The Influence of AI on

Software Development and Education

Harvest Ecclesiano Christ Walukow1   
  
*1* *Faculty of Advanced Technology and Multidiscipline, Universitas Airlangga, Surabaya, Indonesia*

*Abstract*—The advent of Artificial Intelligence (AI), particularly Large Language Models (LLMs) such as ChatGPT and GitHub Copilot, marks a profound shift across various sectors, most notably software development and education. This paper presents a comprehensive literature review examining the multifaceted influence of AI on these two critical domains. In software development, AI has ushered in a new era of AI-assisted programming (AIAP), augmenting developer productivity, enhancing code quality, and facilitating tasks from code generation to bug fixing and testing. However, its integration also presents challenges related to reliability, intellectual property, potential deskilling, and the imperative for new development paradigms. Concurrently, in education, AI offers unprecedented opportunities for personalized learning, adaptive assessment, and curriculum innovation, while also necessitating a re-evaluation of pedagogical approaches. Ethical and societal implications, including academic integrity, data privacy, algorithmic bias, and the digital divide, are central concerns. This review synthesizes current research, highlights validated findings, discusses prevailing challenges, and forecasts future trends, underscoring the urgent need for thoughtful adaptation and policy development to harness AI's transformative potential responsibly in both software engineering and educational practices.

*Keywords*—Artificial Intelligence, Code, Software Development, Education

# INTRODUCTION

T

he rapid evolution and widespread accessibility of Artificial Intelligence (AI), particularly Large Language Models (LLMs) like ChatGPT, GitHub Copilot, and Google Gemini, are fundamentally reshaping industries and societal structures. This transformative impact is acutely felt in two interconnected fields: software development and education. In software engineering, AI-powered tools are moving beyond theoretical concepts to become indispensable assistants, promising to revolutionize how software is designed, implemented, tested, and maintained [14]. Simultaneously, the educational landscape is experiencing significant disruption, as AI tools offer new avenues for learning and assessment while raising profound questions about academic integrity, skill development, and societal preparedness [13].

This paper aims to provide a comprehensive literature review of the influence of AI on software development and education. It will explore the opportunities AI presents, the challenges it introduces, and the future trajectories anticipated in both domains. By drawing on recent empirical studies and expert

perspectives, this review seeks to offer an objective and analytical overview of this fast-evolving phenomenon, highlighting key findings, areas of consensus, and persistent research gaps. The discussion will be structured into two main parts: AI's impact on software development workflows and its implications for educational practices, followed by a synthesis of common themes and a look into future directions.

# Material and Methods

To comprehensively analyze the profound influence of AI on software development and education, this study conducted a thorough search for relevant academic journals.

Table 1. Overview of reviewed literature.

|  |  |  |
| --- | --- | --- |
| **No** | **Title** | **Summary** |
| 1 | Programming Is Hard — Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation | Explores the transformative impact of AI-driven code generation tools on introductory programming education |
| 2 | Need Help? Designing Proactive AI Assistants for Programming | Explores the development and impact of proactive AI assistants for programming |
| 3 | 'It would work for me too': How Online Communities Shape Software Developers’ Trust in AI-Powered Code Generation Tools | Investigates how online communities influence software developers' trust in AI-powered code generation tools |
| 4 | Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks | Investigates targeted data poisoning attacks on AI code generators |
| 5 | Prompt Problems: A New Programming Exercise for the Generative AI Era | Introduces "Prompt Problems" as a novel approach to teaching programming in the age of generative AI |
| 6 | My AI Wants to Know if This Will Be on the Exam: Testing OpenAI’s Codex on CS2 Programming Exercises | Investigates the performance of OpenAI's Codex on computer science programming tasks |
| 7 | The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming | Investigates the capabilities of OpenAI's Codex, an advanced AI model capable of generating computer code from natural language descriptions |
| 8 | Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study | Investigates the security weaknesses in code generated by AI tools like GitHub Copilot, CodeWhisperer, and Codeium |
| 9 | Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation | Investigates how AI code generation tools, specifically GitHub Copilot, impact novice programmers' performance, workload, emotions, and self-efficacy |
| 10 | Unveiling Memorization in Code Models | Investigates the extent to which large code models memorize their training data |
| 11 | My Code Is Less Secure with Gen AI: Surveying Developers | Investigates developers' perceptions of code security when utilizing Generative AI (GAI) tools for software development |
| 12 | "From “Ban It Till We Understand It” to “Resistance is Futile”: How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot | Explores how university programming instructors plan to adapt their teaching methods in response to the increasing use of AI code generation and explanation tools |
| 13 | Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education | Evaluates the accuracy and limitations of existing AI-Generated Content (AIGC) detectors when applied to AI-generated code |
| 14 | Evaluation of Generative AI Models in Python Code Generation: A Comparative Study | Compares various generative AI models on their ability to generate Python code |
| 15 | From Today’s Code to Tomorrow’s Symphony: The AI Transformation of Developer’s Routine by 2030 | Explores the transformative role of Artificial Intelligence (AI) in software development, projecting significant changes by 2030 |
| 16 | Assessing GitHub Copilot in Solidity Development: Capabilities, Testing, and Bug Fixing | Investigates the effectiveness of GitHub Copilot, an AI-powered coding assistant, in developing Solidity smart contracts for blockchain applications |
| 17 | The R5E pattern: can artificial intelligence enhance programming skills development | Introduces and evaluates the R5E pattern, a novel pedagogical framework designed to integrate ChatGPT into programming education effectively |
| 18 | Generation Probabilities Are Not Enough: Uncertainty Highlighting in AI Code Completions | Investigates the effectiveness of highlighting uncertain sections in AI-generated code completions to help programmers |
| 19 | Significant Productivity Gains through Programming with Large Language Models | Investigates the impact of large language models (LLMs) on programmer productivity |
| 20 | Code Ownership in Open-Source AI Software Security | Explores the crucial role of code ownership in enhancing the security of open-source AI software projects |

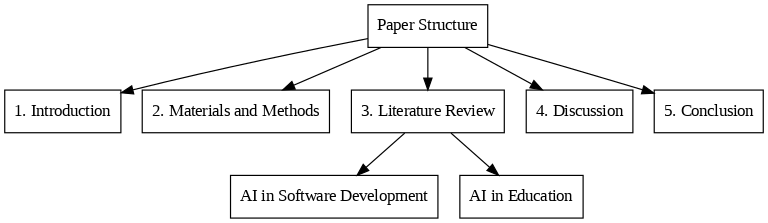
After looking at the studies above, it's clear there's a common idea about how AI is used in both making software and in teaching/learning activities, particularly in areas like computer science.

Figure. 1 Paper structure.

# Literature Review

## AI in Software Development

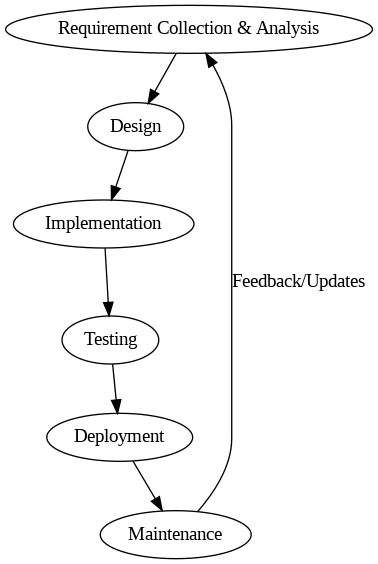
The integration of AI into the Software Development Lifecycle (SDLC) heralds a transformative era for developers, fundamentally altering their roles from manual coders to orchestrators of AI-driven development ecosystems [15]. AI advancements are set to enhance various phases, from planning to maintenance, significantly improving overall software quality and efficiency [15].

Figure. 2 Software Development Lifecycle.

## A.1. AI-Assisted Programming (AIAP)

AI-assisted programming (AIAP) refers to the use of AI tools to aid developers in various coding tasks. Current AIAP tools are diverse and include well-known names such as GitHub Copilot, OpenAI’s ChatGPT, Google Bard, Claude AI, Gemini AI, Amazon CodeWhisperer, and others like Tabnine, Kite, AlphaCode, Code4Me, and Cursor [16], [12], [3], [2], [1]. These tools leverage deep learning algorithms and are trained on vast datasets of code repositories and natural language data [12].

Their capabilities span a wide range of programming activities:

* Code Generation: AI tools can generate code from natural language descriptions (specification-to-code) or through conversational prompts, providing solutions for functions, classes, or entire modules [16]. They can also offer real-time code completions within Integrated Development Environments (IDEs).
* Code Refactoring and Simplification: AI can rewrite existing code to improve readability, style, or maintainability, or simplify code to use more basic features [12].
* Bug Fixing and Vulnerability Detection: AI tools can assist in identifying and fixing bugs and detecting security vulnerabilities. For instance, Copilot Chat can fix a significant percentage of security issues when provided with warning messages from static analysis tools [8].
* Test Generation: AI tools can generate various types of test cases [9], including unit and regression tests, and even for unusual edge cases that human novices might overlook.
* Documentation and Explanation: They can also create documentation and explain the functionality of code in natural language [9], [12].

## A.2. Developer Impact

The integration of AIAP tools has had a notable impact on developers' work:

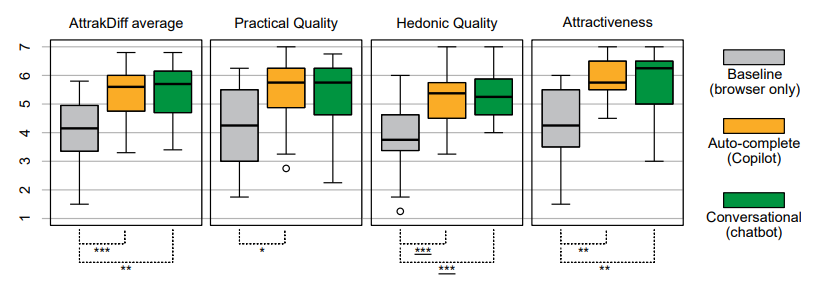
* Increased Productivity and Efficiency: Studies consistently show that AI tools like GitHub Copilot and ChatGPT significantly enhance developer productivity and efficiency [15]. They streamline routine coding tasks, accelerate the software lifecycle, and reduce the time spent on coding [15]. Developers report faster task completion and reduced mental workload and effort when using AI assistance [9], [18].

Figure. 3 Illustrates how AI assistants (Copilot and chatbot) significantly enhance user experience, with AttrakDiff questionnaire results consistently showing them outperforming the browser-only baseline.

* Code Quality: While AI-generated code can streamline development, studies highlight the need for thorough quality checks before integration [14]. However, in general, AI assistants produce code that is about equally good as what a human can produce in terms of objective code quality metrics, and can help reduce code duplication and provide suggestions in a consistent style in larger projects [19].
* Security Implications: A critical concern is the security of AI-generated code. Research indicates that AI tools, often trained on open-source repositories, may inadvertently reproduce vulnerable code [11], [14], [4]. Studies show that Copilot can generate insecure code in a significant percentage of cases (e.g., around 40%) [15]. Experienced developers, in particular, perceive that their proficiency in secure coding might decrease when using GAI tools [11]. This necessitates rigorous review and enhancement of AI-generated code by human experts, particularly in complex or security-critical contexts. Developers acknowledge the need for extensive security reviews and the use of code scanning tools to identify vulnerabilities.

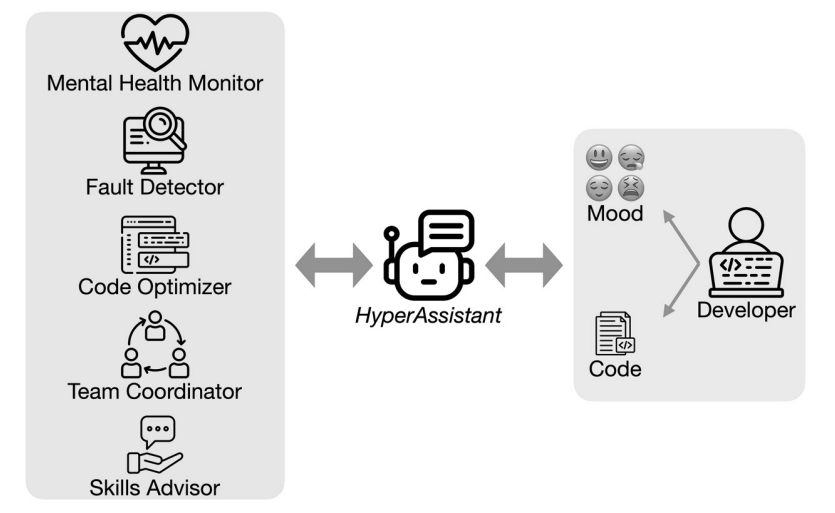
## A.3. Challenges

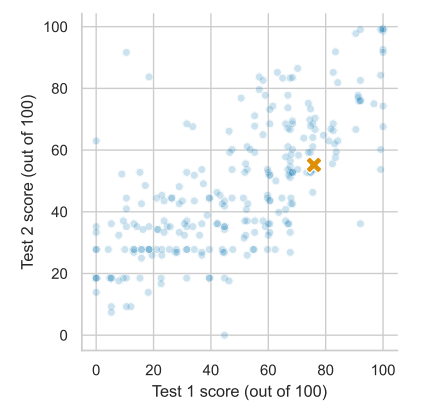
Despite the numerous benefits, AIAP tools introduce several challenges:

* Reliability and Accuracy: A notable limitation is that AI tools can generate inaccurate outputs with no quality guarantees. They may produce suboptimal code, struggle with intricate logical constructs, unique edge cases, or esoteric requirements. The non-deterministic nature of AI outputs and inconsistent results across model updates also pose challenges for reproducibility [12].
* Intellectual Property (IP) and Licensing: Concerns arise regarding the ethical implications of AI models being trained on public code repositories without explicit consent or clear licensing agreements. This raises questions about copyright and attribution for code generated by AI, which may inadvertently include parts of copyrighted or licensed material [1].
* Deskilling and Over-reliance: There is a significant concern that developers, especially novices, may become overly reliant on AI tools, potentially hindering their understanding of programming fundamentals and core problem-solving skills [1]. This "false sense of understanding and proficiency" can leave them under-prepared for careers where deep conceptual knowledge is crucial [1], [9].
* New Paradigms and Learning Curve: The shift from manual coding to "orchestrating AI-driven development ecosystems" requires developers to adapt to new workflows and acquire new skills, such as prompt engineering [1]. Learning to craft effective and reliable prompts to guide AI tools has become an essential skill, as initial interactions with simple prompts may not yield high-quality results.
* Human Factors and Social Aspects: Current AI assistants primarily focus on technical challenges, often overlooking human-factor problems such as developer mental health [15]. They also have limited capabilities in enhancing team interactions or providing personalized learning paths for individual developers, failing to retain significant information about each developer's unique skills and needs [15].

## A.4. Future Trends

The future of AI in software development envisions a more deeply integrated and supportive role for AI:

* HyperAssistant Vision: A hypothetical "HyperAssistant" is envisioned for 2030 [15], offering comprehensive support beyond just coding assistance. This includes a mental health monitor to detect fatigue, a fault detector for bugs and vulnerabilities, a code optimizer for efficiency, a team coordinator to improve collaboration and task distribution, and a skills advisor for personalized learning resources based on a developer's unique needs and evolving language features.

Figure. 4 Overview of HyperAssistant workflow and its components.

* Enhanced Support Across SDLC: AI is expected to extend its influence across all stages of the SDLC, from automatically generating test cases based on requirements and code, ensuring comprehensive coverage, to continuous learning mechanisms that allow AI tools to improve their suggestions and assistance over time [15].
* Complementary Force: AI is seen as an indispensable assistant rather than a replacement for human developers. The synergy between human and AI is anticipated to lead to the creation of more sophisticated, reliable, and secure software solutions [15].

## AI in Education

The emergence of generative AI and its proliferation have brought about a transformative era in education, introducing a complex interplay of challenges and opportunities for both educators and learners [13], [15].

## B.1. Personalized Learning & Assessment

AI offers significant potential for revolutionizing learning experiences:

* Adaptive Pathways and Tailored Experiences: AI can provide personalized and enhanced learning experiences by adapting to individual student needs and learning styles. This includes generating additional examples, explanations, and exercises in response to student questions on-demand. AI-driven tutoring systems can offer tailored feedback and assistance, potentially addressing teacher shortages [17].
* Automated Assessment and Feedback: AI can automate time-consuming tasks for instructors, such as grading and lesson planning [17]. It can provide step-by-step explanations of how code works and check for code style best-practices, akin to a code review. Some studies even propose AI for automating the assessment and feedback process for programming assignments [17].

## B.2. Curriculum Adaptation

The capabilities of AI tools necessitate significant adaptations in computing education curricula:

* Integration into CS/Related Fields: There is a growing consensus on the imperative to integrate AI tools into Computer Science (CS) and Software Engineering (SE) education. This ensures students are prepared for a future workforce where AI tools are commonplace [12].
* Rethinking CS1/CS2: Traditional introductory programming courses (CS1/CS2) that focus heavily on coding mechanics are being re-evaluated, as AI can now solve many typical assignments and exam problems [1]. The shift in emphasis is moving towards software design, code reading, critical analysis, and effective communication with AI via natural language prompts [12].

Figure. 5 OpenAI Codex's strong performance on two CS2 exams (orange 'x') against student scores.

* Designing AI-Proof Assignments: To mitigate academic misconduct, instructors are exploring ways to design "AI-proof" assignments by adding more local or cultural context, or using bespoke starter code that AI tools lack access to [12]. Some educators also propose reverting to paper-based exams or using oral, video, and image-based assessments to prevent AI use [12].

## B.3. Pedagogical Changes

AI's influence extends to the very methods of teaching and learning:

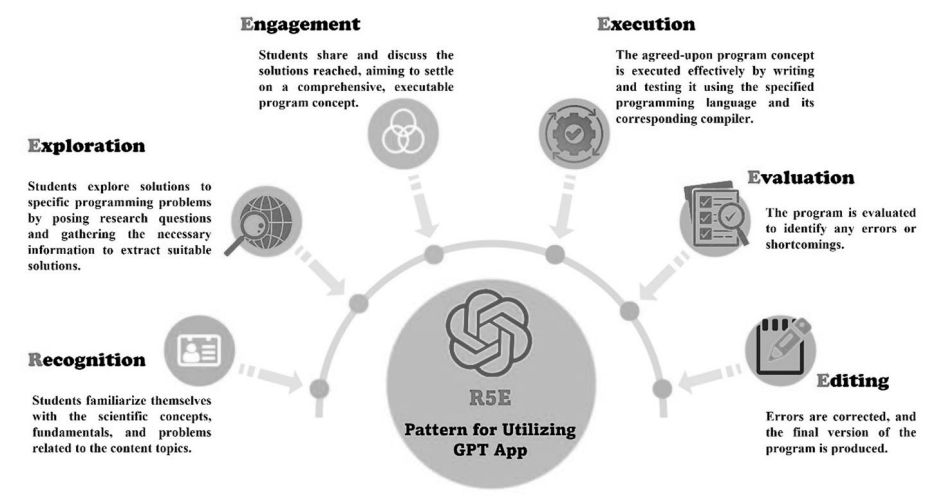
* Focus on Higher-Order Skills: If AI can automate rote coding mechanics, CS1/CS2 courses can focus more on software design, problem-solving, and algorithmic creativity [12].
* Collaborative Learning with AI: New pedagogical models are emerging where students are encouraged to work collaboratively with AI on assignments, viewing AI as a partner in the problem-solving process. This can involve students assuming the role of a "client" specifying requirements to the AI, then testing and critiquing its output [12].
* "Prompt Problems": A new programming exercise type, "Prompt Problems," is proposed where students focus on crafting effective prompts for AI tools to generate correct code, thereby developing prompt engineering skills and understanding AI capabilities and limitations [5].
* Structured AI Utilization (R5E Pattern): Studies emphasize the critical need for structured methodologies to regulate the use of AI tools in education. The R5E pattern (Recognition, Exploration, Engagement, Execution, Evaluation, and Editing) is one such novel pedagogical model designed to promote active learning and critical thinking while integrating ChatGPT into programming education [17]. This model has shown superior efficacy in fostering programming skill development compared to unstructured ChatGPT use.

Figure. 6 The elements of the R5E Pattern for employing ChatGPT in programming education.

* Instructor's Evolving Role: Instructors are called to guide students through ethical reflection on AI use, fostering a culture that promotes responsible and ethical AI usage [6].

## B.4. Ethical & Societal Implications

The widespread adoption of AI in education introduces significant ethical and societal considerations:

* Academic Integrity and Plagiarism: This is a paramount concern for educators. AI tools can generate variations of code that are not exact copies, making them less detectable by traditional plagiarism detectors [12].
* Privacy and Data Concerns: AI models are often trained on vast amounts of public data, including open-source code, which raises concerns about data licensing, privacy, and the use of sensitive information if inadvertently processed by AI tools. Students or developers are recommended to avoid submitting sensitive data or personal information to GAI tools [11].
* Job Displacement and Future Prospects: Students express anxiety about the long-term impact of AI on programming jobs, with some fearing that expertise in coding may become obsolete [5].

## B.5. Future Preparedness

Preparing students for an AI-transformed workforce and society is a crucial responsibility:

* Focus on Evaluation and Critical Analysis: The future curriculum will emphasize skills in evaluating AI-generated output, critically analyzing code, and communicating effectively with AI via natural language prompts [12].
* Developing Policies and Social Norms: The computing education community has a "unique and timely opportunity" to develop policies and social norms that influence how AI tools impact future generations of students, ensuring ethical and equitable use [12].
* Ongoing Research: Continued research is essential to understand the long-term impact of AI tools on student skill development, creativity, and learning engagement [16]. This includes investigating the effectiveness of new pedagogical approaches, developing AI-driven content detection models, and exploring AI's applicability across diverse domains and programming languages.

# Discussion

The literature clearly illustrates that AI is not merely an incremental technological advancement but a fundamental force reshaping both software development and education. A recurring theme is the tension between the immediate benefits of AI in terms of productivity and efficiency, and the long-term concerns regarding reliability, security, and human skill development.

In software development, AIAP tools are increasingly integrated into daily workflows, proving highly effective for code generation, completion, and rudimentary bug fixing. This augmentation allows developers to accelerate tasks and focus on higher-level problem-solving. However, the reliability of AI-generated code, particularly concerning security vulnerabilities, remains a significant challenge. The consensus is that human oversight and rigorous review are indispensable. The vision of a "HyperAssistant" in 2030 underscores a future where AI's role expands beyond technical assistance to encompass aspects like developer well-being and team coordination, signifying a holistic integration into the human-centric aspects of software engineering. This suggests a shift in the developer's role from a manual coder to an orchestrator who leverages AI while maintaining critical judgment.

In education, AI's potential for personalized learning and automated assessment is transformative. It can alleviate instructor workload and provide tailored support for diverse learners. However, the ethical implications, particularly academic integrity, are pressing. The ability of AI to generate varied, seemingly human-written solutions poses a considerable challenge to traditional assessment methods. The debate between "resisting" and "embracing" AI tools in the classroom highlights divergent philosophies: some prioritize preserving fundamental programming skills through AI-proof assignments and traditional exams, while others advocate for integrating AI to prepare students for future careers, emphasizing skills like prompt engineering and critical evaluation of AI outputs. The R5E pattern stands out as a validated pedagogical model demonstrating that structured integration can lead to improved learning outcomes and foster critical thinking, contrasting with the passive learning that can result from unstructured AI use.

A common thread across both domains is the importance of human adaptation and continuous learning. Developers need to learn how to effectively prompt and validate AI output. Educators must redefine learning objectives and assessment strategies to account for AI's capabilities. The ethical concerns surrounding data privacy, bias, and licensing are pervasive, demanding robust policies and responsible AI development practices. The digital divide, while often seen as an equity challenge, is also presented as an opportunity for AI to democratize access to programming skills, provided equitable access to the tools is ensured.

Existing research provides a snapshot of an rapidly evolving field, primarily focusing on introductory programming and widely used AI tools. There are still gaps in understanding the long-term impacts, the effectiveness of AI in more complex programming tasks (e.g., Solidity development), the specific needs of expert vs. novice users, and the generalizability of findings across different programming languages and educational contexts. Future research must prioritize longitudinal studies, explore diverse programming domains, and delve deeper into the socio-technical dynamics of human-AI collaboration.

# Conclusion

The influence of AI on software development and education is undeniable and profoundly transformative. In software development, AI-assisted programming tools have emerged as powerful allies, significantly boosting productivity, streamlining workflows, and offering intelligent assistance across various tasks. While they accelerate development and maintain code quality, the inherent challenges related to security vulnerabilities, intellectual property, and the potential for over-reliance necessitate diligent human oversight and continuous skill adaptation. The future envisions AI as an indispensable, holistic assistant, deeply integrated into the SDLC, capable of addressing not only technical but also human-centric aspects of development.

In education, AI presents unparalleled opportunities for personalized learning, adaptive assessment, and a re-imagining of curricula, shifting the focus from rote coding mechanics to higher-order computational thinking and design. However, this revolution also demands a critical re-evaluation of academic integrity, an active engagement with ethical considerations such as bias and privacy, and a concerted effort to mitigate the digital divide. The imperative for educators is to proactively adapt, preparing students not just to code, but to critically interact with, evaluate, and ethically leverage AI tools in a rapidly evolving technological landscape.

Ultimately, the integration of AI in both these domains is a journey of continuous learning and adaptation. To fully realize AI's transformative potential while safeguarding academic integrity and fostering responsible technological progress, ongoing research, thoughtful policy development, and a strong emphasis on the synergistic collaboration between humans and AI will be paramount. The "AI revolution" has arrived at our classrooms and workplaces, and the response must be one of informed engagement and strategic foresight.

References

[1] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming Is Hard — Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation" in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*, pp. 500–506, 2023.

[2] T. Chen, Y. Zheng, Y. Li, B. Wu, X. Li, Y. Wang, X. Zhang, X. Li, X. Wu, and Y. Liu, "Need Help? Designing Proactive AI Assistants for Programming" in *CHI ’25*, Yokohama, Japan, 2025.

[3] R. Cheng, A. Smith-Renner, K. Zhang, J. Tetreault, and A. Jaimes-Larrarte, "'It would work for me too': How Online Communities Shape Software Developers’ Trust in AI-Powered Code Generation Tools" *ACM Transactions on Interactive Intelligent Systems*, vol. 14, no. 2, pp. 11:1–11:39, 2024.

[4] D. Cotroneo, C. Improta, P. Liguori, and R. Natella, "Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks" in *32nd IEEE/ACM International Conference on Program Comprehension (ICPC ’24)*, Lisbon, Portugal, 2024.

[5] P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, and B. N. Reeves, "Prompt Problems: A New Programming Exercise for the Generative AI Era" in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*, 2024.

[6] J. Finnie-Ansley, P. Denny, A. Luxton-Reilly, E. A. Santos, J. Prather, and B. A. Becker, "My AI Wants to Know if This Will Be on the Exam: Testing OpenAI’s Codex on CS2 Programming Exercises" in Australasian Computing Education Conference (ACE ’23), Melbourne, VIC, Australia, 2023.

[7] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming" in Australasian Computing Education Conference (ACE ’22), Virtual Event, Australia, 2022.

[8] 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" *ACM Transactions on Software Engineering and Methodology*, 2024.

[9] N. Gardella, R. Pettit, and S. L. Riggs, "Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation" in *Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*, Milan, Italy, 2024.

[10] D. Han and Z. Yang, "Unveiling Memorization in Code Models" in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE ’24)*, Lisbon, Portugal, 2024.

[11] A. Kudriavtseva, N. A. Hotak, and O. Gadyatskaya, "My Code Is Less Secure with Gen AI: Surveying Developers’ Perceptions of the Impact of Code Generation Tools on Security" in *SAC ’25*, Catania, Italy, 2025.

[12] S. Lau and P. J. Guo, "From “Ban It Till We Understand It” to “Resistance is Futile”: How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot" in *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1 (ICER ’23 V1)*, Chicago, IL, USA, 2023.

[13] M. K. Lim, "Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education" in *46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET ’24)*, Lisbon, Portugal, 2024.

[14] D. Palla and A. Slaby, "Evaluation of Generative AI Models in Python Code Generation: A Comparative Study" *IEEE Access*, 2025.

[15] K. Qiu, N. Puccinelli, M. Ciniselli, and L. Di Grazia, "From Today’s Code to Tomorrow’s Symphony: The AI Transformation of Developer’s Routine by 2030" *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, Article 121, 2025.

[16] G. Baralla, G. Ibba, and R. Tonelli, "Assessing GitHub Copilot in Solidity Development: Capabilities, Testing, and Bug Fixing" *IEEE Access*, 2024.

[17] Y. A. M. Abouelenein, A. F. A. Ghazala, E. M. M. Mahdy, and M. H. R. Khalaf, "The R5E pattern: can artificial intelligence enhance programming skills development" *Education and Information Technologies*, 2025.

[18] H. Vasconcelos, G. Bansal, A. Fourney, Q. V. Liao, and J. W. Vaughan, "Generation Probabilities Are Not Enough: Uncertainty Highlighting in AI Code Completions" *ACM Transactions on Computer-Human Interaction*, vol. 32, no. 1, Article 4, 2025.

[19] T. Weber, M. Brandmaier, A. Schmidt, and S. Mayer, "Significant Productivity Gains through Programming with Large Language Models" *Proceedings of the ACM on Human-Computer Interaction*, vol. 8, EICS, Article 256, 2024.

[20] J. Wen, D. Yuan, L. Ma, and H. Chen, "Code Ownership in Open-Source AI Software Security" in *2024 International Workshop on Responsible AI Engineering (RAIE ’24)*, Lisbon, Portugal, 2024.

Harvest Ecclesiano Christ Walukow (164231104) is an undergraduate student majoring in Data Science Technology at Universitas Airlangga, class of 2023. His academic interests include machine learning and artificial intelligence. He can be contacted at email: harvest.ecclesiano.christ-2023@ftmm.unair.ac.id.