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

Muhammed Ali Elhasan, Yousef Abbas, Zakaria Aslan
Department of Computer Science, University of Gothenburg

***Abstract*—Generative AI tools like GitHub Copilot enhance software development productivity but introduce significant security vulnerabilities. This study investigates developer perceptions of the trade-offs between coding efficiency and security risks, utilizing surveys and code security analysis. The outcomes will inform practical guidelines for balancing rapid AI-assisted development with robust security practices.**

***Index Terms*—Generative AI, Software Security, Productivity, AI-Assisted Development.**

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. Additionally, research by Peng and Weisz [1], [2] has highlighted the productivity gains associated with AI-assisted coding, suggesting that developers complete tasks more efficiently when using these tools. However, while these studies provide valuable insights, they largely treat security risks and productivity benefits as separate issues rather than examining how developers perceive and navigate the trade-offs between them.

Furthermore, 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 developers write more insecure code when using AI, but it did not explore whether developers are aware of these risks or how they attempt to mitigate them. Additionally, while prior studies have analyzed security vulnerabilities in AI-generated code [3], [4], [6], there is limited research evaluating the effectiveness of security verification methods for AI-assisted development.

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? (Perception)
- **RQ2**: How effective are existing security verification practices for AI-generated code in web applications?

By combining developer surveys with security analysis of AI-generated code, this research will provide a more comprehensive understanding of how developers navigate the balance between efficiency and security. Additionally, it will evaluate whether existing security verification methods are adequate for detecting vulnerabilities in AI-generated code, offering insights into potential improvements in secure AI-assisted software development.

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 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.

The survey data, analyzed using descriptive statistics, correlation tests, and comparative analysis, will determine whether developers consistently report concerns about security risks in AI-assisted coding.

2) For RQ2 (Effectiveness of Security Verification):

- **H_0 (Null Hypothesis)**: Existing security verification methods are sufficient in detecting and mitigating security vulnerabilities in AI-generated code.
- **H_1 (Alternative Hypothesis)**: Existing security verification methods are insufficient in detecting and mitigating security vulnerabilities in AI-generated code.

The security analysis will assess the frequency, severity, and type of vulnerabilities found in AI-generated code. Statistical evaluations, including frequency analysis and comparative benchmarking against industry security standards, will help test this hypothesis.

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.

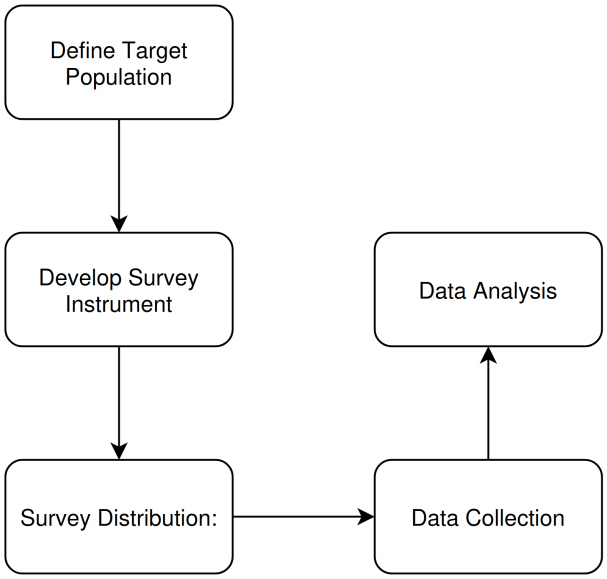

Fig. 1. Workflow for the process of thesis's Developer Survey Research Method.

2) Code Analysis: We will compile a dataset of 25-30 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.

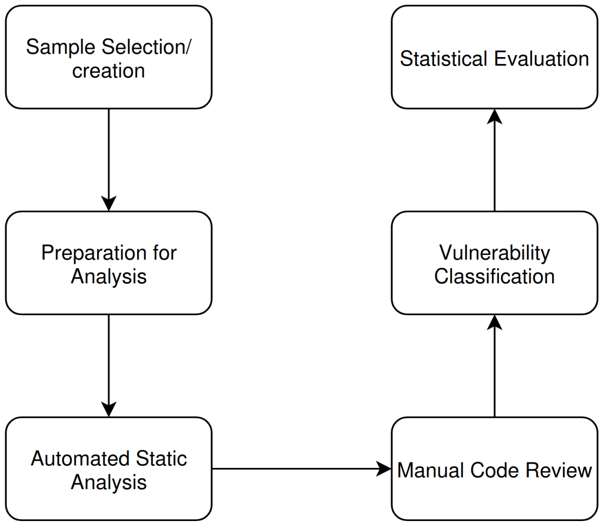
Sample Selection/ creation
Preparation for Analysis
Automated Static Analysis
Manual Code Review
Vulnerability Classification
Statistical Evaluation

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

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) 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:: 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.

Exam Questions

Q1: You are supposed to explore the uses of generative AI in software engineering education via a survey. Write 5 survey questions that would get the core of that content and explain why they cover the research question well. (5 questions plus about 200 words of explanation.)

Survey Question:

Q1: How often do you use Generative AI (For example, ChatGPT, Gemini ....) in your studying as a software engineering student/lecturer?

Explanation: The first question focuses on identifying the groups using Generative AI tools in software engineering (students, teachers, or professionals). We want to understand whether these tools are becoming part of the education process and how often they are used in studying or solving daily tasks. Frequent use can indicate that users find them helpful, potentially filling gaps in current educational practices. We also aim to analyze the acceptance and trust in AI tools over time and determine whether educational institutions are open to these technologies or still cautious.

Q2: What type of tasks do you usually use Generative AI for in software engineering education? (For example: coding, debugging, writing reports, gathering requirements ....)

Explanation: This question explores which tasks are most commonly supported by Generative AI. It helps identify areas where users feel the most need for assistance, such as when stuck in development or needing help with ideation. It reveals whether AI acts more as a learning assistant, a coding partner, or something else.

Q3: When you use generative AI, do you feel like it actually helps you learn or improves your understanding? Why or why not?

Explanation: This question assesses whether AI tools contribute to actual learning or are simply used for completing tasks. It aims to uncover whether users are thinking deeply or becoming overly dependent. The responses will help evaluate if AI is enhancing problem-solving abilities or hindering genuine understanding.

Q4: While doing an assignment or project, what is the instructor's opinion of using AI tools such as ChatGPT, Gemini, etc.?

Explanation: This question investigates educators' perspectives on Generative AI. Are they supportive or skeptical? Understanding the institution's attitude can help determine if AI use is accepted or discouraged. It also exposes any discrepancies between student and instructor viewpoints.

Q5: What concerns do you have when you use Generative AI in your Software Engineering learning?

Explanation: This question uncovers potential risks and concerns such as dependence, cheating, lack of understanding, or buggy AI-generated code. It gives a more comprehensive view of user perspectives and helps balance the benefits and drawbacks of AI in software engineering education.

Q2: When designing surveys, which different types of sampling exist? Name at least three types of sampling and give one concrete example for each (not probabilistic and non-probabilistic is too high-level, be more specific).

Choosing the right sampling method when designing surveys depends on the aim of the research, the goal, the size of the population, and the available time and access. Below are three specific types of sampling:

Simple Random Sampling: This is used to get a fair representation of a large population.
Example: If we have access to a student list across departments, we randomly pick 100 students to survey about their use of Generative AI. Each student has an equal chance, which makes results more generalizable.

Stratified Sampling: Used when comparing specific subgroups.
Example: To compare AI use between Computer Science and Science students, we divide them into those two groups and sample each separately to avoid overrepresenting one group.

Convenience Sampling: Useful when there is limited time or access.
Example: In a bachelor thesis, we ask our classmates to fill a survey on efficiency vs. security in AI tools. It's quick but may lack broader accuracy.

Q3: Imagine an experimental setup evaluating the effects of developer experience on program quality.

In this experiment, the aim is to measure how developer experience influences software quality.

Independent Variable: Developer experience level.
Levels:

- Low experience (¡2 years full time)
- Medium experience (2–5 years)
- High experience (¿5 years)

Dependent Variable: Program quality, measured by bug count, performance, code readability, or alignment with requirements.

Objects: Programming tasks assigned to all participants.

Subjects: Developers performing the tasks.

Control Variables:

- Same programming language/framework
- Same task complexity
- Same time allocation
- Same development environment

Q4: As part of your planned thesis work, what sort of ethical issues may arise and why? What are you planning to do to mitigate ethical issues?

Our thesis, "Trade-offs between efficiency and security in AI-assisted software development," may encounter several ethical issues. The main one is collecting data from real developers, which involves gathering sensitive information, such as personal opinions on tools and security practices.

To mitigate this, we will ensure:

- Clear informed consent from participants.
- Anonymous and confidential handling of responses.
- Secure storage of all data.
- Transparent information about how the data will be used.

This protects participant privacy and meets ethical standards.

Q5: Design Science and Action Research have several similarities, but also some key differences.

Similarities:

- Both aim to solve practical, real-world problems.
- Both are iterative and involve evaluation and improvement.
- Both engage users or stakeholders in the process.

Differences:

- **Design Science (DS):** Focuses on creating and evaluating artifacts like models, systems, or tools. Its goal is to contribute to knowledge by designing useful innovations.
- **Action Research (AR):** Aims to solve issues within a specific organization or context. It requires active participation from researchers and stakeholders and emphasizes reflection and change within the setting.

DS is more about building and proving the utility of something new, while AR is about improving existing situations through collaborative cycles.

IV. Acknowledgements

This study was a collaborative effort by **Muhammed Ali Elhasan, Yousef Abbas, and Zakaria Aslan**. Muhammed led the methodology and data analysis sections, Yousef contributed to the related work and research questions, and Zakaria handled the introduction and discussion of research gaps. We acknowledge the valuable feedback from our research advisors and colleagues who provided insights that helped refine our study.

References

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

[2] J. D. Weisz, S. Kumar, M. Muller, *et al.*, *Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise*, 2025. arXiv: 2412.06603 [cs.HC]. [Online]. Available: https://arxiv.org/abs/2412.06603.

[3] Y. Fu, P. Liang, A. Tahir, *et al.*, *Security weaknesses of copilot-generated code in github projects: An empirical study*, 2025. arXiv: 2310.02059 [cs.SE]. [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. DOI: 10.1145/3576915.3623157. [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. DOI: 10.1145/3660788. [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. arXiv: 2204.04741 [cs.SE]. [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. arXiv: 2108.09293 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2108.09293.