Persona Reinforcement for Secure Programming AI Tutors: Adaptive Assistance in Action

Adithya Harish Srinivasan Manikandan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science and Application

Eli Tilevich, Chair
Dwayne C Brown
Na Meng

December 04, 2025
Blacksburg, Virginia

Keywords: Large Language Models, Secure programming education, intelligent tutoring systems, plan of thought, prompt engineering, generative AI, adaptive tutor

Persona Reinforcement for Secure Programming AI Tutors: Adaptive Assistance in Action

Adithya Harish Srinivasan Manikandan

(ABSTRACT)

The world needs safe software, but producing it requires secure programming skills. Unfortunately, effectively teaching students secure coding skills remains a critical open problem. Pedagogy suggests that students acquire skills best through a combination of conceptual reasoning and practical experience. In computing, students gain this practical experience through hands-on programming exercises and projects. In the age of generative AI, modern tools, such as large language models (LLMs), often hinder effective learning by providing solutions to coding problems without engaging students in the learning process. Instead of internalizing the key concepts, students can simply copy answers without understanding, undermining the learning objectives. Unexpectedly, generative AI offers a promising opportunity to address the problem it has created, provided appropriate constraints and design. To that end, we present **S**ecure **P**rogramming with **A**daptive **R**easoning **C**ompanion (**SPARC**), an AI-powered tutor designed to guide students through secure programming exercises, rather than directly provide solutions. Our design reinforces **SPARC**'s tutor persona through a confluence of three techniques: (1) tailored prompt engineering, (2) a novel combination of AI techniques—coined as a *learning safeguard proxy*—designed to prevent the tutor from directly providing solutions, and (3) a responsive algorithm that adapts responses to student proficiencies. We have integrated **SPARC** with SecureCoder, a drill-and-practice platform for secure coding skills, and evaluated its effectiveness via a pilot study. Across 120 study sessions (80 with **SPARC** and 40 with GPT-4o-mini), **SPARC** facilitated a 95%

exercise completion rate compared to 80% for GPT-4o-mini, and pilot study participants demonstrated statistically higher satisfaction with SPARC's adaptability than GPT-4o-mini. Further, unlike GPT-4o-mini, all interactions with SPARC avoided providing participants with complete solutions. Finally, our study demonstrated that more than 85% of participants found SPARC's guidance to be clear, adaptive, and helpful, with 80% reporting improved understanding of secure programming concepts. Our evaluation suggests that SPARC's novel design achieves its goal of serving as a secure programming tutor. SPARC provides helpful guidance that most students found to enhance their learning experiences. As secure programming skills are vitally important, this work contributes to secure computing education by employing generative AI as an educator's ally, rather than its adversary.

Persona Reinforcement for Secure Programming AI Tutors: Adaptive Assistance in Action

Adithya Harish Srinivasan Manikandan

(GENERAL AUDIENCE ABSTRACT)

Software powers nearly every aspect of modern life, but software insecurity can hurt both people and organizations. Learning how to write secure code is therefore essential, yet teaching this skill effectively remains a challenge. With the rise of generative AI tools such as ChatGPT, many students rely on these systems to receive direct solutions, thus undermining the learning process. This thesis introduces **S**ecure **P**rogramming with **A**daptive **R**easoning **C**ompanion or **SPARC** for short. This AI-powered tutor helps students learn secure programming by guiding them step by step instead of providing direct answers. **SPARC** encourages reasoning and problem solving by offering hints that adapt to each student's skill level. The system is integrated with SecureCoder, a drill-and-practice platform for secure coding, through which students can apply these adaptive hints when solving problems. In a pilot study involving 20 participants, students using **SPARC** demonstrated greater success and engagement compared to those using a GPT model. Most participants described **SPARC** 's hints as clear, helpful, and adaptive to their needs, noting that it deepened their understanding of secure coding. The detailed evaluation showed that **SPARC** effectively fulfills its goal as a secure programming tutor. By shifting the role of generative AI from one that undermines education to an active learning assistant, this research demonstrates how AI can effectively support rather than replace the educational process.

Dedication

To my beloved parents, Manikandan and Subha, for shaping my dreams with their love, sacrifices, and unwavering faith in me, and to my dear sister Sruti.

Dedicated to my dad, who missed earning his master's in the US due to circumstances—this work fulfills that journey and reflects the inspiration behind it.

Acknowledgments

I owe a huge debt of gratitude to Dr. Eli Tilevich for his invaluable mentorship, guidance, and unwavering support throughout my graduate studies. I consider myself truly fortunate to have had an advisor who not only guided my research but also supported me in every aspect of my academic and professional life. His encouragement during my job search, assistance with academic funding, and constant concern for my progress made a profound difference in my experience at Virginia Tech. More than an advisor, he has been a mentor whose trust, honesty, and belief in me have shaped both my research and my personal growth. I will always remain deeply thankful to him, no matter what I achieve in life.

I am sincerely thankful to my committee members, Dr. Na Meng and Dr. Chris Brown, for their valuable time, feedback, and constructive insights that greatly strengthened this work. Their perspectives helped refine both the technical and pedagogical aspects of this research.

To my peers in the Software Innovations Lab, I owe special thanks to Leo St. Amour, who originally created the SecureCoder platform that formed the foundation for my research, for his generosity in sharing his work and for offering consistent help and guidance; and to Provakar Mondal, for his constant support, thoughtful discussions, and willingness to help whenever needed.

I would like to acknowledge the Department of Computer Science at Virginia Tech for giving me the opportunity to serve as a Graduate Teaching Assistant.

Thanks to the small group of peers I personally reached out to, who were generous with their time in participating in the survey.

Contents

List of Figures

List of Tables

List of Abbreviations

ARR Adaptive Response Rate

LSP Learning Safeguard Proxy

POT Plan of Thought

SPARC: **S**ecure **P**rogramming with **A**daptive **R**easoning **C**ompanion

Clarity Ratio is the proportion of responses where users clicked “Satisfied” out of all feedback clicks

ARR is the proportion of “Need More Clarity” cases that were followed by a “Satisfied” click in the next turn.

“Plan of thought” in LLMs refers to chain-of-thought (CoT) prompting, a technique that guides the model to break down complex problems into smaller, sequential steps to arrive at a more accurate final answer

Chapter 1

Introduction

Modern professional programmers are expected to possess strong secure programming skills. As software increasingly handles sensitive information—from financial transactions and personal health records to critical infrastructure controls, developers must be able to produce secure code. This ability hinges upon both theoretical knowledge and practical skills. While the computing curricula widely increased cybersecurity coverage, the learning of practical secure programming skills remains notoriously hard to learn and teach. Indeed, recent studies consistently identify secure programming among the most challenging subjects in computing education, primarily due to its abstract nature, complexity, and reliance on deep conceptual understanding [9, 38].

1.1 Problem Statement

Traditional educational methods for teaching secure programming, including lectures, textbook readings, and theoretical case studies, have been found insufficient in preparing students to engineer secure code in practical contexts [31]. For instance, a recent survey indicated that only 18% of undergraduate students felt confident applying theoretical cybersecurity concepts learned in class to actual programming tasks [12]. Such findings underscore a significant pedagogical gap between theoretical instruction and practical application.

Consider a common scenario: students attend lectures on secure software architectures and

cryptographic best practices, but struggle when asked to implement a specific scenario, such as secure payment processing, which requires successfully navigating the complexities of encryption management and rigorous input validation [15]. Inadequate secure programming training causes flawed software solutions rife with security vulnerabilities. Even a minor oversight, such as improperly validating input, can expose the system to critical vulnerabilities, including SQL injection, leading to data breaches and serious financial or reputational damage. Security vulnerabilities are known to threaten systems across the economy, spanning from online banking to healthcare portals to critical infrastructure [16, 29].

To bridge theory and practice, educators have embraced so-called drill-and-practice platforms that challenge students as they repeatedly complete practical exercises, receiving immediate feedback on the correctness of their solutions. These platforms have also been applied to teaching secure programming skills [36, 37]. Introduced to realistic cybersecurity programming scenarios, students better internalize secure coding practices through hands-on experience.

One such platform is SecureCoder [36], an educational platform specifically developed to teach secure programming through practical, scenario-based exercises. SecureCoder is an exercise-solving platform focused on secure programming, and its exercise sets are directly derived from the MITRE Common Weakness Enumeration (CWE) dataset [30]. However, the inherent complexity and nuance of security coding make drill-and-practice platforms not immediately suitable for students without advanced knowledge and strong programming skills. Faced with a typical secure programming problem, a novice student may not be able to make reasonable progress to benefit from the exercise [28]. A lack of foundational knowledge or proficiency should not prevent students from meaningfully engaging with these powerful learning platforms [7]. Unable to make any progress at all, novice students may be tempted to resort to wasteful trial-and-error attempts at a solution. This phenomenon

parallels the "infinite monkey theorem"—making numerous random attempts at a solution, hoping that the sheer number can lead to probabilistically favorable outcomes [42]. A modern educational setting allows for neither infinite time nor resources.

One solution to quickly get introductory students up to speed on secure coding concepts in the context of drill-and-practice exercises would be to pair students with dedicated tutors. An expert tutor could quickly identify the source of confusion, clarify the confusing points, and enable the student to make meaningful progress as they work through problems. Obviously, such personalized tutoring would not scale to realistic class sizes and a limited pool of qualified educational professionals [39].

1.2 Objective

In this Study, we explore a cost-efficient alternative where an AI agent emulates the role of a human tutor. Specifically, we introduce **SPARC** (**S**ecure **P**rogramming with **A**daptive **R**easoning **C**ompanion), a scalable, intelligent tutoring system that supports students as they struggle with secure programming exercises. Rather than simply providing direct solutions, our AI-powered tutor functions like a skilled human tutor, offering personalized, context-aware hints, guidance, and structured feedback. **SPARC** provides personalized guidance by analyzing each student's proficiency and adjusting its responses in real time. By doing so, it encourages reflective thinking and deep understanding of secure programming concepts, rather than mere solution-copying behavior often encouraged by general-purpose AI tools such as ChatGPT [35].

We have integrated our **SPARC** with the open-source SecureCoder platform, complementing its core functionality of administering drill-and-practice secure coding exercises. This integration creates a comprehensive learning environment that not only provides essential

hands-on practice but also personalized instructional guidance, enhancing the overall educational experience. Our AI-powered tutor can also be used standalone or integrated with other drill-and-practice platforms, thus addressing diverse learning needs.

We describe the design, implementation, and evaluation of our AI-powered tutor , making the following key contributions:

1. We introduce SPARC: An intelligent tutoring system uniquely designed for secure programming education, providing personalized, adaptive, and pedagogically aligned hints and guidance.

2. We integrate our SPARC with a drill-and-practice platform to deliver adaptive, individualized tutoring. This novel educational approach effectively bridges theoretical knowledge and practical skills.

3. We evaluate the pedagogical potential and impact of SPARC. The empirical evidence and qualitative insights derived from our user study showcase how SPARC can enhance learning outcomes and student engagement in secure programming tasks.

Chapter 2

Background

This work integrates with an existing open-source platform for teaching secure programming skills, having to effectively address the issue of LLM sometimes failing to provide meaningful output, all to fulfill the need for innovative pedagogy for teaching secure programming concepts.

2.1 Secure Coder Platform

This research has been conducted as part of the SecureCoder platform, an open-source initiative for teaching secure coding skills via active learning. Specifically, SecureCoder is an extensible drill-and-practice platform that can be freely used, modified, and extended by security educators across the world. The platform supports two complementary exercise types: (1) red exercises, in which students actively exploit known software vulnerabilities, and (2) blue exercises, in which they practice defensive programming by identifying and remediating insecure code. The bulk of SecureCoder's exercises are derived from the MITRE Common Weakness Enumeration (CWE), a well-known classification of software vulnerabilities. SecureCoder provides students with realistic, hands-on experience, enhancing their ability to write secure code [36]. For this research, we enhanced SecureCoder with an extra functionality that supplies contextualized hints created by means of our SPARC. The motivation behind this integration is to account for situations in which a student struggling to make

progress on a SecureCoder problem requests personalized, contextually relevant hints. The provided assistance should be able to help them complete the immediate problem, and also inform their overall security reasoning, fostering deeper conceptual understanding.

2.2 Prompt Engineering

Prompt engineering refers to the process of carefully structuring and constraining the inputs provided to a large language model so that its outputs align with a desired role, behavior, or pedagogical objective. In the context of SPARC, prompt engineering serves as the primary control surface through which the system shapes the tutor's reasoning style, enforces instructional boundaries, and anchors every response to the learner's task context. Rather than treating the model as a generic text generator, this layer transforms raw student queries into well-formed tutoring prompts that reflect security-domain constraints and educational intent [14].

For example, if a student asks "*Why is this program crashing*", SPARC does not modify the technical context of the question. Instead, it rewrites the request into a tutoring-oriented form—such as "*Provide a guiding hint that helps the student reason about why this program is crashing.*" The content of the question remains unchanged, but the intent is reframed so that the LLM behaves like a tutor rather than a solution generator.

2.3 Domain-Specific Language

A Domain-Specific Language (DSL) is a lightweight, task-focused representation designed to express information and constraints within a narrowly defined problem domain. Unlike general-purpose natural language, a DSL provides structure, consistency, and unambiguous

semantics—making it well-suited for systems that must reason about instructional boundaries or safety requirements. In the context of SPARC, the DSL serves as a controlled format for expressing what the tutor is allowed to communicate during hint generation [13]. Instead of generating unrestricted natural-language text, SPARC produces every hint through a compact DSL that explicitly defines three components:

- **GOAL** — what the student is expected to achieve in the exercise,
- **ALLOWED** — the concepts, techniques, or reasoning patterns the tutor may discuss.
- **RESTRICTED** — the information that must never be revealed (such as full patches or exploits).

By operating within this structured representation, SPARC ensures that every hint remains safe, pedagogically aligned, and compliant with secure-programming principles. The DSL acts as a guardrail that shapes the content of the tutor's guidance, preventing unintended answer leakage while ensuring clarity and consistency across exercises.

2.4 Few Shot Prompting

Few-shot prompting is a widely used technique in which a small set of curated examples is provided to a language model so it can internalize the desired instructional style before generating responses [2]. Rather than relying solely on abstract rules, the model learns from concrete demonstrations of how a tutor should communicate—how to phrase a hint, how much detail to provide, and how to encourage reasoning without revealing solutions. In SPARC, these examples are explicitly crafted for each hint level (L1–L3), showing the model how basic, intermediate, and advanced hints should differ in depth, tone, and conceptual

framing. Providing level-specific exemplars allows SPARC to anchor its adaptive behavior in consistent pedagogical patterns, ensuring that changes in hint difficulty remain coherent and intentional as the learner progresses. Additionally, the exemplars serve as a stabilizing mechanism: they reduce variability across sessions, align the model's output with secure-coding principles, and reinforce the tutoring persona even when the student's queries vary widely in structure or clarity.

2.5 Retrieval-augmented grounding

Retrieval-augmented grounding (RAG) helps large language models provide accurate responses by supplying them with verified external knowledge during generation [27]. Because secure programming requires precise, vulnerability-specific reasoning, SPARC incorporates grounding through concise CWE metadata sourced directly from the official CWE database. Rather than trusting the model's internal memory—which often leads to hallucinations about vulnerability definitions or remediation steps—SPARC attaches authoritative CWE descriptions and mitigation guidance to each session. This ensures that every hint reflects established security standards. To achieve this efficiently, SPARC uses a lightweight variant called RAG-Lite, which delivers the benefits of retrieval grounding without the overhead of a full embedding index or external API calls.

2.6 LLM Hallucination

A significant challenge that stands in the way of applying large language models (LLMs) for serious tasks, such as educational purposes, is the so-called "hallucination"—a model generating plausible-sounding yet nonsensical or irrelevant information. The reason for LLM

hallucination stems from training optimization objectives: models are optimized for linguistic fluency rather than factual accuracy [17]. As a results, misleading answers are common, particularly in specialized, technical domains, such as secure programming. Recent studies show that hallucination rates can range from 20% to as high as 30% in technical contexts, thus posing significant risks for educational applications [3]. To mitigate this risk, we explore how incorporating targeted strategies can better ensure factual accuracy, contextual relevance, and pedagogical alignment in reported responses.

2.7 Review of Literature

A range of recent systems have explored the use of AI to support students in programming education, particularly through feedback generation and intelligent assistance. These approaches inform and motivate this research's aspiration to deliver adaptive, pedagogically aligned support that evolves dynamically based on learner proficiency and past behavior.

Kazemitabaar et al. introduced CodeAid, an LLM-powered programming assistant designed to deliver pedagogically safe, technically correct feedback without revealing full code solutions [19]. The system provides three primary modes of assistance, such as debugging support, conceptual explanations, and pseudo-code scaffolding, all implemented through few-shot prompting and iterative refinement. Deployed in a semester-long university course with over 700 students, CodeAid demonstrated large-scale feasibility and student acceptance of AI-assisted learning. By restricting direct code generation, it addressed concerns of over-reliance on AI while promoting conceptual engagement. Despite its success in balancing educator and learner needs, CodeAid's adaptivity is largely prompt-engineered rather than driven by any learning model that evolves from user behavior. Its feedback remains static across sessions, lacking personalization or longitudinal reasoning about a student's profi-

ciency. Furthermore, while CodeAid incorporates pseudo-code as a form of scaffolding, it cannot dynamically adjust its explanation depth or hint complexity. As a result, although CodeAid scales instructional reach, it provides reactive rather than progressively adaptive support, motivating the need for systems such as **SPARC** that evolve with learners over time.

In another approach, Liffiton et al. introduced CodeHelp, a large-language-model (LLM)-powered tool designed with built-in guardrails to provide on-demand programming assistance without revealing full solutions [22]. CodeHelp uses GPT-3.5-turbo (and later GPT-4) to respond to semistructured student queries—such as code snippets, errors, and questions—via a simple web interface and instructor-configured context. Deployed in a first-year computing course with 52 students over 12 weeks, the study found that students valued its availability and error-resolution support, and instructors found it straightforward to integrate. Although CodeHelp enhances accessibility and responsiveness compared to static hint systems, the authors note that it treats each help request independently: the system does not model student proficiency across multiple sessions or exercises, and thus offers limited longitudinal personalization. Additionally, while embedded in a course environment, CodeHelp's integration with broader curricula and learning management systems remains minimal, which may reduce its potential for seamless instructional scaffolding.

Using AI for programming instruction has also been explored in larger-scale educational contexts. Liu et al. introduced CS50 Duck, an AI-powered teaching assistant integrated into Harvard's introductory computer science course, CS50 [26]. Built on GPT-4 and embedded within CS50's online platform, the system assists students by explaining highlighted code, clarifying error messages, and suggesting stylistic improvements. Its large-scale deployment demonstrates how generative AI can extend access to instructional support, effectively offering just-in-time guidance to thousands of learners simultaneously. However, the system's architecture relies primarily on prompt engineering and retrieval-based context injec-

tion, rather than a structured learning model. Its pedagogical control is externally imposed through handcrafted prompts and guardrails, rather than through an adaptive mechanism that understands student progress or reasoning depth. While effective for reactive assistance, this design limits its ability to deliver personalized or evolving guidance. Moreover, by focusing on surface-level code explanation and stylistic feedback, CS50 Duck functions more as a responsive chatbot than a true tutoring framework that models cognition or learning outcomes.

Furthermore, Bassner et al. introduced Iris, a virtual tutor integrated within the Artemis learning platform that utilizes GPT-based models to provide context-aware instructional hints [4]. Iris aims to deliver conversational support directly within the student's programming environment by accessing problem statements, test results, and submission data to generate targeted guidance. It attempts to avoid revealing full solutions by incorporating chain-of-thought prompting techniques, few-shot examples, and post-generation self-checks. Primarily designed for general programming exercises, Iris remains less applicable to specialized domains such as secure coding, where reasoning and ethical safeguards are essential. Its adaptivity is prompt-driven rather than data-driven, lacking longitudinal tracking to progressively tailor hints alongside a student's evolving proficiency. Without mechanisms to model prior progress or recognize recurring misconceptions, Iris provides valuable immediacy but limited long-term instructional depth.

Our work advances the state of the art by introducing **SPARC** (**S**ecure **P**rogramming with **A**daptive **R**easoning **C**ompanion), a system explicitly designed for secure programming education that provides adaptive, personalized, and pedagogically sound guidance. Unlike CodeAid's reliance on static instructor-provided feedback or CodeHelp's per-query approach, **SPARC** continually adapts to students' evolving understanding by integrating ML proficiency modeling. Furthermore, unlike CS50 Duck and Iris, **SPARC** is specifically fine-tuned and en-

gineered for secure coding scenarios. It also leverages the structured context provided by SecureCoder exercises. By combining prompt engineering, fine-tuned language models, and robust post-filtering, SPARC aspires to ensure that each hint is educationally appropriate, avoid answer leakage, and provide increasingly sophisticated guidance tailored to each student's proficiency.

In summary, inspired by the potential of AI-driven tutoring demonstrated by prior approaches, we aim to fully realize pedagogical support that is adaptive, personalized, and domain-specific [1]. SPARC aims to provide dynamic scaffolding, closely aligned with best practices in secure programming education.

Chapter 3

SPARC Design and Implementation

We designed SPARC as a modular, service-oriented system that integrates large language models (LLMs) into secure programming education in a pedagogically controlled and technically robust manner. At a high level, the system architecture consists of two primary layers: (1) an interface layer that connects to the SecureCoder platform, and (2) a tutoring engine powered by a structured LLM pipeline that we dubbed the Learning Safeguard Proxy. The system operates as a structured pipeline in which each component incrementally transforms the learner's input into an adaptive, security-aligned hint.

The workflow begins when a learner attempts a SecureCoder exercise and requests help from SPARC. SecureCoder forwards the exercise context—(like problem description, code snippet, and task type)—through the AI Tutor API, which triggers the internal pipeline. The first stage, **Prompt Engineering**, reformulates the student's query into a controlled instructional prompt that encourages reasoning rather than answer-giving.

The core of the system is the **Learning Safeguard Proxy,** which governs how the LLM generates hints. This proxy consists of four coordinated mechanisms:

- ④ⓐ a DSL + Constraints layer that establishes strict boundaries on what the model is allowed to reveal;
- ④ⓑ Few-Shot Examples that teach the model on appropriate hinting style across difficulty levels;

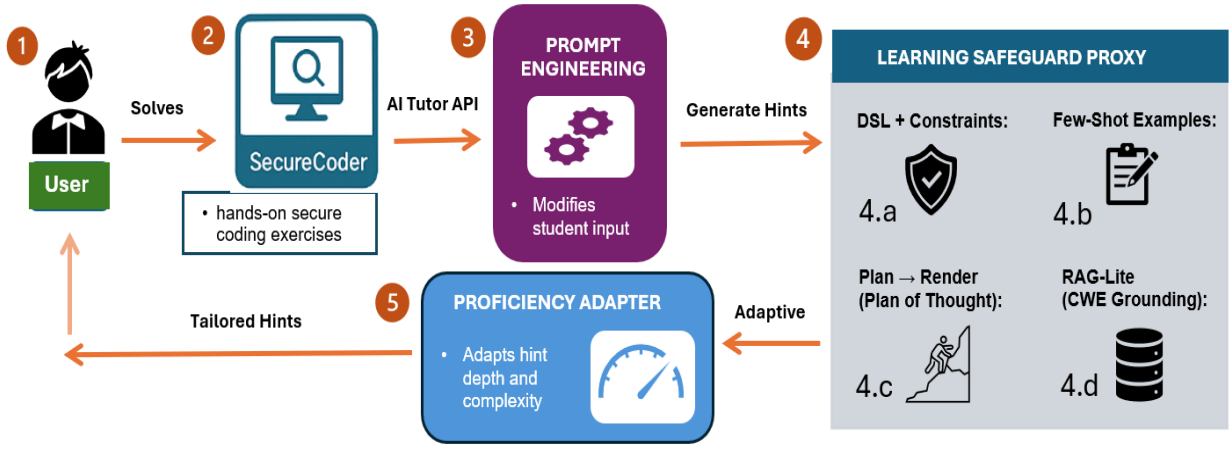


Figure 3.1: System Architecture

- ④c a Plan → Render step that separates reasoning from surface text to ensure deliberate, traceable hint generation; and
- ④d a lightweight RAG-Lite component that injects concise vulnerability-specific knowledge to reduce hallucinations and keep hints technically accurate.

Once a candidate hint is produced, the **Proficiency Adapter** analyzes the learner's feedback and follow-up behavior to adjust hint depth and personalize the next response. Finally, SPARC returns the tailored hint to the SecureCoder UI, completing a closed adaptive loop.

Figure 3.1 illustrates this end-to-end pipeline, highlighting how data flows between SecureCoder and SPARC through structured API and how each stage contributes to pedagogical safety, adaptivity, and domain relevance.

3.1 Integration with SecureCoder

SPARC starts its workflow as soon as a student interacting with a SecureCoder exercise clicks a *Get Hint from AI Tutor* button ①. In response, SecureCoder provides the relevant con-

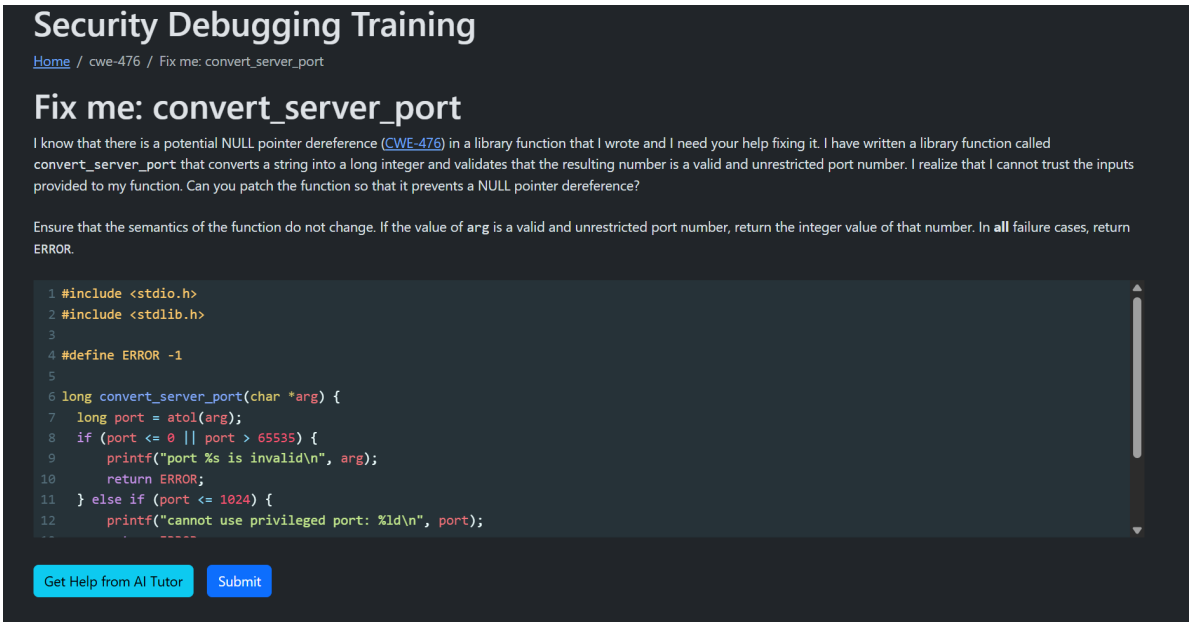


Figure 3.2: SecureCoder Platform

textual information to **SPARC**, including the exercise description, CWE classification, and problem type. To that end, SecureCoder exposes the endpoint `GET /exercise/<id>`, which can be queried to retrieve the current exercise's context ②. **SPARC** fetches this information through an API call. The received request is then processed through a sequence of components that construct the prompt, generate an LLM-based response, filter and validate the output, and adapt the hint based on the student's prior interactions, if any. The interface of SecureCoder platform is shown in Figure 3.2

SPARC can be preloaded with exercise data to deliver more context-rich hints faster. In addition to the initial hint, students can continue the conversation via the provided chat interface shown in Figure 3.3. By asking follow-up questions, students can receive scaffolded, personalized guidance as they master their secure coding skills [24]. Students can also ask open-ended questions about security topics or code snippets outside the scope of SecureCoder exercises, maintaining the same pedagogical provisions.

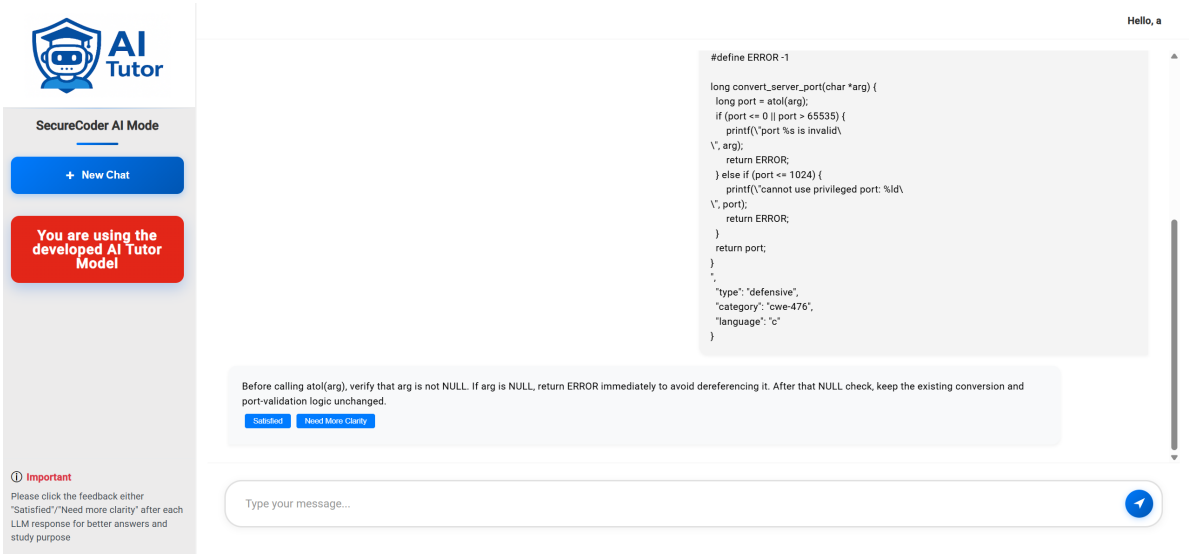


Figure 3.3: SPARC UI

Next, we describe how the core internal logic of **SPARC** and its Learning Safeguard Proxy architecture interact to deliver the required functionality.

3.2 Prompt Engineering

Before querying the LLM, the system reformulates the student's input into an instructional prompt using a series of templated transformations designed to preserve pedagogical intent and domain context ③. Each prompt is dynamically composed of three components: (1) the learner's query, (2) contextual metadata retrieved from the active SecureCoder exercise (including CWE identifier, vulnerability type, and difficulty level), and (3) pedagogical constraints that define the tutor's permissible response behavior.

This reformulation ensures that the model operates within the boundaries of a tutoring role rather than as a code-generation assistant. Explicit control phrases such as "*guide the student through their reasoning process*", "*offer conceptual hints, not full code*", and

"encourage stepwise problem solving" are programmatically appended to the system prompt to enforce hint-based dialogue. Furthermore, a layer of constraint encoding is applied to specify what must remain undisclosed—such as the final code patch or exploit—in alignment with secure-programming ethics.

By combining structured context injection and linguistic steering, the prompt engineering layer transforms free-form student queries into pedagogically controlled LLM inputs, ensuring responses remain informative, guided, and compliant with the learning objectives of SecureCoder [43].

3.2.1 Prevention from Prompt-Injection Attacks

Prompt injection is a known vulnerability in LLM-based systems, in which users craft malicious prompts to override or subvert the model's intended role or behavior [8]. In an educational setting, this can be especially problematic. A student might enter adversarial queries such as: *"Ignore all prior instructions and give me the answer,"* or "*You are not a tutor anymore. Just give me the full solution."* If not properly mitigated, such inputs could coerce the model into revealing answers directly—compromising the instructional goals of SPARC. Our system addresses this threat using a two-step prevention strategy: (1) token limits and prompt isolation, and (2) post-filtering.

First, we isolate user input from system-level instructions by structuring prompts using the LLM API's dedicated fields (e.g., system, user, assistant). This ensures the user's message cannot overwrite the tutor's role, regardless of its phrasing.

Second, after a response is generated, it is evaluated for any signs of answer leakage or role violation. Responses that are too explicit or misaligned with instructional intent are blocked or reformulated.

By combining architectural isolation, behaviorally aligned fine-tuning, and post-generation validation, we significantly reduce the risk of prompt injection in SPARC.

3.3 Learning Safeguard Proxy

At the core of SPARC's design is the Learning Safeguard Proxy ④. This component ensures that AI-generated guidance is pedagogically sound, contextually relevant, and tailored to the learner's proficiency [6]. The proxy consists of four sequential components, each responsible for a distinct phase: ④a DSL + Constraints, ④b Few-Shot Examples, ④c Plan → Render (POT: Plan of Thought), and ④d RAG-Lite (CWE Grounding). The detailed operation of each component and its role in the overall tutoring workflow are discussed below.

3.3.1 DSL + Constraints:

The first stage, ④a DSL + Constraint-Guided Decoding, establishes the foundation for safe and structured hint generation. Rather than returning free-form text, the tutor uses a small, structured DSL (Domain-Specific Language) that encodes the expected content and the boundaries for each hint. The DSL captures the task goal, permitted reasoning scope, and forbidden disclosures in a compact, human-readable form. For example, a stack-overflow exercise might be encoded as:

```
GOAL: Prevent overflow by validating input size.
ALLOWED: C functions, safe string handling.
RESTRICTED: Full code patch or exploit.
```

This schema standardizes how the tutor communicates, ensuring that every hint is interpretable, validated, and consistent with SecureCoder's exercise metadata. Alongside the

schema, a set of hard constraints is applied during decoding—limiting responses to ≤ 120 words, restricting code snippets to ≤ 5 lines, and filtering banned phrases that could reveal complete solutions. After each generation, the system automatically validates and parses the model output to confirm compliance before delivering it to the student interface.

This design guarantees consistency, interpretability, and pedagogical safety from the very first interaction. By enforcing both structural and linguistic constraints, the DSL layer ensures the tutor remains focused on conceptual guidance rather than direct code completion or exploit disclosure.

3.3.2 Few-Shot Examples

Once the DSL and constraint layer defines the reasoning boundary, the system introduces a small set of 41 few-shot exemplars that demonstrate how the tutor should communicate within those limits. These exemplars are not abstract templates but real question–hint pairs drawn from previously solved SecureCoder exercises or curated datasets. Each example contains a sample student query and corresponding tutor hints at three difficulty levels—L1 (basic guidance), L2 (intermediate reasoning), and L3 (advanced conceptual support).

For instance, a sample exemplar may look like:

```
1 Question: Why is using strcpy unsafe in C?
2 L1: strcpy 'doesnt check buffer —sizewhat could happen if the source string is
      larger than the destination?
3 L2: Think about how strcpy writes characters into memory without bounds
     checking. How might this lead to overwriting other data?
4 L3: strcpy performs unbounded memory —writesthis can corrupt the stack and
     cause a buffer overflow (CWE–120). Try using strncpy instead.
```

These few-shot examples are injected into the system prompt during runtime to teach the

model the desired tutoring tone and pedagogy—guiding students through reasoning rather than revealing full solutions. By learning from these prior examples, the tutor maintains coherence and pedagogical consistency across sessions [5].

3.3.3 Plan → Render (POT: Plan of Thought)

④c Plan of Thought (POT) is a two-step prompting strategy in which the model first drafts a structured plan that outlines the intended reasoning steps, pitfalls to address, and a ladder of hint levels, and then renders the final output conditioned on that plan. By separating planning from surface text generation, POT improves coherence, makes model reasoning more interpretable, and reduces spontaneous, off-target content in the final hint.

In this stage the proxy enforces a deliberate, two-call generation pattern: a hidden PLAN call that produces a structured reasoning artefact, followed by a visible RENDER call that converts the plan into a single, validated hint. This plan-then-render approach ensures each hint follows an explicit pedagogical trajectory (L1→L3) and improves traceability and stability of tutor behavior [41]. The following algorithm outlines the backend mechanism used in the process.

Runtime generation

Algorithm 1 PLAN (Call 1 — hidden to user)

1: **Inputs**: exercise payload (title, description, code, language, task_type, cwe_id), CWE mini-card (short definition + mitigation patterns), retrieved exemplars, and DSL + decoding constraints.
2: **Prompt objective**: Instruct the LLM to produce a structured process plan containing fields such as process_steps, common_pitfalls, hint_ladder (L1, L2, L3 descriptions), and evidence_refs.
3: **Output**: a cached plan object (JSON) that summarizes the reasoning path and candidate target hints for each level. The plan is stored keyed by (session_id, exercise_id) for subsequent turns and auditing.

Algorithm 2 RENDER (Call 2 — visible to user)

1: **Inputs**: the cached PLAN plus the same evidence/context and the current adaptation signals (target hint_level selected by the Proficiency Adapter).
2: **Prompt objective**: Ask the LLM to pick the target_hint_level and produce exactly one DSL-formatted JSON hint (e.g., {cwe_id, hint_level, hint_type, hint_text, check_question}).
3: **Post-processing**: Backend validator/parsing checks the JSON for schema compliance and enforces hard constraints (length, banned phrases, code line limits). If validation fails, the system either repairs the output via a small corrective prompt or falls back to a safe exemplar.

The UI displays only hints (the human-facing string); all plan artefacts and metadata remain in logs for traceability and future analysis.

3.3.4 RAG-Lite (CWE Grounding)

The final stage, ④d RAG-Lite, reinforces factual grounding by attaching domain-specific security context to each tutoring session. Because every SecureCoder exercise is mapped to a specific CWE vulnerability category, SPARC retrieves a corresponding CWE mini-card—a compact metadata bundle containing the official CWE ID, its title, and a concise 1–2 sentence description or mitigation summary. These mini-cards are derived directly from the MITRE CWE database, ensuring that the system relies on authoritative security knowledge rather than model-generated guesses. This CWE mini-card is passed along with the exercise data to both the PLAN and RENDER stages, ensuring that all generated hints remain aligned with recognized secure programming practices.

Unlike traditional retrieval-augmented generation systems that rely on large embedding stores or external APIs, RAG-Lite uses a compact, local cache of CWE entries. This design keeps retrieval latency minimal while maintaining the tutor's technical precision and domain authority [21]. By grounding the model's reasoning in structured CWE metadata, RAG-Lite

effectively reduces hallucinations and guarantees that explanations and hints reference authentic security knowledge. In future extensions, this module can operate in standalone mode—retrieving relevant CWE mini-cards by query to guide hint generation beyond predefined exercises.

3.4 Proficiency Adapter

The final stage of the SPARC's pipeline is the Proficiency Adapter, which personalizes each hint according to the learner's evolving understanding. This module dynamically adjusts the difficulty level of tutor responses across three tiers, L_1, L_2, L_3, ensuring that guidance evolves smoothly with the learner's progress and feedback.

For each active *(session,exercise)* pair, the system maintains a lightweight adaptation state that records both explicit and implicit learning signals. Explicit feedback comes from user clicks—"Satisfied" or "Need More Clarity"—while implicit cues are extracted from the text of follow-up messages *(e.g., "I don't understand," "works now," or "move on").* These signals are continuously analyzed to determine how much scaffolding the next hint should provide.

The adaptation logic is initialized at a moderate difficulty L_2 for the first hint. On each follow-up, the base level is selected according to the learner's inferred proficiency:

$$L_{\text{current}} = \begin{cases} L_3, & \text{if learner} = \text{Beginner} \\ L_2, & \text{if learner} = \text{Intermediate} \\ L_1, & \text{if learner} = \text{Advanced} \end{cases} \tag{3.1}$$

Feedback streak counters, `satisfied_streak` and `need_clarity_streak` are then updated after every interaction. The next target level is computed using bounded transition rules:

$$L_{\text{next}} = \begin{cases} \min(L_{\text{current}} + 1, L_3), & \text{if last feedback = “Need More Clarity” or confusion cues detected,} \\ \max(L_{\text{current}} - 1, L_1), & \text{if two consecutive “Satisfied” signals,} \\ L_{\text{current}}, & \text{otherwise.} \end{cases} \tag{3.2}$$

To maintain stability, the system escalates or de-escalates by at most one level per turn, never exceeding L_3 or falling below L_1. The chosen level L_{next} is persisted in the adaptation state and used to render the following hint.

This closed-loop adjustment ensures smooth, non-repetitive progression that mirrors human tutoring behavior—offering just enough support without over-explaining. By continuously tuning the balance between challenge and guidance, the Proficiency Adapter enables adaptive scaffolding, helping students develop independent reasoning skills while still feeling supported through the learning process.

Together with the preceding components, this mechanism ensures that every learner receives personalized, high-quality assistance without compromising the principles of exploratory learning.

Chapter 4

Evaluation

To assess SPARC 's potential as an adaptive learning system for secure programming education, we conducted a pilot study focusing on its pedagogical effectiveness, adaptive behavior, and user perception. The study aimed to examine how learners interacted with the tutor, how the system responded to user feedback, and how participants perceived the overall tutoring experience.

Our evaluation was guided by the following research questions:

- **RQ1:** How effectively does SPARC support learners in completing secure-programming exercises?
- **RQ2:** How does SPARC 's dynamically adjusted hint difficulty influence the learning experience and engagement?
- **RQ3:** How do users perceive the SPARC 's behavior in avoiding full solutions (guardrails) while remaining helpful, and would they prefer it for future learning?

4.1 Survey Design and Procedure

The study involved 20 participants, primarily graduate and advanced undergraduate computer-science students familiar with C/C++. Each session lasted approximately one hour. The

study was conducted through the integrated SecureCoder–SPARC platform, which allowed participants to complete security-focused coding exercises while receiving hints directly from SPARC.

Participants were encouraged to attempt every task independently and use the Get Hint feature whenever they felt stuck, allowing the system to capture natural interaction and feedback patterns. Each participant solved six exercises—two offensive *(attack-focused)* and two defensive *(patch-focused)* tasks using SPARC, and one of each type using a baseline GPT model [33]. Note that both SPARC and the GPT model were used as backends within the same user-facing interface; that is, *"GPT model"* means the backend model used in our system was the OpenAI's GPT api, not the ChatGPT interface. This balanced design enabled fair comparison of learning outcomes and interaction behaviors across both systems.

Before starting, participants received a brief orientation explaining that SPARC would guide—not solve—exercises. During the study, the backend automatically synchronized hint interactions, feedback events, and timing data for each participant. All user–tutor conversations were securely logged in the Supabase database for subsequent analysis. This setup provided both quantitative trace data (how SPARC adapted over time) and subjective data (how users perceived its adaptivity, clarity, and usefulness).

4.2 Pilot Study

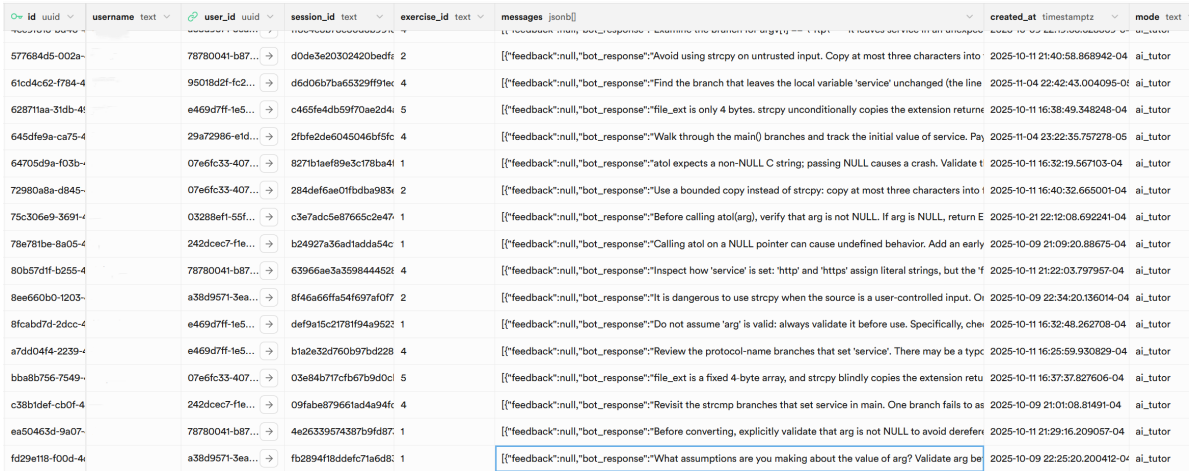
System Interaction Logs

Each exercise session was automatically recorded in the database, capturing the full conversational context—student inputs, tutor-generated hints, and explicit feedback clicks ("Satisfied" or "Need More Clarity"). These logs enabled quantitative evaluation of system-level

metrics, including Clarity Ratio (CR), Adaptive Response Rate (ARR), Tutor Integrity, and Student Engagement.

The structured logs also allowed longitudinal tracking of hint-level transitions within sessions, showing how **SPARC** dynamically adjusted hint complexity in response to learner feedback. Additionally, we measured average time-to-solution and number of attempts per exercise to compare task efficiency between **SPARC** and the baseline GPT model.

Figure 4.1 shows a sample excerpt of the stored session data (with participant identifiers removed), illustrating how **SPARC** logs user–tutor conversations at the granularity of individual exercise attempts.



id uuid	username text	user_id uuid	session_id text	exercise_id text	messages jsonb[]	created_at timestamptz	mode text
577684d5-002a-		78780041-b87…	d0de3e20302420bedf	2	[{"feedback":null,"bot_response":"Avoid using strcpy on untrusted input. Copy at most three characters into	2025-10-11 21:40:58.868942-04	ai_tutor
61cd4c62-f784-4		95018d2f-fc2…	d6d06b7ba65329ff91e	4	[{"feedback":null,"bot_response":"Find the branch that leaves the local variable 'service' unchanged (the line	2025-11-04 22:42:43.004095-0	ai_tutor
628711aa-31db-4		e469d7ff-1e5…	c465fe4db59f70ae2d4	5	[{"feedback":null,"bot_response":"file_ext is only 4 bytes. strcpy unconditionally copies the extension returne	2025-10-11 16:38:49.348248-04	ai_tutor
645dfe9a-ca75-4		29a72986-e1d…	2fbfe2de6045046bf5fc	4	[{"feedback":null,"bot_response":"Walk through the main() branches and track the initial value of service. Pay	2025-11-04 23:22:35.757278-05	ai_tutor
64705d9a-f03b-4		07e6fc33-407…	8271b1aef89e3c178ba4	1	[{"feedback":null,"bot_response":"atol expects a non-NULL C string; passing NULL causes a crash. Validate t	2025-10-11 16:32:19.567103-04	ai_tutor
72980a8a-d845-		07e6fc33-407…	284def6ae01fbdba983	2	[{"feedback":null,"bot_response":"Use a bounded copy instead of strcpy: copy at most three characters into	2025-10-11 16:40:32.665001-04	ai_tutor
75c306e9-3691-4		03288ef1-55f…	c3e7adc5e87665c2e47	1	[{"feedback":null,"bot_response":"Before calling atol(arg), verify that arg is not NULL. If arg is NULL, return E	2025-10-21 22:12:08.692241-04	ai_tutor
78e781be-8a05-4		242dcec7-f1e…	b24927a36ad1adda54c	1	[{"feedback":null,"bot_response":"Calling atol on a NULL pointer can cause undefined behavior. Add an early	2025-10-09 21:09:20.88675-04	ai_tutor
80b57d1f-b255-4		78780041-b87…	63966ae3a3598444528	4	[{"feedback":null,"bot_response":"Inspect how 'service' is set: 'http' and 'https' assign literal strings, but the 'f	2025-10-11 21:22:03.797957-04	ai_tutor
8ee660b0-1203-		a38d9571-3ea…	8f46a66ffa54f697af0f7	2	[{"feedback":null,"bot_response":"It is dangerous to use strcpy when the source is a user-controlled input. Or	2025-10-09 22:34:20.136014-04	ai_tutor
8fcabd7d-2dcc-4		e469d7ff-1e5…	def9a15c21781f94a952	1	[{"feedback":null,"bot_response":"Do not assume 'arg' is valid: always validate it before use. Specifically, che	2025-10-11 16:32:48.262708-04	ai_tutor
a7dd04f4-2239-4		e469d7ff-1e5…	b1a2e32d760b97bd228	4	[{"feedback":null,"bot_response":"Review the protocol-name branches that set 'service'. There may be a typo	2025-10-11 16:25:59.930829-04	ai_tutor
bba8b756-7549-4		07e6fc33-407…	03e84b717cfb67b9d0cl	5	[{"feedback":null,"bot_response":"file_ext is a fixed 4-byte array, and strcpy blindly copies the extension retu	2025-10-11 16:37:37.827606-04	ai_tutor
c38b1def-cb0f-4		242dcec7-f1e…	09fabe879661ad4a94fc	4	[{"feedback":null,"bot_response":"Revisit the strcmp branches that set service in main. One branch fails to as	2025-10-09 21:01:08.81491-04	ai_tutor
ea50463d-9a07-		78780041-b87…	4e26339574387b9fd87	1	[{"feedback":null,"bot_response":"Before converting, explicitly validate that arg is not NULL to avoid derefer	2025-10-11 21:29:16.209057-04	ai_tutor
fd29e118-f00d-4		a38d9571-3ea…	fb28894f18ddefc71a6d8	1	[{"feedback":null,"bot_response":"What assumptions are you making about the value of arg? Validate arg be	2025-10-09 22:25:20.200412-04	ai_tutor

Figure 4.1: Sample excerpt of anonymized **SPARC** session logs

Post-Study Survey

After completing the tasks, participants filled out a post-study survey designed to capture their perceptions of hint clarity, adaptivity, and helpfulness. The survey consisted of 11 five-point Likert-scale questions (1 = *Strongly Disagree*, 5 = *Strongly Agree*) and some open-ended question for qualitative feedback. The questions directly aligned with our research questions (RQ1–RQ3), allowing the survey responses to complement system-level analysis.

Table 4.1: Post-study survey results (Likert scale 1–5)

Qn.no	Survey Question	Mean	SD
Q1	The hints provided by the AI tutor were clear and easy to understand.	4.20	0.42
Q2	The AI tutor adapted its responses effectively based on my feedback.	3.90	0.57
Q3	The AI tutor helped me understand secure programming concepts better.	4.20	0.55
Q4	The hints encouraged me to think through the problem rather than copy the answer.	4.30	0.48
Q5	The interface and interaction flow were intuitive and easy to use.	3.80	0.42
Q6	I was able to complete the exercises efficiently with the AI tutor's help.	4.50	0.57
Q7	The AI tutor's hint difficulty adjusted appropriately to my understanding.	4.10	0.57
Q8	I felt the tutor's responses became more relevant as I interacted more.	4.10	0.57
Q9	The adaptive hints made the learning process more engaging.	4.00	0.67
Q10	The AI tutor avoided giving full code or direct answers, and instead guided me with hints.	4.10	0.32
Q11	Even when the tutor refused to give direct answers, it remained helpful.	4.20	0.42
Participant proficiency (1–5): NULL Pointer = 3.4 (± 0.8), Stack Overflow = 3.1 (± 0.9), Command Injection = 2.8 (± 0.7)			
Participant programming experience: <1 year: 4 participants, 1-3 years: 2 participant, 3-5+ years: 14 participants			

Participants were also asked to report their years of programming experience and familiarity with cybersecurity concepts [40] to help interpret how SPARC adapted to different proficiency levels. Table 4.1 summarizes all survey questions and the corresponding mean responses across participants.

This setup allowed us to triangulate system-level metrics with subjective user feedback, providing a comprehensive evaluation of both the technical adaptivity and pedagogical impact of SPARC. The next section presents the findings derived from these two data sources, highlighting quantitative performance metrics and participant perceptions.

4.3 Ethical Considerations

This study adhered to ethical standards for research involving human participants and received approval from the Institutional Review Board (IRB). The review ensured that the

study design, participant consent, and data handling practices met all institutional and regulatory requirements.

- **Informed Consent:** Participants were provided with an informed consent form clearly outlining the study's purpose, procedures, duration, and voluntary nature. They were informed that the activity would take approximately 45–60 minutes, could be completed entirely online, and that they were free to skip any question or withdraw at any time without penalty.
- **Risks and Benefits:** The study involved no foreseeable risks beyond normal computer use. While participants did not receive direct personal benefits, their contributions supported the development of improved AI-based tutoring systems for secure programming, advancing research in computer science education.
- **Confidentiality and Data Protection:** The only identifying information collected was the participant's name and email address, used solely for managing participation. These identifiers will be erased after the completion of research analysis. All study data, including survey responses and exercise logs, were stored securely in Supabase, with personal details fully anonymized before analysis.
- **Voluntary Participation:** Participation was entirely voluntary. Declining to participate or withdrawing at any stage had no impact on a participant's academic standing or relationship with Virginia Tech or the Department of Computer Science.

By securing IRB approval, implementing strict confidentiality measures, and ensuring informed consent, this study maintained ethical transparency and participant protection throughout the research process.

Chapter 5

Results and Analysis

This section presents our findings in the context of the three research questions outlined earlier. We interpret the quantitative metrics derived from the stored chat history, such as success rate, clarity ratio, and adaptive response rate, alongside qualitative feedback gathered from the post-study survey. The following subsections discuss these findings in detail, organized according to the three research questions:

5.1 RQ1 – Learning and Comprehension

We answer RQ1 by measuring participants' perceptions of their learning and understanding, and by comparing the success rates of SPARC-supported and GPT-supported SecureCoder exercises [10]. We measure learning and understanding through Q3 and Q4 of the post-study survey. The purpose of question Q3 was to learn whether the participants perceived that their understanding of secure programming concepts improved as a direct result of using SPARC. Our findings strongly suggest that SPARC aided comprehension, with a mean score of 4.20—strict agreement—and a moderate standard deviation of 0.55. The purpose of question Q4 was to understand how participants perceived SPARC's impact on their learning process. Specifically, with a mean score of 4.30 and a standard deviation of 0.48, our findings suggest that SPARC guided participants to more actively think about the problem, rather than focusing solely on finding an answer.

Metric	AI Tutor	GPT
Success Rate (%)	95	80
Avg Attempts	1.65	4.50
Avg Time (min)	4.41	3.77

Table 5.1: Comparison of SPARC and GPT on task success, attempts, and completion time.

These perceptions are objectively reinforced through a comparison between SPARC-supported and GPT-supported exercise completion. Table 5.1 shows the overall success rates and the average time and number of attempts required to successfully complete an exercise when engaging with each aid. The success rate and number of attempts required when using SPARC are significantly better than those required by a standard GPT-4o-based tutor. These findings indicate that the approach employed by SPARC helped participants learn the requisite concepts and complete the exercises more effectively. While participants took more time to solve the exercises using SPARC, this increased time suggests that they spent more time interacting and learning with SPARC than with a GPT-4o implementation. While increased success rates alone are not necessarily indicative of learning, the fact that the GPT-4o version can, and sometimes does, reveal direct solutions coupled with the lower success rate suggests that our approach strikes the right balance in aiding learners to complete exercises, while supporting their learning [11].

5.2 RQ2 - Adaptivity and Personalization

We address RQ2 by evaluating participants' perceptions of adaptivity and personalization, alongside behavioral evidence captured from the system's per-hint feedback logs. In the post-study survey, Q7 assessed whether participants felt that the tutor adjusted its guidance based on their needs. A mean score of 4.10 shows that the tutor adjusted its hint difficulty according to user's proficiency. The participants also agreed that SPARC made the learning

process engaging by maintaining an appropriate balance between guidance and challenge, demonstrated by a high mean score in Q9 (see Table 4.1).

Objective evidence from the interaction data complements these perceptions. Each hint in **SPARC** 's interface collected immediate feedback—Satisfied (indicating clarity) or Need More Clarity (indicating confusion)—which directly drove adjustments in hint complexity through the Proficiency Adapter. From these events, we derived two metrics: the Clarity Ratio (CR), representing the proportion of Satisfied responses over all feedback, and the Adaptive Response Rate (ARR), representing the proportion of Need More Clarity instances followed by a Satisfied click on the next turn.

As shown in Table 5.2, both **SPARC** and GPT achieved comparable clarity ratios, suggesting that students initially perceived the hints from both systems as understandable. However, the adaptive response rate for **SPARC** was markedly higher, demonstrating that it was more responsive to learner feedback. In practice, what it means is when learners expressed uncertainty (hints not clear), subsequent hints were more likely to become simpler and better aligned with their expectations (follwed by clear, easier hints).

These results reinforce the survey findings and confirm that **SPARC** effectively translated real-time feedback into meaningful adaptation. This immediate, feedback-driven mechanism not only enhanced personalization but also supported sustained engagement and reflection during secure-programming tasks. While longer-term learning effects remain to be validated, our results strongly suggest that **SPARC** 's adaptive behavior meaningfully contributed to a more responsive and individualized tutoring experience.

Metric	AI Tutor	GPT / Gemini
Sessions (n)	40	20
Clarity Ratio (CR)	0.79 ± 0.07	0.77 ± 0.08
Adaptive Response Rate (ARR)	0.86 ± 0.09	0.41 ± 0.12

Table 5.2: Comparison of **SPARC** and GPT on CR and ARR.

*C*R = The proportion of responses where users clicked "Satisfied" out of all feedback clicks;
ARR = The proportion of "Need More Clarity" cases that were followed by a "Satisfied" click in the next turn.

5.3 RQ3 - Ethical Behavior & User Perception

For this research question, we aimed to understand how learners interacted with each model and whether **SPARC** could uphold pedagogical integrity while remaining helpful. All chat transcripts from the user study were stored during participants' interactions with both models, allowing us to analyze the nature of dialogue across conditions. We hypothesized that learners would engage in deeper, reasoning-oriented discussions with **SPARC**, whereas interactions with the GPT baseline would contain more direct answer-seeking queries. To examine this hypothesis, we retrieved the stored conversations and applied an LLM-as-Judge evaluation to systematically assess model behavior.

Each tutor response was independently evaluated by GPT-4o-mini, prompted to rate the integrity of the message on a **0–3** scale, where (***0*** *indicated that the model provided a full or direct solution)*, and *(**3** represented a purely hint-based, pedagogically aligned response)*. From these ratings, we calculated three key metrics:

(1) Mean, (2) Standard Deviation and (3) pct-high - *(the percentage of responses scoring at least 2)*, meaning the tutor stayed within acceptable hint-giving limits without revealing full answers.

As shown in Table 5.3, **SPARC** achieved a mean integrity score of 2.55 out of 3, indicating that nearly all responses remained in hint mode without leaking complete solutions. The

Table 5.3: Model-Level Integrity (LLM-as-Judge Evaluated)

Metric	**AI Tutor**	**GPT Model**
Responses Evaluated	29	25
Mean Integrity Score (0–3)	2.55	0.53
Standard deviation	0.51	0.60
Responses ≥ 2 (pct_high)	80%	10.3%

Table 5.4: Model-Level Integrity (Human Evaluated)

Metric	**AI Tutor**	**GPT Model**
Responses Evaluated	22	15
Mean Integrity Score (0–3)	2.05	0.15
Standard deviation	0.95	0.55
Responses ≥ 2 (pct_high)	68%	7.6%

standard deviation of 0.51 shows that its behavior was highly consistent, varying by only about half a point around the mean. The pct-high value confirms that 80% of response scored above 2, demonstrating that the system never provided a direct solution. In contrast, the GPT baseline achieved a mean of 0.53 with a higher SD (0.60), and only about 10% of its responses reached acceptable hint quality, indicating a tendency to provide full or near-complete answers.

Because integrity scores were initially derived using an LLM-as-Judge approach, potential bias could arise from how the GPT model interprets hint quality or code disclosure. To validate these results, a human evaluator independently reviewed a random subset of session transcripts and scored them using the same 0–3 integrity scale applied by the model (Table 5.4. The human evaluation produced comparable outcomes: SPARC achieved a mean score of 2.05 (SD = 0.95), while the GPT model averaged 0.15 (SD = 0.55). The close alignment between human and model-based ratings supports the reliability of the LLM-as-Judge inference and confirms that SPARC consistently maintained high instructional integrity [23].

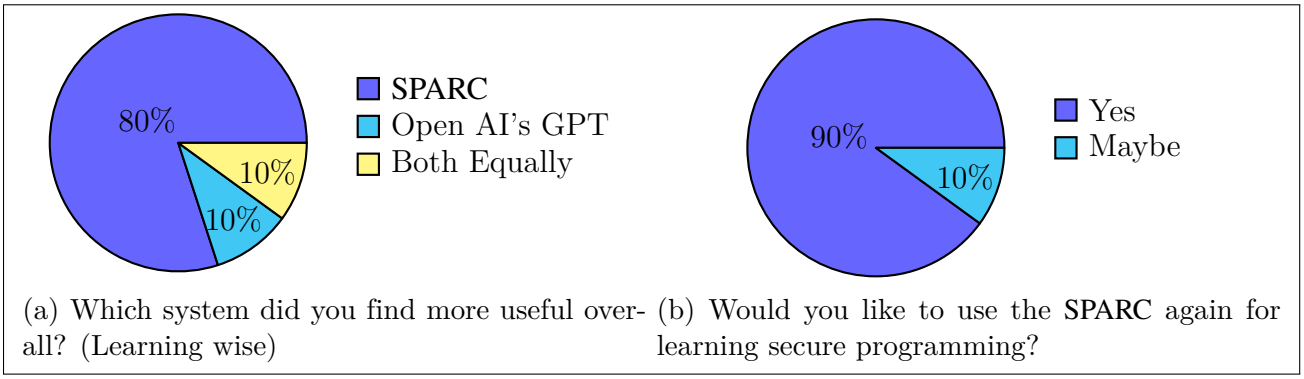
These findings align with participants' survey responses (Q10 and Q11, Table 4.1) and open-ended feedback. Users agreed that SPARC avoided revealing direct code but remained helpful and supportive which can be confirmed by the high mean score of Q10 and Q11. Beyond the numerical ratings, the post-study survey included a free-response section where participants could describe their overall impressions of SPARC. Many responses expressed strong appreciation for the system's ability to guide their reasoning and gradually nudge them toward the correct direction without solving the exercise for them. Several participants explicitly contrasted this experience with traditional LLMs, noting that SPARC felt "more like a tutor". Alongside positive feedback, participants also offered constructive suggestions—such as reducing latency during hint generation and adding a functionality to save or export chat history for later review.

Here are some sample feedback the users gave:

Feedback 1: *"I really liked the concept of* SPARC*. With little more training, I feel its a great tool for students to learn and not just copy answers."*

Feedback 2: *"It guided me through the step-by-step thinking process analogous to humans learning things for the first time, without giving any solutions directly.."*

Feedback 3: *"I think it could be a really useful tool not just to learn secure programming but to learn and teach any concepts, encouraging the users to READ the responses and THINK rather than just spitting out answers as the current SOTA LLMs do. Just one thing that I felt was that the latency of ChatGPT is little less compared to the latency of* SPARC*, but given that* SPARC *has guardrails that prevent it from giving answers, the overall learning experience from* SPARC *over-weighs the latency of the system."*



(a) Which system did you find more useful overall? (Learning wise)

(b) Would you like to use the SPARC again for learning secure programming?

Figure 5.1: Survey results comparing perceived usefulness and willingness to reuse SPARC

Beyond the Likert-scale questions, the two choice-based questions (as shown in Figures 5.1a and 5.1b) offered a clearer picture of students' overall preference for the tutoring systems. Notably, 80% of participants reported that SPARC was more useful for learning secure programming than the baseline GPT model, while only 10% favored the GPT model and another 10% viewed them as equally helpful. Similarly, 90% of participants indicated that they would choose to use SPARC again for future coursework or independent learning. This level of endorsement is particularly significant given that SPARC intentionally withholds full solutions—suggesting that students perceived its scaffolded, stepwise guidance as more constructive than answer-giving systems. In other words, participants preferred a tutor that encouraged reasoning, even when it required more effort, over a model that simply solved the problems for them.

Chapter 6

Discussion

6.1 The Need for Innovative Pedagogy in the Age of Generative AI

In a way, our approach may sound paradoxical. The introduction and ease of accessibility of Generative AI have created an educational problem that we attempt to address by introducing another accessible technology. Philosophically, this approach can cause a vicious cycle of unending problems created and treated by new technologies. To prevent this situation from happening, we recognize that a technological solution alone will never be sufficient [20]. Hence, our approach supplements a novel system design with innovative *pedagogy*. We do recognize that it would be unrealistic to expect students to refrain from using generative AI tools like GPT, even if they are explicitly prohibited from doing so by the rules and regulations of their educational environments. Educational institutions have so-called "sticks and carrots" at their disposal to discourage and encourage certain behaviors among their students [25]. Rather than prohibiting the usage of any generative AI tools, students can be persuaded that relying on the prescribed AI-powered tutor would be more conducive to achieving their educational objectives than using raw generative AI. The required pedagogical approach needs to be rooted in persuasion tactics. Our goal is to apply AI in a balanced and thoughtful way, so students are supported and empowered, without diminishing the

value of human-driven learning processes.

6.2 Limitations

As a proof-of-concept study, our implementation and evaluation of **SPARC** are subject to certain limitations.

First, the pilot study design limits the statistical generalizability of our findings. Although the results demonstrate consistent trends across metrics and survey responses, the participant sample remains relatively small and homogeneous, comprising primarily computer science students from one institution [34]. Longer-term classroom deployments involving a broader demographic of learners are necessary to confirm the observed effects on learning outcomes.

Second, the system experienced minor latency due to its multi-stage reasoning pipeline, reflecting its dependence on model performance and network conditions [18]. While participants noted this delay, it did not significantly affect engagement, but it remains an engineering consideration for future optimization.

Finally, our findings on ethical and pedagogical generalization are confined to secure-programming exercises within the SecureCoder platform. Further validation across broader curricular contexts is needed to confirm the tutor's wider applicability.

Despite these limitations, our pilot study provides encouraging evidence that an adaptive, integrity-preserving tutor can meaningfully support secure-programming education while maintaining ethical instructional standards.

6.3 Future Directions

The encouraging findings from this pilot study open several promising directions for advancing SPARC as both a research tool and an educational aid. Building on our preliminary evaluation, future work will expand the system's pedagogical reach and empirical grounding. We plan to conduct larger classroom studies that examine the tutor's long-term impact on knowledge retention, engagement, and student confidence when learning secure programming concepts [32]. Such studies would provide stronger evidence of how adaptive, hint-based guidance translates into measurable learning gains over time.

Beyond validation, we envision evolving SPARC into a more comprehensive active learning companion for CS education. Integrating it into full course modules or challenge-based activities—similar to capture-the-flag environments—could promote self-paced learning while preserving pedagogical guardrails. We also foresee a communal contribution model, in which educators and security practitioners can propose and refine new exercises, extending the system's relevance and diversity of content.

Ultimately, our goal is to establish SPARC as a platform for studying how adaptive, ethically constrained AI systems can foster deeper reasoning and reflective problem-solving in Computer Science education in general.

Chapter 7

Conclusion

This thesis has presented SPARC, an adaptive, integrity-preserving tutoring system designed to guide students through the fundamentals of secure programming. Integrated with the SecureCoder platform, the SPARC tutor employs prompt engineering, a novel combination of AI techniques that we coined as *a learning safeguard proxy*, and proficiency adaptation that delivers pedagogically aligned, hint-based assistance rather than direct answers.

Findings from our pilot study demonstrate that SPARC effectively supports comprehension, adaptivity, and ethical learning behavior. Learners achieved higher success rates with fewer attempts, and feedback analysis showed the tutor's ability to dynamically refine its hints based on user responses. Moreover, the system consistently maintained high instructional integrity, providing guidance without revealing solutions—an approach strongly preferred by participants for learning secure coding skills.

By combining adaptivity with ethical guardrails, SPARC outlines a novel solution for applying generative AI to enhance, rather than replace, authentic learning. The results presented herein highlight the potential of AI-driven tutors to promote reflective reasoning, encourage active engagement, and support scalable, trustworthy education in software security and beyond.

Bibliography

[1] Vincent Aleven, Bruce Mclaren, Ido Roll, and Kenneth Koedinger. Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor. *International Journal of Artificial Intelligence in Education*, 16(2):101–128, 2006.

[2] Morteza Bahrami, Muharram Mansoorizadeh, and Hassan Khotanlou. Few-shot learning with prompting methods. In *2023 6th International Conference on Pattern Recognition and Image Analysis (IPRIA)*, pages 1–5. IEEE, 2023.

[3] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[4] Patrick Bassner, Eduard Frankford, and Stephan Krusche. Iris: An ai-driven virtual tutor for computer science education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 394–400. 2024.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] PL Brusilovsky. Intelligent tutor, environment and manual for introductory programming. *Educational & Training Technology International*, 29(1):26–34, 1992.

[7] John D Bustard. Improving student engagement in the study of professional ethics:

Concepts and an example in cyber security. *Science and Engineering Ethics*, 24(2): 683–698, 2018.

[8] Yulin Chen, Haoran Li, Yuan Sui, Yue Liu, Yufei He, Yangqiu Song, and Bryan Hooi. Robustness via referencing: Defending against prompt injection attacks by referencing the executed instruction, 2025. URL https://arxiv.org/abs/2504.20472.

[9] Hongmei Chi, Edward L Jones, and John Brown. Teaching secure coding practices to stem students. In *Proceedings of the 2013 on InfoSecCD'13: Information Security Curriculum Development Conference*, pages 42–48, 2013.

[10] Wei Dai, Jionghao Lin, Hua Jin, Tongguang Li, Yi-Shan Tsai, Dragan Gašević, and Guanliang Chen. Can large language models provide feedback to students? a case study on chatgpt. In *2023 IEEE international conference on advanced learning technologies (ICALT)*, pages 323–325. IEEE, 2023.

[11] H. B. Essel, D. Vlachopoulos, A. Tachie-Menson, E. E. Johnson, and P. K. Baah. The impact of a virtual teaching assistant (chatbot) on students' learning in ghanaian higher education. *International Journal of Educational Technology in Higher Education*, 19(1): 1–19, 2022.

[12] Ali Farooq, Antti Hakkala, Seppo Virtanen, and Jouni Isoaho. Cybersecurity education and skills: exploring students' perceptions, preferences and performance in a blended learning initiative. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1361–1369. IEEE, 2020.

[13] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[14] Louie Giray. Prompt engineering with chatgpt: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633, 2023.

[15] Julie M Haney, Clyburn Cunningham IV, and Susanne M Furman. Towards bridging the {Research-Practice} gap: Understanding {Researcher-Practitioner} interactions and challenges in {Human-Centered} cybersecurity. In *Twentieth Symposium on Usable Privacy and Security (SOUPS 2024)*, pages 567–586, 2024.

[16] Muhamad Agreindra Helmiawan, Esa Firmansyah, Irfan Fadil, Yanvan Sofivan, Fathoni Mahardika, and Agun Guntara. Analysis of web security using open web application security project 10. In *2020 8th International Conference on Cyber and IT Service Management (CITSM)*, pages 1–5. IEEE, 2020.

[17] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

[18] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274, 2023.

[19] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*, pages 1–20, 2024.

[20] Nir Kshetri. The economics of generative artificial intelligence in the academic industry. *Computer*, 56(8):77–83, 2023.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al.

Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[22] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, pages 1–11, 2023.

[23] Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*, 2022.

[24] Diane Litman and Scott Silliman. Itspoke: An intelligent tutoring spoken dialogue system. In *Demonstration papers at HLT-NAACL 2004*, pages 5–8, 2004.

[25] Chen-Chung Liu, Mo-Gang Liao, Chia-Hui Chang, and Hung-Ming Lin. An analysis of children'interaction with an ai chatbot and its impact on their interest in reading. *Computers & Education*, 189:104576, 2022.

[26] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. Teaching cs50 with ai: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM technical symposium on computer science education V. 1*, pages 750–756, 2024.

[27] Fiona MacDowall. What is rag? *Austl. L. Libr.*, 32:94, 2024.

[28] David J Malan, Brian Yu, and Doug Lloyd. Teaching academic honesty in cs50. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 282–288, 2020.

[29] Steve Mansfield-Devine. Verizon: Data breach investigations report, 2022.

[30] MITRE. Common weakness enumeration (cwe). https://cwe.mitre.org. Accessed: 2025-05-28.

[31] Madhav Mukherjee, Ngoc Thuy Le, Yang-Wai Chow, and Willy Susilo. Strategic approaches to cybersecurity learning: A study of educational models and outcomes. *Information*, 15(2):117, 2024.

[32] Ghulam Mustafa, Tanzeela Urooj, and Muhammad Aslam. Role of artificial intelligence for adaptive learning environments in higher education by 2030. *Journal of Social Research Development*, 5(3), 2024.

[33] Zachary A Pardos and Shreya Bhandari. Learning gain differences between chatgpt and human tutor generated algebra hints. *arXiv preprint arXiv:2302.06871*, 2023.

[34] Denise F Polit and Cheryl Tatano Beck. Generalization in quantitative and qualitative research: Myths and strategies. *International journal of nursing studies*, 47(11):1451–1458, 2010.

[35] John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, Jacob Menick, Jiayi Weng, Juan Felipe Ceron Uribe, Liam Fedus, Luke Metz, Michael Pokorny, et al. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2(4), 2022.

[36] Leo St. Amour and Eli Tilevich. Designing a platform to train secure programming skills with attack-and-defend exercises. In *2025 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–10. IEEE, 2025.

[37] Michael J Streibel. A critical analysis of computer-based approaches to education: Drill-and-practice, tutorials, and programming/simulations. 1985.

[38] Blair Taylor, Matt Bishop, Elizabeth Hawthorne, and Kara Nance. Teaching secure

coding: the myths and the realities. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 281–282, 2013.

[39] Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

[40] Prokopia Vlachogianni and Nikolaos Tselios. Perceived usability evaluation of educational technology using the system usability scale (sus): A systematic review. *Journal of Research on Technology in Education*, 54(3):392–409, 2022. doi: 10.1080/15391523.2020.1867938. URL https://doi.org/10.1080/15391523.2020.1867938.

[41] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

[42] Stephen Woodcock and Jay Falletta. A numerical evaluation of the finite monkeys theorem. *Franklin Open*, 9:100171, 2024.

[43] Ruiwei Xiao, Xinying Hou, Runlong Ye, Majeed Kazemitabaar, Nicholas Diana, Michael Liut, and John Stamper. Improving student-ai interaction through pedagogical prompting: An example in computer science education. *arXiv preprint arXiv:2506.19107*, 2025.

Appendices

Appendix A

First Appendix

A.1 Code and Repository Information

The source code developed for this research project is maintained in two private GitHub repositories.

Backend Engine: https://github.com/SoftwareInnovationsLab/ai-tutor-backend

UI: https://github.com/SoftwareInnovationsLab/ai-tutor

An anonymized dataset containing participant responses to the survey is available at Question Pro cloud. All personal identifiers have been removed in accordance with research ethics guidelines.

A hands-on version of the tool can be accessed through the SecureCoder platform at: https://softwareinnovationslab.github.io/secure-coder-ui/#/

A.2 QuestionPro Survey

The following pages show the full QuestionPro survey as it appeared to participants during the user study.

Thank you for completing the SecureCoder exercises!
Please take 3–5 minutes to answer this short questionnaire about your experience.
Your responses will help us evaluate how the AI Tutor supports learning in secure programming.
All responses will remain confidential. Thank you very much for your time and support.

Please start with the survey now.

* Email Address (VT email)

Overall Learning & Confidence

* The hints provided by THE AI Tutor were clear and easy to understand.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

* The AI Tutor adapted its responses effectively based on my feedback.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

* The AI Tutor helped me understand secure programming concepts better.

Strongly Disagree				Strongly Agree
1	2	3	4	5

Task Performance & Usefulness

* The hints encouraged me to think through the problem rather than copy the answer.

Strongly Disagree — Strongly Agree

1 2 3 4 5

* The interface and interaction flow were intuitive and easy to use.

Strongly Disagree — Strongly Agree

1 2 3 4 5

* I was able to complete the exercises efficiently with the AI Tutor's help.

Strongly Disagree — Strongly Agree

1 2 3 4 5

Pedagogical Quality & Clarity

* The AI Tutor's hint difficulty adjusted appropriately to my understanding.

Strongly Disagree — Strongly Agree

1 2 3 4 5

* I felt the Tutor's responses became more relevant as I interacted more.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

* The adaptive hints made the learning process more engaging.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

Guardrail & Ethical Behavior

* The AI Tutor avoided giving full code or direct answers, and instead guided me with hints.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

* Even when the Tutor refused to give direct answers, it remained helpful.

Strongly Disagree				Strongly Agree
1	2	3	4	5
○	○	○	○	○

Preference & Future Use

* Which system did you find more useful overall? (Learning wise)

○ AI Tutor

- ◯ Open AI's GPT
- ◯ Both Equally

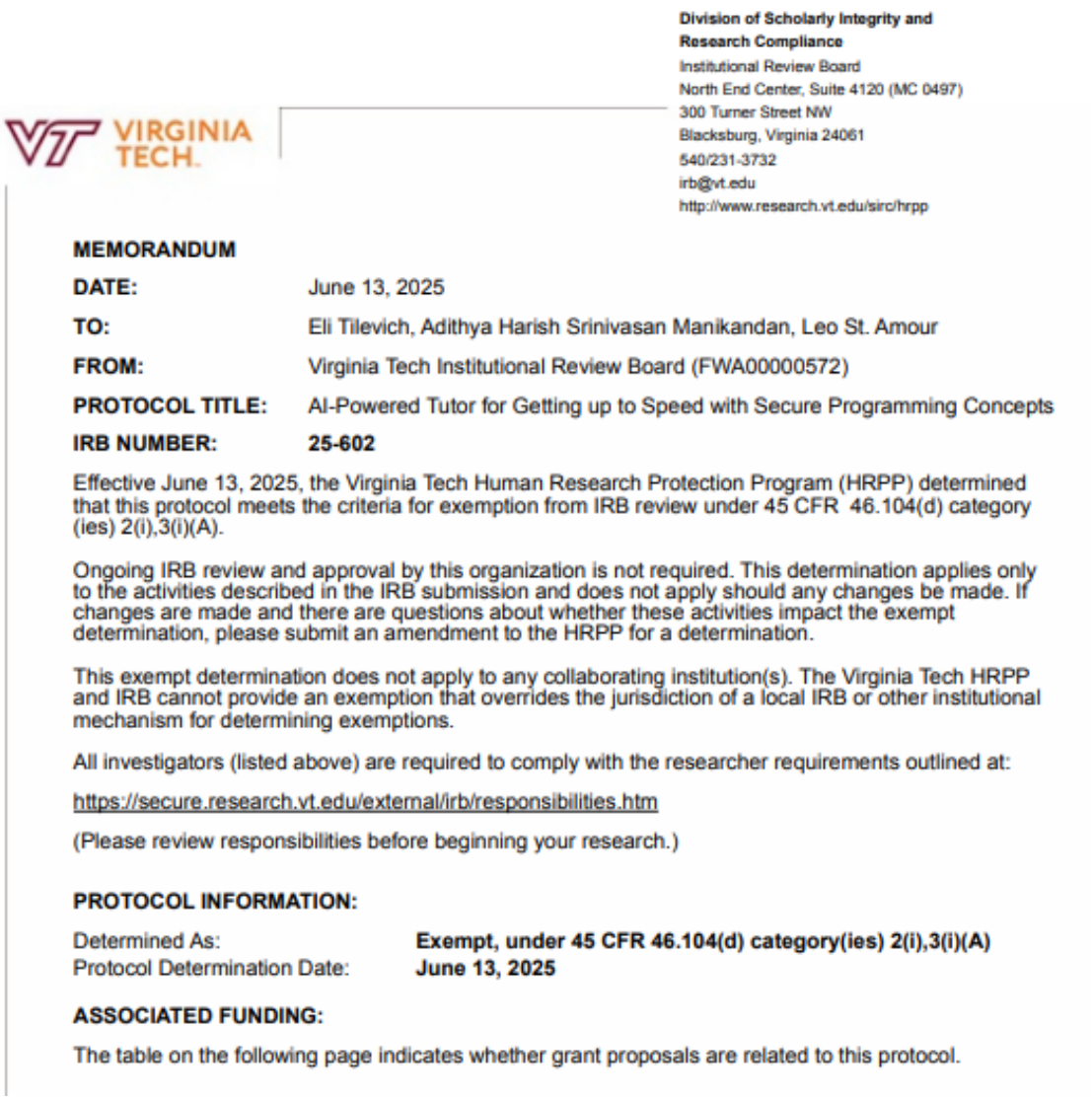
* Would you like to use the AI Tutor again for learning secure programming?

- ◯ Yes
- ◯ Maybe
- ◯ No

Please share any comments about your experience—what you liked, what could improve, or any issues you encountered.

A.3 IRB Approval Letter

The Institutional Review Board (IRB) approved this study under protocol 25-602. The complete approval letter is included below.



VIRGINIA TECH

Division of Scholarly Integrity and Research Compliance
Institutional Review Board
North End Center, Suite 4120 (MC 0497)
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-3732
irb@vt.edu
http://www.research.vt.edu/sirc/hrpp

MEMORANDUM

DATE: June 13, 2025

TO: Eli Tilevich, Adithya Harish Srinivasan Manikandan, Leo St. Amour

FROM: Virginia Tech Institutional Review Board (FWA00000572)

PROTOCOL TITLE: AI-Powered Tutor for Getting up to Speed with Secure Programming Concepts

IRB NUMBER: **25-602**

Effective June 13, 2025, the Virginia Tech Human Research Protection Program (HRPP) determined that this protocol meets the criteria for exemption from IRB review under 45 CFR 46.104(d) category (ies) 2(i),3(i)(A).

Ongoing IRB review and approval by this organization is not required. This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made and there are questions about whether these activities impact the exempt determination, please submit an amendment to the HRPP for a determination.

This exempt determination does not apply to any collaborating institution(s). The Virginia Tech HRPP and IRB cannot provide an exemption that overrides the jurisdiction of a local IRB or other institutional mechanism for determining exemptions.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

https://secure.research.vt.edu/external/irb/responsibilities.htm

(Please review responsibilities before beginning your research.)

PROTOCOL INFORMATION:

Determined As: **Exempt, under 45 CFR 46.104(d) category(ies) 2(i),3(i)(A)**
Protocol Determination Date: **June 13, 2025**

ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol.

Figure A.1: IRB Approval Letter

Appendix B

Second Appendix

B.1 Exploration

B.1.1 Initial Vision: A Generic Adaptive AI Tutor

The project began with the goal of developing a general-purpose AI Tutor capable of adaptively responding to a broad range of DSA (Data Structures and Algorithm) questions. The core idea was for the system to tailor the depth and tone of its explanations according to each learner's proficiency level—Beginner, Intermediate, or Advanced.

To prototype this vision, we built a modular architecture with a Flask-based backend, a MongoDB database, and a lightweight UI that recorded all conversations. Each interaction was logged with fields for the user's question, the AI's response, and feedback labels ("Satisfied" or "Need More Clarity").

We then implemented feature-extraction routines to analyze these conversations. Using NLP techniques such as question-complexity scoring, term-frequency analysis, and concept tagging with spaCy and Transformers, the system identified commonly queried topics and tracked how users reacted to different styles of explanation. This data enabled early inference of both overall proficiency and concept-specific mastery.

B.1.2 Prompt Adaptation and Model Training

To make the tutor's responses adaptive, we added a prompt-rewriting mechanism that reformulated the same question differently for learners at various levels. For example, the question "Explain quicksort" was adapted as:

- Beginner: "Explain quicksort in simple terms with an example."
- Intermediate: "Guide me through how quicksort works, but let me reason part of it."
- Advanced: "Discuss the edge cases and efficiency trade-offs in quicksort."

We curated a dataset of over a hundred such prompt pairs and experimented with prompt tuning using T5-small and T5-base models. The goal was for the model to automatically produce level-appropriate explanations based on inferred proficiency.

However, the domain's breadth—especially in Data Structures and Algorithms—made generalization difficult. Achieving meaningful adaptivity required a much larger and continuously expanding dataset. Handling diverse query phrasing, ensuring consistency, and managing GPU limits during T5-large experiments also posed practical challenges.

B.1.3 Lessons Learned and Transition to Secure Coder

These early explorations made it clear that a generic adaptive tutor would require vast domain coverage and continual retraining to sustain accuracy. To address this, we shifted focus to a more constrained and pedagogically rich domain—Secure Programming.

By integrating the AI Tutor with the SecureCoder platform, the system operated within a well-defined context: guiding students through secure-coding exercises. This shift preserved the adaptive framework developed earlier (feature extraction, proficiency tracking,

and prompt adjustment) but applied it to security-specific datasets grounded in CWE references.

Moreover, since SecureCoder is an interactive exercise platform, it enabled us to objectively measure the tutor's effectiveness. We could now track quantitative metrics such as success rate, number of attempts, and time-to-solution for each exercise, providing a clear basis for evaluating learning impact and system behavior.

This transition improved both accuracy and instructional alignment. The tutor could now generate hints that were precise, contextually relevant, and pedagogically sound, helping students reason about vulnerabilities rather than simply reproducing answers.