Student-AI Interaction: A Case Study of CS1 students

MATIN AMOOZADEH, University of Houston, United States
DAYE NAM, Carnegie Mellon University, United States
DANIEL PROL, Universidad Internacional de la Rioja, Spain
ALI ALFAGEEH, University of Houston, United States
JAMES PRATHER, Abilene Christian University, United States
MICHAEL HILTON, Carnegie Mellon University, United States
SRUTI SRINIVASA RAGAVAN, Indian Institute of Technology, India
MOHAMMAD AMIN ALIPOUR, University of Houston, United States

Generative artificial intelligence tools (Generative AI), such as ChatGPT, allow users to interact with them in intuitive ways (e.g., conversational) and receive (mostly) good-quality answers. In education, such systems can support students' learning objectives by providing accessible explanations and examples even when students pose vague queries. But, they also encourage undesired help-seeking behaviors, such as by providing solutions to the students' homework. Therefore, it is important to better understand how students approach such tools and the potential issues such approaches might present for the learners.

In this paper, we present a case study for understanding student-AI collaboration to solve programming tasks in the CS1 introductory programming course. To this end, we recruited a gender-balanced majority non-white set of 15 CS1 students at a large public university in the US. We observed them solving programming tasks. We used a mixed-method approach to study their interactions as they tackled Python programming tasks, focusing on when and why they used ChatGPT for problem-solving. We analyze and classify the questions submitted by the 15 participants to ChatGPT. Additionally, we analyzed user interaction patterns, their reactions to ChatGPT's responses, and the potential impacts of Generative AI on their perception of self-efficacy.

Our results suggest that, in about a third of the cases, the student attempted to complete the task by submitting the full description of the tasks to ChatGPT without making any effort on their own. We also observed that few students verified their solutions. We discuss the potential implications of these results.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI)**; • **Social and professional topics** → **Computing education**; • **Computing methodologies** → Artificial intelligence.

Additional Key Words and Phrases: Generative Artificial Intelligence, Human-AI Interaction, Self-regulation, CS1, User study, Novice programmers

ACM Reference Format:
Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Student-AI Interaction: A Case Study of CS1 students. 1, 1 (October 2024), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Many interactive learning environments (ILEs) offer on-demand help to positively influence learning; here, the learner actively seeks information, and the systems provide the same [3]. Although such forms of help-seeking behaviors in ILEs, when effective, are linked to improved learning outcomes, many learners do not use the available help resources effectively[1, 26]. This raises concerns that ILEs may not reach their full potential unless we find ways to help students use these support tools better. Furthermore, since seeking help is a crucial skill that affects learning in many situations[2, 30, 31], designing ILEs to encourage effective help-seeking behaviors could greatly enhance their educational value[2, 13].

Generative AI can be considered an interactive learning environment (ILE), as they can provide on-demand help, such as explanations of concepts and working code examples, to students. They play a crucial role as interactive components

Authors' addresses: Matin Amoozadeh, University of Houston, Houston, TX, United States; Daye Nam, Carnegie Mellon University, Pittsburgh, PA, United States; Daniel Prol, Universidad Internacional de la Rioja, , Logroño, , Spain; Ali Alfageeh, University of Houston, Houston, TX, United States; James Prather, Abilene Christian University, Abilene, TX, United States; Michael Hilton, Carnegie Mellon University, Pittsburgh, PA, United States; Sruti Srinivasa Ragavan, Indian Institute of Technology, , Kanpur, India; Mohammad Amin Alipour, University of Houston, Houston, TX, United States.

Manuscript submitted to ACM

within broader educational settings, and contribute to interactive learning by providing immediate responses and facilitating learning interactions in real time. Generative AI tools such as ChatGPT and Copilot have become integral tools for many students, particularly those in introductory programming courses [34]. They can assist students by providing code suggestions, explaining programming concepts, identifying and resolving errors, and generating code documentation [10, 21]. Their widespread adoption signifies a major shift in how programming is learned and practiced. However, this increasing reliance on Generative AI tools presents significant challenges. One primary concern is that students might become overly dependent on these tools, potentially hindering their ability to develop fundamental programming skills [37]. Additionally, AI-generated solutions might lead to academic dishonesty or reduced problem-solving capabilities among students [7]. This dichotomy has sparked a debate among educators: while some argue against the use of Generative AI in learning due to these risks, others advocate for its judicious use to enhance learning outcomes [22].

Despite various studies exploring the impact of Generative AI tools like GitHub Copilot on productivity [19, 48] and learning outcomes [27, 42], there remains a critical gap in understanding how novice programmers interact with these tools without any limitations on using Generative AI, where they can freely utilize AI-driven assistance and guidance. Understanding the level of trust of students in these tools, which impacts their adoption and learning outcomes [4, 5], and analyzing their interactions while using Generative AI for programming tasks is crucial. These insights enable us to identify students' behaviors, actions, decisions, and responses when using Generative AI tools, guiding the design of AI-assisted learning interfaces that enhance conceptual understanding and drive improved learning outcomes.

To this end, our study investigates the interaction of novice programmers with ChatGPT while solving programming tasks. We conducted a mixed-method study involving 15 CS1 students who used a custom VSCode plugin that integrated ChatGPT directly into their coding environments. This setup allowed us to observe their natural use of Generative AI assistance without external constraints, and to gain a deeper understanding of how students use Generative AI in completing programming tasks, namely 1) when, how and why they use them and to what effectiveness, and 2) what strategies they employ to integrate Generative AI into their problem-solving processes.

Our findings highlight that participants extensively interacted with Generative AI, yet successfully provided correct answers only in 65% of the cases; the rest remained unsolved. Some successful interactions involved step-by-step prompting, or hybrid approaches that combine independent programming and Generative AI support. The overall acceptance of the Generative AI responses also varied, from full adoption of the Generative AI's response to using Generative AI to find answers to the queries, or simply comparing the Generative AI responses to their own solutions.

2 Related Work

2.1 Exploring the Impact of Generative AI in Programming

The recent widespread deployment of generative AI programming assistants has motivated much research on the use of these tools. Researchers have conducted empirical studies [14, 23, 39] to evaluate the quality of code and explanations that Generative AI programming assistants can generate. Others have explored the usefulness of Large Language Model (LLM)-based programming tools through user studies [6, 20, 28, 38, 41, 45, 46, 49]. Notably, Vaithilingam *et al.* [45] compared the user experience of GitHub Copilot with traditional auto-complete and found that programmers faced more frequent difficulties in completing tasks with Copilot, although there was no significant impact on task completion time. Barke *et al.* [6] took a step further to understand *how* programmers interact with code-generating models using a grounded theory analysis, identifying two primary modes of interaction: acceleration, where Generative AI is used to speed up code authoring in small logical units, and exploration, where Generative AI serves as a planning assistant by suggesting structure or API calls. While these studies have contributed to our understanding of how developers use Generative AI developer tools, they mainly focused on evaluating the effectiveness of such tools and did not directly address developers' perceptions of them. Some studies, such as those by Liang *et al.* [24] and Ziegler *et al.* [49], have also focused on how developers perceive these tools. For instance, Liang *et al.* [24] conducted a survey involving 410 developers to investigate the usability challenges associated with Generative AI programming assistants, finding that developers appreciate their autocomplete capabilities, but also reported challenges ranging from the quality of code generation to potentially infringing on intellectual property. Although these studies provide valuable insights on how users *perceive* the Generative AI tools, the insights are primarily applicable to professional programmers; our study focuses on the use of such tools by CS students, and more studies are needed in this area.

2.2 The Role of AI in Education

Recent studies have started to explore the applications of Generative AI tools in educational contexts. For example, Denny *et al.* [9] have identified desirable characteristics for Generative AI teaching assistants in programming education. They emphasize the importance of features that provide immediate, engaging support while allowing students to maintain autonomy in their learning process. Liu *et al.* [25] investigated the integration of Generative AI tools in the CS50 course at Harvard University, demonstrating how they can significantly enhance the learning experience for students in introductory programming classes. Their approach involved developing and deploying a suite of Generative AI-based tools designed to emulate a 1:1 teacher-to-student ratio, thereby providing personalized, real-time support to students. These tools, including the CS50 Duck, a virtual rubber-duck debugging assistant, were well received by students, who reported that the Generative AI tools made them feel as if they had a personal tutor available at all times. Prather *et al.* [36] have recently published a series of studies that examine the usability and interaction challenges faced by novice programmers using Generative AI tools, e.g., GitHub Copilot. Their research highlights the dual nature of such tools: while they can significantly accelerate the coding process and aid in overcoming programming blocks, they also introduce unique cognitive and metacognitive difficulties. Students often found it "weird" how accurately Copilot predicted their needs, which led to a mix of fascination and dependency concerns. The study also identified new interaction patterns such as "shepherding" and "drifting," reflecting the nuanced ways novices attempt to guide Generative AI or are misled by it. All these prior works together lend motivation to our research, highlighting the potential usefulness of Generative AI tools, while underscoring the need to balance learning support with independent problem-solving skills and over-reliance on Generative AI.

2.3 Interaction Patterns of Novice Programmers Using LLMs

To develop Generative AI systems that balance fostering independent thinking with learning support and avoiding overreliance, it is necessary to understand *how* students use LLMs in practice. In the past, Kazemitabaar *et al.* [21] conducted a thematic analysis of novice learners using an LLM-based code generator in a self-paced Python course. They identified distinct coding approaches among the learners, noting that the Hybrid approach, which combined manual coding with LLM assistance, produced the best outcomes. However, the study also highlighted signs of over-reliance on LLMs, such as copying LLM output without changes, and positive self-regulation behaviors, like adding code to verify LLM output. In another study, Nguyen *et al.* [32] conducted a large-scale multi-institutional study that explored the challenges faced by near-novices when interacting with Code LLMs. Their study identified barriers such as difficulties in expressing problem understanding and using appropriate coding terminology. Prather *et al.* [37] further investigated the impact of generative AI tools on novice programmers' metacognitive awareness and problem-solving strategies, highlighting the potential widening gap between well-prepared and under-prepared students in the era of generative AI. However, there is a lack of studies elucidating when and how students use LLMs, to what effectiveness, and to what effect on their self-efficacy; this is what we cover in the present study.

2.4 Assessing Student Self-Efficacy in AI-Enhanced Learning

Self-efficacy, or the belief in one's capabilities to achieve a goal or an outcome, is a crucial factor in students' learning processes. Recent studies have examined how interactions with GenAI tools influence students' self-efficacy in programming. Tankelevitch *et al.* [43] discussed the metacognitive demands imposed by GenAI systems, highlighting the need for tools that support students in monitoring and controlling their learning processes. Xue *et al.* [47] investigated the impact of ChatGPT on introductory programming students, finding that the AI tool can enhance students' self-efficacy by providing immediate feedback and support, thereby reducing the anxiety associated with complex problem-solving tasks. Furthermore, Denny *et al.* [9] emphasized the importance of designing AI teaching assistants that not only assist with immediate problem-solving but also help students develop long-term self-efficacy by encouraging independent learning and critical thinking. This study serves to add further evidence on the effect of Generative AI use on student self-efficacy, albeit using a different approach (namely, pre- and post-task comparisons).

3 Methodology

This study focuses on students' use of Generative AI–covering the whats, whys, whens, and hows of doing so. The specific research questions for our study are as follows:

RQ1: How frequently do students use Generative AI while solving programming tasks?

Participant	Age	Gender	Education-level	Racial/Ethnic	College-generation
P1	19	Female	Freshman	African American	Continuing-generation
P2	18	Female	Freshman	African American	Continuing-generation
P3	19	Female	Freshman	African American	Continuing-generation
P4	19	Male	Freshman	Asian	Continuing-generation
P5	18	Female	Freshman	Asian	Continuing-generation
P6	19	Female	Freshman	Asian	First-generation
P7	19	Male	Freshman	African American	Continuing-generation
P8	18	Male	Freshman	Latino	First-generation
P9	22	Female	Postbacc	African American	Continuing-generation
P10	18	Male	Freshman	Asian	Continuing-generation
P11	19	Female	Freshman	White	Continuing-generation
P12	19	Male	Freshman	African American	Continuing-generation
P13	18	Male	Freshman	African American	Continuing-generation
P14	20	Male	Junior	Latino	Continuing-generation
P15	19	Female	Freshman	African American	Continuing-generation

Table 1. Demographic Information of Participants

RQ2: How do students interact with Generative AI while solving a programming task?
RQ3: How does the self-efficacy of students change before and after programming with Generative AI?

To answer these questions, we adopted a mixed-methods approach, gathering data from 15 CS1 students using Generative AI to perform a programming task. As we reason later in this section, the access to Generative AI in our study was via a plug-in that we designed for the study.

3.1 Participants

We recruited students from a CS1 course in a large US public university. We sent an email to the course email list, via the instructor (also an author of this paper), inviting volunteers to participate in a research study involving programming tasks with AI. Each participant was offered *five* bonus points in their course for their participation. We considered offering bonus points so as to not coerce students into participating. Participants could simply fill in an available time slot for the study if they were willing to participate. The course had 110 students, of which 15 students volunteered to participate.

Table 1 summarizes the demographics of the participants. Our study was gender balanced (8 females, 7 males). Two of our participants were first-generation students (i.e., no parent or guardian possessed a four-year college degree); the rest were continuing-generation students and had at least one parent or guardian with a four-year college degree. Most of our participants (13 out of 15) were freshmen; the other two were junior and post-baccalaureate respectively. The participants reported diverse racial/ethnic backgrounds, and everyone used English as their first or second language fluently.

3.2 Study Procedure

The study was conducted in an office within the computer science department during the spring semester of 2024, with one participant per study session. The moderator conducting the study was not part of the CS1 course in any capacity (instructor, tutor, or TA).

Each study session began with a study debriefing, and the participant signing the informed consent form. The participant then filled out a pre-study questionnaire, which contained questions about: 1) demographics, 2) programming background, and 3) self-efficacy questionnaire(from [33]).

Then, the participants were introduced to the study environment. We chose the Python programming language and VSCode environment for the study, since it was already used in the course. Participants were also introduced to the plug-in, and were briefed on how to ensert prompts into it, should they choose to use Generative AI for their tasks. We also emphasized to participants that other Generative AI or aids (e.g., web searches) were not allowed during this time. We also clarified that their performance would not be judged by others, or affect their course grades, to ensure that the participation was non-coercive.

Participants were introduced to the programming tasks on a sheet of paper, and were given one hour to complete the tasks. On completion of the tasks, the participants completed a post-study survey containing the same self-efficacy questionnaire administered as part of the pre-study survey. The study sessions lasted between 30 and 70 minutes each.

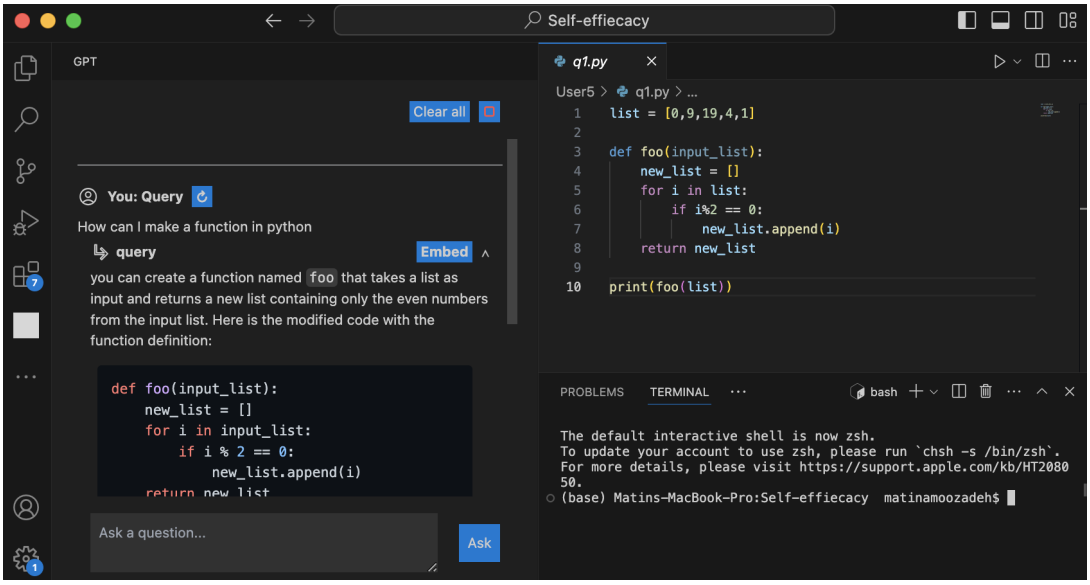


Fig. 1. Interface of the VS Code plug-in used in the study: ChatGPT Response and Participant Code

3.3 Programming tasks

Participants were asked to complete three programming tasks, each of which is tailored to the complexity appropriate for a CS1 level and covered topics taught by the instructor throughout the semester preceding the study.

Participants were provided with input and output examples for each task. Collectively, these three tasks addressed core programming concepts such as conditionals, loops, functions, input/output operations, and simple algorithm design, providing participants with a comprehensive understanding of essential programming principles and problem-solving approaches.

As mentioned earlier, participants had a choice of using Generative AI or not to complete these tasks. The use of Generative AI had to be via the custom plug-in we developed, and the use of any other resource (e.g., web search) was disallowed.

3.4 Generative AI plugin

The goal of this study is to understand when, how, and why CS1 students interact with Generative AI tools and how effective these interactions are. Therefore, we decided to allow only the use of Generative AI for students' help seeking to complete their tasks and disallowed all other sources.

In deciding what Generative AI tool to use for the study, we had several choices: OpenAI ChatGPT, Google Gemini, and Github Copilot, to name a few. We chose to use OpenAI ChatGPT for two reasons: 1) it provided general help beyond programming, and 2) from a prior study, it seemed to be the most familiar Generative AI among students [5].

However, in a recent study, Tankelevitch *et al.* [43] highlighted the metacognitive demands placed on users when interacting with Generative AI systems (e.g., the need to remember sub-tasks and which one they are on), and suggested the need to minimize switching between the task and the Generative AI help-seeking contexts. Therefore, we used a GPT-3.5 plug-in within the VSCode[1] [29] programming environment, to reduce the cognitive load associated with switching between different environments, thereby allowing participants to remain within a single, cohesive coding environment.

Figure 1 shows the VSCode plugin [29] we used for this study. As the figure shows, the plugin had a simple interface comprising of a prompt-writing box (See "Ask a question..." in the image), allowing users to access ChatGPT within the VSCode IDE. This prompt box allows students to freely input prompts without any restrictions on the number or type of questions, or prompt lengths. The Generative AI responses to the prompts are also provided in the same pane, as the figure shows. The fact that users can access Generative AI and see its responses together with their code in the same window allows for a focused environment where users do not need to switch context to another window, and are less tempted to use other AIs.

3.5 Data gathering

Our study employed a multi-faceted approach to data collection, designed to capture a comprehensive view of participants' interactions with Generative AI (here, ChatGPT) during programming tasks. We utilized a custom VSCode plugin to log all

[1]https://code.visualstudio.com/

text editor interactions (including any text selections) and ChatGPT prompts, and responses. The audio and video recordings of the participants, along with recordings of the participants' screens throughout the session provided qualitative data on problem-solving behaviors and reactions to AI assistance. Pre-study and post-study surveys collected demographic information and self-efficacy data using Likert scales. In addition, we collected detailed logs of ChatGPT conversations, including the full text of prompts and responses. This diverse dataset allowed us to analyze the frequency and timing of Generative AI use (RQ1), students' activities and their patterns (RQ2), their Generative AI use strategies (RQ3), and the effects of Generative AI use on students' self-efficacy (RQ4). By combining automated logging, video analysis, and self-reported measures, we ensured a foundation for both quantitative and qualitative analysis of novice programmers' engagement with AI-assisted coding.

3.6 Data analysis

To answer our research questions, we conducted both qualitative and quantitative analyses of the data.

3.6.1 Qualitative. Two authors independently coded the video data for three programming tasks, each varying in difficulty levels and topics, utilizing the initial codebook. Subsequently, they engaged in discussions to review the initial coding outcomes, resolve any conflicts, and enhance the codebook further. With the identification of each new code, we reviewed all prior comments for possible adjustments.

The codebook provides a clear breakdown of how participants interacted with Generative AI prompts throughout various stages of programming tasks. It categorizes prompts based on their types and how participants engaged with them, whether standalone or as follow-up interactions. Additionally, it outlines how participants responded to errors during programming and their preferences for resolving them, including using Generative AI responses or ignoring errors altogether. Moreover, the book details the extent to which participants accepted AI-generated solutions, from copying entire responses to incorporating only certain parts or using them for guidance. In essence, the codebook served as a guide for understanding participant-AI interactions during programming activities in educational settings. The codebook includes such thematic coding of responses allowing us to draw meaningful insights about students' perspectives and shed light on the complex range of attitudes and emotions surrounding Generative AI systems in the context of programming.

3.6.2 Quantitative. We conducted two surveys, one administered before the programming session and another after, to assess participants' level of self-efficacy. Participants rated their self-efficacy levels using 5-level Likert scales, indicating their agreement with statements related to their programming abilities. These Likert scores provided quantitative data to measure changes in participants' self-efficacy before and after engaging in the programming tasks.

4 Results

Fifteen CS1 students participated in our study, and we asked each of them to do their best to complete three programming tasks. Table 2 summarizes the tasks completed and the correctness of solutions for each participant. Notably, most participants (P3, P15) had atleast one correct solution, but only 6 out of 15 participants (P3, P5, P6, P9, P12, P14) solved the three programming tasks successfully. Overall, we had 40 completed (correct + incorrect) solutions submitted by participants. We then analyzed the logs from the Generative AI.

4.1 RQ1: How frequently do students use Generative AI while programming?

We used the logs from the plugin to analyze the frequency of Generative AI use. Of the 40 completed participant submissions, 29 solutions were created with assistance from the Generative AI plug-in provided, and 11 without Generative AI. requests, as a measure of the frequency of participants' Generative AI use during each task. Table 2 displays this frequency of participants' plugin-aided Generative AI use, illustrating the different levels of dependence on Generative AI assistance among participants.

However, these use frequencies did not always translate into task completion successes. Overall, among the 29 Generative AI assisted solutions created by participants, only 19 solutions (65%) were correct and the rest (35%) were incorrect. Even among two participants (P2, P5) used Generative AI extensively for a task, only one instance (P5, Q3) was successful; P2 did not manage to complete Q1 correctly, even after 9 Generative AI queries. Moreover, some participants, such as P3 and P6, achieved correct answers ('C') for one or all tasks even without relying on Generative AI, indicating that some participants may not need Generative AI to perform well.

Thus, participants' frequency of Generative AI use alone did not lead to successes; instead, participants' strategies in using Generative AI and their inherent problem-solving abilities both played crucial roles in their success.

Participant	Tasks	Completed Q1,Q2,Q3	Correct Q1,Q2,Q3	Using Generative AI Q1,Q2,Q3	#Times Generative AI
P1	Condition, String, Function	Y, Y, N	W, W, N	N, Y, N	0, 1, 0
P2	Condition, String, Function	Y, Y, Y	W, W, C	Y, Y, Y	9, 3, 1
P3	Condition, String, Function	Y, Y, Y	C, C, C	N, N, Y	0, 0, 2
P4	Condition, String, Function	Y, Y, N	C, W, N	N, N, N	0, 0, 0
P5	Condition, String, Function	Y, Y, Y	C, C, C	N, Y, Y	0, 1, 10
P6	String, List, Array	Y, Y, Y	C, C, C	N, Y, N	0, 5, 0
P7	String, List, Array	Y, Y, Y	C, C, W	Y, Y, Y	2, 1, 4
P8	String, List, Array	Y, Y, Y	C, C, W	N, N, Y	0, 0, 3
P9	String, List, Array	Y, Y, Y	C, C, C	Y, Y, Y	1, 1, 1
P10	String, List, Array	Y, Y, Y	C, C, W	Y, Y, Y	1, 2, 1
P11	String, List, Array	Y, Y, Y	W, C, C	N, Y, Y	0, 1, 1
P12	String, List, Array	Y, Y, N	C, C, N	Y, Y, N	3, 1, 0
P13	String, List, Array	Y, N, N	W, N, N	Y, N, N	3, 0, 0
P14	String, List, Array	Y, Y, Y	C, C, C	Y, Y, Y	1, 1, 1
P15	String, List, Array	Y, Y, Y	W, W, W	Y, Y, Y	1, 1, 1

Table 2. Usage and Problem Solving by Participants: This table details participants' problem-solving activities. "Participant" identifies participants, "Tasks" shows the main concepts in the tasks attempted, and "Completed" indicates if the task has been completed by the participant ('Y' for yes, 'N' for no). "Correct" shows if the submission was correct ('C'), wrong ('W'), or not attempted ('N'). "Using Generative AI " indicates Generative AI use ('Y' for yes, 'N' for no), and "Times Generative AI" shows the number of times Generative AI used.

4.2 RQ2: How do students interact with Generative AI while programming?

To understand how and why some participants were able to use Generative AI to correctly complete tasks, and others were unsuccessful in doing so, we delved into various qualitative aspects of participants Generative AI use–when and how participants interacted with Generative AI, and how these translated into task completion successes.

4.2.1 When do students use Generative AI? We observed that participants typically engaged with generative AI at three different stages during their task:

- *Early in the task.* In our study, 8 participants, in a total of 18 tasks, turned to ChatGPT right away, at the beginning of the task, without making any initial attempts on their own; see instances of "Early" in Table 3 under "When". Of them, four participants (P9, P10, P14, P15) used Generative AI right from the beginning for all three programming tasks. Typically, during these instances, participants typed in the description of the task as is, and relied on Generative AI to complete the task for them.
- *Middle of the task.* In 9 out of 45 total tasks, participants (7 out of 15) sought the assistance of Generative AI in the middle of their programming; these instances are labeled "Middle" in Table 3, under "When". Initially coding manually, they resorted to using Generative AI when they encountered errors, or to understand programming concepts.
- *After completing the task.* Interestingly, two participants (P3 and P5) employed Generative AI even after they had already arrived at the correct answer on their own. See instances of "After-Solving" in Table 3; Section "When". Here, P3 used Generative AI to validate she understood the problem description correctly by inquiring about the distinction between the terms (i, N) mentioned in the problem. P5 reached the correct solution on their own first, but still sought Generative AI's solution for the entire question, apparently to verify the correctness and/or effectiveness of their solution by comparing their solution with the Generative AI's responses.

4.2.2 What are common Generative AI usage patterns among students? We conducted an activity analysis of the students' programming tasks, coding the activities they engaged in. We identified six common activities users performed when using Generative AI, namely, reading, thinking, writing code, modifying code, prompting, and debugging. Table 4 defines these activities. Participants engaged in these activities in varying levels, with some engaging in the same activity on multiple instances during the same task, and others performing them only once per task.

An analysis of the activity sequences for each task revealed two large patterns, as illustrated in 2; the figure shows the activity sequences[2] of two tasks by P14 and P7 respectively. Whereas P14 followed a straight path across all three programming questions, performing each activity once, P7 in question 4 repeated certain activities such as running, debugging, prompting,

[2]Notice that in the figure "correct" and "wrong" are not activities; we also did not count "running code" as an explicit activity because it was a part of task completion or verification.

User	When (Generative AI Used)			Approach			Usage Pattern		
	Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3
P1	-	Middle	-	-	Hybrid/partial	-	I	I	-
P2	Early	Middle	Early	step-by-step	Hybrid/partial	Full description	I	I	I
P3	-	-	After-Solving	-	-	step-by-step	I	I	I
P4	-	-	-	-	-	-	I	I	-
P5	-	After-Solving	Early	-	Full description	step-by-step	I	L	I
P6	-	Middle	-	Themselves	Hybrid/partial	-	L	I	I
P7	Middle	Middle	Middle	Hybrid/partial	Hybrid/partial	Hybrid/partial	I	L	I
P8	-	-	Middle	-	-	Hybrid/partial	I	I	I
P9	Early	Early	Early	Full description	Full description	Full description	L	L	L
P10	Early	Early	Early	Full description	Full description	Full description	L	I	L
P11	-	Early	Early	-	Full description	Full description	L	L	L
P12	Early	Middle	-	Full description	Hybrid/partial	-	I	I	-
P13	Middle	-	-	step-by-step	-	-	I	I	-
P14	Early	Early	Early	Full description	Full description	Full description	L	L	L
P15	Early	Early	Early	Full description	Full description	Full description	L	L	L

Table 3. Summary of Generative AI Usage and Approaches by Participants: The table summarizes the Generative AI usage and approaches taken by users for different programming questions (Q1, Q2, Q3). The first set of columns under "When (Generative AI Used)" indicates the timing of Generative AI usage classified as "Early", "Middle", "After-Solving", or "-" if not used. The second set of columns under "Approach" details the approach taken by participants, such as "Hybrid/partial", "Step by step", "Full description", and "-" if not used. The third column shows nature of participants' activities, with some repeating activities multiple times as iterative "I" and others performing them only once as linear shown as "L".

Activity	Description
Reading	Reading the question before starting to write code
Thinking	Thinking about solution, or Generative AI prompt and response
Writing Code	Writing code by themselves or copying Generative AI response
Modifying Code	Editing existing code
Prompting	Asking questions from Generative AI
Verification and Debugging	Running code and finding errors

Table 4. Description of six common activities users performed when using Generative AI user activities.

(a) Linear activities of P14 for all questions

(b) Iterative activities of P7 for Question 3

Fig. 2. Examples of activities

modifying, and running again. Overall, the 15 participants in our study completed a total of 40 tasks (Table 2, column Completed); of them, we observed 17 instances of straight or *linear* activity sequences and 23 instances of repetitive activities. These are indicated by "L" and "I" in Table 3, under "Usage Pattern".

The pattern of a student returning to the same activity is an indicator of an iterative process (e.g., several iterations of re-running code, or prompting Generative AI more than once for the same task). In contrast, a linear pattern is an indicator of one-shot task completion, assisted with or without Generative AI. The fact that over 40% instances were linear is suggestive of one of two possibilities. The first is that of high task performance abilities, as marked by minimal help-seeking, and lack of edit-verify loops. The second possibility is that of over reliance on Generative AI, wherein participants made a single prompt to Generative AI the result of which resulted in a direct solution to the problem, following which participants did not have to enter an edit-verify loop. To assess which of these possibilities caused the observed usage patterns, we drilled down into the nature of participants' interactions with Generative AI.

4.2.3 What interaction strategies do students adopt with Generative AI? There were three key aspects to participants interactions with Generative AI: the level of task decomposition, the kind of information they needed, and how participants exploited the Generative AI response.

Levels of task decomposition As Table 3 shows, participants engaged in three prompting strategies, based on the level of task decomposition: *full description*, *step-by-step*, and *partial*:

- In the *full description* strategy, the participant copied the full description of the task into the plugin, thereby offloading task completion entirely to Generative AI. In total, there were 17 instances across 9 participants where participants, early in the task (Table 3), provided the entire problem description to Generative AI to be solved by the latter. Of them, 5 participants adopted this approach for only one or two tasks, and relied on step-by-step or hybrid /partial approach for others. However, 4 participants (P9, P10, P11, P14, P15) entirely relied on this strategy for all tasks, taking to Generative AI very early in the task. Note, however, that this opportunistic approach to offload the task completion to Generative AI did not translate to complete and correct solutions. Among the participants who relied on full descriptions, P10 (for Q3) and P15 (for Q1, Q2, Q3) were unable to find the correct answers.
- In the *step-by-step* interaction strategy, participants broke the question down into structured sub-goals and then used the Generative AI to solve the problem step by step–resulting in an iterative approach. Four participants (P2, P3, P5, P13) adopted this problem-solving strategy. For example, participant P5 used Generative AI for the first time to inquire, 'How to round numbers in Python?' and then followed up with a second query, "How to round numbers in Python only from the tenths place?". Similarly, P13 used Generative AI three times for their first question. The first inquiry was, "In Python, can you turn an integer into a list?" The second query was about arrays: "What is an array?" Lastly, the user asked: "Given an integer n, produce the array: 0123…n, 1234…0, n012…n-1?." Indeed, there is diversity in the nature of prompts, as we describe later.
- The remaining participants used a *hybrid* or partial strategy where participants tackled some parts of the task independently while seeking assistance from Generative AI for some other parts. Notably, participants attempted to write programs independently, but turned to Generative AI for debugging to resolve errors or others. We identified n=8 instances that we categorized as Hybrid or Partial. For example, when stuck with a bug, participant P2 asked Generative AI, "Fix my code to work." User P6 encountered an error and copied the message: `"TypeError: 'int' object is not subscriptable"`. We observed only one instance of a participant successfully completing a task on their own (Table 3, Q1, P6 labeled "Themselves").

Information needs. We analyzed the prompts that users submitted to ChatGPT, via the logger. In all, 15 participants wrote 60 prompts, and we categorized them based on the information need the prompt aimed to serve.

- *Entire Solution.* As described earlier, several participants simply typed in the question descriptions, which we considered as their entire prompt. These accounted for a third of all prompts (20 out of 60). Often, these prompts were issued early on during the task, and participants obtained the entire solution to the task, via this single prompt; 8 out of 15 participants engaged in this behavior.
- *Coding Concepts.* Another popular prompting strategy, namely seeking to understanding programming concepts, was equally common among participants. About a half of participants (7 out of 15) accounting sought such information, accounting for 30% (19 out of 60) of all prompts. These instances of conceptual understanding, arose in largely two cases, namely when the participant had a logical understanding of what to do, but needed help executing them in Python (e.g., P2: "How to append to a list?", P5:"How to round numbers in Python?"), to understand whether the language allowed something (e.g., P7: "In my program, I am trying to move the negative numbers to the front. Could I initialize an empty list first?"), or to understand jargon (e.g., P13: "What is an array?').
- *Program Logic.* In about 15% of the time (10 out of 60 prompts), participants utilized Generative AI because they found it challenging to determine the sequence of steps or instructions needed to accomplish a specific task or goal within the program. In other words, they used Generative AI for planning help. For instance, P2 asked about "a function to return the first n values of the triangular number sequence starting from 1.", expecting to receive the steps to accomplish the task. P12 needed help trying to reverse a list. P7 inquired, "I am trying to produce the array 0, 1, 2, 3 all the way to n, meaning it could also be n, 0, 1, 2. Is my approach correct so far?".
- *Debugging.* Finally, some participants turned to Generative AI for debugging and their questions typically revolved around the question: "How can I solve this error?". In total, 11 out of 60 prompts utilized Generative AI for error resolution. Often, users would simply copy the error message from the console and ask ChatGPT for help. P6 copied and pasted the code, asking, "What is wrong with this code?" P10 asked, "Why does the code not work?" P6 specifically inquired about the error received, in some ways treating it like a search engine: "`TypeError: 'int' object is not subscriptable.`"

Exploiting Generative AI responses We identified various user actions upon receiving responses from Generative AI, which we categorize as acceptance categories. These actions reflect users' reactions to Generative AI answers, and this collaboration can result in either successes or failures in the completion of tasks. These categories are based on the prompts and their reactions to the Generative AI's answers.

- *Entire response.* Participants often accepted the Generative AI's response in its entirety. We had $n = 18$ out of 60 prompts we found that users accepted Generative AI answers, without any evidence of explicitly evaluating the responses. For example, P10 formulated a prompt for Question 3 "Given an integer n, produce the array" and accepted the Generative AI's solution without any modification. Similarly, P7, when addressing the same question, queried "I am trying to produce the array 0,1,2,3 all the way to n, meaning it could also be n,0,1,2 is my correct so far?" fully embraced the Generative AI's response.
- *Selective use.* Another common use of Generative AI responses was that participants tried to understand the Generative AI response, and then translate that understanding to implementing their own solution. There were $n = 23$ out of 60 prompts where users sought ideas from Generative AI responses to write their own code. For instance, P5 asked: "How to round numbers in Python only from the tenths place" and the Generative AI response was "To round numbers in Python only from the tenths place, you can use the round() function", followed by an example of how `round()` function works. The user then completed the task using their own code.
- *Reject and Retry.* Using recorded videos of facial expressions and eye movements, we also identified cases in which participants spent an extended amount of time reading and (most likely) contemplating Generative AI's responses. We observed six participants visibly contemplating the Generative AI's response to 19 (out of 60) prompts. These participants read the responses but did not exploit it, by way of either copying the code from the response, or simply write it in their own way. Instead, they went on to write another prompt, which may indicate their rejection of the Generative AI solution The reasons for this varied from Generative AI responses not meeting their information needs, to a rejection of the implementation choices made in the responses. P12's initial query was "I need help trying to reverse a list." After reading the Generative AI response, the user attempted to write another prompt, "Help me turn a sequence of numbers into a list." Once again, the user read the Generative AI's response. Subsequently, P12 revised his prompt and wrote: "Without using a built-in function, help me turn a sequence of numbers into a list." After this prompt, he began to write code in their own way.

4.3 RQ3: How does student self-efficacy change before and after programming with Generative AI?

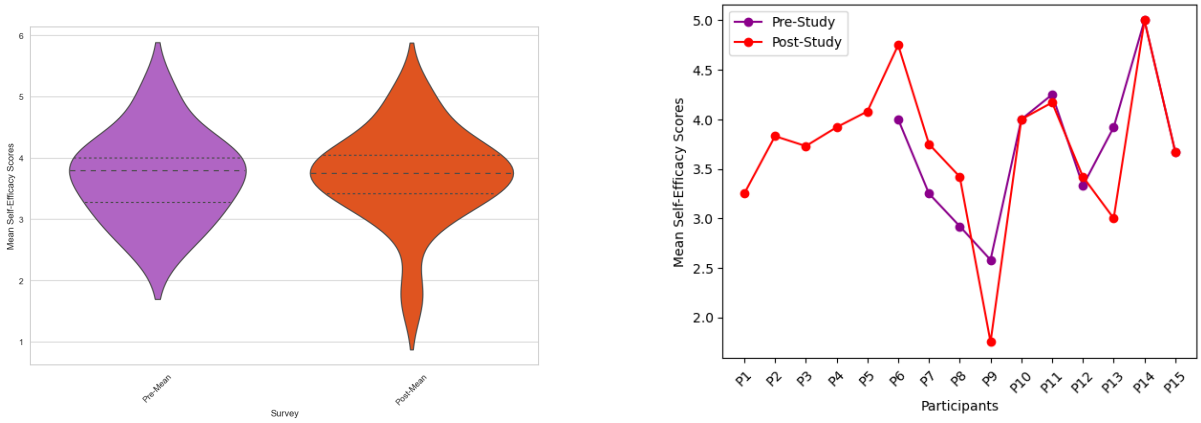
With such diversity in Generative AI usage among participants–in terms of frequencies, usage patterns and strategies, and success rates– we went beyond simple task completion to metacognition, specifically to evaluate the impact of Generative AI tools on students' self-efficacy.

For this, we analyzed the self-efficacy questionnaires we administered as part of the pre-study and post-study surveys. Unfortunately, for the first five participants (P1 to P5), we did not capture the pre-study self-efficacy data; thus, we had data from 10 participants.

Figure 3 presents the participants' perception of self-efficacy before and after using Generative AI. Figure 3a shows the distribution of perception of self-efficacy before and after the study. Given the small number of observations, we cannot draw any statistically-sound conclusion about any difference between the distributions.

Figure 3b compares the self-efficacy of the participants before and after the experiment. It shows that P6 experienced a noticeable increase in self-efficacy, from a self-efficacy score of 4.00 in pre-study to 4.75 after the study, indicating a potential positive impact of using Generative AI on self-efficacy. P6 completed all three programming tasks successfully and demonstrated a mix-model behavior by independently addressing two tasks and using a hybrid approach with Generative AI for one task. The increase in self-efficacy scores suggests that employing Generative AI was effective in improving her perception of self-efficacy.

In contrast, self-efficacy in P9 decreased from 2.58, before the study, to 1.75 after the study. Although P9 completed all programming tasks successfully, she copied the full description to the plug-in, asked ChatGPT for the answer, and pasted the results as the submission. Similarly, this occurred for P13, who began with a pre-study self-efficacy score of 3.92, however, the post-study mean score dropped to 3.00. He submitted an incorrect solution for one task and did not finish the other two tasks, she used Generative AI in a step-by-step approach.



(a) Distribution of Pre-study and Post-study self-Efficacy scores

(b) Pre- vs. Post-study self-efficacy scores by participant

Fig. 3. Self-efficacy of participants

5 Threats to Validity

Replicability *Can others replicate our results?* Because Generative AI and its usage and acceptance are rapidly emerging, we do not know if the results of this study will be replicable due to changing student behaviors and rapid advances in Generative AI. We encourage other researchers to replicate our study for different populations to check for emerging behaviors of students when using Generative AI.

Internal *Did we skew the accuracy of our results with how we collected and analyzed data?* We used open coding to analyze the data and two authors reviewed all the data to make sure that the coding of videos adhered to the codebook that was created.

External *Do our results generalize?* Because our study has a small sample size (n=15) relative to the overall student population, it is not possible to generalize to all students' behavior. However, we attempted to have diversity based on gender. The bias of drawing conclusions from self-selection bias remains a threat to validity.

6 Discussion and Concluding Remarks

In this section, we discuss the potential implications of our findings for computing education research and practice.

6.1 Interactions with Generative AI

In this study, our objective was to investigate how CS1 students utilize Generative AI for programming questions to assess whether Generative AI serves as a help-seeking tool for beginners in programming. As observed in other recent studies [21, 37], our findings indicate that a large number of Generative AI users rely on providing full descriptions of programming questions to find solutions without making sufficient effort on their own, even under supervision. This trend aligns with the patterns of over-reliance on LLMs identified by Kazemitabaar *et al.* [21], particularly among novices using the 'AI single prompt' approach, which resulted in lower performance on subsequent tasks. The observed behavior raises concerns about the potential overreliance on Generative AI in educational settings, where students might increasingly rely on Generative AI to provide all solutions to the detriment of their learning. This echoes the observations of Fernandez and Cornell [12], who emphasized the need for careful integration of AI-driven code generation tools to avoid such overreliance. Help-seeking is crucial for students to grasp new concepts, acquire skills, and tackle challenges in their computing courses [16]. However, when participants use full descriptions of programming tasks as prompts and accept complete Generative AI-generated responses, Generative AI may not effectively fulfill its role as a help-seeking tool that constructively aids struggling students, but an oracle that does the learners' job for them. This concern was also raised by Jošt *et al.*[18], highlighting the need for instructional strategies that emphasize breaking down problems and leveraging Generative AI for incremental learning. Our findings, alongside those of Prather *et al.* [37], suggest that some students may struggle with new metacognitive difficulties when using generative AI tools, including being conceptually behind in course material but unaware of it due to a false sense of confidence. Further exploration is needed to understand the underlying reasons for this undesired behavior and to encourage a more constructive use of Generative AI that promotes deep understanding and problem-solving skills in computing education.

Our observations reveal two main types of behavior in problem solving: iterative and linear. Students who employed an iterative approach refined their prompts to achieve correct answers, while students who used a linear approach used the

full description of the problems to find answers directly. This behavioral split reflects the findings of Vadaparty *et al.* [44], who noted similar patterns in student interactions with Generative AI in a CS1 course. The iterative approach can improve learning by encouraging deeper engagement with problem-solving processes, whereas the linear approach may indicate a tendency to seek quick fixes.

Some prompts were related to coding concepts, indicating that novices struggle the most to understand programming concepts. This is consistent with the findings of Liu *et al.* [25], who found that students often used Generative AI tools to clarify the concepts of coding and the logic of the program. Our study further suggests that while Generative AI can assist with the coding concepts, additional instructional support is needed for the logic and debugging of the program.

Our results show that in 30% of prompts, users accepted the Generative AI response, while in 70%, AI responses were used to learn concepts. This dual role of Generative AI as both an answer provider and a personal tutor aligns with the observations by Prather *et al.* [37] regarding the mixed impact of Generative AI on student learning, where Generative AI facilitated both understanding and dependency. Effective Generative AI integration should balance assistance with promoting independent problem-solving skills.

We can categorize the nature of interactions with Generative AI based on the reliance of a prompt on prior prompts into two groups: (1) stateless and (2) stateful. In the stateless interactions, individual prompts are independent of the prior prompts, hence prompts can be interpreted and answered independently. However, in stateful interactions, the prompt assumes that Generative AI uses the history of the user's interaction. For instance, P7 used stateful interaction, where he asked "So, what was wrong with my initial code?".

We observed that seven participants P4, P7, P9, P10, P13, P14, and P15 employed the same approach on all tasks. We call their strategy for problem-solving, *single-approach* strategy. In contrast, P1, P2, P3, P5, P6, P8, P11, and P12 used different approaches for different tasks that we call *mixed-approach*. As generative AI becomes more commonplace and more learners can access them for diverse sets of topics, a more granular investigation of these approaches becomes more important to guide pedagogy.

6.2 Self-efficacy and Generative AI

Our observations suggest that users who used full descriptions of the questions may prefer smooth interactions without challenges. This behavior correlates with lower self-efficacy scores, similar to the findings of Xue *et al.* [47] and Prather *et al.* [37], who observed that students with lower self-efficacy tend to rely more heavily on Generative AI tools. Overall, some users' self-efficacy levels increased after programming with Generative AI. This suggests the varied impacts of integrating Generative AI in educational contexts, shaping self-efficacy outcomes according to individual learning strategies and initial confidence levels. Enhancing self-efficacy through scaffolded Generative AI interactions could support more confident and independent problem-solving.

6.3 Implications for Computing Education

The current advancement in prompt engineering focuses on productivity, aiming to help developers find final solutions quickly. However, in an educational context, the goal is not merely to reach a solution, but to ensure that students achieve the learning objectives. This goal is inherently different from prompt engineering for productivity improvement, which aims to minimize and eradicate failures, a proper learning strategy may require expecting or even encouraging a significant amount of purposeful failures along the way. Thus, prompt engineering research prioritizing individual learning, as seen in works like Jin *et al.* [17], should be further investigated.

Our results suggest that, even in the physical presence of a researcher and with knowledge of being recorded, in a considerable number of cases, the participants directly resorted to Generative AI to provide solutions to the tasks without making any attempts on their own. This undesirable way of help-seeking, if used as the default approach in solving homework problems, can negatively impact students' learning. This suggests the need for students to improve their self-regulation skills, e.g. self-monitoring [40] to monitor and reflect the intensity and frequency of their use of Generative AI. Similarly, Generative AI educational tool builders should consider supporting such strategies that enhance students' self-regulation and help-seeking behavior when they use Generative AI.

The findings of Margulieux *et al.* [27] are particularly relevant here. In their study, they found that some students used Generative AI to support and not replace their critical thinking and problem solving. However, they also noticed that the weakest students tended to use Generative AI earlier in the problem solving process. Their findings mirror those in our study. However, Margulieux *et al.*were somewhat optimistic that their findings meant that users who needed help and support could receive it. Our findings, as well as those by Prather *et al.* [37], show that lower-performing students using Generative AI

earlier in the problem solving process is likely an indication of overreliance and a failure to properly self-regulate with these tools.

One can imagine that a possible remedy for this problem is to devise novel homework assignments that can be difficult for Generative AI to solve. However, as Generative AI tools become more sophisticated and powerful, the arms race between educators and Generative AI seems to be a losing proposition, especially in introductory courses such as CS1 that only include basic algorithmic thinking that there is an upper limit to the appropriate complexity of problems [10, 34]. To discourage students from using prompts to find complete solutions to homework problems, we should investigate novel *types* of problems that are compatible with the era of Generative AI. Recent work such as Prompt Problems [8] that uses the graphical representation for problem description instead of a textual description is a step in this direction. Even though the newest models with visual modality have been able to solve Prompt Problems [15], they remain useful as a way to scaffold students usage of Generative AI by helping them learn problem decomposition, iterative problem solving, describing a problem, and prompt engineering [35].

An intriguing observation from our study was that most participants did not frequently execute their programs during development to verify the correctness or identify syntax errors. This lack of regular testing suggests a gap in their understanding of the importance of iterative debugging in the programming process. Consequently, students may miss out on early detection of mistakes, which can lead to more complex issues and frustration later in the development cycle. Addressing this behavior through new pedagogical tools such as [11] that visualizes the value of variables as students develop their solutions could improve students' coding practices and overall comprehension of programming concepts.

Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant No. 2225373. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Vincent Aleven and Kenneth R Koedinger. 2000. Limitations of student control: Do students know when they need help?. In *International conference on intelligent tutoring systems*. Springer, 292–303.

[2] Vincent Aleven and Kenneth R Koedinger. 2001. Investigations into help seeking and learning with a cognitive tutor. In *Papers of the AIED-2001 workshop on help provision and help seeking in interactive learning environments*. 47–58.

[3] Vincent Aleven, Elmar Stahl, Silke Schworm, Frank Fischer, and Raven Wallace. 2003. Help seeking and help design in interactive learning environments. *Review of educational research* 73, 3 (2003), 277–320.

[4] Matin Amoozadeh, David Daniels, Stella Chen, Daye Nam, Aayush Kumar, Michael Hilton, Mohammad Amin Alipour, and Sruti Srinivasa Ragavan. 2023. Towards Characterizing Trust in Generative Artificial Intelligence among Students. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 2*. 3–4.

[5] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Trust in Generative AI among students: An exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 67–73.

[6] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1 (2023), 85–111. https://doi.org/10.1145/3586030

[7] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[8] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (<conf-loc>, <city>Portland</city>, <state>OR</state>, <country>USA</country>, </conf-loc>) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 296–302. https://doi.org/10.1145/3626252.3630909

[9] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. *arXiv preprint arXiv:2405.14178* (2024). https://arxiv.org/abs/2405.14178

[10] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (jan 2024), 56–67. https://doi.org/10.1145/3624720

[11] Kasra Ferdowsi, Ruanqianqian Huang, Michael B James, Nadia Polikarpova, and Sorin Lerner. 2024. Validating AI-Generated Code with Live Programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–8.

[12] Amanda S. Fernandez and Kimberly A. Cornell. 2024. CS1 with a Side of AI: Teaching Software Verification for Secure Code in the Era of Generative AI. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)* (Portland, OR, USA). ACM, New York, NY, USA. https://doi.org/10.1145/3626252.3630817

[13] Cornelia Gräsel, Frank Fischer, and Heinz Mandl. 2000. The use of additional information in problem-oriented learning environments. *Learning Environments Research* 3 (2000), 287–305.

[14] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *arXiv* (2023). https://doi.org/10.1145/3568813.3600139 arXiv:2306.05715

[15] Irene Hou, Owen Man, Sophia Mettille, Sebastian Gutierrez, Kenneth Angelikas, and Stephen MacNeil. 2024. More Robots are Coming: Large Multimodal Models (ChatGPT) can Solve Visually Diverse Images of Parsons Problems. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA, 29–38. https://doi.org/10.1145/3636243.3636247

[16] Irene Hou, Sophia Mettille, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The Effects of Generative AI on Computing Students' Help-Seeking Preferences. In *Proceedings of the 26th Australasian Computing Education Conference*. 39–48.

[17] Hyoungwook Jin, Seonghee Lee, Hyungyu Shin, and Juho Kim. 2024. Teach AI How to Code: Using Large Language Models as Teachable Agents for Programming Education. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 652, 28 pages. https://doi.org/10.1145/3613904.3642349

[18] Gregor Jošt, Vasko Taneski, and Sašo Karakatič. 2024. The Impact of Large Language Models on Programming Education and Student Learning Outcomes. *Applied Sciences* 14, 10 (2024), 4115.

[19] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–23.

[20] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *arXiv* (2023). https://doi.org/10.1145/3544548.3580919 arXiv:2302.07427

[21] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2023. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–12.

[22] Sam Lau and Philip Guo. 2023. From" Ban it till we understand it" to" Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*. 106–121.

[23] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. *arXiv* (2023). arXiv:2304.03938

[24] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2023. Understanding the Usability of AI Programming Assistants. *CoRR* abs/2303.17125 (2023). https://doi.org/10.48550/arXiv.2303.17125 arXiv:2303.17125

[25] Raymond Liu, Chris Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. 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*. 750–756. https://cs.harvard.edu/malan/publications/V1fp0567-liu.pdf

[26] Rosemary Luckin, Benedict Du Boulay, et al. 1999. Ecolab: The development and evaluation of a Vygotskian design framework. *International journal of artificial Intelligence in Education* 10, 2 (1999), 198–220.

[27] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. In *Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. ACM, NY, USA.

[28] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *CoRR* abs/2210.14306 (2022). https://doi.org/10.48550/arXiv.2210.14306 arXiv:2210.14306

[29] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 97, 13 pages. https://doi.org/10.1145/3597503.3639187

[30] Sharon Nelson-Le Gall. 1981. Help-seeking: An understudied problem-solving skill in children. *Developmental review* 1, 3 (1981), 224–246.

[31] Richard S Newman. 2023. Adaptive help seeking: A strategy of self-regulated learning. In *Self-regulation of learning and performance*. Routledge, 283–301.

[32] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis)read Each Other. arXiv:2401.15232 [cs.HC]

[33] Paul R Pintrich et al. 1991. A manual for the use of the Motivated Strategies for Learning Questionnaire (MSLQ). (1991).

[34] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education* (Turku, Finland) *(ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, 108–159. https://doi.org/10.1145/3623762.3633499

[35] James Prather, Paul Denny, Juho Leinonen, David H Smith IV, Brent N Reeves, Stephen MacNeil, Brett A Becker, Andrew Luxton-Reilly, Thezyrie Amarouche, and Bailey Kimmel. 2024. Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models. *arXiv preprint arXiv:2401.10759* (2024).

[36] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (nov 2023), 31 pages. https://doi.org/10.1145/3617367

[37] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Melbourne, VIC, Australia) *(ICER '24)*. Association for Computing Machinery, New York, NY, USA, 469–486. https://doi.org/10.1145/3632620.3671116

[38] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces, IUI 2023, Sydney, NSW, March 27-31, 2023*. ACM, 491–514. https://doi.org/10.1145/3581641.3584037

[39] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations using Large Language Models. *arXiv* (2022). https://doi.org/10.48550/arxiv.2206.11861 arXiv:2206.11861

[40] Dale H Schunk and Barry J Zimmerman. 2012. Self-regulation and learning. *Handbook of Psychology, Second Edition* 7 (2012).

[41] Sruti Srinivasa Ragavan and Mohammad Amin Alipour. 2024. Revisiting Human Information Foraging: Adaptations for LLM-based Chatbots. *arXiv e-prints* (2024), arXiv–2406.

[42] Xin Tan, Xiao Long, Xianjun Ni, Yinghao Zhu, Jing Jiang, and Li Zhang. 2024. How far are AI-powered programming assistants from meeting developers' needs? *arXiv preprint arXiv:2404.12000* (2024).

[43] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The Metacognitive Demands and Opportunities of Generative AI. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 680, 24 pages. https://doi.org/10.1145/3613904.3642902

[44] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 297–303. https://doi.org/10.1145/3649217.3653584

[45] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022, Extended Abstracts*. ACM, 332:1–332:7. https://doi.org/10.1145/3491101.3519665

[46] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE Code Generation from Natural Language: Promise and Challenges. *ACM Trans. Softw. Eng. Methodol.* 31, 2 (2022), 29:1–29:47. https://doi.org/10.1145/3487569

[47] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT Help With Introductory Programming?An Experiment of Students Using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (Lisbon, Portugal) *(ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 331–341. https://doi.org/10.1145/3639474.3640076

[48] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 21–29.

[49] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, Swarat Chaudhuri and Charles Sutton (Eds.). ACM, 21–29. https://doi.org/10.1145/3520312.3534864