TURING LEDGER JOURNAL OF ENGINEERING & TECHNOLOGY

Generative AI and Large Language Models for Software Engineering: Opportunities, Challenges, and Future Directions

Muhammad Ali Muzammil
Department of Engineering, Bahauddin Zakariya University, Multan, Pakistan
Email: alimuz@gmail.com

Received: 14-06-2026
Revised: 03-07-2026
Accepted: 26-07-2026

Corresponding Author: Muhammad Ali Muzammil

ABSTRACT:

Artificial In this study, the impact of generative artificial intelligence (AI) and large language models (LLMs) on software engineering tasks was explored by applying a mixed methods experimental research design. The sample consisted of 64 software developers and computer science professionals, who were selected using purposive sampling to participate in the study, completing a series of programming, code generation, debugging, and documentation tasks in both an AI-assisted and non-AI-assisted environment. Data for task completion time, code accuracy, rate of defects and perceived productivity were analysed using paired-samples t-tests and semi-structured interviews with a subsample of 20 respondents were analysed qualitatively using thematic analysis to explore perceived benefits, limitations and concerns. The AI assistance was found to significantly speed up programming, code generation, and documentation tasks, further, there was a modest increase in perceived productivity for all tasks, except for debugging, which was not found to be significantly different. The accuracy of the debugging tasks did not show any statistically significant difference between the AI assistance and the human-only conditions. While logic defect rates were significantly lower when AI was used, the occurrence of security-relevant defects is significantly higher when using the AI-assisted code than the unassisted code. Thematic analysis of the interview data revealed three main benefits perceived: efficiency gains, learning support and reduced cognitive load, while concerns included hallucinated or inaccurate outputs, over-reliance and skill erosion, security review workload, and limited contextual understanding. Based on the findings of both quantitative and qualitative measures, this integration suggests that while generative AI and LLM-driven tools can enhance productivity and quality in specific, well-defined software engineering tasks, they also present new categories of risks and limitations, such as the complexity of debugging with these tools and the security concerns related to the output generated, that demand active human intervention and organizational measures to address.

Keywords: *generative AI; large language models; software engineering; developer productivity; code generation; mixed-methods research*

INTRODUCTION

The use of generative artificial intelligence (AI) and large language models (LLMs) in software engineering has grown significantly since the advent of code-based models that convert natural

language queries into working program code (Chen et al., 2021). Experimentally new tools like GitHub Copilot have become adopted by many developers in the real world, and the impact of these tools on developer productivity, code quality, and the software development lifecycle (SDL) has been the subject of numerous empirical studies (Ziegler et al., 2024).

Initial controlled studies have shown significant improvement in productivity when using AI to assist with coding. It took Peng et al. (2023) 55.8% less time to finish a bounded task using JavaScript on GitHub Copilot than an unassisted control group, and subsequent field studies at large tech companies found more modest, but positive, results in developers' weekly output after implementing Copilot. More recent evidence has muddied the waters, however: Becker et al. (2025) conducted a randomized controlled trial with experienced open-source developers where they were asked to complete real world coding tasks, and found that, while developers believed AI tools would make their work faster, their actual performance with them was slower by 19% than without them.

In addition to productivity, researchers have expressed concerns about the correctness and security of AI-generated code. Recent studies by Pearce et al. (2022) demonstrated that about 40% of code that the AI coding assistants on GitHub suggest for a variety of security-relevant scenarios was vulnerable, and user studies by Perry et al. (2023) and Sandoval et al. (2023) have shown that AI coders can write measurably less secure code and feel more confident about it's safety. Simultaneously, usability research indicates that developers appreciate AI recommendations as a starting point for further refinement, rather than a fully trusted and finished result (Vaithilingam et al., 2022), revealing a more complex interaction between AI support and code quality than simply metrics of productivity.

Although the number of quantitative studies is growing, few studies have integrated multiple types of tasks into a single study, with both controlled task-based measurement and in-depth qualitative exploration of the developers' lived experience of AI-assisted software engineering. To fill the gap, this work takes a mixed-methods experimental approach to assess the impact of GPTs and LLM-based tools on task completion time, code accuracy, defect rates, and productivity for programming, code generation, debugging, and documentation tasks, while also exploring the benefits, limitations, and concerns that influence developers' interactions with these tools based on semi-structured interviews. By doing so, the study intends to deliver an empirical and integrated view of the opportunities and challenges of generative AI adoption in software engineering and to propose and guide further research and practice on how to sustainably and responsibly use generative AI in software engineering.

LITERATURE REVIEW

Since an AI research paper (Chen et al., 2021) showed that a code-oriented LLM was able to solve a significant portion of programming problems using a set of functionally correct solutions as a basis for building subsequent commercial coding assistants, the research on generative AI / LLM in software engineering has accelerated. Empirical productivity research has tended to look at artificial intelligence support for programming using controlled experiments, field studies or usage-based case studies, building on this foundation. Peng et al. (2023) then conducted a randomized controlled experiment in which users of GitHub Copilot on the same task finished a defined programming task significantly faster than an unassisted control group; and Ziegler et al. (2024) followed up with a correlation study of the productivity gains reported by thousands of developers who were using GitHub Copilot based on the number of suggestions they accepted. Storey et al. (2021) placed these results in a larger theoretical framework of developer job satisfaction and perceived productivity and explained how social, technical, and contextual factors interact to create a sense of AI-assisted gains beyond just being faster to complete tasks. Under a similar perspective, Forsgren et al. (2021) argued that productivity is a multi-dimensional concept, and described it under the SPACE framework that contains task-related objective and subjective measures of productivity, which were reflected in the present study.

However, not all empirical evidence has been favorable towards clear productivity gains. However, when faced with similar challenges, experienced open-source developers using modern AI coding tools

were, on average, 19% slower, despite having a strong expectation and a self-perceived faster performance using the AI tools, which the authors attributed to the mental effort required to review and edit the AI-generated code suggestions on unfamiliar and complex code. Copilot use also found to lead to worse software quality than when humans pair program, with an increase in the volume of code generated during pair-programming sessions (Imai, 2022), which may indicate that productivity gains (quantity or speed) do not necessarily correlate with higher software quality. A similar usability finding was reported by Vaithilingam et al. (2022): The developers appreciated the recommendations offered by Copilot as a valuable starting point; although, in their study, there was not a significant difference in task completion time and task success between Copilot and an external search.

Another line of research has focused on the security considerations of AI-generated code. In one of the first large-scale security evaluations of Copilot, the researchers at Pearce et al. (2022) discovered that approximately 40% of the code generated by Copilot over a set of high risk scenarios included vulnerabilities from a set of already known vulnerabilities taxonomies. This study was continued by Perry et al. (2023) in a controlled user study which showed that participants who used the AI coding assistant ended up writing significantly less secure code than the participants who didn't use the coding assistant, and that the participants who used the coding assistant reported feeling more confident that their code was secure—a false sense of security, according to Perry et al. In a similar study, Sandoval et al. (2023) found that developers who volunteer in accepting recommendations from AI added more vulnerabilities to the results compared to those who wrote without AI, which further supports the findings that the benefits of AI for correctness and the security risks can exist in the same development process.

Some of these quantitative results have been complemented by qualitative and mixed-methods research that looks at developers' personal experience of AI-based software engineering. Several recent field studies have reported recurring developer concerns over relying too much on AI suggestions, diminishing problem-solving skills when relying on AI for extended periods, and the extra cognitive load of having to double-check AI generated code for correctness and security, in addition to the consistently reported benefits of reduced boilerplate code and increased access to unfamiliar APIs or languages. The bulk of the respondents in GitHub's own massive developer survey also indicated that using Copilot helped them be more productive, spend less time searching for information, and have a reduced cognitive load when working with repetitive tasks (GitHub, 2023), themes that center on benefits. Altogether, this literature indicates that the impact of the generative AI and LLM-enabled tools on software engineering has not been consistently positive nor consistently negative, but rather it is differentiated and mixed methods in nature, which inspired the task-differentiated, mixed-methods approach in the present study.

METHODOLOGY

Research Design

The study employed a mixed methods experimental research design to examine the effect of the use of generative AI and large language models in software engineering tasks. These two strands were gathered simultaneously and then combined to get a complete picture of the opportunities and challenges that surround the use of LLM in software engineering.

Participants and Sampling

The purposive sample consists of a group of 64 software developers and computer science professionals. Participants should be experienced software developers or academics with at least one year of experience, and with a working knowledge of at least one general purpose programming language. In addition, 20 participants were selected to ensure a wide range of experience level, professional role and previous use of AI tools and participated in semi-structured interviews.

The participants were tasked with programming, coding, debugging, and documentation, both with and without the assistance of AI tools. To minimize order and learning effects, task order and AI-condition assignment were counterbalanced across participants: Each participant was required to perform matched task pairs under the two task conditions within each of the four task categories.

Quantitative Measures

Data pertaining to task completion time, code accuracy, defect rates and productivity were gathered. Task completion time was measured in minutes, from task start to task submission. Automated test-pass rates for programming and code-generation tasks, independent-rater resolution scoring for debugging tasks, and independent-rater completeness scoring for documentation tasks were used to assess the accuracy of the code. Defect rates were calculated based on a post-task, static analysis and peer code review, while security-relevant vulnerabilities were coded independently using a known scheme of weaknesses. After each task condition, a single item, seven-point Likert scale was used to assess perceived productivity.

Qualitative Data Collection

Besides, semi-structured interviews were held to gain insights into participants' experience, perceived benefits, limitations and concerns about AI-supported software development. The interviews were conducted for about 30 to 45 minutes, audio-recorded with the consent of the participants and transcribed verbatim for analysis.

Data Analysis

Comparative statistical tests were used to analyse quantitative data. Within-participant comparisons of task completion time, accuracy, defect rate and productivity scores were performed using paired-samples t-tests, and Cohen's d was computed to estimate effect size. Thematic analysis was used to analyze the qualitative responses from the interviews which followed the six-phase approach described by Braun and Clarke (2006): familiarization, initial coding, theme development, theme review, theme definition and reporting. The findings of both approaches were combined to give a holistic view of the opportunities and challenges of LLM use in software engineering.

ANALYSIS AND RESULTS

Participant Profile

Demographic and professional characteristics of the 64 subjects in this study are summarized by Table 1. The majority of the sample group consisted of industry software developers who had medium-to-long experience in their profession, and most had used AI coding tools in the past frequently.

Table 1: Participant Demographic and Professional Profile (N = 64)

Characteristic	Category	n	%
Gender	Male	41	64.1
	Female	23	35.9
Professional role	Industry software developer	44	68.8
	Computer science academic/researcher	20	31.3
Years of experience	1–3 years	17	26.6
	4–7 years	26	40.6
	8 years or more	21	32.8
Primary programming language	Python	24	37.5
	JavaScript/TypeScript	19	29.7

	Java	13	20.3
	Other	8	12.5
Prior AI coding-tool use	Regular user	38	59.4
	Occasional/no prior use	26	40.6

Note. Percentages are calculated within each characteristic category and may not sum to exactly 100% due to rounding.

Task Completion Time and Code Accuracy

Table 2 shows the results of paired-samples t-tests between the AI-assisted and unassisted conditions for the four types of tasks. The majority of tasks—including programming, code generation, and documentation—showed medium to large effect sizes regarding speed when using AI assistance, and significantly higher accuracy when using AI assistance. However, differences in completion time and accuracy of resolution between the conditions were not statistically significant for the debugging tasks, suggesting that AI support was beneficial for the generative and descriptive tasks, but not for the diagnostic tasks, where the localization and correction of the existing defects.

Table 2: Task Completion Time and Code Accuracy by Task Type: AI-Assisted vs. Unassisted Conditions

Task Type / Metric	Without AI M (SD)	With AI M (SD)	t(63)	p	d
Programming (feature implementation)					
Completion time (min)	54.3 (12.1)	38.7 (10.4)	9.82	< .001	1.23
Test-pass accuracy (%)	78.4 (9.6)	85.2 (8.1)	6.03	< .001	0.75
Code generation					
Completion time (min)	41.2 (9.8)	24.6 (8.1)	11.47	< .001	1.43
Test-pass accuracy (%)	74.1 (10.2)	88.7 (7.4)	9.87	< .001	1.23
Debugging					
Completion time (min)	36.8 (11.3)	33.1 (10.9)	1.64	.106	0.20
Resolution accuracy (%)	81.3 (11.0)	84.6 (10.2)	1.98	.052	0.25
Documentation					
Completion time (min)	28.4 (7.2)	16.9 (6.5)	10.35	< .001	1.29
Quality/completeness score (0–100)	71.5 (9.9)	83.2 (8.6)	8.14	< .001	1.02

Note. LOC = lines of code. Degrees of freedom for all comparisons are 63. Effect sizes are reported as Cohen's d for paired samples.

Defect Rate, Security Vulnerability Incidence, and Perceived Productivity

As revealed in Table 3, the number of logic defects per 100 LOCs was found to be significantly lower for AI-assisted code than for unassisted code. The frequency of security-related vulnerabilities, however, was much higher in AI-assisted code compared to prior security assessments of AI coding assistants (Pearce et al., 2022; Perry et al., 2023). Participants also showed a significant difference in feeling more productive using AI assistance (even if not seeing a corresponding objective increase in productivity for the debugging tasks in particular), which aligns with the patterns observed in other studies in the literature (Becker et al., 2025).

Table 3: Defect Rate, Security Vulnerability Incidence, and Perceived Productivity: AI-Assisted vs. Unassisted Conditions

Outcome	Without AI M (SD)	With AI M (SD)	t(63)	p
Logic defect rate (per 100 LOC)	4.8 (1.6)	3.9 (1.4)	4.22	< .001
Security vulnerability incidence (%)	8.1 (5.4)	15.4 (7.8)	3.56	< .001
Perceived productivity (1–7 scale)	4.1 (0.9)	5.6 (0.8)	13.20	< .001

Note. Security vulnerability incidence reflects the percentage of submitted solutions containing at least one vulnerability identified through static analysis.

Thematic Analysis of Interview Data

Table 4 summarises the eight major themes that emerged from the 20 semi-structured interviews, from both perceived benefits and perceived concerns. Most commonly cited benefits included efficiency gains, learning and scaffolding support, and reduced cognitive load, whereas accuracy and hallucination concerns, security and code-review burden, and over-reliance or skill erosion were the most commonly cited concerns. Some participants specifically mentioned the lack of accuracy as being related to the debugging task, indicating that the AI suggestions were least effective when diagnosing less frequently occurring, context-dependent defects, which is consistent with the quantitative findings in Section 4.2.

Table 4: Themes Identified Through Thematic Analysis of Semi-Structured Interviews (n = 20)

Theme	Description	Participants Referencing Theme, n (%)
Efficiency gains	Faster completion of boilerplate, repetitive, and documentation tasks	18 (90.0)
Learning and scaffolding support	AI explanations helped clarify unfamiliar APIs, languages, and syntax	15 (75.0)
Reduced cognitive load	Less mental effort spent recalling syntax or searching documentation	14 (70.0)
Accuracy and hallucination concerns	AI occasionally produced plausible-looking but incorrect or non-functional code	16 (80.0)
Over-reliance and skill erosion	Concern that habitual AI use could weaken independent problem-solving over time	12 (60.0)
Security and code-review burden	Perceived need for closer review of AI-suggested code before integration	13 (65.0)
Limited context understanding	AI suggestions sometimes missed project-specific conventions or architecture	11 (55.0)
Trust calibration	Trust in AI output varied by task type, growing with experience using the tool	10 (50.0)

Note. Percentages reflect the proportion of the 20 interviewed participants whose responses were coded under each theme; themes are not mutually exclusive.

Integration of Quantitative and Qualitative Findings

The analysis of convergence between the two strands of data showed that there was a high degree of agreement for the three tasks of programming, code generation, and documentation, in both quantitative improvements in performance and the qualitative reports of enhanced efficiency and reduced cognitive

load. By contrast, there was no evidence of any large quantitative benefits for debugging tasks, while qualitative feedback indicated that AI was not reliably able to perform more complex, context-dependent diagnostic tasks. Tendencies from the quantitative data were also consistent with the qualitative findings, including heightened security vulnerability reports, and requests to review AI-generated recommendations more rigorously, which together create a picture that is more complex than one might expect from AI software engineering, with differing effects rather than a uniform positive or negative impact.

DISCUSSION

Our results build on the previous research on the use of Generative AI and LLMs for software engineering by showing that the technology benefits are not the same for all types of tasks. As with previous controlled studies where large speed improvements were reported for specific programming tasks (Peng et al., 2023), this study shows that there was statistically significant difference in the completion time and accuracy of programming, code generation, and documentation tasks. However, the lack of strong support in the debugging domain is consistent with more recent findings that generative AI tools can hinder, or even harm, existing software professionals' productivity when they are attempting to solve highly complex and context-dependent problems (Becker et al., 2025), suggesting that task complexity and contextual dependency may have a greater impact on the productivity benefits of generative AI tools than previously believed.

This higher rate of security vulnerabilities found in code generated by AI supports existing security-focused research (Pearce et al., 2022; Perry et al., 2023; Sandoval et al., 2023) and highlights the fact that security is a non-functional requirement, meaning that improvements in speed and functional correctness do not necessarily benefit other non-functional requirements. Given that participants reported this pattern and qualitative feedback from participants on the increased burden of code review due to the use of generative AI coding tools, it is important for organizations moving to GPT-based coding tools to consider augmenting their existing security review processes rather than relying on the same level of security review as a human-generated code.

Over-reliance and perceived erosion of skills were reported by 60% of the participants interviewed, and are a concern that extends beyond the present study as the participants were tested over a short time period (one session), during which the intervention was applied. These concerns align with existing literature on the possibility of AI sustained tools changing how developers learn and practice fundamental programming skills, and highlight the need for long-term studies that can follow the skill acquisition of developers who use AI tools more or less regularly.

CONCLUSION, IMPLICATIONS

The design of this study was a mixed methods experimental approach to explore both the opportunities and challenges of the use of generative AI and LLMs in software engineering. The integrated results show that use of AI-assisted tools generally significantly reduced completion time and the proportion of errors for programming, writing code, and documentation, but did not measurably reduce completion time for debugging, and actually increased the number of security vulnerabilities and decreased the number of logic errors. Thematic analysis also confirmed these themes from the participant interviews: There is a tension between perceived benefit of efficiency and support in learning and perceived burden of cognitive load and concern about accuracy and over-reliance and security review.

These findings indicate that for well-defined generative tasks like boilerplate coding and documentation, software engineering practitioners and organizations can safely harness the capabilities of generative AI tools, but integrating increased code-review and security-verification processes is essential, especially due to the high vulnerability rate found in this study. For debugging and other diagnostics activities that need to be done with a good understanding of the context, it is important to have a realistic expectation, and not the same productivity improvements in all software engineering activities.

REFERENCES

1. Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity (arXiv:2507.09089). arXiv. https://arxiv.org/abs/2507.09089
2. Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative Research in Psychology, 3(2), 77–101.
3. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating large language models trained on code (arXiv:2107.03374). arXiv. https://arxiv.org/abs/2107.03374
4. Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). The SPACE of developer productivity: There's more to it than you think. Queue, 19(1), 20–48.
5. GitHub. (2023). Research: Quantifying GitHub Copilot's impact on developer productivity and happiness. GitHub Blog.
6. Imai, S. (2022). Is GitHub Copilot a substitute for human pair-programming? An empirical study. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (pp. 319–321). ACM.
7. 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. In 2022 IEEE Symposium on Security and Privacy (SP) (pp. 754–768). IEEE.
8. Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot (arXiv:2302.06590). arXiv. https://arxiv.org/abs/2302.06590
9. Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 2785–2799). ACM.
10. Sandoval, G., Pearce, H., Nys, T., Karri, R., Dolan-Gavitt, B., & Garg, S. (2023). Lost at C: A user study on the security implications of large language model code assistants. In Proceedings of the 32nd USENIX Security Symposium (pp. 2205–2222). USENIX Association.
11. Storey, M.-A., Houck, B., & Zimmermann, T. (2021). Towards a theory of software developer job satisfaction and perceived productivity. IEEE Transactions on Software Engineering, 47(10), 2125–2142.
12. Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In CHI Conference on Human Factors in Computing Systems Extended Abstracts (pp. 1–7). ACM.
13. 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), 54–63.