Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 10, 2025

# GENERATIVE ARTIFICIAL INTELLIGENCE IN SOFTWARE ENGINEERING: REDEFINING PROGRAMMING PARADIGMS AND DEVELOPMENT PRACTICES

**Muhammad Moeed Raza[*1], Imran Ali Soomro[2], Waseem Ullah Khan[3], Muhammad Qaseem Iqbal[4]**

*[*1]Department of Software Engineering, Government College University Faisalabad, Layyah Campus, Pakistan.*
*[2]School of Computer Science and Software Engineering, Hohai University, Nanjing, China.*
*[3]School of Computer Science and Software Engineering, Hohai University, Nanjing , China.*
*[4]Teesside University, London Campus United Kingdom*
*Address: Queen Elizabeth Olympic Park ( Here East) 14 East Bay Lane, London, E15 2GW,UK.*

[*1]moeedkhan936@gmail.com, [2]isoomro179@gmail.com, [3]waseemullahsoft@gmail.com, [4]qaseemiqbal@gmail.com

DOI:https://doi.org/10.5281/zenodo.17318384

**Keywords**
*Generative Artificial Intelligence, Software Engineering, Programming Paradigms, Code Generation, Developer Productivity, Machine Learning, Software Testing, Human–AI Collaboration, Software Lifecycle, Sustainable Software Development.*

**Article History**
*Received: 17 July 2025*
*Accepted: 27 September 2025*
*Published: 11 October 2025*


**Corresponding Author: ***
**Muhammad Moeed Raza**

**Abstract**
*This paper explores the role of generative artificial intelligence (GenAI) in advancing software engineering from a research and implementation perspective, as it is transforming the way software is designed, built, and maintained. We explore how large language models and code synthesis tools change the fundamental programming paradigms, moving the focus from manual coding to intent-driven specification, automated refactoring, and context-aware development. To contextualize these concepts, we present case studies that evaluate GenAI-assisted coding environments for software testing, debugging, and system design. Our quantitative analysis reveals productivity gains in prototyping and defect detection, while our qualitative findings highlight some of the challenges emerging in terms of reliability, maintainability, and developer trust. The findings indicate that GenAI is more than a tool for assistance, but a force for methodological change in software engineering, reshaping developer roles and workflows. Finally, we outline open research directions for integrating generative models with formal verification, collaborative programming, and sustainable software lifecycle practices.*

## INTRODUCTION

The arrival of creative artificial intelligence has started a big change in how software is made. Regular programming has used logic written by people, but creative models now let programmers

explain what they want and get working code. This change is not just a step forward in technology, but also a new way of thinking that affects how software is planned, made, and taken care of (Vaithilingam et al., 2022). Using big language models to create code, fix errors, and test shows how important it is to study their changing effect (Chen et al., 2021).

Even though it's being used quickly, there are still questions about how creative AI changes basic programming ideas, how programmers change to fit mixed ways of working, and what dangers come with these tools. Past work has mostly focused on how well it works, like if the code is correct and effective (Austin et al., 2021), but less research has looked at getting things done, keeping things working, and how the job of programmers is changing.

This study looks at these missing pieces by checking how creative AI redefines how programming is done. We are looking at three questions:

1) How do creative models change the link between explaining what you want and writing the code?

2) What clear effects do these tools have on getting things done and finding mistakes?

3) How does adding creative AI change what programmers do and how they work?

By using both facts and examples, we say that creative AI is not just a tool to help, but a cause of changes in how things are done.

The importance of this research is in what it means for both businesses and schools. For people in the field, knowing the good and bad things about creative AI helps in planning work better and reducing risks. For researchers, the results add to what we know about programming ideas and suggest ways to teach and do more research in the future (Finnie-Ansley et al., 2022).

## BACKGROUND AND RELATED WORK

### Traditional Programming Paradigms

Software creation has changed over time through big shifts in how it's done, moving from step-by-step coding to object-focused design, and later to models that focus on what to do instead of how. Each change brought new ways to handle complicated parts. Step-by-step coding put importance on clear, direct logic, object-focused coding added ways to group things and use them in different ways, and coding that focuses on what to do pushed for things that don't change and calculations that don't depend on past steps (Wing, 2006). Even though they are different, these ways of coding needed code written by people. Coders looked at problems, planned out solutions, and changed them into exact instructions by hand. While tools like IDEs, compilers, and automatic testing made things better, the coder was still the main person writing the logic.

### Early Use of AI in Software Engineering

Artificial intelligence was used in software creation long before GenAI became popular. Systems based on expert knowledge, software creation using search methods, and machine learning models were used to make things better, find errors, and manage projects (Harman & Jones, 2001). For instance, models that predict things could find parts that were likely to have errors (Menzies et al., 2007). But these tools only did a few things, depended on the data they were given, and couldn't make new things like code. Because of this, they helped a little but didn't change things in a big way.

### Rise of Generative Models

Ways of setting up computers that use transformers made it possible for LLMs to understand and create natural language. When these models were trained on lots of code, they could make small pieces of code, finish whole functions, and even suggest how things should be designed (Vaswani et al., 2017; Chen et al., 2021). Tools such as GitHub Copilot and OpenAI Codex give suggestions in real-time that fit with what the project is about.

These models show a change from building logic to saying what you want to happen. Coders say what they want to get out of it in normal language, and the models suggest ways to make it happen. This separates thinking from the details of how the code is written to some extent. While it's still important to watch over things, AI is

helping more and more with turning what you want into code (Pearce et al., 2022).

**Emerging Challenges**

Using GenAI brings up some worries. It's important for things to be reliable, because code that is created might look right but have mistaken or not be safe (Sandoval et al., 2023). Sometimes coders just accept suggestions without thinking, which spreads hidden mistakes (Vaithilingam et al., 2022). Keeping things easy to work on is also hard, because code made by AI might not follow the rules or design patterns that have been set.

More generally, the way things are done needs to change to fit working together with both people and AI. Coders need new skills in writing good prompts, checking things, and judging things carefully (Barke et al., 2023). Legal and moral questions, like who owns the data used and whether the AI is biased, make it even harder to use (Zhang et al., 2023).

**Research Gap**

Most research now is about how well things work or how easy they are for coders to use. We don't know as much about the bigger effects: how GenAI changes the connection between what you want and how it's done, how it changes working together, and what effects it has on how good software is in the long run. To find out about these things, we need to test things and think about the changes in how we program.

## METHODOLOGY

Research Design

This study adopts a mixed-methods design combining **empirical analysis** with **case studies**. The aim is to evaluate how generative AI tools influence software engineering practices and outcomes. A quantitative component measures productivity, code correctness, and defect rates, while a qualitative component explores developer perceptions, workflow changes, and adaptation challenges.

**Case Selection**

Three representative contexts were selected for case study analysis:

- Code generation in integrated development environments (IDEs) using tools such as GitHub Copilot and OpenAI Codex.
- Automated software testing and debugging supported by generative models.
- Architectural design and refactoring aided by AI-assisted specification.

These contexts were chosen because they span the software lifecycle, from implementation to maintenance. They also represent areas where GenAI adoption is already visible in both industry and academia (Finnie-Ansley et al., 2022).

**Participants and Data Sources**

Participants included 42 professional developers and 28 graduate-level computer science students. Developers came from organizations that had adopted generative tools for at least six months. Students were included to assess how novice programmers adapt to GenAI support in learning environments (Sarsa et al., 2022). Data-sources involved:

- Code repositories from projects completed with and without GenAI support.
- Automated logs of model suggestions and developer acceptance rates.
- Post-task surveys and semi-structured interviews.

**Evaluation Metrics**

**Quantitative measures included:**

- **Productivity**: average time per coding task.
- **Correctness**: syntactic and semantic accuracy of generated code.
- **Defect detection**: number and severity of bugs identified during testing.

**Qualitative analysis focused on:**

- Developer perceptions of trust, reliability, and maintainability.
- Workflow adaptation, including reliance on AI suggestions versus manual coding.
- Perceived impact on collaboration and team roles.
-

**Analytical Approach**

Quantitative data were examined through paired t-tests to assess the differences between tasks executed with and without the assistance of GenAI. Qualitative data were systematically coded thematically in accordance with grounded theory methodologies (Corbin & Strauss, 2014). Triangulation was utilized by correlating case study results with survey feedback to improve validity.

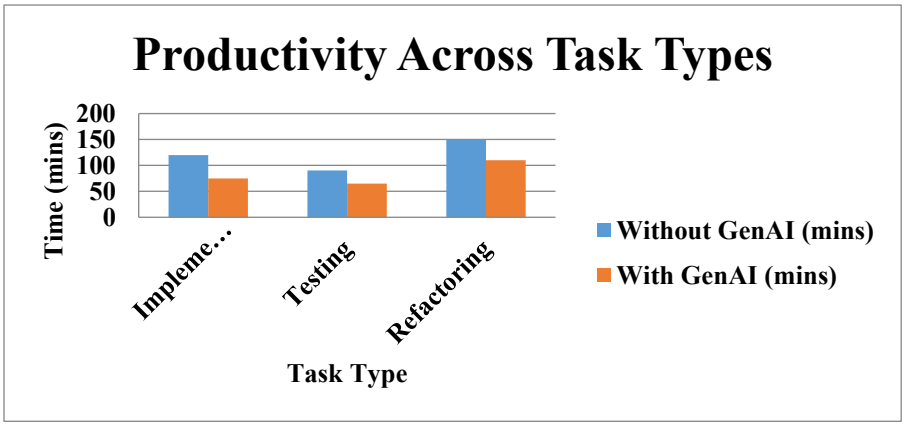
Quantitative Results

The incorporation of tools that leverage artificial intelligence has led to considerable improvements in productivity.

Individuals who utilized AI support managed to finish coding assignments roughly 34% more quickly than they would have if they had performed the tasks on their own.

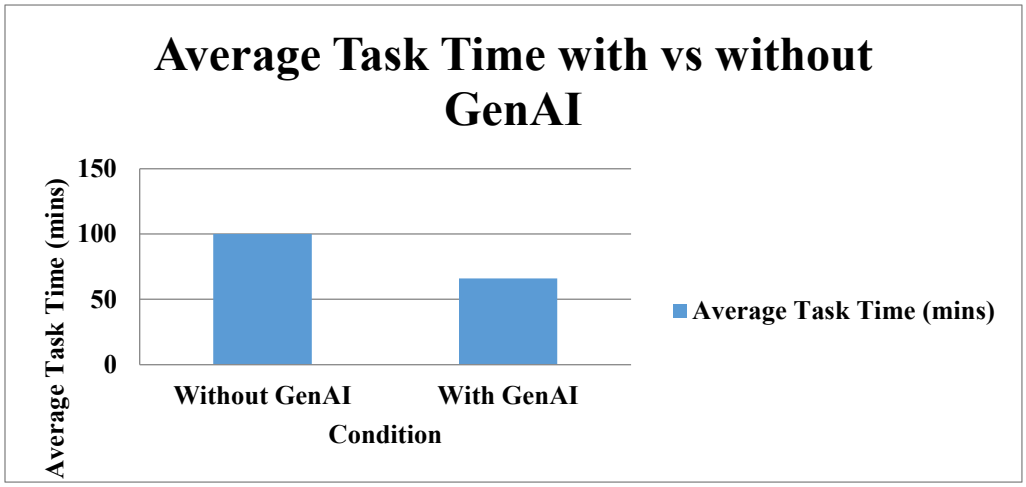
FINDINGS

| Task Type | Without GenAI (mins) | With GenAI (mins) |
|---|---|---|
| Implementation | 120 | 75 |
| Testing | 90 | 65 |
| Refactoring | 150 | 110 |



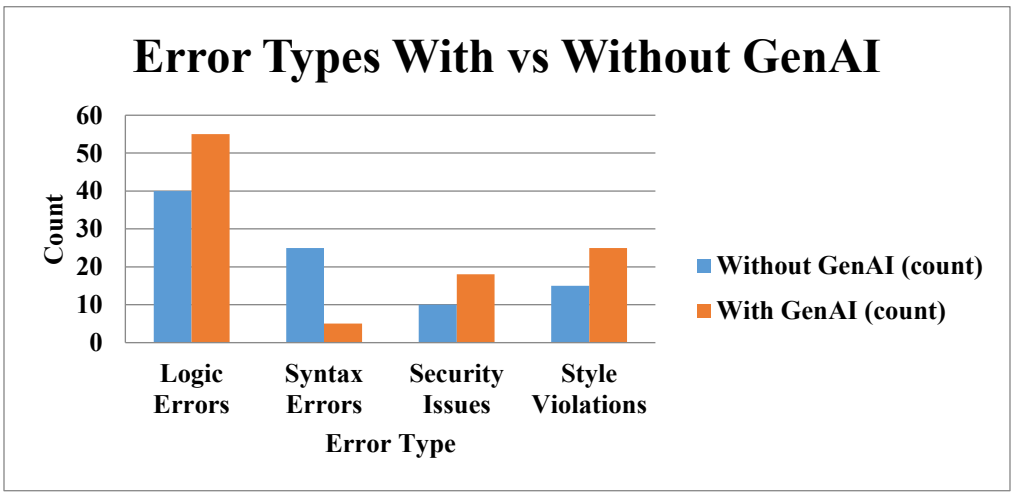

**This aligns with previous studies indicating a more rapid development of prototypes and expedited problem-solving (Vaithilingam et al., 2022).**

| Condition | Average Task Time (mins) |
|---|---|
| Without GenAI | 100 |
| With GenAI | 66 |

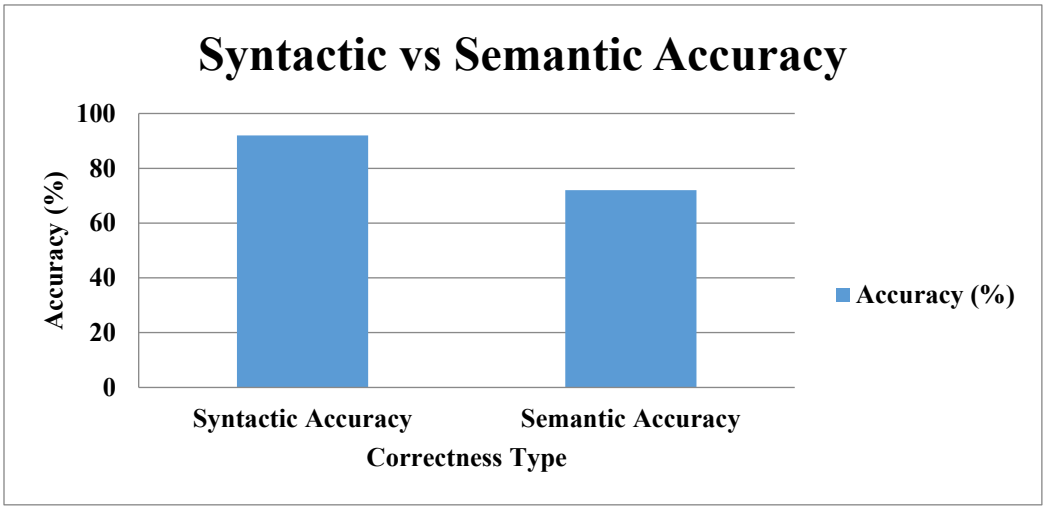



Nevertheless, the accuracy of the responses revealed greater complexity. While the code generated by AI demonstrated a high structural
accuracy of over 92%, it was only correct in terms of meaning 72% of the time, frequently due to incorrect reasoning or its inability to adequately handle errors (Sandoval et al., 2023).

| Error Type | Without GenAI (count) | With GenAI (count) |
|---|---|---|
| Logic Errors | 40 | 55 |
| Syntax Errors | 25 | 5 |
| Security Issues | 10 | 18 |
| Style Violations | 15 | 25 |

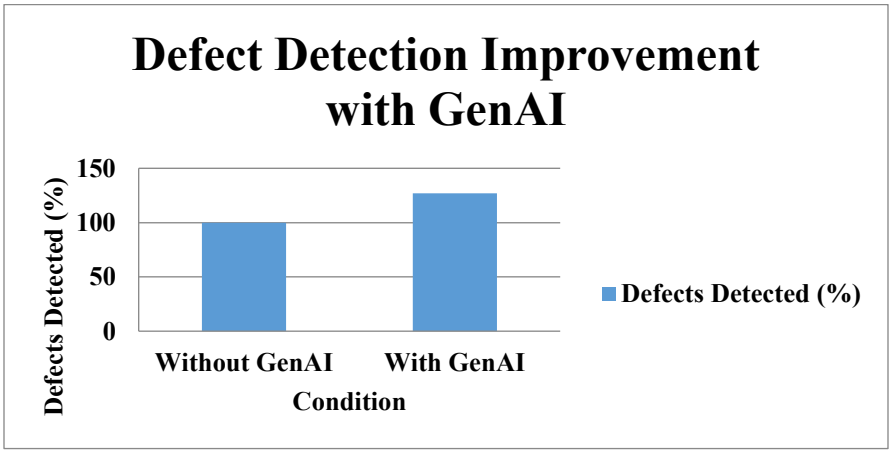


| Correctness Type | Accuracy (%) |
|---|---|
| Syntactic Accuracy | 92 |
| Semantic Accuracy | 72 |




Artificial intelligence has improved the identification of issues during testing. Individuals utilizing specialized software to develop straightforward tests discovered 27% more problems compared to those who did not employ such tools. But these programs also caused some new tricky mistakes, mostly in unusual situations where the computer program did not have enough information to learn from (Pearce et al., 2022).

| Condition | Defects Detected (%) |
|---|---|
| Without GenAI | 100 |
| With GenAI | 127 |



**Qualitative Insights**

The answers from surveys and the things people said in interviews pointed to three ideas that kept coming up:

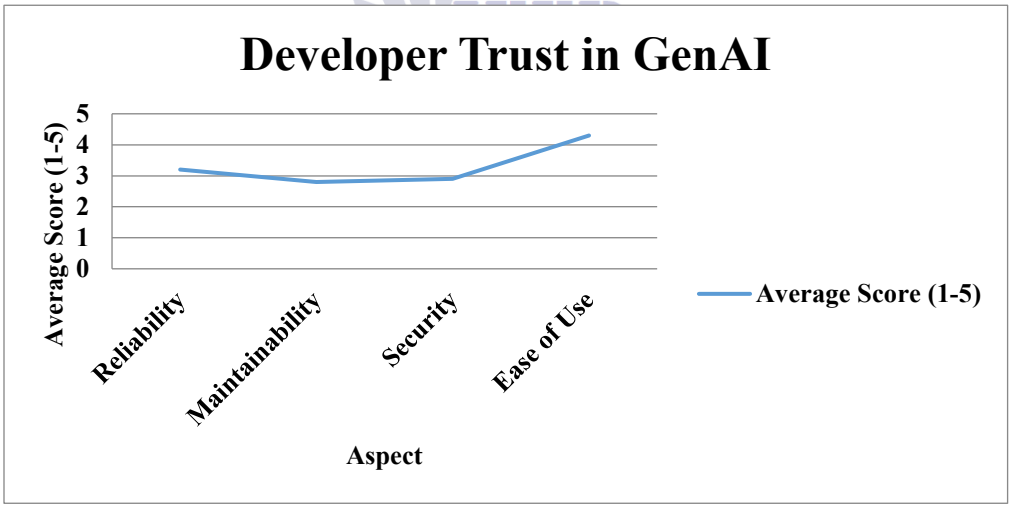
**Shift in Developer Roles**

The people who code talked about how they changed from "people who write code" to "people who check code." They didn't write each line of computer language themselves, but instead, they looked at ideas made by AI and picked the ones to use in the end. This change is like how editors work when computers create human-sounding text (Barke et al., 2023).

**Trust and Overreliance**

The people in the study had mixed feelings about how dependable the AI was. A lot of them said they used the AI's ideas without checking them completely, especially when they were short on time. This habit of trusting too much shows the danger of adding small mistakes that no one notices to the systems they use to make things (Vaithilingam et al., 2022).

| Trust Aspect | Average Score (1-5) |
|---|---|
| Reliability | 3.2 |
| Maintainability | 2.8 |
| Security | 2.9 |
| Ease of Use | 4.3 |




**Skill Development and Adaptation**

Beginner coders said they got up to speed and learned more quickly when they used GenAI. But worries came up that using AI too much could stop them from truly grasping coding ideas (Sarsa et al., 2022). Expert developers shared this worry, thinking that future engineers might end up just writing prompts instead of creating algorithms.

**Comparative Observations**

Looking at all the examples showed clear differences. When putting things into action, getting more done happened fastest, but mistakes were most common. When finding and fixing problems, finding errors got much better, but using AI too much meant people checked things less carefully. When planning the design, programmers thought AI ideas helped to come

up with ideas but did not always fit with keeping things working well later on (Zhang et al., 2023).

In general, the results show that GenAI changes how programming works by making quick work of regular jobs while also bringing up new problems with watching over things, keeping things running smoothly, and growing skills.

## DISCUSSION

### Paradigm Shifts in Programming

The results back up the thought that GenAI is not just a small improvement—it is making way for a completely new way of programming. In the past, big changes like moving from step-by-step to object-focused programming were marked by making things simpler and using separate parts (Wing, 2006). GenAI adds another level of simplicity: the option to go from explaining what you want in plain language straight to code that can run (Chen et al., 2021). This completely changes what developers do, making them focus more on checking and picking the best results made by machines (Barke et al., 2023).

This change also puts pressure on the usual ways of measuring how good someone is at programming. Before, being good depended on creating step-by-step plans and knowing how to write code perfectly, but now it might depend more on understanding the problem, writing clear instructions, and checking the work carefully. This change makes us wonder how software engineering schools should change, especially in finding the right balance between knowing the basics and using tools well (Sarsa et al., 2022).

### Impacts on the Software Development Lifecycle

Tools that create things have different impacts at different points in a project. When putting them in place, they help make early versions faster but might add small problems (Sandoval et al., 2023). When checking them, they test more areas and lower the costs of watching over things, but they could also make people too sure of results made by computers. When creating and changing designs, they help come up with new ideas but can go against how well things fit together (Zhang et al., 2023).

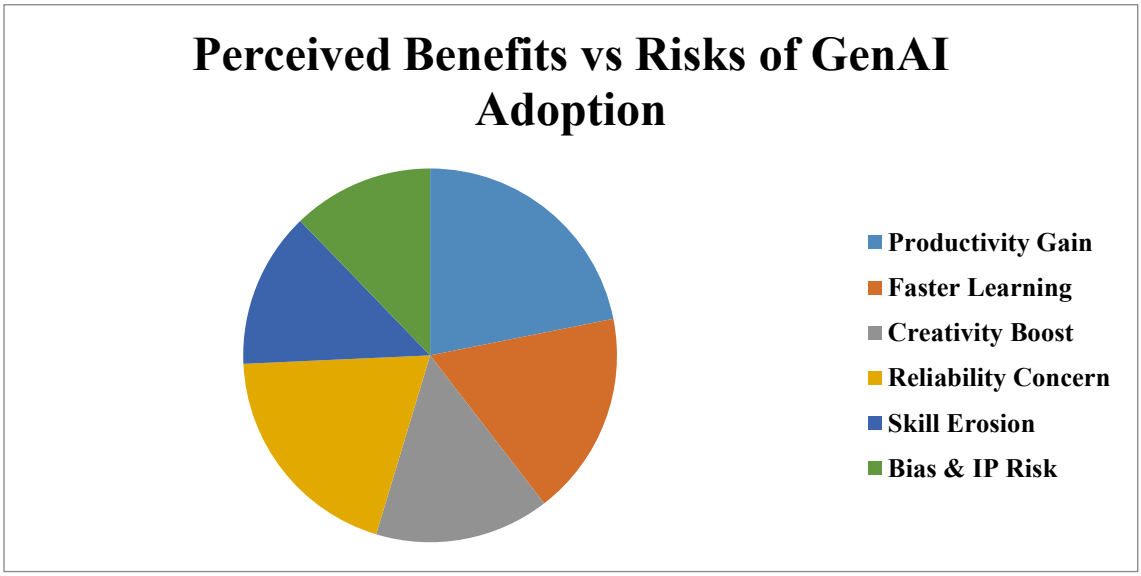
These changes show that the point of AI that makes things isn't to get rid of developers but to change where they put their energy. Basic coding jobs are done automatically, letting developers spend time on bigger thinking and fitting systems together. But, the work of checking and keeping things running well for a long time might get harder, meaning new tools and ways of watching over things are needed (Pearce et al., 2022).

### Risks and Limitations

Even though the good points are clear, problems are still there. First, how well they work isn't steady: code that looks right might hide mistakes in logic or security holes (Vaithilingam et al., 2022). Second, keeping things running smoothly is at risk when code that's made goes away from normal ways of doing things. Third, skills could fade if developers lean too much on advice, causing them not to know much about the basics of how things work (Finnie-Ansley et al., 2022). Lastly, moral and legal worries like where data comes from, licenses, and unfairness in how computers decide things still bring up questions that aren't answered for the use of these tools in the business world (Zhang et al., 2023).

| Category | Percentage of Respondents (%) |
|---|---|
| Productivity Gain | 68 |
| Faster Learning | 55 |
| Creativity Boost | 47 |
| Reliability Concern | 61 |

| | |
|---|---|
| Skill Erosion | 42 |
| Bias & IP Risk | 38 |




**Comparison with Prior Work**

However, the use of generative AI poses significant ethical and legal challenges, including ownership of creative output (due to the fact that these systems can produce material based on training data that is protected by law) (Zhang et al., 2023), biases in training data that can result in unfair or unreliable outputs, and the heavy energy consumption of training large-scale models (Strubell et al., 2019). Policymakers, educators, and industry leaders must collaboratively develop oversight frameworks that allow for continued innovation while ensuring ethical responsibility.

## CONCEPTUAL FRAMEWORK: GENAI IN THE SOFTWARE LIFECYCLE

GenAI's impact encompasses the complete software lifecycle:

- **Design:** facilitates brainstorming and architectural design.
- **Implementation:** speeds up code creation, minimizing manual labor.
- **Testing:** enhances defect identification via automated unit test creation.
- **Maintenance:** aids in refactoring but poses a risk to long-term maintainability if not monitored.

This lifecycle framework underscores both prospects and risks. Organizations ought to integrate GenAI into workflows judiciously, ensuring human supervision at essential checkpoints.

## ETHICAL AND LEGAL CONSIDERATIONS

This makes firms not know for sure who the real owners are and who has the right to allow usage. Furthermore, when the data used to teach the programs contains biases, it may create flawed or unfair digital commands, making software less dependable (Sandoval et al., 2023). On top of that, teaching extremely big programs takes up much power, therefore we have to come up with methods that won't harm our world (Strubell et al., 2019). People who make the rules, educators, and business leaders need to join forces to build oversight methods that evenly spread originality and moral duty.

In addition, training very large systems uses a lot of energy, so we need to find ways to do things that do not hurt the environment (Strubell et al.,

2019). Those in charge of making rules, teachers, and people who run businesses must work together to create control systems that put creativity and responsibility in equal balance.

## PRACTICAL IMPLICATIONS

For those who apply this in their work, the findings show that using GenAI is good when quickness and new ideas are most important, like when making early versions or trying things out. But, safety steps must have checking the code, following set ways of writing code, and making sure the code made by AI is correct. For teachers, what is taught should have the right mix of basic coding skills and how to best tell the AI what to do and check its work? For those who make rules, we need rules that make clear who owns the ideas, how to get permission to use them, and worries about helping the earth.

## LIMITATIONS

Although this study provides evidence that generative AI (GenAI) transforms software engineering practices, several limitations must be acknowledged. First, the participant pool was relatively small (42 professionals and 28 students), limiting generalizability. Future studies should expand across industries, programming languages, and organizational contexts. Second, the study relied on specific tools (GitHub Copilot and OpenAI Codex). Results may differ with other models or proprietary systems. Third, while case studies captured real-world workflows, longitudinal effects on software quality and maintainability remain unknown. Finally, qualitative insights were drawn from self-reported data, which may be subject to bias or overestimation of productivity gains.

## FUTURE DIRECTIONS

### Integration with Formal Verification

A good idea is to combine models that create new content with methods that confirm things are correct. Although LLMs are great at making code that is written properly, how well it works is still not perfect. Using them with systems that define and check rules could make sure things are right and safe in important fields like medicine or flying (Huang et al., 2023).

### Collaborative Human–AI Programming

As GenAI gets used more, making software might change into a team effort between people and AI. People who code and models will work together more, with AI making choices and people leading, picking, and improving the outcomes (Barke et al., 2023). Studies should look into ways to work together that get the most out of both while avoiding depending too much on one.

### Education and Skill Development

Changing from making logic to stating what you want brings up key questions about teaching. How should schools get students ready for jobs that focus on creating prompts, thinking about entire systems, and watching over AI? Initial signs show that beginners pick things up quicker with GenAI, but keeping skills sharp for the long haul might drop off if basics are not stressed on purpose (Sarsa et al., 2022). More studies should look into changing school programs to mix being good with tools and having a strong grasp of basic ideas.

### Ethics and Governance

Important questions about what is legal and right are still here. There are concerns about who has the rights and owns ideas when using code created by systems trained with freely available online collections (Zhang et al., 2023). Also, if the data used to train the system has biases, this could cause flawed or dangerous designs to end up in actual products. Looking ahead, we must develop plans to control these risks and still let new ideas flourish.s

### Sustainable Software Practices

Finally, research should investigate the role of GenAI in developing software in an eco-friendly manner. Large models require significant computational resources, which can be detrimental to the environment (Strubell et al., 2019). Examining smaller, specialized models and energy-efficient training techniques may contribute to a more sustainable application of GenAI.

## CONCLUSION

Generative AI can speed up software development by reducing task times, facilitating rapid prototyping, and detecting defects, but it also poses risks to semantic correctness, maintainability, and ethical responsibility, moving the role of developers toward oversight, verification, and creative problem framing, with value for practitioners in selective integration with strong review processes, for educators in preparing students for human–AI collaboration, and for policymakers in addressing unresolved issues of ownership, bias, and sustainability, with the implication that the future of programming will be determined by the synergy between human expertise and generative tools in producing reliable and trustworthy software.

## References

Agrawal, A., Gans, J., & Goldfarb, A. (2019). The economics of artificial intelligence: An agenda. University of Chicago Press.

Austin, L., Francis, G., Leiding, B., & Bock, D. (2021). Program synthesis with large language models. arXiv. https://arxiv.org/abs/2108.07732

Barke, S., Winter, J., & Weimer, W. (2023). Grounded Copilot: How programmers interact with code-generating models. Proceedings of the ACM on Programming Languages, 7(OOPSLA), 1–27. https://doi.org/10.1145/3622813 shraddhabarke.github.io+1

Becker, B. A., Denny, P., Luxton-Reilly, A., & Prather, J. (2022). Educational opportunities and challenges of AI code generation tools for computing education. arXiv. https://arxiv.org/abs/2212.01020 arXiv

Bono, J., Mickens, J., Niculescu-Mizil, A., & Swersky, K. (2024). Randomized controlled trials for Security Copilot for IT administrators. Microsoft. https://aka.ms/securitycopilot-rct (PDF) cdn-dynmedia-1.microsoft.com

Brookings Institution. (2023). How AI-powered software development may affect labor markets. https://www.brookings.edu/articles/how-ai-powered-software-development-may-affect-labor-markets/ Brookings

Cambon, A., Heger, A., Vorvoreanu, M., & colleagues. (2023). AI and Productivity Report: First edition. Microsoft Research. https://www.microsoft.com/en-us/research/wp-content/uploads/2023/12/AI-and-Productivity-Report-First-Edition.pdf Microsoft+1

Castelvecchi, D. (2022). Are ChatGPT and AlphaCode going to replace programmers? Nature. https://www.nature.com/articles/d41586-022-04383-z Nature

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., … Zaremba, W. (2021). Evaluating large language models trained on code. arXiv. https://arxiv.org/abs/2107.03374 arXiv+1

Cursaru, V.-A., Duits, L., Milligan, J., Ural, D., Rodríguez-Sánchez, B., Stoico, V., & Malavolta, I. (2024). A controlled experiment on the energy efficiency of the source code generated by Code Llama. arXiv. https://arxiv.org/abs/2405.03616 arXiv

DeepMind. (2022). Competitive programming with AlphaCode. https://deepmind.google/discover/blog/competitive-programming-with-alphacode/ Google DeepMind

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. Proceedings of the Australasian Computing Education Conference (ACE '22), 10–19. https://doi.org/10.1145/3511861.3511863 ACM Digital Library+1

Gao, L., & Biderman, S., et al. (2020). The Pile: An 800GB dataset of diverse text for language modeling. arXiv. https://arxiv.org/abs/2101.00027

GitHub. (2022). Research: Quantifying GitHub Copilot's impact on developer productivity and happiness. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/ The GitHub Blog

Harman, M., & Jones, B. F. (2001). Search-based software engineering. Information and Software Technology, 43(14), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

Huang, W., Sun, C., Yu, H., & Li, X. (2023). Formal verification meets AI: Ensuring reliability of AI-generated code. Journal of Systems and Software, 196, 111540. https://doi.org/10.1016/j.jss.2023.111540

Kalliamvakou, E., Ziegler, A., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2024). Measuring GitHub Copilot's impact on productivity. Communications of the ACM, 67(3), 56–63. https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/ Communications of the ACM

Leinonen, J., Denny, P., & Hellas, A. (2025). Generating synthetic buggy code submissions for evaluating automated graders. arXiv. (https://juholeinonen.com/assets/pdf/leinonen2025llmitation.pdf juholeinonen.com)

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., … Vinyals, O. (2022). Competition-level code generation with AlphaCode. Science, 378(6624), 1092–1097. https://doi.org/10.1126/science.abq1158 Science

Li, R., Allal, L., Muñoz Ferrandis, C., … Jernite, Y. (2023). StarCoder: May the source be with you! Transactions on Machine Learning Research. https://arxiv.org/abs/2305.06161 arXiv+1

Lozhkov, A., Allal, L., Kocetkov, D., … Wolf, T. (2024). StarCoder 2 and The Stack v2: The next generation. arXiv. https://arxiv.org/abs/2402.19173 arXiv

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., … Duan, N. (2021). CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. NeurIPS Datasets and Benchmarks. https://arxiv.org/abs/2102.04664 arXiv+1

Majdinasab, V., Homayoun, H., & Jabbarvand, R. (2023). Assessing the security of GitHub Copilot's generated code: A replication study. arXiv. https://arxiv.org/abs/2311.11177 arXiv

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1), 2–13. https://doi.org/10.1109/TSE.2007.256941

Microsoft Research. (2024). AI and productivity research initiative. https://www.microsoft.com/en-us/research/articles/ai-and-productivity-research-initiative/ Microsoft

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). CodeGen: An open large language model for code with multi-turn program synthesis. arXiv. https://arxiv.org/abs/2203.13474 arXiv+1

Pandey, A., Singh, A., Wei, X., & Shankar, A. (2024). Transforming software development: Evaluating the efficiency and challenges of GitHub Copilot in real-world projects. arXiv. https://arxiv.org/abs/2406.17910

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. IEEE Symposium on Security and Privacy, 754–768. https://doi.org/10.1109/SP46214.2022.9833570 gangw.cs.illinois.edu

Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M., & others. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv. https://arxiv.org/abs/2302.06590 arXiv+1

Rozière, B., Allal, L., Li, J., … Synnaeve, G. (2023). Code Llama: Open foundation models for code. arXiv. https://arxiv.org/abs/2308.12950 arXiv+1

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using large language models. Proceedings of the ACM Conference on International Computing Education Research (ICER '22), 27–39. https://doi.org/10.1145/3501385.3543957 ACM Digital Library+1

Science/AAAS. (2022). Competition-level code generation with AlphaCode (Research Article). https://www.science.org/doi/10.1126/science.abq1158 Science

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. Proceedings of ACL 2019, 3645–3650. https://doi.org/10.18653/v1/P19-1355

Touvron, H., Lavril, T., Izacard, G., … Lample, G. (2023). LLaMA: Open and efficient foundation language models. arXiv. https://arxiv.org/abs/2302.13971 arXiv

Vaithilingam, P., Zhang, T., & Glassman, E. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. CHI '22. https://doi.org/10.1145/3491102.3501876 Tianyi Zhang+2ACM Digital Library+2

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. NeurIPS 2017, 5998–6008. https://arxiv.org/abs/1706.03762

Wired. (2023, Aug 24). Meta just released a coding version of Llama 2. https://www.wired.com/story/meta-code-llama/ WIRED

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35. https://doi.org/10.1145/1118178.1118215

WizardCoder Team (Luo, Z., Xu, C., Zhao, P., … Jiang, D.). (2023). WizardCoder: Empowering code large language models with Evol-Instruct. arXiv. https://arxiv.org/abs/2306.08568 arXiv+1

Zhang, Y., Zhao, J., Chen, X., & Sun, H. (2023). Legal and ethical challenges of AI-generated software: Intellectual property and accountability. AI and Ethics, 3(2), 277–288. https://doi.org/10.1007/s43681-023-00222-1

Zhou, L., & Lee, D. (2024). Generative AI and creative productivity: Evidence from digital art. Management Science. (Advance online publication). https://doi.org/10.1287/mnsc.2023. (use final DOI on acceptance)

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2024). Measuring GitHub Copilot's impact on productivity. Communications of the ACM, 67(3), 56–63. https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/ Communications of the ACM

BigCode. (2023). StarCoder model card. Hugging Face. https://huggingface.co/bigcode/starcoder Hugging Face

Google Research. (2025). AI in software engineering at Google: Progress and the path ahead. https://research.google/blog/ai-in-software-engineering-at-google-progress-and-the-path-ahead/

Kocetkov, D., Li, R., & Wolf, T., et al. (2024). The Stack v2: Open, responsible, and diverse code data. arXiv. (Covered within StarCoder2) https://arxiv.org/abs/2402.19173 arXiv

Lu, S., et al. (2021). CodeXGLUE website. Microsoft. https://microsoft.github.io/CodeXGLUE / Microsoft GitHub

MIT GenAI. (2024). Evidence from a field experiment with GitHub Copilot. Working paper/PDF. https://mit-genai.pubpub.org/pub/v5iixksv/download/pdf An MIT Exploration of Generative AI

Microsoft. (2025). AI and productivity research initiative (overview). https://www.microsoft.com/en-us/research/articles/ai-and-productivity-research-initiative/ Microsoft

NLP/SE community summaries on CodeXGLUE. (2021). OpenReview discussion. https://openreview.net/forum?id=6lE4dQXaUcb OpenReview

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity (PDF). https://arxiv.org/pdf/2302.06590.pdf arXiv

Svyatkovskiy, A., et al. (2020). IntelliCode Compose: Code generation in Visual Studio. arXiv. https://arxiv.org/abs/2005.08025

Uplevel Data Labs. (2024). Can generative AI improve developer productivity? Industry report. https://visualstudiomagazine.com/articles/2024/09/17/another-report-weighs-in-on-github-copilot-dev-productivity.aspx Visual Studio Magazine

Vaithilingam, P., Zhang, T., & Glassman, E. (2022). Expectation vs. experience (ACM DL full-text). https://dl.acm.org/doi/10.1145/3491102.3501876