International Journal Science and Technology
IJST Vol 5 No. 2 | July 2026 | ISSN: 2828-7223 (print), ISSN: 2828-7045 (online), Page 261-273

A Systematic Analysis of AI-Assisted Vibe Coding in Software Development: Opportunities, Challenges, and Risks

Amrin Fakhruddin Jauhari[1*], Fahmi Fathullah[2] , Indra Adi Permana[3], Robby Nugraha[4]
[1,2]Sistem Informasi, Universitas Nusa Megarkencana, Yogyakarta, Indonesia
[3,4]Teknologi Informasi, Universitas Gunadarma, Depok, Indonesia

Article History
Received : June 28, 2026
Revised : July 18, 2026
Accepted : July 27, 2026
Published : July 29, 2026

Corresponding author*:
amrinjauhari@gmail.com

Cite This Article [*APA Style*] :
jauhari, amrin, Fathullah, F., Permana, I. A., & Nugraha, R. (2026). A Systematic Analysis of AI-Assisted Vibe Coding in Software Development: Opportunities, Challenges, and Risks . *International Journal Science and Technology*, *5*(2), 261–273.

DOI:
https://doi.org/10.56127/ijst.v5i2.3000

Abstract: The literature on vibe coding has grown rapidly; however, it remains fragmented and is largely dominated by industry reports, leaving its position relative to traditional manual programming and low-code development insufficiently examined. This gap makes it difficult for both researchers and practitioners to determine when vibe coding is appropriate and what risks should be anticipated. **Purpose:** This study aims to systematically map the current landscape of vibe coding, develop a comparative framework against manual and low-code software development approaches, and propose practical risk mitigation recommendations for software development practitioners. **Methodology**: A Systematic Literature Review (SLR) was conducted following the PRISMA protocol. Relevant publications from 2023 to 2026 were retrieved from IEEE Xplore, ACM Digital Library, Springer, ScienceDirect, and arXiv, resulting in 61 studies that were analyzed using thematic analysis. **Findings**: The results indicate that vibe coding can accelerate software prototyping by approximately 40–60% compared with manual development. However, it introduces a verification bottleneck by shifting developers' workload from code implementation to quality assurance and validation. Compared with low-code development, vibe coding provides greater flexibility in expressing user intent but exhibits lower output predictability. In comparison with manual development, it offers significant gains in development speed while sacrificing architectural control and code security, thereby increasing the risks of technical skill degradation, hidden security vulnerabilities, and accumulated technical debt. **Implications**: The findings provide practical guidance for software development teams in identifying project phases that are suitable for extensive adoption of vibe coding and those that still require manual architectural review. The study also emphasizes the importance of integrating security auditing and technical debt monitoring into AI-assisted software development workflows. **Originality/Value:** The novelty of this study lies in its explicit comparative framework, which systematically positions vibe coding alongside manual and low-code development across six technical dimensions, extending previous studies that have generally examined vibe coding in isolation.

Keywords: Vibe Coding; AI-Assisted Software Development; Systematic Literature Review; Low-Code Development; Software Engineering; Technical Debt

INTRODUCTION

Digital transformation in the global software industry has accelerated the large-scale adoption of generative artificial intelligence across the entire software development lifecycle, from requirements specification to code maintenance. Substantial investments in

large language model (LLM)-based development tools such as GitHub Copilot, Cursor AI, and Claude Code reflect a fundamental shift from repetitive manual coding practices toward more intuitive natural language-driven interactions (Nettur & et al., 2025; Sergeyuk et al., 2024). Empirical evidence indicates that integrating generative AI assistants can reduce the time required to develop a minimum viable product (MVP) by approximately 40–60% compared with conventional development approaches, thereby enhancing the competitiveness of software organizations facing increasing market pressure for shorter release cycles (Mohamed et al., 2026). This transformation is no longer confined to large technology companies but has also extended to independent developers and small organizations that previously faced limitations in software engineering resources.

Amid this rapid acceleration, the concept of vibe coding has emerged as a new paradigm in which developers interact with autonomous AI agents through natural language conversations to generate software artifacts without explicitly writing deterministic programming instructions (Sapkota et al., 2025). Unlike traditional manual development, which requires precise syntactic coding, and unlike low-code platforms that constrain logical expression through predefined visual components, vibe coding positions functional intent specification as the primary mode of interaction. While this approach significantly improves development convenience, real-world incidents have revealed critical challenges, including undetected security vulnerabilities in AI-generated code, the accumulation of technical debt from the early stages of development, and the gradual decline of developers' ability to independently debug complex systems (Sartori, 2025; Zhao et al., 2025). Consequently, investigating this phenomenon has become increasingly urgent, as its adoption has outpaced empirical understanding of its technical mechanisms and long-term implications for software quality and security.

The first stream of previous research has primarily focused on qualitative investigations and in-depth interviews exploring developers' experiences when collaborating with generative AI agents. Pimenova found that trust, communication, and workflow dynamics are key determinants of successful human–AI co-creation (Pimenova et al., 2025), whereas Huang reported that professional developers generally retain full control over architectural decisions rather than relying passively on AI-generated outputs (Huang et al., 2025). Collectively, these studies consistently emphasize that the success of vibe coding depends more on developers' ability to critically validate AI-generated results than on the sophistication of the underlying language models.

The second stream consists of systematic reviews and mapping studies that seek to establish vibe coding as a formal research discipline. Meske conceptualized vibe coding as a reconfiguration of intent mediation within software engineering (Meske et al., 2025), while Beaulieu conducted a multivocal mapping study that consolidated the diverse definitions and taxonomies proposed by researchers and practitioners (Beaulieu et al., 2025). Elgendy further advanced this line of work by proposing an architectural framework for responsible vibe coding practices; however, empirical validation of this framework in industrial-scale environments remains limited (Elgendy & et al., 2026).

The third stream encompasses quantitative and data-driven studies investigating the technical risks associated with vibe coding, particularly software security vulnerabilities and code quality degradation. Zhao performed a systematic benchmarking study and demonstrated that AI-generated code in real-world tasks is susceptible to security flaws that often remain undetected by both developers and AI agents. Likewise (Zhao et al., 2025), Aiersilan examined the phenomenon of cognitive offloading and found that excessive reliance on AI assistance may gradually reduce developers' independent problem-solving capabilities (Aiersilan, 2026).

Although these three streams provide valuable insights into the emerging field of vibe coding, the existing literature remains fragmented and has yet to position vibe coding within an explicit comparative framework alongside conventional manual development and low-code approaches. Consequently, a significant research gap persists regarding how these three development paradigms systematically differ in terms of development speed, architectural control, required expertise, and security-related risks.

To address this gap, the present study aims to conduct a systematic literature review to map the current landscape of vibe coding, develop a comparative framework positioning vibe coding alongside manual and low-code software development across multiple technical dimensions, analyze the implications of AI-generated code for software quality and security, and formulate practical risk mitigation recommendations applicable to both software practitioners and development organizations.

The practical value of this comparative framework is particularly relevant for small and medium-sized organizations that often lack dedicated research units or specialized cybersecurity teams to assess the risks associated with adopting vibe coding. By explicitly positioning vibe coding alongside manual and low-code development using a common set of evaluation dimensions, this study provides a practical reference that enables technical

decision-makers to evaluate trade-offs among development speed, architectural control, and software security according to their specific project contexts, without requiring them to synthesize numerous publications that employ inconsistent terminology and conceptualizations.

This study is guided by the initial proposition that vibe coding offers significant advantages in development speed compared with conventional manual programming and provides accessibility comparable to low-code platforms for less experienced developers. However, it is hypothesized that vibe coding systematically lags behind manual development in terms of architectural control and software security unless supported by rigorous deterministic validation mechanisms. This proposition is examined through a thematic analysis of selected literature and synthesized into a comprehensive comparative framework encompassing the three major software development approaches.

RESEARCH METHOD

The unit of analysis in this study comprises peer-reviewed journal articles, technical reports, and preprints that examine the application of generative artificial intelligence agents throughout the software engineering lifecycle, with a particular focus on vibe coding practices and their comparison with conventional manual and low-code development approaches. The analyzed materials include conceptual definitions, productivity experiment results, software security vulnerability findings, and risk mitigation frameworks proposed across the selected publications.

This study adopts a Systematic Literature Review (SLR) design following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines to ensure objectivity, transparency, and reproducibility throughout the literature synthesis process. This methodological approach was selected because vibe coding is an emerging multidisciplinary research topic distributed across multiple scientific domains, requiring a rigorous screening procedure to ensure that only studies directly relevant to the research objectives are included in the final analysis.

The research data were collected from reputable scientific databases, including IEEE Xplore, ACM Digital Library, SpringerLink, and ScienceDirect, covering publications from 2023 to 2026 to capture the most recent developments in Large Language Models (LLMs) and their integration into AI-powered software engineering agents. In addition to these primary databases, preprint repositories such as arXiv were included because research

in this rapidly evolving field is frequently disseminated through preprints prior to formal peer review.

The data collection process involved systematic keyword searches using terms such as "AI-assisted programming," "agent-generated code," "vibe coding," and "low-code development" to identify studies addressing software security challenges and architectural risks associated with AI agent-based software development. The screening process was conducted in multiple stages, beginning with title and abstract screening, followed by full-text assessment of eligible studies, as illustrated in Figure 1. The inclusion criteria prioritized empirical studies, systematic reviews, and technical reports that explicitly investigated autonomous AI agents within software development processes. In contrast, studies focusing solely on AI-assisted text editing without autonomous agent involvement were excluded to preserve thematic relevance. An additional quality assessment was conducted during the eligibility stage by evaluating methodological rigor, data source transparency, and consistency of reported findings. Consequently, non-peer-reviewed publications lacking sufficient methodological documentation were excluded despite their thematic relevance. Following this comprehensive selection procedure, 61 studies satisfied the predefined quality criteria and were included in the final analysis.

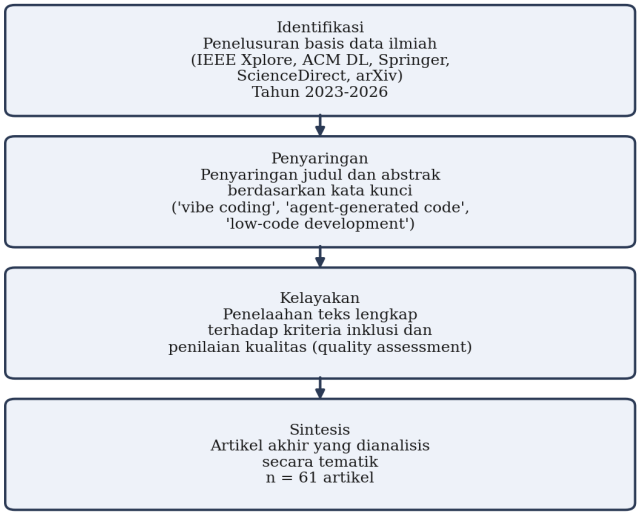

Figure 1. Literature Screening Flow Diagram Based on the PRISMA Protocol

The data extracted from the 61 selected studies were analyzed using thematic analysis to identify recurring patterns related to development effectiveness, human–AI collaboration challenges, and the technical risks associated with AI agent-based software development. The selected studies were subsequently classified into four methodological categories according to their

primary research focus. Cross-category synthesis was then performed to develop a comprehensive comparative framework positioning vibe coding, conventional manual development, and low-code development across multiple technical dimensions, including development speed, architectural control, required expertise, software security risk, flexibility in expressing functional intent, and technical debt accumulation.

RESULT

Landscape and Distribution of Vibe Coding Publications

Among the 61 studies included in the analysis, the distribution of research approaches is presented in Table 1**.** The studies were classified according to their primary methodological focus to provide an overview of the current research landscape surrounding vibe coding.

Table 1. Distribution of Vibe Coding Research Categories (2023–2026)

Research Category	Number of Studies	Primary Focus	Representative Findings
Qualitative Studies / Interviews	18	Developer experiences and trust	Developers retain architectural control despite AI assistance (Huang et al., 2025)
Systematic Reviews / Mapping Studies	15	Taxonomy and formal conceptualization	Vibe coding as a reconfiguration of *intent mediation* (Meske et al., 2025)
Data-Driven Quantitative Studies	16	Software security vulnerabilities and code quality	MVP development accelerated by 40–60%, accompanied by security vulnerabilities (Zhao et al., 2025)
Framework / Prescriptive Studies	12	Risk mitigation and governance	The VIBE4M framework for sustainable code maintenance (Maes, 2025)

As shown in Table 1, data-driven quantitative studies and qualitative interview-based investigations together account for more than half of the analyzed literature (34 out of 61 studies), while systematic reviews and prescriptive frameworks contribute to the formalization of vibe coding as an emerging research discipline. Most existing studies primarily focus on the effectiveness of autonomous AI agents in automating routine programming tasks rather than supporting higher-level architectural reasoning. IEEE Xplore and the ACM Digital Library constitute the dominant sources of theoretical and empirical evidence concerning the technical limitations of vibe coding. Meanwhile, the relatively small proportion of prescriptive studies (12 out of 61) indicates that practical risk

mitigation strategies remain at an early stage of development compared with descriptive and exploratory research. Overall, these findings suggest that vibe coding is still in a predominantly exploratory phase, where understanding the phenomenon and identifying associated risks have progressed more rapidly than empirically validated governance and mitigation solutions suitable for industrial-scale adoption.

Comparison of Vibe Coding, Low-Code, and Manual Development

Cross-study synthesis produced a comparative framework describing the operational characteristics of three software development paradigms across six technical dimensions, as presented in Table 2.

Table 2. Comparison of Manual Development, Low-Code, and Vibe Coding

Dimension	Manual Development	Low-Code	Vibe Coding
Development speed	Low to moderate	High	High (40–60% faster during MVP development)
Architectural control	High	Moderate, constrained by platform architecture	Low to moderate, depending on intent specification
Technical expertise required	High	Moderate	Moderate, with emphasis shifting toward verification skills
Security risk	Dependent on manual code review	Dependent on vendor platform security	High without systematic security auditing
Flexibility of intent expression	Limited by programming syntax	Limited by visual components	High through natural language interaction
Technical debt accumulation	Manageable through regular refactoring	Constrained by platform architecture	High risk without deterministic validation

Table 2 demonstrates that vibe coding outperforms both manual and low-code development in terms of development speed and flexibility in expressing functional intent. However, it performs less favorably in architectural control and software security compared with conventional manual development. Low-code platforms occupy an intermediate position, where limitations are primarily imposed by vendor-specific platform architectures rather than developers' technical capabilities. Manual software development continues to provide the highest level of architectural control, albeit at the expense of longer development time and greater technical expertise. The most significant security concerns arise within vibe coding, particularly when AI-generated code is deployed without rigorous verification and security auditing procedures. These findings highlight a fundamental trade-off among development speed, architectural control, and software security, indicating

that the selection of an appropriate development approach should be aligned with project characteristics, system criticality, and an organization's tolerance for software security risks.

Furthermore, the relative advantages of these three approaches are not uniform across all software domains. For web applications and user interface development, where design patterns and implementation practices are relatively standardized, vibe coding tends to produce more reliable outcomes because large language models have been extensively trained on similar code examples. In contrast, specialized domains such as embedded systems, high-performance data processing, and organization-specific business logic exhibit a substantially larger gap between developers' intended functionality and AI-generated implementations. Consequently, these application domains require considerably greater levels of manual verification and expert review than conventional web application development.

Technical and Security Risks in Vibe Coding-Based Development

Thematic analysis of the quantitative and data-driven studies identified three major categories of technical risk associated with vibe coding: hidden software security vulnerabilities, technical debt accumulation, and developer skill atrophy. Zhao reported that AI-generated code used in real-world development tasks frequently contains significant security vulnerabilities that often remain undetected by both developers and AI agents, largely due to the absence of systematic code review practices within typical vibe coding workflows. Similarly, van Hurne emphasized the necessity of incorporating technical debt management frameworks beginning at the prompt engineering stage, addressing a phenomenon described by Sartori (2025) as "Architectures of Error," in which developers become overly dependent on large language model-generated code without sufficient critical evaluation (van Hurne, 2025). Aiersilan further demonstrated that developers' independent debugging capabilities gradually deteriorate as reliance on autonomous AI agents increases through the process of cognitive offloading (Aiersilan, 2026)

As an initial response to these challenges, Maes proposed the VIBE4M framework, a collection of governance policies and verification practices designed to preserve code readability, maintainability, and long-term software quality despite the accelerated development pace enabled by vibe coding (Maes, 2025). Collectively, these three

categories of risk demonstrate that the productivity benefits offered by vibe coding cannot be separated from the need for deterministic validation mechanisms, rigorous code review, and continuous security auditing, particularly in the development of software systems where security, reliability, and long-term maintainability are mission-critical requirements.

DISCUSSION

Vibe coding shifts the role of software developers from writing repetitive programming syntax to reasoning about system specifications and overseeing software quality. Although functional prototypes can be produced considerably faster, these productivity gains come at the cost of increased verification effort, greater software security risks, and a higher likelihood of technical debt accumulation compared with both conventional manual development and low-code approaches.

From an explanatory perspective, this shift can be understood through the mechanism of probabilistic intent mediation that underpins vibe coding, whereby AI agents translate qualitative natural language descriptions into executable source code without providing formal guarantees of architectural correctness. The absence of deterministic specifications during the early stages of development increases the possibility of discrepancies between developers' intended functionality and the resulting implementation. Such inconsistencies often remain undetected until software testing or even post-deployment, resulting in repeated correction cycles commonly referred to as the verification bottleneck. This mechanism also reflects a redistribution of cognitive workload throughout the software development lifecycle. In conventional manual development, cognitive effort is relatively balanced between logical design and syntactic implementation. In contrast, vibe coding concentrates cognitive demands at two critical stages: prompt formulation at the beginning of development and output validation at the end, while the transformation from intent to executable code occurs within the opaque internal processes of a large language model. This concentration of effort explains why productivity improvements are particularly evident during the early stages of development but gradually diminish as project complexity increases and the volume of AI-generated code requiring verification expands.

From a comparative standpoint, these findings are consistent with (Sapkota et al., 2025)who distinguished vibe coding from agentic coding based on the degree of human involvement in validation, and they reinforce the argument of Huang that professional software developers continue to retain explicit authority over architectural decision-making

(Huang et al., 2025). Unlike previous studies that have generally examined vibe coding in isolation, the present study contributes a novel comparative framework positioning vibe coding alongside conventional manual development and low-code platforms. This comparison clarifies the relative position of vibe coding within the broader spectrum of software development methodologies, including the democratization of software engineering through no-code and low-code approaches (Gadde, 2025). A fundamental distinction also emerges when comparing these findings with earlier research on low-code development, which primarily identified platform constraints as the major limitation to developer productivity. Although vibe coding is not restricted by predefined visual components as low-code platforms are, it introduces a new form of epistemic limitation, namely uncertainty regarding the extent to which large language models genuinely understand domain-specific constraints and the non-functional requirements of the systems they generate.

From a software engineering perspective, these findings suggest that vibe coding is most appropriately applied during idea exploration and rapid prototyping, where software components are not highly security-critical. Conversely, mission-critical modules—including authentication systems, sensitive data management, and transactional business logic—require integration with manual verification procedures and deterministic intent-to-structure frameworks to preserve architectural integrity. For example, a development team creating a user registration module may effectively employ vibe coding to rapidly generate user interface components and navigation workflows, as implementation errors within these components are generally straightforward to identify and pose relatively limited security risks. However, when development progresses to password storage mechanisms, session management, or authorization logic, the findings presented in Table 2 and the technical risk analysis indicate that relying exclusively on AI agents substantially increases the likelihood of introducing security vulnerabilities that may remain undiscovered until the system enters production.

A critical reflection also reveals both the functional benefits and potential dysfunctions associated with this division of responsibilities. On the positive side, distinguishing among the three development paradigms enables software teams to allocate verification resources more efficiently toward components that genuinely present high levels of technical or security risk. However, the distinction between "low-risk" and "critical" components is not always clearly defined, particularly in small-scale projects with limited personnel and

quality assurance resources. Consequently, careful judgment remains essential when determining which parts of a software system can safely rely on vibe coding.

Based on these findings, several practical recommendations can be proposed. Software development organizations should establish governance policies that clearly distinguish development phases suitable for extensive use of vibe coding from those requiring mandatory manual architectural review. These governance practices should be complemented by automated security auditing and continuous technical debt monitoring integrated into Continuous Integration and Continuous Deployment (CI/CD) pipelines. Furthermore, organizations should define verification competency standards for developers, emphasizing the ability to critically inspect, interpret, and validate AI-generated code while identifying common software security vulnerabilities. Such measures are intended to prevent the emergence of the "hollow senior" phenomenon, in which experienced developers gradually lose advanced debugging and architectural reasoning capabilities through excessive dependence on AI automation, while simultaneously ensuring that junior developers acquire a sufficiently strong foundation in software architecture and engineering principles.

Nevertheless, the systematic literature review approach adopted in this study has several methodological limitations. A substantial proportion of the included studies originated from preprint repositories that had not yet undergone formal peer review, reflecting the exceptionally rapid publication cycle characteristic of the vibe coding research domain. Although this publication pattern is common across rapidly evolving artificial intelligence research, it nevertheless requires caution when generalizing the reported findings. In addition, the frequently cited 40–60% productivity improvement reported in the literature was derived from studies involving diverse project types, organizational contexts, and development teams. Accordingly, this figure should be interpreted as an indicative performance range rather than a universally applicable benchmark for all software development projects.

CONCLUSION

Vibe coding significantly accelerates software prototyping; however, these productivity gains come at the expense of reduced architectural control unless supported by additional validation mechanisms. Compared with conventional manual development and low-code approaches, vibe coding differs systematically in terms of required technical

expertise and software security risks. These findings address the research objectives established at the outset of this study and confirm its initial proposition. The primary scholarly contribution of this research lies in the development of an explicit comparative framework that positions vibe coding alongside manual and low-code software development across six technical dimensions. Such an integrated comparison has received limited attention in previous literature and therefore contributes to a more comprehensive understanding of the role of vibe coding within the broader spectrum of software engineering methodologies. Furthermore, the proposed framework provides a foundation for developing risk-based governance policies that can guide software practitioners and development organizations in adopting AI-assisted programming responsibly.

This study is subject to several limitations. The analysis was restricted by the scope of the selected scientific databases and the relatively short publication period (2023–2026), reflecting the emerging nature of vibe coding research. Consequently, the findings should be further validated through empirical investigations involving real-world software development projects. Future research is recommended to develop quantitative measurement instruments for assessing technical debt accumulation and software security vulnerabilities in AI-generated code, as well as to evaluate the effectiveness of deterministic intent-to-structure frameworks in large-scale industrial software engineering environments.

REFERENCES

Aiersilan, A. (2026). The vibe-check protocol: Quantifying cognitive offloading in AI programming. *Cornell University*. https://doi.org/10.48550/arxiv.2601.02410

Beaulieu, N., Dascalu, S. M., & Hand, E. (2025). Vibe coding: A multivocal systematic mapping study. 266-273. https://doi.org/10.1109/snpd65828.2025.11254176

Elgendy, I. A., & et al. (2026). Responsible vibe coding: Architecture, opportunities, and research agenda. *Journal of Computer Information Systems*, 1-19. https://doi.org/10.1080/08874417.2026.2621186

Gadde, A. (2025). Democratizing software engineering through generative AI and vibe coding: The evolution of no-code development. *Journal of Computer Science and Technology Studies*, *7*(4), 556-572. https://doi.org/10.32996/jcsts.2025.7.4.66

Huang, R., Reyna, A. M., Lerner, S., Xia, H., & Hempel, B. (2025). Professional software developers don't vibe, they control: AI agent use for coding in 2025. https://doi.org/10.48550/arxiv.2512.14012

Maes, S. H. (2025). The gotchas of AI coding and vibe coding. It's all about support and maintenance. https://doi.org/10.5281/zenodo.15343349

Meske, C., Hermanns, T., Weiden, E. V. D., Loser, K., & Berger, T. (2025). Vibe coding as a reconfiguration of intent mediation in software development: Definition, implications, and research agenda. *IEEE Access*, *13*, 213242-213259. https://doi.org/10.1109/ACCESS.2025.3645466

Mohamed, A., Assi, M., & Guizani, M. (2026). The impact of LLM-assistants on software developer productivity: A systematic review and mapping study. *Cornell University*. https://doi.org/10.1145/3809494

Nettur, S. B., & et al. (2025). The role of GitHub Copilot on software development: A perspective on productivity, security, best practices and future directions. https://doi.org/10.48550/arxiv.2502.13199

Pimenova, V., Fakhoury, S., Bird, C., Storey, M., & Endres, M. (2025). Good vibrations? A qualitative study of co-creation, communication, flow, and trust in vibe coding. https://doi.org/10.48550/arxiv.2509.12491

Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic AI. https://doi.org/10.48550/arxiv.2505.19443

Sartori, C. C. (2025). Architectures of error: A philosophical inquiry into AI and human code generation. https://doi.org/10.48550/arxiv.2505.19353

Sergeyuk, A., Golubev, Y., Bryksin, T., & Ahmed, I. (2024). Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, *178*, 107610. https://doi.org/10.1016/j.infsof.2024.107610

van Hurne, M. H. K. (2025). A technical debt-aware prompting framework for sustainable vibe coding: Addressing the production readiness crisis in AI-assisted software development. https://doi.org/10.36227/techrxiv.175459417.76916566/v1

Zhao, S., Wang, D., Zhang, K., Luo, J., Li, Z., & Li, L. (2025). Is vibe coding safe? Benchmarking vulnerability of agent-generated code in real-world tasks. https://doi.org/10.48550/arxiv.2512.03262