Exploring Student Behaviors and Motivations when using AI Teaching Assistants with Optional Guardrails

Amanpreet Kapoor
University of Florida
Gainesville, FL, USA
kapooramanpreet@ufl.edu

Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Leo Porter
University of California San Diego
La Jolla, CA, USA
leporter@ucsd.edu

Stephen MacNeil
Temple University
Philadelphia, PA, USA
stephen.macneil@temple.edu

Marc Diaz
University of Florida
Gainesville, FL, USA
marcgabe15@gmail.com

Abstract

AI-powered chatbots and digital teaching assistants (AI TAs) are gaining popularity in programming education, offering students timely and personalized feedback. Despite their potential benefits, concerns about student over-reliance and academic misconduct have prompted the introduction of "guardrails" into AI TAs—features that provide scaffolded support rather than direct solutions. However, overly restrictive guardrails may lead students to bypass these tools and use unconstrained AI models, where interactions are not observable, thus limiting our understanding of students' help-seeking behaviors. To investigate this, we deployed a novel AI TA tool with optional guardrails in one lab of a large introductory programming course. As students completed three code writing and debugging tasks, they had the option to receive guardrailed help or use a "See Solution" feature which disabled the guardrails and generated a verbatim response from the underlying model. We investigate students' motivations and use of this feature and examine the association between use and their course performance. We found that 50% of the 885 students used the "See Solution" feature for at least one problem and 14% used it for all three problems. Additionally, low-performing students were more likely to use this feature and use it close to the deadline as they started assignments later. The predominant factors that motivated students to disable the guardrails were assistance in solving problems, time pressure, and lack of self-regulation. Our work provides insights into students' solution-seeking motivations and behaviors, which has implications for the design of AI TAs that balance pedagogical goals with student preferences.

CCS Concepts

• **Social and professional topics → Computing education**.

Keywords

AI tutor, Feedback, Digital TAs, Automated Tutors, Programming

ACM Reference Format:
Amanpreet Kapoor, Paul Denny, Leo Porter, Stephen MacNeil, and Marc Diaz. 2026. Exploring Student Behaviors and Motivations when using AI Teaching Assistants with Optional Guardrails. In *28th Australasian Computing Education Conference (ACE 2026), February 09–13, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3786228.3786233

1 Introduction

Providing students with timely and high-quality help is essential in programming courses. However, scaling the support that can be provided by human teaching assistants (TAs) is difficult in large classes where the needs of students seeking help can vary considerably. Thus, there has been increasing interest in the use of AI TAs for providing equitable support to learners in introductory courses [34]. Recent work has shown that such tools can generate responses that are both correct and helpful [6], provide students with the sense of having a personal tutor [24], and produce timely, tailored feedback [22]. However, the growing adoption of AI TAs has also raised concerns such as inaccuracies in the AI-generated content [7], student over-reliance on such tools [7, 43] which can potentially compound metacognitive difficulties [35], and the risk that students may generate and copy solutions instead of learning concepts [20].

To address concerns about over-reliance and misuse, researchers have explored incorporating 'guardrails' into AI TAs to provide scaffolded support that encourages learning rather than offering direct answers [6, 16, 22]. While students often appreciate such safeguards, they may also seek direct solutions when under time pressure, highlighting the complex dynamics of help-seeking behaviors [19, 23]. If the guardrails are perceived as overly restrictive, students may turn to unconstrained tools like ChatGPT [33], which lack pedagogical safeguards and instructor oversight and can undermine learning goals. Understanding how students interact with and value these systems is thus critical to designing digital TAs that balance educational objectives with student preferences.

To explore student-interactions with guardrailed AI TAs further, we conducted an observational study in a large introductory programming course. We built and deployed a novel AI TA with *optional* guardrails in our Edugator tool [8], and asked students to complete three programming tasks—focused on code writing and debugging—while interacting with the AI TA. Students could

---

This work is licensed under a Creative Commons Attribution 4.0 International License.

ACE 2026, Melbourne, VIC, Australia

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2352-0/26/02
https://doi.org/10.1145/3786228.3786233

choose whether to use the guardrailed version, which would not reveal code solutions, or disable the restrictions via a "See Solution" feature to generate a verbatim response from the underlying model which may include a code solution. Once they had completed the tasks, the students provided feedback about the tool and described their motivations for using (or not using) the "See Solution" feature. We investigate interaction patterns and performance based on the use of this feature, guided by the following research questions:

RQ.1a To what extent is the "See Solution" feature used, and how does this usage relate to student performance in the course?
RQ.1b How does the timing of student engagement with the lab tasks relate to the use of the "See Solution" feature?
RQ.2 What factors motivate students to use or refrain from using the "See Solution" feature when engaging with an AI TA?

Understanding these student behaviors and motivations can guide the development of AI TAs that provide meaningful support, encourage academic integrity, and help students achieve desired learning outcomes.

2 Related Work

2.1 Help-Seeking in Computing Courses

Effective help-seeking is an important metacognitive skill [2], especially for novice programmers who can obtain help in many ways, but may engage in unproductive behaviors like avoiding or over-relying on help [30, 41]. Human TAs have traditionally provided front-line support when students are stuck, although prior research has revealed mismatches between student expectations and educational goals during help-seeking interactions with TAs. In a recent study by Krause-Levy et al., researchers observed that TAs frequently provide solutions during help sessions rather than guiding students through the problem-solving process [19]. Similarly, Villegas-Molina et al. interviewed TAs and found that TAs sometimes report giving away solutions when faced with a long line of students seeking help as doing so takes less time [44]. Regarding students seeking assistance, Lim et al. interviewed students and found that under the pressure of task deadlines, they often prioritize completing tasks over mastering concepts, and therefore sometimes prefer receiving direct answers [23]. These findings suggest that both students and TAs seek or provide direct solutions due to situational factors such as time pressure and workload.

In addition, it can be challenging for tutors to spend the necessary time with each student who seeks help when handling multiple requests at busy times [21, 28, 38]. This can be exacerbated by student procrastination, which is a well-known barrier to success in programming courses [9, 17]. Students who delay starting assignments not only risk poorer academic outcomes but also leave insufficient time to seek meaningful help [29, 46]. The increased demand for help close to deadlines can place a significant strain on TAs, who are already limited in their availability outside typical working hours. Moreover, not all students feel equally confident to seek help in-person [42]. These challenges have driven interest in more consistent, scalable, personalized, and private support options, such as digital AI TAs [6, 34].

2.2 Emergence of AI-Powered Assistants

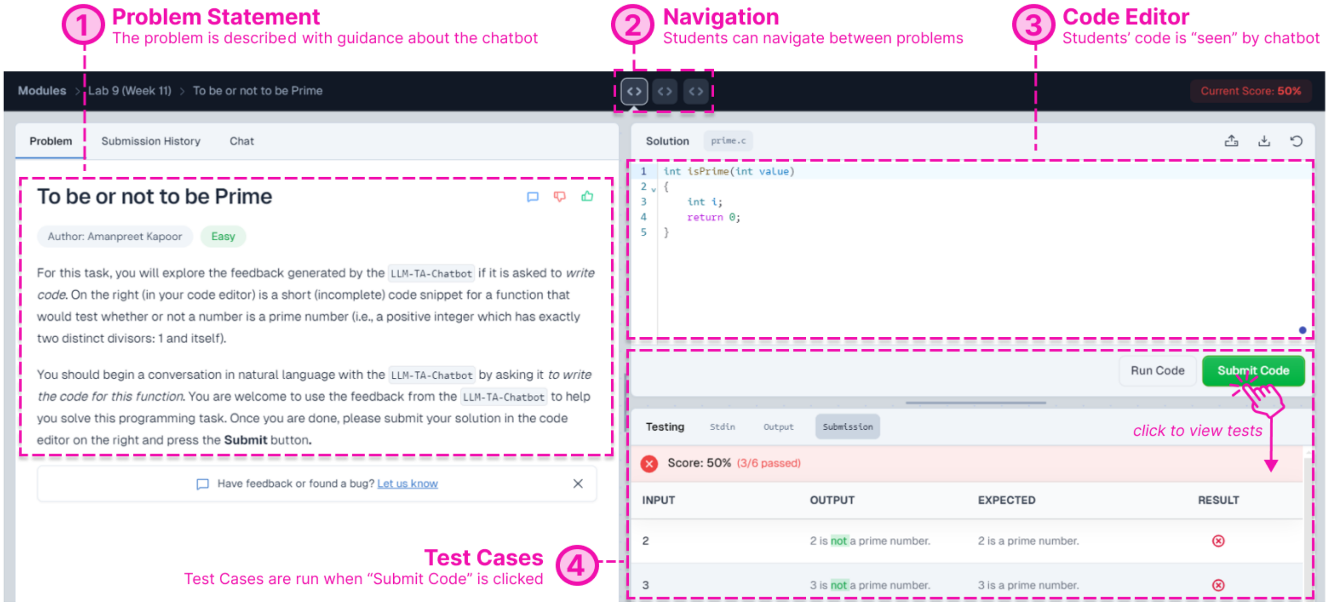
The rapid improvement in capabilities of large language models (LLMs) has enabled the creation of AI-powered digital teaching assistants (AI TAs). Initial work in this space evaluated the quality of responses produced by open-source and proprietary LLMs to historical repositories of student questions, finding that LLMs can provide rich meaningful feedback to students and can solve a variety of programming problems [12, 13, 18]. More recently, tools have been developed with integrated LLM support and deployed to answer student queries in real-time [24, 32]. However, ensuring that responses are pedagogically sound remains a challenge. Qiao et al. address this by providing instructor oversight of the responses generated by the LLM, but this limits scalability [37]. More commonly, AI-powered tutoring tools are being designed to provide appropriate help automatically so that students can call on them when needed [26]. It is also worth noting that as model capabilities advance, questions about pedagogical quality continue to be examined. Recent work by Brown et al. has shown that, when paired with carefully designed prompts, LLMs can generate next-step programming hints that are judged by experienced educators to be pedagogically superior to those written by human experts [5].

A common approach to prevent misuse when designing AI TAs is the use of 'guardrails'. Examples are implemented in tools like CodeHelp [22] and CodeAid [16] which offer scaffolded support instead of producing direct code solutions. Interestingly, students often say they prefer such scaffolded tools [6]. *However, if the guardrails in an AI TA are perceived as overly restrictive, students may turn to unconstrained LLMs like ChatGPT, which lack pedagogical safeguards.* Such interactions are not monitored, and thus valuable insights into student learning behaviors are lost. Understanding how students interact with and value AI TAs with or without guardrails is essential for addressing these challenges and ensuring their integration meets learning objectives.

2.3 Over-Reliance on GenAI

One of the most commonly cited concerns regarding GenAI use in CS education is that of student over-reliance [7, 34, 43]. Tools like ChatGPT and GitHub Copilot can generate detailed solutions to many programming-related tasks, which may undermine the development of problem-solving and computational thinking skills [27, 45, 47]. Lau and Guo warn that such reliance can harm foundational learning, while Sheard et al. highlight the challenges GenAI poses to maintaining academic integrity [20, 40]. In addition, students themselves recognize the risks of over-reliance. Hou et al. found that while students value the convenience of GenAI, they acknowledge that improper use can harm independent learning [15]. Despite this, GenAI tools are frequently used. Nearly a quarter of surveyed students reported using tools like ChatGPT daily or hourly, often for debugging and code generation [15].

Within the context of general education, Zhang et al. identified factors associated with students' over-reliance on GenAI tools such as ChatGPT, finding that academic self-efficacy influences AI dependency, and this relationship is mediated by academic stress and performance expectations [47]. This suggests that psychological factors, such as stress and low self-efficacy, can increase students' reliance on GenAI tools. However, literature examining how these



The problem is described with guidance about the chatbot

Students can navigate between problems

Students' code is "seen" by chatbot

Test Cases are run when "Submit Code" is clicked

To be or not to be Prime

Author: Amanpreet Kapoor Easy

For this task, you will explore the feedback generated by the `LLM-TA-Chatbot` if it is asked to *write code*. On the right (in your code editor) is a short (incomplete) code snippet for a function that would test whether or not a number is a prime number (i.e., a positive integer which has exactly two distinct divisors: 1 and itself).

You should begin a conversation in natural language with the `LLM-TA-Chatbot` by asking it *to write the code for this function*. You are welcome to use the feedback from the `LLM-TA-Chatbot` to help you solve this programming task. Once you are done, please submit your solution in the code editor on the right and press the **Submit** button.

Have feedback or found a bug? Let us know

```
int isPrime(int value)
{
    int i;
    return 0;
}
```

Score: 50% (3/6 passed)

INPUT	OUTPUT	EXPECTED	RESULT
2	2 is not a prime number.	2 is a prime number.	⊗
3	3 is not a prime number.	3 is a prime number.	⊗

Figure 1: Interface for Edugator Tool and Lab Activity 1: *isPrime* Code Writing Task

factors manifest in computing education is scarce, highlighting the need for the community to investigate how and why students over-rely on AI tools in CS education.

The risks associated with over-reliance grow as GenAI tools advance. Recent studies have demonstrated that tools like GPT-4 can solve a wide range of programming problems, including visual challenges like Parsons Problems and computer graphics tasks [10, 11, 14]. While this capability can accelerate problem-solving for advanced students, Prather et al. caution that struggling students may adopt "shepherding" behaviors, relying heavily on AI without meaningful engagement [35]. This reliance may create an illusion of competence, leaving essential skills underdeveloped.

The challenge lies in balancing the need for pedagogical safeguards with the flexibility to meet students' immediate help-seeking preferences. In this study, we examine student interactions with an AI TA that incorporates scaffolded support through guardrails. It also includes a feature that allows students, when needed, to remove these restrictions to access unfiltered LLM-generated responses. This dual approach aims to balance pedagogical safeguards with student preferences, and provide insight into how such features influence help-seeking behaviors.

3 Methods

3.1 Study Design and Context

We designed an observational study to investigate the behaviors and motivations of computing students' when interacting with an AI TA chatbot with optional guardrails. Our study was conducted in the context of a large introductory programming course offered at a public research university in the Australasian region in Fall 2024. Data collected in this study was covered by an ethics approval for the archival analysis of naturally occurring coursework data.

A total of 1,034 students were enrolled in the course, which spanned a 12-week teaching term. Students completed weekly lab exercises, each contributing 1% toward their final grade. Data for this study was collected in one of the weekly labs (described in Section 3.3) where students were asked to complete one code writing task and two code debugging tasks (using the C language), and complete two quantitative and two qualitative questions. This lab was graded as part of the coursework. The lab tasks were hosted on our novel tool (described in Section 3.2) which offered students access to an AI chatbot with optional guardrails. The three tasks used in this study were similar to the ones used in a previous study on evaluating the efficacy of AI TAs with guardrails on students' learning outcomes [6]. However, the previous study used a different tool with mandatory guardrails — no "See Solution" feature — and a distinct mode of student interaction with the AI assistant. In the current study, the tool provided a conversational chatbot interface that had access to the student's code in the editor, whereas the previous study relied on a one-shot AI response after students manually pasted their code, error, and prompt in the tool's interface.

All enrolled students were sent an invitation to participate (909 students accepted the invitation to the tool; 885 participated in at least one of the three programming tasks by submitting code to the system). In this paper, we analyze data collected from the 885 participating students.

3.2 GenAI Coding Tool

We developed the Edugator web-based GenAI-enabled coding tool [1, 8] that allows instructors to write programming problems and offers students a clean interface to preview the problem description, code their solution in the browser, and receive immediate feedback on their solutions (see Figure 1). Additionally, the tool provides access to an LLM-powered AI TA (see Figure 2).

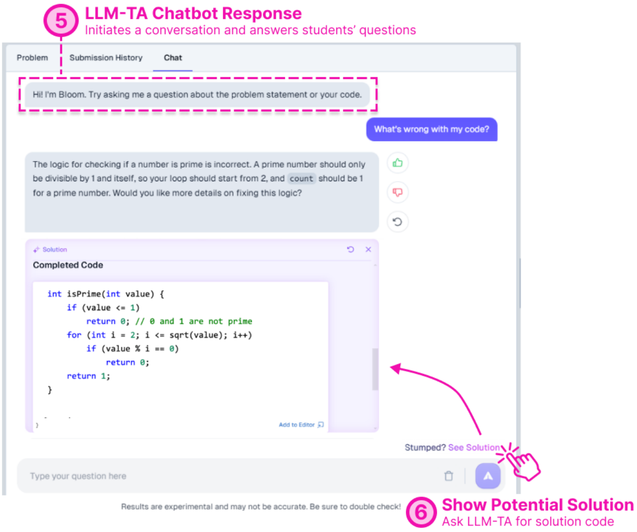
AI TA with guardrails. Our AI TA chatbot employs guardrails by way of a prompt and had access to a student's chat prompt, chat history, course programming language, and their solution in the code editor. The prompt used to generate a response to a student's query was similar to the one used in CodeHelp [22] with a few minor changes to support a conversational interaction:

> Respond to the student with a brief educational explanation, helping the student figure out the issue and understand what they're doing incorrectly. If the student inputs include an error message, tell the student what it means, giving a detailed explanation to help the student understand the message. Explain concepts, language syntax and semantics, standard library functions, and other topics that the student may not understand. Be positive and encouraging, and keep it conversational, meaning try to push the student in the right direction before outright explaining everything.

In the same prompt, instructions were added to support guardrails and prevent the LLM from providing a student with any example code snippets, a solution, or a corrected version of their code (part of the prompt is omitted for brevity — signified using [...]):

> If the student's issue requests code, tell them you cannot provide any code. [...] Do not write any example code blocks. [...]

The prompt used for AI TA with guardrails was not fed the problem description. This design was intended to encourage students to develop prompting skills by providing sufficient context to the AI TA chatbot when conversing with it.


⑤ LLM-TA Chatbot Response
Initiates a conversation and answers students' questions
Problem
Submission History
Chat
Solution
Completed Code
Add to Editor
Stumped? See Solution
Type your question here
⑥ Show Potential Solution
Ask LLM-TA for solution code

Hi! I'm Bloom. Try asking me a question about the problem statement or your code.

What's wrong with my code?

The logic for checking if a number is prime is incorrect. A prime number should only be divisible by 1 and itself, so your loop should start from 2, and `count` should be 1 for a prime number. Would you like more details on fixing this logic?

```
int isPrime(int value) {
    if (value <= 1)
        return 0; // 0 and 1 are not prime
    for (int i = 2; i <= sqrt(value); i++)
        if (value % i == 0)
            return 0;
    return 1;
}
```

Results are experimental and may not be accurate. Be sure to double check!

Figure 2: Chat interface to ask questions and solicit solution

"See Solution" feature — no guardrails. To evaluate solution-seeking behaviors for the purposes of this study, we added a new feature to the AI TA where students have the option to press the "See Solution" button after initiating the chat which allows them to deactivate the guardrails to see LLM generated-solution code along with the scaffolded response (see Figure 2). The solution code was not preset and was instead dynamically generated using a prompt to show the AI-TA's capability to produce contextualized solutions on demand rather than relying on fixed, instructor-authored answers.

The button was labeled "See Solution" so that students were aware that the feature can generate a solution, as they were unlikely to be familiar with the 'guardrail' terminology used in the literature. The prompt fed to retrieve the solution, asked the LLM to assume that it is *"a coding expert tasked with providing solutions to C programming problems"*. Further, the model was instructed to provide the student with a detailed explanation of the problem, including the correct code solution using the following prompt:

> Your goal is to provide a detailed, educational explanation of the problem, including the correct code structure and logic. Your responses should be concise, clear, and easy to understand. Ensure that the solution is accurate, follows best practices for the given programming language, and leverages the provided template code. [...] Your response should include:
> - Explanations for any non-trivial code or logic used, including why certain approaches were chosen.
> - Best practices related to the solution (e.g., code efficiency, readability, and maintainability).
> - A clear and well-documented solution to the problem, starting from the provided template code.
> - A solution that takes into account edge cases and challenges discussed with the code assistant.

The problem statement was also sent to this prompt in addition to the aforementioned attributes that were used for the AI TA with guardrails.

Our AI TA was backed by OpenAI's 4o model and the total cost of LLM usage during the study (9 days) was US$53.00.

3.3 Lab Activities

For all lab tasks, the course used the CodeRunner platform [25]. For this study, which was run as part of "Lab 9" in the course, students were provided some instructions within CodeRunner as well as a link to the Edugator tool [8]. The instructions on CodeRunner stated that in order to earn the 1 mark (out of the lab's 10 marks) for the task, they needed to solve the three problems in the Edugator tool. The remaining marks were for other programming problems that are not relevant for this study. All programming tasks corresponded to the course content covered in Week 9 (primarily targeting nested loops and two-dimensional arrays). Additionally, a lab instructions handout provided an overview of the Edugator system [8], including system navigation, how to test code, converse with the AI-TA chatbot, and viewing potential solutions via the "See Solution" button (similar to Figure 1 and 2). A figure (similar to Figure 2) in this handout explained to the students that,

> The chatbot will respond to your questions helping you approach a problem without giving you the solutions. In case you want to see potential solution code, you can click "See Solution".

There were no penalties for using the "See Solution" feature. All students saw the instructions on CodeRunner and were expected to read the lab handout prior to accessing the Edugator tool. To avoid students being overly cautious about using a feature labelled "See Solution", several approaches were used to communicate that its use was explicitly allowed. Edugator was demonstrated in class, providing an overview of the tool and its features. Attendance in lectures was optional, but all lectures were recorded and available online. So every student had access to a recording of this demonstration. During this demonstration, it was made clear to students that they were welcome to make use of any of the features of Edugator, including the "See Solution" feature. Students were informed that both the Chatbot and the "See Solution" feature were powered by an AI model, and that if students chose to use the "See Solution" feature they could do so without any penalties and that they were *"welcome to make use of the generated solution if they wanted to"*.

In addition, as part of this demonstration, students were shown the reflection questions they were going to be asked after using Edugator. This included a question about their choice regarding whether or not to use the "See Solution" feature and the extent to which they found it useful. Within the Edugator tool itself, for each problem description, students could see messages such as: *"You are welcome to use the feedback from the LLM-TA Chatbot to help you solve this programming task"*. It appears that the messaging around allowed use of the "See Solution" feature was broadly successful. For one thing, it was widely used by many students in the course. In addition, we observed just one question on the class forum that explicitly asked whether usage of this feature would be penalised.

For context, the study took place in the second half of 2024, around a year and a half after ChatGPT became public. Students were generally discouraged from using tools like ChatGPT which were not designed for learning purposes. Instead, several custom AI-powered teaching tools like Edugator were woven throughout the course. Such tools included CodeHelp (a guardrailed question and answer tool that students could use for open questions when working on their project) [22], and Prompt programming (a tool where students practice writing code-generating prompts for solving programming tasks) [36]. In general, the expectations in the course were that students would make use of the tools that were provided to them by the course instructors, such as Edugator, but that they should not rely on unconstrained tools such as ChatGPT.

As with all labs in the course, given the large cohort size, the lab was run over a period of one week. Students were free to complete the lab at any time during that week (online), and they could also attend an in-person lab (which is optional) that was scheduled throughout the week, if they wanted assistance from a human TA.

In addition to the three programming tasks that are used in this study, students were also asked to complete two open-ended and two quantitative reflection questions that were configured as separate questions within CodeRunner. Completing these questions allowed students to get 1 mark (out of the 10 marks). These questions were ordered to appear after the CodeRunner question that linked to the Edugator tool. It would be possible for students to skip ahead to the reflection question without completing the Edugator tasks. However, given the high number of students that both completed the coding task on Edugator (n=885) and gave an open reflection response (n=839), we expect that most students did these in order (there is not much benefit or incentive for students to do them out of order). We now briefly describe the three activities.

Activity 1: IsPrime (Code Writing). Activity 1 required students to finish the incomplete *IsPrime()* function provided (as shown in Figure 1). Students were prompted to *"begin a conversation in natural language with the LLM TA Chatbot by asking it to write the code for this function"*. Such a request would not result in a direct code solution, due to the presence of the guardrails. For instance, a sample interaction between the AI TA and a student (S251) for this lab activity is shown in Table 1.

User	Chat Prompt and Response
Student (S251)	write a function so that it returns 1 if the input is a prime number and 0 otherwise
AI TA	I'm sorry, but I can't write the function for you. Do you have any specific questions about how to check for prime numbers?
Student (S251)	how should i check in C if a number is prime
AI TA	To check if a number is prime in C, you'll need to iterate from 2 up to the square root of the value and check for divisibility. If the value is divisible by any number in this range, it's not a prime. Do you need help with how to implement the iteration and the condition check?
Student (S251)	What should my if statement look like
AI TA	Your loop should go up to `i <= sqrt(value)` instead of `i < sqrt(value)`. Also, you should return 0 if a divisor is found and 1 if none are found. Is this what you were looking for?

Table 1: Transcript of a student conversing with the AI TA for Activity 1

Activity 2: IsRepeated (Code Debugging). Activity 2 provided a buggy implementation of an *IsRepeated()* function, which should return true if a one-dimensional array contains any repeated values. A nested loop was incorrectly defined by initialising the inner loop variable (j) to be equal to ($= i$) rather than larger than ($= i + 1$) the value of the outer loop variable. Students were asked to: *"Copy the function (which has a bug) and provide it as input to the TA chatbot. You should construct a short natural language description explaining what the function intends to do and asking it to debug the code"*. Students were encouraged to follow this step, even if they could see the bug, so that they could view and critique the chatbot output.

Activity 3: SurroundingSum (Code Debugging). Activity 3 involved computing the sum of all values surrounding a specified location in a two-dimensional array. The provided code contained a bug by failing to guard against out-of-bounds array accesses in the case that the location was on the border of the array. Similar instructions were provided as for Activity 2.

When completing all three activities, students were instructed: "*Once you are done, please submit your solution (with the fixed code) in the code editor on the right and press the 'Submit' button*". The 'Submit' button would run their code against a test suite created by the instructor and would give them immediate feedback on whether they passed test cases as well as show their score for the problem (see Figure 1). Students could press the 'Submit' button any number of times and their last submission was used for grading purposes.

Of the 885 students who participated in at least one of the three activities, 865 students completed and passed all test cases for Activity 1 (IsPrime), 865 students completed and passed all test cases for Activity 2 (IsRepeated), and 850 students completed and passed all test cases for Activity 3 (SurroundingSum).

3.4 Data Collection and Analysis

3.4.1 Quantitative Data. We analyze log data of student interactions with our tool, and summarize the activity in two ways: for how many of the three lab problems the "See Solution" feature was used, and how the overall use of the tool was distributed over the 9 day period from release of the lab until the deadline. The actions of interest were: pressing the "See Solution" button, and making a code submission (either a compilation request with a user defined input using the "Run" button or a request to run code against the instructor test suite using the "Submit" button). We also calculate the course mark for each student based on the invigilated assessments in the course (i.e., the average of three proctored pen-and-paper exams: two midterms and one cumulative exam), and use this score as a proxy for their course performance. Further, we grouped students into quartiles for the purpose of our analysis similar to Salguero et al's study [39] (see Table 2). Data analysis was conducted in Microsoft Excel.

Quartile	No. of students (N=885)	Score Range in the Course
Q_1	227	[7, 59]
Q_2	224	(59, 75]
Q_3	223	(75, 86]
Q_4	211	(86-99]

Table 2: Student grouping into quartiles based on performance in invigilated assessments (Q_1=low, Q_4=high)

3.4.2 Qualitative Data. We asked students an open-ended question after they completed the three lab tasks: *If you used the "See Solution" feature to generate a code solution for any of the three problems, explain your rationale for using this feature. Reflect on the usefulness of this feature and the extent you used the generated solution in your final submission for the respective problem. Alternatively, if you did not use this "See Solution" feature, comment on why you didn't use it.*

A total of 839 non-blank responses (completion rate: 95%) to this question were analyzed by a single researcher using a reflexive thematic analysis approach [4]. The process involved 1) the researcher familiarizing themselves with the responses, 2) open-coding the responses, and 3) iteratively identifying themes. The themes were selected based on their importance and relevance to the research question. Rather than focusing on coder agreement or inter-rater reliability (which are not appropriate for the reflexive nature of thematic analysis [31]), our process ensures validity by reflexively engaging with the data by revisiting it multiple times. The researcher actively and iteratively constructed their understanding of the responses, rather than seeking an objective 'truth' in the data. Finally, to exemplify and contextualize the themes and improve reliability, we present representative examples from the students' responses.

4 Results

4.1 Quantitative Analysis

4.1.1 Solution-seeking and course performance. Students clicked the "See Solution" button 1049 times in total across the three problems (ranging from 0 to 10 clicks per student, as some students pressed the button more than once for a problem). We observed that 50% of the 885 students (n=445) did not use the "See Solution" feature for any of the three problems (see Figure 3). Of the remaining 50% of students who used the feature at least once (n=440), 36% (18% overall) used the feature for one of the problems (n=160), 36% (18% overall) used the feature in two of the problems (n=158), and 28% (14% overall) used it in all three problems (n=122). Across problems, 39% of students used the feature in the *IsPrime* problem (n=349), 17% used it in the *IsRepeated* (n=150), and 38% students used it to see the solution to the *SurroundingSum* problem (n=339).

Students who used the "See Solution" feature for all three problems scored, on average, lower on the invigilated tests ($\mu_3 = 66$) compared to students who did not use the feature ($\mu_0 = 73$) or used it to see solutions to one or two of the problems ($\mu_1 = 72$, $\mu_2 = 73$). Since the performance data was not normally distributed, a Kruskal-Wallis H test was conducted to test the null hypothesis—*the median invigilated scores are the same across the four groups of students based on their usage of the "See Solution" feature.* This test revealed that there is a significant difference in the average invigilated scores among the four groups (those who did not use the feature and those who used it for one, two or three problems), $\chi^2(3) = 14.4$, p = .002.

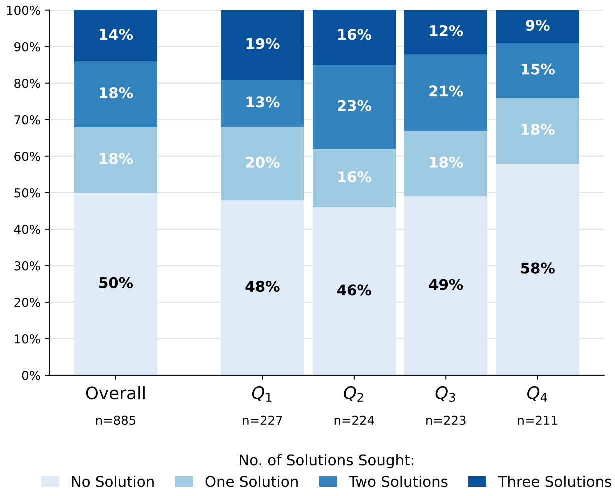


Figure 3: Usage of "See Solution" feature by course performance quartile (Q_1=low, Q_4=high)

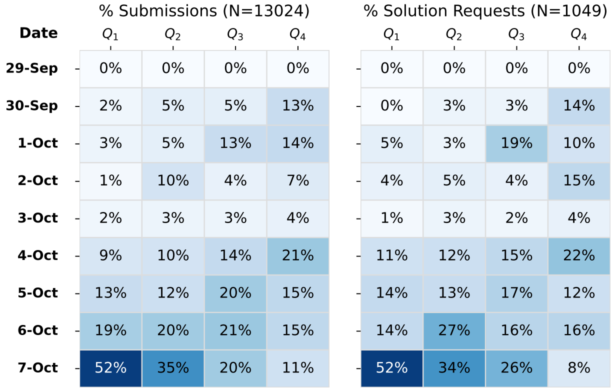
Figure 3 shows the frequency of use of the "See Solution" feature across performance quartile (Q_4 = highest performing). Students in Q_4 were more likely to *not* use the "See Solution" feature and were less likely to use it in all three problems compared to students in lower quartiles (Q_1, Q_2, and Q_3). For instance, a student in Q_1 was twice as likely to use it in all problems than a student in Q_4.

4.1.2 Usage of "See Solution" feature over time. We observed that many students start the lab close to the deadline as 29% of the students (n=258) used the Submit/Run button to test their code in our system for their first lab task on the same day as the deadline (11:59pm, 7th Oct, see Table 3). However, their decision to use the "See Solution" feature was independent of when they started the lab, as approximately 50% of the students would use the feature regardless of when they started working on the tasks ($\chi^2(7) = 5.92$, $p = 0.55$ indicating no significant differences between students who did and did not use the "See Solution" feature across start dates). One exception to this were students who started the lab very early (as one student who started the lab on Sep 29 and 65% of students who started on Sep 30 did not use the solution feature).

Used "See Solution"?	Date when a student started the lab (Deadline: 7-Oct)							
	30-Sep n=54	1-Oct n=88	2-Oct n=47	3-Oct n=18	4-Oct n=124	5-Oct n=125	6-Oct n=169	7-Oct n=258
No	65%	50%	49%	50%	46%	48%	51%	50%
Yes	35%	50%	51%	50%	54%	52%	49%	50%

Table 3: "See Solution" feature usage over time (N=883)[1]

We also computed the interaction between student performance, student code submission activity on our system, and "See Solution" feature usage. Figure 4 shows a heatmap of this interaction. We observe that 52% of the submissions and "See Solution" clicks by students in the lowest quartile, Q_1 were within the last day before the assignment was due and this behavior close to the deadline gradually decreased from Q_1 to Q_4.



Date	% Submissions (N=13024)				% Solution Requests (N=1049)			
	Q_1	Q_2	Q_3	Q_4	Q_1	Q_2	Q_3	Q_4
29-Sep	0%	0%	0%	0%	0%	0%	0%	0%
30-Sep	2%	5%	5%	13%	0%	3%	3%	14%
1-Oct	3%	5%	13%	14%	5%	3%	19%	10%
2-Oct	1%	10%	4%	7%	4%	5%	4%	15%
3-Oct	2%	3%	3%	4%	1%	3%	2%	4%
4-Oct	9%	10%	14%	21%	11%	12%	15%	22%
5-Oct	13%	12%	20%	15%	14%	13%	17%	12%
6-Oct	19%	20%	21%	15%	14%	27%	16%	16%
7-Oct	52%	35%	20%	11%	52%	34%	26%	8%

Figure 4: Heatmap of "See Solution" feature usage rate and submission rate *w.r.t.* student performance (N=884[2], Q_1=low, Q_4=high)

These results suggest that low-performing students procrastinate more compared to high-performing students, but the decision to use the "See Solution" feature does not seem to significantly alter with the timing of when students start their labs. Thus, based on descriptive patterns in the aggregate log data, using the "See Solution" feature appears largely independent of procrastination, although procrastination is related to student performance.

4.2 Qualitative Analysis

4.2.1 Reasons not to view solutions. Thematic analysis of open response data (n=839) revealed a variety of reasons why participants chose not to use the "See Solution" feature. These reasons included expressing an appreciation for self-directed problem-solving, not needing support, or ethical concerns.

Perceived learning value and sense of accomplishment. Most of the students who refrained from using the solution feature did so because of the intrinsic value they associated with independently solving the problems and perceived learning benefits. For example, P36 emphasized the importance of preserving learning experience:

> *"I did not use the see solution tool because I find that when you see an already completed solution, it takes away the learning experience that comes from crafting a unique solution yourself." - P36*

Similarly, P1018 shared this concern, *"I did not, as I think that would take away from my learning."* From a constructivist perspective, the prevalence of students wanting to build their own understanding by working independently is promising.

Not needing help. Another theme identified from the students' responses was that the guardrailed chatbot already provided sufficient support with feedback, debugging support, and helped them identify missing edge cases. For those cases, students claimed that the feature was not needed and therefore was not used. For example:

> *"I did not use the see solution feature as I felt the chat bot offered more than enough assistance for each of the problems." - P137*

Many students described how the chatbot allowed them to quickly develop their understanding and debug any problems that arose. For example, P99 stated that the *"chatbot was helpful in understanding the code and getting to the root of the issues really quickly."* For many students like P99 and P137, this level of support was already sufficient, with one student, P15, describing the use of "See Solution" feature as equivalent to a 'cheat code'.

Ethical concerns. While we explicitly permitted the use of the chatbot and the "See Solution" feature, some students described using the solutions as cheating:

> *"I chose not to use the 'See Solution' feature as it felt a bit like cheating and I wanted to figure it out on my own. Overall, once I learnt how to ask the [chatbot] the right questions, I was able to figure it out much more easily." - P400*

> *"I think seeing the solution may be a step to (sic) far for learning as it allows users to cheat their way through without actually using the hints to fix the code. The hints were really good for situations where users don't know how to proceed." - P611*

Interestingly, students seemed to feel less concerned about interacting with the chatbot with guardrails. P683 describes how using the chatbot was not 'cheating' because it didn't reveal the code, but they equated the "See Solution" feature to cheating:

> *"I think [the chatbot] is generally good, because it don't (sic) show any code so technically you can't use the AI to get the answer quickly. However there is no restriction with see solution feature so it's still easy to cheat" - P683*

4.2.2 Reasons to view solutions. Participants who used the "See Solution" feature described various motivations, including problem-solving assistance, time pressure, curiosity, and lack of self-regulation skills. These themes suggest both cognitive and metacognitive factors that influence students' decision-making.

---

[1] Two students who did not use the solution feature were excluded in the table as they started the lab on Sep 29 and an hour after the Oct 7 deadline.

[2] One student who did not use the solution feature was excluded in the heatmap analysis as they started the lab an hour after the Oct 7 deadline.

Problem-solving assistance. The most common reason for using the feature was to seek help when students were stuck or needed to verify their work. Several students described the concept of using the solution to work backward and understand the process:

> *"This feature was used to reveal what the true answer is. It helped in another way by showing the answer so I can work backwards." - P308*

Others used the feature as a last resort when they were stuck after multiple attempts to solve the problem independently or with the chatbot. These behaviors align with models of self-regulated learning, where students monitor their performance and adjust their strategies by seeking help when needed. Similarly, others used it as a spark of inspiration to get unstuck:

> *"If the chat bot wasn't helpful or I wasn't getting it, I used the solution for inspiration. Solution wasn't always accurate tho." - P978*

Finally, students talked about using the solutions to verify their own process. A few students talked about how this saved them time by confirming they were on the right track. Others talked about how seeing the solution could be incorporated into their problem-solving process. P801 described their verification process:

> *"... after asking the bot 5 to 6 questions, it does give me an opportunity to check and see if my thought process was indeed in the right direction. I would definitely say that using the solution straight away would defeat the purpose of such a learning platform." - P801*

Time pressure. Time constraints emerged as a recurring theme in students' decisions to view the solution. Participants cited tight deadlines and competing priorities as reasons for bypassing exploratory learning and opting for the quickest path to completion:

> *"I am submitting this lab quite late so I did not really have time to completely rewrite and debug the code myself so I clicked see solution. [However, without the time pressure] I would genuinely give it a go before wanting to peak at the see solution option." - P277*

Students also used it to save time or to avoid wheel spinning [3]:

> *"I used the See Solution ... because I wasn't sure how to do 'boundary checks' on indexed elements and I didn't want to spend too much time figuring out a way to do them. For the first two questions, I didn't use the see solution feature as I fully understood what was required..." - P161*

Lack of self-regulation skills. Some students quite candidly talked about their lack of self-control when it came to using the solution feature. Where some students described being tempted when facing the time pressure of a deadline, others candidly talked about their lack of self-regulation as a primary factor in their decision to use the feature. For example, P941 said, *"I honestly just got lazy, but when it was more straight forward I would not use [the "See Solution"] feature."* Similarly, P181 stated "*don't give us the option to be lazy.*" These admissions reflect gaps in self-control and intrinsic motivation, which are critical components of self-regulated learning.

Curiosity. The final motivation shared by students was curiosity. A few participants used the solution to assess its accuracy or to compare their correct solution to the solution offered by the tool:

> *"I used the see solution to see if it generated the correct solution, as I was curious to what it would produce... it was useful to show other ways to solve the code in an efficent manner." - P681*

> *"I used the 'see solution' feature after I'd done the problem, simply to compare my solution with a model one. I'd followed the advice of the AI bot so the solutions looked pretty much the same." - P975*

Based on the log data, 8.4% (n=37) of the 440 students who used the solution feature clicked the "See Solution" button for the first time for one or more of the three problems *after* they had already solved the problem correctly and passed all associated test cases. This corroborates that a small subset of students used the feature out of curiosity, to verify their work, or to explore alternative solutions and further develop their problem-solving skills after they had correctly solved the problem.

5 Discussion

Contrary to previous work where students expressed their desire to independently arrive at a solution rather than being provided with solutions directly from AI TAs [6], 50% of students in our study used the "See Solution" feature (RQ1a., see Section 4.1.1). However, this behavior aligns with student preferences for receiving answers when working with human TAs [23]. Our work provides insight into why students may elect to use such a feature.

Reported Benefits of Solutions. Based on our qualitative findings and log data analysis, some students reported disabling guardrails to help their learning rather than in order to copy solutions directly. Some students reported exploring solutions to verify their work after completion, while others reported using it to explore alternative solutions (RQ2., see Section 4.2).

Reported Concerns over Solutions. Some students felt they used the solution feature because they lacked self-regulation and self-control. This raises some concerns about over-reliance on the feature, which aligns with recent work on GenAI use [15, 43, 45].

Potential Impact on Learning. Students who performed worse in the course were more likely to use the solution feature when solving all three tasks in our study. Although we cannot determine causality from this correlative study in a single lab of a course, it seems likely that struggling students may have been more apt to need to use the solution feature to solve the assignment than higher performing students. Concerningly, if struggling students were to become dependent on using solutions, it could cause an over-reliance on the feature that might hinder learning [7, 15, 43, 47].

Student Procrastination The results of our study also confirm that low-performing students are more likely to procrastinate, similar to the findings of previous work [9, 21, 46]. However, empirical log data from our observational study shows that the students' decision to use the "See Solution" feature was independent of student procrastination behavior (RQ1b., see Section 4.1.2). Qualitative feedback from students indicated that time pressure contributed to their decision to view the solution, mirroring patterns observed in student interactions with human TAs [23, 44]. These mixed results indicate that further research is needed to draw stronger conclusions.

Future Studies. Future research could extend this analysis across a course term to draw stronger conclusions. If low-performing students consistently rely on the solution feature over longer periods, their interactions with systems that allow disabling guardrails could serve as an indicator to identify at-risk students who may lack self-regulation skills. Additionally, future research could explore the relationship between effort exerted and solution-seeking behaviors by conducting a more fine grained analysis on students' code submissions, chat prompts, and their interaction with "See Solution" feature. Such work could also examine when AI TAs should grant access to solutions, particularly when students remain stuck on a problem for extended periods.

AI TA Design. Our findings may have implications for designers of AI TA tools. Students reported conflicting opinions about the use of the "See Solution" feature with some reporting they used it to help their learning and others believing it might hinder their learning. Therefore, further evidence is needed to determine whether such a feature can be helpful, specifically under what conditions and for which students it can be helpful. For the students who use solutions to aid their learning, there is a value to having an option in AI TAs that provides students with the option to view alternate solutions, particularly after they have solved the problem or when they are assigned formative low-stakes practice problems. For the latter, instructors could be given an an option to enable or disable such features in tools that offer AI-TAs at a problem level.

Limitations. This study was conducted in one lab that occurred late in an introductory programming course at a large research-focused university. Findings may not generalize outside this context. Additionally, student use of the "See Solution" feature may have been influenced by the task problem statements and the lab's design that directed students to interact with a guardrailed AI TA. This design choice might have affected students' comfort level when seeking help potentially leading them to use the "See Solution" feature more frequently.

Furthermore, because the AI TA did not receive the problem description as part of its prompt, students may have become frustrated with its limited understanding and less effective guidance. Six percent of the students in the sample disagreed that the AI TA's responses were correct or helpful. This may have encouraged some of these students to rely on the "See Solution" feature. Without the AI TA having problem context, students had to write meaningful prompts to receive effective help which was a deliberate design choice we made as we wanted them to develop prompting skills. Future studies could explore how refining task instructions to encourage productive use of the AI TA and providing it with full problem context might influence study outcomes, as well as assess the accuracy and reliability of AI-generated feedback.

Another limitation is students' access to the lab task for over a week, which may have allowed them to consult external resources such as ChatGPT or seek assistance from their peers. While it is hard to determine how external influences affected student decisions, the low stakes of the three programming tasks in the lab (10% of lab grade or 0.1% of course grade) and the option to use the "See Solution" button without penalty may have reduced the incentive for misconduct. Future studies in controlled environments could help isolate the effects of the intervention more clearly. However, our current study allowed us to scale the intervention to a larger cohort of students than would not be feasible in a controlled setting.

6 Conclusion

In this study, we explore the behavior and motivations of students learning programming when using an AI TA with *optional* guardrails. Despite students reporting that they valued guardrails on the AI TA to help their learning, many students elected to bypass the guardrails when given the option. Students reported removing these guardrails from the AI TA for reasons beyond directly copying solutions, such as assistance in problem solving or curiosity to explore alternative solutions. We also found that low-performing students were more likely to request solutions for every problem in the lab assignment, and more likely to procrastinate and complete the lab closer to the deadline. Our findings contribute to a broader understanding of how students interact with AI tools in education with implications for designing AI TAs that can support educational goals and student motivations.

Acknowledgments

This work was partially supported by the National Science Foundation Award #2417374. Sections of this paper were refined using GenAI tools (ChatGPT and Writefull on Overleaf) to improve grammar and clarity. The authors thank Dustin Karp and Prayuj Tuli for implementing the "See Solution" feature in Edugator and the reviewers for their feedback.

References

[1] 2025. Edugator. https://edugator.app/.

[2] Vincent Aleven, Bruce Mclaren, Ido Roll, and Kenneth Koedinger. 2006. Toward Meta-cognitive Tutoring: A Model of Help Seeking with a Cognitive Tutor. *Int. J. Artif. Intell. Ed.* 16, 2 (April 2006), 101–128.

[3] Joseph E Beck and Yue Gong. 2013. Wheel-spinning: Students who fail to master a skill. In *Artificial Intelligence in Education: 16th International Conference, AIED 2013.* Springer Berlin Heidelberg, Berlin, Heidelberg, 431–440.

[4] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. doi:10.1191/1478088706qp063oa

[5] Neil C. C. Brown, Pierre Weill-Tessier, Juho Leinonen, Paul Denny, and Michael Kölling. 2025. Howzat? Appealing to Expert Judgement for Evaluating Human and AI Next-Step Hints for Novice Programmers. *ACM Trans. Comput. Educ.* 25, 3, Article 31 (July 2025), 43 pages. doi:10.1145/3737885

[6] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024).* 408–414. doi:10.1145/3649217.3653574

[7] 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. doi:10.1145/3624720

[8] Marc Diaz, Dustin Karp, Prayuj Tuli, and Amanpreet Kapoor. 2025. Edugator: An AI-enabled Tool for Creating and Delivering Interactive Computing Content. In *Proc. of the 56th ACM Technical Symposium on CS Education V. 2* (Pittsburgh, PA, USA) *(SIGCSETS 2025).* ACM, NY, USA, 1732. doi:10.1145/3641555.3705025

[9] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing effective and ineffective behaviors of student programmers. In *Proc. of the 5th Int. Workshop on Comp Ed Research (ICER '09).* 3–14. doi:10.1145/1584322.1584325

[10] Tony Haoran Feng, Paul Denny, Burkhard Wuensche, Andrew Luxton-Reilly, and Steffan Hooper. 2024. More Than Meets the AI: Evaluating the performance of GPT-4 on Computer Graphics assessment questions. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24).* ACM, New York, NY, USA, 182–191. doi:10.1145/3636243.3636263

[11] Sebastian Gutierrez, Irene Hou, Jihye Lee, Kenneth Angelikas, Owen Man, Sophia Mettille, James Prather, Paul Denny, and Stephen MacNeil. 2025. Seeing the

Forest and the Trees: Solving Visual Graph and Tree Based Data Structure Problems using Large Multimodal Models. In *Proc of the 27th Australasian Comp Ed Conference (ACE '25)*. ACM, NY, USA, 124–133. doi:10.1145/3716640.3716654

[12] 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. In *Proc of the 2023 ACM Conf on Int Comp Ed Research (ICER '23)*. 93–105. doi:10.1145/3568813.3600139

[13] Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. AI-TA: Towards an Intelligent Question-Answer Teaching Assistant using Open-Source LLMs. arXiv:2311.02775 [cs.LG] https://arxiv.org/abs/2311.02775

[14] 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 *Proc of the 26th Australasian Comp Ed Conf (ACE '24)*. 29–38. doi:10.1145/3636243.3636247

[15] 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 (ACE '24)*. 39–48. doi:10.1145/3636243.3636248

[16] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. In *Proc of the 2024 CHI Conf on Human Factors in Computing Systems (CHI '24)*. Article 650, 20 pages. doi:10.1145/3613904.3642773

[17] Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 2017. Quantifying Incremental Development Practices and Their Relationship to Procrastination. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. 191–199. doi:10.1145/3105726.3106180

[18] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. 52–58. doi:10.1145/3649217.3653612

[19] Sophia Krause-Levy, Rachel S. Lim, Ismael Villegas Molina, Yingjun Cao, and Leo Porter. 2022. An Exploration of Student-Tutor Interactions in Computing. In *Proc of the 27th ACM Conf on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE '22)*. 435–441. doi:10.1145/3502718.3524786

[20] 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 *Proc of the 2023 ACM Conf on International Computing Education Research - V1 (ICER '23)*. 106–121. doi:10.1145/3568813.3600138

[21] Soohyun Nam Liao, Sander Valstar, Kevin Thai, Christine Alvarado, Daniel Zingaro, William G. Griswold, and Leo Porter. 2019. Behaviors of Higher and Lower Performing Students in CS1. In *Proc of the 2019 ACM Conf on Innovation and Technology in CS Ed (ITiCSE '19)*. 196–202. doi:10.1145/3304221.3319740

[22] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proc of the 23rd Koli Calling Int Conf on Comp Ed Research (Koli Calling '23)*. Article 8, 11 pages. doi:10.1145/3631802.3631830

[23] Rachel S. Lim, Sophia Krause-Levy, Ismael Villegas Molina, and Leo Porter. 2023. Student Expectations of Tutors in Computing Courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. 437–443. doi:10.1145/3545945.3569766

[24] Rongxin Liu, Carter 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 *Proc of the 55th ACM Technical Symposium on CS Ed1 (SIGCSE 2024)*. 750–756. doi:10.1145/3626252.3630938

[25] Richard Lobb and Jenny Harlow. 2016. Coderunner: a tool for assessing computer programming skills. *ACM Inroads* 7, 1 (Feb. 2016), 47–51. doi:10.1145/2810041

[26] Iris Ma, Alberto Krone-Martins, and Cristina Videira Lopes. 2024. Integrating AI Tutors in a Programming Course. In *Proc of the 2024 on ACM Virtual Global Comp Ed Conf (SIGCSE Virtual 2024)*. 130–136. doi:10.1145/3649165.3690094

[27] Eric D. Manley, Timothy Urness, Andrei Migunov, and Md. Alimoor Reza. 2024. Examining Student Use of AI in CS1 and CS2. *J. Comput. Sci. Coll.* 39, 6 (April 2024), 41–51.

[28] Julia M. Markel and Philip J. Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. 502–508. doi:10.1145/3408877.3432533

[29] Joshua Martin, Stephen H. Edwards, and Clfford A. Shaffer. 2015. The Effects of Procrastination Interventions on Programming Project Success. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. 3–11. doi:10.1145/2787622.2787730

[30] Samiha Marwan, Anay Dombe, and Thomas W. Price. 2020. Unproductive Help-seeking in Programming: What it is and How to Address it. In *Proc of the 2020 ACM Conf on Inn and Tech in CS Ed (ITiCSE '20)*. 54–60. doi:10.1145/3341525.3387394

[31] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proc ACM on Human-Computer Int.* 3, CSCW (2019), 1–23.

[32] Moritz Mueller, Corinna List, and Michael Kipp. 2025. The Power of Context: An LLM-based Programming Tutor with Focused and Proactive Feedback. In *Proceedings of the 6th European Conference on Software Engineering Education (ECSEE '25)*. ACM, New York, NY, USA, 1–10. doi:10.1145/3723010.3723034

[33] OpenAI. 2025. *ChatGPT*. https://chatgpt.com/ [Large language model].

[34] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Vee Pettit, Leo Porter, Brent N. Reeves, Jaromir Savelka, IV Smith, David H., Sven Strickroth, and Daniel Zingaro. 2025. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (Milan, Italy) *(ITiCSE 2024)*. ACM, New York, NY, USA, 300–338. doi:10.1145/3689187.3709614

[35] 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 - V1 (ICER '24)*. 469–486. doi:10.1145/3632620.3671116

[36] Victor-Alexandru Pădurean, Paul Denny, Alkis Gotovos, and Adish Singla. 2025. Prompt Programming: A Platform for Dialogue-based Computational Problem Solving with Generative AI Models. In *Proc of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. ACM, NY, USA, 458–464. doi:10.1145/3724363.3729094

[37] Shuying Qiao, Paul Denny, and Nasser Giacaman. 2025. Oversight in Action: Experiences with Instructor-Moderated LLM Responses in an Online Discussion Forum. In *Proc of the 27th Australasian Computing Education Conference (ACE '25)*. ACM, New York, NY, USA, 95–104. doi:10.1145/3716640.3716651

[38] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2021. Challenges Faced by Teaching Assistants in Computer Science Education Across Europe. In *Proc of the 26th ACM Conf on Inn and Tech in CS Ed* (Virtual Event, Germany) *(ITiCSE '21)*. ACM, NY, USA, 547–553. doi:10.1145/3430665.3456304

[39] Adrian Salguero, William G. Griswold, Christine Alvarado, and Leo Porter. 2021. Understanding Sources of Student Struggle in Early Computer Science Courses. In *Proc of the 17th ACM Conf on International Comp Ed Research* (Virtual Event, USA) *(ICER 2021)*. ACM, New York, NY, USA, 319–333. doi:10.1145/3446871.3469755

[40] Judy Sheard, Paul Denny, Arto Hellas, Juho Leinonen, Lauri Malmi, and Simon. 2024. Instructor Perceptions of AI Code Generation Tools - A Multi-Institutional Interview Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. 1223–1229. doi:10.1145/3626252.3630880

[41] James Skripchuk, Neil Bennett, Jeffrey Zhang, Eric Li, and Thomas Price. 2023. Analysis of Novices' Web-Based Help-Seeking Behavior While Programming. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE'23)*. 945–951. doi:10.1145/3545945.3569852

[42] Aaron J. Smith, Kristy Boyer, Jeffrey Forbes, Sarah Heckman, and Ketan Mayer-Patel. 2017. My Digital Hand: A Tool for Scaling Up One-to-One Peer Teaching in Support of Computer Science Learning. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. ACM, New York, NY, USA, 549–554. doi:10.1145/3017680.3017800

[43] 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 *Proc of the 2024 on Innovation and Technology in CS Education (ITiCSE 2024)*. 297–303. doi:10.1145/3649217.3653584

[44] Ismael Villegas Molina, Jeannie Kim, Audria Montalvo, Apollo Larragoitia, Rachel S. Lim, Philip J. Guo, Sophia Krause-Levy, and Leo Porter. 2025. Undergraduate Computing Tutors' Perceptions of their Roles, Stressors, and Barriers to Effectiveness. In *Proc of the 56th ACM Tech Sym on CS Ed* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. ACM, NY, USA, 1155–1161. doi:10.1145/3641554.3701784

[45] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in Computing Education: Perspectives of Students and Instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*. IEEE Computer Society, 1–9. doi:10.1109/FIE58773.2023.10343467

[46] Jiayi Zhang, Taylor Cunningham, Rashmi Iyer, Ryan Baker, and Eric Fouh. 2022. Exploring the Impact of Voluntary Practice and Procrastination in an Introductory Programming Course. In *Proc of the 53rd ACM Technical Symposium on Computer Science Education - V1 (SIGCSE 2022)*. 356–361. doi:10.1145/3478431.3499350

[47] Shunan Zhang, Xiangying Zhao, Tong Zhou, and Jang Hyun Kim. 2024. Do you have AI dependency? The roles of academic self-efficacy, academic stress, and performance expectations on problematic AI usage behavior. *International Journal of Educational Technology in Higher Education* 21, 1 (2024), 34.