IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, October 2026
Impact Factor: 8.2

The Impact of Generative AI on Modern Software Development: Opportunities and Challenges

Ashwini D. Pawar[1], Harshada T. Ghute[2], Vaishnavi R. Garudkar[3], Prof. Ulhare P. S[4], Prof. Ranmalkar V. S[5]
[1,2,3,4,5] Department of MCA (Under Engineering)
Adsul Technical Campus Chas, Ahilyanagar, India
ashwinidpawar75@gmail.com[1], ghuteharshada267@gmail.com[2],
vaishugarudkar2005@gmail.com[3]

Abstract: *Generative Artificial Intelligence (GenAI) is increasingly influencing modern software development by providing intelligent assistance across multiple activities of the software development lifecycle. Large Language Models (LLMs) can support developers in tasks such as source-code generation, debugging, software testing, documentation, code refactoring, requirements analysis, and software maintenance. This paper presents a structured review of recent research examining the opportunities and challenges associated with the adoption of GenAI in modern software engineering. The review considers evidence related to AI-assisted programming, developer productivity, code quality, testing, security, documentation, architecture support, and human–AI collaboration. A systematic literature review approach is used to organize and analyze relevant studies and identify major trends in the field. The reviewed literature indicates that GenAI can reduce repetitive development effort, accelerate selected software engineering tasks, and assist developers in producing and understanding software artifacts. However, its effectiveness varies across models, programming languages, development tasks, and usage contexts. The literature also identifies concerns related to inaccurate or unreliable outputs, security vulnerabilities, privacy, intellectual property, explainability, trust, and excessive dependence on AI-generated solutions. Therefore, human judgment, code review, testing, security assessment, and responsible governance remain essential for reliable adoption. The paper further identifies research gaps involving standardized evaluation, secure code generation, domain-specific AI assistants, trustworthy human–AI collaboration, and long-term developer outcomes. Overall, this review provides a balanced analysis of the opportunities and limitations of GenAI in modern software development*

Keywords: Generative artificial intelligence, large language models, software development, code generation, developer productivity, software security, systematic literature review

I. INTRODUCTION

Software development is being reshaped by the growing use of Generative Artificial Intelligence (GenAI) in development environments and engineering workflows. GenAI systems, particularly Large Language Models (LLMs), can process natural-language instructions and produce software-related outputs such as source code, explanations, test cases, and documentation. Research on AI-assisted programming has therefore expanded from simple code completion toward broader forms of assistance within software engineering [1], [4], [15].

The use of GenAI is not restricted to programming. Developers can use these systems for understanding existing code, identifying possible defects, preparing test cases, generating documentation, refactoring implementations, and supporting maintenance [1], [3], [7], [12]. Research has also examined software architecture and Agile development, showing that AI assistance can occur at multiple stages of the Software Development Life Cycle (SDLC) [6], [9], [10].

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/568

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 6, October 2026 Impact Factor: 8.2

Modern foundation models such as GPT-4, Gemini, and Code Llama have increased interest in AI-assisted software engineering [16]–[18]. Studies show that generated-code quality depends on factors including the selected model, programming language, development task, and evaluation method [2], [4], [11], [14]. GenAI output should therefore be treated as assistance requiring interpretation and validation rather than as a direct replacement for established engineering practices.

GenAI adoption also introduces technical and responsible-use concerns. AI-generated programs may contain functional mistakes, inappropriate implementations, or security weaknesses [3], [5], [20]. Research further discusses responsible AI use, developer trust, privacy, intellectual property, and risks associated with excessive reliance on automated recommendations [8], [13]. Productivity studies have examined changes in development time, task completion, and developer experience [7], [19], but the practical value of GenAI also depends on the effort needed to inspect, test, modify, and validate generated outputs.

This paper reviews research on code generation, productivity, testing, documentation, architecture, Agile development, security, and human–AI collaboration. It synthesizes recurring opportunities and limitations and identifies research gaps concerning evaluation, secure generation, trustworthy assistance, domain-specific applications, and long-term developer outcomes.

II. BACKGROUND

Generative Artificial Intelligence refers to AI systems designed to produce new content from patterns learned during model training. In software engineering, generated content may include source code, technical explanations, documentation, test artifacts, and other development outputs. A major development behind current GenAI use is the emergence of transformer-based architectures and large foundation models. LLM capabilities can be applied to both natural and programming languages, enabling systems such as GPT-4, Gemini, and Code Llama to support software-related tasks [16]–[18].

LLM-based systems can be integrated into development environments to provide code completion, modification, testing, documentation, and maintenance assistance [1], [3], [12]. Their role is best understood as an additional assistance layer: developers can use generated outputs as starting points, suggestions, or explanations and then apply contextual knowledge to evaluate and refine them. This is important because reliability, security, productivity, and trust concerns remain part of AI-assisted development [7], [8], [13], [19], [20].

III. LITERATURE REVIEW

Research on GenAI in software engineering covers programming, software quality, productivity, testing, security, documentation, and human factors. Nguyen and Nadi evaluated GitHub Copilot code suggestions across programming languages and considered correctness and understandability, showing the usefulness of generated code together with the need for evaluation [1]. Clark et al. examined the quality and consistency of AI-generated Python programs, while Shehab et al. assessed code-generation accuracy, quality, and performance [2], [4].

Testing has become a significant application area. Fakhoury et al. investigated interactive LLM-based code generation in a test-driven development setting [3]. Their work illustrates how GenAI can participate in workflows that combine implementation and testing. Generated tests can reduce manual effort, but completeness and correctness still require human checking.

Security is a major concern. Criollo-Ramírez et al. systematically reviewed the security of LLM-generated code and emphasized the need to evaluate generated software rather than assume that syntactically valid code is secure [5]. Negri-Ribalta et al. similarly reviewed the security impact of AI models on code generation and identified vulnerability-related concerns across prior studies [20].

The literature increasingly considers GenAI across broader engineering activities. Şimşek et al. discuss changing software-development roles and responsibilities [6]. Research on Agile development examines GenAI across coding,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, October 2026
Impact Factor: 8.2

testing, documentation, and coordination [9], while architecture-oriented work explores AI assistance in software architecture [10]. Gadamsetty et al. study GenAI effectiveness and efficiency for software documentation [12].

Human factors are also important. Sabouri et al. examine trust dynamics in AI-assisted development [13]. Butler et al. studied generative AI coding tools in the workplace using surveys, a randomized controlled trial, and a diary study; their reported results distinguish perceived usefulness and enjoyment from trust in AI-generated code [7]. Rodriguez-Cardenas argues that evaluation of LLM-generated code should go beyond conventional accuracy and robustness metrics [14]. Pakiş and Kolukısa-Tarhan compare code-generation quality across LLM families, while Umama et al. provide a systematic review covering technical and demographic aspects of LLM-based code generation [11], [15].

Across these studies, three themes recur: GenAI can assist several software engineering activities; output quality varies with model, task, language, and evaluation conditions; and responsible adoption requires attention to security, trust, privacy, human review, and evaluation [1], [2], [4], [5], [8], [11], [13], [14], [20]

IV. COMPARATIVE SUMMARY OF LITERATURE

Ref	Year	Research focus	Major Key Finding	Limitations
[1]	2022	GitHub Copilot code suggestions	Evaluates AI-generated code suggestions	Human validation required
[2]	2024	Quality and consistency of AI-generated code	Examines quality and consistency	Quality varies across generations
[3]	2024	LLM-based test-driven code generation	Studies interactive LLM-assisted development	Generated output requires validation
[4]	2024	LLM code-generation evaluation	Evaluates accuracy, quality, performance	Results depend on task and model
[5]	2024	Security of LLM-generated code	Reviews vulnerabilities in LLM-generated code	Mitigation needs further research
[6]	2025	Future software-development roles	Examines changing software personas	Long-term role changes remain open
[7]	2025	GenAI coding tools in workplace	Studies developer experience and use	Trust requires continued evaluation
[8]	2025	Responsible GenAI across SDLC	Discusses responsible SDLC adoption	Governance needs further development
[9]	2025	GenAI in Agile development	Surveys GenAI use in Agile activities	Human–AI integration needs study
[10]	2025	AI-assisted software architecture	Examines AI assistance in architecture	Architectural decisions need validation
[11]	2025	LLM family comparison for code generation	Compares different LLM families	Performance varies by model/task
[12]	2025	GenAI for software documentation	Studies GenAI documentation support	Generated documentation needs review
[13]	2025	Trust in AI-assisted development	Examines trust in AI-assisted development	Trust is context-dependent

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 6, October 2026 Impact Factor: 8.2

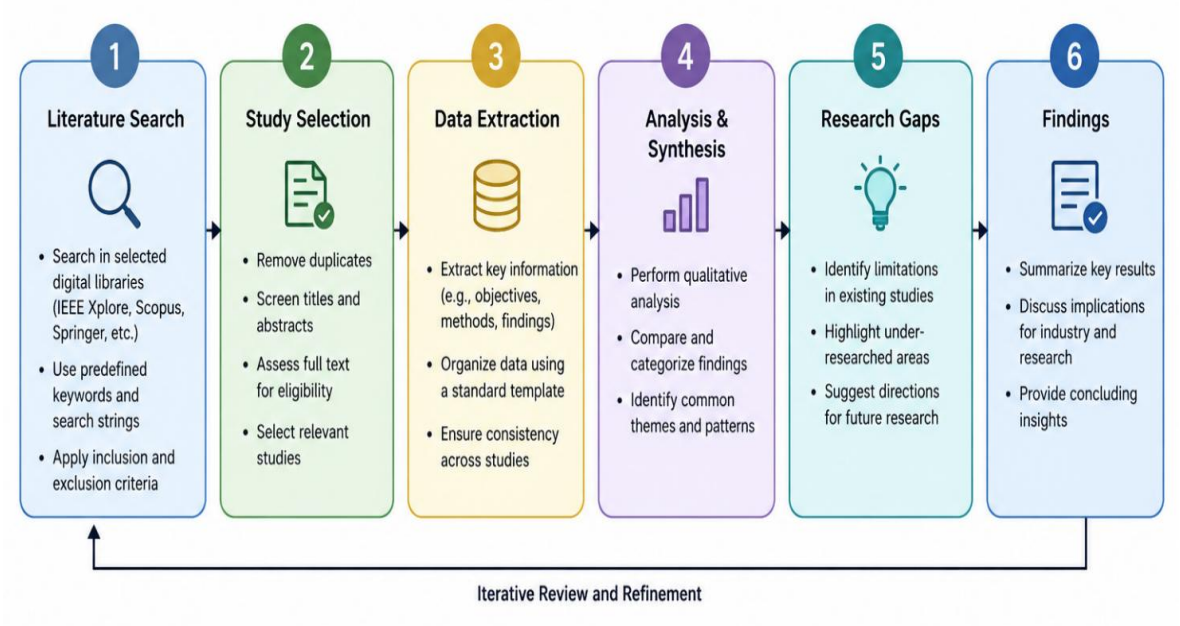
[14]	2024	Evaluation of LLM-generated code	Discusses evaluation beyond accuracy	Broader metrics are needed
[15]	2025	Systematic review of LLM-based code generation	Reviews LLM-based code-generation research	Evaluation gaps remai
[16]	2024	Security impact of AI-generated code	Reviews security impacts of AI models	Vulnerability mitigation remains open

V. RESEARCH METHODOLOGY

This study adopts a Systematic Literature Review (SLR) approach to identify, examine, and synthesize research on GenAI in modern software development. The review is organized around software engineering applications, productivity, quality, testing, security, documentation, trust, and responsible use. The general SLR process follows established software-engineering review guidance [21].

Relevant literature was identified using combinations of terms such as "Generative AI," "Large Language Models," "AI-Assisted Programming," "Code Generation," and "Software Development." The source material considered in this review includes research articles, conference publications, technical reports, reviews, and supporting technical or industry sources directly related to the topic. Studies outside the software-development context or unrelated to the research focus were excluded.

During analysis, studies were compared according to application area, reported benefits, limitations, evaluation concerns, security implications, and human–AI interaction. The synthesis was then used to identify recurring opportunities and research gaps. The review is qualitative in nature; no new experimental dataset or fabricated performance measurements are introduced.



IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, October 2026
Impact Factor: 8.2

VI. APPLICATION

Generative Artificial Intelligence is being applied across several activities of the software development lifecycle. Its applications extend beyond source-code generation and include programming assistance, testing, documentation, maintenance, and developer support. The major applications identified in the reviewed literature are discussed below.

A. Code Generation and Completion

GenAI can generate code from natural-language descriptions and provide completion suggestions during programming [1], [4], [15]. This can support initial implementation and reduce repetitive coding, while developers remain responsible for checking correctness and suitability.

B. Debugging, Review, and Testing

LLMs can help explain code, suggest changes, and generate test-related artifacts [3], [4]. In test-driven workflows, interactive AI assistance can connect implementation with testing, but generated tests and fixes require validation for correctness and coverage [3].

C. Documentation and Knowledge Support

GenAI can assist with technical documentation, code explanation, and API-related descriptions [12]. Such support can reduce documentation effort and improve access to software knowledge, but generated documentation should be reviewed against the actual implementation.

D. Requirements, Agile, and Architecture Support

Research considers GenAI for broader development activities, including software roles, Agile practices, and architecture support [6], [9], [10]. These applications indicate that GenAI can contribute beyond source-code production, although requirements and architectural decisions require contextual human judgment.

E. Security, Refactoring, and Maintenance

GenAI may assist refactoring, maintenance, and security-related analysis, but generated code can contain vulnerabilities [5], [20]. Security assessment, testing, and review are therefore necessary before AI-generated changes are incorporated into production systems.

VII. OPPORTUNITIES

The adoption of Generative AI creates several opportunities for improving software development activities. The reviewed literature indicates that GenAI can support developers by reducing repetitive effort, accelerating implementation, improving access to technical knowledge, and assisting with different stages of the development lifecycle.

A. Productivity and Reduced Repetition

AI coding tools may reduce effort for selected programming activities and provide rapid access to coding assistance [7], [19]. Benefits depend on the task and must be considered alongside verification effort.

B. Faster Prototyping and Development Support

Natural-language code generation can help developers create initial implementations and explore alternatives more quickly [4], [11]. This can be useful during prototyping when outputs are treated as drafts rather than final solutions.

C. Documentation and Knowledge Sharing

Automated assistance can support software documentation and explanations, potentially making technical information easier to produce and maintain [12].

D. Learning and Onboarding

GenAI can provide explanations and examples that support developer learning and onboarding [6], [13]. The value depends on the accuracy of explanations and the developer's ability to evaluate them.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, October 2026
Impact Factor: 8.2

VIII. CHALLENGES

Despite the opportunities offered by Generative AI, its use in software development introduces several technical, security, privacy, and human-related challenges. The reviewed literature indicates that AI-generated software artifacts cannot be assumed to be correct, secure, or suitable for direct use without appropriate evaluation and human review.

A. Incorrect or Unreliable Code

AI-generated code can be syntactically plausible while still being incorrect or unsuitable. Quality and consistency vary across models, prompts, tasks, and evaluation conditions [2], [4], [11], [14].

B. Security and Privacy

Generated software may contain security weaknesses, and the use of AI systems can raise concerns about sensitive information and responsible handling of development data [5], [8], [20].

C. Intellectual Property and Explainability

Responsible adoption includes questions about intellectual property, provenance, and the ability to understand why an AI system produced a recommendation [8], [13], [14].

D. Evaluation, Trust, and Overreliance

Traditional accuracy alone may not capture the practical quality of AI-generated software [14]. Developers also need appropriate levels of trust and skepticism. Research on workplace use shows that perceived usefulness and trust in generated code are not necessarily identical [7].

IX. RESEARCH GAP

Although recent studies have examined several applications of Generative AI in software development, important areas still require further investigation. The reviewed literature identifies a need for more trustworthy and explainable AI-assisted software development, particularly for generated code that must satisfy requirements related to correctness, security, and maintainability [5], [13], [14], [20].

A major gap is the lack of standardized evaluation metrics for AI-generated code. Existing studies evaluate generated outputs using different criteria, making it difficult to establish consistent comparisons across models, programming languages, and development tasks [2], [4], [11], [14].

Another gap concerns the long-term impact of GenAI on developer skills and human–AI collaboration. Although existing research discusses developer productivity, changing software-development roles, and trust in AI-assisted development, further research is required to understand how continuous use of GenAI may influence developers over longer periods [6], [7], [13].

The literature also indicates limited research on the integration of GenAI into Agile and DevOps practices and a need for more domain-specific AI models designed for specialized software-engineering tasks [8], [9], [15]. In addition, secure AI-generated software remains an important research area because generated code may introduce security vulnerabilities [5], [20].

Therefore, future research should focus on standardized evaluation frameworks, explainable and secure code generation, domain-specific AI assistants, long-term developer outcomes, effective human–AI collaboration, and responsible governance of GenAI in software development.

X. FUTURE SCOPE

The continued development of Generative AI is expected to create new research opportunities in software engineering. Based on the trends identified in the reviewed literature, future research can extend GenAI applications toward more autonomous, collaborative, and context-aware software-development environments.

A. Autonomous and Multi-Agent Development

Future systems may coordinate multiple specialized AI agents for coding, testing, documentation, and review. Such systems will require strong validation and governance.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 6, October 2026 Impact Factor: 8.2

B. AI-Assisted Architecture and DevOps

Further work can investigate how GenAI supports architecture decisions, continuous integration, deployment, monitoring, and maintenance while preserving human oversight [9], [10].

C. Personalized and Domain-Specific Assistants

Assistants adapted to programming languages, organizational standards, project documentation, or specialized domains may improve contextual usefulness [15].

D. Long-Term Evaluation

Longitudinal studies can examine sustained productivity, skill development, trust, software quality, and security outcomes rather than short-term task performance [6], [7], [13].

XI. CONCLUSION

Generative Artificial Intelligence is increasingly influencing modern software development by providing assistance across activities such as code generation, testing, debugging, documentation, maintenance, and software engineering support. The literature reviewed in this study indicates that GenAI can reduce repetitive development effort, accelerate selected development activities, and provide developers with interactive assistance throughout the software development lifecycle.

At the same time, the adoption of GenAI introduces important challenges related to code correctness, security vulnerabilities, privacy, intellectual property, explainability, evaluation consistency, and excessive dependence on AI-generated outputs. The reviewed studies also indicate that the effectiveness of GenAI can vary according to the model, task, programming context, and evaluation approach. Consequently, AI-generated software artifacts should be appropriately reviewed, tested, and validated before being incorporated into software systems.

The findings of this review suggest that GenAI is more appropriately considered a collaborative development assistant rather than a replacement for software engineers. Human expertise remains important for requirements interpretation, architectural decisions, security-critical activities, code review, testing, and responsible decision-making. Future progress in this area should therefore emphasize reliable and secure AI systems, standardized evaluation methods, explainability, domain-specific solutions, and effective human–AI collaboration.

Overall, the responsible adoption of GenAI can contribute to the evolution of modern software development when its capabilities are combined with appropriate human oversight, engineering practices, security controls, and governance.

REFERENCES

[1] A. T. Nguyen and T. N. Nadi, “An empirical evaluation of GitHub Copilot’s code suggestions,” in Proceedings of the 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), 2022, pp. 1–5. doi: 10.1145/3524842.3528470.

[2]. A. Clark, D. Igbokwe, S. Ross, and M. F. Zibran, “A quantitative analysis of quality and consistency in AI-generated code,” in Proceedings of the 7th International Conference on Software and System Engineering (ICoSSE), 2024. doi: 10.1109/ICoSSE62619.2024.00014.

[3] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, M. Musuvathi, and S. K. Lahiri, “Exploring the effectiveness of LLM based testdriven interactive code generation: User study and empirical evaluation,” in Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, 2024, pp. 390–391. doi: 10.1145/3639478.3643525.

[4] M. A. Shehab, M. Wardat, S. Omari, and Y. Jararweh, “Evaluating large language models for code generation: Assessing accuracy, quality, and performance,” in Proceedings of the 2nd International Conference on Foundation and Large Language Models (FLLM), 2024, pp. 407–416. doi: 10.1109/FLLM63129.2024.10852439.

[5] L. Criollo-Ramírez, X. Limón, Á. J. Sánchez García, and J. C. Pérez-Arriaga, “State of the art of the security of code generated by LLMs: A systematic literature review,” in Proceedings of the 12th International Conference in

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 6, October 2026
Impact Factor: 8.2

Software Engineering Research and Innovation (CONISOFT), 2024, pp. 331–339. doi: 10.1109/CONISOFT63288.2024.00050.
[6] T. Şimşek, A. Çağlar Gülşeni, and G. Arkali Olcay, "The future of software development with GenAI: Evolving roles of software personas," IEEE Engineering Management Review, vol. 53, no. 4, pp. 34–39, 2025. doi: 10.1109/EMR.2024.3454112.
[7] J. L. Butler, J. Suh, S. Haniyur, and C. Hadley, "Dear diary: A randomized controlled trial of generative AI coding tools in the workplace," in Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2025, pp. 319– 329. doi: 10.1109/ICSE-SEIP66354.2025.00034.
[8] S. Shah and A. Bajpai, "Responsible generative AI for software development life cycle," in Proceedings of the IEEE World AI IoT Congress (AIIoT), 2025. doi: 10.1109/AIIoT65859.2025.11105309.
[9] Malladi Nagarjun and Sudheer Reddy, "Generative AI in agile software development: A comprehensive survey," in Proceedings of the 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA), 2025. doi: 10.1109/ICIDCA66325.2025.11280486.
[10] R. Ramachandran, "Empowering software architects with artificial intelligence: Analyzing GitHub Copilot's role in modern architecture design," in Proceedings of the 7th International Conference on Software Engineering and Computer Science (CSECS), 2025. doi: 10.1109/CSECS64665.2025.11009821.
[11] A. E. Pakiş and A. Kolukısa-Tarhan, "Code generation in LLM families: A quality comparison of reasoner and regular models," in Proceedings of the 10th International Conference on Computer Science and Engineering (UBMK), 2025. doi: 10.1109/UBMK67458.2025.11207012.
[12] R. Gadamsetty, K. A. Demir, and B. Liu, "Generative AI for software engineering: Generative AI effectiveness and efficiency in software development documentation," in Proceedings of the Conference on AI x Software Engineering (AIxSE), 2025. doi: 10.1109/AIxSE64906.2025.00017.
[13] S. Sabouri et al., "Trust dynamics in AI-assisted development: Definitions, factors, and implications," in Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE), 2025. doi: 10.1109/ICSE55347.2025.00199.
[14] D. Rodriguez-Cardenas, "Beyond accuracy and robustness metrics for large language models for code," in Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, 2024, pp. 159– 161. doi: 10.1145/3639478.3639792.
[15]Umama et al., "LLM-based code generation: A systematic literature review with technical and demographic insights," IEEE Access, vol. 13, pp. 194915–194939, 2025. doi: 10.1109/ACCESS.2025.3631952.
[16] OpenAI, "GPT-4 Technical Report," 2023.
[17] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," 2024.
[18] Meta AI, "Code Llama: Open Foundation Models for Code," 2023.
[19] GitHub, "Research: Quantifying GitHub Copilot's Impact on Developer Productivity," 2023.
[20] C. Negri-Ribalta, R. Geraud-Steward, A. Sergeeva, and G. Lenzini, "A Systematic Literature Review on the Impact of AI Models on the Security of Code Generation," Frontiers in Big Data, 2024, doi: 10.3389/fdata.2024.1386720.
[21] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," EBSE Technical Report EBSE-2007-001, Keele University and University of Durham, Version 2.3, 2007.