Trade-Offs Between Efficiency and Security in AI-Assisted Software Development

1st M. Ali Elhasan
Department of Computer Science and Engineering
Gothenburg, Sweden
guselhmu@gu.se

2th Zakaria Aslan
Department of Computer Science and Engineering
Gothenburg, Sweden
gusxxx@gu.se

3th Yousef Abbas
Department of Computer Science and Engineering
Gothenburg, Sweden
gusxxx@gu.se

***Abstract*—Generative AI tools like GitHub Copilot enhance software development productivity but introduce significant security vulnerabilities ...**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

While AI-assisted coding tools such as GitHub Copilot and ChatGPT have demonstrated productivity benefits [1], [2], security concerns remain a significant challenge [3], [4]. Existing research has explored different aspects of AI-generated code, including productivity impact [1], [2], security risks [3], [4], and how developers interact with these tools [5]. However, a major gap in the literature lies in understanding how developers balance efficiency and security concerns when using AI-assisted coding tools, as well as evaluating the effectiveness of security verification practices in mitigating vulnerabilities.

Several studies have investigated the security weaknesses of AI-generated code. Fu et al. [3] and Perry et al. [4] found that AI-generated code often introduces common vulnerabilities such as SQL injection and improper authentication, while Asare et al. [6] examined how AI-generated code compares to human-written code in terms of security flaws. Meanwhile, Peng and Weisz [1], [2] have highlighted the productivity gains associated with AI-assisted coding, showing that developers can complete tasks more efficiently with these tools. Despite these insights, existing research largely treats security and productivity as separate concerns, without examining how developers perceive or navigate the trade-offs between them.

Additionally, recent research has assessed how developers interact with AI tools. The observational study by Khojah et al. [5] found that developers often use AI assistants for guidance rather than copying code verbatim, suggesting that AI tools influence coding decisions beyond just code generation. Similarly, the study by Perry et al. [4] investigated whether AI usage leads to more insecure code but did not explore developer awareness of these risks or their strategies for mitigation.

Human review remains a fundamental line of defense for all code, including AI-generated snippets. However, with the rapid pace at which AI tools can generate code, is manual verification still a practical and efficient approach? Does the speed of AI-generated output outpace the ability of security teams to audit and validate it effectively? Or have other verification tools and practices been developed that can match the speed of code generation while maintaining the security and reliability of human review?

Our study aims to bridge these gaps by addressing the following research questions:

- **RQ1**: How do developers perceive the trade-off between speed in writing code and security when using AI tools?
- **RQ2:** What are the existing security verification practices for AI-generated code in web applications?
  - **RQ2.1:** How effective are the existing security verification practices for AI-generated code in web applications?

By combining developer surveys and literature review with a security analysis of AI-generated code, this research seeks to provide a more comprehensive understanding of how developers balance efficiency and security. Additionally, it evaluates whether current security verification methods are sufficient for detecting vulnerabilities in AI-generated code, offering insights into potential improvements for secure AI-assisted software development. Study conclusion [bla bla bla]

II. Related Work

A. *Impact of AI on Developer Code Writing Speed*

AI-powered coding assistants have been widely adopted, with studies suggesting significant productivity benefits. Microsoft Research found that developers using GitHub Copilot completed tasks 55.8% faster than those coding manually [1]. IBM's study on AI pair programming in an enterprise setting observed perceived productivity gains but noted variability based on task complexity and developer experience [2].

B. Security Risks in AI-Generated Code

Security concerns in AI-generated code have been extensively documented. One of the first empirical evaluations of GitHub Copilot revealed that approximately 40% of AI-generated code contained vulnerabilities, including SQL injection and buffer overflow issues [3]. Perry et al. found that AI-assisted developers wrote significantly more insecure code than those without AI assistance, despite being more confident in their solutions [4]. Additionally, Asare et al. demonstrated that AI-generated code often replicated past vulnerabilities, repeating the same mistakes found in historical software security incidents [6].

These findings suggest that while AI tools accelerate development, they also introduce new security risks that must be mitigated. After these studies were conducted, a study by Peslak and Kovalchick in 2024 analyzed data from the Stack Overflow Annual Developer Survey to assess the usage patterns of ChatGPT and GitHub Copilot among programmers. The findings revealed that 58.8% of respondents regularly used ChatGPT, while 24.8% utilized GitHub Copilot [7]. This confirms that a significant number of developers use AI tools to generate code, even when evidence indicates that AI-generated code contains security vulnerabilities and risks.

C. Developer Adoption and Perceptions

Despite security risks, the adoption of AI coding assistants has grown rapidly. A 2023 survey by GitHub reported that over 70% of developers believed Copilot helped them stay focused and avoid mental fatigue [8]. However, Snyk's industry report found that while 75% of developers believed AI-generated code was more secure than human-written code, 56% of respondents admitted to encountering security issues in AI-generated suggestions [9].

Additionally, empirical research further emphasizes this security concern. A study evaluating the security of GitHub Copilot's code contributions found that approximately 40% of generated code contained security vulnerabilities, including common weaknesses such as SQL injection and improper input validation. The likelihood of generating vulnerable code was influenced by the context and specificity of the prompts provided to the AI tool, as well as the type of programming language and the domain of the coding task [10].

These findings collectively indicate a potential overconfidence bias in AI-assisted development, where developers may unknowingly trust insecure AI-generated code, underlining the critical importance of careful review and validation practices.

D. Research Gaps

Despite existing research providing insights into productivity enhancements and security vulnerabilities associated with AI-assisted coding tools, several critical gaps remain. Firstly, current literature primarily addresses productivity improvements and security risks independently, with limited exploration into the inherent trade-offs developers face between coding speed and security when leveraging AI. There is insufficient research on developer perceptions and decision-making processes regarding these trade-offs, particularly how developers balance the immediate productivity benefits against the potential introduction of security vulnerabilities. Secondly, the effectiveness of existing security verification practices specifically tailored to AI-generated code has not been adequately investigated. While traditional verification methods are well-established for manually-written code, their effectiveness and adaptability for assessing AI-generated code remain unclear. Given the unique vulnerabilities introduced by AI coding assistants, research into specialized verification and validation methodologies is crucial. Addressing these gaps will enable the development of comprehensive guidelines and security frameworks, supporting developers in effectively utilizing AI tools without compromising application security.

III. Research Design and Methodology

This study employs a mixed-methods approach, integrating both quantitative and qualitative research techniques to investigate the trade-offs between productivity and security in AI-assisted software development. By combining developer surveys, literature review, and code security analysis, the study aims to quantify and contextualize the security risks associated with AI-generated code while assessing their impact on development efficiency.

A. Hypotheses

To systematically investigate the research questions, the study formulates the following hypotheses:

1) For RQ1 (Perception of Trade-Offs):

- **H_0 (Null Hypothesis)**: Developers do not perceive a significant trade-off between speed and security when using AI-assisted coding tools.
- **H_1 (Alternative Hypothesis)**: Developers perceive a significant trade-off between speed and security when using AI-assisted coding tools.

2) For RQ2 (What are the existing security verification practices):

- **H_0 (Null Hypothesis)**: There are no existing security verification methods designed to operate at a speed comparable to the rapid generation of AI-generated code.
- **H_1 (Alternative Hypothesis)**: There are security verification methods specifically designed to verify AI-generated code at a pace that matches the rapid speed of its generation.

3) For RQ2.1 (How effective are the existing security verification practices):

- **H_0 (Null Hypothesis)**: Existing security verification practices are not effective in identifying vulnerabilities in AI-generated code.
- **H_1 (Alternative Hypothesis)**: Existing security verification practices are effective in identifying vulnerabilities in AI-generated code.

B. Research Method

1) Developer Survey: The survey will target 40-50 software developers, including 20 web developers, 5 test engineers, 5 DevOps engineers, and 10 backend developers, who actively use AI coding tools. It is designed to capture developers' perspectives on trade-offs between development speed and security risks, as well as their mitigation strategies.

Survey participants will be asked to indicate their level of agreement with various statements using a 5-point Likert scale, supplemented with open-ended questions for qualitative insights.

a) Survey Statements::

- AI tools help me write code faster, but I worry they introduce security risks.
- The time saved by using AI tools outweighs the effort required to fix security issues in AI-generated code.
- I prioritize code security over development speed when using AI tools.
- AI tools make it harder to follow secure coding practices (e.g., input sanitization, secure authentication).
- I feel pressured to deliver code quickly when using AI tools, even if it means compromising security.

b) Follow-Up Questions::

- What percentage of time saved by AI tools is spent fixing security issues?
- When using AI tools, how often do you review generated code for security flaws?
- Describe a situation where you had to choose between development speed and code security while using AI tools.

These responses will help quantify perceptions of security risks, assess common mitigation practices, and evaluate whether AI-generated efficiency gains are offset by additional security verification work.

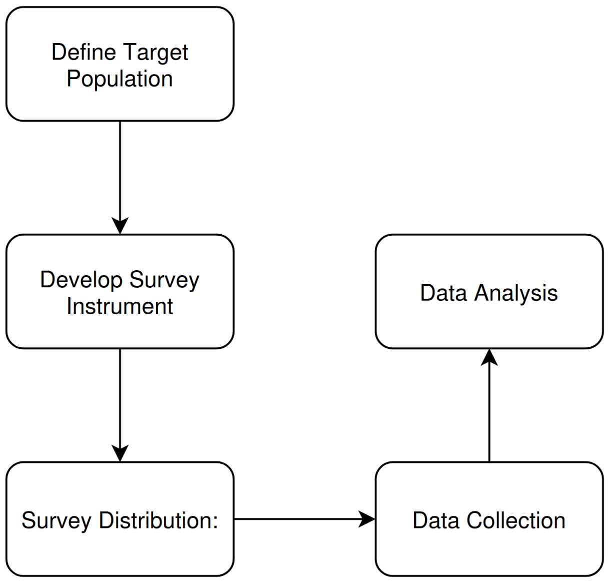


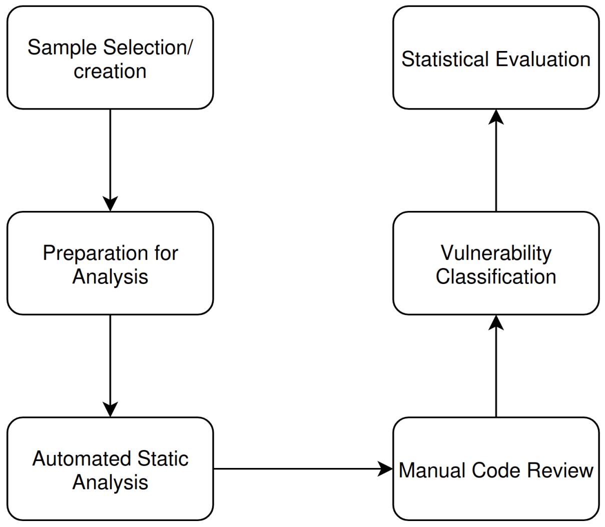
Fig. 1. Workflow for the process of thesis's Developer Survey Research Method.

2) Security Verification: Landscape Review: To address the question of what security verification practices currently exist for AI-generated code, this section will review academic publications, industry reports, and documentation from security tools. The goal is to map out the landscape of available methods, examining their design, purpose, and level of automation. The review will pay particular attention to whether these methods were developed with AI-generated code in mind, and how they are integrated into modern development workflows. It will also highlight any gaps in existing practices such as overreliance on post-generation manual review that could hinder the secure adoption of AI coding tools. This investigation will provide the necessary context for evaluating the novelty and limitations of current security approaches in relation to the rapid pace of AI code generation.

3) Code Analysis: We will compile a dataset of 9-13 AI-generated code samples by utilizing various AI coding tools and sourcing existing publicly online available examples across multiple programming languages (e.g., Python, JavaScript, Java) and use cases (e.g., web development, API security, cryptographic functions). The selected samples will represent realistic coding scenarios to facilitate an in-depth security analysis. The methodology for security analysis includes:

- **Static Analysis** – Using industry-standard tools (e.g., SonarQube, Bandit, ESLint Security Plugin) to detect potential vulnerabilities automatically.
- **Manual Code Review** – Conducting expert analysis of AI-generated code to identify security flaws missed by automated tools, using OWASP guidelines and secure coding principles.
- **Severity Classification** – Categorizing vulnerabilities based on severity (low, medium, high) and type (e.g., SQL injection, insecure authentication, hardcoded secrets).

By examining patterns in security flaws and comparing them with survey responses, this analysis provides a direct measure of security trade-offs and how developers perceive and mitigate AI-generated risks.



Sample Selection/ creation
Preparation for Analysis
Automated Static Analysis
Manual Code Review
Vulnerability Classification
Statistical Evaluation

Fig. 2. Workflow for the process of thesis's Code Analysis Research Method.

4) Code Sample Selection: To ensure a balanced and representative dataset, code samples will be selected using a combination of randomized and purposeful sampling methods, adhering the following selection criteria:

- **Programming Languages**: JavaScript, Java, c++, Python.
- **Types of Applications**: Web apps, APIs, cryptographic functions, data processing, authentication systems.
- **AI Tools Used**: GitHub Copilot, ChatGPT, AWS CodeWhisperer, Google Gemini.
- **Complexity**: Moderate complexity, code snippet, standalone components.
- **Randomized selection (70% of the samples, 5-7 samples)**: Randomly generated code snippets from common developer communities such as GitHub, Stack Overflow, AI code generation examples publicly shared by developers, and our own AI-generated examples using standard prompts, which if happened the prompts and conversation will be available to view.
- **Purposeful selection (30% of your samples, 4-6 samples)**: Intentionally selecting a few samples:
  - Known faulty examples: AI-generated code publicly flagged for vulnerabilities or security issues.
  - Known clean examples: AI-generated code recognized or peer-reviewed as secure or adhering closely to security best practices.
  - Popular scenarios from well-documented sources to represent standard industry practices (e.g., OWASP-related implementations).

A more detailed structure specifying the functionality, corresponding test cases, the version of the AI generation tool used, and the total number of samples for each programming language and AI tool will be provided at a later stage.

Justification of sample selection The dataset should represent realistic scenarios that reflect common developer practices and include but are not limited to languages commonly used, not just extremes of faulty or perfect code. Randomized Selection: Prevents bias, reflecting general AI usage realistically. Purposeful Selection: Captures extremes for detailed comparison and identifying recurring patterns or exceptions in vulnerabilities.

TABLE I
SAMPLE SELECTION CRITERIA

Criteria	**Number of samples**
Randomly selected	5-7
Intentionally faulty	2-3
Intentionally clean	2-3
Total samples	9-13
Programming Languages	At least 3 (Python, JS, Java)
AI Generation Tools	At least 2 (Copilot, ChatGPT)

C. Data Analysis

1) Survey Data: Survey responses will be analyzed using statistical methods, including:

- **Descriptive Statistics**: Mean, median, and standard deviation will be calculated for Likert-scale responses to summarize general trends in perceptions of AI-generated code security and efficiency.
- **Correlation Analysis**: Pearson or Spearman correlation tests will be used to identify relationships between developers' concerns about security risks and their reported efficiency gains.
- **Comparative Analysis**: Independent t-tests or ANOVA will be applied to compare responses across different groups (e.g., experience levels, programming languages used) to assess variation in perceptions.
- **Thematic Analysis**: Open-ended responses will be categorized using thematic coding to extract qualitative insights on developers' security concerns and mitigation strategies.

a) Motivation for Methods:: These statistical techniques allow for a robust examination of trends and relationships in the data. Descriptive statistics provide an overview of general trends, while correlation and comparative analyses help identify patterns in how different developer subgroups perceive AI tool trade-offs. Thematic analysis ensures that qualitative responses add depth to the statistical findings.

b) Addresses RQ1:: The survey data will directly reflect how developers perceive the security-efficiency trade-off, revealing trends in how security concerns impact AI tool adoption and coding behavior.

2) Literature Review Findings: The literature review findings will be synthesized through qualitative analysis, focusing on identifying patterns, gaps, and emerging trends in security verification practices related to AI-generated code. Key themes such as automation, scalability, and tool adaptability will be extracted and categorized. The review will also compare the characteristics of modern verification methods with traditional approaches to determine whether current practices are evolving in response to the growing use of AI in software development.

a) Motivation for Methods:: Thematic synthesis enables a structured evaluation of diverse sources, allowing for the identification of recurring concepts and underexplored areas. This approach supports an informed assessment of whether existing verification practices are adequate for the pace and nature of AI-generated code.

b) Addresses RQ2:: The literature review findings will clarify what verification methods are currently in use, whether they are tailored to AI-generated code, and to what extent developers continue to depend on slower, manual security checks. This directly informs the investigation of existing practices in secure AI-assisted development.

3) Code Security Evaluation: The identified vulnerabilities will be categorized based on severity, type, and frequency to assess the effectiveness of current security verification methods. This analysis will provide insights into whether AI-generated code introduces systematic risks and how they align with developer concerns reported in the survey. The security analysis will include:

- **Frequency Analysis**: Counting the occurrence of specific vulnerability types across AI-generated code samples.
- **Severity Distribution**: Categorizing vulnerabilities by severity (e.g., low, medium, high) to assess the risk posed by AI-generated code.
- **Comparative Evaluation**: Comparing AI-generated vulnerabilities with known industry benchmarks or previous research studies on AI-assisted coding security.

a) Motivation for Methods:: These techniques provide an empirical foundation for assessing AI-generated code security. Frequency analysis identifies recurring risks, severity classification contextualizes the impact of these vulnerabilities, and comparative evaluation allows benchmarking against existing security standards.

b) Addresses RQ2.1:: The security evaluation will measure the effectiveness of existing security verification methods for AI-generated code, helping to assess whether these practices are sufficient for mitigating security risks in real-world applications.

References

[1] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," 2023. [Online]. Available: https://arxiv.org/abs/2302.06590

[2] J. D. Weisz, S. Kumar, M. Muller, K.-E. Browne, A. Goldberg, E. Heintze, and S. Bajpai, "Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise," 2025. [Online]. Available: https://arxiv.org/abs/2412.06603

[3] Y. Fu, P. Liang, A. Tahir, Z. Li, M. Shahin, J. Yu, and J. Chen, "Security weaknesses of copilot-generated code in github projects: An empirical study," 2025. [Online]. Available: https://arxiv.org/abs/2310.02059

[4] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Nov. 2023, pp. 2785–2799. [Online]. Available: http://dx.doi.org/10.1145/3576915.3623157

[5] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto, "Beyond code generation: An observational study of chatgpt usage in software engineering practice," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, 2024. [Online]. Available: https://doi.org/10.1145/3660788

[6] O. Asare, M. Nagappan, and N. Asokan, "Is github's copilot as bad as humans at introducing vulnerabilities in code?" 2024. [Online]. Available: https://arxiv.org/abs/2204.04741

[7] A. Peslak and L. Kovalchick, "Ai for coders: An analysis of the usage of chatgpt and github copilot," *Issues in Information Systems*, vol. 25, no. 4, pp. 252–260, 2024.

[8] G. Blog, "Github copilot: A survey of developers," 2023. [Online]. Available: https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/

[9] Snyk, "Ai-generated code leads to security issues for most businesses: Report," *CIO Dive*, 2023, [Online]. Available: https://www.ciodive.com/news/security.

[10] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," 2021. [Online]. Available: https://arxiv.org/abs/2108.09293