My code is less secure with Gen AI: surveying developers' perceptions of the impact of code generation tools on security

Kudriavtseva, A.; Hotak, N.A.; Gadyatskaya, O.

My Code Is Less Secure with Gen AI: Surveying Developers' Perceptions of the Impact of Code Generation Tools on Security

Arina Kudriavtseva
Leiden University
Leiden, The Netherlands
a.kudriavtseva@liacs.leidenuniv.nl

Nisar Ahmad Hotak
Leiden University
Leiden, The Netherlands
n.a.hotak@umail.leidenuniv.nl

Olga Gadyatskaya
Leiden University
Leiden, The Netherlands
o.gadyatskaya@liacs.leidenuniv.nl

Abstract

Background: Generative AI (GAI) tools like GitHub Copilot and ChatGPT are transforming software development by automating code generation and enhancing developers' productivity. However, since these tools are often trained on open-source repositories, they may inadvertently reproduce vulnerable code, raising concerns about the security of AI-generated outputs. **Aims:** In this paper, we aim to investigate how developers perceive code security when using GAI tools. **Method:** We conducted a survey with 105 software developers with diverse experience levels to gather their perceptions regarding the security of generated code and their suggestions for improving it. **Results:** While developers reported increased development speed when using GAI tools, many spend additional time on security reviews and documentation of the generated code, and they are worried about the overreliance on AI and vulnerabilities in the code. Only about a quarter of the developers expressed confidence in the code generated by AI, and, moreover, experienced developers perceive that their proficiency in secure coding decreases when using GAI tools. Our results provide organizations with a better understanding of the risks associated with GAI tools and help improve their software security programs.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Security and privacy** → **Software security engineering**.

Keywords

Secure software development; developer study; generated code security

ACM Reference Format:
Arina Kudriavtseva, Nisar Ahmad Hotak, and Olga Gadyatskaya. 2025. My Code Is Less Secure with Gen AI: Surveying Developers' Perceptions of the Impact of Code Generation Tools on Security. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707778

SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707778

1 Introduction

Generative artificial intelligence (GAI) tools such as GitHub Copilot[1] or OpenAI's Codex[2] and ChatGPT[3] have demonstrated impressive capabilities in generating code, allowing developers to quickly translate ideas into functional code [46].

However, since GAIs are often trained on data from open-source repositories like GitHub, they may inadvertently learn and replicate code that contains software bugs and security vulnerabilities [46]. The 2024 Open Source Security and Risk Analysis (OSSRA) report[4] revealed that 84% of the analyzed codebases contain at least one vulnerability, with 74% having high-risk vulnerabilities. Consequently, there is a concern that the GAI tools could perpetuate vulnerabilities learned from the training during the code generation process, resulting in flawed and highly exploitable code [46]. For example, studies have shown that Copilot generates insecure code about 40% of the time [31], and only 5 out of 21 programs produced by ChatGPT were initially secure [20]. Additionally, participants who had access to an AI assistant tended to write less secure code compared to those without [32]. As AI-driven programming becomes more prevalent, it is important to understand what are the effects perceived by software developers on security of their code and on their own behavior. Are they concerned about potential insecurities in the generated code? What do they do to mitigate these issues?

In this paper, we aim to investigate developers' perceptions regarding code security when using GAI tools, such as GitHub Copilot and ChatGPT. We explore these perceptions from several directions: we study the changes in secure coding proficiency and behavior that developers report, the perceived benefits and risks of using GAI, the security best practices followed by developers, and the factors contributing to security of generated code, the GAI tools used by developers, and the level of trust they have in these tools. Concretely, we formulated these research questions for our study:

RQ1 *What is the effect of using code generation tools on developers' behavior (with respect to code security)?*
RQ2 *What are the perceived benefits and risks of using generated code?*
RQ3 *What do developers recommend to improve the security of generated code?*
RQ4 *How do developers perceive the security of code generated by AI tools they use?*
RQ5 *How does the level of development experience affect developers' perceptions regarding security of generated code?*

[1] https://github.com/features/copilot
[2] https://openai.com/index/openai-codex/
[3] https://chatgpt.com/
[4] https://www.synopsys.com/blogs/software-security/open-source-trends-ossra-report.html

To address these RQs, we conducted a survey with 105 developers to gather their perceptions of the security of GAI code. We analyzed the survey data and discussed the findings in the context of existing literature on the use of AI-generated code in software development. Our main findings are as follows:

- Developers perceive substantial changes in their secure coding proficiency when not using GAI tools compared to when using them. While self-reported proficiency without GAI aligns robustly with their development experience, their proficiency with GAI varies: novice developers tend to perceive that they become better in secure coding, while advanced developers tend to perceive their proficiency to be degraded. More experienced developers are also more likely to have less trust in the security of AI-generated code compared to less experienced developers.
- Developers also report changes in their behavior when using GAI: while they see AI programming assistants as productivity boosters, they also report paying more attention to code review before integration and to documenting the generated code properly.
- The major GAI-introduced risks that developers perceive are the overreliance on AI, security vulnerabilities in the code, and ethical and privacy concerns.
- To reduce these risks and improve security of generated code, developers recommend thoroughly reviewing the AI-generated code and using code scanning tools for vulnerability detection. Another recommended security practice is avoiding submitting sensitive data to GAI tools.
- Developers' perceptions of security of AI-generated code are influenced by various factors, including clarity and understandability of the code and the absence of known vulnerabilities on the generated code.
- Among all prominent GAI tools that most participants are familiar with, IntelliCode and Copilot are considered to be somewhat more reliable.
- Overall, the perceptions of security of AI-generated code appear to be consistent across all levels of developers' experience

Our results provide organizations with a better understanding of the risks associated with GAI tools and generated code, and help improve their software security.

2 Research Methodology

2.1 Questionnaire Design

We designed our survey questions to answer the research questions using the Qualtrics platform[5]. To ensure quality, we followed the guidelines for conducting survey studies in Software Engineering [26]. The survey took about 15 minutes to complete. It was anonymous and participants were required to read the study information and provide their consent before answering the questionnaire. The target audience of the survey was software developers. We aimed to survey both experienced and junior developers, as well as students learning to develop software, to capture the perceptions and trends among the different groups of developers. The complete survey questionnaire is available in the supplementary materials [22].

To construct the multiple-choice questions, we relied on the existing literature on developers' behaviors, potential benefits, challenges, and risks associated with using AI-generated code, as well as factors contributing to security [8, 34, 35, 42, 45]. Additionally, we engaged in discussions with developers in our network to gain insights into their experiences with using AI-generated code, which helped us validate the constructed multiple-choice questions. Furthermore, a pilot study involving five participants from our professional networks was conducted to assess the clarity, flow, and overall quality of the survey questions. Based on the feedback received from these participants, the survey was refined and improved.

2.2 Survey Response Collection and Analysis

The survey was run from May 19, 2024, to 13 July, 2024. We invited people to participate in our survey via the personal network of the authors, as well as via invitations given out during several large industry and academic conferences dedicated to software engineering and software security. In addition, prospective participants within our professional networks were invited by mail and direct messages on the LinkedIn platform[6]. The participants were not compensated, but we offered to share the study findings if the participants were interested in them.

We received 196 responses to the survey in the study period. Out of these collected responses, we considered 105 to be valid responses by participants that 1) were not considered by Qualtrics to be bots (based on the CAPTCHA scores and time spent on the survey) and 2) viewed and answered all questions. The 91 discarded answers included 10 bots and 81 people who did not complete the whole survey.

The demographic information of the 105 participants with valid responses is depicted in Figure 1. For further analysis, we group the participants into *junior* (student/learning, 0-2 years of experience), *medior* (3-5 years), and *senior* (more than 6 years) groups based on their years of experience. The distribution of the study participants by experience level is then as follows: junior – 41%, medior – 28%, and senior – 31%. It is worth noting that about half of our respondents do not identify themselves as professional developers, but they are, e.g., students, hobbyists, and freelancers. We deliberately did not exclude them from the target audience as today such people play a significant role in global software development, including open-source software projects.

For closed-ended questions, we aggregated the results using descriptive statistics. Answers to open questions were analyzed and common themes were identified.

We did three statistical tests: two Kruskal–Wallis tests, and one Brunner-Munzel test. Accounting for multiple comparisons, we apply the Bonferroni correction and set the significance level p=0.016.

2.3 Ethical considerations

The study design was approved by the Science Ethics Committee at Leiden University. To protect participants' privacy, we limited the collection of their personal data as much as possible. We stored and analyzed the data in compliance with the EU General Data Protection Regulation (GDPR). All participants were presented with a consent form explaining the goals of the research, the collected

[5]https://www.qualtrics.com/

[6]https://linkedin.com/

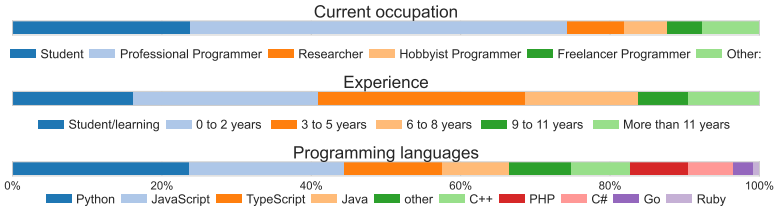

Figure 1: Demographic information of the participants.

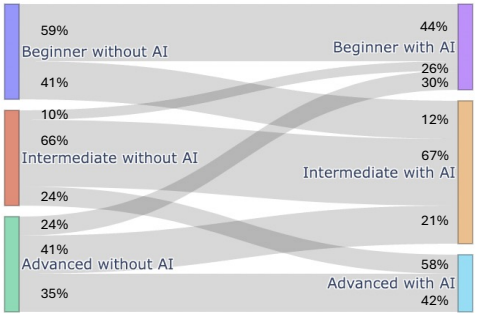

Figure 2: Change in the proficiency level in writing secure code with GAI tools.

data, and the research method. We only considered answers from participants who explicitly consented to participate in the study and allowed us to publish the results in an anonymized, aggregated format.

3 Results

3.1 RQ1: Perceived Secure Coding Proficiency and the Effect of GAI Tools

The influence of AI on secure coding proficiency. To understand the impact of GAI on self-assessed programming proficiency, we asked participants to rate their proficiency levels in writing secure code with and without the GAI tools. The results are summarized in Table 1. As we can see, more than half of the participants rated their code security proficiency levels as intermediate, both with and without GAI. However, as we examine further, these are not the same people.

We explored the relationship between self-reported proficiency in writing secure code *without GAI tools* and years of professional experience. There is a strong correlation between the level of proficiency in writing secure code without AI tools and experience (Spearman's rank correlation $\rho = 0.94$). That means that, as expected, the level of proficiency in writing secure code without GAI tools increases with years of professional experience. The participants's self-assessment of proficiency in secure coding without generative AI can thus be considered to be reasonably objective.

We note that the self-assessed secure coding proficiency levels *with GAI tools* are nuanced – some participants reported that their proficiency increases with GAI, while others indicated that their proficiency actually decreases. To show how developers move from one proficiency group to another, we depict the results in the Sankey chart 2. This figure indicates that GAI tools have some positive impact on the perceived secure coding proficiency levels of more junior developers, while more senior developers appear to be more concerned about the security of the code they write with GAI. Overall, 20% of participants estimated that their secure coding proficiency increases with GAI, while 24% considered that their proficiency decreases. Proficiency of the rest (56%) remained at the same level. There is now almost no correlation between the level of proficiency in writing secure code with GAI and the years of experience (Spearman's rank correlation ρ = -0.028). Thus, developers perceive that the presence of GAI tools substantially affects their proficiency in writing secure code, and this proficiency no longer reflects their actual professional experience.

This interesting phenomenon with the self-assessed proficiency in secure coding with and without AI can be potentially attributed to the Dunning-Kruger effect: perhaps, more senior and experienced people are more aware of the security issues with generated code [21, 31, 32, 36] and they suspect that the security of their code would overall diminish, while more junior developers might be prone to over-estimate their capabilities in secure coding with AI, because they are less aware. It would be interesting to investigate this more in-depth in future studies.

Behavior change with AI. To understand the impact of GAI on the software development process, we investigated the changes in developers' behavior when using GAI tools. Table 2 shows that the majority of the participants (61%) agreed with **spending less time writing code** with generative AI tools compared to the code written by themselves. Neutral participants were 13%, while almost 26% replied that the time spent on coding using generative AI tools was not decreased. This result is consistent with studies from the literature that show faster coding with GAI tools [18, 40].

Likewise, the majority of the participants (61%) were **more cautious about deploying generated code** to production compared to deploying their own code, while 26% remained neutral on this aspect, and a small percentage (13%) reported not being cautious about deploying AI-generated code to production. Spending **more time on reviewing security of AI-generated code** was reported by almost half of the participants (44%), while nearly 36% of respondents were neutral. One-fifth (20%) of the participants disagreed on spending more time on security reviews. **Understanding the logic** behind generative AI code proved to be a challenge for a significant portion of the respondents (41%) who disagreed with finding it easier than understanding their own code. Around a third (30%) of the responses neither agree nor disagree. While 29% found

Table 1: Distribution of secure coding proficiency levels with and without AI by experience level

Experience Level	Beginner (%)		Intermediate (%)		Advanced (%)	
	Without AI	With AI	Without AI	With AI	Without AI	With AI
Student/learning	47	35	53	35	0	23
0 to 2 years	19	15	81	77	0	8
3 to 5 years	13	21	59	52	28	28
6 to 8 years	0	0	44	56	56	44
9 to 11 years	0	29	29	43	71	29
> 10 years	0	50	30	40	70	10
Total population	16	22	56	55	28	23

the logic behind AI-generated code less challenging to grasp. Almost half of the participants (49%) found it **more important to document** the purpose and limitations of generated code for future reference. A neutral response was provided by around one-third (32%) of participants. One-fifth (19%) of the participants disagreed.

To investigate whether the reduction in coding time was perceived differently based on the experience level (junior, medior, senior), we performed cross-tabulation. As shown in Figure 3, a higher percentage of junior developers (29%) expressed skepticism about spending less time on coding with AI assistance, compared to 28% of medior developers and 21% of advanced developers. In contrast, the majority of senior developers (67%) agreed that they spent less time on coding with AI. Similarly, 61% of medior developers and 57% of junior developers agreed with this perception. Our analysis using the Kruskal-Wallis test with the Bonferroni correction did not show that the experience level of the developers significantly influenced the time spent on coding with GAI tools compared to code written by themselves ($H = 2.4$, $p = 0.3$).

We also cross-tabulated the experience level with the perception of spending more time reviewing security of generated code. The Figure 3 shows that 39% of senior developers, 48% of medior developers, and 47% of junior developers agreed that they spend more time on security reviews of generated code. On the other hand, 15% of senior developers, 14% of medior developers, and 29% of junior developers disagreed. We observed that the largest percentage of developers (29%) who disagreed with spending more time reviewing GAI code came from the junior experience level, compared to senior and medior developers. However, the Kruskal-Wallis test with the Bonferroni correction did not show that the experience level of the developers influences the time spent on security reviews of generated code ($H = 6.9$, $p = 0.03$).

Figure 3: Agreement with statements "*I spend less time writing code with GAI*" and "*I spend more time reviewing generated code*": comparison among experience levels.

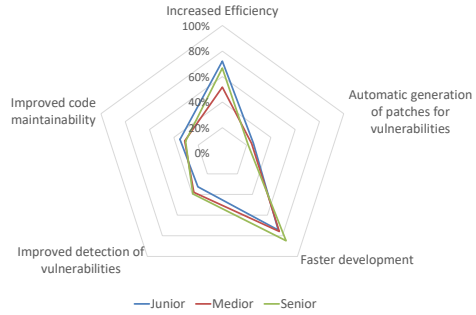

Figure 4: Perceived benefits for code security of using generative AI tools, per experience level.

3.2 RQ2: Perceived Benefits and Risks of GAI Tools

Perceived benefits. We asked the developers about the benefits of generative AI tools they perceived. A significant proportion of respondents (78%) identified **faster development** (decreased time to production) as a key advantage. Furthermore, 65% of the participants viewed **increased efficiency** (i.e. spending less resources on a project) as a major benefit. Almost equal proportion received **improved detection of vulnerabilities** (36%) and **improved code maintainability** (32%). **Automatic generation of patches for vulnerabilities** was indicated by 24%. A small number of participants (4%) used the opportunity to list their own answers under "Other". These answers included increased security awareness, standardization of code implementation, and easier development due to AI-refactored code.

The mapping of perceived benefits with the level of experience is depicted in Figure 4. The graph shows that the percentage of participants who identified each benefit as a key advantage of using generative AI tools across different levels of experience is nearly the same. The only gap is in the increased efficiency, where the percentage of participants who identified this benefit as a key advantage is higher for junior developers (72%) compared to intermediate-level developers (52%).

Table 2: Participants' self-assessment of behavioral changes when using GAI tools. Percentages in italics on the chart (*N%*) represent the percent of the distribution that reported "*Strongly Disagree*"/"*Somewhat Disagree*" (left) and "*Somewhat Agree*"/"*Strongly Agree*" (right).

Behavioral changes	Distribution
I spend less time writing code with GAI compared to writing myself	*26%* *61%*
I am more cautious about deploying GAI code to production compared to deploying without GAI	*13%* *61%*
I spend more time on security reviews when using GAI code compared to code written by myself	*20%* *44%*
I find it less challenging to understand the logic behind GAI code compared to the code I write myself	*41%* *29%*
I find it more important to document the purpose and limitations of GAI code for future reference compared to the code I write myself	*19%* *49%*

Strongly Disagree | Somewhat Disagree | Neither Agree nor Disagree | Somewhat Agree | Strongly Agree

Perceived risks and challenges. We then examined the perceived risks of generative AI in code security. The **overreliance on AI without human oversight** was found to be the greatest risk perceived by 65% of the participants. Additionally, the other important perceived risks and challenges were: likely to **generate code with similar vulnerabilities** (59%), the **introduction of new vulnerabilities** (52%), **privacy and ethical concerns** in AI-generated solutions (48%), **integration challenges with the code/infrastructure** (29%), and **integration challenges with existing security tools** (21%). Some participants (3%) suggested additional risks associated with using generative AI code, including supply chain-related risks (the possibility of incorporating expired or conflicting libraries) and the risk of hacked GAI tools, which could lead to bugs in many pieces of software. The prominent concern of our participants about vulnerabilities aligns with the literature showing that generated code is frequently vulnerable [9, 31] as well as software professionals being wary about this [21].

After mapping the perceived risks with years of professional experience (Figure 5), we found that overreliance on AI without human oversight is the highest perceived risk for junior (76%) and senior (64%), with a drop for medior developers (52%). In contrast, the integration challenges with existing security tools are perceived to be the smallest risk by all levels. Notably, perceptions of the risk of introducing new vulnerabilities and the integration challenges with existing code and infrastructure increase with experience. Privacy and ethical concerns in AI-generated solutions and the risk of generating code with similar vulnerabilities are perceived to be somewhat more significant with increased experience, with their perceived importance rising from junior to medior and then decreasing slightly for senior (46%).

Looking forward, we investigated the challenges related to the security of AI-generated code in five years envisaged by our participants. **Unintended biases in generated code** (56%), **privacy and ethical concerns** in AI-generated code (52%), and **lack of transparency and auditability** of AI-generated code (51%) were considered the most important challenges. A small portion of participants (5%) provided own suggested challenges regarding the high risk of similar code with vulnerabilities, having less experienced developers, and overreliance on AI.

When comparing the current perceived risks that developers have with the challenges anticipated for generative AI in code security in five years, we see that privacy and ethical concerns are perceived to become even more important in five years (3% increase). The integration challenges are at the bottom of both lists with a small percentage of developers selecting them as a current risk and a predicted challenge in five years.

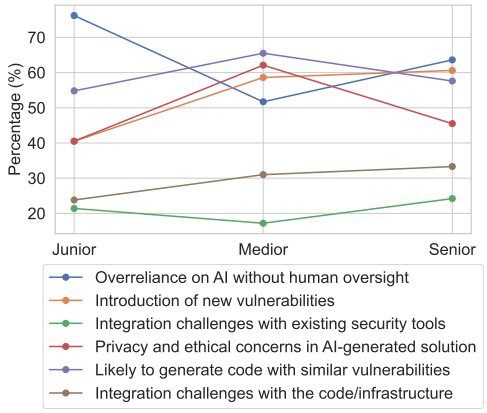

Figure 5: The perceived risks of using generative AI in code security, per experience level.

3.3 RQ3: Improving Security of Generated Code

Recommended security best practices. Figure 6 presents the security best practices that developers recommend 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 suggested **avoiding submitting sensitive data or personal information to GAI tools**. Furthermore, 62% of the participants mentioned **using active code scanning tools** to identify vulnerabilities as one of the best security practices. Two participants provided their own responses, suggesting to "*use safe tools*" and "*utilize an enterprise plan like Copilot, which ensures source code confidentiality on their servers*". Thus, to eliminate security risks, developers recommend reviewing, testing, and fully understanding generated code before integrating it and being mindful of confidentiality and privacy issues when using remote GAI tools. We note that other studies also consistently highlight the importance of maintaining vigilant security reviews to prevent security issues (e.g. [21, 43]).

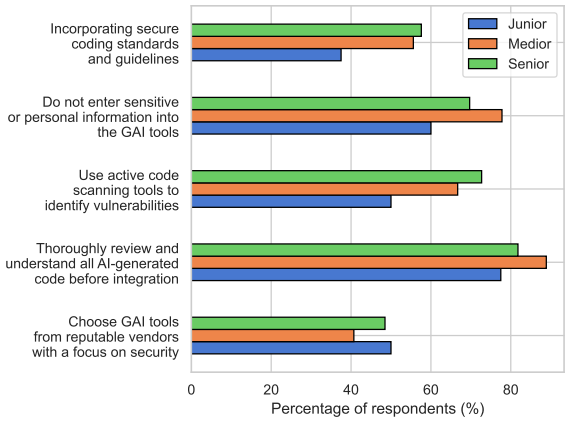

Figure 6: Security practices recommended by developers.

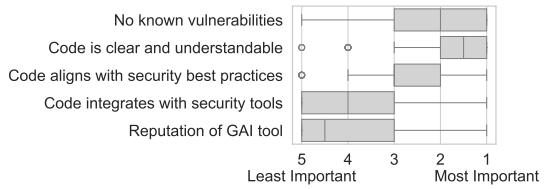

Figure 7: Boxplots of the rating of factors that contribute to code security according to the participants.

After mapping the recommended security practices with the experience level, we observe that junior developers were less inclined to choose the recommended security practices compared to medior and senior developers. The only exception was for choosing a GAI tool from a reputable vendor, where junior developers were more likely to recommend this practice. Overall, all security practices received the same share of medior and senior respondents.

We also asked developers which security best practices they currently follow when coding with generative AI tools. The answers mostly came from juniors (64%). In total, 20 security best practices were reported. Although the question was open-ended, most of the answers were similar to those we gave as options to discover the security practices recommended by developers, suggesting a priming effect. The most frequently reported option (provided by 12 participants) emphasized a thorough review of AI-generated code, including manual review to ensure the correctness and full understanding of the code before implementing it. Several participants (10) stated that they do not share sensitive information with AI tools to preserve privacy and prevent data leakage, including "*turning off data sharing so the AI will not learn from my input when working with sensitive data*". There was also an emphasis on sanitizing inputs and avoiding the use of real credentials or identifiable data. In addition, six developers mentioned using tools such as SonarQube for code scanning to identify security vulnerabilities and code smells as part of the build process. One participant described their work with AI-generated code in this way: "*Usually, I take the generated code, throw it away, and write it again myself. Not saying that this is more secure. The reason for this, however, is not primarily security, but maintainability of the code*".

Factors contributing to security. Participants ranked the security factors they should consider when reviewing AI-generated code. The factors are ranked from 1 (most important) to 5 (least important) (Figure 7). What we see is that the **clarity and understandability of the code** are the most important factors for developers when reviewing code generated by AI. The **absence of known vulnerabilities in the generated code** is the second most important factor. The **reputation and security track record of the GAI tool** is the least important factor for developers when reviewing code generated by AI.

We note that the factors contributing to security align closely with the security practices that developers follow and recommend. The clarity and understandability of the code were consistently highlighted as the most important factor and a thorough review of the generated codes was the most frequently recommended security practice. Similarly, the use of scanning tools to find vulnerabilities was both a prominent recommended security practice and the absence of vulnerabilities is an important factor that contributes to security. On the other hand, choosing a generative AI tool from reputable vendors was considered the least recommended practice and ranked as the least important factor in ensuring security.

Around 20% of the participants also responded to the open question about what other security factors are important when reviewing generative AI code. One prevalent idea was the importance of protecting and checking confidential data in the code, like API keys, IDs, and personal information when using generative AI tools: "*Secrets like API keys and IDs must not be shared with the GAI provider*". Some of the responses focused on the need to check that the AI-generated code followed secure coding practices, such as input validation, proper error handling, and avoiding insecure coding patterns. Another group of respondents suggested that generative AI might be less suitable for certain programming languages with unique characteristics. Solidity[7] was mentioned as an example: "*Consider Solidity, it is less likely to give an optimal code for it.*"

The need for proper testing procedures to validate the generated code was another important theme. Furthermore, the structure should be up to date with current versions of the chosen programming language to reduce the risk of backdoors or newly discovered vulnerabilities. Two participants also recommended abstaining from using GAI: "*Code without any GAI tools, practice yourself*", considering writing code without AI tools as a factor that contributes to security. As we can see, the responses to the open question were consistent with the responses to the open question regarding the security practices developers follow. We can conclude that developers consider the factors that contribute to security while using GAI tools and reflect these factors in the security practices they follow.

[7]Solidity, a programming language designed for creating smart contracts for the Ethereum blockchain platform https://soliditylang.org/

3.4 RQ4: Used GAI Tools and Confidence in Generated Code

Security of different GAI tools. We asked participants if they trust the security of code generated by different generative AI tools varies significantly. The most substantial part of the participants (49%) agree with this statement, while a much smaller proportion (15%) disagree. Interestingly, one-third (36%) of developers were unsure about whether there are significant variations in the security of code generated by different GAI tools.

We then asked developers about the factors that contribute to the differences in generated code security among different GAI tools. The majority (74%) of the respondents chose **transparency and explainability of the generated code** as the main factor. The **specific functionalities of the tool**, such as code completion versus full program generation, was the second most popular choice (64%), which agrees with the results from [29]. **Reputation of the GAI tool** and **regular updates and patching of the tool** seem to be less important factors with equal amounts of votes (38%). Other factors mentioned by the participants include the nature of the coding language, true positive and false negative rates, and the goal of the tool (e.g., logical reasoning, coding, or answering complex questions). Two participants highlighted the importance of the amount of training the AI tool had and the quality of the source code on which the tool was trained: "*It also depends on how much training this AI tool has had, the less training – the more likely it is to screw up a line or two in the sense of security.*"

Commonly used GAI tools. We asked which GAI tools the participants used and how they would assess confidence in the security of the code generated by those tools. ChatGPT was the most (89%) used GAI tool, followed by GitHub Copilot (59%). This aligns with the most recent literature [11, 21] and the StackOverflow developers' survey [39]. Table 3 presents the confidence levels of the participants in security of the code generated by the AI tools they use. It appears that IntelliCode is perceived to be the most reliable tool because more than half of developers who use it (62%) are confident in the security of the code it generates, while nobody stated that they were not sure about it. GitHub Copilot creates doubts in 20% of the participants, while 46% of satisfied users. The Tabnine AI assistant received the most conflicting votes. However, very few people answered about this tool, so no conclusion can be drawn about it. Similarly, most of the participants did not select the OpenAI CodeX model. Overall, among the two most used GAI tools, Copilot from GitHub is considered to be more reliable.

As GitHub Copilot and ChatGPT were the most commonly used tools, we compared the confidence in the security of the code generated by these tools using the Brunner-Munzel test with the Bonferroni correction. We found that, overall, the confidence in the security of the code generated by GitHub Copilot was not significantly higher than the confidence in the security of the code generated by ChatGPT (Z = -2.4, p-value: 0.017).

Program creation versus code completion. We asked the participants whether there is a higher risk of vulnerabilities in code generated by full program creation tools compared to code completion tools. A slight majority (59%) agreed that full program creation tools pose a higher risk of security vulnerabilities compared to code completion tools, with 19% disagreeing with this assessment. The remaining 22% of respondents remained neutral on this issue.

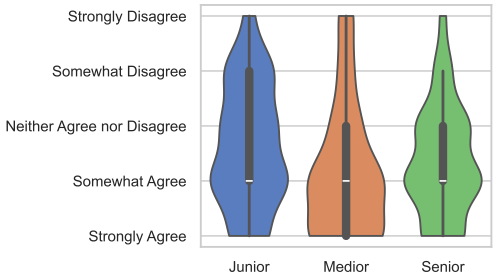

Figure 8: Agreement with the statement "*There is a higher risk of security vulnerabilities in code generated by full program creation GAI tools compared to code completion tools*".

When mapping the answers with the experience levels, we observed that more mediors (36%) strongly agreed, while only 17% of juniors and 21% of seniors strongly agreed with this statement. In contrast, the percentage of juniors (21%) who somewhat disagreed with the statement was higher compared to mediors (7%) and seniors (9%). The results in Figure 8 suggest that more experienced developers are slightly more likely to perceive that full program creation tools pose a higher risk of security vulnerabilities compared to code completion tools.

Confidence in AI-generated code. Figure 9 illustrates the participants' confidence in security of AI-generated code compared to code written by themselves. The results indicate that a significant portion (43%) of the respondents expressed a lack of confidence in the security of AI-generated code. A substantial proportion of the participants (32%) stated that they were neutral on this question. A quarter (25%) reported feeling somewhat or very confident in the security of AI-generated code. Notably, a larger percentage (11%) lacked confidence entirely, compared to the small group (1%) who were very confident. This general distrust in security of the generated code aligns with the findings from a recent qualitative study by Klemmer et al. [21].

When examining the confidence levels based on experience, it is interesting to note that the small group of developers who were very confident in AI-generated code were only juniors. On the other hand, medior (14%) and senior (15%) developers were more likely to express a complete lack of confidence compared to juniors (5%). This suggests that more experienced developers are more likely to have less trust in the security of AI-generated code compared to less experienced developers.

3.5 RQ5: The Effect of Experience on Perceptions

As discussed above, for all major questions we have mapped the responses with the level of developers' experience to discover the difference in perceptions regarding security of GAI-generated code. We now synthesize these results and answer our final research question. Overall, our statistical analysis did not reveal a substantial

Table 3: Participants' confidence in the security of code generated by generative AI tools. Percentages in italics on the chart (*N%*) represent the percent of the distribution that reported "*Not Confident at All*"/"*Somewhat Unsure*" (left) and "*Somewhat Confident*"/"*Very Confident*" (right).

GAI tools	% users	Distribution
ChatGPT	89%	*33%* *34%*
Github Copilot	59%	*20%* *46%*
IntelliCode	14%	*0%* *62%*
OpenAI CodeX	7%	*43%* *43%*
Tabnine	4%	*25%* *50%*

Not Confident at All | Somewhat Unsure | Neutral | Somewhat Confident | Very Confident

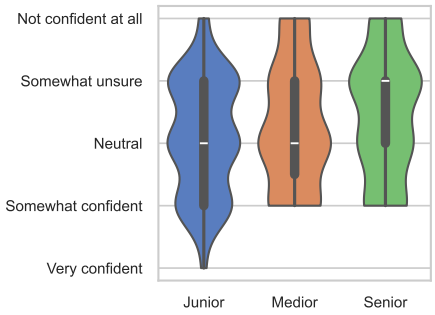

Figure 9: Answers to the question "*How confident are you in the security of code generated by AI tools compared to code written by yourself?*".

difference in perceptions between the groups, but some notable distinctions emerged. Particularly, senior developers spend less time coding with AI assistance, while junior developers do not perceive such a substantial productivity gain, and both medior and junior developers perceive spending more time on security reviews of generated code compared to senior developers. However, our statistical analysis did not find any significant difference between the groups, and an industry study [13] previously found productivity gains with GAI to be the most pronounced for junior developers. So, our observation needs further investigation in a larger study.

The benefits of using GAI were found to be consistent across all levels of experience, with one exception. Junior developers were more likely to identify increased efficiency as a key benefit, compared to medior-level developers. Interestingly, while junior developers were less inclined to mention spending less time coding with GAI tools, they were more likely to perceive increased efficiency.

We observed slight variations in perceived risks based on experience level, but there were no visible major trends. In terms of security practices, medior and senior developers showed similar eagerness to recommend best practices, whereas junior developers were less likely to do it. Additionally, more experienced developers tend to see full program generation tools as posing higher security risks compared to code completion tools and generally trust AI-generated code less than less experienced developers.

4 Limitations

Our survey has several limitations that may impact our results. Firstly, there is a possibility of self-selection bias [1] because we had no control over sample selection. This may introduce a bias towards individuals who are more interested or knowledgeable about the topic. Furthermore, the non-response bias could be present because the participants who did not respond to the survey could have different opinions than those who did respond [14].

Additionally, our survey relies on self-reported answers. That means that participants could give wrong or incomplete answers about themselves because they want to appear better than they are or if they misunderstand the questions [7]. We tried to mitigate this in the pilot test.

Due to the limitations of the survey format, we were unable to ask many follow-up questions or add many options for multi-select questions because it would increase the cognitive load on the participants and could decrease the response rate [12]. We attempted to compensate for this by designing and pilot-testing concise questions, with an open-ended option for the most critical questions to receive additional information or clarify the responses.

Lastly, our results may not be generalizable to the whole population of developers [5]. The main reason for this is our sample size of 105 respondents. However, as we discuss in Section 5 our results broadly align with findings from several previous studies. Thus, we believe that our results still provide valuable insights into the current landscape and can offer meaningful implications for organizations.

5 Related Work

We now overview the literature on security issues with generated code, the effects of generated code on developer productivity, and the perceptions of developers towards AI tools.

Security concerns about AI-generated code. Recent research highlights the security vulnerabilities associated with code generated by LLMs [2, 6, 15, 17, 20, 31, 32, 36, 38, 41, 49]. For instance, Pearce et al. [31] reported that Copilot produced insecure code in about 40% of cases, while Perry et al. [32] observed that AI-assisted developers introduced more vulnerabilities. Thus, our more experienced participants perhaps correctly perceive their proficiency in secure coding to degrade when using GAI tools.

Developers' perceptions of AI tools. Research demonstrated that developers' perceptions and experiences influence their actual productivity [33]. Thus, the literature on developers' perceptions regarding AI tools in software development provides an understanding of the impact of these tools on software engineering. One

important aspect that has been investigated is trust, as it plays a crucial role in shaping user interaction with AI [10, 25]. Thus, if developers do not trust AI-generated code, they are less likely to use it [16].

Several studies show that perceived utility and trust of AI tools contribute to users' intention to use them [3, 4, 50]. For example, Ge and Wu [16] found that trust and utility have a significant impact on the intention to use ChatGPT for bug fixing. Wang et al. [47] discovered that developers trust AI tools as these increase productivity and improve code quality, but some developers prefer to use AI tools only for simple tasks such as writing unit tests. Sergeyuk et al. [37] report that the main reasons why developers avoid using AI assistants are the lack of need, inaccuracy of AI-generated outputs, the lack of trust, and failure of AI assistants to understand the context. Indeed, several studies [23, 28, 30, 44] demonstrated that both ChatGPT and Copilot show a limited ability to understand the context and requirements, thus generating code containing errors and requiring additional debugging. Focusing on the usability of AI coding assistants, Liang et al. [24] found that, while developers appreciate AI assistants for reducing keystrokes and speeding up tasks, they face significant usability issues.

Despite the limitations of AI-generated code, academic and industry research show that developers prefer using AI tools in their daily programming tasks as it provides a helpful starting point [13, 40, 44] and can improve developer productivity by reducing the time spent on coding [48]. However, studies also highlight the problems associated with using AI-generated code, such as code correctness [49].

Regarding perceived productivity, Ziegler et al. [51] found that if developers frequently accept the Copilot suggestions, they tend to feel more productive. Mendes et al. [27] explored benefits and challenges related to GAI tools in a qualitative study, finding that increased productivity is the main perceived benefit.

The annual developer survey from Stack Overflow measures the usage of GAI tools among a large population of developers and evaluates developers' perceptions towards them [39]. Davila et al. [11] surveyed 72 developers in a Brazilian company regarding their GAI tools adoption. These surveys are broad and not geared toward security, unlike our study.

Developers' perceptions of security of AI-generated code. Kholoosi et al. [19] studied the perceptions of security professionals and software developers who shared their views on using ChatGPT for security tasks on Twitter (X). Vulnerability detection was the most discussed task, and users generally expressed positive sentiment towards this tool, though concerns about credibility and practicality persisted, especially for complex security tasks.

Klemmer et al. [21] reviewed 190 relevant Reddit posts and interviewed 27 software professionals to investigate how they use AI assistants in secure software development. Oh et al. [29] surveyed 238 developers (mostly students) to investigate how much they use code completion and code generation tools and whether they trust the security of the code produced by such tools. These studies confirm that, although there are security and quality concerns, software professionals widely use GAI tools. Some developers reported that they verify AI outputs similarly to human code [21]. This is in line with our findings, as our participants also reported an increased time spent on code review and stressed the importance of a thorough examination of the generated code.

Using a different instrument (quantitative survey) we come to some similar conclusions compared to [21], which used qualitative methods. Thus, our work independently confirms [21]. At the same time, our work has several important distinctions from [21, 29]. First, to the best of our knowledge, we are the first to observe the phenomenon when developers rather objectively (i.e., in line with their actual years of programming experience) assess their own proficiency in secure coding without AI assistants, but their self-assessment of proficiency with AI is practically disconnected with the objective experience measure. Second, our study also examined many other aspects: the behavioral changes that the participants self-reported when using GAI tools, the perceived benefits and risks that generated code entails, factors and practices contributing to code security, factors that contribute to a GAI tool being perceived as more secure, and the differences in developers' perceptions based on their experience.

6 Conclusions

We investigated how developers perceive the impact of AI-generated code on security and found that GAI tools can bring both benefits and risks. While developers reported faster coding, they also noted the need for additional time on security reviews and documentation of AI-generated code. And, despite the perceived efficiency gains from using GAI, there is a clear concern about the security of the generated code, with many experienced developers perceiving their secure coding proficiency to decrease when using AI. To address the security risks, developers recommend several best practices: thoroughly understanding AI-generated code before integration, using scanning tools to identify vulnerabilities, and avoiding sharing sensitive information with the GAI tool.

Our findings underscore the need for caution when integrating AI-generated code into the development process. While GAI tools positively impact developer productivity, the broader security implications must be addressed before these tools can be fully embraced. Building on our results, future research can investigate more in-depth how developers use and adapt to AI technologies.

Acknowledgments

This research has been partially supported by the Dutch Research Council (NWO) under the project NWA.1215.18.008 Cyber Security by Integrated Design (C-SIDe).

References

[1] Chittaranjan Andrade. 2020. The limitations of online surveys. *Indian journal of psychological medicine* 42, 6 (2020), 575–576.
[2] Owura Asare, Meiyappan Nagappan, and N Asokan. 2023. Is github's copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering* 28, 6 (2023), 129.
[3] Janarthanan Balakrishnan, Salma S Abed, and Paul Jones. 2022. The role of meta-UTAUT factors, perceived anthropomorphism, perceived intelligence, and social self-efficacy in chatbot-based services? *Technological Forecasting and Social Change* 180 (2022), 121692.
[4] Janarthanan Balakrishnan and Yogesh K Dwivedi. 2024. Conversational commerce: entering the next stage of AI-powered digital assistants. *Annals of Operations Research* 333, 2 (2024), 653–687.
[5] Sebastian Baltes and Stephan Diehl. 2016. Worse than spam: Issues in sampling software developers. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–6.
[6] Manish Bhatt, Sahana Chennabasappa, Cyrus Nikolaidis, Shengye Wan, Ivan Evtimov, Dominik Gabi, Daniel Song, Faizan Ahmad, Cornelius Aschermann, Lorenzo

Fontana, et al. 2023. Purple llama cyberseceval: A secure coding benchmark for language models. *arXiv preprint arXiv:2312.04724* (2023).
[7] Philip S Brenner and John DeLamater. 2016. Lies, damned lies, and survey self-reports? Identity as a cause of measurement bias. *Social psychology quarterly* 79, 4 (2016), 333–354.
[8] Matteo Ciniselli, Niccolò Puccinelli, Ketai Qiu, and Luca Di Grazia. 2024. From Today's Code to Tomorrow's Symphony: The AI Transformation of Developer's Routine by 2030. *arXiv preprint arXiv:2405.12731* (2024).
[9] Domenico Cotroneo, Cristina Improta, Pietro Liguori, and Roberto Natella. 2024. Vulnerabilities in ai code generators: Exploring targeted data poisoning attacks. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. 280–292.
[10] Arun Das and Paul Rad. 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371* (2020).
[11] Nicole Davila, Igor Wiese, Igor Steinmacher, Lucas Lucio da Silva, André Kawamoto, Gilson José Peres Favaro, and Ingrid Nunes. 2024. An Industry Case Study on Adoption of AI-based Programming Assistants. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 92–102.
[12] Elisabeth Deutskens, Ko De Ruyter, Martin Wetzels, and Paul Oosterveld. 2004. Response rate and response quality of internet-based surveys: An experimental study. *Marketing letters* 15 (2004), 21–36.
[13] Thomas Dohmke, Marco Iansiti, and Greg Richards. 2023. Sea change in software development: Economic and productivity analysis of the AI-powered developer lifecycle. *arXiv preprint arXiv:2306.15033* (2023).
[14] Mark Damian Duda and Joanne L Nobile. 2010. The fallacy of online surveys: No data are better than bad data. *Human Dimensions of Wildlife* 15, 1 (2010), 55–64.
[15] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, and Jiaxin Yu. 2023. Security weaknesses of copilot generated code in github. *arXiv preprint arXiv:2310.02059* (2023).
[16] Haotong Ge and Yuemeng Wu. 2023. An Empirical Study of Adoption of ChatGPT for Bug Fixing among Professional Developers. *Innovation & Technology Advances* 1, 1 (2023), 21–29.
[17] Stefan Goetz and Andreas Schaad. 2024. " You still have to study"–On the Security of LLM generated code. *arXiv preprint arXiv:2408.07106* (2024).
[18] Begum Karaci, Deniz Chandra Gnanasambandam, Martin Harrysson, Alharith Hussin, and Shivam Srivastava. 2019. Unleashing developer productivity with generative AI. (2019). Available at https://digital-strategy.ec.europa.eu/en/library/communication-building-trust-human-centric-artificial-intelligence.
[19] M Mehdi Kholoosi, M Ali Babar, and Roland Croft. 2024. A Qualitative Study on Using ChatGPT for Software Security: Perception vs. Practicality. *arXiv preprint arXiv:2408.00435* (2024).
[20] Raphaël Khoury, Anderson R Avila, Jacob Brunelle, and Baba Mamadou Camara. 2023. How secure is code generated by chatgpt?. In *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2445–2451.
[21] Jan H Klemmer, Stefan Albert Horstmann, Nikhil Patnaik, Cordelia Ludden, Cordell Burton Jr, Carson Powers, Fabio Massacci, Akond Rahman, Daniel Votipka, Heather Richter Lipford, et al. 2024. Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns.
[22] Arina Kudriavtseva, Nisar Ahmad Hotak, and Olga Gadyatskaya. 2024. Questionnaire for the survey. https://doi.org/10.5281/zenodo.14524893
[23] Mohammad Amin Kuhail, Sujith Samuel Mathew, Ashraf Khalil, Jose Berengueres, and Syed Jawad Hussain Shah. 2024. "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming* 235 (2024), 103111.
[24] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2024. A large-scale survey on the usability of AI programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
[25] Q Vera Liao and S Shyam Sundar. 2022. Designing for responsible trust in AI systems: A communication perspective. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 1257–1268.
[26] Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. 2015. Guidelines for conducting surveys in software engineering. (2015).
[27] Wendy Mendes, Samara Souza, and Cleidson De Souza. 2024. "You're on a bicycle with a little motor": Benefits and Challenges of Using AI Code Assistants. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*. 144–152.
[28] Nikolaos Nikolaidis, Karolos Flamos, Daniel Feitosa, Alexander Chatzigeorgiou, and Apostolos Ampatzoglou. 2023. The End of an Era: Can AI Subsume Software Developers? Evaluating ChatGPT and Copilot Capabilities Using Leetcode Problems. *SSRN* (2023).
[29] Sanghak Oh, Kiho Lee, Seonhye Park, Doowon Kim, and Hyoungshick Kim. 2024. Poisoned GhatGPT finds work for idle hands: Exploring developers' coding practices with insecure suggestions from poisoned AI models. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1141–1159.
[30] Ruchika Pandey, Prabhat Singh, Raymond Wei, and Shaila Shankar. 2024. Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects. *arXiv preprint arXiv:2406.17910* (2024).
[31] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 754–768.
[32] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2785–2799.
[33] Abdul Razzaq, Jim Buckley, Qin Lai, Yun Ting, and Goetz Botterweck. 2024. A Systematic Literature Review on the Influence of Enhanced Developer Experience on Developers' Productivity: Factors, Practices, and Recommendations. *Comput. Surveys* (2024).
[34] Daniel Russo. 2024. Navigating the complexity of generative ai adoption in software engineering. *ACM Transactions on Software Engineering and Methodology* (2024).
[35] Siva Sai, Utkarsh Yashvardhan, Vinay Chamola, and Biplab Sikdar. 2024. Generative ai for cyber security: Analyzing the potential of chatgpt, dall-e and other models for enhancing the security space. *IEEE Access* (2024).
[36] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at c: A user study on the security implications of large language model code assistants. In *32nd USENIX Security Symposium (USENIX Security 23)*. 2205–2222.
[37] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2024. Using AI-Based Coding Assistants in Practice: State of Affairs, Perceptions, and Ways Forward. *arXiv preprint arXiv:2406.07765* (2024).
[38] Mohammed Latif Siddiq and Joanna CS Santos. 2022. SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*. 29–33.
[39] Stack Overflow. 2024. Developer Survey 2024. https://survey.stackoverflow.co/2024/
[40] Maxim Tabachnyk, Stoyan Nikolov, et al. 2022. Ml-enhanced code completion improves developer productivity. *Google Research Blog. July* 26 (2022).
[41] Maryam Taeb, Hongmei Chi, and Shonda Bernadin. 2024. Assessing the Effectiveness and Security Implications of AI Code Generators. In *Journal of The Colloquium for Information Systems Security Education*, Vol. 11. 6–6.
[42] Sumanth Tatineni. 2024. *Integrating Artificial Intelligence with DevOps: Advanced Techniques, Predictive Analytics, and Automation for Real-Time Optimization and Security in Modern Software Development*. Libertatem Media Private Limited.
[43] Rebeka Tóth, Tamas Bisztray, and László Erdődi. 2024. LLMs in Web Development: Evaluating LLM-Generated PHP Code Unveiling Vulnerabilities and Limitations. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 425–437.
[44] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI conference on human factors in computing systems extended abstracts*. 1–7.
[45] Jibin Rajan Varghese and Divya Susan Thomas. 2024. Survey of Generative AI in Code Generation: Privacy, Security and Ethical Considerations. (2024).
[46] Jiexin Wang, Xitong Luo, Liuwen Cao, Hongkui He, Hailin Huang, Jiayuan Xie, Adam Jatowt, and Yi Cai. 2024. Is Your AI-Generated Code Really Secure? Evaluating Large Language Models on Secure Code Generation with CodeSecEval. *arXiv preprint arXiv:2407.02395* (2024).
[47] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. 2024. Investigating and designing for trust in ai-powered code generation tools. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*. 1475–1493.
[48] Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. 2024. Significant Productivity Gains through Programming with Large Language Models. *Proceedings of the ACM on Human-Computer Interaction* 8, EICS (2024), 1–29.
[49] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *arXiv preprint arXiv:2304.10778* (2023).
[50] Xiao Yu, Lei Liu, Xing Hu, Jacky Wai Keung, Jin Liu, and Xin Xia. 2024. Fight Fire with Fire: How Much Can We Trust ChatGPT on Source Code-Related Tasks? *arXiv preprint arXiv:2405.12641* (2024).
[51] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 21–29.