Do Hackers Dream of Electric Teachers?: A Large-Scale, In-Situ Measurement of Cybersecurity Student Behaviors and Educational Performance with AI Tutors

Michael Tompkins
Arizona State University
Tempe, Arizona, USA
mctompk1@asu.edu

Nihaarika Agarwal
Arizona State University
Tempe, Arizona, USA
nagarw48@asu.edu

Ananta Soneji
Arizona State University
Tempe, Arizona, USA
asoneji@asu.edu

Robert Wasinger
Arizona State University
Tempe, Arizona, USA
rwasinge@asu.edu

Connor Nelson
Arizona State University
Tempe, Arizona, USA
connor.d.nelson@asu.edu

Kevin Leach
Vanderbilt University
Nashville, Tennessee, USA
kevin.leach@vanderbilt.edu

Rakibul Hasan
Arizona State University
Tempe, Arizona, USA
rakibul.hasan@asu.edu

Adam Doupé
Arizona State University
Tempe, Arizona, USA
doupe@asu.edu

Daniel Votipka
Tufts University
Medford, Massachusetts, USA
dvotipka@cs.tufts.edu

Yan Shoshitaishvili
Arizona State University
Tempe, Arizona, USA
yans@asu.edu

Jaron Mink
Arizona State University
Tempe, Arizona, USA
jaron.mink@asu.edu

Abstract

To meet the ever-increasing demands of the cybersecurity workforce, AI tutors have been proposed for personalized, scalable education. But, while AI tutors have shown promise in introductory programming courses, no work has evaluated their use in hands-on exploration and exploitation exercises (e.g., "Capture the Flag") commonly used to teach cybersecurity. In particular, it is unclear how students use AI tutors, or what types of use correlate with greater success in solving the challenges in real, large-scale cybersecurity courses.

To answer this, we conducted a semester-long observational study of an embedded AI tutor with 309 students in an upper-division introductory cybersecurity course. By analyzing 142,526 student queries sent to the AI tutor across 383 cybersecurity challenges spanning 9 core cybersecurity topics and an accompanying end-of-semester survey, we find (1) what queries and conversation styles students use with AI tutors, (2) how these styles relate to challenge completion, and (3) students' perceptions of AI tutors in cybersecurity education. In particular, we identify three broad AI tutor conversation styles among students: Short (bounded, few-turn exchanges), Reactive (repeatedly submitting code and errors), and Proactive (driving problem-solving through targeted inquiry). We also find that these styles are significantly correlated with challenge completion, and that the completion-rate gap between styles widens as materials become more advanced. Furthermore, students valued the tutor's availability but reported that it became less useful for harder material. Based on our results, we provide suggestions for security educators and developers on practical AI tutor use.

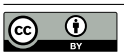
CCS Concepts

• **Applied computing** → **Computer-assisted instruction**; • **Social and professional topics** → *Computing education*; • **Security and privacy** → Social aspects of security and privacy.

Keywords

AI tutors; LLMs; cybersecurity education; help-seeking behavior; conversation analysis; computing education

ACM Reference Format:
Michael Tompkins, Nihaarika Agarwal, Ananta Soneji, Robert Wasinger, Connor Nelson, Kevin Leach, Rakibul Hasan, Adam Doupé, Daniel Votipka, Yan Shoshitaishvili, and Jaron Mink. 2026. Do Hackers Dream of Electric Teachers?: A Large-Scale, In-Situ Measurement of Cybersecurity Student Behaviors and Educational Performance with AI Tutors. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846694



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

CCS '26, The Hague, Netherlands.

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

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846694

1 Introduction

Cybersecurity experts are essential to maintaining the security of organizations, yet the cybersecurity workforce demand is consistently unmet [16]. Bridging this gap is hard in part because cybersecurity training is fundamentally hands-on, with the current trend consisting of practice-based learning such as Capture the

Flag (CTF) challenges [45] and structured vulnerability discovery exercises [31, 46].

Because of their effectiveness, university curricula are increasingly adopting practice-based approaches in which students solve progressively difficult challenges [30–32, 47]; however, two obstacles make practice-based approaches difficult to scale. *First*, these exercises are often open-ended explorations of complex systems; this can be particularly demanding for learners as they must navigate unfamiliar tools, complex system internals, and open-ended problems with minimal structure [25, 46], which can lead to discouragement or attrition [11, 25]. *Second*, prior work has shown that mentoring help from experts is important for overcoming these challenges [11, 45]; however, formal courses often lack the teaching assistant (TA) capacity to provide individual mentoring at scale, and students may hesitate to approach authority figures [11].

In response to such concerns, AI tutoring has emerged as a proposed solution to provide individual student mentoring at scale. In contrast to general chatbots, which educators increasingly worry are used to bypass students' productive struggle rather than support learning [1, 18, 24], purpose-built AI tutors have been proposed as an alternative, and introductory CS deployments report that personalized feedback and self-paced progress benefit students [20, 41, 50, 51]. However, it remains unclear whether these benefits transfer to cybersecurity education given its exploratory nature, looser structure, and cross-domain reasoning. More specifically, we ask:

RQ1 How do students interact with AI tutors while solving cybersecurity challenges?
RQ2 How do AI tutor conversations relate to challenge performance, and how does this vary by cybersecurity topic?
RQ3 What do students believe are the benefits and drawbacks of AI tutors in cybersecurity courses?

To answer these questions, we conducted a semester-long observational study of an embedded AI tutor with 309 students in an upper-division introductory cybersecurity course. By analyzing 142,526 student queries sent to the AI tutor across 383 cybersecurity challenges spanning 9 core cybersecurity topics and an accompanying end-of-semester survey on their opinions of the system, we provide insights on the effectiveness of cybersecurity AI tutors in practice.

First, we find that participants commonly ask seven types of queries, and perhaps surprisingly, only a small minority of queries (3.5%) are direct solution requests. *Second*, using this taxonomy of queries to fit a mixture hidden Markov model, we find that students' conversations naturally partition into three broad styles: Short (bounded, single-purpose exchanges), Reactive (tending to begin with a confused student), and Proactive (tending to begin with focused, specific questions). Compared to Proactive conversations, Reactive conversations are twice as likely to contain a direct solution request (25.8% vs. 11.8%) and six times as likely to contain more than one (10.9% vs. 1.8%). *Third*, we find that conversation style is associated with challenge completion: Proactive conversations show equal or higher completion rates than Reactive ones in all nine modules, and the gap between styles widens from under 1% on introductory material to 14.3% on the most advanced. *Fourth*, from students' survey responses we find that they valued the tutor's constant availability, but reported that it became less useful as material grew harder.

Based on these findings, we provide a set of recommendations for practical and effective AI tutor usage for educators, identify opportunities for improvements in AI tutor system designs, and discuss how these results can be interpreted within educational theory.

2 Background & Related Work

LLM-Assisted Coding. Deployments of large language model (LLM) assistants into introductory programming and systems courses have seen students use them for code generation, debugging, and explanation, while highlighting the importance of tool design to preserve student agency [20, 41, 51]. Jošt et al. [18] link real-world LLM usage patterns to outcomes, finding that reliance on code generation and debugging may correlate with lower performance, while explanation-focused use appears less harmful. Bastani et al. [1] provide causal evidence for this concern through a large-scale study where students given unrestricted access to GPT-4 performed worse on unassisted exams than students with no access.

AI Tutors in CS Education. Many systems now treat LLMs as conversational tutors rather than simple code generators. Studies of these systems emphasize opportunity for personalized feedback and guidance while warning about over-reliance and academic integrity [4]. Several deployments integrate prompt construction into curriculum, framing prompt design and refinement as learning objectives [7, 37], while others design and deploy LLMs as conversational tutors for CS coursework, enabling multi-turn conversational assistance instead of one-shot queries [35, 50]. A growing body of research also suggests that the design of these systems matters more than the underlying model, with evidence that AI tutor design can mitigate learning harms from unrestricted LLM access [1]. A common element is the use of Socratic-style questioning rather than direct answers, encouraging students to reason through problems rather than passively receive solutions [8, 10]. However, no consensus exists for which designs are best.

Help-Seeking in Computing Education. The way students seek help has long been recognized as a predictor of learning outcomes, with Karabenick and Gonida [19] highlighting adaptive learners who build targeted understanding, executive learners who seek help to complete without engaging, and avoidant learners who avoid help despite needing it. In computing education, these manifest in observable ways: students group into distinct help-seeking profiles by which resources they use [2], students whose forum questions show their reasoning earn higher course grades [44], and students in hint-based environments exhibit both help abuse (requesting hints without thinking) and help avoidance (struggling without requesting hints), both of which can impede learning [24]. Recent work extends these findings to LLM-based tools, showing that students who prompt for conceptual understanding perform better on assignments and exams [27]. However, no study has examined this in cybersecurity education, where tasks are more open-ended and span multiple knowledge domains, or whether the style–outcome associations observed in other contexts reflect effects of interaction strategy or selection of which students adopt each style.

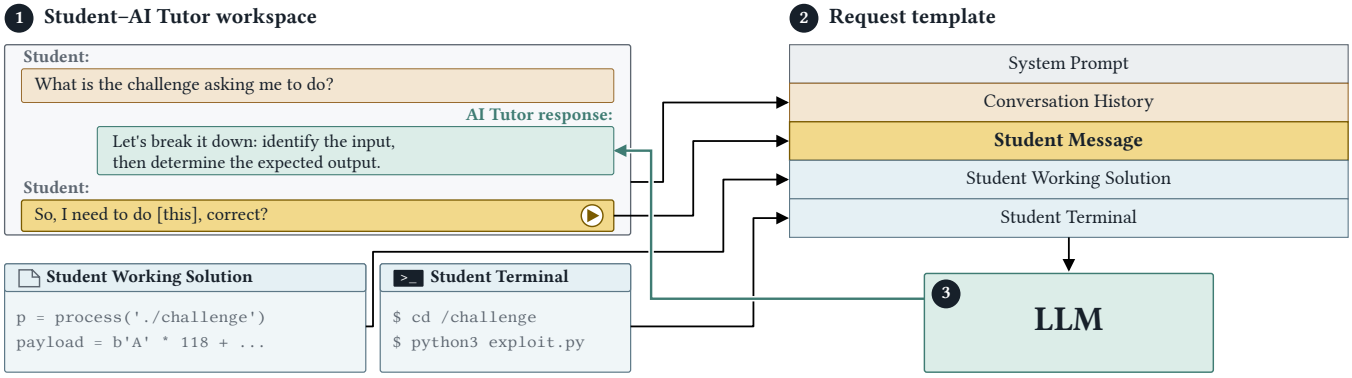


Figure 1: AI Tutor Overview – Participants solve challenges inside an instrumented container, capturing their active terminal and file context. When a student interacts with the AI tutor (❶), the system bundles up the AI tutor's system prompt, the participant's conversation history and query, and the captured context from the container (❷). This gets sent to the LLM (❸) which processes the query and responds to the participant.

Module	Topics
1. Linux	Linux command line interface
2. Data & Access	Encodings (ASCII, Base64) and permissions
3. Web & SQL	HTTP and SQL queries, inspecting traffic
4. Web Security	Path traversals, XSS, command injections
5. Comp. Arch.	x86-64 assembly, memory, debugging
6. Net. Security	Traffic analysis, firewalls, DoS, TCP/UDP
7. Cryptography	Symmetric/asymmetric, hashing, DHKE
8. Reverse Engr.	Iterative reversing of a file format parser
9. Binary Security	Memory corruption, buffer overflows

Table 1: Cybersecurity Course Modules

Cybersecurity Education. Unlike general programming education, cybersecurity education centers on hands-on application across diverse knowledge domains. Yet, existing work on generative AI in these classrooms largely focuses on high-level pedagogy and self-directed learning [9, 23, 43], with fewer efforts building dedicated tutoring systems [28, 29] or giving students direct LLM access for security tasks [48]. Thus, while prior work has begun to integrate generative AI into cybersecurity education and reports on students' perceptions of AI tutors, no work focuses on how students use and perform with these systems.

3 Methodology

To understand how students use AI tutors in cybersecurity education, we integrated SENSAI [29], an AI tutoring system, into a cybersecurity course at a public US university in the spring of 2025 and assessed students' AI tutor conversations, course performance, and perceptions of the AI tutor. Our IRB-approved processes ensured compliance with FERPA regulations, fairness in grading, and general standards of ethical research practices (see Appendix A).

Course Content and Challenges. We study a 15-week online cybersecurity course in which students access all course materials through pwn.college [30, 32]. The course is divided into nine modules (see Table 1) covering commonly taught security concepts:[1] the course begins by teaching the Linux command line and file access control, and then moves to cover a range of security topics such as web security, cryptography, and binary exploitation. Each module consists of pre-recorded lecture videos and 21 to 92 (avg. 44) CTF-style challenges students solve for course credit. In total, students are assigned 396 challenges.

Each challenge contains two components: (1) a description of tasks students must solve to complete the challenge that decreases in detail and direction as the challenges increase in difficulty, and (2) a Dockerized Linux system that hides a unique text string (a "flag").[2] To complete the challenge, students apply concepts taught in the lectures in the Linux environment to access the flag and submit it to the platform. For example, a web security challenge requires students to perform a command injection attack against a vulnerable HTTP endpoint to retrieve the flag (see the extended version [42]). These challenges made up 80% of the course grade.

AI Tutor Architecture and Workflow. Our AI tutor was designed to assist students when solving challenges by answering conceptual questions and providing light coding assistance. As shown in Figure 1, the architecture of the tutor consists of ❶ a chat-based interface for student queries, ❷ a request template that combines (a) a system prompt instructing the LLM to be a Socratic mentor, (b) the conversation history between the tutor and a student, (c) the new student query, and (d) the current Linux system context, composed of the latest saved file and active terminal history to capture command usage and outputs, and ❸ an LLM to which the templated request is sent and from which a response is received; we specifically use GPT-4o's API as it was the state-of-the-art model at the time. We chose to instruct our LLM to be a Socratic mentor based on emerging evidence that guided questioning produces better learning outcomes than direct answers [1, 10] and best practices for educational prompt templates [5, 8]. The full system prompt is provided in the extended version [42]. The included Linux system

[1] We removed a tenth capstone module due to low participation caused by course grading policies, i.e., students could earn high grades without completion.
[2] All flags come in a predefined format (e.g., "flag{XXXX.XX}").

context simulates a TA's ability to "look over the shoulder," enabling the model to provide targeted guidance without requiring the student to describe their working state. In addition, the system *does not* have access to a solution, discouraging extraction and encouraging students to build one cooperatively.

At any time during a challenge, students can open the chat interface and submit free-form queries to the AI tutor. By default, all elements shown in ❷ in Figure 1 are included when a query is made, but can be optionally removed via a toggle button (which, in practice, was rarely used [42]). When a query is submitted, the request is bundled, sent to the LLM, and the LLM's response is presented to the student. Students may submit additional queries as needed. The tutor's context window is not preserved across challenges or challenge restarts.

3.1 Observation Data

We collected data on students' (1) tutor conversations, (2) challenge completion, and (3) perceptions of the AI tutor.

AI Tutor Queries. Participants interacted with the system by sending queries. A *query* consists of the participant's message submitted to the tutor with its accompanying system context. A *response* is the message the AI tutor provides back to the student. The complete sequence of queries and responses between a student and the AI tutor for a given challenge makes up a *conversation*. Once a student moves on to the next challenge, a new conversation begins. For our analysis, if a student restarts a challenge or returns to it after working on others, we merge these sessions into a single conversation (i.e., students can have at most one conversation per challenge). We collected participating students' conversations throughout the semester, resulting in at least one conversation for 383 of the 396 assigned challenges.

Challenge Completions. Participating students' *performance* (completion rate) for all challenges is captured throughout the semester.

End-of-Semester Survey. We conducted an end-of-semester survey to understand and measure students' perception of the AI tutor (see the extended version [42]). The survey contained five sections with both open- and closed-ended questions about their background knowledge (Q1–Q3), perceptions of utility (Q4–Q9), system usability (Q10–Q13), comparisons to TAs (Q14–Q17), and possible system improvements (Q18).

3.2 Qualitative Data Analysis

To understand student-tutor conversations and how students perceive our AI tutor, we qualitatively coded students' queries to the AI tutor and open-ended responses to the end-of-semester survey.

AI Tutor Queries. To analyze student queries, we used an iterative open coding approach [6] with coders familiar with the course and the cybersecurity concepts to code students' queries.[3] The first author read a random sample of 500 queries (from the full 142,526 query set) to construct the initial codebook. This initial codebook was then explained to a second coder (another author), and two coders independently coded queries in rounds of 500. After each round, the coders met to calculate inter-rater reliability (IRR) for each code. We calculated IRR using Cohen's κ, as it is a conservative measure that accounts for chance agreement [26]. Coders resolved disagreements and adjusted the codebook to refine code definitions or add codes as necessary. After 8 rounds of independent coding (4,000 queries total), they reached a high level of agreement across codes ($\kappa = .84$) [26], producing the final codebook.

Due to the scale of our data, we chose to use an LLM (GPT-5.2 [33]) to apply our finalized codebook to the full set of 142,526 queries. To code each query, we submitted a prompt containing strict code definitions and few-shot examples of correct coding to the LLM, along with the target query, tutor-accessible system context, and the tutor's response. Following best practices for LLM-based coding [14, 21], we iteratively refined this prompt by comparing it against a ground-truth set of 1,500 queries from the previously manually coded set. We randomly sampled 500 queries and had the LLM code them, then calculated the IRR between the LLM's codes and our ground-truth coding using Cohen's κ, treating the LLM as an additional coder measured against human consensus. Two coders reviewed disagreements and iteratively revised the prompt. This process was repeated for 8 rounds until the LLM demonstrated high overall agreement ($\kappa = .82$) [26].[4] The full codebook, including the κ values for each task family, is provided in the extended version [42]. We then used the refined LLM prompt to code the entire dataset that served as input into our quantitative analyses, specifically the mixture hidden Markov model (MHMM) conversation-style analysis (§4.3) and mixed-effects logistic regression (§5.1).

End-of-Semester Survey. To analyze the five open-response reflection questions (Q8, Q9, Q13, Q17, Q18), we used an inductive open coding approach [6]. Two coders independently coded student reflections in rounds of 150 responses, randomly selected from the 293 total survey responses. The coders met between rounds to resolve disagreements and update the codebook as necessary. After 5 rounds, the coders achieved Cohen's κ above .80 for all codes. The primary coder then coded the remaining responses. The final codebook is provided in the extended version of this work [42].

3.3 Quantitative Data Analysis

Using the coded student queries, we first evaluate how patterns in conversations emerge. To do this, we fit MHMMs [13] to the coded query sequences (see §4.3 for details). After finding these conversational patterns, or "conversation styles", we then evaluated how they relate to performance by modeling challenge completion as a function of conversation style and the module being attempted. To do this, we fit mixed-effects logistic regression using whether the participant completed the challenge associated with each conversation as our dependent variable. We then fit participants' conversation styles and module as fixed effects, and each participant as a random effect (see §5.1 for details).

3.4 Eligibility, Recruitment, and Consent

All students enrolled in the course were eligible for study participation. To enroll, students must have completed prerequisite courses in basic computer architecture, programming languages, and data structures and algorithms, and must be on-campus students. Prior to

[3] Because some queries are only classifiable in context of the full conversation, the coder had access to the tutor-visible system context and the tutor's response.

[4] We randomly sample from our 1,500-query ground truth each round. While this can introduce bias for human coders, we simply delete the LLM's context between rounds.

	Count (%)
Prior Cybersecurity Understanding	
Not at All Knowledgeable	135 (46.1)
Slightly Knowledgeable	92 (31.4)
Somewhat Knowledgeable	31 (10.6)
Moderately Knowledgeable	25 (8.5)
Extremely Knowledgeable	10 (3.4)
Frequency of AI Use (outside course)	
Never	8 (2.7)
Rarely (≤1 a week)	61 (20.8)
Sometimes (2–3 times a week)	126 (43.0)
Often (Once a day)	55 (18.8)
Very Often (More than once a day)	43 (14.7)

Table 2: Participants' Cybersecurity Knowledge and AI Use – Self-reported background knowledge and AI use of the 293 participants (of 309) included in the survey analysis (Q1–Q2).

Module	Chal. #	*n* (%)	Convs. (%)	Avg. C : Q
1. Linux	84	258 (83)	3,802 (15)	14.7 : 3.8
2. Data & Access	38	279 (90)	3,029 (12)	10.9 : 6.3
3. Web & SQL	46	268 (87)	3,899 (15)	14.5 : 4.5
4. Web Sec.	27	261 (84)	2,815 (11)	10.8 : 8.0
5. Comp. Arch.	65	263 (85)	4,124 (16)	15.7 : 5.1
6. Network Sec.	29	269 (87)	3,103 (12)	11.5 : 6.6
7. Cryptography	34	231 (75)	2,134 (8)	9.2 : 6.8
8. Reverse Engr.	39	209 (68)	1,433 (6)	6.9 : 5.2
9. Binary Sec.	21	181 (59)	922 (4)	5.1 : 6.6

Table 3: Tutor Usage by Module – Percentages indicate the proportion of all participants (n=309) who queried the AI tutor during a given module, and the percentage of total conversations (m=25,261) in a module. "Avg. C : Q" is the average number of conversations per participant (C) and the average number of queries per conversation (Q). "Chal. #" is the count of challenges with ≥ 1 tutor-student conversation.

using the tutor, students had to consent to have their conversation logs collected and analyzed. An additional, separate consent form was also provided for the end-of-semester survey, which enabled the collection and association of survey responses, LLM conversations, and students' course performance. Only students who (1) opted into the AI tutor's data collection, (2) completed the end-of-semester survey and opted to share their responses, and (3) had at least one conversation with the AI tutor in the studied modules participated in our study. If students chose to opt out of using the AI tutor, they were still able to access and use other course resources.

3.5 Participants

In total, 309 of the 720 enrolled students participated. Each participant completed the course, used the AI tutor at least once, and submitted an end-of-semester survey. Within the end-of-semester survey, 16 of our 309 participants self-reported never having used the tutor (despite their logged conversations); we therefore only exclude them from our survey analysis. As shown in Table 2, of these 293 participants, most had little prior cybersecurity education (Q1), with 77% (n=227) describing themselves as "Not at all knowledgeable" or "Slightly knowledgeable" of cybersecurity concepts. This level of cybersecurity knowledge is expected of introductory cybersecurity course students, our target demographic. Most participants were active AI users, with 76% (n=224) reporting they use AI at least 2–3 times a week for problem-solving, and only 3% (n=8) reported never using AI, which aligns with the levels of college student AI use found in prior work [15].

3.6 Limitations

In providing and analyzing real-world student conversations and educational performance, our study also has several limitations. *First*, while an observational study of real-world AI tutor-student interactions provides high ecological validity, we cannot separate correlation from causation, and thus cannot conclude whether conversation styles cause better outcomes, or whether students with better outcomes approach conversations differently. This stems from necessary ethical constraints: studying AI tutor use by real students at scale means we cannot experimentally assign or deny access. While we run additional analysis decomposing this association into within- and between-student components (see the extended version [42]), future work should ethically confirm these findings within a controlled setting. *Second*, while it is possible that students who opt in to our study differ from the full student population, by comparing participants' per-module challenge completion against aggregate course statistics, we find that our participants' completion closely tracks the full course: median completion differs by at most 4.8 percentage points in all nine modules (with no consistent bias). *Third*, while using other AI tools and consulting with other students was against both the study's and the university's academic integrity policies, we are unable to verify whether students used other resources that may have affected their performance. However, unauthorized resource use is a common issue among educators, and thus, these results are representative of practical course deployments. *Fourth*, while we use challenge completions as our proxy for performance, they do not necessarily reflect student learning. Thus, while we cannot strictly claim learning outcomes, these results bear weight on educational outcomes (i.e., grades) commonly accepted across modern educational institutions. *Fifth*, our use of GPT-4o may introduce model-specific idiosyncrasies; as such, we focus on student behavior (queries to the model, educational performance) rather than how the tutor responds, which we leave to future work.

4 Student Interactions With the AI Tutor (RQ1)

Throughout the semester, 309 participants made 142,526 queries across 25,261 conversations with the AI tutor. Table 3 shows the distribution of conversations, student adoption rates, and average query count per conversation.[5] Participants sent a median of 353 queries across 75 conversations, with most conversations being brief (53.7% contain ≤3 queries). Participation declined throughout the course, though 41% of participants used the tutor in all modules. A comprehensive description of use is available within the extended version [42].

[5]Query count represents the length of the conversation in queries from the student to the AI tutor, not factoring in the tutor's responses.

Query Type	Subtype	Count (%)
Verify & Fix *Total: 49,496 (34.7%)*	Debugging	33,325 (23.4)
	Confirmation	16,171 (11.3)
Provide Info *Total: 31,005 (21.8%)*	Paste Context	17,334 (12.2)
	Observations	13,671 (9.6)
Implement *Total: 24,519 (17.2%)*	Procedure Guidance	20,800 (14.6)
	Code Generation	3,719 (2.6)
Get Unstuck *Total: 18,940 (13.3%)*	Direction Request	9,421 (6.6)
	Confusion	4,458 (3.1)
	Vague Request	2,963 (2.1)
	Help Request	2,098 (1.5)
Understand *Total: 11,166 (7.8%)*	Concept Guidance	8,744 (6.1)
	Challenge Guidance	2,422 (1.7)
Solution Request *Total: 4,984 (3.5%)*	Solution Request	4,984 (3.5)
Peripheral *Total: 2,416 (1.7%)*	Social Turn	1,361 (1.0)
	Non-Sequitur	663 (0.5)
	Course/Platform	392 (0.3)

Table 4: Submitted Student Queries – We present the m=142,526 participant queries made, grouped by type and subtype.

4.1 Queries

As Table 4 shows, participants submitted 16 query subtypes, which we organized via axial coding [6] into 7 primary query type families based on each subtype's perceived goal. Below, we describe each query type (and subtypes as "[**subtype**]").

Verify & Fix. In 34.7% (m=49,496) of queries, participants asked the AI tutor to double-check their assumptions and fix issues. In about two-thirds of these queries (m=33,325), participants used the tutor to fix issues they discovered in code [**Debugging**]. Participants typically pasted error messages and included file context in their query (see §3), and optionally asked the tutor to help interpret or fix the issue. These could include highly technical issues such as P70 informing the AI tutor, *"the lea produces an error saying absolute address can not be RIP-relative"*, or assistance in using other debugging methods. In the remaining third of these queries (m=16,171), participants used the tutor to determine whether their challenge approach was correct [**Confirmation**]. This ranged from simple verification questions, e.g., *"is my [jump] table implementation correct"* (P164), to broadly confirming understanding before implementation, e.g., *"could i first shift right 32 then shift left 64 then move it to rax?"* (P250).

Provide Info. In 21.8% (m=31,005) of queries, participants communicated their perception of the state of the problem to the AI tutor. In about half of these cases (m=17,334), they directly submitted terminal output or code snippets without any accompanying interpretation or explanation [**Paste Context**]. In the other half of cases (m=13,671), they provided information not captured by the included context [**Observations**]; e.g., P238 submitted a query observing *"the address for win() is at 0x1d15"*.

Implement. In 17.2% (m=24,519) of queries, participants asked for help implementing a solution to a challenge. Nearly all of these queries (m=20,800, 84.8%) asked for help with a scoped and clear task [**Procedure Guidance**], like the how-to question *"how would i send a curl command with a header"* (P303). In the other 15.2% of cases, participants asked for direct help with building code artifacts [**Code Generation**]. These code generation queries either focused on scaffolding out exploit code, e.g., P3 starting a conversation with *"I need a python script with [security tool's] process"*; modifying existing code, e.g., *"can you alter my html code to have a nested javascript request"* (P114); or requesting example function usage, e.g., *"can you give me an example of using substr in SQL?"* (P131).

Get Unstuck. In 13.3% (m=18,940) of queries, participants turned to the AI tutor when they realized they did not know how to proceed on the problem and needed direction. In half (m=9,421, 49.7%), participants asked orientation questions [**Direction Request**] to determine next steps, e.g., *"How should I start this assignment"* (P260) or *"What do you do after get cookie?"* (P117). Most used these to reorient themselves before returning to the challenge, though a few participants repeated similar queries seemingly in an attempt to extract solutions from the tutor. We distinguish these repeated queries for direction in particular as **Solution Requests** (see below). In much smaller frequencies, participants ask for help (*"can you help me,"* P4) [**Help Request**; m=2,098], express general confusion [**Confusion**; m=4,458], or make vague requests (*"next ?,"* P178) [**Vague Request**; m=2,963]. In such cases, the AI tutor typically responds by restating the challenge goal and listing subtasks or relevant tools, helping students past unknown unknowns.

Understand. 7.8% (m=11,166) of queries focused on building an understanding of the concepts the challenge is designed to teach or specific information about the challenge at hand. 78.3% (m=8,744) of these queries seek understanding of conceptual elements and not implementation procedure [**Concept Guidance**]. These ranged from simple clarification questions such as *"what is exit code -4"* (P172) to technical questions such as *"Why can't I do jmp 0x403000, but if I put 0x403000 in rax and then jmp rax, I get the flag"* (P49) in a challenge meant to teach assembly. These queries are focused on concepts beyond the specific challenge and can help students understand *why* something works. The other 21.7% (m=2,422) of these queries focused on details unique to the current challenge [**Challenge Guidance**], such as P222 asking *"Is this challenge using TCP or UDP"*. Participants also ask questions about challenge artifacts, such as *"what would be my actual parameters according to this code"* (P260) when looking at a vulnerable web server.

Solution Request. In a small number of queries (3.5%; m=4,984), participants made a [**Solution Request**] by querying the tutor for the full solution. These requests often occur at the start of conversations and with minimal evidence of prior work. Participants made requests for end-to-end solutions, such as *"give me full python script please for this challenge to solve"* (P271), which the AI tutor usually pushed back on. In P271's case, the AI tutor responded, *"I understand you're looking for a complete solution, but it's crucial for your learning to work through the problem step-by-step."* We also see participants basing solution requests on the challenge description, such as P284 querying: *"i have to break out of ls -l, can you give me a quick layout of how i can break out and access the flag."*

Peripheral. Finally, 1.7% (m=2,416) of queries were not related to challenge-solving. This included social messages like *"ok"* and *"you are useless"* (P135) [**Social Turn**; m=1,361], non-sequiturs

Rank	Single-query (m=6,036)		Two-query (m=4,459)		Three-query (m=3,074)	
1	Verify & Fix	27.9%	Verify & Fix→Verify & Fix	11.8%	Verify & Fix→Verify & Fix→Verify & Fix	5.5%
2	Implement	23.5%	Implement→Verify & Fix	7.4%	Implement→Verify & Fix→Verify & Fix	3.1%
3	Get Unstuck	21.5%	Get Unstuck→Verify & Fix	6.9%	Get Unstuck→Verify & Fix→Verify & Fix	2.6%
4	Provide Info	12.2%	Verify & Fix→Provide Info	6.7%	Verify & Fix→Provide Info→Verify & Fix	2.6%
5	Understand	9.5%	Get Unstuck→Provide Info	6.0%	Verify & Fix→Verify & Fix→Provide Info	2.4%
6	Solution Request	4.0%	Implement→Implement	5.7%	Get Unstuck→Provide Info→Provide Info	2.0%
7	Peripheral	1.3%	Get Unstuck→Implement	4.4%	Implement→Implement→Verify & Fix	2.0%
8	-	-	Get Unstuck→Get Unstuck	4.4%	Implement→Implement→Implement	2.0%
9	-	-	Implement→Provide Info	3.7%	Verify & Fix→Provide Info→Provide Info	1.9%
10	-	-	Verify & Fix→Implement	3.6%	Get Unstuck→Provide Info→Verify & Fix	1.8%

Table 5: Most Common Queries in Short Conversations – Top sequences for conversations with ≤3 queries, by count. Percentages are relative to all conversations of each length, and the listed patterns cover 100.0% of single-query, 60.5% of two-query, and 25.8% of three-query conversations. Percentages are computed from unrounded counts and may not sum exactly.

which were uninterpretable or minimally relevant [**Non-Sequitur**; m=663], and queries about the course or platform unrelated to the specific challenge [**Course/Platform**; m=392], e.g., *"how can i request an extension for the assignment"* (P129).

4.2 Short Conversations

We find most conversations were short, with slightly more than half consisting of 3 or fewer queries (53.7%; m=13,569). We treat these as their own conversation style: quick, bounded, and typically single-purpose exchanges (e.g., confirming an approach or fixing a single bug), distinct from the sustained back-and-forth we analyze separately in §4.3. We present the 10 most common short conversations (≤3 queries) in Table 5.

Shorter Conversations Are the Most Common, but Least Varied. Single-query conversations were most common (44.5% of Short conversations; m=6,036), with longer ones becoming rarer in the dataset as length increases. As sequence length increased, patterns inevitably became more fragmented: two-query sequences had 49 unique patterns with the top 5 covering 38.7%, while three-query sequences had 266 patterns with the top 5 covering only 16.2%.

Short Conversations Are Dominated by Debugging, Implementation, and Problem Orientation. Short conversations are primarily composed of three query types: *Verify & Fix*, *Implement*, and *Get Unstuck*. They account for 5 of the 10 most common three-query conversations (15.2%), 7 of the 10 most common two-query conversations (44.2%), and they were the top 3 most common single-query conversations (72.9%).

Most Short Conversations Are Formed by Common Query Patterns. Short conversations hold several recurring patterns.

Starting Conversations With Direction: While *Get Unstuck* is a fairly common query, among the top 10 short conversations it appears in only two positions: when beginning a conversation, or as a follow-up to another *Get Unstuck* query. These sequences make up 4 of the top 10 two-query conversations (21.7%) and 3 of the top 10 three-query conversations (6.4%). Afterward, some participants attempt the challenge and hit errors, others paste challenge materials for the tutor to interpret, and others ask for implementation.

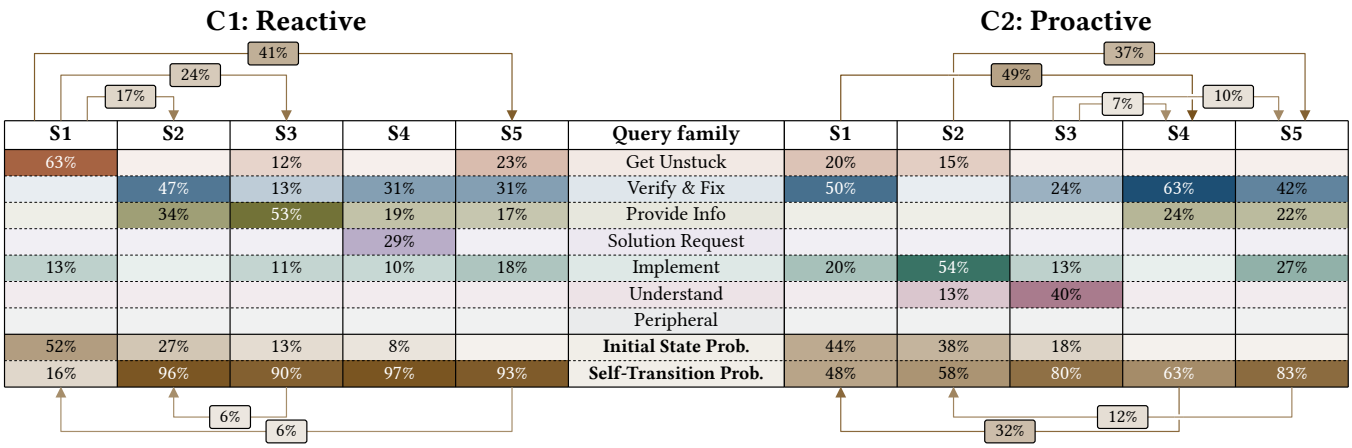
Building Up a Solution: Participants repeatedly build up a solution via multiple *Implement* queries, or *Implement* followed by *Verify & Fix* or *Provide Info*. These make up 3 of the top 10 two-query conversations (16.8%) and 3 of the top 10 three-query conversations (7.1%). For example, P289 in an AES challenge asked the following sequence of queries: *"do you need to pad this challenge?"*→*"how can I turn the hex of hKey to bytes"*→*"how can I determine what the IV is?"*, with each question digging one layer deeper into the challenge.

Repeated Debugging: In an effort to get a working solution, participants often use the AI tutor to repetitively debug code via consecutive *Verify & Fix*→*Verify & Fix* queries, provide additional information to a request via *Verify & Fix*→*Provide Info*, or fix an issue and then ask for an example implementation via *Verify & Fix*→*Implement*. These account for 3 of the top 10 two-query conversations (22.1%) and 6 of the top 10 three-query conversations (18.1%). Notably, repetitive *Verify & Fix* is also the most common two- and three-query conversation. While some participants appear to stall on the same issue, these queries can also show progression through the multiple steps a problem has. For instance, P277 repeatedly asked for help when solving multiple issues in their assembly, shifting from *"What might be causing my program to crash?"* in early requests to *"My code is now no longer crashing, but it is not properly lower-casing the strings"* in later requests.

4.3 Long Conversations

To understand longer conversational patterns, we analyze conversations of 4 queries or longer (m=11,692; 46.3%). Similar to prior work modeling complex patterns for problem-solving [34], we use an MHMM to identify patterns.

We use MHMMs as they are particularly well-suited for modeling sequences in which (1) sequences are of varied length and (2) observed outputs map to hidden states [39]. In our case, the *states* represent a student's goals at a given point in the conversation; in a particular state, a student is more likely to send queries of a particular type. A *cluster* represents a conversation style: a set of states (student goals) and transitions characterizing how students' goals shift during the conversation. Following best practices, our MHMM was fit to 11,692 conversations with at least 4 queries from 303 students (truncated to 22 queries in length [17]), and selected according to the Integrated Completed Likelihood criterion [3].



S1	S2	S3	S4	S5	Query family	S1	S2	S3	S4	S5
63%		12%		23%	Get Unstuck	20%	15%			
	47%	13%	31%	31%	Verify & Fix	50%		24%	63%	42%
	34%	53%	19%	17%	Provide Info				24%	22%
			29%		Solution Request					
13%		11%	10%	18%	Implement	20%	54%	13%		27%
					Understand		13%	40%		
					Peripheral					
52%	27%	13%	8%		**Initial State Prob.**	44%	38%	18%		
16%	96%	90%	97%	93%	**Self-Transition Prob.**	48%	58%	80%	63%	83%

Figure 2: Observed Conversation Styles – The conversation structure for m=11,692 query sequences. Columns are hidden states (S1–S5), rows are emission probabilities per query family, and the bottom two rows give initial-state and self-transition probabilities. Arrows show only transitions ≥5% and states show only emissions ≥10%.

Long Conversations Are Reactive or Proactive. As shown in Figure 2, our fitted MHMM yielded 2 clusters of 5 states, capturing two distinct conversation styles which we denote as "Reactive" and "Proactive", named based on their states and structural makeup. Broadly, Reactive styles account for 51.4% (m=6,008) of conversations; in these, participants provide context and ask the AI tutor to identify problems. The remaining 48.6% of conversations are Proactive, in which participants first produce their own solution to some extent and then ask the tutor to verify it, answer questions about specific subtasks of the challenge, or address targeted questions that help the student better understand the underlying concepts.

Reactive Conversations Begin With General Confusion, Proactive With Focused Questions. As the Initial State Prob. row of Figure 2 shows, the initial-state probabilities differed between Reactive and Proactive conversations. Notably, over half of Reactive conversations (52%) enter through a single state (S1) which was dominated by *Get Unstuck* queries, and 27% enter through a state focused on providing context and fixing bugs. This suggests Reactive conversations arrive either directionless or focused on fixing a bug.

Conversely, Proactive conversations enter primarily through two states: S1 (44%) and S2 (38%). While both feature some *Get Unstuck* queries to help with challenge orientation, bug-fixing and code implementation dominate S1, and code implementation dominates S2 with some focus on understanding. This may imply Proactive conversations begin with a targeted entry point rather than general disorientation, entering mid-debugging or with implementation tasks and concept questions.

Reactive Conversations Converge, Proactive Cycle. After starting at an initial state, the conversation may transition to other states; the probability of this is shown via the arrows and the "self-loop" probability in Figure 2. Together, these indicate the likelihood that the next query will remain in the same state and determine whether a state is transient (quickly exited), absorbing (likely to stay in the state), or a hub (cycled through repeatedly).

Reactive conversations quickly settle into one of four absorbing states: verification paired with context provision, context-dumping loops, confused-verification cycling, or verification with solution requests. For example, while working on a firewall challenge, P205 ended on a nine-query loop containing: *"REJECT?"*, *"it didnt do anytihng"*, *"this is what im using and its not doing anyting"*, *"so did not work"*, and pasted `iptables` commands. This oscillation showcases the absorbing verification/context loop. Proactive conversations more often cycle between states as students work through the challenge, potentially building on what came before. Conversations begin in three states with three different focuses: verification, implementation, or understanding. Three hub states (S1, S3, and S4) cycle among themselves: for example, students move from verification to context-checking and back (S1→S4 at 49%, S4→S1 at 32%).

Reactive Conversations Provide Info Ubiquitously, Proactive Selectively. Looking at Figure 2, we see *Provide Info* queries in all four absorbing states (S2–S5) in the Reactive cluster, while they only occur at notable rates in two states (S4 and S5) in Proactive conversations, indicating that participants in Reactive conversations utilize *Provide Info* queries more often compared to Proactive ones. Reactive conversations instead treat the tutor as an oracle: participants submit artifacts and wait for explanation. In S2 we see this as an absorbing verification-context loop, such as P194 submitting a seven-query sequence of pasted code, error messages, and *"how does this look?"* queries, without directly interpreting any of the tutor's responses. This mirrors the short-sequence evidence-provision pattern (Verify & Fix→Provide Info), but Short conversations resolve after a few turns while the Reactive absorbing topology sustains these loops for more exchanges.

Proactive conversations also contain context queries, but the context serves a different purpose. Rather than standing alone as an artifact for the tutor to interpret, context tends to co-occur with targeted queries. For example, P199 on a reverse-engineering challenge needed to craft a valid input for a parser, but rather than simply dumping code, P199 provided context and asked targeted questions: *"how can I find the pixel values I need?"* and *"if I use*

binary ninja where can I find the value I need to solve the challenge?", supplemented with decompiled C code to understand the required format. Here, each piece of context is evidence for a specific question or problem the student is working to overcome.

Reactive Conversations Ask for Solutions. A distinctive minority of Reactive conversations included explicit attempts to outsource problem-solving to the AI tutor. While both styles contain *Solution Request* queries, only the Reactive cluster contains a dedicated absorbing state where 29% of emissions are solution requests. Overall, 25.8% of Reactive conversations contain at least one *Solution Request* versus 11.8% of Proactive conversations, and 10.9% versus 1.8% contain more than one. When the tutor does not provide solutions, participants ask again, often with frustration. For instance, to complete a cryptography challenge, P113 sent six solution-request queries: *"provide me the commands of the challenge step to step"* and repeated it with minor variations until the tutor eventually provided enough information for a solution. When the AI tutor pushes back on solution requests, it asks students what they have tried and learned. P249, in one such instance, responded *"yes i tried"* and followed with *"give correct commands pleaseee"*. Lastly, we see participants pivot to solution requests after extended struggling, such as P133 asking the AI tutor *"can you just give me the answer?"* after 13 queries debugging their x86-64 code. This pattern is largely absent from Proactive conversations.

Proactive Conversations Ask for Insights. The Proactive cluster uniquely contained a state for conceptual inquiry (40% understanding, 24% verification) that participants cycled through to build understanding. For example, P117 on an assembly challenge asked three consecutive *Understand* queries to construct a mental model of the stack: *"could you explain how stack and base pointer are being accessed in this line of code?"*, then *"what are the differences between the stack pointer and base pointer?"*, then *"What is the impact of me reading the address of base pointer when input buffer initialized versus before read() being called?"* However, this sustained *Understand* query style is relatively rare; more often, *Understand* queries appear within implementation-heavy conversations to check understanding, e.g., *"so the correct password isn't 1010 but rather it is in bytes? and my goal is to convert it to said bytes?"* (P28). This allowed P28 to verify their mental model of the problem and continue.

Reactive Debugging Is AI-Driven, Proactive Debugging Is Student-Driven. While debugging is common across Reactive and Proactive conversations, they differ in how they relate to other potential states. In Reactive conversations, states with high debugging probabilities (S2, S4, and S5) are often not left, with self-looping transitions ≥93%. This may manifest as repetitive trial-and-error queries where participants repeatedly rely on the tutor to identify the problem rather than acting on the response before submitting another query. For example, P173 asked, *"can you check input.py and see if my code is correct"*. After the tutor suggested a fix, P173 asked for help implementing, *"how do I directly convert from binary to hex"*, then *"can you check input.py now"*. When the tutor confirmed P173's overall direction, they replied *"its not working"* and again asked, *"how do I encode using latin-1,"* then *"its not working"* again.

In contrast, Proactive states with high debugging probabilities (S1, S4, and S5) are more likely to cycle between understanding and implementation-focused states like S2. This takes the form of specific debugging queries that participants then use to build a solution through *Implement* or *Understand* queries. For example, P198 was working on a base64 challenge and queried *"I have this for the entered password so far [...] it matches that, but it is still incorrect."* After two debugging exchanges, the student identified the root cause as a newline character, *"is it because there is an endline?,"* and immediately pivoted to implementation in an attempt to fix the problem: *"how would I load this into a file to pipe into the challenge"*.

Module	Short	Reactive	Proactive
Linux	66.4% (2,524)	21.8% (827)	11.9% (451)
Data & Access	48.9% (1,482)	28.3% (856)	22.8% (691)
Web & SQL	60.4% (2,356)	19.5% (760)	20.1% (783)
Web Sec.	38.2% (1,075)	33.8% (951)	28.0% (789)
Comp. Arch.	58.8% (2,425)	18.2% (752)	23.0% (947)
Network Sec.	45.8% (1,421)	26.7% (828)	27.5% (854)
Cryptography	48.4% (1,032)	23.4% (500)	28.2% (602)
Reverse Engr.	56.3% (807)	21.3% (305)	22.4% (321)
Binary Sec.	48.5% (447)	24.8% (229)	26.7% (246)

Table 6: Conversation Style Distribution by Module – We present the distribution of conversations classified as Short, Reactive, or Proactive within each module.

Style Distribution Across Modules. We also find that the distribution of conversation styles varied across modules. Among long conversations, the distribution of Reactive vs. Proactive conversations varied significantly by module ($\chi^2(8) = 168.04$, $p < .001$; Cramér's $V = .120$). Table 6 reports the three-way breakdown by module. Short conversations were the most common style in every module but their share varied substantially (38.2%–66.4%), while Reactive conversations ranged from 18.2% to 33.8% and Proactive from 11.9% to 28.2%. The Linux and Web & SQL modules were dominated by Short conversations (66.4% and 60.4%, respectively), consistent with their simple structure where just one or two commands can solve the challenge. Conversely, Web Security had the lowest Short conversation share (38.2%) and the highest Reactive share (33.8%); its multi-step exploitation workflows (crafting payloads, observing outcomes, adjusting) naturally extend conversations beyond two or three queries. Among long conversations specifically, the Reactive share declined from 64.7% in the introductory module to roughly 44–49% from module 5 onward, suggesting students shifted toward Proactive conversations as they gained experience with the tutor; however, total conversation volume also declined across the semester (from 3,802 down to 922), so this shift may partly reflect survivorship bias rather than individual learning.

5 Performance Across Conversations (RQ2)

Having identified three conversation styles (§4.2 and §4.3), we examine how conversation styles relate to challenge completion.

Summary Statistics. By dividing the participants' 25,261 conversations across the style categories of Short (53.7%, m=13,569), Reactive (23.8%, m=6,008), and Proactive (22.5%, m=5,684), we aggregate their respective completion rates across modules in Figure 3.

Looking across styles, we find that Short conversations had the highest overall completion rate (94.6%), followed by Proactive

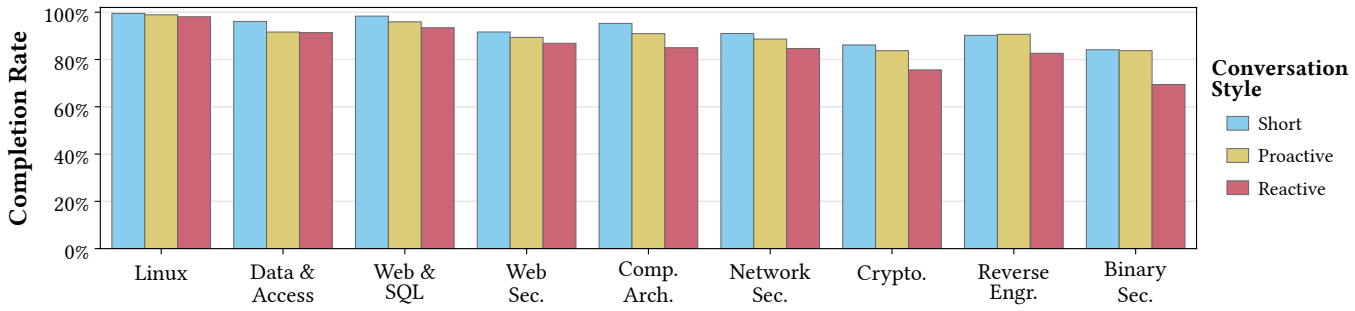


Figure 3: Module Completion by Conversation Style – Observed completion rates by module and style.

Factor	LR χ^2	df	p-value
Primary Effects			
Module	**965.27**	8	**<.001**
Conversation Style	**168.18**	2	**<.001**
Interaction			
Module × Conversation	**46.67**	16	**<.001**

Table 7: Factors Associated With Challenge Completion – Via likelihood ratio tests comparing our fitted mixed-effects model to one without each fixed effect, we find that all fixed effects are significantly associated with challenge completion. **Bold** indicates $p < .05$.

		Short–React		Short–Proact		Proact–React	
Module	m	OR	p	OR	p	OR	p
Linux	3,802	**3.64**	**.002**	2.37	.249	1.54	.695
Data & Access	3,029	**2.29**	**<.001**	**2.56**	**<.001**	0.89	.829
Web & SQL	3,899	**3.74**	**<.001**	**2.76**	**<.001**	1.35	.422
Web Sec.	2,815	**1.64**	**.004**	**1.57**	**.022**	1.05	.957
Comp. Arch.	4,124	**3.61**	**<.001**	**2.49**	**<.001**	1.45	.058
Network Sec.	3,103	**1.64**	**.001**	**1.51**	**.017**	1.09	.854
Cryptography	2,134	**1.75**	**<.001**	**1.64**	**.004**	1.07	.908
Reverse Engr.	1,433	1.44	.170	0.92	.927	1.57	.183
Binary Sec.	922	**1.92**	**.005**	1.19	.728	1.62	.113

Table 8: Association Between Conversation Styles and Completion Across Modules – We report the estimated associations between conversation styles and the likelihood of challenge completion. OR>1 indicates that AI tutor conversations of the first style are more likely to result in a completed challenge than conversations of the second style. **Bold** indicates $p < .05$; underline indicates $p < .10$.

(90.7%) and Reactive (87.5%). This ordering held in every module except Reverse Engineering, where Proactive conversations narrowly exceeded Short. Short conversations' consistently high completion rates likely reflect participants arriving with a specific question rather than an unresolved problem, not that brief interaction is itself more effective. Across modules, we find completion rates declined throughout the semester, from 99% in Linux to 80.4% in Binary Security, while the gap between Proactive and Reactive conversations widened from less than 1% to 14.3%.

5.1 Statistical Analysis

Given the repeated-measures structure of our data, where the same participants contribute conversations across challenges, we used a mixed-effects logistic regression to model this data appropriately. We modeled challenge completion as a function of conversation style, module, and the interaction effect between conversation style and module (i.e., conversation styles may have different effects depending on the module). To account for participant-specific differences in performance, we modeled participants as a random effect.

Conversation Style and Module Are Associated With Completion. To evaluate whether each fixed effect is significantly associated with challenge completion, we conduct Type II likelihood ratio tests [49]. As shown in Table 7, we find module ($\chi^2(8) = 965.27$, $p < .001$) and conversation style ($\chi^2(2) = 168.18$, $p < .001$) significantly predict challenge completion (**RQ2**). We also find a significant interaction between these effects ($\chi^2(16) = 46.67$, $p < .001$), indicating that the association between style and completion varies by module. To avoid misleading conclusions, we evaluate effects within each module rather than interpret main effects independently [38, pp. 164–169].

Reactive Conversations Correlate With Less Completion Than Short Conversations Across Most Modules. After performing separate Tukey-adjusted post-hoc pairwise tests between the different conversation styles for each module, we find that in eight of nine modules, Reactive conversations are associated with significantly lower completion rates than Short conversations (Table 8). Across these eight modules, odds ratios (ORs) ranged from 1.64 to 3.74 (a 39–73% decrease in the odds of completion for Reactive conversations). The Reactive–Short gap varied across modules: it was more pronounced in modules such as Computer Architecture and Binary Security, where failed attempts often produce minimal error messages that give little insight and do not identify the root cause of the failure. Because Reactive students rely on pasting their current state and asking the tutor for interpretation, an uninformative failure leaves the tutor little to work from, and it can only respond with generic suggestions or guesses. By contrast, the gap is smaller in modules such as Web Security or Network Security, where students craft a payload, observe its outcome, and adjust, so attempts can yield partial progress. Reverse Engineering was the one module where Reactive did not differ significantly from Short, and the only module in which completion rates for participants with Proactive conversations closely matched those with Short conversations (90.7% vs. 90.2%). Because the module focuses on iteratively

constructing inputs to satisfy a parser and provides specific error messages (e.g., "ERROR: Invalid magic number!", "ERROR: Incorrect width!"), both Reactive and Proactive conversations follow similar test-then-fix patterns. This tight feedback loop may explain the small difference between Reactive and Short conversations, since students do not have to guess why their solution fails.

Proactive Conversations Are Associated With Less Completion Than Short Conversations Across Several Modules. Proactive conversations also showed significantly lower completion than Short conversations across several modules, with odds ratios ranging from 1.51 to 2.76 (a roughly 34–64% decrease for Proactive conversations) in six of nine modules. We did not find any differences for the other three modules. The first was the introductory Linux module, where challenges could be solved by running one or two commands; we observed significantly fewer Proactive conversations and all types had high completion rates, likely contributing to the lack of any observed differences. The second was Reverse Engineering, which as discussed previously has built-in feedback that reduces the difference between conversation styles. Finally, Binary Security also showed no significant difference, likely due to a smaller sample size and course attrition limiting statistical power.

Pairwise Proactive vs. Reactive Contrasts Did Not Reach Significance Within Any Module. Although Proactive conversations showed equal or higher raw completion rates than Reactive in all nine modules, no per-module contrast achieved statistical significance. The closest were the Computer Architecture (OR = 1.45 [0.99, 2.12], $p = .058$) and Binary Security (OR = 1.62 [0.92, 2.84], $p = .113$) modules. While this may be because modules like Binary Security are the smallest and may have limited statistical power, we cannot draw any conclusions.

Neither Student Background nor AI Use Predicts Challenge Completion. To investigate whether students' background, i.e., self-reported prior cybersecurity knowledge (Q1), or frequency of general AI use outside the course (Q2) predict challenge completion, we added these self-reported metrics as independent variables in our mixed-effects regression described above and re-ran our analysis. We *do not* find evidence that either factor significantly correlates with performance, while the statistically significant associations with module and conversation style remain when controlling for student background (see the extended version [42] for details).

6 Perceptions of the AI Tutor (RQ3)

We now present students' reported perceptions of our AI tutor via our end-of-semester survey (see the extended version [42]) with n=293 participants.

6.1 Utility and Usability of the AI Tutor

Participants rated the tutor's utility and usability on a Likert scale across seven items (Figure 4), and answered two open-response questions on which tasks the AI tutor was most or least useful.

While Useful, the Tutor Is Better at Solving Issues Than Alleviating Confusion. Roughly half of participants agreed that the tutor helped them understand concepts (55.6%; n=163), complete challenges (51.5%; n=151), find bugs (49.1%; n=144), and alleviate

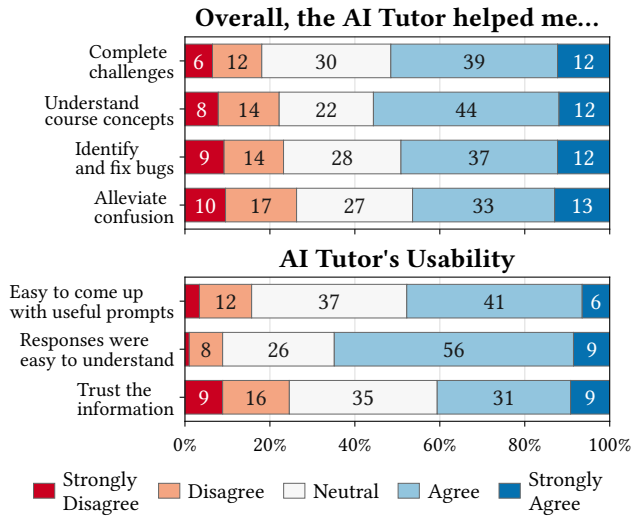


Figure 4: Perceived Utility and Usability of the AI Tutor – Participants' Likert ratings of the AI tutor's usefulness (Q4–Q7) and usability (Q10–Q12) across tasks.

confusion (46.4%; n=136). To evaluate whether these differences between task types are significant, we conducted a Friedman test and found that significant differences existed ($\chi^2(3) = 13.21$, $p = .004$). Then, via post-hoc pairwise Wilcoxon signed-rank tests with Bonferroni correction, we found that participants agreed that the tutor helped more in completing challenges (W = 3,034.5, $p = .01$) and understanding concepts (W = 3,187.0, $p = .04$) than in alleviating confusion.

Students Valued Direction-Seeking but Were Split on Debugging. When asked which tasks the AI tutor was most and least useful for (Q8, Q9), *direction-seeking and reconnaissance* was the most frequently cited task (n=169) and predominantly positive (119 pos. vs. 50 neg.), with P47 finding the tutor useful for *"getting some sort of direction in solving the problems... a good initial idea to work off of."* *Concept guidance* (n=53) and *code generation* (n=49) were less frequently cited but also largely positive (39 vs. 14 and 38 vs. 11, respectively), with P120 stating that it was helpful for *"conceptual questions and [they] would get a useful response[s]... it was great at breaking down lines of code, if I saw a new unfamiliar function[s] or one I had forgotten."* *Debugging* (n=103) drew the most mixed assessments, with slightly more negative than positive mentions (54 neg. vs. 49 pos.) and participants praising the tutor's ability to catch syntactic errors but finding it unhelpful for deeper reasoning.

Responses Were Easier to Read Than to Prompt For or Trust. Most found responses easy to understand (64.8%; n=190), and roughly half agreed it was easy to create useful prompts (47.8%; n=140). However, trust was notably lower: only 40.6% (n=119) agreed they could trust the tutor's information. A Friedman test confirmed significant differences ($\chi^2(2) = 46.29$, $p < .001$). Post-hoc tests showed responses were easier to understand than to prompt for (W = 2,464.0, $p < .001$) or to trust (W = 2,960.0, $p < .001$), and that prompting was rated higher than trust (W = 5,216.0, $p = .005$).

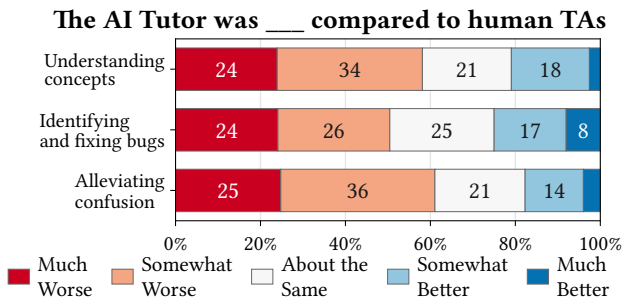


Figure 5: AI Tutor vs. TA Comparison – Participants' ratings comparing the AI tutor to human TAs (Q14–Q16).

Unfavorable Ratings Reflect Tutor Limitations, Not Ineffective Use. If unfavorable ratings reflected ineffective use rather than the tutor's limitations, participants with more Proactive conversations should rate the tutor more favorably. We correlated each participant's Proactive share (Proactive / (Proactive + Reactive) among their long conversations) with their utility and usability ratings (Q4–Q7, Q10–Q12; Spearman, Holm-corrected; n=289, as 4 of the 293 participants had only Short conversations and thus no defined share). No individual rating item was associated with Proactive share after correction except trust: all utility items had Holm-adjusted $p \geq .50$ (Q4–Q7 composite: $\rho = -.10$, $p = .10$, uncorrected), while trust (Q12) significantly decreased with Proactive use ($\rho = -.23$, Holm-adjusted $p < .001$). Unfavorable ratings therefore appear to reflect the tutor's limitations rather than ineffective use, consistent with its reported decline on harder modules (§6.3).

6.2 Comparison to Human TAs

Participants rated the AI tutor against human TAs on a Likert scale across three items (Figure 5) and answered one open-response question comparing the two.

Students Preferred TAs for Quality but Valued the Tutor's Availability. Participants who had not used the course TAs for the specified use case could answer "N/A" to these items (e.g., if a participant had not used a TA to help them debug), so the number of comparisons varies by question (Figure 5). Among those who did provide a comparison, a majority rated the tutor worse across all dimensions: alleviating confusion (61.1%; 138 of 226), understanding concepts (58.1%; 136 of 234), and finding bugs (50.4%; 113 of 224). Among the 217 participants who rated all three items, a Friedman test confirmed significant differences across them ($\chi^2(2) = 8.23$, $p = .016$). Post-hoc corrected tests revealed that bug-fixing was rated significantly better than both concept understanding (W = 2,515.5, $p = .021$) and confusion alleviation (W = 1,765.5, $p = .011$), while the latter two did not differ significantly (W = 2,186.5, $p = .58$). In contrast, in the free-response TA comparison (Q17), the most commonly cited advantages of the tutor were *availability* (n=25) and *lower social barriers* (n=8). P48 noted *"I can use [AI tutor] whenever I want… that's the most dominant advantage,"* and P79 highlighted that they didn't feel judged by the AI tutor.

6.3 Reasons for Stopping AI Tutor Use

In Q13, 168 (57.3%) participants reported stopping or reducing their use of the tutor during the semester; 157 of these provided a codable reason, which we now evaluate.

Difficulty With Harder Challenges. The most frequently cited reason for stopping use was that the tutor was *bad at harder challenges* (n=77; 45.8% of those who reported stopping), reinforcing participants' reports that the tutor was useful early but less so on harder material. Participants consistently reported the tutor's usefulness dropping sharply across modules 5–7, corresponding to the modules where the Reactive–Short completion gap widens substantially (Figure 3). Further, we observe a decrease in conversations from module 5 onward (per-module distributions in the extended version [42]). P20 noted, *"around cryptography, I quickly found out [AI tutor] was going to be of no help,"* and P36 reported, *"the modules just became too complicated."*

The difficulties participants cited were often not generic complaints but mapped to module characteristics. For example, modules with opaque errors such as Computer Architecture or Binary Security made it difficult for students to observe partial progress. Many participants complained of generic debugging advice, e.g., *"stopped [using AI tutor] once we got to working with the stack/IDA as it just repeated steps that I found confusing"* (P289). In a conceptually driven module like Cryptography, participants reported trouble translating the AI tutor's explanations into implementation, such as P114 finding the tutor *"useful in understanding what each encryption did"* but *"failing to help users construct a successful script caused me to stop using [AI tutor]"*. This pattern of opaque errors yielding generic guidance surfaced often and aligns with our quantitative findings that the completion gaps between styles widen the most in these modules.

Poor Responses. *Repetitive responses* (n=28), *wrong or misleading information* (n=20), and *vague responses* (n=9) further drove discontinuation. Repetitive response complaints describe the student-side observations of the *absorbing states* identified in Reactive conversations (§4.3). Where the MHMM captures these as cycles of sustained verification or context-dumping loops, participants described the same phenomenon: P93 reported the tutor *"just keeps going in the same circle which did not take me anywhere,"* and P162 said the tutor would *"copy [and] paste the same thing."* Wrong or misleading information (n=20) compounded the issue: P5 noted the tutor was sometimes *"blatantly incorrect,"* and they would have to correct it.

Preference for Other Resources. Thirteen participants reported switching to other resources entirely, commonly citing the course Discord and other LLMs: P125 *"became more active in the discord and found it much more useful and constructive,"* while P305 *"did switch over to using ChatGPT o3… because it did sometimes seem like [AI tutor] would not really think that hard about the problem."*

Educational Concerns and Outgrowing the Tutor. Finally, ten participants expressed *educational concerns*, worrying that relying on the tutor might hinder their own learning. P26 stated *"at some point I just wanted to understand for myself what to do and [AI tutor] can't do that."* Similarly, 11 participants reported *outgrowing* the tutor, finding that their skills had advanced beyond what it could offer, such as P78 stating: *"I stopped needing it as much to give me syntax."*

7 Discussion

In summary, we find that students used the AI tutor heavily, submitting 142,526 queries across seven distinct categories, with the most common being *Verify & Fix* (checking students' work, 34.7%), *Provide Info* (supplying context, 21.8%), and *Implement* (coding a solution, 17.2%) queries (**RQ1**). Most conversations were short and single-purpose, while longer conversations separated into two styles: Reactive conversations, characterized by context-dumping, absorbing verification loops, and AI-driven debugging, and Proactive conversations, characterized by student-driven problem solving and targeted inquiry that cycles between implementation and understanding (§4.1, §4.3). We then examine how these styles relate to challenge completion (**RQ2**), finding that Short conversations had the highest completion rate (94.6%), followed by Proactive (90.7%) and Reactive (87.5%), and that the gap between Proactive and Reactive conversations widened as material grew harder, from under 1% in the first module to 14.3% in the most advanced (§5.1). Finally, we asked students for their opinions on the AI tutor (**RQ3**), finding that roughly half judged it useful, that a majority rated it worse than human TAs on every dimension measured, and that 57.3% stopped or reduced their usage during the semester. From these results, we consider how these conversation styles relate to established help-seeking behavior, provide recommendations for educators deploying AI tutors, and discuss implications for the design of future systems.

7.1 Conversation Styles in Help-Seeking Theory

Our identified Reactive and Proactive conversation styles map onto established help-seeking profiles [19]: Reactive conversations resemble *executive* help-seeking, where students seek assistance to complete tasks without deeply engaging with the underlying material, and Proactive conversations resemble *adaptive* help-seeking, where students build understanding through deliberate, focused requests. This is consistent with the longstanding finding that question quality, not frequency, correlates with achievement [12], and with traditional CS domains where unproductive help-seeking behaviors such as hint abuse predict poorer performance [18, 24, 27]. Our work extends these findings to hands-on cybersecurity education, where the open-ended and cross-domain nature of the challenges makes the distinction between adaptive and executive help-seeking even more important. Because these are known student behaviors rather than something our tutor created, they are both teachable and detectable.

7.2 Recommendations for Educators

Deploy AI Tutors to Complement, Not Replace, TAs. Students preferred human TAs across every measured dimension and often complained that the AI tutor's usefulness declined on harder challenges, though they found it useful for scaffolding and syntactic debugging (§6). Notably, these unfavorable assessments were not confined to students using the tutor ineffectively: ratings did not improve with more Proactive use (§6.1), suggesting they reflect genuine tutor limitations. Students also rarely asked the tutor for conceptual help: conceptual queries made up only 7.8% of total volume, and unlike a human TA who may organically weave conceptual explanation into debugging help, the AI often only did so when students explicitly asked. We therefore recommend deploying AI tutors as a complement to human help, and framing the tutor as a resource for problem orientation and debugging, while encouraging students to seek human help for conceptual depth.

Educate Students on How to Use AI Tutors. Students shifted between Reactive and Proactive conversations across the semester, and Proactive conversations showed equal or higher raw completion rates than Reactive in every module. No single module reached significance (§5.1), though the average across modules did (see the extended version [42]). Encouraging Proactive engagement therefore remains a defensible, if modest, recommendation. We also did not observe systematic prompt engineering attempts to circumvent Socratic guardrails. The few solution-seeking attempts consisted of repetition under frustration, suggesting light query auditing plus Socratic guardrails could be enough in practice. Educator effort therefore could be better directed at metacognitive feedback [40] such as self-assessment, structured reflections, or requiring more detailed reasoning from the student when requesting help, thus nudging students to more Proactive approaches. Educators can also build clearer feedback into their challenges, since the tutor's guidance degrades when a failed attempt gives little indication of what went wrong.

7.3 Opportunities for AI Tutor System Design

As difficulty increases, challenges shift away from known procedures toward open-ended exploration, and the gap between Reactive and Short conversations widens (Figure 3), aligning with student reports that the tutor became less useful on harder modules (§6.3). The tutor's Socratic system prompt constrains it to guide without solving, but as challenges become exploratory there is less procedure to scaffold and guidance becomes more generic. This is clearest in opaque-error domains (where diagnosing failures requires debugging context students lack) and in abstract-concept domains (e.g., Cryptography where bridging explanation to implementation risks revealing the answer). In both cases the tutor remains underinformed even when students try to compensate: students supplied context manually in 21.8% of queries (§4.1) despite the system already including their file and terminal state. Nonetheless, Proactive engagement retained its numerical advantage over Reactive on harder material (§5.1).

Reducing Context Burden on Students. This suggests future AI tutor designs should surface what context the system has access to and provide easy ways for students to choose what to include.

Detecting Ineffective Student Conversations. Our study shows that conversations can be classified into Reactive and Proactive styles. Future systems could detect Reactive patterns and intervene or escalate to an instructor or TA, treating these patterns as diagnostic markers of student engagement rather than purely causal indicators, consistent with the substantial between-student component of the association we observe (see the extended version [42]).

Discovering Areas for Improvement Within Courses. In addition to ineffective patterns, we also find students often share when they are frustrated, on the wrong path, or generally confused about how to proceed (§4.1). This information serves as a window into the hidden issues that plague courses. Future research could investigate whether consistent indicators of confusion can automatically

identify either curriculum gaps or prevalent misunderstandings, and then alert instructors to areas they may want to address.

Observational Study as a Stepping Stone. As the first large-scale observational study of AI tutor use in cybersecurity education, this work establishes what interaction patterns exist and how they relate to outcomes. Future controlled studies should test whether the within-student association reported in the extended version [42] can be increased through various means such as intervention, training, or system design, and measure metacognitive skills directly, such as through surveys or think-aloud protocols, which may be a remaining unmeasured driver of both conversation style and performance. Given evidence that self-efficacy, prior performance, and metacognitive preparedness shape how novices engage LLM assistants [22, 36], future work should also examine whether student background predicts conversation-style adoption itself, which our completion-side covariate analysis (see the extended version [42]) does not address. Although we cannot show that AI tutors affected grades relative to other resources in our setting (see the extended version [42]), they generate fine-grained behavioral data that can help educators identify struggling students at scale.

8 Conclusion

We present the first large-scale, in-situ measurement of how students used an embedded AI tutor over a semester of upper-division cybersecurity coursework. The identified interaction patterns mirror help-seeking behaviors documented before LLMs, suggesting AI tutor design choices interact with longstanding student-side behaviors rather than creating new ones. Our findings reframe conversation styles as observable signals of student engagement, and show that the tutor's usefulness falls off for harder material. These results inform AI tutor design and educator best practices, and provide a foundation for future research.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001124C0362. 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 Defense Advanced Research Projects Agency (DARPA).

References

[1] Hamsa Bastani et al. Generative AI without guardrails can harm learning: Evidence from high school mathematics. *Proceedings of the National Academy of Sciences*, 2025.
[2] Lina Battestilli, Matthew Zahn, and Sarah Heckman. Academic Help Seeking Patterns in Introductory Computer Science Courses. In *Proc. of ASEE*, 2022.
[3] C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.
[4] Doga Cambaz and Xiaoling Zhang. Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review. In *Proc. of SIGCSE*, 2024.
[5] Edward Y Chang. Prompting large language models with the Socratic method. In *Proc. of IEEE CCWC*, 2023.
[6] Juliet Corbin and Anselm L. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications, fourth edition, 2014.
[7] Paul Denny, Viraj Kumar, and Nasser Giacaman. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proc. of SIGCSE*, 2023.
[8] Yuyang Ding et al. Boosting large language models with Socratic method for conversational mathematics teaching. In *Proc. of CIKM*, 2024.
[9] Mahmoud Elkhodr and Ergun Gide. Integrating Generative AI in Cybersecurity Education: Case Study Insights on Pedagogical Strategies, Critical Thinking, and Responsible AI Use. https://arxiv.org/abs/2502.15357, 2025.
[10] Lucile Favero, Juan Antonio Pérez-Ortiz, Tanja Käser, and Nuria Oliver. Enhancing critical thinking in education by means of a Socratic Chatbot. In Francisco Bellas and Oscar Fontenla-Romero, editors, *AI in Education and Educational Research*, 2025.
[11] Kelsey R. Fulton et al. Vulnerability Discovery for All: Experiences of Marginalization in Vulnerability Discovery. In *Proc. of IEEE Symposium on Security and Privacy*, 2023.
[12] Arthur C. Graesser and Natalie K. Person. Question asking during tutoring. *American Educational Research Journal*, 1994.
[13] Satu Helske and Jouni Helske. Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R. *Journal of Statistical Software*, 2019.
[14] Chenyu Hou et al. Prompt-based and Fine-tuned GPT Models for Context-Dependent and -Independent Deductive Coding in Social Annotation. In *Proc. of LAK*, 2024.
[15] Irene Hou, Hannah Vy Nguyen, Owen Man, and Stephen MacNeil. The Evolving Usage of GenAI by Computing Students. In *Proc. of SIGCSE*, 2025.
[16] ISC2. ISC2 Cybersecurity Workforce Study: Looking Deeper into the Workforce Gap. Technical report, ISC2, 2024. https://www.isc2.org/Insights/2024/10/ISC2-2024-Cybersecurity-Workforce-Study.
[17] Renzhi Jing and Ning Lin. Tropical Cyclone Intensity Evolution Modeled as a Dependent Hidden Markov Process. *Journal of Climate*, 2019.
[18] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. The Impact of Large Language Models on Programming Education and Student Learning Outcomes. *Applied Sciences*, 2024.
[19] Stuart A. Karabenick and Eleftheria N. Gonida. Academic help seeking as a self-regulated learning strategy: Current issues, future directions. In *Handbook of self-regulation of learning and performance*. Routledge/Taylor & Francis Group, 2nd edition, 2018.
[20] Majeed Kazemitabaar et al. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. In *Proc. of CHI*, 2024.
[21] Xiner Liu et al. Qualitative Coding with GPT-4: Where it Works Better. *Journal of Learning Analytics*, 2025.
[22] Lauren E. Margulieux et al. Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. In *Proc. of ITICSE*, 2024.
[23] Jim Marquardson. Embracing Artificial Intelligence to Improve Self-Directed Learning: A Cybersecurity Classroom Study. *Information Systems Education Journal*, 2024.
[24] Samiha Marwan, Anay Dombe, and Thomas W. Price. Unproductive Help-seeking in Programming: What it is and How to Address it. In *Proc. of ITICSE*, 2020.
[25] James Mattei, Christopher Pellegrini, Matthew Soto, Marina Sanusi Bohuk, and Daniel Votipka. "I'm trying to learn... and I'm shooting myself in the foot": beginners' struggles when solving binary exploitation exercises. In *Proc. of USENIX Security Symposium*, 2025.
[26] Mary L. McHugh. Interrater reliability: the kappa statistic. *Biochemia Medica*, 2012.
[27] Hunter McNichols, Fareya Ikram, and Andrew Lan. The StudyChat Dataset: Analyzing Student Dialogues With ChatGPT in an Artificial Intelligence Course. In *Proc. of LAK*, 2026.
[28] Madhav Mukherjee, John Le, and Yang-Wai Chow. Generative AI-Enhanced Intelligent Tutoring System for Graduate Cybersecurity Programs. *Future Internet*, 2025.
[29] Connor Nelson, Adam Doupé, and Yan Shoshitaishvili. SENSAI: Large Language Models as Applied Cybersecurity Tutors. In *Proc. of SIGCSE*, 2025.
[30] Connor Nelson and Yan Shoshitaishvili. DOJO: Applied Cybersecurity Education in the Browser. In *Proc. of SIGCSE*, 2024.
[31] Connor Nelson and Yan Shoshitaishvili. PWN The Learning Curve: Education-First CTF Challenges. In *Proc. of SIGCSE*, 2024.
[32] Connor Nelson, Robert Wasinger, Adam Doupé, and Yan Shoshitaishvili. Open Cybersecurity Education: Five Years of pwn.college. In *Proc. of SIGCSE*, 2026.
[33] OpenAI. Update to GPT-5 System Card: GPT-5.2. https://openai.com/index/gpt-5-system-card-update-gpt-5-2/, 2025.
[34] Fan Ouyang, Weiqi Xu, and Mutlu Cukurova. An artificial intelligence-driven learning analytics method to examine the collaborative problem-solving process from the complex adaptive systems perspective. *International Journal of Computer-Supported Collaborative Learning*, 2023.
[35] Julieto Perez and Ethel Ong. Designing an LLM-Based Dialogue Tutoring System for Novice Programming. In *Proc. of ICCE*, 2024.
[36] James Prather et al. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proc. of ACM ICER*, 2024.
[37] Victor-Alexandru Pădurean, Paul Denny, Alkis Gotovos, and Adish Singla. Prompt Programming: A Platform for Dialogue-based Computational Problem Solving

with Generative AI Models. In *Proc. of ITICSE*, 2025.
[38] Helen C Purchase. *Experimental human-computer interaction: a practical guide with visual examples*. Cambridge University Press, 2012.
[39] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 1989.
[40] Ido Roll, Vincent Aleven, Bruce M. McLaren, and Kenneth R. Koedinger. Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction*, 2011.
[41] Brad Sheese, Mark Liffiton, Jaromir Savelka, and Paul Denny. Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant. In *Proc. of ACE*, 2024.
[42] Michael Tompkins, Nihaarika Agarwal, Ananta Soneji, Robert Wasinger, Connor Nelson, Kevin Leach, Rakibul Hasan, Adam Doupé, Daniel Votipka, Yan Shoshitaishvili, and Jaron Mink. Do Hackers Dream of Electric Teachers?: A Large-Scale, In-Situ Measurement of Cybersecurity Student Behaviors and Educational Performance with AI Tutors (Extended Version). arXiv:2602.17448, 2026.
[43] William Triplett. AI-Enhanced Cyber Science Education: Innovations and Impacts. *Information*, 2025.
[44] Mickey Vellukunnel et al. Deconstructing the Discussion Forum: Student Questions and Computer Science Learning. In *Proc. of SIGCSE*, 2017.
[45] Daniel Votipka, Rock Stevens, Elissa M. Redmiles, Jeremy Hu, and Michelle L. Mazurek. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. In *Proc. of IEEE Symposium on Security and Privacy*, 2018.
[46] Daniel Votipka, Eric Zhang, and Michelle L Mazurek. Hacked: A pedagogical analysis of online vulnerability discovery exercises. In *Proc. of IEEE Symposium on Security and Privacy*, 2021.
[47] Jan Vykopal, Valdemar Švábenský, and Ee-Chien Chang. Benefits and Pitfalls of Using Capture the Flag Games in University Courses. In *Proc. of SIGCSE*, 2020.
[48] Yang Wang, Margaret McCoey, Qian Hu, and Maryam Jalalitabar. Teaching Security in the Era of Generative AI: A Course Design of Security + ChatGPT. In *Proc. of IEEE ISEC*, 2024.
[49] Bodo Winter. Linear models and linear mixed effects models in R with linguistic applications. arXiv:1308.5499, 2013.
[50] Ruiwei Xiao et al. A Preliminary Analysis of Students' Help Requests with an LLM-Powered Chatbot when Completing CS1 Assignments. In *Proc. of CSEDM*, 2024.
[51] Stephanie Yang, Hanzhang Zhao, Yudian Xu, Karen Brennan, and Bertrand Schneider. Debugging with an AI Tutor: Investigating Novice Help-seeking Behaviors and Perceived Learning. In *Proc. of ACM ICER*, 2024.

A Ethical Considerations

All procedures were approved by Arizona State University's IRB and were designed to be FERPA-compliant, given the inclusion of student participants and use of course assignments. We detail each stakeholder with its risks and our mitigations, then discuss our decision to conduct and publish this research.

Student Participants in Course.

Ensuring Autonomy and Consent in Participation. We obtained active consent at two points, before students began using the AI tutoring system, and after they submitted their end-of-semester survey. Students who declined either were excluded and their data was not analyzed, and participants could ask to be removed at any time. Participation carried no preferential treatment, the 1% extra credit was awarded for completing the end-of-semester survey regardless of consent, and students could use the AI tutor without joining the study.

Ensuring Privacy. Data collection began only after informed consent. We anonymized conversation logs, and query metadata held no student PII or FERPA-relevant information. Students could still type PII into a query, but the system only exists to help with challenges and students knew their queries went to the platform's AI system, so we judged the risk no greater than using a typical AI system. Model training was disabled for the APIs powering the tutor. We sent the API only the student's query, our own course content, and the challenge's system context. Our IRB covered this use of the course content, and the context comes from a course Docker container that holds no student data and exists only during the attempt.

Impacts of Tutor on Grades and Learning. Introducing an AI tutor into a required course risks affecting how students learn, and we do not assume the effect is positive. Students already use commercial LLMs for coursework, so our tutor offers a more controlled and observable alternative rather than a new exposure. We believe systematically studying these effects provides societal benefits that outweigh the marginal risks. Tutor use was never mandated, and TAs communicated the limitations of these systems at the start of the semester. Longer-term studies are needed to assess whether AI tutor use affects skill retention and preparation for subsequent courses.

Student Non-Participants in Course. Students who did not consent had access to the same platform, course materials, and AI tutoring system. We collected no research data from them, and opting out carried no penalty and could be done at any time.

Course Instructors. Course instructors are co-authors of this work and were involved in the study design. The AI tutor supplemented existing instructional support channels, rather than replacing instructor or TA involvement. We present our findings to inform similar systems, not to diminish the role of human educators.

General Cybersecurity Students and Instructors. Publication of this work could influence how cybersecurity courses integrate LLM-based tutoring tools. We present both the strengths and weaknesses of the AI tutor to avoid overstating its effectiveness, and we discuss failure modes to help instructors make informed decisions.

Decision to Conduct and Publish Research.

Conducting Research. As described above, our study carries a minimal risk profile, which we determined was outweighed by the potential to advance understanding of LLM-assisted cybersecurity education at scale.

Publishing Research. The primary risk of publication is re-identification of participants, but we believe this risk is low as all reported data is aggregated and anonymized, and we did not find any concerns sufficient to warrant withholding publication.

B Open Science

We provide all analysis scripts necessary to reproduce our findings, including the LLM-based coding pipeline with our finalized codebook prompts, the mixed-effects logistic regression analyses and the scripts for processing the reflection data, along with synthetic sample data for each pipeline stage. To minimize re-identification risk, we do not release raw data. These are available at: https://github.com/frqmod/SENSAI_behavior_CCS_artifacts Appendix material omitted here for space is provided in full in the extended version [42], at: https://arxiv.org/abs/2602.17448

C Generative AI Usage

We did not use generative AI tools to draft, rewrite, or substantially edit the main text of this submission; AI-based grammar checkers (e.g., Grammarly) were used for light editorial polishing. As described in §3, we used GPT-5.2 [33] to apply our finalized codebook to all 142,526 queries. The codebook was developed entirely through human coding before any LLM involvement.