE prompts), and in Fill, the student must type in an answer (similar to scaffolded SE prompts). As retrospective SE prompts, both were designed to prompt students to explain the code program they had just completed in a Parsons puzzle. In this practice flow, students first attempted to solve a two-dimensional adaptive Parsons puzzle (see Figure 1); and then they worked on an SE activity (Step 2) that was either: Select or Fill.

Both Select and Fill share a three-layer structure. They begin with a complete puzzle solution at the top. Next, there is a code line summary, followed by a dash, and then the line of code. Below that, there is a detailed explanation of that line of code that may contain one or more SE areas. This structure aims to avoid the split-attention effect (Chandler & Sweller, 1992) and the associated unnecessary mental effort. Also, the line summary draws inspiration from subgoal labels in CS education (Margulieux & Catrambone, 2014). In a Select prompt (Figure 2), there are multiple clickable areas embedded throughout the detailed, line-by-line explanation. Students need to select the correct option from a list to complete an explanation. In a Fill prompt (Figure 3), students need to fill in one or more blanks to complete an explanation. Once students have answered a SE prompt, they can click the "Check me" button to receive immediate detailed feedback (see Figure 2 and Figure 3). In a Select prompt, the immediate feedback includes three parts: (1) the detailed corrective feedback; (2) visual highlights about the correct and incorrect selections; and (3) hints in natural language.

Figure 2

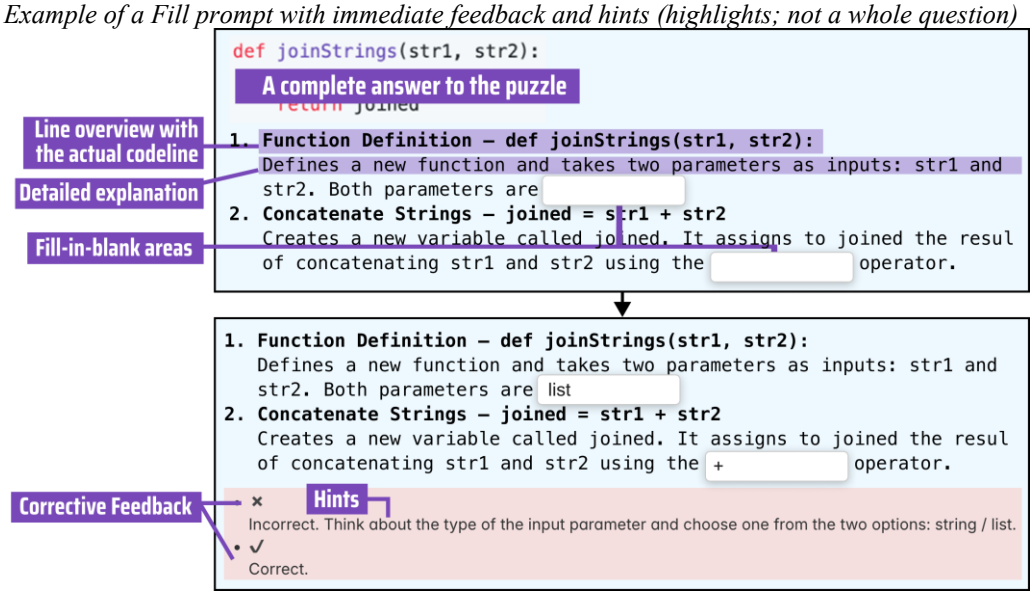
Example of a Select prompt with immediate feedback and hints (highlights; not a whole question)



In a Fill prompt, the feedback (Figure 3) also includes three parts: (1) the detailed corrective feedback for each blank; (2) hints in natural language; and (3) a list of option candidates to narrow down the answer space. To accommodate answer variation, each blank accepts multiple correct answers through regular expressions and a set of predefined synonyms.

Figure 3

Example of a Fill prompt with immediate feedback and hints (highlights; not a whole question)



In-depth Within-subjects Study: Student perceived values and preferences

In the new practice flow, participants first worked on a Parsons puzzle and then answered an SE prompt (Select or Fill). We conducted a within-subjects study with 10 novices to examine their perceptions of this flow and their attitudes toward the two SE prompt types.

Method

Participants and procedure

This study received Institutional Review Board (IRB) approval as exempt. As the study targeted novices with limited CS background, we distributed a recruitment announcement in Summer 2024 at a U.S. institution. We sought Python programming novices, such as students who had taken no more than two Python courses and had no additional Python experience beyond course-related activities. Studies were conducted remotely through Zoom (about 60-90 minutes). Practice sessions and follow-up interviews were recorded. Each participant received a $25/hour gift card upon

completing the study, with the amount depending on the session duration. After confirming eligibility and consent, participants received an ID and introductions. We asked participants to solve six Python Parsons puzzles. After solving each puzzle, students received one of the two types of paired SE prompts (puzzle+Select or puzzle+Fill). To reduce order effects (Pollatsek & Well, 1995), participants were counterbalanced across different versions. After practice, novices were interviewed about their experience and preferences. Interview questions included *"What did you consider when you decided on your preferred explanation question type?"*

Materials

The six Parsons puzzles focused on Python 3 basics (lists and loops). For the corresponding SE prompts, a researcher with Python expertise first used OpenAI's model GPT-4o to create line-by-line code explanations for each solution to a puzzle, serving as the basis of SE prompts. Then the researcher manually refined each explanation and created blanks for Fill and clickable areas for Select. These answer areas focus on the key concepts or flow control structures of a code program. For the two SE prompt types, the same clickable (or blank) locations were used, and a consistent list of potential answers as choices (or in the feedback) was provided. As the length of the code program increased, the number of blanks or clickable areas in the SE prompt increased (e.g., five for easy and seven for hard).

Data analysis

We first counted participants' quantifiable attitudes (whether they valued solving pairs of Parsons puzzles and SE prompts, and preferred SE type). To better understand the reasoning behind the preference, we conducted a thematic analysis of their transcripts to identify potential insights. Two researchers independently developed initial codes from materials, then collaboratively developed themes and co-coded one transcript to align understanding. They then independently coded remaining data, resolving disagreements through discussion.

Findings

RQ1: All novices reported that the post-puzzle SE prompts supported their code understanding, even after they had solved the puzzle.

We asked whether novices felt that the SE prompts helped them better understand the code, even after completing the puzzle independently. *All 10 novices gave a positive response (100%).* They agreed that the SE prompts amplified the instructional effectiveness of solving a puzzle. Engaging with a SE prompt helped them move from surface-level memorization to a deeper understanding of underlying principles, making it particularly valuable. In addition, P1, P8 and P9 specifically valued this *retrospective sequence of post-puzzle SE prompts: "I liked that I could kind of view through something I was already given ... And this took a second to make me think about why I was doing what I was doing."* (P1) and *"it reinforced the puzzle you just solved ... and having that reinforcement is much better than just having the puzzle solved."* (P8)

Eight participants highlighted how answering the SE prompt helped *clarify code details,* such as illustrating difficult points they did not fully understand: *"it just made it a lot more sense knowing what each step was doing in the greater context of the whole program."* (P5) Similarly, P1 highlighted how the SE activity drew their attention to details by comparing answer options: *"it helped me take into account little details about the puzzle itself, and then also take into account why I didn't do other things."* Three also indicated that it helped *prevent misunderstandings.* When going through the SE answers, P8 recalled their puzzle-solving process and identified a previous misunderstanding, *"it's not trying to find a 2 that is next to 2, it needs every 2 to be next to the 2. Okay, that's why I was misunderstanding what was being asked."* P6 also mentioned how this process tested their initial explanations based on prior knowledge, and helped them revise misconceptions: *" the actual explanation ended up being different. So it's super helpful to have that explained."* (P6)

RQ2: Select was strongly preferred, as Fill added unnecessary, learning-irrelevant mental effort.

For the question, *"Which type do you prefer to combine with the mixed-up puzzle in general?" 80% of the novices had a strong preference for the Select type.* A main factor influencing their preference was *the high efficiency in solving SE prompts.* Most said the predefined choices in Select questions speeded up the SE prompt answering process: *"I think I completed these a lot faster and more correctly, just because I knew ... what were the options for me."*(P4) Students demonstrated a streamlined process through the contextual options: *"the list ones (Select) ... you kind of read through them, you can piece together what the answer is."* (P5)

The two types also introduced *varying ambiguity*. For Select, five noted that it provided an additional layer of scaffolding, which prevented clueless guessing. One common challenge for novices is using appropriate programming language terminologies to articulate ideas (Lehtinen et al., 2021; Qian & Lehman, 2017). Fill demanded this skill directly, but Select provided support for this: *"my difficulty sometimes is translating that*

accurately in the way it's supposed to show up … I think it's easier for me to see the options, and then choose from the options that align with what I already know versus trying to generate it on my own." (P10)

In contrast, a primary concern with Fill was that the blanks led to *unnecessary guessing that was no longer relevant to the underlying knowledge.* P1 even described answering Fill questions as *"trying to read the mind of the test rather than actually knowing the content."* (P1) Also, although the auto-grading system included diverse synonyms, novices could come up with additional ones but got rejected. This added the burden of evaluating exact wording in Fill, which was learning-irrelevant. By contrast, with Select, as P2 said, *"I don't have to worry about accidentally putting in the wrong synonym."* (P2)

When ranking their preferred practice types for CS learning, two students expressed a strong dislike for the Fill type, ranking it even lower than the puzzle-only option. They saw *Fill as a distraction to learning,* promoting system-pleasing behavior instead of learning: *"I wasn't actually thinking about why something else was wrong, and I was more just thinking about what the question was looking for from me"* (P1) and *"it was more figuring out how to game the system … how do I give it what it wants, rather than how do I answer the question."* (P8) For the two who preferred Fill, a key feature that enabled their comfort was the list of options in the feedback for wrong answers, *"it was helpful to have the different options listed once I got it wrong, and once I tried for myself."* (P6) Because of this, they valued the potential for deeper learning: *"it allows you to engage more critically with the concepts."* (P9)

Classroom between-subjects study: Student learning

We then compared learning outcomes in a between-subjects study in an undergraduate intro-level CS class. As revealed in the above section, the current Fill design may have introduced high extraneous cognitive load, potentially burdening students with unnecessary mental effort irrelevant to learning (Sweller, 2010). Fill was also reported to cause high frustration and potentially hindered students' ability to focus on core content (Van Merrienboer & Ayres, 2005). Therefore, to prevent disengagement with SE questions because of frustration or cognitive overload, this classroom study only included Select. The two conditions were: solving a puzzle with a Select SE prompt (PZ+SE) versus just solving the puzzle (PZ).

Methods

Participants and procedure

This study received Institutional Review Board (IRB) approval as exempt. To align with the regular course schedule, the study was conducted as an in-class activity at the same institution in Winter 2025, covering the same topic. Before the study, students were informed that the activity was part of a research study, that usage data might be collected for research purposes, and that they were not required to complete the study materials. Students were provided with researcher contact information for questions or withdrawal requests. In this activity, students completed an introduction to the practice types and a pre-survey, followed by a 10-minute pretest. They were then randomly assigned to one of two practice conditions: PZ+SE (Parsons puzzles with a Select prompt) or PZ (Parsons puzzles only). Each student completed four practice units: four Parsons puzzles in the PZ condition, or four Parsons puzzles paired with Select SE prompts in the PZ+SE condition.

After each unit, students rated their perceived in-practice mental effort. We used a 9-point Likert scale adapted from Paas (1992), the same one used in Haynes & Ericson (2021). We chose this for its conciseness, which was more appropriate for our case (students had to fill out the scale four times). For each practice unit, PZ+SE students rated the mental effort required for both puzzle-solving and SE tasks as: "*During the process of solving the puzzle and (if applicable) completing the explanation question, I invested"*, while PZ students rated only about puzzle-solving: *"During the process of solving the puzzle, I invested"*. After the practice session, students completed a 10-min posttest with isomorphic questions. In the end, a total of 110 students completed the materials as intended (54 in PZ+SE and 56 in PZ).

Materials and grading

The practice included four adaptive Parsons puzzles. The course team validated them for proper difficulty. The pretest and the isomorphic posttest (a total of 22 points per test) included four open-ended short-answer questions (two examples are in Figure 4) and one Parsons puzzle. Test puzzles included the same key elements as practice puzzles, but were not adaptive. Students did not receive any feedback on short-answer attempts. For each incorrect puzzle attempt, students received feedback that highlighted incorrect blocks. Parsons puzzles were automatically graded using the system's percent-based partial scoring. Two graders manually scored open-ended questions. One

grader evaluated all answers and created an initial rubric, then both independently graded about 10% of answers before meeting to refine it. To verify the refined rubric, both graders independently re-graded the same responses, achieving an inter-rater reliability of 0.72, indicating substantial agreement (McHugh, 2012), using Cohen's Kappa (Cohen, 1960). One grader then scored all remaining responses using the refined rubric.

Figure 4

Example of two open-ended short-answer test questions

```
def count_valid_pairs(nums, threshold):
    """
    The first parameter is a list of numbers,
    and the second parameter is a single number.
    """
    count = 0
    for i in range(len(nums) - 1):
        if nums[i] > nums[i + 1]:
            continue
        count += 1
        if count > threshold:
            return True
    return False
```

Q1: What is the purpose of using **len(nums) - 1** in the for loop of the above function? Explain why it is necessary to use this specific range.

Q2: What does the condition **if nums[i] > nums[i + 1]:** check for?

Results

We first checked the pretest scores to see if the two groups were comparable. As the data were normally distributed, we applied a t-test. The result showed no significant condition difference on pretest performance, $t(108) = 1.31$, $p = .194$ (Table 2). Next, we answered RQ3 and RQ4.

RQ3: PZ+SE students had significantly higher learning gains from pre- to posttest than PZ students.

Learning gain (*posttest - pretest*) is used to report improvement while accounting for prior knowledge differences. Students with perfect pretests were evaluated on posttest performance only (5 students). Since the practice time was considered a common factor in SE studies, we added it as a covariate. Practice time per student was the sum of time spent on each practice question, where each question's time was computed by summing consecutive interaction intervals within that question, excluding gaps exceeding five minutes as idle or off-task time. We conducted a one-way ANCOVA test, and it indicated a significant condition difference on learning gain, $F(1, 102) = 4.55$, $p = .035$, partial $\eta^2 = .04$ (a small to medium effect size (Cohen, 2013)), between PZ+SE and PZ students, while controlling for practice time. For students who achieved perfect pretests (2 PZ+SE students and 3 PZ students), we compared posttest performance by condition, controlling for practice time, and found no difference between conditions, $F(1, 2) = 0.06$, $p = .831$.

Table 2

Test Performance Statistics by Condition (M SD)

Condition	Pretest (total 22 points)	Posttest (total 22 points)
PZ+SE	$M = 9.99$ $SD = 5.69$	$M = 15.27$ $SD = 4.81$
PZ	$M = 11.43$ $SD = 5.84$	$M = 14.26$ $SD = 5.60$

RQ4: PZ+SE and PZ students showed no significant difference in perceived in-practice mental effort.

We compared students' average perceived in-practice mental effort between conditions. We conducted a one-way ANCOVA test with practice time as covariate, and no significant difference between conditions was found in perceived in-practice mental effort, $F(1, 107) < 0.001$, $p = .992$. This indicated that PZ students ($M = 5.4$, $Mdn = 5.0$, $SD = 1.3$) and PZ+SE students ($M = 5.5$, $Mdn = 5.6$, $SD = 1.5$) experienced similar levels of perceived in-practice mental effort, even though PZ+SE group received paired SE prompts after code puzzles.

Discussion

This work examined whether a retrospective SE prompt after a Parsons puzzle supports novices' code understanding. Consistent with prior learning sciences evidence, all novices found the SE prompt valuable for their understanding (*RQ1*), echoing findings that novices often lack full comprehension of programs they produce (Lehtinen et al., 2021). This work also examined novices' preferences for two SE designs, Select and Fill, as well as the underlying reasons for those preferences (*RQ2*). Students who aimed to avoid random guessing and prioritize efficiency preferred Select, aligning with the assistance dilemma where greater assistance enables more efficient access to correct information and less frustration (Koedinger & Aleven, 2007). Since programming is complex and cognitively demanding for novices, more help in SE prompts was favored. The current Fill design

might lead students to consistently report excessive, learning-irrelevant mental effort (Sweller, 2011), as reported in Study 1. One likely explanation is that the current immediate feedback mechanism could not accommodate such a relatively open-ended solution space, which resulted in false negative cases in feedback. This burden conflicted with our goal of keeping the SE prompts lightweight and learning focused. As for future work, it is worth investigating whether real-time AI feedback (Dey et al., 2025) can better support the Fill design. In addition, based on the idea of faded scaffolding (Weinman et al., 2021), future research should also examine conditions under which Fill benefits learning more than Select.

Classroom results showed that adding Select SE prompts to Parsons puzzles improved learning gains over puzzles alone (*RQ3*). A paired retrospective SE prompt provides deliberate practice for code comprehension, requiring students to apply existing knowledge and connect it with new concepts (Chi & Wylie, 2014). In our study, students reported reflecting on their understanding when comparing distractor options with the correct answer, which likely led to greater cognitive engagement as they monitored their learning. Upon finishing the SE prompt, students can also go back through the explanation to revise their knowledge and fix misconceptions. Therefore, students gained more through practicing Parsons puzzles with SE prompts. One concern is that extra prompted SE questions may increase mental effort. However, students rated in-practice mental effort similarly for the puzzle alone and puzzle with the SE prompt (*RQ4*). Since students had already invested mental effort when trying to solve the puzzle, paired SE prompts based on the same question likely add minimal additional effort. Also, our question design keeps related information spatially integrated (code and explanations) (Renkl, 2002), limiting extra mental effort.

Limitations and future work

This work has several limitations. (1) The studies were limited to small samples from a single institution, reducing their generalizability. Future work should explore whether this practice is also effective in other programming languages and student groups. (2) When understanding students' preferences, this work only compares two formats. Others could also be explored. (3) This practice flow was only tested as in-class practice. Future research can look into its effect in other learning contexts, such as homework, exam review, and even as an exam question. (4) Only one level of SE prompt was investigated in this work: a step-by-step explanation with an expert-decided number of incomplete areas. To address specific misconceptions, future work can personalize the prompted SE activities based on students' knowledge level and problem-solving situations. For example, SE prompts can be further personalized by the granularity of the explanations, number of incomplete areas, and locations, as well as the choices in each SE area.

Conclusion

This work explores pairing Parsons code puzzles with self-explanation (SE) prompts to foster novices' code understanding beyond puzzle solving alone. In a within-subjects study, all novices valued the SE prompts even when they had solved the puzzle. Most students preferred Select due to high efficiency and more scaffolding compared to Fill. However, the Fill design caused frustration through unnecessary mental effort that failed to support learning. A between-subjects study showed higher learning gains for students who completed pairs of Parsons puzzles and Select SEs versus puzzles alone. Both conditions reported similar in-practice mental effort, meaning the paired SE prompts did not require additional mental effort. In response to the call for strengthening code understanding in AI-era CS education, we explored various designs and provided insights from both student perceptions as well as learning evidence. Our findings show how pairing Parsons puzzles with SE prompts can dedicate time to code understanding practice alongside problem-solving in early CS education.

References

Aleven, V., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor. *Cognitive Science, 26*(2), 147–179.

Bisra, K., Liu, Q., Nesbit, J. C., Salimi, F., & Winne, P. H. (2018). Inducing self-explanation: A meta-analysis. *Educational Psychology Review, 30,* 703–725.

Chandler, P., & Sweller, J. (1992). The split-attention effect as a factor in the design of instruction. *The British Journal of Educational Psychology, 62*(2), 233–246.

Chiarelli, V., Markova, N., & Muldner, K. (2023). Evaluating the Utility of Notional Machine Representations to Help Novices Learn to Code Trace. *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*, 314–328.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13*(2), 145–182.

Chi, M. T. H., & Wylie, R. (2014). The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational Psychologist, 49*(4), 219–243.

Chiu, J. L., & Chi, M. T. H. (2014). Supporting self-explanation in the classroom. *Applying Science of Learning in Education: Infusing Psychological Science into the Curriculum*, 91–103.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. routledge.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. *Proceedings of the Fourth International Workshop on Computing Education Research*, 113–124.

Dey, I., Gnesdilow, D., Smith, R., Kim, C., Passonneau, R. J., & Puntambekar, S. (2025). Factors Influencing Students' Perceptions of Automated Feedback and Their Impact on Revision. *Proceedings of the 19th International Conference of the Learning Sciences-ICLS 202*5.

Ericson, B. J., McCall, A., & Cunningham, K. (2019). Investigating the Affect and Effect of Adaptive Parsons Problems. *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*.

Fabic, G. V. F., Mitrovic, A., & Neshatian, K. (2017). Learning with Engaging Activities via a Mobile Python Tutor. In *International Conference on Artificial Intelligence in Education*.

Fabic, G. V. F., Mitrovic, A., & Neshatian, K. (2019). Evaluation of Parsons problems with menu-based self-explanation prompts in a mobile python tutor. *International Journal of Artificial Intelligence in Education, 29,* 507–535.

Fowler, M., Smith, D. H., IV, Hassan, M., Poulsen, S., West, M., & Zilles, C. (2022). Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education, 32*(3), 355–383.

Haynes, C. C., & Ericson, B. J. (2021). Problem-solving efficiency and cognitive load for adaptive parsons problems vs. writing the equivalent code. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.*

Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education Innovations in Practice, 10*, 119.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., & Mirolo, C. (2019). Fostering program comprehension in novice programmers-learning activities and learning trajectories. *In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 27–52).

Joo, H., Lee, J., & Kim, D. (2020). Advancing the design of self-explanation prompts for complex problem-solving. *International Journal of Learning Teaching and Educational Research, 19*(11).

Koedinger, K. R., & Aleven, V. (2007). Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psychology Review, 19*, 239–264.

Kwon, K., Kumalasari, C. D., & Howland, J. L. (2011). Self-Explanation Prompts on Problem-Solving Performance in an Interactive Learning Environment. *Journal of Interactive Online Learning, 10*(2).

Lehtinen, T., Lukkarinen, A., & Haaranen, L. (2021). Students Struggle to Explain Their Own Program Code. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 206–212.

Mahon, J., Mac Namee, B., & Becker, B. A. (2024). Guidelines for the evolving role of generative AI in introductory programming based on emerging practice. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*.

Margulieux, L., & Catrambone, R. (2014). Improving programming instruction with subgoal labeled instructional text. *Proceedings of the Annual Meeting of the Cognitive Science Society, 36.*

McCarthy, K. S. (2025, June 10). Toward bridging the gap: Undergraduate perspectives on implementing self-explanation as a reading strategy. *Proceedings of the 19th International Conference of the Learning Sciences-ICLS 2025*.

McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia Medica, 22(3)*, 276–282.

Paas, F. G. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: a cognitive-load approach. *Journal of Educational Psychology, 84*(4), 429.

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, 157–163.

Pollatsek, A., & Well, A. D. (1995). On the use of counterbalanced designs in cognitive research: a suggestion for a better and more powerful analysis. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 21*(3), 785.

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

Price, T. W., Williams, J. J., Solyst, J., & Marwan, S. (2020). Engaging students with instructor solutions in online programming homework. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1–24.

Renkl, A. (2002). Worked-out examples: Instructional explanations support learning by self-explanations. *Learning and Instruction, 12*(5), 529–556.

Robins, A. V. (2019). Novice programmers and introductory programming. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge handbook of computing education research*. Cambridge University Press.

Su, M., Bonaventura, K., & Nagashima, T. (2025). Fostering Strategic Choice-Making Through Motivational Pedagogical Agents in an Adaptive Learning System. *Proceedings of the 19th International Conference of the Learning Sciences-ICLS 2025.*

Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational Psychology Review, 22*, 123–138.

Sweller, J. (2011). Cognitive load theory. In *Psychology of learning and motivation* (Vol. 55, pp. 37–76). Elsevier.

Sweller, J. (2020). Cognitive load theory and educational technology. *Educational Technology Research and Development: ETR & D, 68*(1), 1–16.

Van Merrienboer, J. J. G., & Ayres, P. (2005). Research on cognitive load theory and its design implications for e-learning. *Educational Technology Research and Development, 53*(3), 5–13.

Vieira, C., Magana, A. J., Roy, A., & Falk, M. L. (2019). Student explanations in the context of computational science and engineering education. *Cognition and Instruction, 37*(2), 201–231.

Weinman, N., Fox, A., & Hearst, M. A. (2021). Improving instruction of programming patterns with faded parsons problems. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.*

Wylie, R., & Chi, M. T. H. (2014). 17 the self-explanation principle in multimedia learning. *The Cambridge Handbook of Multimedia Learning*, 413–432.

Zhang, Q., & Fiorella, L. (2024). Effects of self-explaining feedback on learning from problem-solving errors. *Contemporary Educational Psychology, 79*, 102326.

Acknowledgements

The funding came from the National Science Foundation award 2143028. Any conclusions expressed in this material do not necessarily reflect the views of NSF. Generative AI assisted with brainstorming throughout the process and with manuscript refinement for clarity, spelling, and grammar, and all content generated was reviewed, revised if needed, and verified by the authors.