IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 28, Issue 3, Ser. II (May. – June. 2026), PP 22-28
www.iosrjournals.org

Disadvantages of AI-Assisted Code Generation in Software Development

Feruzbek Azimov[1], Edip Senyurek[2]
[1](Department of Computer Engineering, Vistula University, Poland)
[2](Department of Computer Engineering, Vistula University, Poland)

Abstract: *Despite being used by over 80% of software developers, AI coding tools have been quick to integrate into professional software development but slow to be studied. In this article, some of those drawbacks are discussed. According to PRISMA guidelines, 28 primary studies were retrieved from IEEE Xplore, ACM Digital Library, Scopus and Google Scholar between 2020 and 2026, and categorized into six categories. Security: About 40% of the code generated was vulnerable, and 62% under formal verification. Code quality and technical debt: churn more than doubled and refactoring was cut in half. Skill atrophy: Randomized trial showed a 17 point comprehension loss of AI-assisted learners. Privacy and IP risk: unresolved copyleft license propagation. Productivity paradox: Developers were 19% slower, but thought they were 20% faster. Ambiguous responsibility for AI errors at the workplace. Such instruments can bring about local productivity gains, but the literature points to a number of reinforcing risks. Most research has been short-term and lab-based and thus provides only a limited perspective.*

Keywords: *AI-assisted code generation, developer productivity, large language models, software engineering, software security, technical debt*

Date of Submission: 28-05-2026 Date of Acceptance: 06-06-2026

I. Introduction

In just about four years, generative AI is a standard part of professional software engineering. The adoption process of developer tooling has historically few parallels: the 2024 Stack Overflow survey found that 76% of professional developers were using or intending to use AI tools, and some 82% of those developers who used AI tools used them for writing code [1]. While the general sentiment about such tools dropped from more than 70% to around 60% of the developers surveyed, the rate of adoption increased to about 80% of developers in all by the 2025 iteration of the same survey [2]. This situation — almost universal acceptance and dwindling confidence — is the context of the problem that this article tackles.

Industry coverage of AI-assisted coding has been dominated by claims of productivity uplift. Vendor-sponsored studies frequently quote double-digit percentage improvements in task completion time and developer satisfaction. Independent academic and grey-literature research, however, presents a more mixed picture. The 2024 DORA report, drawing on responses from approximately 39,000 tech professionals, found that a 25% increase in AI adoption correlated with a 1.5% decrease in delivery throughput and a 7.2% decrease in delivery stability — the two metrics the DevOps community has spent more than a decade attempting to improve [3]. The METR randomized controlled trial by Becker et al. (2025) measured what experienced open-source developers actually gained from AI. It found they took 19% longer to complete real tasks on their own repositories with early-2025 AI tools than without, despite reporting a perceived speedup of approximately 20% [4]. Anthropic's own randomized trial published in January 2026 reported that developers using AI assistance while learning a new library scored approximately 17% lower on comprehension assessments than a matched control group [5].

These findings collectively suggest that the academic literature, the industry survey literature, and the marketing literature about AI code generation are not aligned. The research question motivating this article is therefore: what are the documented disadvantages of AI-assisted code generation in professional software development, as reported in the peer-reviewed and high-quality industry literature published between 2020 and 2026?

The scope is limited to professional software engineering. Educational studies, opinion pieces, and non-software uses of generative AI fall outside this review. The article has three objectives. First, it identifies and groups the recurring disadvantages reported across the literature. Second, it weighs how strong and consistent the evidence is for each one. Third, it points out where the current evidence falls short and which questions matter most for future work.

II. Background

2.1 How AI Code Generation Works

Today code-generation systems rely on large language models, or LLMs, which are neural models based on transformers and capable of processing vast amounts of publicly available software from software repositories, natural-language text and source code. The model is then fed a "context window" of past code and comments and outputs code turn by turn, using a probability function during inference to model the continuation of the code. Programs are not executed nor formal specifications are checked. Its outputs are predictions as to what code might look like, based on what it has seen during training, for a prompt [6].

2.2 Principal Tools and Adoption Timeline

At the time of writing, there are four main tools dominating the use of products by professionals: GitHub Copilot (the de facto reference implementation [6] and launched in 2021); ChatGPT and its heirs through plug-ins and IDE integration; Amazon Code Whisperer (now part of the Amazon Q suite); and Google's Gemini Code Assist. In late 2024, agents like Claude Code and Cursor debuted, expanding the use of in line completion to multi-step tasks. Adoption took place in three stages. In 2021-2022, the early enthusiasts picked up the tools. After the general release of ChatGPT [7] organizations followed up through 2023. The market hit peak saturation in 2024: Surveys show more than 80% usage in many of the developer populations [1] and [2].

2.3 Prior Reviews and the Gap This Work Addresses

Previous studies evaluating code generation via LLM have primarily centered around abilities evaluation with HumanEval, MBPP, and defect-repair scores. When addressing risk, they generally hampered a single aspect of risk, frequently security risk [6],[8]. In this article we look more broadly. Puts together disadvantage in six aspects: security, code quality, skill formation, IP, productivity, organizational impact. All are assessed with a single PRISMA process. It also incorporates findings from randomized controlled trials published in 2025 and 2026, following the publication of many of the studies included in previous reviews.

III. Methodology

3.1 Protocol, Databases, and Search Strategy

This study is a systematic literature review (SLR) based on Kitchenham and Charters [19]. The reporting is done in line with the PRISMA 2020 statement [20]. A review protocol was developed prior to conducting the search. It formulated the research question, the strategy and inclusion criteria. No modification of the protocol was made during the review. The databases that were searched are IEEE Xplore, ACM Digital Library, Scopus and Google Scholar. The first searches were conducted in March 2026, with subsequent searches conducted in May 2026. The base query is the combination of two blocks with AND. Block 1 focused on the technology: ('AI code generation' OR 'AI coding assistant' OR 'GitHub Copilot' OR 'large language model' OR 'code completion' OR 'automatic code generation'). Block 2 aimed to capture the construct of interest: ('disadvantage' OR 'risk' OR 'limitation' OR 'vulnerability' OR 'security' OR 'technical debt' OR 'skill atrophy' OR 'productivity' OR 'over-reliance'). The query was adapted to accommodate the various syntaxes of the different databases. The actual search phrases are listed in Table 1.

Table 1 Search String Per Database

Database	Search String
IEEE Xplore	("Abstract":LLM OR AI OR "artificial intelligence" OR LLMs OR "large language models") AND ("Abstract":"code generation" OR "code creation" OR "generate code" OR "code writing" OR "code production" OR "code correction" OR "code quality") AND ("Abstract":security OR "cyber security" OR insecure OR secure OR insecurity OR vulnerab* OR threat* OR exploit OR fault OR failure OR disadvantage OR risk OR limitation OR "technical debt" OR "skill atrophy" OR productivity OR "over-reliance")
ACM Digital Library	[[Abstract: "ai"] OR [Abstract: "large language models"] OR [Abstract: "llm"] OR [Abstract: "artificial intelligence"]] AND [[Abstract: "code generation"] OR [Abstract: "code creation"] OR [Abstract: "generate code"] OR [Abstract: "code production"] OR [Abstract: "code correction"] OR [Abstract: "code quality"]] AND [[Abstract: "security"] OR [Abstract: "cyber security"] OR [Abstract: "secure"] OR [Abstract: "insecure"] OR [Abstract: vulnerab*] OR [Abstract: threat*] OR [Abstract: exploit] OR [Abstract: fault*] OR [Abstract: failure] OR [Abstract: disadvantage] OR [Abstract: risk] OR [Abstract: limitation]]

Scopus	`(TITLE-ABS-KEY("LLM") OR TITLE-ABS-KEY("artificial intelligence") OR TITLE-ABS-KEY("large language models") OR TITLE-ABS-KEY("LLMs")) AND (TITLE-ABS-KEY("code generation") OR TITLE-ABS-KEY("code creation") OR TITLE-ABS-KEY("generate code") OR TITLE-ABS-KEY("code writing") OR TITLE-ABS-KEY("code quality") OR TITLE-ABS-KEY("code correction") OR TITLE-ABS-KEY("code production")) AND (TITLE-ABS-KEY("security") OR TITLE-ABS-KEY("cyber security") OR TITLE-ABS-KEY("insecure") OR TITLE-ABS-KEY(vulnerab*) OR TITLE-ABS-KEY(exploit*) OR TITLE-ABS-KEY(risk) OR TITLE-ABS-KEY(disadvantage) OR TITLE-ABS-KEY(limitation))`
Google Scholar	`("AI" OR "LLM" OR "large language model" OR "artificial intelligence") ("code generation" OR "code quality" OR "code creation") (security OR vulnerability OR risk OR disadvantage OR limitation OR "technical debt" OR "skill atrophy" OR productivity OR "over-reliance")`

3.2 Inclusion/Exclusion Criteria and Selection

Studies were considered if they fulfilled all the following criteria. These have been released from 1 January 2020 to 1 May 2026. They were papers from a peer-reviewed conference, a journal or a large-scale industry report with delineated methodology (e.g., DORA, GitClear, Stack Overflow). They included primary or secondary empirical material related to AI code generation in software development. They were composed in the English language. Studies were rejected if they were opinion papers with no empirical content, if they reported on non-software domains or if they were duplicated. The first query to the database yielded 487 records. When duplicate records were removed, 412 unique records were left. After title and abstract screening, 71 abstracts were left to be considered. A total of 43, mostly those that concentrated on capability benchmarking instead of on disadvantages, were excluded following full-text screening. In the final synthesis pool, there were 28 primary studies, as summarized in Fig. 1.

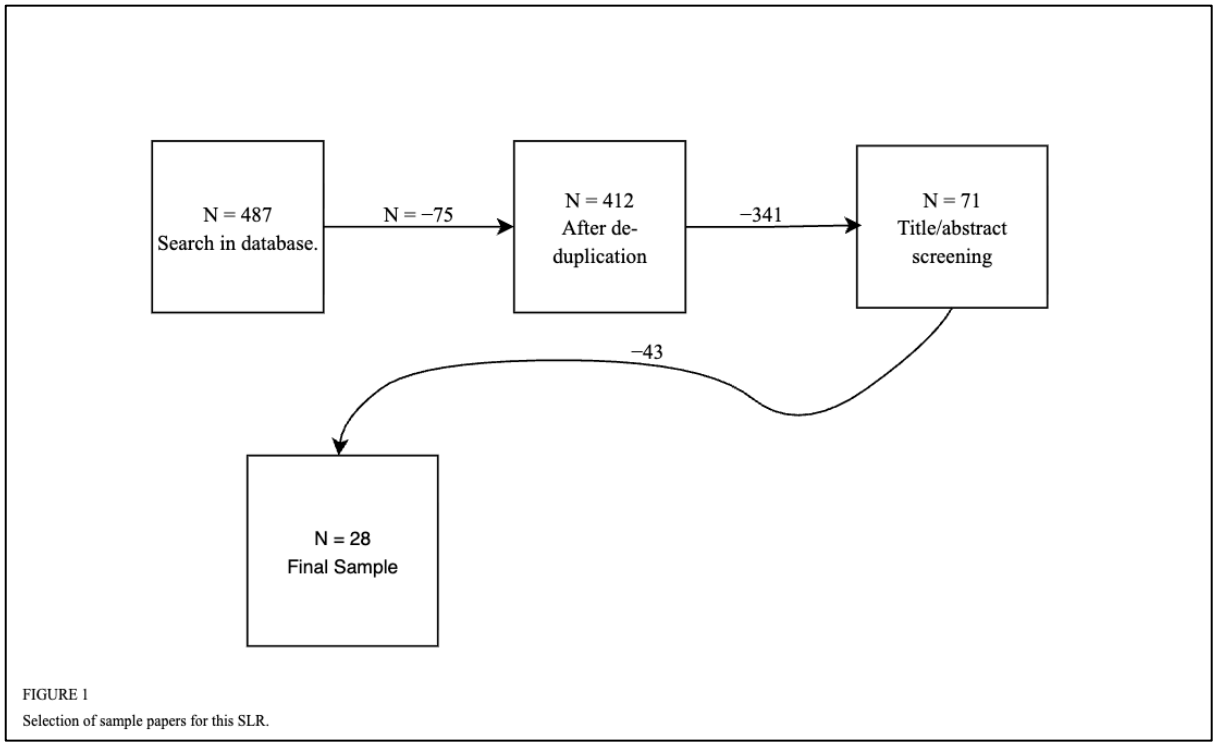

FIGURE 1
Selection of sample papers for this SLR.

Fig. 1 PRISMA-style selection.

3.3 Data Extraction and Synthesis

For each included study, several items were extracted. These include bibliographic metadata, study design, population and sample size, the AI tool studied, the specific disadvantage(s) reported, and the strength of the empirical claim (descriptive, correlational, or causal). Disadvantages were coded in two passes. The first pass used inductive coding to label each disadvantage as it appeared. The second pass grouped those labels into the six

categories presented in Section IV through thematic synthesis. Where studies disagreed, the conflict is flagged in the relevant subsection.

IV. Findings — Documented Disadvantages

4.1 Security Vulnerabilities

Security is the most researched disadvantage. AI code is prone to contain vulnerabilities at a high rate. The work "Asleep at the Keyboard?" A Foundational Study of First Principles presented by Pearce et al. was published at the 43rd IEEE Symposium on Security and Privacy. The authors have created 89 scenarios for the most common entries in the MITRE Common Weakness Enumeration (CWE). They then asked GitHub Copilot to finish each of them. Of the 1,689 produced, approximately 40% of them were vulnerable. All of the weaknesses identified were common. These ranged from broken cryptographic primitives, to improper input validation, to OS command injection to SQL injection. These are the patterns that secure-coding training is supposed to weave out. This was later confirmed and extended by other works. At least 62% of the generated programs in the FormAI dataset were found to be vulnerable under formal verification [23] and Khoury et al. [21] got similar results for the code generated by ChatGPT.

Perry et al. [8] turned the focus from code to the developer. The 2023 controlled CCS study found that the participants writing code with the help of AI generally produced less secure code in most of the tasks. Meanwhile, they were also more likely to think their code was secure. It's a failure of the confidence calibration. A similar trend was observed by Sandoval et al. [9] at USENIX Security 2023 for C-language tasks. Two qualifications apply. First, the percentage of vulnerabilities is dependent on the model version and the prompt wording; the 40% is a current snapshot of mid-2022 Copilot and not a fixed property. Second, participants who explicitly distrust the AI and actively think about the output they create generate code that is at least as secure as the control [8]. This is not to say that the disadvantage is "AI writes insecure code", but rather "AI writes code that seems correct which is more likely to be taken by the developer without security audits."

4.2 Code Quality and Technical Debt

The second strand of evidence relates to the maintainability of AI-assisted code, at the repository level. For both 2024 and 2025, GitClear reports on a longitudinal sample of around 153 million and 211 million changed lines, respectively, from a pre-Copilot baseline in 2020 to the end of 2024 [10], [11]. Three metrics changed in a direction that is associated with decreased maintainability in previous studies of software-engineering metrics.

First, the percentage of code churned in the short-term (within 2 weeks) increased from 3.1% in 2020 to 5.7% in 2024. Second, the percentage of change that was identified as refactoring decreased from 25% in 2021 to less than 10% in 2024. Thirdly, the number of duplicated five-line blocks grew about eight times. These findings are correlational, rather than causal. But the trend is in line with experimental data. With AI assistants, there is a tendency to optimize for local completion, not systemic integration [22]. They leave out any suggestion to reuse an existing function by using the tab-key to insert new code. This is because their context window does not cover the entire repository.

What ensues is a distinct pattern. Copy-paste reuse is taking the place of reuse that has historically characterized mature code bases, which is driven by refactoring.

4.3 Skill Atrophy and Developer Over-Reliance

This is a disadvantage to the developer and not the code. In January 2026, Anthropic ran an RCT with 52 junior Python developers [5]. No one was aware of the Trio concurrency library. In two programming tasks, half of the subjects were provided with up to 15 AI questions to answer. The other half relied solely on official documents. The AI group, in a comprehension quiz after the tasks, performed about two letter grades (17 percentage points) worse than the control group. This negative impact was most pronounced in participants who used AI to directly write code (those with scores lower than 40%) and least pronounced in participants who used AI to explain concepts (those with scores greater than 65%). Proposal of cognitive offloading mechanism. When a problem is solved by an external system, memory traces would be laid down during the problem solving activity, but they are not laid down. Similar effects have been reported for the tasks of navigation, arithmetic, and search. The ramifications are especially dire in software engineering. The skills learned by "everyday" code run most frequently are skills of debugging, system level reasoning, and architectural judgement [5].

This worry is backed up by Stanford payroll numbers. Since the end of 2022, the number of software developers ages 22-25 has decreased about 20%. The number of people aged 26 and over in paid work has been unchanged or increased. Part of this trend is attributed to AI taking over repetitive tasks that were once used to develop skills by junior developers.

4.4 Privacy and Intellectual Property Risks

There are two types of IP risk with AI code generation. The first one is on the input side. When code is uploaded to an AI service running on the cloud, it can be kept, used for training or even made available to other users. This is problematic from the perspective of trade-secret protection and GDPR compliance. The second is on the output side. The underlying model was based on the code licensed in a restrictive manner, the most prominent being the GNU General Public License (GPL). This means that its output might be protected expression that lacks the original credit and/or copyleft requirements [12].

The most prominent lawsuit is Doe v. GitHub, which was filed in November 2022. Microsoft, GitHub, and OpenAI are accused of violating open-source license terms and profiting from open-source code. In summer 2024, a U.S. district court dismissed the direct copyright claims. The court decided that AI-generated content does not necessarily reproduce content from the training set. Claims pursuant to §1202 of the DMCA, however, have persevered and are still pending in litigation [13]. The U.S. Copyright Office has thrown in the added uncertainty. It holds that if there is no human control, no human creativity, no human participation, then it is not copyrightable [14]. This places organizations, who heavily rely on AI generation, unsure about who has the right to use their own product.

A distinct risk is simpler to tackle, but still quite severe. Occasionally, AI assistants can regenerate API keys and credentials from their training data. Production secrets have also been seen being pasted by developers into a prompt window.

4.5 The Productivity Paradox

This is the largest productivity divide, compared to all other topics in this review, of the AI code generation literature. The most meticulously designed experiment to date is a randomized control trial by the METR project [4] in which 16 experienced open source developers each completed specific real issues in their repositories under randomized AI/no-AI conditions, for which 246 task completions were analyzed. Prior to the study, developers estimated that they could expect an AI speedup of about 24%, and after the study, they estimated that they were sped up by about 20%, and measured a 19% slowdown (+2%, +39%). METR has since pointed out [15] that there was significant selection bias in the 2025 replication and that the estimates of the magnitude of the original effect should be interpreted with that bias in mind, but the directionality of the original effect (that developers tend to overestimate the productivity gains of AI) has been confirmed at the organizational level.

According to the 2024 DORA report [3], delivery stability dropped 7.2% and throughput dropped 1.5% with a 25% boost in AI adoption, while self-reported code quality increased by 3.4%. The 2025 DORA report [16] shows that there is a persistent negative connotation associated with delivery stability within the population surveyed in relation to AI. Instead of removing the bottleneck, this is a shift: AI tools facilitate the creation of code, which leads to a higher number, size, and complexity of pull requests – but the number of reviewers does not increase proportionately, and the instability gets compounded by the larger batch sizes. The pattern follows a well-established DevOps principle that high software delivery performance comes from small, well-tested batches [3] [16].

4.6 Workplace and Organizational Impact

A final cluster of disadvantages operates at the organizational level. Three sub-themes keep appearing. The first is shifting role expectations. Junior positions have historically been defined by volume of routine code. These roles are now being redefined — and in some markets, eliminated — faster than educational pipelines can adapt [5]. The second is unsettled accountability. Under the EU AI Act (Regulation (EU) 2024/1689) and consistent U.S. tort doctrine, liability for software defects falls on the deploying organization and its staff — not on the AI tool or its vendor [17]. Vendor terms of service routinely disclaim responsibility for output [18]. The practical result is clear. The developer who accepts an AI suggestion remains legally and professionally responsible for it, even when the AI did most of the thinking. The third is collaboration friction from uneven adoption. Stack Overflow's 2025 survey [2] reports that 46% of developers actively distrust AI output. Only 33% trust it. The most experienced developers are the most skeptical. When teams mix high-trust and low-trust adopters, code-review norms diverge. AI-assisted commits in some organizations get merged several times faster than human-authored commits of similar size. This bypasses the review depth that skeptical team members consider necessary.

V. Discussion

In the 28 studies included, the six categories of disadvantage are not mutually exclusive. They support each other in significant ways. Skill atrophy [5] reduces the ability of the human security operators to identify the vulnerabilities which the AI assistants statistically generate [6, 8]. The productivity paradox [4] is one thing that comes as a downstream effect of the increased churn and decreased refactoring that is reported in the code-quality literature [10,11]. Accountability gaps [17] remain as the productivity narrative (even where measurement belies

it) makes deployment seem economically inevitable. The literature comes together the most around two findings. The first is about security. AI-generated code is much more likely to have vulnerabilities than code that is written without being reviewed. The result of Pearce et al., of about 40% vulnerable generations [6] has been confirmed (more or less) by all subsequent independent measurements reviewed here [8, 9, 23]. The second relates to productivity. The results developers see are always greater than the ones in the controlled experiment. The METR finding [4], the DORA findings [3] [16] and the Anthropic comprehension-test results [5] all indicate the same direction, and, significantly, they can be traced back to independent research programs with different methodologies and incentives.

There are two areas of weak convergence. The GitClear code- quality results are correlational and not causal. As of now, there is no clear precedent on the legalities regarding copyright of AI-generated content [13], [14] and the issue is ongoing litigation. The costs and the disadvantages do not fall equally. At least three factors help to explain why their magnitude varies. The first is the seniority of the developers: juniors incur the most in skill formation and security acceptance costs [5,8] and seniors incur the greatest measured productivity losses [4]. The second is team review maturity: companies that have well-established code review and CI practices seem to be able to take AI output with a pinch of salt [3] [16]. The third is domain: all of the disadvantages mentioned in this review can be exacerbated in safety critical and regulated domains. The evidence base has two main limitations. The first one is temporal. Most controlled studies measure outcomes immediately after the task, but the most concerning phenomena – skill atrophy, technical-debt accumulation and organizational deskilling – occur on the timescale of months to years. The second is generalizability. Many of the controlled evidence pieces are open source contributors or junior developers working on separate pieces of work, away from the more common proprietary-codebase scenario.

There are still a number of key questions that have yet to be answered. In the context of a shrinking pool of junior talent, what will the fallout be for AI assistance in providing senior talent? How do the drawbacks listed here apply to agentic tools such as Claude Code and Cursor agents? But when will it be determined what happens to AI-generated code under copyright/licensing law?

VI. Conclusion

This article has explored the drawbacks to AI-generated code in professional software development. It is based on 28 primary studies from the period 2020–2026. Six prevalent categories of disadvantage are identified: security risks in generated code; degradation of code quality and accumulation of technical debt; skill atrophy and over-reliance; privacy and intellectual-property risks; a productivity paradox in which perceived productivity gains do not translate to measurable productivity gains; and there are workplace and organizational impacts, including unresolved accountability issues. These aren't just drawbacks to a tool that's otherwise benign. The tools speed up production, but slow down verification. They move cognitive load from humane to humane, but leave legal responsibility to the human. They give local benefit, while they cost codebases, junior developers, and organizations that deploy them, distributed costs.

The literature is consistent in two conclusions: First, generated code is more vulnerable than other code; second, there is a significant difference between perceived and actual productivity. In either realm, either would be sufficient to induce significant precautionary process.

We then present three actionable suggestions for organizations using AI coding tools on a large scale. To start, security review of AI-generated code shouldn't be optional. The automated static analysis and human review should be tailored to the higher vulnerability rates reported in [6] and [8]. Second, there is a need for junior-developer training programs to incorporate the practice of core skills like debugging, refactoring, and system design, which are most susceptible to cognitive offloading [5] without AI. Assessment should be in relation to an unaided baseline. Third, metrics of organizational productivity should be based on outcome measures such as delivery stability, lead time, or change failure rate and not self-reported or based on commits, since there is a well-documented disconnect between the two [3] [4] [16].

There are three main research priorities. The first involves multi-year, longitudinal assessments of skill retention, codebase maintainability, and security posture in AI-assisted organizations. The second is the controlled measurement of the new generation of agentic coding tools, like Claude Code and Cursor, that exhibit autonomous multi-step behavior, which is not represented by the current state of the evidence base—inline completion. The third is empirical research on the legal and contractual structures that allocate the costs recorded in this article between AI vendors, deploying organizations and individual developers, and that currently fall primarily on the latter.

References

[1] Stack Overflow, 2024 developer survey: AI (Stack Overflow, 2024). [Online]. Available: https://survey.stackoverflow.co/2024/ai
[2] Stack Overflow, 2025 developer survey: AI (Stack Overflow, Dec. 2025). [Online]. Available: https://survey.stackoverflow.co/2025/ai

[3] DORA, Accelerate state of DevOps report 2024 (Google Cloud / DORA, Oct. 2024). [Online]. Available: https://dora.dev/research/2024/dora-report/

[4] J. Becker, N. Rush, E. Barnes, and D. Rein, Measuring the impact of early-2025 AI on experienced open-source developer productivity, arXiv:2507.09089, Jul. 2025. doi: 10.48550/arXiv.2507.09089.

[5] Anthropic, How AI assistance impacts the formation of coding skills (Anthropic Research, Jan. 29, 2026). [Online]. Available: https://www.anthropic.com/research/AI-assistance-coding-skills

[6] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions, Proc. 2022 IEEE Symposium on Security and Privacy (SP), May 2022, 754–768. doi: 10.1109/SP46214.2022.9833571.

[7] GitHub, Octoverse: the state of open source and rise of AI in 2023 (GitHub, Nov. 2023). [Online]. Available: https://github.blog/news-insights/research/the-state-of-open-source-and-ai/

[8] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, Do users write more insecure code with AI assistants?, Proc. 2023 ACM SIGSAC Conf. CCS, Copenhagen, Nov. 2023, 2785–2799. doi: 10.1145/3576915.3623157.

[9] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, Lost at C: a user study on the security implications of large language model code assistants, Proc. 32nd USENIX Security Symposium, Anaheim, CA, Aug. 2023, 2205–2222.

[10] B. Harding and GitClear, Coding on Copilot: 2023 data suggests downward pressure on code quality (GitClear, Jan. 2024). [Online]. Available: https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality

[11] GitClear, AI Copilot code quality: 2025 data suggests 4x growth in code clones (GitClear, Feb. 2025). [Online]. Available: https://www.gitclear.com/ai_assistant_code_quality_2025_research

[12] W. Ma, S. Wen, Y. Xiang, et al., The 'code' of ethics: a holistic audit of AI code generators, arXiv:2305.12747, May 2023. doi: 10.48550/arXiv.2305.12747.

[13] J. Doe et al. v. GitHub, Inc., Microsoft Corp., and OpenAI LP, Case No. 4:22-cv-06823, N.D. Cal., filed Nov. 3, 2022; partial dismissal of copyright claims Jun. 24, 2024.

[14] U.S. Copyright Office, Copyright and artificial intelligence, part 2: copyrightability, report (U.S. Copyright Office, Jan. 2025).

[15] METR, We are changing our developer productivity experiment design (METR Blog, Feb. 24, 2026). [Online]. Available: https://metr.org/blog/2026-02-24-uplift-update/

[16] DORA, 2025 DORA report: state of AI-assisted software development (Google Cloud / DORA, Sep. 2025). [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report

[17] European Parliament and Council of the European Union, Regulation (EU) 2024/1689 of 13 June 2024 (Artificial Intelligence Act), Official Journal of the EU, L series, 12 Jul. 2024.

[18] M. Agostini et al., Accountable agents in software engineering: an analysis of terms of service and a research roadmap, arXiv:2605.04532, 2026.

[19] B. Kitchenham and S. Charters, Guidelines for performing systematic literature reviews in software engineering, EBSE Tech. Rep. EBSE-2007-01 (Keele University and Durham University, 2007).

[20] M.J. Page et al., The PRISMA 2020 statement: an updated guideline for reporting systematic reviews, BMJ, 372, n71, 2021. doi: 10.1136/bmj.n71.

[21] R. Khoury, A.R. Avila, J. Brunelle, and B.M. Camara, How secure is code generated by ChatGPT?, Proc. 2023 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC), 2023, 2445–2451.

[22] Z. Liu, Y. Tang, X. Luo, Y. Zhou, and L.F. Zhang, No need to lift a finger anymore? Assessing the quality of code generation by ChatGPT, IEEE Trans. Software Engineering, 50(6), 2024, 1548–1584.

[23] N. Tihanyi et al., The FormAI dataset: generative AI in software security through the lens of formal verification, Proc. 20th Int. Conf. on Predictive Models and Data Analytics in Software Engineering (PROMISE), 2024.