Enabling Secure Coding: Exploring GenAI for Developer Training and Education

Sathwik Amburi
Siemens AG, Munich, Germany
Universität der Bundeswehr München, Germany

Tiago Espinha Gasiba
Siemens AG, Munich, Germany

Ulrike Lechner
Universität der Bundeswehr München, Germany

Maria Pinto-Albuquerque
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Abstract

The rapid adoption of GenAI for code generation presents unprecedented opportunities and significant security challenges. Raising awareness about secure coding is critical for preventing software vulnerabilities. To investigate how Generative AI can best support secure coding, we built an AI Secure Coding platform, an interactive training environment that embeds a GPT-4 based chatbot directly into a structured challenge workflow. The platform comprises a landing page, a challenges page with three AI-generated tasks, and a challenge page where participants work with code snippets. In each challenge, developers (1) identify vulnerabilities by reviewing code and adding comments, (2) ask the AI for help via a chat based interface, (3) review and refine comments based on AI feedback, and (4) fix vulnerabilities by submitting secure patches. The study involved 18 industry developers tackling three challenges. Participants used the AI Secure Coding Platform to detect and remediate vulnerabilities and then completed a survey to capture their opinions and comfort level with AI assisted platform for secure coding. Results show that AI assistance can boost productivity, reduce errors, and uncover more defects when treated as a "second pair of eyes," but it can also foster over-reliance. This study introduces the AI Secure Coding platform, presents preliminary results from a initial study, and shows that embedding GenAI into a structured secure-coding workflow can both enable and challenge developers. This work also opens the door to a new research field: leveraging GenAI to enable secure software development.

2012 ACM Subject Classification Applied computing → Learning management systems; Security and privacy → Software security engineering; Applied computing → Distance learning; Applied computing → E-learning

Keywords and phrases Secure Coding, Industry, Software Development, Generative AI, Large Language Models, Teaching

Digital Object Identifier 10.4230/OASIcs.ICPEC.2025.2

Funding *Ulrike Lechner*: Ulrike Lechner acknowledges funding from the European Union's Next-GenerationEU program for the LIONS project as part of dtec.bw.
Maria Pinto-Albuquerque: This work is partially financed by national funds through FCT - Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and UIDP/04466/2020. Furthermore, Maria Pinto-Albuquerque acknowledges and thanks the Instituto Universitário de Lisboa and ISTAR, for their support.

1 Introduction

Secure coding is the practice of writing code in a way that proactively mitigates security vulnerabilities in software. It is essential because a large share of security incidents stem from flaws in software code. In fact, according to the U.S. Department of Homeland Security, 90% of reported security incidents result from exploits against defects in the design or code of software [5]. Secure software development is crucial in industrial settings because neglecting it can lead to safety issues or significant financial and reputational damage. Educating developers about secure coding practices has therefore become a priority for reducing the risk of breaches and costly fixes. Organizations recognize that educating developers about security can dramatically improve outcomes [24, 3, 14].

Artificial intelligence (AI) is rapidly transforming software development, forcing software teams to balance rapid feature delivery with security considerations. While AI's impressive code-generation capabilities accelerate development, they also introduce a wave of new security risks [17]. Because Generative AI (GenAI) is so effective at generating code, some AI model providers and AI evangelists claim that AI will soon write almost all software [23]. However, an empirical study by Fu et al. analyzed Copilot-generated snippets drawn from public GitHub projects and found that 29.5% of Python and 24.2% of JavaScript samples contained at least one Common Weakness Enumeration (CWE) vulnerability [8]. Although AI coding assistants have impressive capabilities, security remains a concern in the code they produce. As AI increasingly generates code, developers' roles are shifting toward supervising AI systems rather than manually writing every line. This transition requires developers to maintain strong security awareness to guide AI-generated code toward secure outcomes.

To keep pace with GenAI-augmented development, developers must learn not only classical secure-coding principles but also new skills for using GenAI responsibly. Prompt engineering and understanding of limitations of GenAI are becoming as important as traditional secure-coding checklists. Without explicit guidance, developers may over-rely on AI suggestions, unknowingly propagating subtle vulnerabilities.

To investigate how GenAI can best support secure coding, we built an AI Secure Coding platform, an interactive training environment that embeds a chatbot directly into the challenge workflow. The AI Secure Coding platform was inspired by the Sifu platform [10]. The chatbot has access to participants' source code and review comments. It can suggest mitigations, generate secure patches, and explain underlying principles. The platform logs all interactions, allowing us to observe usage patterns while participants solve secure-coding challenges.

Using the AI Secure Coding platform with industry developers, we aim to answer the following research questions:

- **RQ1**: How do developers use GenAI to identify and remediate security vulnerabilities during secure coding challenges?
- **RQ2**: What are developers' opinions about GenAI and how do they feel about using GenAI for secure coding?

Insights from the above research questions would help inform a curriculum that enables developers to leverage GenAI efficiently without over-reliance while prioritizing security when coding. This study introduces the AI Secure Coding platform and presents preliminary results from the initial study. The platform is being developed iteratively following Hevner's Design Science Research (DSR) methodology [12].

This paper reports one DSR cycle [12] with a focus on problem understanding, it outlines the initial design of the AI Secure Coding platform, and shares initial findings from the initial study. By focusing on developers in the industry, this study delivers industry-relevant guidance on weaving GenAI into secure-coding training while ensuring that security remains an uncompromised top priority.

In Section 2, we review the related work for our research. In Section 3, we describe the experimental setup of the initial study used to address our research question. Section 4 presents the results of the initial study. In Section 5, we discuss and reflect on the obtained findings. Finally, Section 6 concludes the paper and outlines directions for future work.

This paper also opens the door and provides a first step toward a new and rich field of research: how to leverage GenAI to enable *secure software development*.

2 Related Work

In this section, we discuss several studies that highlight recent research about large language models, GenAI techniques and review their role in security, education and secure software development. Since the advent of Github Copilot, use of GenAI for code generation has been increasing at an alarming rate. Perry et al. [19] found that users produced more insecure code, yet remained confident in its security when using AI assistants. The study included 47 participants: 33 assigned to the "AI-assistant" experiment group and 14 to the "no-assistant" control group. Participants each solved five security-focused coding tasks (Python: encryption/decryption, ECDSA signing, sandboxed file access; JavaScript: SQL insertion; C: formatted integer-to-string) under a two-hour time limit. The authors concluded that AI code assistants can inadvertently erode security by fostering overconfidence and automating flawed coding patterns. Participants who actively structured prompts, verified outputs, and understood security APIs performed better, suggesting that future assistants should integrate security-aware prompting, library-usage warnings, and automated vulnerability checks in-tool.

Belzner et al. [4] in their study argue that LLM-assisted software engineering holds immense potential and if key challenges are addressed, it would revolutionize the software engineering process. One such challenge they discuss is maintaining accountability for generated content. This issue is also prevalent in the code that is generated by these LLMs. It is essential that developers who use LLMs to generate code remain accountable for security of the code LLMs generate. Belzner et al. also emphasize the need for robust evaluation of generative outputs to guard against hallucinations and ensure factuality and coherence in generated code. They further highlight challenges in integrating LLMs into large development contexts and the importance of defining structured interfaces and knowledge-graph–based approaches to maintain contextual continuity. Sallou et al. [22] also echo similar concerns about the threats of using LLMs in software engineering and provide guidelines for software engineering researchers on using LLMs. One such concern they discuss is reproducibility. It is well known that LLMs generate variable texts for the same questions, making it difficult to address issues if one cannot reproduce the same output. They recommend assessing output variability and providing execution metadata to address reproducibility concerns. The guidelines, concerns, and challenges outlined by Belzner et al. and Sallou et al. make clear that developers leveraging LLMs must recognize their limitations and remain accountable for the code these models produce. Another study by Kharma et al. [15] examines the security and quality of LLM-generated code, revealing that while LLMs can automate code creation, their effectiveness varies by language. Many models neglect modern security features introduced in recent compiler and toolkit updates (for example, Java 17), and outdated practices persist,

especially in C++. Their study also uncovers stark discrepancies in semantic evaluation success rates, averaging just 8.5% across models, with Gemini-1.5 peaking at 13.5%. These gaps suggest that training data and model design heavily influence code-logic processing. Such studies help both the software engineering and security communities understand these constraints and incorporate corresponding guidelines into their workflows.

GenAI and LLMs can also enhance programming education and secure-coding training. Gasiba et al. [9, 6] demonstrate in their studies how ChatGPT can raise secure-coding awareness. They note that although models like GPT-3.5 and GPT-4 are not 100% accurate at finding vulnerabilities, they serve as a helpful initial tool for spotting security issues. However, excessive dependence on LLMs may discourage critical thinking. Balaz et al. [2] illustrate this in an empirical study with computer-science students, showing that while AI tools increase efficiency, they can also hinder the development of critical thinking and problem-solving skills. A larger Microsoft study by Lee et al. [16] reaches similar conclusions. The study finds that although GenAI can boost productivity, it may reduce deep engagement and foster long-term reliance, eroding independent problem-solving abilities.

One way to enable developers to learn these guidelines, best practices, and limitations is through targeted training and education programs in the industry; where, conveniently, LLMs themselves can be employed to deliver interactive, hands-on exercises and personalized learning experiences. Wang et al. [25] present a literature survey on the role of large language models (LLMs) in education. The survey provides a comprehensive review of LLM applications across various educational scenarios and envisions LLMs aiding in question solving (QS), in error correction (EC), as confusion helper (CH), in question generation (QG), in automatic grading (AG), in material creation (MC), at knowledge training (KT), and at content personalization (CP). Huber et al. [13] state that LLMs in education may pose some risk of over-reliance, potentially and unintentionally limiting the development of domain expertise, and argue that challenges such as bias, misinformation, and deskilling might be addressed through playful and game-based learning. They argue that effective human–AI collaboration depends on deliberate practice through three interrelated learning processes. First, learners apply abstract concepts to real-world tasks to gain hands-on experience. Second, they analyse those experiences to develop a clear understanding of the underlying ideas. Third, they critically reflect on both their actions and their newly formed understanding to refine and improve their problem-solving strategies. Together, this cycle of doing, understanding, and reflecting builds the deep expertise needed to work effectively with LLMs. Routine educational tasks often demand effortful practice, but the convenience of LLMs can undermine this struggle, leading novices to engage superficially, accept AI-generated content uncritically, and miss critical opportunities to develop judgment competence. To counter these risks, the authors propose playful approaches, including prompt engineering as a form of playful interaction with LLMs, which invites experimentation, role-play, and iterative refinement of prompts. Huber et al. [13] further advocate game-based learning to harness human–AI interaction by embedding LLMs within well-designed game environments that foster motivation, reflection, and sustained practice. They highlight gamifying learning materials using generative AI to create game-like educational experiences, as a promising pathway to deepen engagement and preserve essential expertise development.

Focusing on GenAI for computer programming education, Bauer et al. [3] present the challenges and opportunities in automating the creation of programming exercises for automated assessment. Ultimately, a refined feature to assist teachers in generating such exercises was integrated into Agni [21], a web-based computer programming learning playground. Similar work by Ricardo Queirós [20] and by Freitas et al. [7] demonstrates exercise generation using GenAI.

In summary, this related work shows that GenAI and LLMs can dramatically speed up code generation, strengthen security analysis, and enrich developer education but also raise serious challenges in accountability, reproducibility and overreliance on automated outputs. Addressing these issues requires robust evaluation methods, thoughtful integration strategies and targeted training programs that reinforce developers' secure coding skills.

To build on prior studies, we investigate real-world developer usage to reveal how GenAI tools influence secure coding practices. Our results will inform practical guidelines and educational interventions designed to help developers leverage GenAI advantages while maintaining rigorous security standards.

3 Experiment

To set up our experiment, we created an AI powered secure coding platform. The platform was developed in February 2025 and debuted at a Secure Coding Workshop in March 2025. The platform consisted of a simple landing page, a challenges page containing three AI-generated challenges, and a challenge page where users interacted with a code snippet.

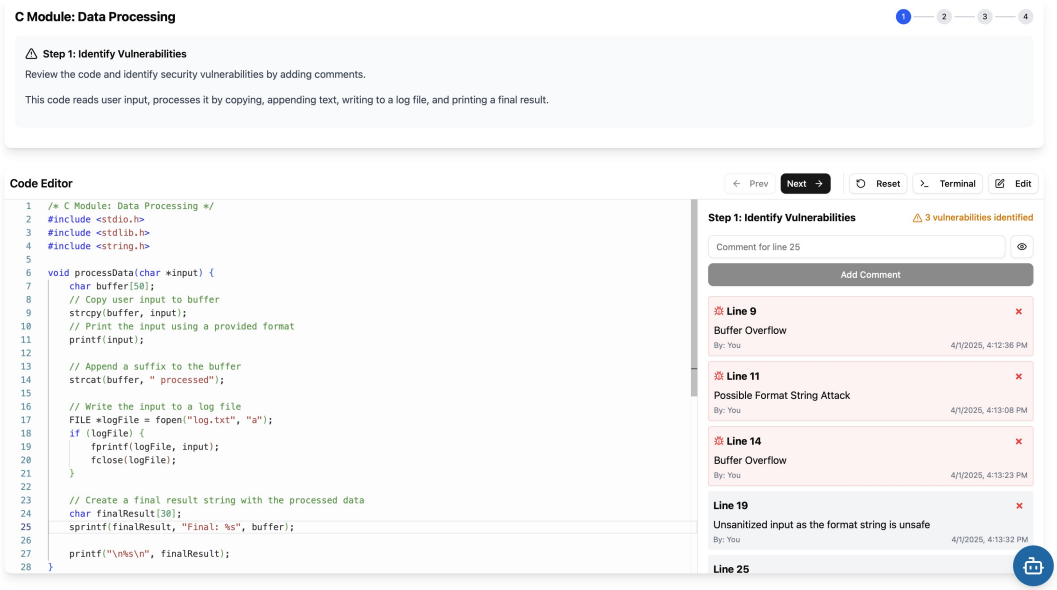


Figure 1 Challenge Page: Step 1 - Identify Vulnerabilities.

The challenge page comprised a problem section, a code editor featuring a vulnerable code snippet, and a floating chat widget (as shown in Figure 1). The code editor component consisted of a Monaco code editor [18] displaying a vulnerable code snippet, a review section where users could identify vulnerabilities and leave code comments, and finally, a terminal section that was included to simulate a realistic development environment but remained non-functional as a mock interface element.

Each challenge followed a structured workflow (see Figure 2) consisting of the following steps:

- **Step 1: Identify Vulnerabilities**: Review the code and identify security vulnerabilities by adding comments.
- **Step 2: Ask AI for Help**: Ask the AI assistant for help understanding the vulnerabilities you found.

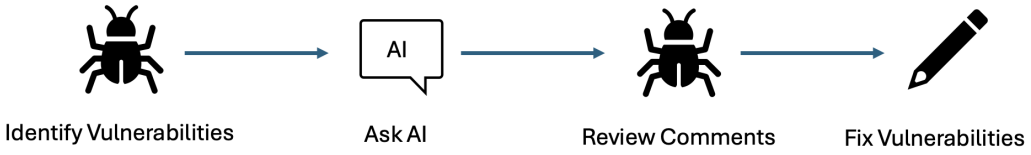


Figure 2 Challenge Workflow.

- **Step 3: Review Comments**: Review and refine your vulnerability comments based on AI feedback.
- **Step 4: Fix Vulnerabilities**: Fix the vulnerabilities in the code. Use Edit Mode to make your changes.

This workflow was designed to highlight independent thinking without overreliance on the AI chat. In the first step, participants were asked to identify vulnerabilities to ensure they applied their learnings from the workshop. Users were only allowed to access the AI chat from Step 2 to enable developers to understand and identify vulnerabilities on their own without any assistance. They could then interact with the AI chat and ask questions about the code snippet and their comments on the snippet (as shown in Figure 3). For the AI chat, we used the GPT-4o-mini model via the Azure OpenAI API endpoint. We explicitly selected this model because it serves as the base model for ChatGPT's free tier, and Microsoft Copilot's Model is likewise built on the GPT-4 architecture. The system prompt for the AI chat specified its role as a security assistant, responsible for helping users understand security vulnerabilities, suggest best coding practices, and address security-related questions.

In Step 3, users were allowed to go back to change their review comments and were given one last chance to identify all vulnerabilities in the code.

Finally, in Step 4, users were given the opportunity to edit the code snippet, fix the vulnerabilities they had identified, and submit the updated snippet for evaluation.

After the participants had interacted with the platform, they were asked to fill out a survey using Microsoft Forms. The survey had fourteen questions in total, questions ranging from professional coding experience to how much they can trust LLMs [1]. Some questions included Likert scale responses. The goal of this survey was to understand where developers in the industry stand on LLMs and how to better enhance secure software development using LLMs.

In summary, this experiment setup provided a controlled evaluation of our AI powered secure coding platform, allowing industry developers to identify, discuss, and remediate vulnerabilities in the challenges with real-time GenAI guidance. The purpose of this experiment was to assess the platform's usability, effectiveness, and impact on developers' secure-coding practices.

4 Results

The experiment was conducted at a Secure Coding Workshop in March 2025, which provided a controlled setting for the initial study. The initial study evaluates an AI secure coding platform designed to provide developers with real-time secure coding guidance. The workshop was held for three days; the first two days presented secure coding principles and security pitfalls in secure software development, and the third day was reserved for cybersecurity challenges from the Sifu platform [10]. The initial study took place at the end of the second day after presenting all the topics for secure coding and secure software development.

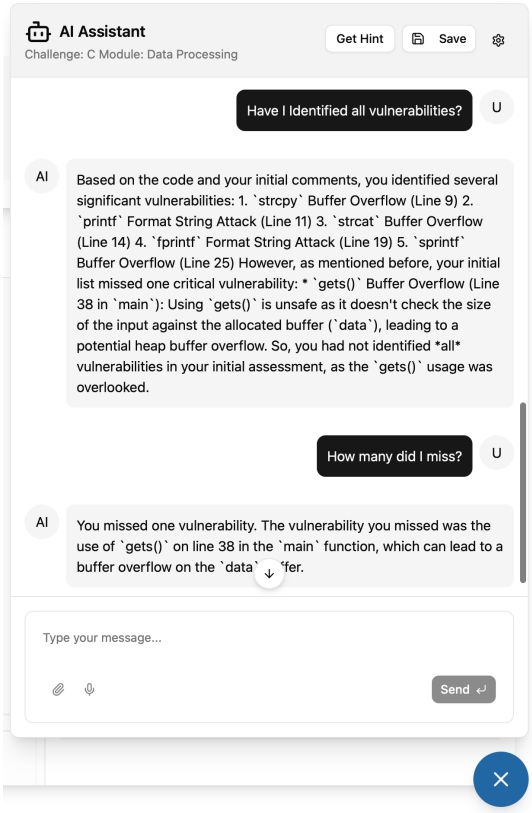


Figure 3 Challenge Page: Step 2 - Ask AI for Help.

Participants voluntarily interacted with the platform and completed a survey based on their interaction with the platform while also answering some general questions about LLMs and GenAI. Of the 19 participants in the workshop, 18 people decided to interact with the platform and participate in the survey, and 14 interacted with the platform and submitted at least one challenge. As depicted in Figure 4, the majority of survey participants have more than 10 years of experience.

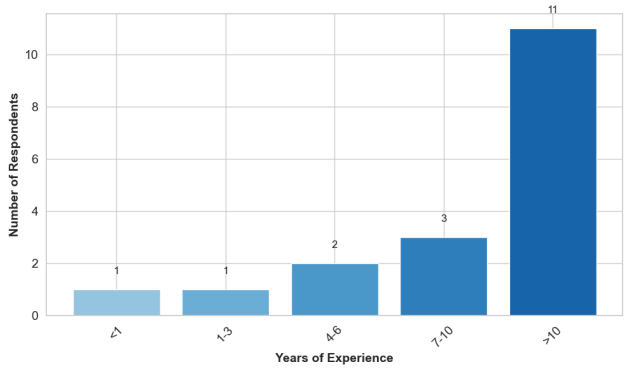
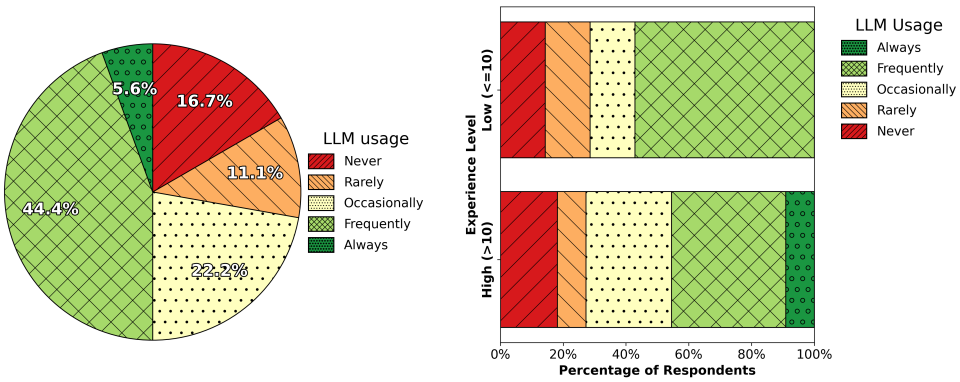


Figure 4 Professional Coding Experience Among Participants.

Figures 5a and 5b depict the LLM usage frequency graphs. At least 50% of the participants use LLMs always or frequently. Specifically, 44.4% of respondents reported using LLMs frequently and an additional 5.6% indicated they always rely on them for coding support, while the remaining participants fell into the occasional (22.2%), rare (11.1%), or never (16.7%) usage categories (Figure 5a). Developers with ten or fewer years of experience show the highest proportion of frequent LLM use, whereas the most experienced professionals exhibit a wider range of usage patterns and this includes the lone "always" user in our sample. Moreover, when broken down by experience level, 7 of 11 high experience developers (63%) perceived themselves at least "somewhat confident" in spotting vulnerabilities, including the only two "very confident" responses and 3 remained neutral and one was not confident at all, whereas in the low experience level, 5 of 7 (71%) felt somewhat confident, two participants expressed some level of doubt, and none claimed very high confidence. This is shown in Figure 6a. In parallel, roughly three quarters of the high experience level (8 of 11; 72%) completed each code review within 5–10 minutes, and only one person exceeded that window, compared with two people (nearly 30%) in the low experience level who spent over 10 minutes per review, hinting that more seasoned engineers may work more efficiently on vulnerability detection. This is shown in Figure 6b.

Figure 7a shows the quality of feedback provided by the LLMs compared to the manual review (by experience level). High-experience participants tend to rate LLM feedback more positively (as shown in Figure 7a). In detail, 3 of 11 senior developers (27%) judged the platform's suggestions as "good" and another 2 (18%) as "very good," with only one "poor" and two "fair" ratings, whereas in the low experience level, 4 of 7 (57%) found the feedback "good," one was "average," and two rated it "fair," and none called it "very good." Figure 7b depicts the willingness of developers by experience levels to adopt LLMs in code reviews. Both experience levels show a moderate to high willingness to adopt LLM-based code reviews (Figure 7b): among high experience users, 4 of 11 (36%) were "somewhat likely" and 4 (36%) "very likely" to integrate the tool (the remainder neutral), while the low experience cohort was even more enthusiastic; 5 of 7 (71%) said "very likely," 1 "somewhat likely," and only one remained neutral indicating towards broad acceptance across experience levels.

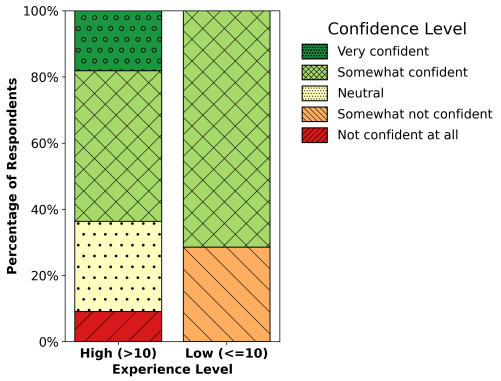
Figure 8 shows us opinions of developers on LLMs and GenAI in Software Development. When asked whether "Generative AI increases productivity," a strong majority backed the proposition: 12 of 18 respondents (67%) agreed or strongly agreed, while only three (17%) remained neutral and three (17%) voiced disagreement (including one strong dissenter).



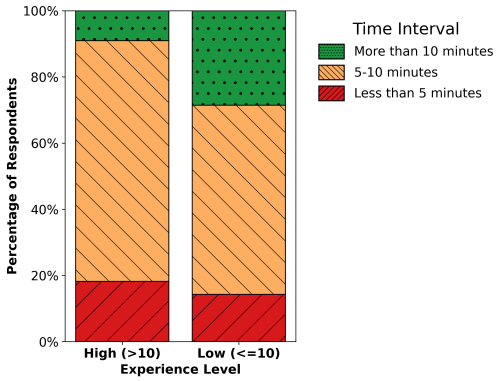
(a) LLM usage frequency among users.

(b) LLM usage by experience level.

Figure 5 LLM Usage Frequency Graphs.

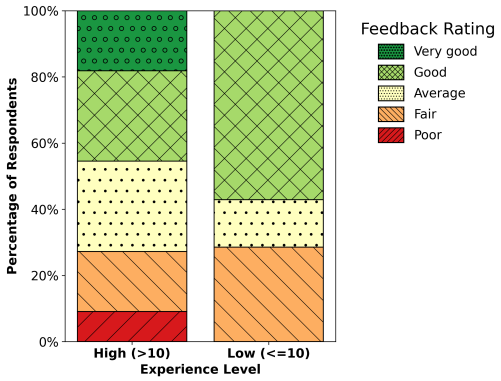


(a) Confidence in identifying vulnerabilities by experience level.

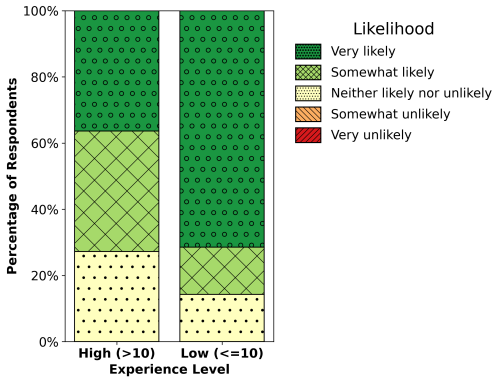


(b) Time spent reviewing code snippets by experience level.

Figure 6 Impact of Experience on Vulnerability Identification and Code Review Time.



(a) Quality of feedback provided by the LLM compared to manual review? (By experience level).



(b) Likelihood to incorporate LLM-Assisted code reviews by experience level.

Figure 7 Quality and Adoption of LLM-Assisted Code Reviews by Experience Level.

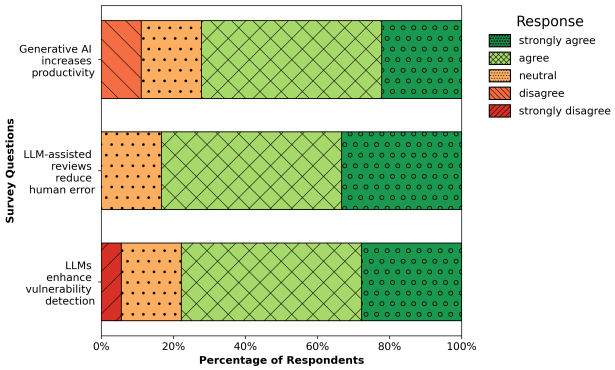


Figure 8 Opinions on LLMs and Generative AI in Software Development.

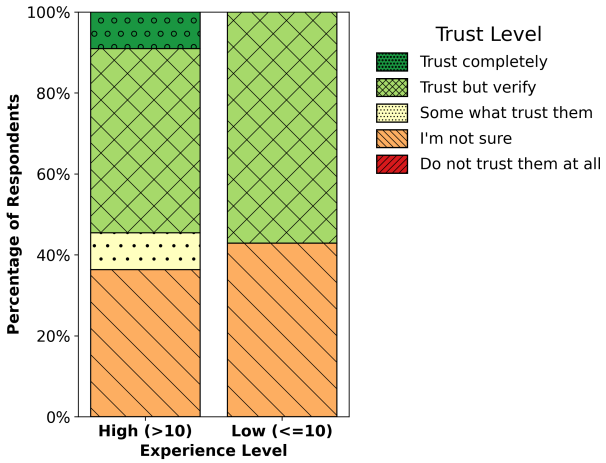


Figure 9 Trust in LLMs.

Similarly, on the statement "LLM-assisted reviews reduce human error," 11 participants (61%) registered agreement or strong agreement, with four (22%) neutral and three (17%) in partial or full dissent. Finally, regarding "LLMs enhance vulnerability detection," again 12 respondents (67%) agreed or strongly agreed and the remaining six split evenly between neutrality and disagreement. Taken together, these results reveal clear enthusiasm for AI-powered tools in productivity, error reduction, and vulnerability spotting, but the presence of up to one quarter neutral or dissenting responses highlights ongoing concerns about trust, reliability, and the necessity of human oversight when integrating LLMs into security workflows (Figure 8). Building on these mixed sentiments, the trust level breakdown (as depicted in Figure 9) reveals that while no participant in either experience levels "does not trust at all," the majority across both experience levels adopt a "trust but verify" stance, with 5 of 11 high experience developers (45%) and 4 of 7 low experience developers (57%) falling into this category. Only one senior engineer express full confidence(9%) and "trusts completely" and another one (9%) "somewhat trusts", whereas none of the less experienced cohort do so, and uncertainty persists (36% of seniors vs. 43% of juniors "not sure").

We recorded a total of 74 submissions from 14 unique users. Some users submitted the same challenge multiple times. For each submission, we recorded challenge details, user review comments, chat history, and the final code submission. We then cleaned the data by deduplicating entries and removing submissions with no chat history or those that primarily used other AI services (users were allowed to use services like Copilot or ChatGPT). After cleaning, we salvaged 28 submissions from 13 unique users that had valid chat history for analysis. The majority of submissions began with user messages such as "Have I found all vulnerabilities?" and "Find all vulnerabilities," showing that users expected the AI to give correct answers even though it did not have access to the vulnerabilities in the code.

Taken together, these findings demonstrate that seasoned developers not only engage readily with LLM-based secure coding tools but also report high confidence and efficiency gains when reviewing code. Although usage patterns vary by experience; junior developers lean more heavily on AI support while veterans show broader adoption; both cohorts express strong productivity and error-reduction benefits. Feedback quality and adoption willingness are likewise high across the board, yet the pervasive "trust but verify" mindset and a notable minority of neutral or dissenting opinions point to an enduring need for transparent model behavior and human oversight.

Finally, we observe risky interaction patterns: User submission patterns beginning with prompts like "Have I found all vulnerabilities?" reveal a clear expectation that the AI will identify every vulnerability. In practice, GenAI models tend to respond with hallucinations when prompted in such a way. This underscores the importance of setting appropriate tool limitations and educating users on AI capabilities. These insights motivate our subsequent discussion of design guidelines, deployment strategies, and future validation studies.

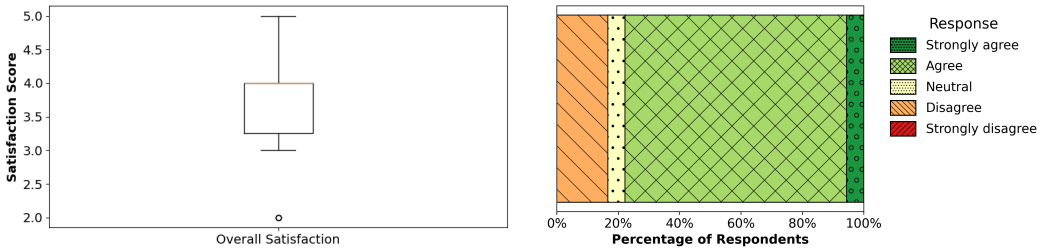
5 Discussion

The initial study reveals that embedding GenAI into a structured secure-coding workflow can both empower and challenge experienced developers. The strong uptake and positive ratings for feedback quality (Figure 7a) and adoption likelihood (Figure 7b) demonstrate that developers see real value in LLM-assisted reviews. Yet the persistent "trust but verify" stance (Figure 9) and the appearance of neutral or dissenting voices in our productivity and error-reduction measures (Figure 8) underscore that no matter how polished an AI interface becomes, human expertise and oversight remain indispensable.

Figures 10 and 10a depict user feedback on the platform. It affirms that the four-step challenge workflow: identify vulnerabilities manually, ask AI for help, review comments, and fixing vulnerabilities strike an effective balance between independent problem solving and AI support. Participants reported that forcing an initial round of manual review sharpened their mental model of the code, while the subsequent AI interaction helped them catch subtle issues they might otherwise have missed.

Our analysis of submission patterns and chat logs further highlights a critical user expectation: many participants opened each challenge with prompts like "Have I found all vulnerabilities?" reflecting an assumption that the AI "knows everything" in the snippet. This over-reliance risk points to the need for clearer in-tool guidance and just-in-time reminders about the model's knowledge limitations and inability to access hidden or out-of-scope code. Introducing lightweight static analysis hints or confidence indicators alongside AI suggestions could help calibrate trust and prevent blind acceptance of flawed recommendations [22]. In short, AI assistance can boost productivity, reduce errors, and uncover more defects when treated as a "second pair of eyes," but it can also foster over-reliance highlighting the need for confidence indicators and static-analysis cues.

Although generative models can accelerate secure-coding workflows, developers often fall into familiar traps. A widespread misbelief is that "AI = secure by default," leading users to accept generated code without scrutinizing its security properties. Under-specifying prompts exacerbates this risk: vague requests can yield insecure defaults or omit critical checks such as input validation and boundary enforcement. Over-reliance on AI suggestions can erode developers' own vulnerability-hunting skills, creating a feedback loop where the model's omissions become blind spots in human review. Despite impressive fluency, LLMs lack true code understanding and reasoning [11]. Their knowledge is frozen at the training cut-off and does not reflect emergent threats or zero-day exploits, so they cannot guarantee coverage of evolving vulnerabilities. Moreover, biases in training data may skew recommendations toward popular but insecure libraries or design patterns. Because LLMs do not produce provably correct solutions, they offer no formal assurance of completeness; unseen edge cases and logic flaws can persist. Performance constraints both in latency and computational cost may limit real-time integration for large codebases. Finally, ethical and licensing implications of repurposing copyrighted code snippets demand careful governance, and ultimately the human remains accountable for all production code.



(a) Overall satisfaction.

(b) Evaluation of Multi-step workflow of a challenge.

Figure 10 User feedback on the AI-powered secure coding platform.

5.1 Threats to Validity

Several factors may temper the internal validity and generalizability of our findings. First, our sample of 14 active users and 18 survey participants is modest and self-selected from volunteers at a secure-coding workshop in a single industrial context, introducing participation bias toward security-minded professionals. Second, the controlled three-day setting and curated Sifu challenges may not reflect real-world pressures, team dynamics, or code complexity. Third, we excluded submissions that lacked chat history or used alternate AI services, which could skew results toward more thorough or AI-reliant users. Finally, the short duration of the initial study precludes assessment of long-term learning effects or skill retention, and our focus on industry developers may not apply to less experienced developers or other domains.

6 Conclusion and Future Work

This paper presents an experiment where developers engaged with GenAI in a structured, four-step workflow: they first identified vulnerabilities, then used the AI chat to validate their findings, refined their comments based on AI feedback, and finally fixed the vulnerabilities. Analysis of 28 cleaned challenge submissions from 13 participants shows that the vast majority of interactions began with prompts like "Have I found all vulnerabilities?" or "Find all vulnerabilities," indicating a strong tendency to rely on the AI for comprehensive vulnerability discovery even though the model lacks full contextual awareness of hidden code paths. Developers completed manual reviews more efficiently, 72% of high-experience participants finished each review in 5–10 minutes versus nearly 30% of their less-experienced peers taking over 10 minutes. Across all levels, GenAI served primarily as a second pair of eyes, helping users catch subtle issues that manual inspection alone might miss (e.g., nuanced input-validation flaws), and then proposing remediation code snippets that aligned with secure-coding best practices. However, this pattern also surfaced a risk of over-reliance: by expecting the model to "know everything," some developers accepted its suggestions uncritically, which in practice can lead to hallucinated fixes or overlooked edge-case vulnerabilities.

The study also presents results from the survey taken during the experiment. They reveal a broadly positive yet cautious stance toward GenAI in security workflows. A strong majority agreed that generative AI increases productivity (67%), reduces human error (61%), and enhances vulnerability detection (67%). When asked about the quality of AI feedback compared to manual review, 45% of high-experience developers rated it "good" or "very good," and 57% of low-experience developers found it "good," underscoring broad satisfaction across experience levels. Adoption intent was equally high: 72% of high-experienced developers and 86% of low-experience developers indicated they were "somewhat" or "very likely" to integrate LLM-assisted reviews into their workflow. Yet, trust remained tempered, no

participants endorsed blind confidence, and the dominant mindset was "trust but verify," with 45% of high-experience developers and 57% of low-experience developers falling into this category. Only one high-experience developer (9%) "trusted completely," while 36% of high-experienced developers and 43% of low-experience developers were "not sure," reflecting persistent concerns about model limitations and the necessity of human oversight. These insights suggest developers value GenAI as an efficiency and quality booster but insist on maintaining critical review practices to guard against the model's blind spots.

The consistent "trust but verify" stance we observed underscores the need for transparent model explanations and robust human-in-the-loop safeguards in any production tool in the software development lifecycle.

In future iterations, we plan to integrate deterministic static-analysis results alongside LLM feedback in a multi-agent architecture: one agent will possess ground-truth challenge solutions while another engages developers in a conversational review, better simulating real-life collaboration and surfacing both provable and learned vulnerabilities. We will also enrich the platform with real-time code evaluation and terminal access so that users can compile snippets, execute test cases, and immediately validate AI suggestions within the same interface.

Complementing these platform enhancements, our research will explore safe prompting techniques to shore up security. By embedding explicit security requirements in prompts: such as requiring OWASP-compliant input validation and providing full contextual information, developers can guide the model toward more robust suggestions. Encouraging step-by-step reasoning prompts and having the AI review and improve its own output will foster deeper critical thinking. Assigning a security-savvy persona to the model can further align its responses with best practices. Together, these refinements aim to balance the productivity gains of GenAI with the rigors of secure software development and drive broader developer adoption.

References

1 Sathwik Amburi. Enabling Secure Coding: Exploring GenAI for Developer Training and Education: Survey + Submission Data. https://github.com/Sathwik-Amburi/ascend-pilot-study, May 2025. commit `997596879e8acd1251cef457d19aa50c4e9a547e`.

2 Norbert Baláž, Jaroslav Porubän, Marek Horváth, and Tomáš Kormaník. Using ChatGPT During Implementation of Programs in Education. In André L. Santos and Maria Pinto-Albuquerque, editors, *5th International Computer Programming Education Conference (ICPEC 2024)*, volume 122 of *Open Access Series in Informatics (OASIcs)*, pages 18:1–18:9, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2024.18`.

3 Yannik Bauer, José Paulo Leal, and Ricardo Queirós. Authoring Programming Exercises for Automated Assessment Assisted by Generative AI. In André L. Santos and Maria Pinto-Albuquerque, editors, *5th International Computer Programming Education Conference (ICPEC 2024)*, volume 122 of *Open Access Series in Informatics (OASIcs)*, pages 21:1–21:8, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2024.21`.

4 Lenz Belzner, Thomas Gabor, and Martin Wirsing. Large language model assisted software engineering: Prospects, challenges, and a case study. In Bernhard Steffen, editor, *Bridging the Gap Between AI and Reality*, volume 14380 of *Lecture Notes in Computer Science*, pages 355–374, Cham, Switzerland, 2024. Springer Nature Switzerland. `doi:10.1007/978-3-031-46002-9`.

5 DHS Cybersecurity. Software assurance. Technical report, U.S. Department of Homeland Security, 2012. Accessed: 2025-04-20. URL: https://www.cisa.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf.

6 Tiago Espinha Gasiba, Kaan Oguzhan, Ibrahim Kessba, Ulrike Lechner, and Maria Pinto-Albuquerque. I'm Sorry Dave, I'm Afraid I Can't Fix Your Code: On ChatGPT, CyberSecurity, and Secure Coding. In Ricardo Alexandre Peixoto de Queirós and Mário Paulo Teixeira Pinto, editors, *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASIcs)*, pages 2:1–2:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2023.2.

7 Tiago Carvalho Freitas, Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques. NLP/AI Based Techniques for Programming Exercises Generation. In Ricardo Alexandre Peixoto de Queirós and Mário Paulo Teixeira Pinto, editors, *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASIcs)*, pages 9:1–9:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2023.9.

8 Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu, and Jinfu Chen. Security weaknesses of copilot-generated code in github projects: An empirical study. *ACM Trans. Softw. Eng. Methodol.*, February 2025. doi:10.1145/3716848.

9 Tiago Espinha Gasiba, Andrei-Cristian Iosif, Ibrahim Kessba, Sathwik Amburi, Ulrike Lechner, and Maria Pinto-Albuquerque. May the source be with you: On ChatGPT, cybersecurity, and secure coding. *Information*, 15(9):572, 2024. doi:10.3390/info15090572.

10 Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Sifu - a cybersecurity awareness platform with challenge assessment and intelligent coach. *Cybersecurity*, 3(1):24, 2020. doi:10.1186/s42400-020-00064-4.

11 Sabaat Haroon, Ahmad Faraz Khan, Ahmad Humayun, Waris Gill, Abdul Haddi Amjad, Ali R. Butt, Mohammad Taha Khan, and Muhammad Ali Gulzar. How accurately do large language models understand code?, 2025. arXiv:2504.04372.

12 Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004.

13 Stefan E. Huber, Kristian Kiili, Steve Nebel, Richard M. Ryan, Michael Sailer, and Manuel Ninaus. Leveraging the potential of large language models in education through playful and game-based learning. *Educational Psychology Review*, 36(1):25, 2024. Published online 27 February 2024, accepted 9 February 2024. doi:10.1007/s10648-024-09868-z.

14 International Information System Security Certification Consortium $(ISC)^2$. ISC2 cybersecurity workforce study 2023: How the economy, skills gap and artificial intelligence are challenging the global cybersecurity workforce. Technical report, $(ISC)^2$, 2023. URL: https://media.isc2.org/-/media/Project/ISC2/Main/Media/documents/research/ISC2_Cybersecurity_Workforce_Study_2023.pdf.

15 Mohammed Kharma, Soohyeon Choi, Mohammed AlKhanafseh, and David Mohaisen. Security and quality in LLM-generated code: A multi-language, multi-model analysis, 2025. arXiv:2502.01853.

16 Hao-Ping (Hank) Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *CHI 2025*, April 2025. doi:10.1145/3706598.3713778.

17 Shahar Man. How AI-generated code is unleashing a tsunami of security risks. *Forbes Technology Council*, April 2025. Accessed: 2025-04-20. URL: https://www.forbes.com/councils/forbestechcouncil/2025/04/18/how-ai-generated-code-is-unleashing-a-tsunami-of-security-risks/.

18 Microsoft Corporation. Monaco editor. https://microsoft.github.io/monaco-editor/, 2025. Accessed: 2025-05-04.

19 Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do Users Write More Insecure Code with AI Assistants? In *CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, pages 2785–2799, New York, NY, USA, 2023. ACM. doi:10.1145/3576915.3623157.

20 Ricardo Queirós. Exercisify: An AI-Powered Statement Evaluator. In André L. Santos and Maria Pinto-Albuquerque, editors, *5th International Computer Programming Education Conference (ICPEC 2024)*, volume 122 of *Open Access Series in Informatics (OASIcs)*, pages 19:1–19:6, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2024.19.

21 Ricardo Alexandre Peixoto de Queirós. Integration of a learning playground into a LMS. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*, ITiCSE '22, page 626, New York, NY, USA, 2022. ACM. doi:10.1145/3502717.3532175.

22 June Sallou, Thomas Durieux, and Annibale Panichella. Breaking the silence: the threats of using LLMs in software engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER'24, pages 102–106, New York, NY, USA, 2024. ACM. doi:10.1145/3639476.3639764.

23 Kwan Wei Kevin Tan. Anthropic's CEO says that in 3 to 6 months, AI will be writing 90% of the code software developers were in charge of. *Business Insider*, March 2025. Accessed: 2025-04-20. URL: https://www.businessinsider.com/anthropic-ceo-ai-90-percent-code-3-to-6-months-2025-3.

24 Inc. Veracode. 2025 state of software security: A new view of maturity. Technical report, Veracode, Inc., February 2025. URL: https://www.veracode.com/wp-content/uploads/2025/02/State-of-Software-Security-2025.pdf.

25 Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S. Yu, and Qingsong Wen. Large language models for education: A survey and outlook. *CoRR*, abs/2403.18105, 2024. doi:10.48550/arXiv.2403.18105.