Leiden University
ICT in Business and the Public Sector

Developers' perceived impact of generative AI approaches on code security

Name: Nisar Ahmad Hotak
Student No: s3734234

1st supervisor: Olga Gadyatskaya
2nd supervisor: Nusa Zidaric
3rd supervisor: Arina Kudriavtseva

MASTER'S THESIS

Leiden Institute of Advanced Computer Science
(LIACS) Leiden University
Gorlaeus Gebouw, Einsteinweg 55
2333 CC Leiden
The Netherlands

ABSTRACT

Software development has seen a surge in the usage of generative AI tools for coding leading to an increase in developers' productivity. However, using AI-generated code has led to security concerns due to the potential introduction of vulnerabilities into the code generated by AI. To understand this security concern, it is important to understand how developers perceive and deal with AI-generated code.

This thesis examined developers' perceived impact of generative AI approaches on code security. A survey was conducted with developers who have used AI tools for coding. The research investigated how developers evaluate the security of AI code, the benefits and risks they perceive, and the security best practices they recommend. Furthermore, the study examined the influence of different AI tools and functionalities on developers' confidence in the security of the code produced.

The findings suggest that generative AI code should be used with care, as developer behavior is crucial in preventing security issues. Moreover, full program code generation tools should be used with even more caution than code completion tools. The research also provides some security best practices and code-reviewing factors to consider when using AI-generated code.

Generative AI tools improve coding efficiency and development speed, but they also introduce security risks that developers must carefully manage. Developers perceive AI-generated code as vulnerable, and the importance of reviewing and adhering to security best practices is recognized.

ACKNOWLEDGEMENT

I want to thank Dr. Olga Gadyatskaya, my first supervisor, for her support and guidance throughout this master's thesis. The weekly and biweekly check-ins, along with the feedback and suggestions, have been invaluable in writing this thesis. I am particularly grateful for the extra resources you sometimes recommended such as articles.

I also want to thank Arina Kudriavtseva for her help in this thesis. Your dedication and support, even during vacation periods, have been invaluable. I am grateful for the thorough reviews, guidance, and feedback throughout the research.

Finally, I thank my family for their support and encouragement during this research. Thank you all.

CONTENTS

CHAPTER 1

INTRODUCTION

In this digital era, software is present in every aspect of our daily lives. From smartphones and TVs to critical and essential infrastructure, all of these technologies rely on software. As software systems become increasingly complex and critical to our society, ensuring their security has become paramount. Vulnerabilities in software can lead to data breaches, and system failures, and even bring society to a halt.

In this context of security concerns, the use of generative AI has introduced new dimensions to software development. The software development cycle has swiftly incorporated generative AI. With its ability to automate code generation, generative AI offers both opportunities and challenges.

The use of generative AI in software development using large language models (LLMs) for code generation has increased. These AI-based code generation tools (CGTs) influence developers' productivity [62] and efficiency. However, alongside these benefits, there are concerns about the security implications of AI-generated code. That's why understanding the impact of generative AI and developers' perceptions of these impacts is important.

AI-based code generation can be broadly classified into two distinct approaches. The first approach is code completion tools that come within the editors and display a menu of context-sensitive options [38]. While these tools are capable of generating code snippets, they are not the same as code generators, which require developers to input code, feed it into the LLM, and then return the answer to the programmer [60].

Previous research highlights the limitations and security concerns associated with AI-generated code. LLMs trained on data from programming communities may inadvertently produce code containing security vulnerabilities or outdated practices [39]. Vaithilingam et al. [58] suggest that the accelerated development offered by tools like Copilot might be offset by the time invested in debugging errors introduced by the code generation process. These findings stress the need for caution when using LLM-based code assistants, as they may introduce security vulnerabilities [39, 47, 53].

The role of developer perceptions and behaviors in ensuring code security is also an important issue when dealing with AI-generated code. Despite the focus on security, human actions are often seen as the biggest security risk [48]. The social cultures of software engineers greatly influence the development process [16, 30, 51]. While many developers consider security in their coding practices, there is considerable variation in how security issues are identified and addressed [44].

Due to the increasing adoption of generative AI in different domains of software development, it is important to understand the impact of this technology. This study aims to explore the developers' perceptions and behaviors when working with generative AI. By understanding factors such as perceived benefits, risks, security practices, and confidence levels, this research will contribute to the understanding of AI code and improve integration of the AI-generated code in software development.

1.1 Research focus

The first step in any research is to define the entity examined in the investigation. This entity is called the unit of analysis. The unit of analysis is defined as the "who" or "what" from which data is examined and conclusions are drawn [41]. Since this research focuses on developers' perceived impact of generative AI approaches on code security, the unit of analysis is defined as software developers who have experience working with generative AI tools in the context of code development. Such software developers can be anyone including, but not limited to, students, professional programmers, freelance programmers, hobbyist programmers, and researchers.

1.2 Research questions

Due to the nature of training data, generative AI introduces vulnerabilities in its output. Different approaches can introduce these vulnerabilities, so it is important to understand developers' perceptions and how they address these vulnerabilities. Hence, the main research question is:

How do developers perceive the impact of generative AI approaches on code security?

To address this research problem, the following sub-questions are formulated:

1: How do developers perceive and evaluate the security of code generated by generative AI (GAI)?

- 1.1: How do developers perceive the benefits and risks of using generative AI for code security?
- 1.2: What security best practices do developers recommend when using generative AI code?
- 1.3: What factors do developers consider most important when evaluating the security of code generated by generative AI?
- 1.4: How confident are developers in the security of generative AI code compared to the code they write themselves?

2: What factors influence developers' behavior while using GAI?

3: How does the functionality of different GAIs affect developers' perceptions of security?

- 3.1: Are there differences in how developers perceive the security of code generated by different generative AI tools?
- 3.2: How does the perceived functionality of different AI tools (e.g., code completion vs. full program generation) influence developers' perception of the impact on code security?

1.3 Contribution

Firstly, the study contributes to understanding the potential benefits and risks of using generative AI code. Secondly, it also provides the best security and reviewing practices used by developers when dealing with AI code. Thirdly, the research adds by comprehending the changes in developers' behavior and their confidence in using these generative AI tools. These findings can provide organizations with a better understanding of the risks and advantages associated with generative AI code, and help improve their software security.

1.4 Thesis Structure

The upcoming chapters are organized as follows: The chapter 2 presents a literature review on various approaches, the limitations of generated code, and developers' perceptions when addressing security concerns. Chapter 3 outlines the research methodology, including the survey and questionnaire design and data collection strategies. Chapter 4 presents the survey data results. Chapter 5 builds on these findings, discussing them in relation to previous research, and exploring their implications. This chapter also addresses the study's limitations. Finally, Chapter 6 concludes the study and provides directions for future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Security implications of AI tools

Software development is undergoing a transformation due to the emergence of Large Language Models (LLMs). AI assistants powered by LLMs, like GitHub Copilot, JetBrains AI Assistant, and Visual Studio IntelliCode, have become indispensable tools [50]. According to Klemmer et al. [24], despite widespread concerns regarding the security and quality of AI-generated code, participants actively utilize AI assistants for critical tasks such as code generation, threat modeling, and vulnerability detection. Developers are also using AI assistants for various coding tasks, such as writing, testing, debugging, reviewing, and documenting code [54].

Even though the usage of AI assistants has increased, previous research has revealed their security implications. A study by Pearce et al. [39] found that GitHub Copilot's reliance on unvetted code data led to 40% of its generated code containing vulnerabilities. Developers using AI assistants for programming tasks in Python, JavaScript, and C were less likely to write secure code, despite their perception of having improved their security [40]. Additionally, a common issue with LLMs is their tendency to generate inaccurate or hallucinatory content, especially in the context of code generation that involves complex dependencies. Hallucinations are a major problem for the most advanced generative LLMs [61]. The study by Klemmer et al. [24] revealed that software professionals exhibit a general mistrust of AI suggestions, often verifying them similarly to human-generated code. Moreoever, participants anticipate that improvements in AI capabilities will lead to increased reliance on these tools for security tasks in the future.

Considering the security challenges faced by AI assistants, developers should maintain a high level of awareness and vigilance [39]. The research by Klemmer et al. [24] also emphasizes the need for software professionals to critically evaluate AI suggestions and calls for AI developers to improve the security and ethical considerations of their tools. On the other hand, according to Sandoval et al. [47], AI-assisted users produced critical security bugs at a rate no more than 10% higher than the control group (non-assisted users). The study seems to suggest that their use does not introduce significant new security risks compared to traditional programming methods, at least

for the low-level C programming tasks examined in this study. This suggest that the tools may be used without too much critical evaluation.

2.2 Approaches in AI-based code generation

Automated code generation (ACG) is growing in importance, primarily due to the use of machine learning (ML) and artificial intelligence (AI). There are many techniques for automated code generation including rule-based systems, template-based methods, ML, natural language processing (NLP), rec urrent neural networks (RNNs), and evolutionary algorithms (EAs) [36]. Rule-based code generation uses established guidelines and examples to create code based on detailed instructions or everyday language [26]. Template-based code generation offers a template-based approach, providing developers with ready-made code structures that can be tailored to their needs [36]. ML-based code generation techniques often train models on vast amounts of code to acquire knowledge of patterns, connections, and standard coding styles [15]. NLP-based systems can be used to create code from descriptions in plain language [18]. RNNs can identify patterns and relationships within sequences of code and create code segments or entire functions. EAs are a kind of machine learning algorithm that can create code by repeatedly changing and improving existing code [36].

Code completion tools, according to Omar et al. [38], are those that come with most editors and display a floating menu of context-sensitive variables, fields, methods, types, and other code snippets. Among the popular code completion solutions are JetBrains integrated development environment's built-in code completion and IntelliSense in Visual Studio Code. These tools are not the same as code generators, even though they can produce code snippets. Code-generating tools (CGTs) use the power of large language models (LLMs) by directly feeding the programmer's input into the LLM. The programmer's development environment receives the resulting code. This is different from code completion tools, which make suggestions based on limited context [60]. The number of artificial intelligence (AI) based code-generating tools has rapidly increased as a result of advances in deep learning and natural language processing (NLP) [1].

Integrated Development Environments (IDEs) and text editors can incorporate code completion tools, according to Asare et al. [1]. Examples include Github Copilot [13], IntelliCode [55], and Tabnine [56] among others. On the other hand, code-generating tools exist as independent language models that are not integrated into the text editor or IDEs [1], among them are Codex [8], CodeGen [35], and codeBERT [9]. It is important to understand whether the impact of these different approaches to AI code generation is perceived differently by developers. In our research, we explore this issue by comparing how developers perceive the risk involved with full program generation tools compared to code completion tools.

2.3 Limitation of generated code

Many tasks related to natural language processing require language models [7] and large language models (LLMs) are trained on enormous datasets [42]. Regrettably, when large Language Models

(LLMs) are trained using data extracted from programming communities, like StackOverflow, they acquire the capability to produce code containing security vulnerabilities or defects [39]. Sandoval et al. [47] examine the time when the Message Digest Algorithm 5 (MD5) was a prevalent method for securing sensitive data, including passwords. Since MD5 contains a cryptographic flaw, it should no longer be utilized. However, the persistence of code examples utilizing MD5 within publicly accessible repositories presents a potential challenge. Large language models (LLMs) trained on such data may inaccurately recommend the insecure MD5 algorithm for password-hashing applications.

According to Pearce et al. [39], among currently available models of its kind, GitHub Copilot stands out as potentially possessing the most extensive dataset and the most advanced capabilities. Github Copilot leverages a large language model that is trained on public and open-source code [8] that contains unsafe coding patterns. This means that code that has been synthesized may have these unwanted patterns [39]. According to Vaithilingam et al. [58], Copilot creates code far more quickly than typing it by hand or reusing code from external sources. But the authors also postulate that this efficiency gain might be offset by the potential introduction of errors within the generated code, meaning that time saved by not having to write code may instead need to be used troubleshooting Copilot-generated code. New findings indicate that completions generated by large language models (LLMs) might have significant security flaws [39, 53]. This implies that even though using LLM-based code assistants can enhance developer productivity, it is advisable to exercise caution or refrain from their use entirely due to potential security risks [47].

Looking at the functional and qualitative aspects of code generated by AI tools such as Bard and ChatGPT, the author found that these tools produce code that works just as well and is of the same quality as code that can be found online for specific coding problems. Also, the large language models' (LLMs) models follow coding standards even when creating complex code. However, humans are needed to guide LLMs in the right direction, especially when it comes to giving them the right software requirements [57]. The research of Autumn et al. [3] looks at the quality and consistency of code generated by ChatGPT. While ChatGPT consistently generates good code, it occasionally also introduces errors, similar to human programmers. This means code review remains essential before integrating the AI-generated code into the codebase. Due to the inherent nature of vulnerabilities in code as stated in this review, understanding how developers' confidence changes when using these tools and what factors influence this confidence is crucial for better tackling these problems.

2.4 Developer perceptions and security

AI-based code generation tools as mentioned above inherently contain vulnerabilities in their code. Kholoosi et al. [22] focused on the perception of security professionals on using ChatGPT and found that practitioners most often discussed vulnerability detection as a key software security task. While ChatGPT can be helpful for secondary tasks like writing vulnerability reports, security professionals should always carefully review and verify its content to guarantee accuracy and completeness. Nam et al. [34] examined the developers' perception by using a specific tool built into the AI tools for code understanding called generation-based information support with LLM

Technology (GILT). Participants acknowledge GILT's capability to integrate their code as context; however, they also mentioned that creating an effective prompt remains a challenge. Furthermore, the way participants used GILT was shaped by their learning preferences and their experience with other AI tools. Barke et al. [4] researched developers' interaction and behavior when using code-generating models specifically Github Copilot and found that developers interact with two different modes. In acceleration mode, the programmer has a specific goal and uses AI tools such as Copilot to achieve it faster. On the other hand, the programmer is uncertain of the best approach and uses Copilot to investigate alternatives in the exploration mode.

Even though there is a lot of focus on making things more secure, people's actions are often seen as the biggest security risk [48]. It is crucial to understand that software engineers interact with one another and have rich, distinctive social cultures [30, 51]. These cultures influence and mold the process of developing software [16]. Programmers frequently bear responsibility for neglecting security considerations [21, 33]. Rauf et al. [44] mention in their experiment that even though security was not specifically mentioned, a good portion of developers (56%) wrote code securely, while nearly half (44%) said they thought about security. These results highlight the variability in developers' approaches to identifying, discussing, and addressing security issues. Therefore, understanding the perception of developers when dealing with code security is crucial even when dealing with the security of AI-generated code.

The evaluation of the quality of software is largely based on human judgment of the source code [11]. Developers use manual reviews to ensure the quality of the software. However, this consumes a lot of time and errors can occur [17]. There are several static analysis tools available that can automatically detect software bugs [12]. These tools are superior to manual audits because they are faster, allowing for more frequent evaluations of programs. Additionally, these tools incorporate security knowledge, eliminating the need for the operator to possess the same level of security expertise as a human auditor [31]. However, recent studies suggest that programmers do not fully utilize the benefits of these tools [5, 20]. According to Johnson et al. [20], the main reasons are too many false positives adding to the developer workload, poor presentation or analysis of bugs, and not providing the context of what the problem is, why the problem is there and how the developer should try to resolve this problem. Developers' behavior plays a crucial role in their adoption of AI tools in their coding. Our research tries to understand how developer behavior changes when using AI-generated code and what factors are considered important by developers when integrating code generated by AI tools.

CHAPTER 3

METHODOLOGY

3.1 Survey

Surveys are commonly used to collect data from a group of individuals, and the collected data is usually smaller than the total population being studied [49]. In this research, the target population of software developers is large and geographically dispersed. Web-based surveys offer an efficient way to collect data from a large and diverse group of developers, allowing for a better representation of perspectives. The use of surveys is well established within this research field, as evidenced by their application in studies such as Ziegler et al. [62], Kononenko et al. [25], and Assal et al. [2]. Quick data analysis is another benefit of using a survey [10]. To create the survey, the Qualtrics software platform was used.

3.1.1 Survey design

The survey began with an introductory page of information and ended with a thank you message and the survey link, encouraging developers to share it further if they found it interesting. Those interested in the results could leave their email to receive the survey's findings. The survey was made with help from previous studies. [28, 62, 25, 16, 2]

The survey contains the following parts: 1) Introduction, 2) Demographics, 3) Developers' competence, 4) Generative AI code benefits & risks, 5) Developers' behavior, 6) Reviewing generative AI code, 7) Generative AI tools and approaches, 8) Self-coding vs generative AI code.

3.1.2 Research questions mapping

The first research question (RQ1.1) considers the benefits and risks, RQ1.2 asks about the security best practices when using AI code, RQ1.3 focuses on security factors when using AI-generated code, RQ1.4 considers the code confidence of developers. RQ2 presents the behavior changes when using AI code. RQ3.1 the perception differences in AI tools, lastly RQ3.2 tries to understand

the different AI approaches. These research questions are then mapped to the survey questions in Table 3.1. All survey questions can be found in Appendix A.

Introduction

The landing page of the survey is the information sheet that provides the surveyors with the research topic, the estimated amount of time to complete the survey, and the types of questions included in the survey. The introduction also explains the anonymity and voluntary participation of the respondents. Furthermore, the page includes the researchers' names and email addresses, as well as asking for the participants' consent.

Demographics

The second part of the survey focuses on the demographics of the respondents. The questions ask about the current occupation of the participants, their level of coding experience, and the primary languages the developers are familiar with. In addition to gathering basic demographics, these questions act as a way to categorize participants. This categorization could be useful for future analysis, as developer perceptions of generative AI and code security might differ depending on the proficiency levels of secure coding, current occupation, and different types of language used with generative AI tools.

Developers' competence

The third part of the survey asks participants to self-report their proficiency in writing secure code without the assistance of AI tools. This establishes a baseline for their secure coding knowledge. The survey then asks them to gauge their perceived proficiency in secure coding when using generative AI tools. By comparing these two sets of responses, we can explore whether using generative AI tools influences developers' confidence in their ability to write secure code.

Generative AI code benefits & risks

The survey asks about the perceived benefits developers see in using these AI tools. This includes advantages like increased efficiency, faster development cycles, or improved detection of vulnerabilities. On the other hand, the survey focuses on the potential security concerns developers have about generative AI. These involve issues like introducing new vulnerabilities, Privacy, and ethical concerns in AI-generated solutions, or challenges with integrating these tools with existing security practices. This section ends by asking developers about the challenges they anticipate with the security of code generated by AI tools.

Developers' behavior:

This section starts with a 5-point Likert scale about potential changes in the behaviors of developers when using generative AI tools. The changes included: time spent writing code, time spent on code review, cautious deployment, understanding the logic of the code, and documentation and referencing of the code. The question compares all of these different aspects of the code created by

Research questions:	**Survey questions:**
RQ1.1: How do developers perceive the benefits and risks of using generative AI for code security?	Q6: In your opinion, what are the perceived benefits of using generative AI in code security? Q7: In your opinion, what are the perceived risks of using generative AI in code security? Q8: Five years from now, which of the following challenges related to security do you foresee for AI-generated code?
RQ1.2: What security best practices do developers recommend when using generative AI code?	Q11: Which security best practices do you recommend developers follow when using generative AI code (GAI)? Q12: Which security practices do you use when coding using generative AI tools?
RQ1.3: What factors do developers consider most important when evaluating the security of code generated by generative AI?	Q13: When reviewing generative AI code, how important do you consider the following security factors? Q14: In your opinion, what other security factors are important when reviewing generative AI code?
RQ1.4: How confident are developers in the security of generative AI code compared to the code they write themselves?	Q19: How confident are you in the security of code generated by generative AI compared to code written by yourself?
RQ2: What factors influence developers' behavior while using generative AI?	Q9: The following statements represent changes in developers' behaviors. Q10: In your opinion, what other changes in behaviors occur?
RQ3.1: Are there differences in how developers perceive the security of code generated by different generative AI tools?	Q15: Do you believe the security of code generated by different generative AI tools varies significantly? Q15A: Which of the following factors do you believe contribute most to the differences in security between generative AI tools? Q15B: In your opinion, what other factors contribute to the differences in security between different generative AI tools? Q16: Which generative AI tools do you use for coding? Q17: How confident are you in the security of code generated by the selected Generative AI tools?
RQ3.2: How does the perceived functionality of different AI tools (e.g., code completion vs. full program generation) influence developers' perception of the impact on code security?	Q18: Code completion vs. Full program generation: "There is a higher risk of security vulnerabilities in code generated by full program creation tools compared to code completion tools."

Table 3.1: Mapped research questions to survey questions

generative AI tools with the code produced by the developers themselves. The section ends with a question about any other behavioral change that may occur.

Security practices

In line with the sub-questions of the research, the survey asks developers to identify the best practices when working with generative AI code. The survey provides participants with multiple-choice options and field text. This is followed by a question that asks about the security practices that developers themselves use.

Security factors

The survey asks about the review of generative AI code by allowing participants to rank the answers provided from 1 (most important) to 5 (least important). The survey continues with an open question about other security factors that may be important in reviewing AI-generated code.

Generative AI tools and approaches

This section asks whether developers believe the security of the generated code differs based on the specific AI tool used. If the answer is yes, then it is followed by a multiple-choice question about the main reasons for these security differences. Furthermore, the survey asks which generative AI tools developers use for coding. The selected answers are displayed on a 5-point Likert scale to show how confident developers are in the security aspects of the selected tools. This section ends with a question on what developers perceive as a higher risk: code completion or full-program generation.

Self-coding vs generative AI coding

The last section compares the confidence developers have in the code security of code written by themselves with the code generated by AI tools.

3.1.3 Open-ended question

In addition to the quantitative data from the closed-ended survey questions, qualitative data was also gathered using open-ended questions. The open-ended questions in the survey are Q10, Q12, Q14 and Q15B. We used document analysis to code the data and identify the themes, patterns, and relationships in these responses.

3.1.4 Pilot test

The survey was run for the first time as a pilot test with 5 participants. The participants were selected through the author's personal network. This pilot test allowed us to assess the clarity, flow, and overall effectiveness of the questions. Feedback collected from participants was used to refine the survey questions.

3.1.5 Survey distribution

The survey was distributed within the personal network of the author. It was also shared as a LinkedIn post on the personal page of the author and on developers' groups on LinkedIn. During conferences, QR codes of the survey were distributed within the developer community. In addition, responses were also collected by mail and direct messages on LinkedIn to prospective participants.

3.1.6 Ethical consideration

To ensure the research is conducted in an ethical and fair manner, the participants were informed that their participation was voluntary. They could leave at any moment during the survey. The responses were all anonymized. It was stated that the collected data would only be used for research purposes and to write the master thesis.

CHAPTER 4

RESULTS

A total of 105 participants completed the survey out of 189 participants with a response rate of 55.55%. The survey was run from May 19, 2024, to 13 July, 2024 using Qualtrics software. In total 196 respondents had completed the survey, but the software indicated 10 responses as potential bots due to a low captcha score. Out of the 10 responses identified as potential bots, only those that had relevant answers and completed the survey with a minimum of 5 minutes or more were selected to be part of the responses. For analysis purposes, the responses that had progress of 100% in Qualtrics were considered. Out of the total responses of 196, the completed responses of 105 were considered. The remaining 91 incomplete responses were not part of the analysis for this research.

4.1 Respondents

The first demographic question was about the current occupation of the respondents. As shown in Table 4.1, the highest percentage of participants were professional programmers (50%) and students (24%).

Current Occupation	**Percentage**
Professional Programmer	50%
Student	24%
Researcher	8%
Other:	8%
Hobbyist Programmer	6%
Freelancer Programmer	5%
Total	**100%**

Table 4.1: Responses to (Q1) Which of the following best describes your current occupation?

Figure 4.1 represents years of professional experience in software development. The largest group (28%) consisted of developers with 3-5 years of experience, followed by 0-2 years(25%), students or learners (16%), 6-8 years (15%), 9-11 years (7%), and more than 11 years were 9%. More than 59% of the total participants had 3 or more years of professional experience. This shows an experienced pool of participants.

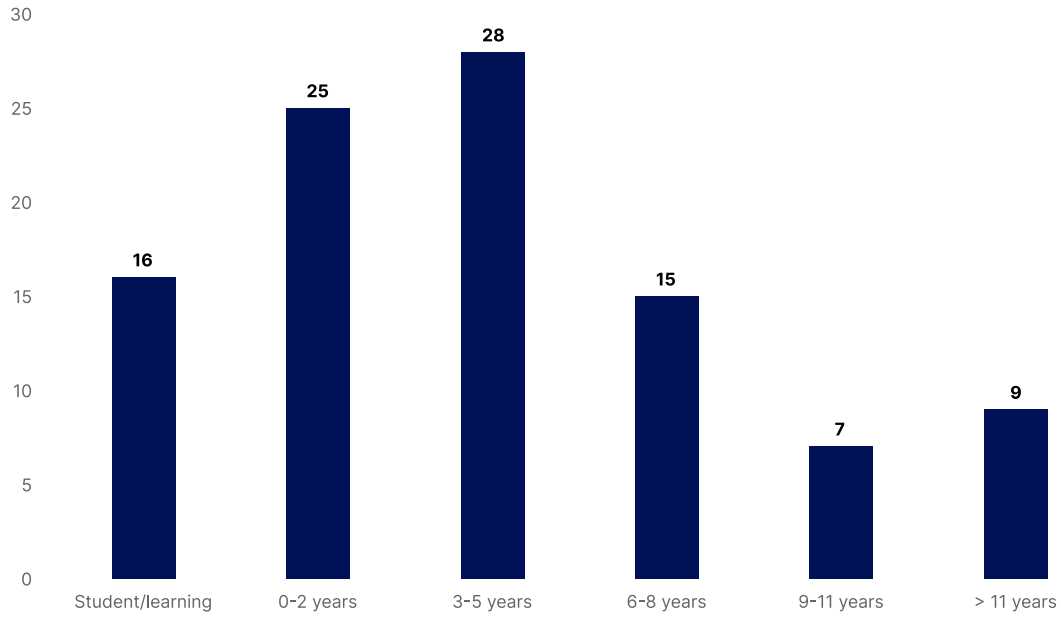

Figure 4.1: Responses to (Q2) How many years of experience do you have in software development?

The types of primary programming languages used by developers are represented in Figure 4.2. The main languages used by the participants were Python (23.6%), Javascript (20.9%), Typescript (13%), and Java (9.1%).

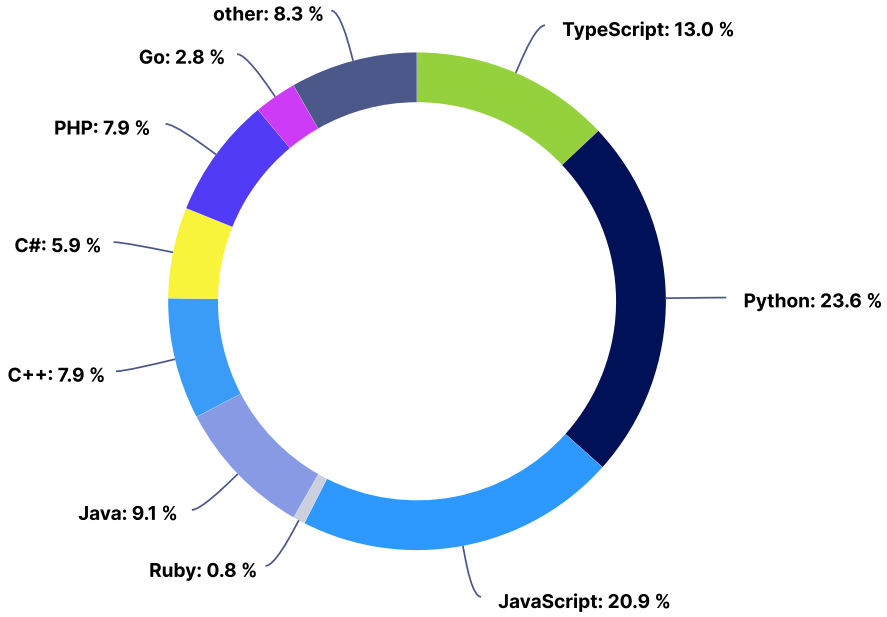

Figure 4.2: Responses to (Q3) What programming languages do you primarily use?

Figure 4.3 illustrates the distribution of participants' proficiency levels in writing secure code without using AI tools. The data showed that 56.19% of the respondents are intermediate, 27.62% are advanced, and 16.19% are beginners.

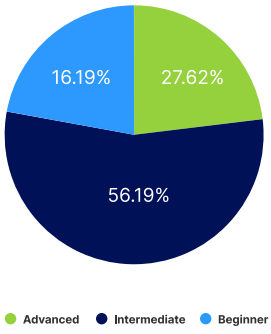

Figure 4.3: Responses to (Q4) How proficient are you at writing secure code Without AI tools in the language you use most?

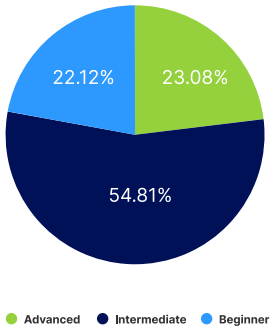

Figure 4.4: Responses to (Q5) How proficient are you at writing secure code With AI tools in the language you use most?

Figure 4.4 also shows the proficiency levels of writing secure code, but now using AI. More than half (54.81%) considered themselves as intermediate, 23.08% as advanced, and 22.12% as beginner.

4.2 Benefits and Risks

This section contains responses about the pros and cons of using generative AI. Figure 4.5 summarizes the responses to Q6 'What are the perceived benefits of using generative AI in code security?'. A significant portion of respondents (82/78.09%) identified faster development as a key advantage. Furthermore, 62.96% (68) of the participants viewed increased efficiency as a major benefit. Around 3.08% (4) participants had mixed answers to the types of benefits presented, including increased security awareness, standardization of code implementation, and easier development due to AI refactored code.

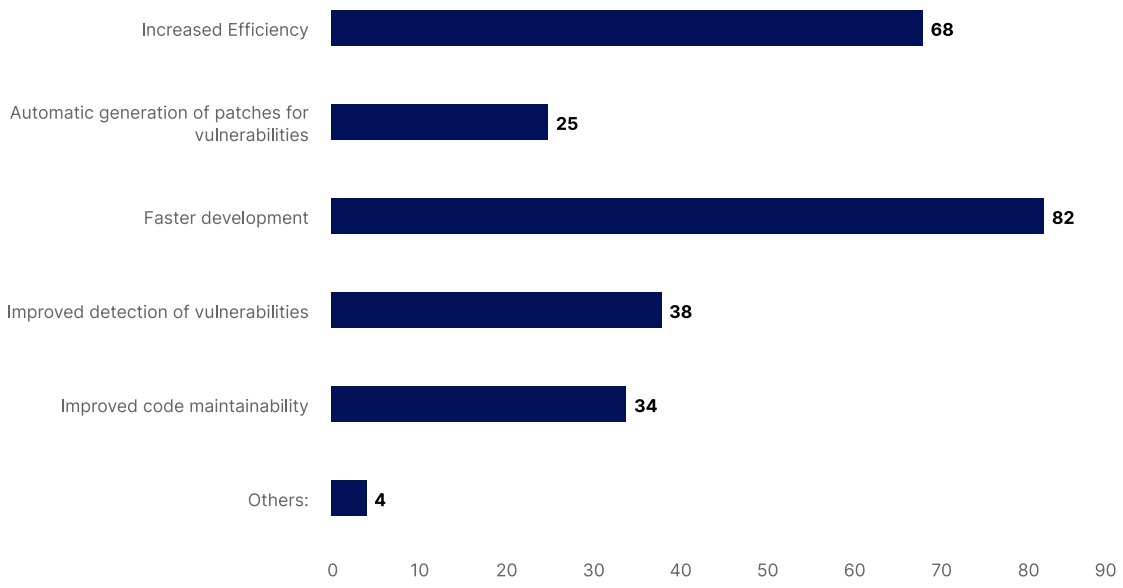

Figure 4.5: Responses to (Q6) In your opinion, what are the perceived benefits of using generative AI in code security?

Cross-tabulation analysis was used to compare developers' professional experience with the benefits selected as shown in Figure 4.6. Faster development was favored by 88.23% of students (15/17 participants), 75.86% of participants with 3 to 5 years of experience (22/29 participants), and 65.38% with 0 to 2 years (17/26 participants). Increase efficiency was selected by 88.23% students (15/17 participants), 81.25% of participants with 6 to 8 years of experience (13/16 participants), and 61.53% with 0 to 2 years (16/26 participants). In both of these benefits, students/learners had the highest percentage of respondents.

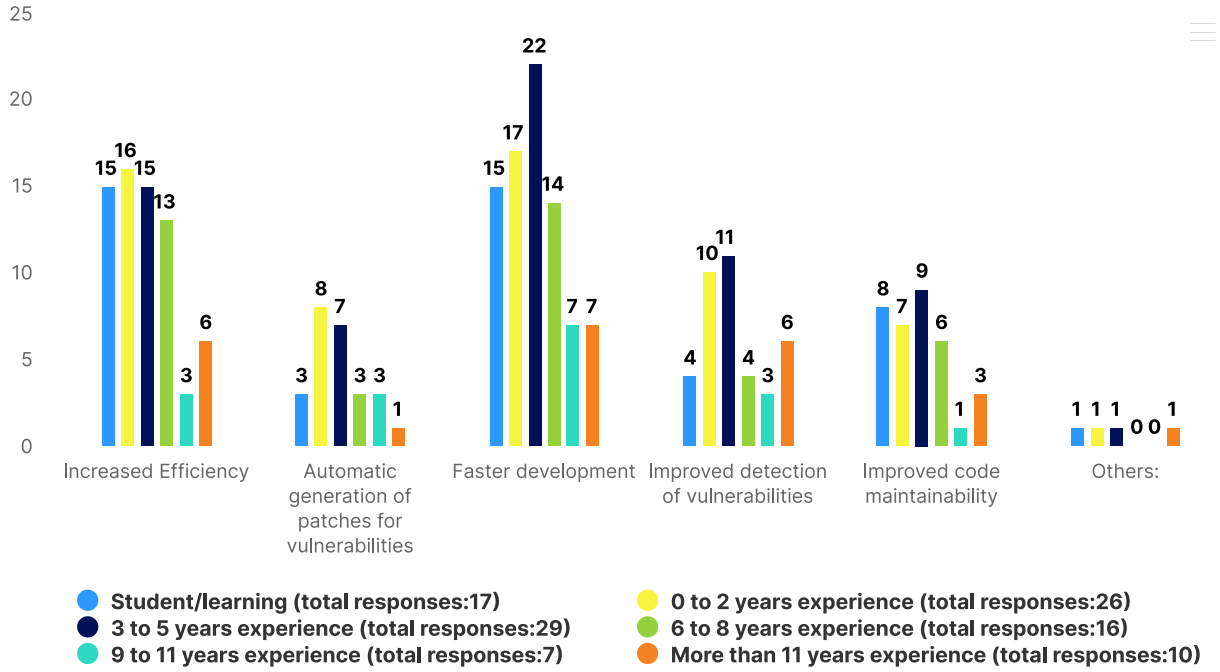

Figure 4.6: Perceived benefits of generative AI with developers' experience

As shown in Figure 4.7, about 64.76% of the participants believed that overreliance on AI without human oversight was the greatest risk. Additionally, the other important perceived risks were: likely to generate code with similar vulnerabilities (58.09%), introduction of new vulnerabilities (51.42%), and privacy and ethical concerns in AI-generated solutions (47.61%). Some of the participants (3.08%) suggested additional risks associated with the use of generative AI code. The main risks were the possibility of incorporating expired or conflicting libraries and the potential of hacking generative AI tools, which could lead to bugs in many pieces of software.

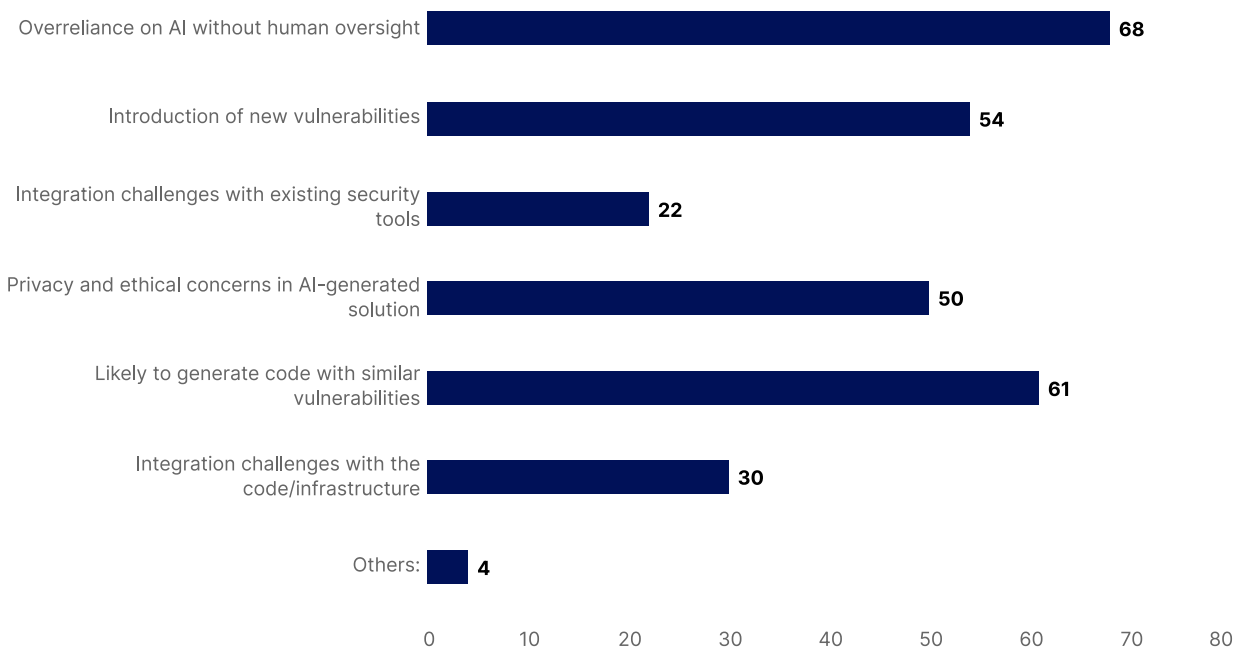

Figure 4.7: Responses to (Q7) In your opinion, what are the perceived risks of using generative AI in code security?

Figure 4.8 represents the answer to (Q8) ”Five years from now, which of the following challenges related to security do you foresee for AI-generated code?”. Unintended biases in generating specific code (56.31%), privacy and ethical concerns in AI-generated code (52.42%), and lack of transparency and auditability in AI-generated code (51.45%) were considered the main challenges by participants. A small portion of participants (4.85%) added what other risks may be present 5 years from now. The main risks were the high risk of similar code with vulnerabilities, less experienced developers, and reliance on AI.

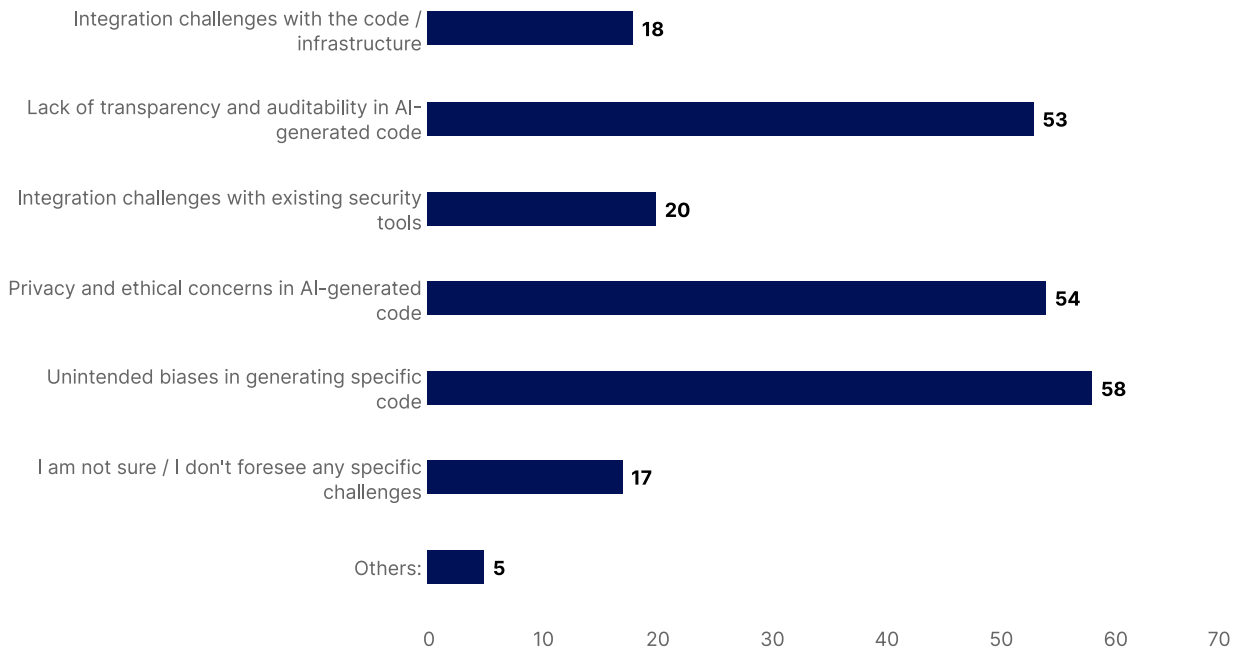

Figure 4.8: Responses to (Q8) Five years from now, which of the following challenges related to security do you foresee for AI-generated code?

4.3 Behavioral changes

Figure 4.9 shows the behavioral changes when using generative AI code compared to code written by developers themselves without AI assistance.

4.3.1 Time spend

About (61.16%) of participants agreed that using generative AI tools to write code saves time compared to writing it themselves. 12.62% were neutral participants. Almost 26.21% replied that the time spent on coding using generative AI tools had not decreased.

4.3.2 Deploying Code

More than 60% of participants were more cautious about deploying generative AI code to production than deploying their own code. Around 26.47% remained neutral on this aspect, and a very small percentage (12.74%) reported not being cautious about deploying AI-generated code to production.

4.3.3 Security review

Only 44.11% of the participants agreed to spend more time reviewing AI-generated code, while nearly 35% of respondents were neutral.. Approximately one-fifth (20.58%) of the participants disagreed on spending more time on security reviews.

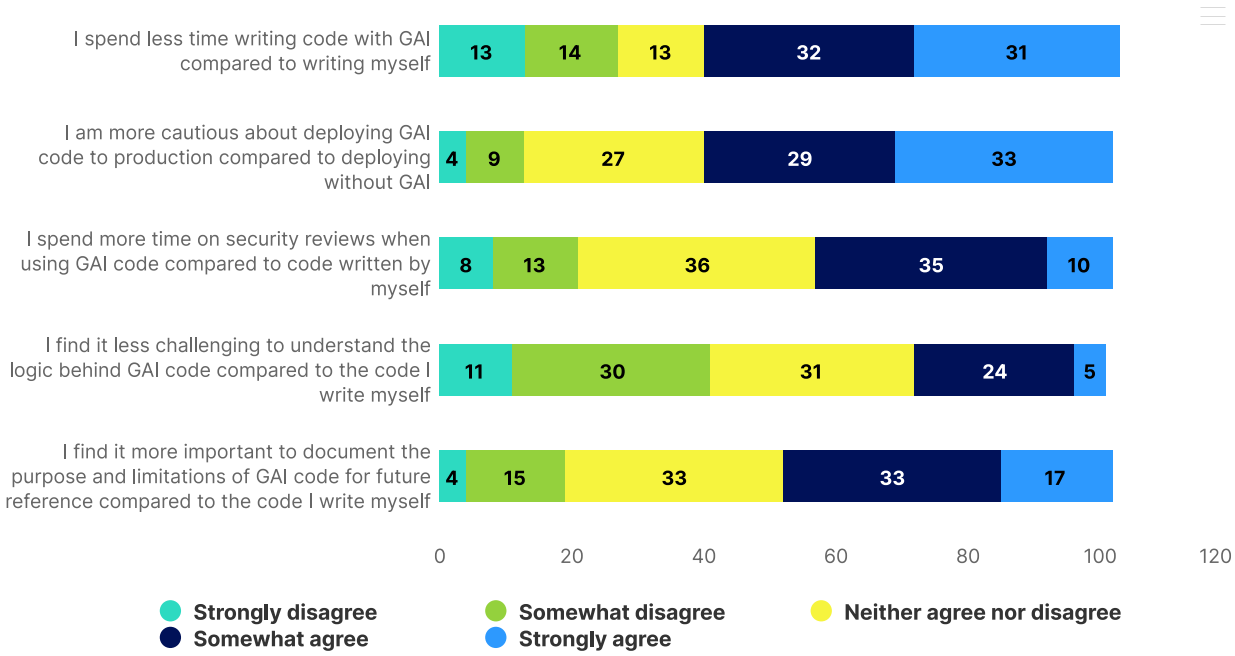

Figure 4.9: Responses to (Q9) The following statements represent changes in developers' Behaviors

4.3.4 Understanding logic

Understanding the logic behind generative AI code proved to be a challenge for a significant portion of the respondents (40.59%) who disagreed with finding it easier than understanding their own code. Around a third (30.69%) of the responses neither agree nor disagree. While 28.71% found the logic behind AI-generated code less challenging to grasp.

4.3.5 Documentation

Around 49.01% of participants found it more important to document the purpose and limitations of generative AI code for future reference. A neutral response was provided by around one-third (32.35%) of participants. A small percentage (18.62%) of the participants disagreed.

Other behavior changes

Around a quarter of responses (23.81%) provided an answer to the open question "In your opinion, what other changes in behavior occur?". As mentioned above, document analysis was used to

derive themes from these responses.

More debug time: One of the behavior changes highlighted by respondents was the extra time spent on debugging errors in AI-generated code. One participant stated: ”Code writing time has been reduced but sometimes generative code costs more time on debugging for false positive code generation.”

More dependent on generative AI solutions: The participants had become reliant on generative AI code. Survey responses suggested this could lead to a weakening of core coding skills. Some developers expressed feeling lazier and less inclined to brainstorm solutions themselves, relying more on generative AI tools to generate code.

Alternative security review approaches: Respondents reported a reduction in consulting online resources like Stack Overflow. This has led to a reduction in code writing time. Alternatively, developers mentioned a more iterative approach to improving code security instead of doing extensive testing on AI-generated code.

4.4 Development time reduction

Cross-tabulation was used to understand if the reduction in time spent on coding was perceived differently by the different proficiency levels (beginner, intermediate, advanced) of coding using AI. As shown in Figure 4.10 almost 69.56% of advanced-level user believed their time spent on coding has decreased when using AI tools. Around 63.15% of intermediate-level participants and 50% of beginners thought their coding time had decreased.

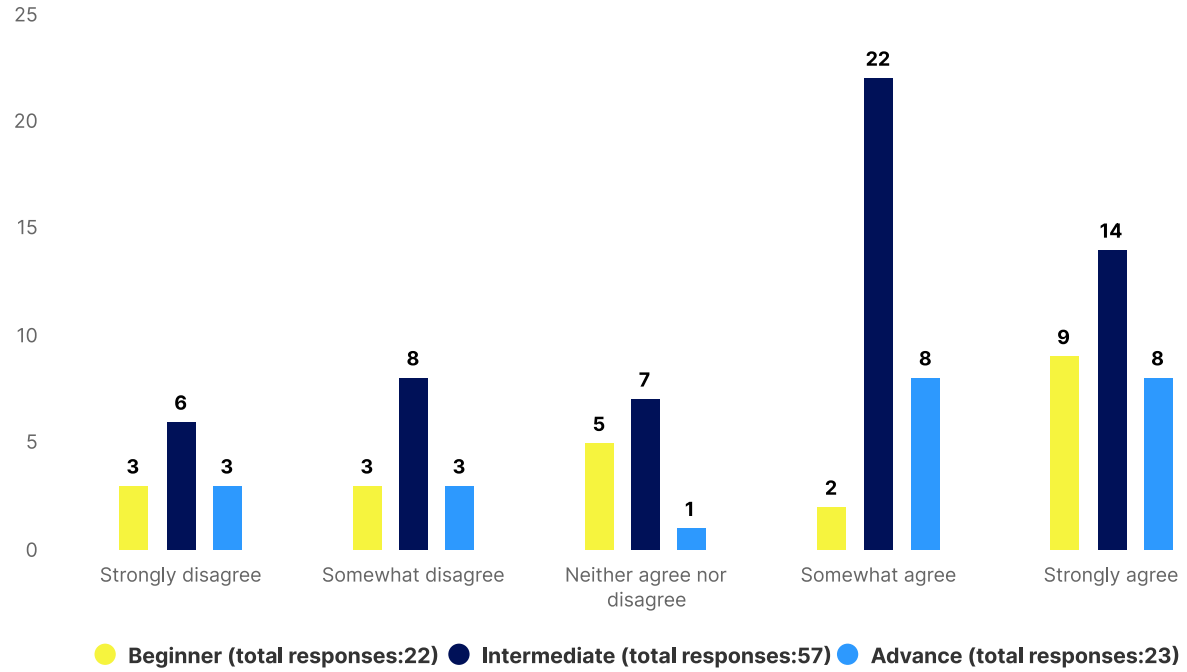

Figure 4.10: Spend less time using generative AI compared to writing myself

Similar to the above results we used cross-tabulation by comparing faster development benefit with the proficiency levels using AI. Around 91.66% of advanced-level respondents selected faster development as the main benefit as displayed in Figure 4.11. Intermediate-level developers (77.19%) and beginners (69.56%) thought faster development was the main benefit of using AI generative tools. The higher the level of proficiency using AI tools, the more participants believed their software development time had decreased.

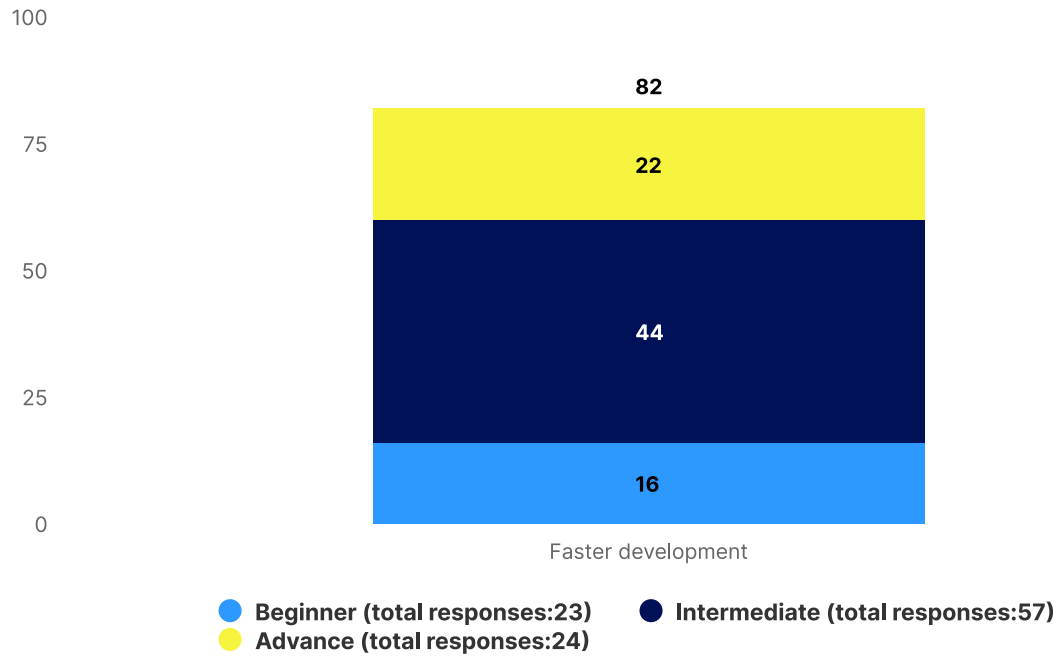

Figure 4.11: Faster development compared with the proficiency level using AI tools

4.5 Security practices

Figure 4.12 represents the security best practices recommended by developers to follow when incorporating AI-generated code. Most of the participants (82%) believed that a thorough review and understanding of all AI-generated code before integration is the most important. In addition, 68% of the respondents recommended not entering sensitive data or personal information. Furthermore, 62% of the participants mentioned using active code scanning tools to identify vulnerabilities as one of the best security practices.

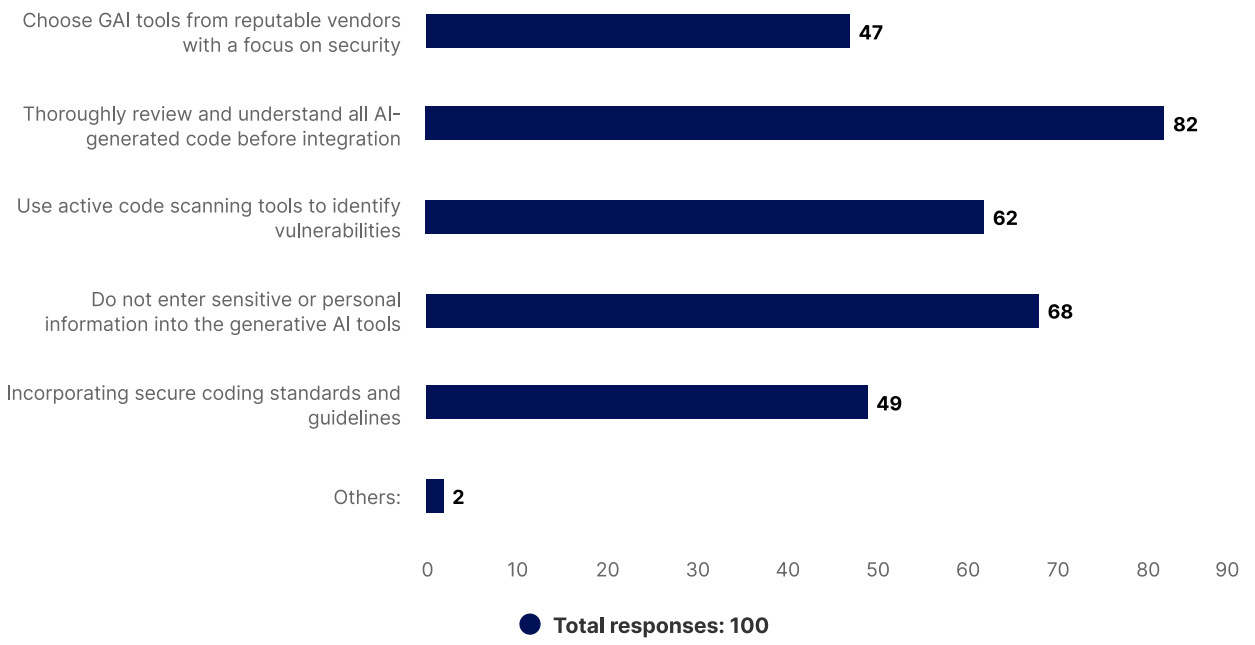

Figure 4.12: Responses to (Q11) Which security best practices do you recommend developers follow when using generative AI code (GAI)?

More than a quarter (29.52%) of the participants answered the open-ended question about what security practices they use themselves when coding with generative AI tools. These security practices are discussed below.

4.5.1 Manual review

Some respondents (22.58%) emphasized thorough reviewing of AI-generated code. This includes manual review to ensure the correctness and understanding of the code.

4.5.2 Data privacy

Many participants (32.25%) were also focused on not sharing sensitive information with AI tools to preserve privacy and prevent data leakage. There was an emphasis on sanitizing inputs and avoiding the use of real credentials or identifiable data.

4.5.3 Existing security practices

Some respondents (22.58%) focused on adhering to established best practices for secure software development [14]. In addition, using tools such as SonarQube [1] for code scanning, identifying security vulnerabilities, and code smells as part of the build process.

[1]SonarQube is an open-source platform developed by SonarSource that automatically reviews code for quality issues. It uses static analysis to identify bugs and code smells in 29 programming languages.

4.5.4 Code restructuring and ethics

The remaining respondents (22.59%) highlighted several points, including the importance of restructuring the code for better maintainability and understanding before integrating the code into the codebase and considering the ethics and validity of data use, especially in data science roles.

4.6 Reviewing

Figure 4.13 shows security factors that are ranked from 1 (most important) to 5 (least important) when reviewing generative AI code. Most of the respondents (50%) considered the clarity and understandability of the code to be the most important factor. In addition, alignment of the generated code with security best practices was chosen by 29.59% of participants. Moreover, the absence of known vulnerabilities in the generated code was chosen by 31.63% and 28.57% participants, respectively. On the other hand, reputation and security track record of generative AI tools was considered as the least important factor by 50% of the participants.

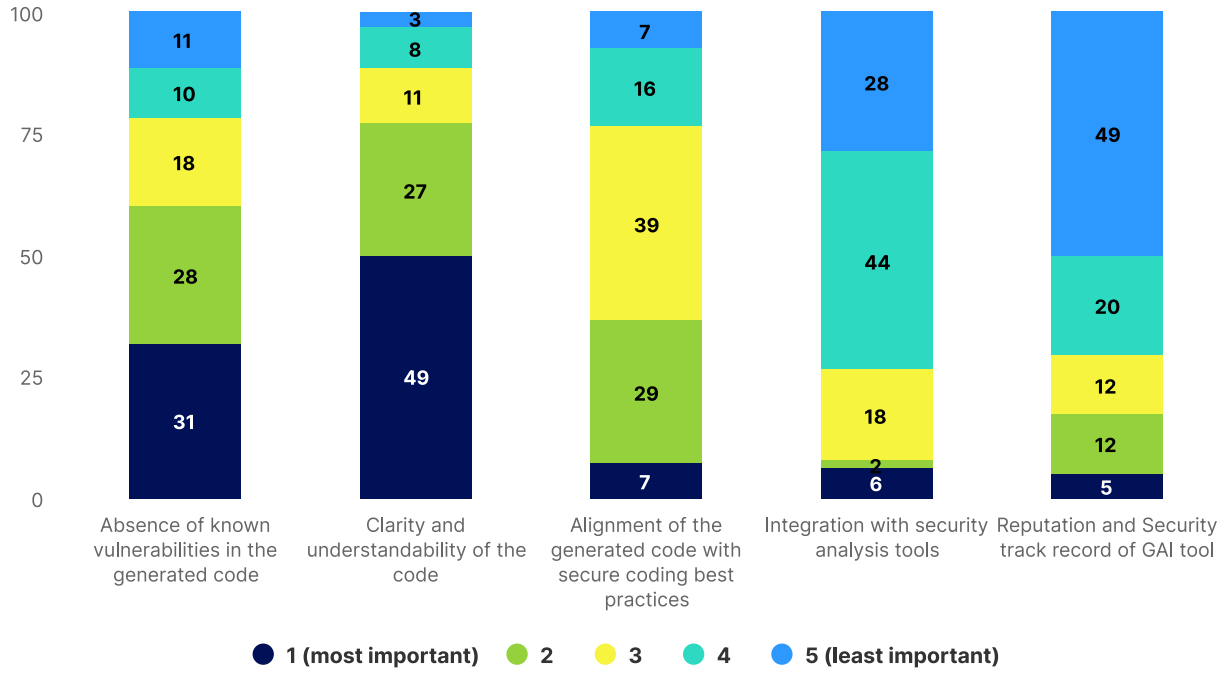

Figure 4.13: Responses to (Q13) When reviewing generative AI code, how important do you consider the following security factors : (Please rank from 1 (most important) to 5 (least important))

Around 20.40% of the participants also responded to the open question asked about what other security factors are important when reviewing generative AI code. For these participants, the following factors were important:

4.6.1 Sensitive information protection

One important idea was the importance of protecting and checking confidential data when reviewing like API keys, IDs, and personal information when using generative AI tools. One respondent said: ”Secrets like API keys and IDs must not be shared with the GAI provider”.

4.6.2 Following best practices

Some of the responses focused on the need to check that the AI-generated code followed the best security practices. For instance, input validation, proper error handling, and avoiding insecure coding patterns.

4.6.3 Language specificity

Some responses suggested that generative AI might be less suitable for certain programming languages with unique characteristics. Solidity [2] is mentioned as an example. One respondent said: ”Consider Solidity, it is less likely to give an optimal code for it...”

4.6.4 Manual testing

The need for proper manual code testing procedures to validate the generated code was another important theme. The code should be updated to reduce the risk of backdoors or newly discovered vulnerabilities.

4.7 Generative AI tools differences

A total of 49.51% of respondents answered yes when asked do they believe the security of code generated by different AI tools varies significantly as shown in Figure 4.14. Around 35.92% were unsure if there were any significant differences in security between different AI tools, while only 14.56% answered No.

[2]Solidity, a programming language designed for creating smart contracts, is widely used on blockchain platforms like Ethereum. It operates under the GNU General Public License v3.0.

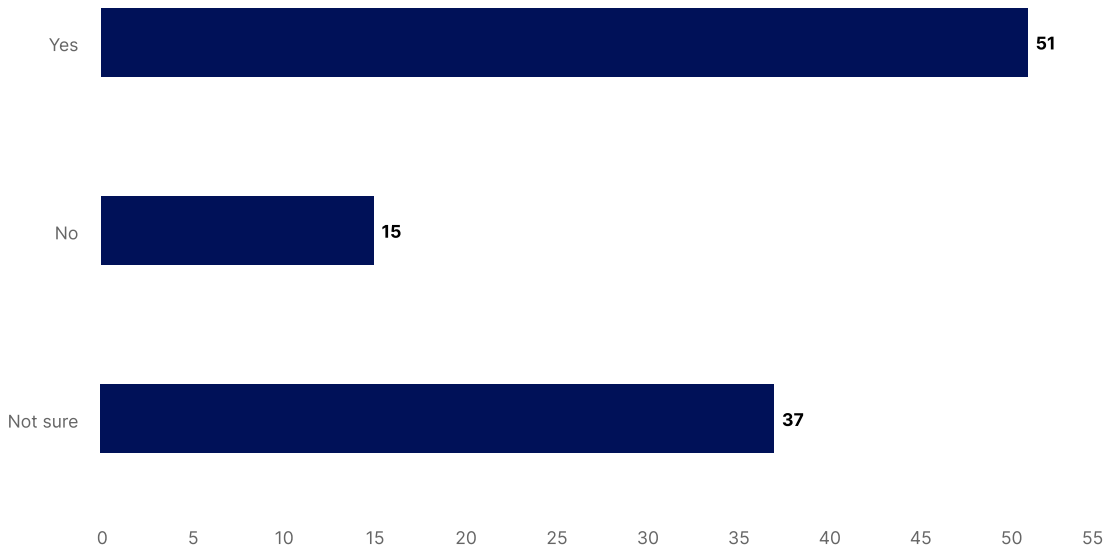

Figure 4.14: Responses to (Q15) Do you believe the security of code generated by different generative AI tools varies significantly?

Figure 4.15 shows the main factors that contribute to the differences in the security of different AI tools. Almost 73% of the respondents believed that the transparency and explainability of the generated code was the main factor. Around 62.74% also thought that the specific functionalities of the tools, i.e. code completion vs full program generation, was a deciding factor and 45.09% believed the availability of security features within the tools themselves to be a key difference. Only 37.25% of respondents considered the reputation of the tool and the regular updates and patching of the tools important.

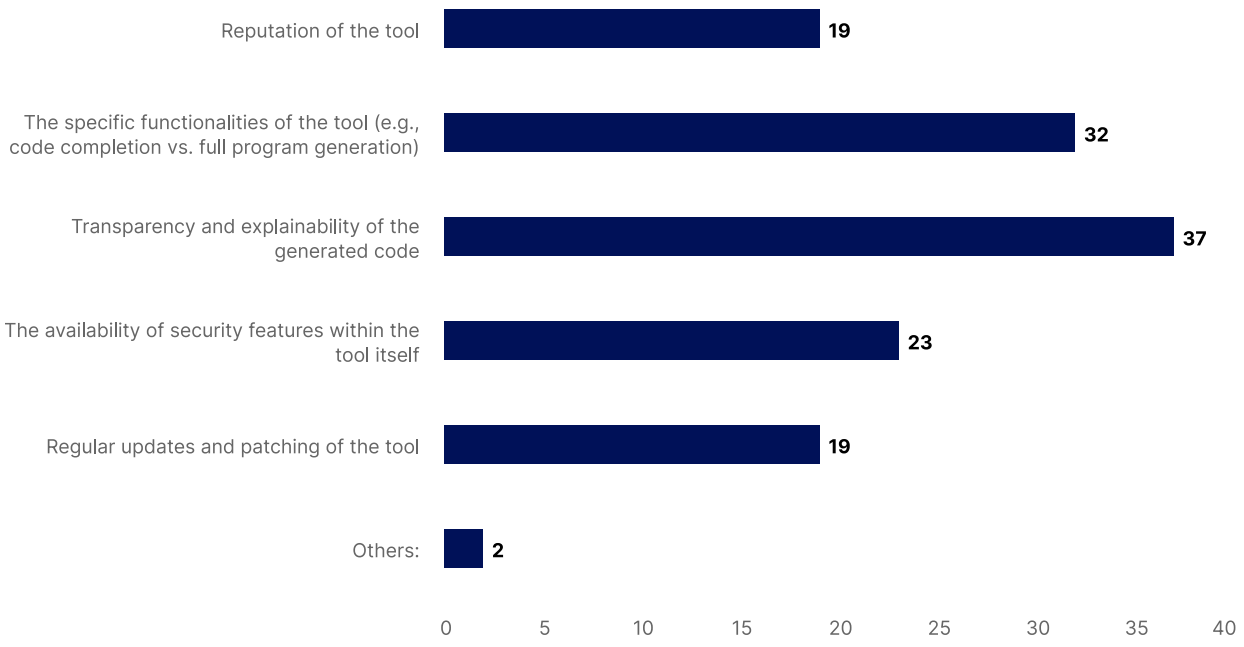

Figure 4.15: Responses to (Q15A) Which of the following factors do you believe contribute most to the differences in security between generative AI tools?

Around 12% of respondents added some other factors that could contribute to the security differences between generative AI tools. These findings are presented below.

4.7.1 AI tool development and training

Respondents viewed the importance of security measures taken by AI tool developers as critical. Furthermore, the quality and source of the data used to train the AI model were thought to significantly impact the security of the generated code, providing a clear difference in the security of these tools.

4.7.2 Purpose of AI tools

The responses also focused on the idea of different AI tools being designed with specific goals in mind, such as logical reasoning or coding, and their effectiveness can vary based on these specializations.

4.8 Generative AI tools confidence

Figure 4.16 shows the different types of generative AI tools used by participants. ChatGPT was the most (85.71%) used generative AI tool of the respondents, followed by Github Copilot (57.14%). Around 13.33% of participants used Intellicode, while around 8.57% used Gemini.

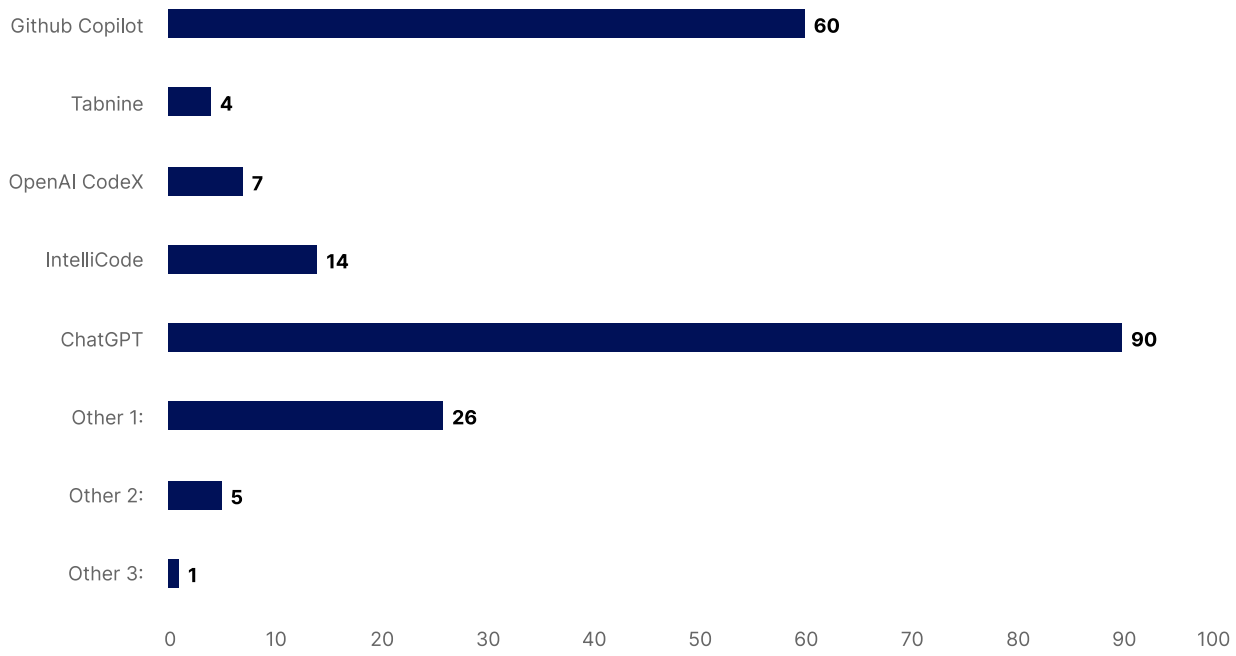

Figure 4.16: Responses to (Q16) Which generative AI tools do you use for coding?

Figure 4.17 represents the confidence that the participants had in the security of the code generated by the AI generative tools they use. Overall, a combined 74.28% reported feeling somewhat or very confident, and a significant portion (52.38%) remained unsure or not confident at all. Neutrality was also a common response (64.76%). One thing to note was that only a small minority (15.23%) expressed strong confidence, and an even smaller group (13.33%) lacked confidence entirely.

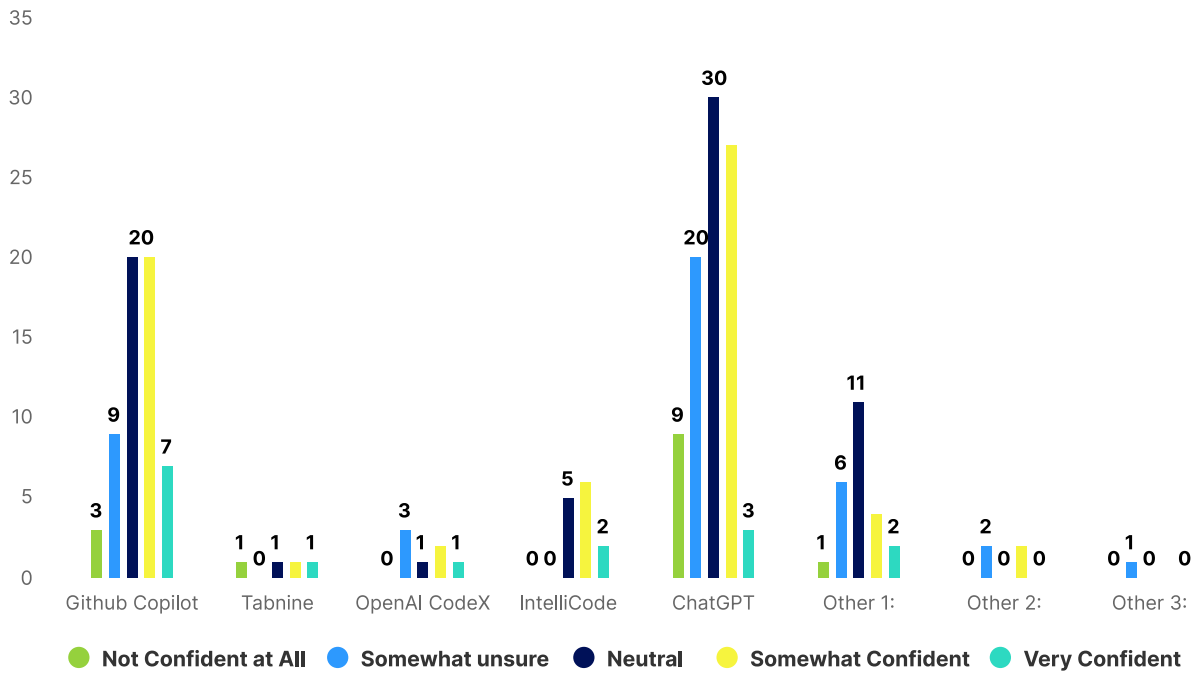

Figure 4.17: Responses to (Q17) How confident are you in the security of code generated by the selected Generative AI tools?

4.9 Code completion vs full program generation

Figure 4.18 shows the answer to the statement: ”There is a higher risk of security vulnerabilities in code generated by full program creation tools compared to code completion tools.” A majority (59%) agreed that full program creation tools pose a higher risk of security vulnerabilities compared to code completion tools with 23% strongly agreeing with the statement. Only 19% disagreed with this assessment. The remaining 22% of respondents remained neutral on this issue.

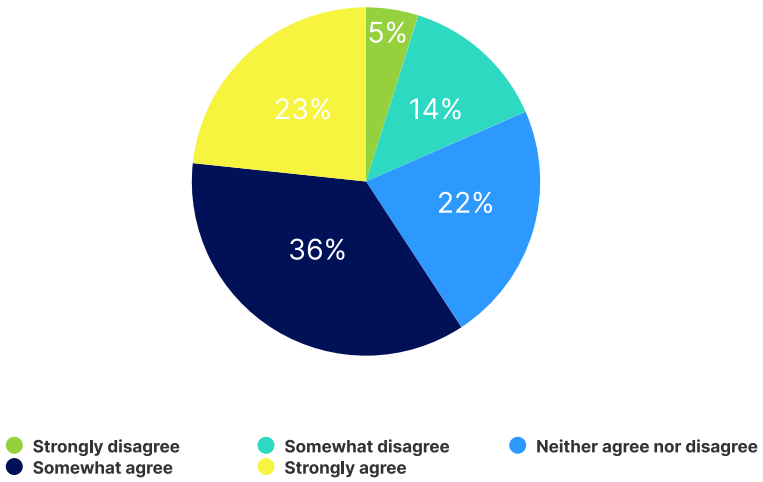

Figure 4.18: Responses to (Q18) Code completion vs. Full program generation: ”There is a higher risk of security vulnerabilities in code generated by full program creation tools compared to code completion tools.”

4.10 Confidence in generated code

Figure 4.19 shows the confidence participants had in AI-generated code compared to self-generated code. A significant portion (42.7%) of respondents expressed a lack of confidence in AI-generated code security. Neutrality was also common, with 32% of respondents being neutral. Only 25.3% reported feeling somewhat or very confident in AI-generated code security. Notably, a larger percentage (10.7%) lacked confidence entirely, compared to the small group (1%) who were very confident.

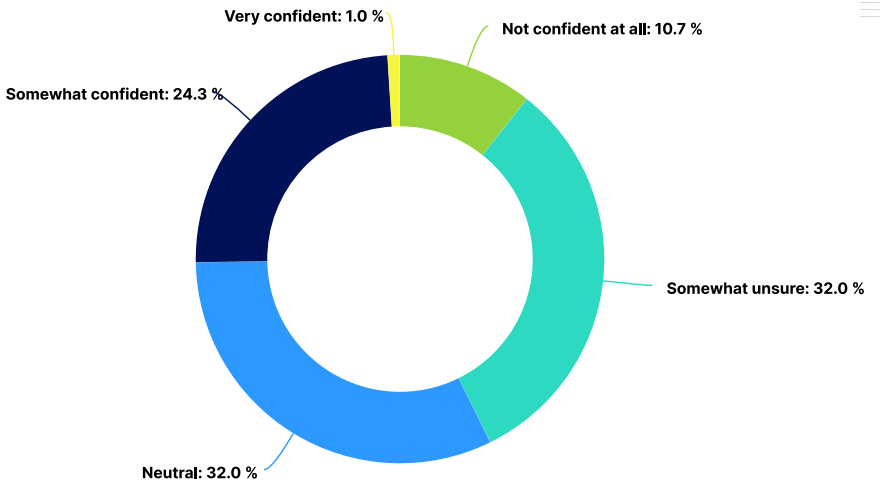

Figure 4.19: Responses to (Q19) How confident are you in the security of code generated by generative AI compared to code written by yourself?

CHAPTER 5

DISCUSSION

5.1 Perceptions and evaluation of generative AI code

Developers' perceptions and evaluations of generative AI code security (RQ1) are a complex interaction between the perceived benefits and risks (RQ1.1), the used security practices (RQ1.2), the factors prioritized when reviewing that security (RQ1.3), and the confidence in generative AI (RQ1.4) code compared to their own code. This section discusses these four aspects.

5.1.1 Perceived benefits and risks

In answer to the research question (RQ1.1) "How do developers perceive the benefits and risks of using generative AI for code security?", the participants considered the main benefits of using the generative AI code as faster development and increased efficiency. This suggests an increase in developer productivity. Developer productivity is often measured by how much code they write (lines of code) or how many functionalities they implement (function points) in a given amount of time [59]. The boost in productivity can be explained by the reduction of time spent on coding as shown in the results (Figure 4.9). Furthermore, this productivity increase aligns with previous research [62, 54] on the positive impact of generative AI on developer productivity. One interesting finding was that the higher the developer experience (beginner, intermediate, advanced) with using AI tools for coding was the more they believed their coding time was decreased. This could be explained by the fact that experienced developers are more proficient in using AI, leading to faster development and more code integration into the codebase. Additionally, this proficiency can also lead to more confidence in the code generated by the AI tools leading to even faster integration.

On the other hand, the participants found the top risk to be an overreliance on AI without human oversight. Klemmer et al. [24] urge developers to carefully evaluate all AI suggestions. This overdependence on AI can cause people to accept potentially incorrect decisions without proper validation and instead of analyzing every AI suggestion, people develop basic guidelines for when

to trust and follow AI recommendations [6]. Another risk was generating code with vulnerabilities. This could be due to the buggy nature of the coding data used to train the model. Chapter 2 already touched on this idea, where LLMs trained in code from programming communities may inherit security vulnerabilities or bugs present in that data [39].

Furthermore, privacy and ethical considerations were identified as a risk for both current and future AI models. This concern stems from the fundamental link between data and AI performance [45]. Large language models (LLMs) trained on massive datasets, including potentially biased or unethically sourced data (e.g., copyrighted content, pirated material) [37, 63], risk inheriting and perpetuating these biases or privacy violations. According to Klemmer et al. [24], organizations also see privacy as the primary driving force of concern when using AI-generated code. Beyond data concerns, the lack of transparency and auditability in many AI codebases makes it difficult to understand how these models arrive at their decisions. This lack of explainability can lead to the perpetuation of unintended biases within the code itself.

5.1.2 Security best practices

A significant number of participants (82%) in response to the research question (RQ1.2) "What security best practices do developers recommend when using generative AI code?" mentioned thorough review and understanding of AI-generated code before integration as the most important security best practice. Furthermore, nearly half of the participants (44.11%) agreed that they spend more time reviewing generative AI code, according to Figure 4.9. This emphasis on thorough review aligns with the concerns regarding the non-deterministic nature of AI models [27]. Because AI-generated code can produce code with vulnerabilities as discussed in Chapter 2, careful examination is important for identifying and mitigating potential security vulnerabilities before integration.

The second most important practice to follow was to not enter any sensitive or personal information. The participants also mentioned this in the answer to the open question of security factors when reviewing generative AI code. This reason could be because most LLMs are retrained on the input they receive from users to improve their responses. This retraining process raises concerns about the potential for user-entered data, especially sensitive information, to be memorized by the LLM [52, 29]. Even if anonymized, the training data used for LLMs might contain sensitive details.

Moreover, using active code scanning tools to identify vulnerabilities was considered an important security practice. The respondents to the open question also suggested using SonarQube code for scanning the generative AI code. Vulnerability scanners are important for proactive code scanning. In a project with multiple developers, there's no guarantee each one followed the best practices. Vulnerability scanners become a critical tool in such scenarios, aiding in the detection of potential weaknesses that manual review alone might miss [23].

5.1.3 Security factors

This subsection discusses the answers to the research question (RQ1.3) ”What factors do developers consider most important when evaluating the security of code generated by generative AI?”. Half of the participants (50%) believed the clarity and understandability of the code as the most important factors when reviewing generative AI code. This aligns with the finding that thorough code reviewing and understanding of AI-generated code is the most important security practice. Throughout the research, the idea of understanding and reviewing code has been prominent. The primary reason is that even the best models are prone to vulnerabilities due to being trained on a large volume of code with vulnerabilities.

On the second rank, the participants mentioned alignment of generated code with security best practices as the main factor. AI models can inherit vulnerabilities from buggy code during training (as discussed in Chapter 2). Generative AI code might also reflect these weaknesses if it does not adhere to established security best practices. Reviewing such alignment helps identify and address potential security gaps before integration.

Third, the absence of known vulnerabilities in the generated code was mentioned as the main factor when reviewing. The participant identified this as one of the primary risks associated with AI-generated code. This is mainly due to the nature of training data. After examining all these findings, the importance of training data is highlighted. As mentioned in prior research by Roh et al [46], training data is considered as a major bottleneck in AI training. In the context of generative AI code security, the quality of training data becomes even more crucial. While the absence of known vulnerabilities in the generated code is a positive sign, it can not guarantee secure code if the underlying training data contains vulnerabilities.

Interestingly, the least important factor was believed to be the reputation and security track record of generative AI tools by 50% of the respondents. This finding was also consistent when the different generative AI tools were compared. The reason could be that developers believe in reviewing, understanding generated code, and aligning with security best practices rather than solely depending on the tool’s reputation.

5.1.4 Confidence in generative AI code

The respondents answers to the research question (RQ1.4) ”How confident are developers in the security of generative AI code compared to the code they write themselves?” are discussed. Compared to their own code, participants clearly showed less confidence in AI-generated code. Nearly 43% reported feeling this way, while only 25.3% expressed positivity towards AI-generated code. Interestingly, a seemingly contradictory finding emerged when participants were asked about their confidence in the security of code generated by the generative AI tools that they use. Here, a significant number (74.28%) reported feeling somewhat or very confident, and a significant portion (52.38%) remained unsure or not confident at all.

A well-documented human bias, the tendency to overestimate one’s own abilities, could explain this contradiction. Research indicates a systematic bias in how individuals process information

regarding their own capabilities [32]. Additionally, people generally lack self-awareness of their biases, and may even resort to rationalizing their biased viewpoints [19]. In this context, developers might generally trust AI-generated code but exhibit less confidence when directly comparing it to code they have written themselves.

This intriguing finding between general confidence in AI-generated code and lack of confidence, when the comparison is made between AI-generated code and personally written code, is a potential area for future research. Investigating how developers' self-perceptions of their coding abilities influence their trust in AI-generated code could provide valuable insights for improving the integration and adoption of generative AI tools in the software development process.

5.2 Behavior changes using AI-generated code

To understand "What factors influence developers' behavior while using generative AI?" (RQ2) compared to code written by themselves, participants were presented with various behavioral changes and asked about their opinions. The results are discussed below.

5.2.1 Faster development

The majority of participants (61.16%) reported a reduction in coding time when using generative AI tools, aligning with the identified benefit of increased developer efficiency and faster development. This time saving likely comes from two factors: first, the ability to generate additional lines of code using AI, and second, the utility of generative AI code as a foundation for further development. As Ziegler et al. [62] suggest, "a suggestion that serves as a useful template to tinker with may be as good or better than a perfectly correct (but obvious) line of code that only saves the user a few keystrokes".

5.2.2 Cautious deployment

A significant number of participants (60%) expressed a preference for exercising caution when deploying generative AI code to production compared to code they wrote themselves. This cautious approach is likely driven by the identified security concerns surrounding AI-generated code producing vulnerability due to training on open communities data with vulnerabilities. This increased scrutiny, while time-consuming, can help to mitigate potential vulnerabilities, and protect the integrity and reliability of software applications in the long run.

5.2.3 Code reviewing

Almost half of the participants (44.11%) said they spend more time on security reviews. This increase in review time could be explained by the recommended best practice of thoroughly examining AI-generated code prior to integration. Additionally, some respondents noted spending more time on debugging, suggesting that while generative AI can accelerate development, it may also introduce new challenges that require additional testing and troubleshooting.

5.2.4 Understanding logic

A significant portion of participants (40.59%) reported challenges in comprehending the logic underpinning AI-generated code compared to their own. This difficulty can be attributed to the black-box nature of AI models, where the decision-making processes remain largely opaque. We have limited comprehension of the decision-making processes within AI systems [43]. This lack of transparency can hinder troubleshooting, debugging, and the overall adoption of AI-generated code within development workflows.

5.2.5 Documentation importance

Nearly half of the respondents (49.01%) found it more important to document the purpose and limitations of AI-generated code for future reference. This increased importance given to the documentation is crucial to the maintainability of the codebase in the long run. By documenting the role and limitations of AI code, developers can facilitate future maintenance, updates, and collaboration.

5.3 GAIs functionality & developers' perceptions of security

Different GAIs provide different functionalities such as full program generation or code completion. Developers may perceive differences in the security of code generated by the various AI tools. Additionally, to fully understand developer perceptions of the security of these tools, it is important to consider what factors cause these differences and the confidence developers have in these tools. This section discusses these aspects.

5.3.1 Difference in security of generated code

This subsection discusses the research question (RQ3.1) "Are there differences in how developers perceive the security of code generated by different generative AI tools?". Almost half of the participants (49.51%) believed that there are significant differences in the security of code generated by different AI tools. Out of these, transparency and explainability of the generated code were considered to be the primary factors in the differences in code security. This explainability and transparency of the code have been an important recurring theme in this discussion.

The second deciding factor was the specific functionalities of the tools, i.e. code completion vs full program generation. For the research question (RQ3.2) "How does the perceived functionality of different AI tools (e.g., code completion vs. full program generation) influence developers' perception of the impact on code security?", a majority of participants (59%) mentioned that full program creation tools pose a higher risk of security vulnerabilities compared to code completion tools. This concern may be from the increased complexity involved in creating entire programs compared to completing code snippets. As discussed in Chapter 2, code completion tools often work within the existing tools and context of a developer's code, potentially making them more

aligned with the specific project's architecture and security practices. Full program generation tools may lack this nuanced understanding of the project's context. To further explore this difference in perceived risk, future research could delve deeper into the specific security challenges associated with full program generation tools compared to code completion.

Participants expressed confidence in the overall security of the code of the generative AI tools they use. However, only a small minority expressed strong confidence. The fact that only a small minority expressed strong confidence suggests a cautious approach to the usage of generative AI code. This caution in the security of AI-generated code is likely driven by the understanding that these tools are susceptible to vulnerabilities.

5.4 Implications for organizations

The main insights from this research for organizations are the following:

- Organizations should prioritize code review practices, especially when dealing with code produced by generative AI tools. Organizational teams should be trained not to rely only on AI outputs and to closely analyze them for vulnerabilities.

- Secure coding practices should be focused more on developers in organizations, emphasizing the importance of following established guidelines and standards. Introducing mandatory peer reviews and automated security checks for AI-generated code could further reduce risks.

- Active code scanning tools should be integrated as part of the organizations' development pipelines. These tools will help detect security flaws that manual review might miss, especially in large, complex projects where different team members may have varying levels of adherence to security standards.

- Companies should implement strict data-handling policies that ensure no confidential or sensitive information is entered into AI tools. Additionally, developers should be educated about the importance of anonymizing data inputs.

- Organizations should approach full program generation tools with caution and where possible, organizations might prefer code completion tools that work within the developer's existing context, reducing the likelihood of introducing security vulnerabilities.

5.5 Limitations

This research was conducted through a survey. The survey may lead to some biases and limitations.

- The first bias was related to self-selection bias. Participants who choose to complete the survey may not represent the entire population of software developers who are familiar with generative AI tools. Participants interested in generative AI might have been more likely to respond to the survey. The focus on the researcher's network could also have introduced bias. Although snowballing methods were utilized, future research would benefit from extending to other countries and regions. In order to combat this bias, the survey was shared through personal contact, in online communities on LinkedIn, and in developers' communities as a QR code.

- The perceptions and experiences of developers who participated in the survey may not be generalizable to all developers, especially those using different generative AI tools or programming languages other than the one mentioned in this research. However, the findings of this study offer valuable insights into the current landscape of the mentioned languages and generative AI tools. Future research could replicate this study using different AI tools and programming languages that are not so well known.

- One potential limitation lay in the design of the survey questions and the generative AI tools selected for the survey. To address this, we conducted a pilot test to assess the clarity of the survey and identify any shortcomings. Based on the feedback, the survey was refined before final distribution.

- The length of the survey may have led to participant fatigue, particularly towards the latter parts of the survey. This could have affected the quality and completeness of the data collected. To assess participant fatigue, attention checks could have been implemented. However, these checks themselves can contribute to fatigue, so they were not used in this study. To mitigate this fatigue, skip logic was used where necessary and a progress indicator was displayed. Additionally, participants were told an estimate of the completion time before starting the survey.

- Surveys do not provide a deeper understanding of participants' perceptions, and integrating interviews would improve this aspect.

- The study relies on self-reported data which may introduce bias to the results as mentioned earlier. To obtain more accurate insights into developer behavior, it would be beneficial to complement the study with practical tasks, such as asking participants to complete coding assignments both with and without generative AI.

CHAPTER 6

CONCLUSION

The study aimed to understand the perception of developers on code security when working with AI-generated code. The main question was, "How do developers perceive the impact of generative AI approaches on code security?". We conducted the research using a surveyccollecting 105 responses, from online developer groups on LinkedIn, the author's personal contacts, and the distribution of QR codes at a software developers' gathering. The main respondents were professional programmers, students, and researchers.

Faster development and increased efficiency were the main benefits of using generated AI code, while overreliance on AI-generated code without human oversight and the presence of vulnerabilities in AI code were the main risks. Moreover, privacy and ethical considerations were considered as risks not only for current models but also for future models 5 years from now. Understanding these benefits and risks will help developers improve their interaction with AI-generation tools.

The research identified the main security best practices as, 1) thorough review and understanding of AI-generated code before integration, 2) not entering any sensitive or personal information, 3) use active code scanning tools to identify vulnerabilities. During the review process, the most crucial factors included the code's clarity and understandability, its alignment with security best practices, and the absence of known vulnerabilities in the generated code. One important thing was that the least important factor was believed to be the reputation and security track record of generative AI tools. Developers believed in thorough reviewing and understanding of generated code, and alignment with security best practices rather than depending on the reputation of the tool.

The behavior of developers suggested that they were now coding faster, but also spending more time on reviewing and understanding the logic. Moreover, most of the developers were cautious about deploying generative AI code to production and had increased the importance of documenting the generated AI code for future reference. These changes give insights into how development practices may need to change to improve the usage of generative AI in developers' workflow.

The research suggested that developers perceived that full program creation tools pose a higher risk of security vulnerabilities compared to code completion tools. This suggests that developers should consider tools like ChatGPT that produce full programs in one go more carefully, however, it is important to be cautious of using AI-generated code no matter the type of tool used. Moreover, the developers exhibited a moderate level of confidence in the security of various AI tools; however, when they compared the security of generative AI code with their own code, they displayed greater skepticism. This presents an interesting finding that these differences may be influenced by the self-confidence of developer in their own code leading to bias.

6.1 Future Work

The following discussion provides an opportunity for future work.

- Future work could use another research method along with the survey, such as in-depth interviews with both professional programmers and students, to gain a better understanding of developers' perceptions of AI-generated code security.

- As mentioned in the discussion, future research could aim to understand the differences in general confidence in the security of AI-generated code compared to developers' own code.

- Further research could also look into the specific security issues that come up with full program generation tools versus code completion tools and try to figure out why developers think that full program generation tools are riskier.

BIBLIOGRAPHY

[1] Owura Asare, Meiyappan Nagappan, and N. Asokan. Is GitHub's Copilot as Bad as Humans at Introducing Vulnerabilities in Code? *Empirical Software Engineering*, 28(6):129, 4 2023.

[2] Hala Assal and Sonia Chiasson. "Think secure from the beginning": A survey with software developers. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery, 5 2019.

[3] Autumn Clark, Daniel Igbokwe, Samantha Ross, and Minhaz F. Zibran. A Quantitative Analysis of Quality and Consistency in AI-generated Code. *2024 7th International Conference on Software and System Engineering (ICoSSE)*, pages 37–41, 4 2024.

[4] Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1), 4 2023.

[5] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. Analyzing the state of static analysis: A large-scale evaluation in open source software. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016*, volume 1, pages 470–481. Institute of Electrical and Electronics Engineers Inc., 5 2016.

[6] Zana Buçinca, Maja Barbara Malaya, and Krzysztof Z. Gajos. To Trust or to Think: Cognitive Forcing Functions Can Reduce Overreliance on AI in AI-assisted Decision-making. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1), 4 2021.

[7] Nicholas Carlini, Florian Tramèr, Katherine Lee, Adam Roberts, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. *Extracting Training Data from Large Language Models*. 2021.

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings,

Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. 7 2021.

[9] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. 2 2020.

[10] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate : building and scaling high performing technology organizations*. IT Revolution, 2018.

[11] Steve Freeman and Nat Pryce Aaddison-Wesley. Growing Ob j ect-Oriented Software, Guided by Tests. Technical report, 2009.

[12] Mohammad Ghafari, Pascal Gadient, and Oscar Nierstrasz. Security Smells in Android. 6 2020.

[13] Github Inc. GitHub Copilot · Your AI pair programmer, 2021.

[14] Mark G Graff. Secure Coding The State of the Practice. Technical report, 2001.

[15] Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. Semantic Similarity Metrics for Evaluating Source Code Summarization. In *IEEE International Conference on Program Comprehension*, volume 2022-March, pages 36–47. IEEE Computer Society, 2022.

[16] Matthew Ivory, Miriam Sturdee, John Towse, Mark Levine, and Bashar Nuseibeh. Can you hear the ROAR of software security? How Responsibility, Optimism And Risk shape developers' security perceptions. 2023.

[17] Jason Cohen, Steven Teleki, and Eric Brown. *Best kept secrets of peer code review. Smart Bear Incorporated.* 2006.

[18] Jiaqi Zhu and Mingzhu Shen. Research on Deep Learning Based Code Generation from Natural Language Description. *EEE 5th International Conference on Cloud Computing and Big Data Analytics*, pages 188–193, 2020.

[19] Johanna Johansen, Tore Pedersen, and Christian Johansen. Studying human-to-computer bias transference. *AI and Society*, 38(4):1659–1683, 8 2023.

[20] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and João Paulo. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? Technical report, 2013.

[21] Angelos. Keromytis, Anil. Somayaji, Christian W. Probst, Matt. Bishop, ACM Digital Library., and Association for Computing Machinery. The developer is the enemy. *Proceedings of the 2008 New Security Paradigms Workshop*, pages 89–97, 9 2008.

[22] M. Mehdi Kholoosi, M. Ali Babar, and Roland Croft. A Qualitative Study on Using ChatGPT for Software Security: Perception vs. Practicality. 8 2024.

[23] Cassidy Khounborine. A Survey and Comparative Study on Vulnerability Scanning Tools. Technical report, 2023.

[24] Jan H. Klemmer, Stefan Albert Horstmann, Nikhil Patnaik, Cordelia Ludden, Cordell Burton, Carson Powers, Fabio Massacci, Akond Rahman, Daniel Votipka, Heather Richter Lipford, Awais Rashid, Alena Naiakshina, and Sascha Fahl. Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns. 5 2024.

[25] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. Code review quality: How developers see it. In *Proceedings - International Conference on Software Engineering*, volume 14-22-May-2016, pages 1028–1038. IEEE Computer Society, 5 2016.

[26] Heiko Koziolek, Andreas Burger, Marie Platenius-Mohr, Julius Rückert, Hadil Abukwaik, Raoul Jetley, and Abdulla Pp. Rule-based Code Generation in Industrial Automation: Four Large-scale Case Studies applying the CAYENNE Method. Technical report, 2020.

[27] Kim Tuyen Le and Artur Andrzejak. Rethinking AI code generation: a one-shot correction approach based on user feedback. *Automated Software Engineering*, 31(2):60, 11 2024.

[28] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, 2 2024.

[29] Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Yuguang Yao, Chris Yuhao Liu, Xiaojun Xu, Hang Li, Kush R. Varshney, Mohit Bansal, Sanmi Koyejo, and Yang Liu. Rethinking Machine Unlearning for Large Language Models. 2 2024.

[30] Tamara Lopez, Thein T. Tun, Arosha K. Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. Taking the Middle Path: Learning about Security through Online Social Interaction. *IEEE Software*, 37(1):25–30, 1 2019.

[31] Gary Mcgraw. Building Security In. Technical report, 2004.

[32] Markus M Möbius, Muriel Niederle, Paul Niehaus, U C San, Diego Nber, Tanya S Rosenblat, Nageeb Ali, Roland Benabou, Gary Chamberlain, Rachel Croson, Gordon Dahl, David Eil, Glenn Ellison, Asen Ivanov, John List, Justin Rao, Al Roth, Joel Sobel, Lise Vesterlund, and Roberto Weber. Managing Self-Confidence *. Technical report, 2014.

[33] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel Von Zezschwitz, and Matthew Smith. "If you want, I can store the encrypted password." A Password-Storage Field Study with Freelance Developers. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 1–12. Association for Computing Machinery, 5 2019.

[34] Daye Nam, Andrew MacVean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an LLM to Help with Code Understanding. In *Proceedings - International Conference on Software Engineering*, pages 1184–1196. IEEE Computer Society, 2024.

[35] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. 3 2022.

[36] Ayman Odeh, Nada Odeh, and Abdul Salam Mohammed. A Comparative Review of AI Techniques for Automated Code Generation in Software Development: Advancements, Challenges, and Future Directions. *TEM Journal*, 13(1):726–739, 2024.

[37] Daniel E O'leary, Michael Cox, and David Ellsworth. AI INNOVATION IN INDUSTRY Artifi cial Intelligence and Big Data What Is Big Data? Technical report, 2013.

[38] Cyrus Omar, Youngseok Yoon, Thomas D Latoza, and Brad A Myers. Active Code Completion. Technical report, 6 2012.

[39] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768, 2022.

[40] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do Users Write More Insecure Code with AI Assistants? In *CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2785–2799. Association for Computing Machinery, Inc, 11 2023.

[41] Philip Sedgwick. Erratum: Unit of observation versus unit of analysis (BMJ (2014) 348 (g3840)), 10 2015.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Technical report, 2020.

[43] Arun Rai. Explainable AI: from black box to glass box, 1 2020.

[44] Irum Rauf, Tamara Lopez, Helen Sharp, Marian Petre, Thein Tun, Mark Levine, John Towse, Dirk Van Der Linden, Awais Rashid, and Bashar Nuseibeh. Influences of developers' perspectives on their engagement with security in code. In *Proceedings - 15th International Conference on Cooperative and Human Aspects of Software Engineering, CHASE*, pages 86–95. Institute of Electrical and Electronics Engineers Inc., 2022.

[45] Huw Roberts, Josh Cowls, Jessica Morley, Mariarosaria Taddeo, Vincent Wang, and Luciano Floridi. The Chinese approach to artificial intelligence: an analysis of policy, ethics, and regulation. *AI and Society*, 36(1):59–77, 3 2021.

[46] Yuji Roh, Geon Heo, and Steven Euijong Whang. A Survey on Data Collection for Machine Learning: a Big Data – AI Integration Perspective. 11 2018.

[47] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. Technical report, 2023.

[48] Martina Angela Sasse, Sacha Brostoff, and Dirk Weirich. Transforming the "Weakest Link": A Human-Computer Interaction Approach for Usable and Effective Security. *BT Technology Journal*, 19(3):122–131, 2001.

[49] Fritz Scheuren. What is a Survey. Technical report, 6 2004.

[50] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. Using AI-Based Coding Assistants in Practice: State of Affairs, Perceptions, and Ways Forward. 6 2024.

[51] Helen Sharp, Hugh Robinson, and Mark Woodman. Software_engineering_community_and_culture. *IEEE Software*, 17(1):40–47, 2000.

[52] Nianwen Si, Hao Zhang, Heyu Chang, Wenlin Zhang, Dan Qu, and Weiqiang Zhang. Knowledge Unlearning for LLMs: Tasks, Methods, and Challenges. 11 2023.

[53] Mohammed Latif Siddiq, Shafayat H. Majumder, Maisha R. Mim, Sourov Jajodia, and Joanna C.S. Santos. An Empirical Study of Code Smells in Transformer-based Code Generation Techniques. In *Proceedings - IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 71–82. Institute of Electrical and Electronics Engineers Inc., 2022.

[54] Stack Overflow. Stack Overflow Developer Survey 2023. , 2023.

[55] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. IntelliCode compose: Code generation using transformer. In *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443. Association for Computing Machinery, Inc, 11 2020.

[56] Tabnine. Tabnine - The AI coding assistant that you control, 2022.

[57] Davide Tosi. Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation. *Future Internet*, 16(6), 6 2024.

[58] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery, 4 2022.

[59] Stefan Wagner and Melanie Ruhe. A Systematic Review of Productivity Factors in Software Development. 1 2018.

[60] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. 4 2023.

[61] Ziyao Zhang, Yanlin Wang, Chong Wang, Jiachi Chen, and Zibin Zheng. LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation. 9 2024.

[62] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Measuring github copilot's impact on productivity. *Communications of the ACM*, 67(3):54–63, 2 2024.

[63] P Zikopoulos, D Deroos, K Parasuraman, T Deutsch, J Giles, and D Corrigan. Big data in action Helps users explore large, complex data sets Streamlines the process of developing big data applications Monitors and manages big data systems for secure and optimized performance Speeds time-to-value with analytical and industry-specific modules. Technical report, 2013.

APPENDIX A

Survey questions

Page break

Developers’ Perceived Impact of Generative AI Approaches on Code Security

Information Sheet

You're invited to participate in a research study for a Master's thesis project about developers' perceived impact of generative AI approaches on code security. Your involvement is voluntary, and you can withdraw at any time. Any data gathered will be strictly used for research purposes and may be included in the writing of the master thesis.

The study involves completing a survey comprising multiple-choice questions, descriptive text box questions, and other quantifiable responses. The survey can take around 13 minutes to complete, and you can withdraw your participation at any point without any explanation.

If you have questions before starting the survey, please reach out to the researcher. Additionally, if you have any concerns about the study, feel free to contact either the researcher or the supervisors.

You can reach us via email at:
Researcher: n.a.hotak@umail.leidenuniv.nl
Supervisors: a.kudriavtseva@liacs.leidenuniv.nl, o.gadyatskaya@liacs.leidenuniv.nl

---

I have read the above information about the study

- ○ Yes (1)

CONSENT
I agree to participate in this study

- ○ Yes (4)
- ○ No (5)

Skip To: End of Survey If QID27 = No

Q1 Which of the following best describes your current occupation?

- ○ Student (1)
- ○ Hobbyist Programmer (2)
- ○ Professional Programmer (3)
- ○ Researcher (4)
- ○ Freelancer Programmer (5)
- ○ Other: (6) _____

Q2 How many years of experience do you have in software development?

- ○ Student/learning (1)
- ○ 0 to 2 years of professional experience (2)
- ○ 3 to 5 years of professional experience (3)
- ○ 6 to 8 years of professional experience (4)
- ○ 9 to 11 years of professional experience (5)
- ○ More than 11 years of professional experience (6)

Q3 What programming languages do you primarily use? (Select up to 3 from the list below)

- [ ] TypeScript (1)
- [ ] Python (2)
- [ ] JavaScript (3)
- [ ] Ruby (4)
- [ ] Java (5)
- [ ] C++ (9)
- [ ] C# (6)
- [ ] PHP (10)
- [ ] Go (7)
- [ ] other (8) _____

Q4 How proficient are you at writing secure code **Without** AI tools in the language you use most?

- ○ Beginner – I can write a correct implementation for a simple function (1)
- ○ Intermediate - I can design and implement secure programs, but I still have some concerns about potential vulnerabilities (2)
- ○ Advanced – I can design and implement complex, secure system architectures with a high degree of confidence in the code's robustness (3)

Q5 How proficient are you at writing secure code **With** AI tools in the language you use most?

- ○ Beginner – I can write a correct implementation for a simple function (1)
- ○ Intermediate - I can design and implement secure programs, but I still have some concerns about potential vulnerabilities (2)
- ○ Advanced – I can design and implement complex, secure system architectures with a high degree of confidence in the code's robustness (3)

Q6 In your opinion, what are the perceived benefits of using generative AI in code security? (Select all that apply)

- ☐ Increased Efficiency (1)
- ☐ Automatic generation of patches for vulnerabilities (2)
- ☐ Faster development (3)
- ☐ Improved detection of vulnerabilities (4)
- ☐ Improved code maintainability (5)
- ☐ Others: (6) _____

Q7 In your opinion, what are the perceived risks of using generative AI in code security?
(Select all that apply)

- [ ] Overreliance on AI without human oversight (1)
- [ ] Introduction of new vulnerabilities (2)
- [ ] Integration challenges with existing security tools (3)
- [ ] Privacy and ethical concerns in AI-generated solution (4)
- [ ] Likely to generate code with similar vulnerabilities (5)
- [ ] Integration challenges with the code/infrastructure (7)
- [ ] Others: (6) _____

Q8 Five years from now, which of the following challenges related to security do you foresee for AI-generated code?
(Select all that apply)

- [ ] Integration challenges with the code / infrastructure (1)
- [ ] Lack of transparency and auditability in AI-generated code (2)
- [ ] Integration challenges with existing security tools (3)
- [ ] Privacy and ethical concerns in AI-generated code (4)
- [ ] Unintended biases in generating specific code (5)
- [ ] I am not sure / I don't foresee any specific challenges (9)
- [ ] Others: (6) _____

9 The following statements represent **changes** in developers' **Behaviors**

	Strongly disagree (27)	Somewhat disagree (28)	Neither agree nor disagree (29)	Somewhat agree (30)	Strongly agree (31)
I spend less time writing code with GAI compared to writing myself (1)	○	○	○	○	○
I am more cautious about deploying GAI code to production compared to deploying without GAI (2)	○	○	○	○	○
I spend more time on security reviews when using GAI code compared to code written by myself (3)	○	○	○	○	○
I find it less challenging to understand the logic behind GAI code compared to the code I write myself (4)	○	○	○	○	○
I find it more important to document the purpose and limitations of	○	○	○	○	○

GAI code for future reference compared to the code I write myself (5)	

Q10 In your opinion, what other changes in BEHAVIORS occur?

_____

_____

_____

_____

_____

Q11 Which security best practices do you recommend developers follow when using generative AI code (GAI)
(Select all that apply)

- ☐ Choose GAI tools from reputable vendors with a focus on security (1)
- ☐ Thoroughly review and understand all AI-generated code before integration (2)
- ☐ Use active code scanning tools to identify vulnerabilities (3)
- ☐ Do not enter sensitive or personal information into the generative AI tools (4)
- ☐ Incorporating secure coding standards and guidelines (5)
- ☐ Others: (6) _____

Q12 Which security practices do you use when coding using generative AI tools?

_____

_____

_____

_____

_____

Q13 When reviewing generative AI code, how important do you consider the following security factors :

(Please rank from 1 (most important) to 5 (least important))

_____ Absence of known vulnerabilities in the generated code (1)

_____ Clarity and understandability of the code (2)

_____ Alignment of the generated code with secure coding best practices (3)

_____ Integration with security analysis tools (4)

_____ Reputation and Security track record of GAI tool (5)

Q14 In your opinion, what other security factors are important when reviewing generative AI code?

_____

_____

_____

_____

_____

Q15 Do you believe the security of code generated by different generative AI tools varies significantly?

- ○ Yes (1)
- ○ No (2)
- ○ Not sure (3)

Display This Question:

If Q15 = Yes

Q15A Which of the following factors do you believe contribute most to the differences in security between generative AI tools?
(Select all that apply)

- ☐ Reputation of the tool (1)
- ☐ The specific functionalities of the tool (e.g., code completion vs. full program generation) (2)
- ☐ Transparency and explainability of the generated code (3)
- ☐ The availability of security features within the tool itself (4)
- ☐ Regular updates and patching of the tool (5)
- ☐ Others: (7) _____

Display This Question:

If Q15 = Yes

Q15B In your opinion, what other factors contribute to the differences in security between different generative AI tools?

_____

_____

_____

_____

_____

Q16 Which generative AI tools do you use for coding?
(Select all that apply)

- ☐ Github Copilot (1)
- ☐ Tabnine (2)
- ☐ OpenAI CodeX (3)
- ☐ IntelliCode (4)
- ☐ ChatGPT (5)
- ☐ Other 1: (6) _____
- ☐ Other 2: (7) _____
- ☐ Other 3: (8) _____

Display This Question:

If If Which generative AI tools do you use for coding? (Select all that apply) q://QID19/SelectedChoicesCount Is Greater Than or Equal to 1

Carry Forward Selected Choices from "Q16"

Q17 How confident are you in the security of code generated by the selected Generative AI tools?

	Not Confident at All (1)	Somewhat unsure (2)	Neutral (3)	Somewhat Confident (4)	Very Confident (5)
Github Copilot (x1)	○	○	○	○	○
Tabnine (x2)	○	○	○	○	○
OpenAI CodeX (x3)	○	○	○	○	○
IntelliCode (x4)	○	○	○	○	○
ChatGPT (x5)	○	○	○	○	○
Other 1: (x6)	○	○	○	○	○
Other 2: (x7)	○	○	○	○	○
Other 3: (x8)	○	○	○	○	○

Q18 Code completion vs. Full program generation:

”There is a higher risk of security vulnerabilities in code generated by full program creation tools compared to code completion tools.”

- ○ Strongly disagree (6)
- ○ Somewhat disagree (7)
- ○ Neither agree nor disagree (8)
- ○ Somewhat agree (9)
- ○ Strongly agree (10)

Q19 How confident are you in the security of code generated by generative AI compared to code written by yourself?

- ○ Not confident at all (1)
- ○ Somewhat unsure (2)
- ○ Neutral (3)
- ○ Somewhat confident (4)
- ○ Very confident (5)

Q20 To receive the findings of this research, kindly share your email address with us.

_____