Learner and Instructor Needs in AI-Supported Programming Learning Tools: Design Implications for Features and Adaptive Control

Zihan Wu, Yicheng Tang, and Barbara J. Ericson

University of Michigan {ziwu,tangyc,barbarer}@umich.edu

Abstract. AI-supported tools can help learners overcome challenges in programming education by providing adaptive assistance. However, existing research often focuses on individual tools rather than deriving broader design recommendations. A key challenge in designing these systems is balancing learner control with system-driven guidance. To explore user preferences for AI-supported programming learning tools, we conducted a participatory design study with 15 undergraduate novice programmers and 10 instructors to gather insights on their desired help features and control preferences, as well as a follow-up survey with 172 introductory programming students.

Our qualitative findings show that learners prefer help that is encouraging, incorporates visual aids, and includes peer-related insights, whereas instructors prioritize scaffolding that reflects learners' progress and reinforces best practices. Both groups favor shared control, though learners generally prefer more autonomy, while instructors lean toward greater system guidance to prevent cognitive overload. Additionally, our interviews revealed individual differences in control preferences.

Based on our findings, we propose design guidelines for AI-supported programming tools, particularly regarding user-centered help features and adaptive control mechanisms. Our work contributes to the human-centered design of AI-supported learning environments by informing the development of systems that effectively balance autonomy and guidance, enhancing AI-supported educational tools for programming and beyond.

1 Introduction

Programming is an important skill, but learning to program is not easy [25]. To address this problem, researchers have designed various tools to provide programming help such as intelligent tutors [12], programming games [22], and practice environments with adaptive hints and explanations [14, 27]. AI-supported tools, in particular, have shown promise in offering personalized guidance and adaptive feedback [9, 17, 20, 30]. However, most prior research has focused on designing and evaluating individual tools rather than exploring broader design guidelines regarding the needs and expectations of novice learners and instructors. In this work, we focus our research on AI-supported learning tools that provide on-demand help when learners are writing code.

To assist future work in better designing learner-centered AI-supported programming learning tools, we investigate the desired **help features**, referring

to the abstract qualities of help that users find beneficial. Our first research question asks: **RQ1: What *help features* do novice learners and instructors desire from an adaptive learning tool that provides on-demand programming help?**

A key challenge in AI-supported adaptive learning systems is determining how much decision-making power and responsibility should be given to the learner versus the system [15]. If the system has full control, learners might feel frustrated when the system's intervention does not align with their expectations; if the learners have full control, they might be overwhelmed [15, 26]. While some prior work has investigated the impact of control on learners [11, 34], little research has explored how to balance control in the context of programming education, particularly from the perspectives of both learners and instructors. In this paper, we refer to this shared control dynamic as **learner-system control**.

AI-supported learning systems can provide adaptive assistance in many dimensions, including: (1) the *types of help* provided (e.g., hints, worked examples, and visualizations), and (2) the *level of help*, which refers to the amount of assistance provided (e.g., high-level subgoal labels versus a detailed breakdown of sub-subgoals). Unlike the abstract **help features** described in RQ1, we use **types of help** to refer to the concrete ways of providing scaffolding. Despite the growing number of AI-supported programming tools [20], little research has explored how to balance learner-system control across various types and levels of help. To address this gap, we propose the second research question: **RQ2: What are novice learners' and instructors' preferences for *learner-system control*?**

To answer these questions, we conducted a participatory design study with both undergraduate novice learners and programming instructors. We created realistic programming practice scenarios for learners and instructional scenarios for instructors, asked them to design features for an "ideal smart programming help" tool to highlight the desired **help features**, and interviewed them regarding their preferred **learner-system control** mechanisms. Based on the findings from the first two studies, we administered a survey on the abstract help features and learner-system control mechanisms and analyzed responses from 172 undergraduate students. Finally, we present our findings and present design guidelines for AI-supported programming practice tools, especially for the learner-system control mechanisms.

2 Related Work

Learner-System Control Dynamics. A key challenge in adaptive learning is determining who controls the adaptation process: does the system make all decisions based on a student model, or does the learner control their learning experience [15]? The former characterizes an *adaptive* system, while the latter characterizes an *adaptable* system [24]. AI-supported learning systems can be placed on a spectrum between these extremes, depending on the balance of control between the system and its users [15, 26]. A fully adaptive system can be problematic because its user model may misrepresent the learner [7], resulting in mismatches between system decisions and learner expectations and causing

frustration [15]. Conversely, fully adaptable systems can overwhelm learners at times, particularly novices, by providing too many choices [15].

The AIED community has studied 'learner control,' often with a focus on self-regulation and motivation [33]. Empirical studies have examined how learner control influences learning outcomes and behavior [5, 11, 34]. Corbalan et al.[11] found that students who shared control with the system engaged more deeply in tasks than those in a system-controlled condition; Xie et al.[34] demonstrated that granting students agency in a programming context increased engagement; Recent research in intelligent tutoring systems (ITS) [5] indicates that learners highly value control as a means to enhance their learning experience and self-regulation. While prior work offers valuable insights into control dynamics, these studies are often system-specific and have rarely investigated the *preferences* of both learners and instructors. To fill this gap, our work is one of the first to directly study learners' and instructors' preferences using a design-focused approach, deriving guidelines for AIED systems in general.

Recent advances in AI, particularly in large language models, have enabled more personalized learning support [1, 3, 35]. However, as Brusilovsky [8] pointed out, despite growing interest in human-AI collaboration, "*the field of AIED is now lagging behind the work on user control in 'big AI'*". To emphasize the *collaborative* decision-making process between learners and AI-supported systems, we refer to these evolving control dynamics as **learner-system control**.

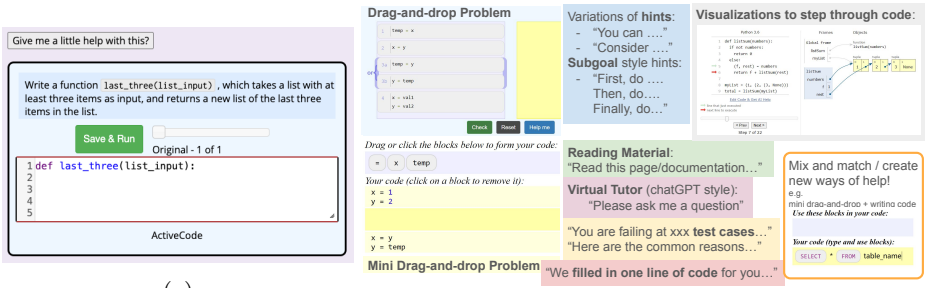
Participatory Design in Educational Technology Participatory design (PD) comprises a set of human-centered methods that actively involve stakeholders in the design process [13]. Engaging instructors leverages their domain expertise in teaching; however, involving learners is equally crucial [18]. PD methods have been used to create educational technologies in many fields, including math, history, and social skills [4, 6, 16, 23, 28]. In computing education, rather than focusing on designing tools and systems for learners, PD methods have primarily been used to develop curricula for specific populations, such as to foster computational thinking skills for K-12 learners [29] and to create culturally relevant curriculum [10]. Other studies have directly integrated PD as learning activities for students [2, 21]. Our work is among the first to employ PD methods that directly engage both instructors and learners in the design process of AI-supported programming learning tools.

3 Participatory Design with Learners and Instructors

We conducted participatory design (PD) sessions with 15 undergraduate novice programmers and 10 instructors for both undergraduate and graduate programming courses. All studies were conducted virtually via Zoom, received approval from the local institutional review board, and obtained participant consent.

PD with Learners We recruited 15 undergraduate participants who had either recently completed an introductory programming course or were enrolled to take it in the upcoming semester. Prior to the PD sessions, participants filled out a demographic survey that included a standard self-efficacy survey for computing [32], consisting of five questions rated on 7-point Likert scales (with 1 representing the lowest and 7 the highest).

We established a context designed to elicit participants' help-seeking behaviors. First, learners were asked to attempt one or two introductory Python programming problems using a web interface (Fig. 1a) that allowed them to type and execute their code, view error messages from the Python interpreter, and receive unit test results. The interface also featured a help button labeled "Give me a little help with this?" that was non-functional. Participants were instructed to click the button when they desired assistance from the programming tool, which was designed to deliver the "ideal" help. When they clicked the button, the researcher would: (1) inquire about the reason for the learner's help request, (2) provide tutoring to assist in resolving the current issue, and (3) ask the learner to retrospectively describe the "ideal" help they would have preferred from the tool in the absence of a human tutor. During the design brainstorming process, the researcher presented learners with common methods of providing help in existing programming practice tools (fig. 1b).



(a) (b)

Fig. 1: (a): An example problem in the interface, with a help button on top. (b): Examples of different types of help, provided as inspirations for PD sessions.

Then, the researcher interviewed the participants for their desired **learner-system control** model for the programming tool. The participants used an interface design tool to help articulate their desired design. Learners were asked to consider two dimensions of the control model: (1) the fading of scaffolding as they become more experienced (*levels of help*), and (2) managing their preferences for the different types of help available (*types of help*).

PD with instructors We recruited 10 programming course instructors, including both faculty members and graduate student instructors. We compiled learners' help-seeking events from the PD sessions into a document (screenshots of learners' code at the moment they pressed the help button). For each event, instructors were asked to: (1) describe what the student was most likely to need help with, and (2) design the ideal support that the tool could provide. Instructors were also provided with help examples (fig. 1b) as design inspirations. Finally, the researcher posed the same interview questions regarding the instructors' desired learner-system control model for the programming tool, focusing on both the *types of help* and the *levels of help*.

Data Analysis We collected screen and audio recordings for each session. After anonymizing and transcribing the recordings, we conducted thematic analy-

sis [31] on the transcripts, with the help of screen recordings to understand the context. We developed one codebook for learners and another for instructors. One researcher, responsible for all observation studies, performed an initial open coding to develop both codebooks. Subsequently, two researchers independently coded a subset of the transcripts (two from learners and one from instructors) using the respective codebooks. The researchers then discussed any discrepancies and refined the codebooks accordingly. Finally, the two researchers independently coded two additional transcripts from both learners and instructors. The two researchers reached a Krippendorf's alpha above the recommended inter-rater agreement threshold of $\alpha > 0.80$ [19] ($\alpha_{learner} = 0.82$, $\alpha_{instructor} = 0.81$).

4 Findings from Participatory Design Sessions

4.1 Findings from Learners

Fifteen undergraduate participants (L01-L15) participated in the study. Eight participants were information majors, six were computer science majors, and one was a business administration major. Thirteen participants had just completed their sophomore year, and two had just completed their freshman year. Nine participants identified as male, five as female, and one as non-binary. The participants' average self-reported self-efficacy was 5.03 (SD = 1.94) on a 7-point scale, where 1 indicates the lowest and 7 the highest level.

Help-Seeking Events. On average, participants requested help 1.2 times per problem, and their reasons for seeking help were categorized into three codes.

When seeking an answer to a specific question. Eight (53%) learners sought help when they had a specific question, such as requesting clarification on the problem or an explanation for a syntax error. Learners did not hesitate long before pressing "help" in this case, knowing that they should receive the *ideal* help from the tool in the study. However, they worried that in a real setting, the tool might fail to accurately recognize their specific questions, potentially providing irrelevant or incorrect information.

When they attempted to implement a solution, encountered a problem, and were unable to resolve it despite believing they were on the right track. Nine (60%) learners were categorized under this reason. They attempted their ideas but encountered logical or syntactical problems that they failed to resolve, such as syntax errors or unexpected outputs. Unlike the previous case, they were not sure what they needed to resolve the issue. They generally delayed requesting help, because they wanted to continue making attempts: "*at least I wanna try some of those (ways I know) before I ask for help.*" (L02) Eventually, when minor details became too frustrating, they asked for help. They preferred not to receive a direct answer from the tool, so they could feel that their effort to resolve the problem paid off.

When unsure of how to begin and in need of guidance. Six (40%) learners asked for help when they were uncertain about how to start the problem. These learners were uncertain about the type of help required and had not yet attempted any solution. Although some learners had preliminary ideas, they lacked confidence or recognized potential flaws in their planned solutions. As a result,

they decided to request help rather than proceed independently. Learners wanted more help in this situation, expressing doubt about their ability to complete the problem, and wanted to make sure they were learning the desired way to code: "*Especially with getting started, I'd like to see how the textbook does it... it gives that sort of efficient logic, rather than sometimes my inefficient logic still works, but it's not always the best way.*" (L10)

Preferences for Help Features. Five themes emerged from the PD sessions.

1. *Provide feedback relevant to the student code* (n=15, 100%). Desired features include locating syntax or logic errors in their code, explaining failed test cases, and walking them through their code.
2. *Avoid providing the solution directly* (n=8, 53%). The learners did not want the tool to provide a solution directly: "*The help that I would be receiving would almost be meaningless... Because I'm not getting that conceptual understanding that I need.*" (L13)
3. *Provide visual components beyond plain text* (n=4, 27%). Learners referred to a variety of things as "visual", such as Parsons problems (i.e. rearranging mixed-up code blocks), debuggers, and code tracing tools. While some valued the graphics, others highlighted the need for visuals that effectively organize or chunck textual information.
4. *Be encouraging* (n=2, 13.4%). Learners noted that encouragement is not only for boosting motivation, but also helps prevent them from doubting themselves and starting over when their current code is on the right track.
5. *Provide information about and from peers* (n=2, 13.4%). Desired features included showing common misconceptions, offering alternative solutions from peers, and retrieving relevant discussions on Q&A platforms such as Piazza.

Preferences for learner-system control. We identified four distinct models of learner-system control. Table 1 outlines each model and the implementation for *types of help* and *levels of help*, and fig. 2 provides example interface designs for implementing the control models.

In our study, learners were categorized based on their preferences regarding learner-system control. **Three (20%) learners preferred the L model**, and were confident about their ability to choose the appropriate type and level of help. They had strong preferences towards the types and levels of help to receive, and wanted to manage their learning experience. **Eight (53%) learners fell into the L-S model**, wanting to be able to make the decisions, but would like to receive information provided by the system to assist with their decision. For instance, participants L02 and L09 favored having predefined 'go-to' help types, yet still wanted system suggestions for individual help events. **Two learners (13%) preferred the S-L model** and would like the system to make decisions, but were concerned that the system would fail to provide the types of help that they wanted. For example, L07 preferred having access to a menu offering access to all available help types, while L12 suggested a button to switch to an alternative help type if the provided one was unsatisfactory. They were particularly concerned that the system would take away too much help when they needed it, therefore making the problem too difficult for them. **Only one**

Table 1: Learner-system control models for *types of help* and *levels of help*

Model	Explanation	Controlling Types of Help	Controlling Levels of Help
L	**Learners have full control** and are not influenced by the AI system.	The learner selects from a wide range of help types and receives only the type they choose, without system recommendations.	The learner manually adjusts scaffolding, either increasing or decreasing the level of support as needed.
L-S	**learners have major control**, but the *AI system provides recommendations.*	The learner selects a preferred help type, while the system subtly suggests alternative types to assist decision-making.	The learner manually adjusts scaffolding while the system provides gentle suggestions based on the student model.
S-L	The **AI system has major control**, but *learners can override* system decisions.	The system recommends a help type based on learner progress, but the learner can override and choose a different type.	The system adjusts levels of scaffolding automatically, but learners can override these decisions when necessary.
S	The **AI system has full control**, and the learners receive help as determined by the system.	The learner receives the one help type recommended by the system. No learner-initiated adjustment is needed or supported.	The system fully controls scaffolding, automatically adjusting support based on learner progress, without learner overrides.

learner (6.7%) preferred the S model. They felt that if the system is smart enough, they should not need to expend time and effort deciding on the type or level of help.

Beyond the control models, several additional considerations emerged during the PD sessions. Some learners (n=2, 13%) worried they might lack self-discipline at times, ending up abusing their power to control the help provided to them. Additionally, some learners (n=4, 27%) mentioned that they did not want to make decisions for every help-seeking event, suggesting that the frequency of learner-initiated control should be moderated. Due to time constraints, we were unable to explore this issue in detail.

4.2 Findings from Instructors

Desired help features. Although many instructors offered suggestions similar to those of the students, two unique recommendations emerged.

1. Promoting good practices. (n=5, 50%). Instructors recommended that the tool emphasize best programming practices, e.g. following good variable naming conventions and using print statements for debugging.
2. Provide information on individual learning progress (n=2, 20%). Instructors proposed features that offer reminders of how learners previously resolved similar issues. They believe this could help learners feel more confident and encourage them to connect what they have learned in the past to what they are currently learning.

Learner-System control. The instructors' interview responses were coded using the control model derived from the student interviews. Most instructors (n = 6, 60%) preferred S-L (mostly AI system control, but learners can override settings), and three (30%) preferred L-S (mostly learner control, but the system provides recommendations). Although instructors valued learner control, they expressed concern that requiring learners to make too many decisions might

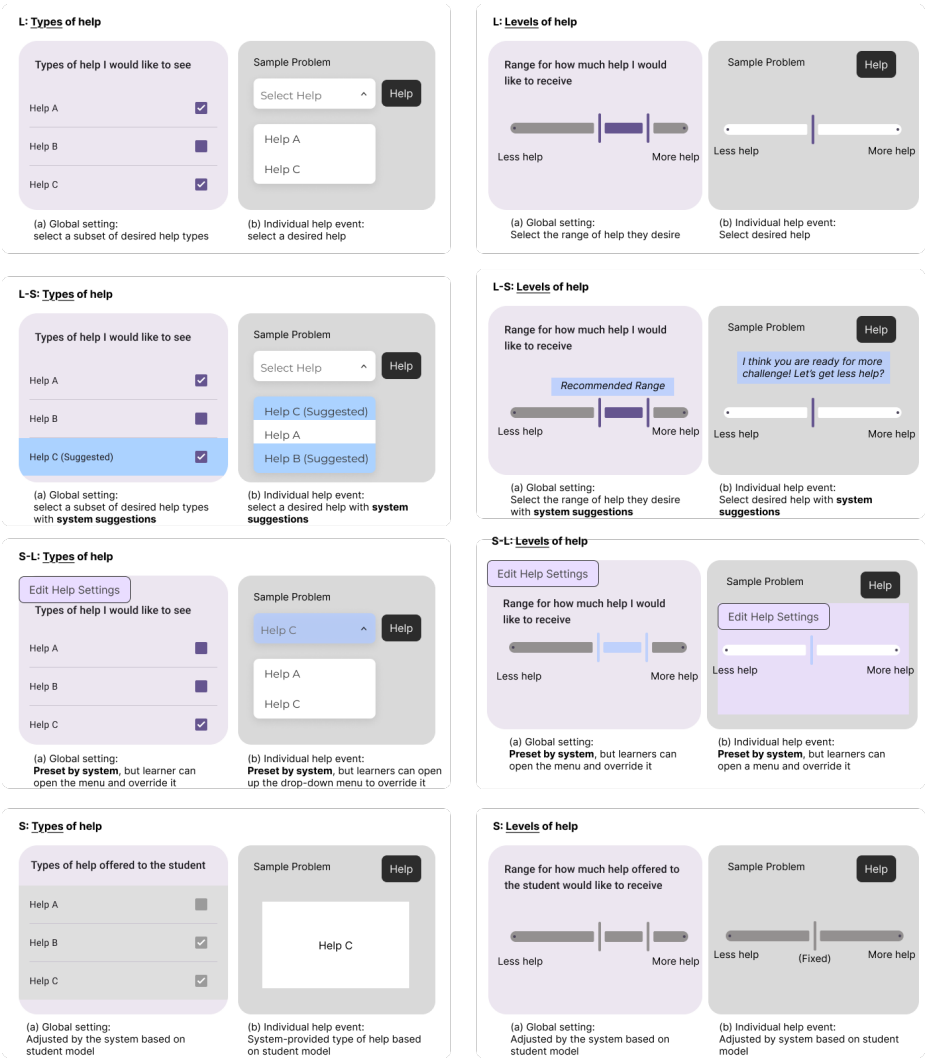


Fig. 2: Example UIs that implement different learner-system control models.

overwhelm them: "*It's important for students to have the ultimate control... [but] that's adding more stuff that they need to think about.*" (I10)

Interestingly, one participant (I08) proposed a "meta-control" approach, customizing to students' preferences for how much control they would like to have: "*I will say that the most important thing is to understand what students want. So if the tool understands that the students want to make choices, then just give them the choices... If the students don't like making choices, it makes choices for the student.*" (I08)

Additional considerations that are related to learner control were brought up by the instructors. For example, all instructors (n=10, 100%) mentioned that

they would need input from students to provide the appropriate help, especially for determining the type of help, and would value the ability to **check in with students**. Three (30%) instructors expressed that they were **skeptical about the accuracy of student models** in AI-supported learning tools. As a result, they believe that the learners should always have the final call on what help they receive, in case the provided system suggestion is incorrect. Three (30%) instructors mentioned their concerns that learners might "game the system" and abuse help, and suggested that the tool should be able to **adapt to learners' motivation in different contexts**. For example, I06 noted that for homework assignments, "*most students will like the help that can help them get the full solution as fast as possible*", whereas when preparing for exams, they are more inclined to use sophisticated features to enhance learning. Three (30%) instructors also mentioned that they would like to be able to **directly control the settings** for the tool based on what they observe in class or receive insights on how learners use the tool. Some wanted functionality that allows students to request help from instructors directly, such as shortcuts for students to send their current problem and incorrect solution to course communication channels like Slack and Piazza, or send them to the instructors directly during office hours.

5 Student survey

5.1 Methods

We distributed an optional survey to learners in an introductory programming course and received 172 responses. The survey consisted of questions for (1) self-efficacy, (2) perceived importance of help features identified from the PD sessions, and (3) preferred learner-system control models. The survey began with a standard self-efficacy survey for computing [32] with five questions on 7-point Likert scales from 1 (lowest) to 7 (highest). To reduce the length of survey, we extracted a total of five help features and created 5-point Likert scales for learners to rate the importance of each feature: (a)(Peer): Gives the learner information related to their peers, e.g., what other students' solutions are and what problems other students are facing; (b)(Encouraging): Be encouraging, e.g. celebrating their progress on a problem even when not yet passing all test cases; (c) (Challenging): Challenges the learners, not giving them the answer directly; (d)(Visual): Has visual components aside from text, e.g. blocks or visualizations; (e)(Progress) Gives the learner information about their learning progress, e.g. similar problems they have solved or struggled with before.

We asked learners to select their preferred learner-system control model in terms of (1) *type of help* and (2) *level of help* by describing the implementation of our proposed models in text.

5.2 Results

On a scale of 1 (low) to 7 (high), learners' average self-efficacy was 4.7 (SD=1.3).

Perceived importance of help features. Learners rated the perceived importance (1 - not important, 5 -essential) of the abstract help features. The average importance of providing peer information was 3.53 (SD=0.96), being encouraging was 3.56 (SD=1.18), being challenging was 3.78 (SD=0.99), having

visual components was 4.09 (SD=1.03), and providing information about their learning progress was 4.38 (SD=0.79) (fig. 3).

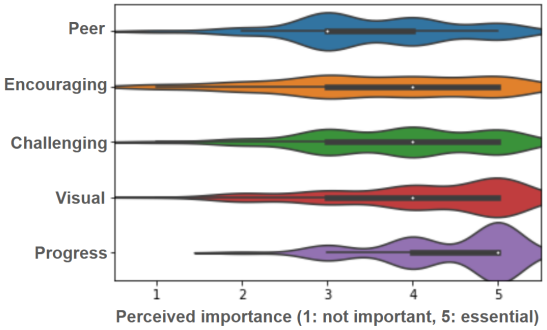


Fig. 3: Violin plot of learners' perceived importance of the features.

We calculated the Pearson correlation between each learner's self-efficacy and their perceived importance of each of the five features. Only the "challenging the learner" factor had a significant correlation with self-efficacy ($p<.001$), and the correlation was moderately positive ($r=0.36$), which indicates that this was more important to those with higher computing self-efficacy.

Preferred learner-system control model. For the preferred learner-system control for the **type of help**, 39 (23%) learners preferred L (learner-controlled), 92 (53%) preferred L-S (mainly controlled by the learner), 34 (20%) preferred S-L (mainly controlled by the AI system), and only 7 (4%) chose S (AI-system-controlled). The preferences for learner-system control models for the **level of help** were similar, where 38 (22%) learners preferred the L model, 85 (49.4%) preferred L-S, 43 (25%) preferred S-L, and only 6 (3.5%) chose S. By assigning numerical values to these different learner-system control models (L: 0, L-S: 1, S-L: 2, S: 3), we performed a Pearson correlation between learners' preferred model and their self-efficacy. At $\alpha = 0.01$ level, We found a significant correlation between preferences for learner-system control models for the **level of help** ($p = 0.008$), which was weak negative ($r=-0.20$). This indicate that learners with higher self-efficacy desire more control for the **level of help** they receive from the AI system. However, no statistically significant correlation was found between the preferred **type of help** with learners' self-efficacy.

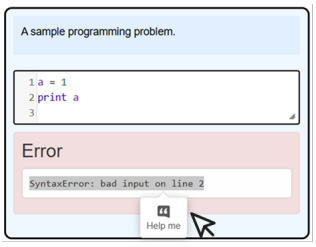
6 Discussion

In this section, we answer our research questions and offer design guidelines for designing help features and learner-system control mechanism for an "ideal" AI-supported learning tool, for programming learning and beyond.

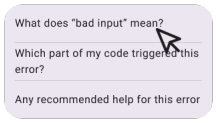
Desired help features. First, we provide design guidelines for **abstract features** that are desired from concrete help functionalities.

Guideline 1: Seek explicit input on why learners need help, and provide help based on learners' need. Our findings indicate that learners seek help from an AI-supported programming tool under three conditions: (a) when they have a specific question, (b) after unsuccessful attempts to resolve an issue, or (c) when they are unsure how to begin. These help-seeking reasons

reveal the type of information and assistance learners expect from the tool. When a specific question is in mind, learners anticipate a direct and precise answer. When confronting issues in their code, learners expected explanations that would guide them toward a resolution. If a learner is unsure of how to start, they expect the tool to provide clear initial guidance. However, even experienced instructors find it challenging to infer a learner's intent only from their interactions with the programming interface. Fig. 4 illustrates an example design that gathers explicit student input with minimal interaction overhead by (a) highlighting the area that is confusing for them or (b) asking learners to select an option that describes their current confusion.



(a) Learners can ask for help about the **error message**, their **code**, or **problem description** by selecting text in these areas to ask for help



(b) The tool provides a list of potential questions, and learners can select the questions student may have regarding the current help-seeking event. **The options can be especially helpful if there are many different misconceptions suggested by their current code.**

Fig. 4: An example implementation of Guideline 1 with low interaction overhead.

Guideline 2: Provide personalized feedback on a micro level and a macro level. At the micro level, the tool should offer feedback specific to the current problem, such as annotating a learner's solution to highlight errors or suggest improvements. At the macro level, the tool should address a learner's overall progress. For instance, it can suggest similar problems they have encountered previously and remind them how they resolved a similar issue before to help them connect the dots.

Guideline 3: Support diverse needs and provide space for customization. The perceived importance of many help features was very diverse. While some participants repeatedly emphasized the value of an encouraging tool, a considerable amount of learners rated it as less important. To accommodate these differing preferences and avoid frustrating or alienating users, AI-supported learning tools should offer customizable functionalities that adapt to individual needs.

Guideline 4: Engage instructors by providing data access, control, and involvement opportunities. Many instructors also mentioned their desire to be more engaged with the system, instead of making the practice process an independent interaction between learners and the system. The tool can provide analytical feedback to the instructors and allow instructors to customize the type of help available to learners. Additionally, the system could include shortcuts that allow learners to directly request assistance from instructors.

Learner-System Control We identified four preferred models of **learner-system control** (see table 1 and Fig. 2): full learner control (L), predominantly

learner control with system suggestions (L-S),predominantly system control with learner override (S-L), and full system control (S).

Guideline 5: Provide personalized learner-system dynamics. While the PD sessions suggested similar preferences for both the **type** and **level** of control, survey results revealed differences between them. Learners with higher self-efficacy tended to desire greater control over the level of help provided, while this pattern was not observed with the types of help. Although further investigation is warranted, it appears that preferences regarding the level of help may be more closely linked to cognitive factors such as self-efficacy and metacognitive skills, whereas preferences for the type of help may be driven more by individual taste.

Guideline 6: Create context-aware learner-system control mechanism. Both learners and instructors voiced concerns about potential 'gaming the system' behavior. They also noted that the context in which programming practice occurs can significantly influence help-seeking behavior, especially for the **levels of help**. For mandatory assignments where learners' main goal is to complete the problems, they are motivated to use help to reduce their time and effort. For practicing their problem-solving skills and enhance their knowledge, they often hesitate to request help, thinking that receiving help might negatively affects their learning. As a result, AI learning tools should be context-aware and adjust the range of control available for learners.

7 Limitations and Future Work

One limitation is that we only studied self-reported data, which means that our work focused more on improving learners' satisfaction and engagement instead of learning outcome. Learner participants in the PD sessions had a slightly higher average self-efficacy (mean=5.0) than the average responses from the survey (mean=4.7), indicating potential selection bias. In terms of study design, although we provided examples in the participatory design sessions to help speed up learners' brainstorming process, their choices or preferences during the session could be affected by these examples. While we provided a model to describe learner-system control dynamics, it is worth noting that there are factors not discussed in our proposed model, e.g. the desired frequency for learners to make control-related decisions, or the amount of information provided to learners regarding system decisions [34]. Future work can also explore more in using PD to create AI-supported learning tools. Aside from having individual researcher-participant design sessions, future work can explore workshop activities that allow groups of learners to design culturally or community-relevant tools. Larger workshops that engage learners and instructors at the same time can also provide an opportunity for different stakeholders to communicate their thoughts directly. Future work can also expand outside of programming education to understand learners' and instructors' preferences in math or language learning.

Bibliography

[1] Abd-Alrazaq, A., AlSaad, R., Alhuwail, D., Ahmed, A., Healy, P.M., Latifi, S., Aziz, S., Damseh, R., Alrazak, S.A., Sheikh, J., et al.: Large language models in medical education: opportunities, challenges, and future directions. JMIR Medical Education **9**(1), e48291 (2023)

[2] Agbo, F.J., Oyelere, S.S., Suhonen, J., Laine, T.H.: Co-design of mini games for learning computational thinking in an online environment. Education and information technologies **26**(5), 5815–5849 (2021)

[3] Bernacki, M.L., Greene, M.J., Lobczowski, N.G.: A systematic review of research on personalized learning: Personalized by whom, to what, how, and for what purpose (s)? Educational Psychology Review **33**(4), 1675–1715 (2021)

[4] Booker, A., Goldman, S.: Participatory design research as a practice for systemic repair: Doing hand-in-hand math research with families. Cognition and Instruction **34**(3), 222–235 (2016)

[5] Borchers, C., Ooge, J., Peng, C., Aleven, V.: How learner control and explainable learning analytics on skill mastery shape student desires to finish and avoid loss in tutored practice. To appear in LAK 2025 (2025)

[6] Borges, L.C., Araujo, M.R., Maciel, C., Nunes, E.P.: Participatory design for the development of inclusive educational technologies: A systematic review. In: 2016 IEEE Frontiers in Education Conference (FIE). pp. 1–9. IEEE (2016)

[7] Brusilovsky, P.: Methods and techniques of adaptive hypermedia. In: Adaptive hypertext and hypermedia, pp. 1–43. Springer (1998)

[8] Brusilovsky, P.: Ai in education, learner control, and human-ai collaboration. International Journal of Artificial Intelligence in Education **34**(1), 122–135 (2024)

[9] Caughey, M., Muldner, K.: Investigating the utility of self-explanation through translation activities with a code-tracing tutor. In: International Conference on Artificial Intelligence in Education. pp. 66–77. Springer (2023)

[10] Coenraad, M., Palmer, J., Eatinger, D., Weintrop, D., Franklin, D.: Using participatory design to integrate stakeholder voices in the creation of a culturally relevant computing curriculum. International Journal of Child-Computer Interaction **31**, 100353 (2022)

[11] Corbalan, G., Kester, L., van Merriënboer, J.J.: Selecting learning tasks: Effects of adaptation and shared control on learning efficiency and task involvement. Contemporary Educational Psychology **33**(4), 733–756 (2008). https://doi.org/https://doi.org/10.1016/j.cedpsych.2008.02.003, https://www.sciencedirect.com/science/article/pii/S0361476X08000118

[12] Crow, T., Luxton-Reilly, A., Wuensche, B.: Intelligent tutoring systems for programming education: a systematic review. In: Proceedings of the 20th Australasian Computing Education Conference. pp. 53–62 (2018)

[13] DiSalvo, B., Yip, J., Bonsignore, E., Carl, D.: Participatory design for learning. In: Participatory design for learning, pp. 3–6. Routledge (2017)

[14] Ericson, B.J., Pearce, J.L., Rodger, S.H., Csizmadia, A., Garcia, R., Gutierrez, F.J., Liaskos, K., Padiyath, A., Scott, M.J., Smith IV, D.H., et al.: Multi-institutional multi-national studies of parsons problems. In: Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education, pp. 57–107 (2023)

[15] Imhof, C., Bergamin, P., McGarrity, S.: Implementation of adaptive learning systems: Current state and potential. Online teaching and learning in higher education pp. 93–115 (2020)

[16] Ismail, R., Ibrahim, R., Yaacob, S.: Participatory design method to unfold educational game design issues: a systematic review of trends and outcome. In: 2019 5th International Conference on Information Management (ICIM). pp. 134–138. IEEE (2019)

[17] Kazemitabaar, M., Ye, R., Wang, X., Henley, A.Z., Denny, P., Craig, M., Grossman, T.: 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. pp. 1–20 (2024)

[18] Könings, K.D., Seidel, T., van Merriënboer, J.J.: Participatory design of learning environments: integrating perspectives of students, teachers, and designers. Instructional Science **42**, 1–9 (2014)

[19] Krippendorff, K.: Content analysis: An introduction to its methodology. Sage publications (2018)

[20] Le, N.T., Strickroth, S., Gross, S., Pinkwart, N.: A review of ai-supported tutoring approaches for learning programming. Advanced computational methods for knowledge engineering pp. 267–279 (2013)

[21] Limke, A., Lytle, N., Lin, M., Mahmoud, S., Hill, M., Cateté, V., Barnes, T.: Empowering students as leaders of co-design for block-based programming. In: Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems. pp. 1–7 (2023)

[22] Lindberg, R.S., Laine, T.H., Haaranen, L.: Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games. British journal of educational technology **50**(4), 1979–1995 (2019)

[23] Mack, N.A., Rembert, D.G.M., Cummings, R., Gilbert, J.E.: Co-designing an intelligent conversational history tutor with children. In: Proceedings of the 18th ACM International Conference on Interaction Design and Children. p. 482–487. IDC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3311927.3325336, https://doi.org/10.1145/3311927.3325336

[24] Miller, C.A., Funk, H., Goldman, R., Meisner, J., Wu, P.: Implications of adaptive vs. adaptable uis on decision making: Why "automated adaptiveness" is not always the right answer. In: Proceedings of the 1st international conference on augmented cognition. pp. 22–27 (2005)

[25] Milne, I., Rowe, G.: Difficulties in learning and teaching programming—views of students and tutors. Education and Information technologies **7**, 55–66 (2002)
[26] Nwana, H.S.: Intelligent tutoring systems: an overview. Artificial Intelligence Review **4**(4), 251–277 (1990)
[27] Parsons, D., Haden, P.: Parson's programming puzzles: a fun and effective learning tool for first programming courses. In: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. pp. 157–163 (2006)
[28] Pnevmatikos, D., Christodoulou, P., Fachantidis, N.: Stakeholders'involvement in participatory design approaches of learning environments: A systematic review of the literature. EDULEARN20 proceedings pp. 5543–5552 (2020)
[29] Sunday, A.O., Agbo, F.J., Suhonen, J.: Co-design pedagogy for computational thinking education in k-12: A systematic literature review. Technology, Knowledge and Learning pp. 1–56 (2024)
[30] Taneja, K., Maiti, P., Kakar, S., Guruprasad, P., Rao, S., Goel, A.K.: Jill watson: A virtual teaching assistant powered by chatgpt. In: International Conference on Artificial Intelligence in Education. pp. 324–337. Springer (2024)
[31] Terry, G., Hayfield, N., Clarke, V., Braun, V., et al.: Thematic analysis. The SAGE handbook of qualitative research in psychology **2**(17-37), 25 (2017)
[32] Wiggins, J.B., Grafsgaard, J.F., Boyer, K.E., Wiebe, E.N., Lester, J.C.: Do you think you can? the influence of student self-efficacy on the effectiveness of tutorial dialogue for computer science. International Journal of Artificial Intelligence in Education **27**, 130–153 (2017)
[33] Williams, M.D.: A comprehensive review of learner-control: The role of learner characteristics. (1993)
[34] Xie, B., Nelson, G.L., Akkaraju, H., Kwok, W., Ko, A.J.: The effect of informing agency in self-directed online learning environments. In: Proceedings of the Seventh ACM Conference on Learning @ Scale. p. 77–89. L@S '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3386527.3405928, https://doi.org/10.1145/3386527.3405928
[35] Yan, L., Sha, L., Zhao, L., Li, Y., Martinez-Maldonado, R., Chen, G., Li, X., Jin, Y., Gašević, D.: Practical and ethical challenges of large language models in education: A systematic scoping review. British Journal of Educational Technology **55**(1), 90–112 (2024)