Assessing the Impact and Implications of AI-Driven Code Generation and Review: An Empirical and Legal-Scientific Analysis

Dr. Andrei Dragunov[1], **Dr. Anna Tomskova**[2]

[1, 2]Inha University in Tashkent, Uzbekistan

Abstract: ***This paper investigates the impact of Large Language Models (LLMs) on developer productivity and software development processes, focusing on the legal and procedural challenges introduced by AI-generated code. The research problem centers on the tension between demonstrable productivity gains and the risks related to intellectual property, liability, security, and auditability. Employing a mixed-methods approach, this study synthesizes findings from a systematic literature review of recent empirical studies and a doctrinal legal analysis of relevant U.S. copyright and tort law principles. Key findings indicate that while LLM-assisted tools like GitHub Copilot can enhance task completion speed, these benefits are contingent on task complexity and are offset by significant verification overhead and security concerns. Legally, the integration of AI-generated code complicates provenance chains and liability attribution, challenging established doctrines of fair use and the standard of care for software engineers. This study's significance lies in its interdisciplinary synthesis, providing a structured, evidence-based framework for researchers, legal scholars, and practitioners to navigate the adoption of AI coding assistants. It concludes by proposing a tripartite governance framework encompassing enhanced process integration, traceability protocols, and contractual clarity to mitigate risks while harnessing productivity benefits.***

Keywords: AI-generated code, developer productivity, legal liability, GitHub Copilot, software verification

1. Introduction

The integration of Large Language Models (LLMs) into software development, exemplified by tools such as GitHub Copilot and Amazon CodeWhisperer, marks a paradigm shift in how code is authored and reviewed. These tools promise to augment developer productivity by automating routine tasks and offering intelligent suggestions. However, their widespread adoption introduces significant interdisciplinary challenges spanning empirical software engineering, legal doctrine, and professional practice. The significance of this research lies in addressing a critical knowledge gap: while technical efficacy is being measured, a comprehensive assessment of the concomitant legal and procedural ramifications remains underdeveloped.

The primary concern lies in the disconnect between measurable productivity outcomes and the unpredictable nature of LLM outputs, which introduces uncertainty regarding intellectual property infringement, liability for defects, and the erosion of traditional software verification and audit processes. This study is guided by two core research questions:

1) To what extent do empirical studies substantiate claims of enhanced developer productivity through LLM-assisted code generation and review, and what factors moderate these outcomes?
2) What are the principal legal and procedural risks associated with integrating AI-generated code into commercial software, and how can existing legal frameworks and development practices be adapted to mitigate them?

This article is structured as follows: following a literature review synthesizing current research, the methodology outlines our mixed-methods approach. The results section presents findings on productivity impacts and legal analyses. A discussion interprets these results, leading to a conclusion that summarizes contributions and proposes a risk-mitigation framework.

2. Literature Review

1) **Existing Research on Productivity.** Empirical studies on LLM-assisted development are nascent but growing. Foundational work by *Chen et al. (2021)* established the technical capability of models like Codex against programming benchmarks. Subsequent human-centered studies indicate conditional productivity gains. *Peng et al. (2023)*, in a controlled field experiment, reported a statistically significant increase in task completion speed, noting variance based on task complexity. Conversely, *Vaithilingam et al. (2022)* highlighted usability gaps and developer frustration, suggesting that promised efficiency is not always realized in practice. Research on code quality, such as that by *Yetistiren et al. (2022)*, systematically identifies rates of incorrect or suboptimal AI-generated code, introducing a critical "verification cost" that net productivity calculations must account for.
2) **Legal and Security Scholarship.** The legal literature grapples with profound doctrinal questions. The application of copyright law's fair use doctrine to model training is contentious. *Grimmelmann (2023)* argues expansively for fair use, while *Samuelson (2023)* details the risks of reproducing protected expression, particularly through non-literal copying. Liability frameworks are examined by *Scherer (2022)* and *Gupta et al. (2023)*, who analyze how negligence or products liability theories might apply to AI-generated defects. Parallel security research by *Pearce et al. (2022)* demonstrates that LLMs can frequently generate vulnerable code, directly linking technical output to legal risk.
3) **Critical Analysis and Identified Gaps.** A significant gap exists between these disciplinary discourses.

Technical studies often treat legal risk as an externality, while legal analyses may lack grounding in the empirical realities of developer workflow. Furthermore, there is a lack of synthesized, actionable frameworks that translate these identified risks into specific, implementable controls for development teams and corporate policy. This study seeks to contextualize its research within this interdisciplinary gap, aiming to bridge empirical evidence with legal doctrine to inform both practice and policy.

3. Methodology

This study employs a mixed-methods research design, integrating a systematic literature review (qualitative synthesis) with a doctrinal legal analysis to address its interdisciplinary research questions.

3.1 Research Design

The approach is sequential explanatory. Phase 1 involves a systematic review of empirical studies (2021-2024) measuring the impact of LLMs like GitHub Copilot on developer productivity and code quality. Phase 2 consists of a traditional legal doctrinal analysis of U.S. case law, statutory law (17 U.S.C. § 107), and scholarly commentary pertaining to copyright, tort liability, and software development.

3.2 Data Collection and Sources

- **Phase 1 (Empirical):** Academic databases (IEEE Xplore, ACM Digital Library, arXiv) were searched using keywords: "large language model," "developer productivity," "GitHub Copilot," "code generation," "empirical study." Inclusion criteria: peer-reviewed studies or prominent pre-prints featuring quantitative or qualitative human-subject experiments or rigorous case studies. Key studies analyzed include those by Peng et al. (2023), Vaithilingam et al. (2022), and Yetistiren et al. (2022).
- **Phase 2 (Legal):** Legal databases (Westlaw, LexisNexis) and law reviews were searched for analyses of AI, copyright fair use, software liability, and professional duty of care. Foundational legal texts and seminal articles by Grimmelmann (2023), Samuelson (2023), and Lemley & Casey (2021) form the core of this analysis.

Data Analysis Procedures.

- A thematic synthesis of the empirical literature identified recurring patterns, moderating factors, and success metrics.
- For the legal analysis, standard doctrinal methods were applied: identifying relevant legal rules (e.g., fair use factors), analyzing how they apply to the novel context of AI-generated code, and synthesizing scholarly interpretations to assess legal risk and uncertainty.

Ethical Considerations

As a review and analysis of publicly available literature and legal doctrine, this study did not involve human subjects or primary data collection. All sources are cited appropriately to ensure academic integrity.

4. Results

4.1 Findings on Productivity and Code Quality

The synthesis of empirical studies reveals a nuanced picture. A majority of controlled studies confirm an increase in the *speed* of task completion for specific, well-scoped coding tasks (e.g., implementing standard algorithms, writing boilerplate code). However, this benefit diminishes or reverses for more complex, creative, or novel problem-solving tasks. A critical and consistent finding across multiple studies is the introduction of a **verification overhead**. Developers must carefully review, test, and often correct AI-generated suggestions. Research by *Yetistiren et al. (2022)* and *Pearce et al. (2022)* indicates that AI-generated code can contain subtle logical errors, inefficiencies, and security vulnerabilities (e.g., SQL injection flaws), necessitating this rigorous oversight. Therefore, net productivity is highly context-dependent.

4.2 Findings from Legal Doctrinal Analysis

a) **Intellectual Property Risk:** The application of the fair use doctrine to LLM training is legally uncertain. While the *purpose* factor (research/transformation) may favor fair use, the critical *market effect* factor is untested. Courts have not ruled on whether an LLM generating functionally similar code to a copyrighted work constitutes market harm. The doctrine of non-literal copying (*Computer Associates v. Altai*) presents a direct risk, as models may replicate the structure, sequence, and organization of protected code.
b) **Liability Risk:** Traditional software liability hinges on identifying a responsible actor (developer, company). The integration of AI-generated code complicates this chain. Under negligence theory, a court could find that a "reasonable software engineer" must apply a heightened standard of review to AI outputs. In a products liability action, arguments could be made that the AI tool itself is a defective product, implicating the model provider.
c) **Audit and Compliance Risk:** The non-deterministic generation of code undermines the establishment of a clear provenance chain and design rationale, creating evidentiary challenges in regulated industries (e.g., medical, aviation) and during litigation discovery.

5. Discussion

a) **Interpretation of Findings.** The results confirm that LLM-assisted development is not a simple productivity multiplier but a process transformation that substitutes one set of costs (manual coding) for another (verification and legal risk management). The perceived productivity gain is thus a function of how efficiently an organization can manage this new verification overhead and associated risks.

b) **Comparison with Existing Literature.** These findings align with and extend prior work. They corroborate the conditional productivity gains reported by *Peng et al. (2023)* and the quality concerns of *Yetistiren et al. (2022)*. The legal analysis deepens the concerns raised by *Samuelson (2023)* and *Gupta et al. (2023)* by concretely applying specific legal doctrines to the workflow described in the empirical studies.

6. Implications

- **For Theory:** This research argues for an integrated "socio-technical-legal" model of software development tool adoption, where efficacy cannot be assessed without considering legal and procedural externalities.
- **For Practice:** Development teams must institutionalize **mandatory, rigorous review of all significant AI-generated code** as a core component of their definition of "done." This is no longer a best practice but a potential legal necessity.
- **For Policy:** Regulators and industry bodies should work towards clarifying liability allocation and promoting standards for **explainability and traceability** in AI-assisted development tools.

7. Limitations and Future Research

This study is limited by the pace of change in both AI technology and law. New model capabilities and court rulings could rapidly alter the landscape. Future empirical research should conduct longitudinal studies on productivity in real-world projects and investigate the efficacy of different review protocols for AI-generated code. Legal scholarship must continue to analyze evolving case law and propose statutory clarifications.

8. Conclusion

This study contributes an interdisciplinary analysis of AI-driven code generation and review, systematically bridging empirical evidence on productivity with a rigorous legal risk assessment. The key finding is that the adoption of LLM-assisted tools requires a fundamental re-evaluation of software development governance. The purported benefits are real but conditional and are inextricably linked to significant new responsibilities.

The research questions are addressed as follows: 1) Productivity is enhanced primarily for specific, routine tasks, but these gains are moderated by verification costs and security risks. 2) The principal legal risks involve uncertain copyright infringement, complex liability attribution, and compromised auditability.

Therefore, we recommend a structured governance framework comprising: (1) **Enhanced Process Integration** (formalizing AI-output review), (2) **Traceability Protocols** (logging AI use for audit trails), and (3) **Contractual Clarity** (delineating responsibilities among stakeholders). By adopting such a framework, the software industry can strive to harness the transformative potential of LLMs while upholding the legal, ethical, and professional standards required for trustworthy software creation.

References

[1] Becker, B., et al. (2023). Shifting Left with AI: Reimagining the Software Development Lifecycle. *IEEE Software*.
[2] Chen, M., et al. (2021). Evaluating Large Language Models Trained on Code. *arXiv:2107.03374*.
[3] Grimmelmann, J. (2023). The Case for AI-Output Fair Use. *Texas Law Review*.
[4] Gupta, M., et al. (2023). Liability for AI-Generated Software Defects: A Roadmap. *Journal of Law & Technology*.
[5] Lemley, M. A., & Casey, B. (2021). Fair Learning. *Texas Law Review*.
[6] Pearce, H., et al. (2022). Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *IEEE Symposium on Security and Privacy*.
[7] Peng, S., et al. (2023). The Impact of AI on Developer Productivity: A Field Experiment. *Harvard Business School Working Paper*.
[8] Samuelson, P. (2023). Generative AI Meets Copyright Law. *Communications of the ACM*.
[9] Scherer, M. U. (2022). Regulating Artificial Intelligence Systems: Risks, Challenges, Competencies, and Strategies. *Harvard Journal of Law & Technology*.
[10] Vaithilingam, P., et al. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *CHI Conference on Human Factors in Computing Systems*.
[11] Yetistiren, B., et al. (2022). Assessing the Quality of GitHub Copilot's Code Generation. *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*.