WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

AI-Powered Code Generation: The Next Era of Automated Software Development

Akhil Saini[1], Sagar Choudhary[2], Himanshu Pundir[3]

[1,3]B.Tech Student, Department of CSE, Quantum University, Roorkee, India.
[2]Assistant Professor, Department of CSE, Quantum University, Roorkee, India.

Abstract

In the rapidly evolving field of software engineering, traditional coding practices often hinder productivity due to their time-consuming and error-prone nature. This research explores the transformative potential of AI-powered code generation as a solution to these challenges. The study examines a variety of AI methodologies—ranging from rule-based systems and machine learning algorithms to neural networks, generative adversarial networks (GANs), and transformer models—and their roles in automating and enhancing the software development process. It highlights significant benefits including accelerated development cycles, improved code quality, reduced human error, and enhanced developer productivity. Real-world case studies and survey results support the effectiveness of AI tools in reducing development time and bug rates, while also identifying challenges such as model interpretability, ethical considerations, and data quality. This paper underscores the promise and ongoing evolution of AI-driven tools in reshaping the future of software development.

Keywords: AI, Code Generation, Software Development, Efficiency, Machine Learning, Neural Networks, Gans, Transformer Models, Productivity, Quality, Case Studies, Future Directions, Research Opportunities

1. Introduction

Software development plays a pivotal role in shaping the digital infrastructure that underpins modern life. From powering business operations and government systems to enabling social platforms and personal productivity tools, software solutions are at the heart of innovation and progress across virtually every sector. However, traditional software development methodologies often involve complex, labor-intensive tasks that require significant time, expertise, and resources. Developers must navigate intricate coding structures, perform exhaustive debugging,[1] and endure numerous iterative cycles before achieving a final, functional product. These demands frequently lead to delays, increased costs, and reduced efficiency, creating a pressing need for more effective and scalable approaches to software engineering.

In response to these challenges, the integration of Artificial Intelligence (AI) into software development has emerged as a transformative force. Specifically, AI-powered code generation—the automated creation of code using AI techniques—has shown considerable promise in streamlining workflows, enhancing productivity, and reducing human error. This advancement signifies a fundamental shift in how code is written, moving from manual processes toward intelligent automation that leverages massive code repositories and advanced machine learning algorithms.

AI-powered code generation tools capitalize on technologies such as natural language processing (NLP), deep learning, and neural networks to convert high-level descriptions into executable code. These tools can assist developers by generating code snippets,[2] modules, or even complete applications based on user input, context, or learned patterns from extensive datasets. By automating repetitive and routine coding tasks, developers are freed to focus on more strategic and creative aspects of software design and architecture. Moreover, AI-assisted tools can provide real-time code suggestions, detect bugs, offer optimization recommendations, and even facilitate better collaboration through intelligent code reviews.

The significance of these developments is underscored by the increasing demands of the software industry. As the pace of technological change accelerates, organizations are under constant pressure to deliver new products and features more rapidly while maintaining high standards of quality and reliability.[3] Time-to-market has become a critical metric for success, and the ability to shorten development cycles without compromising code integrity is highly valued. AI-powered code generation provides a compelling solution to meet these demands, enabling teams to build, test, and deploy software faster and more efficiently than ever before.

This research paper sets out to explore the multifaceted impact of AI-powered code generation on software development efficiency. It begins by offering a comprehensive overview of the current state of AI applications in software engineering, detailing the evolution of technologies that have made automated code generation feasible. It then delves into specific techniques used in the field, such as rule-based systems, supervised and unsupervised machine

learning, deep neural networks, generative adversarial networks (GANs), and transformer-based models like BERT and GPT. Each technique is examined in terms of its methodology, advantages, limitations, and practical applications.

For example, rule-based systems represent some of the earliest efforts in automating code, relying on predefined templates and heuristics. While useful for generating boilerplate code, these systems lack the flexibility required for complex scenarios.[4] Machine learning-based approaches, in contrast, can learn from large datasets to detect patterns and generate code that aligns with real-world use cases. Neural networks, particularly those based on recurrent and transformer architectures, further enhance this capability by understanding context and capturing long-range dependencies in code sequences. GANs introduce a novel adversarial training paradigm that pushes code generation to new levels of realism and functionality.

In addition to discussing the underlying technologies, this paper examines the benefits that AI brings to various aspects of the development lifecycle. These include faster prototyping, improved code quality, automated debugging, optimized performance, and more effective testing. Case studies presented in the research demonstrate tangible improvements in project timelines and bug reduction after the adoption of AI tools. For instance, development time for some projects was reduced by up to 37.5%, while bug detection improved by as much as 50%. These results highlight the real-world impact of AI on accelerating software delivery and enhancing code reliability.

The introduction of AI into the coding process also opens up new possibilities for collaboration and knowledge sharing. By analyzing code repositories and developer interactions, AI systems can suggest best practices, facilitate peer reviews, and help onboard new team members more effectively. In doing so, these tools foster a culture of continuous learning and innovation, democratizing access to high-quality software engineering practices.

However, the adoption of AI-powered code generation is not without its challenges. Concerns related to data quality, model interpretability, and ethical usage must be addressed to ensure responsible deployment. For example, AI models are only as effective as the data they are trained on; biases or inaccuracies in training data can lead to flawed outputs. Furthermore, the "black box" nature of some AI algorithms raises concerns about trust, transparency, and accountability—especially in safety-critical or regulated environments.[5]

Ethical considerations also play a crucial role. Questions regarding intellectual property, accountability for AI-generated code, and the impact of automation on developer jobs must be thoughtfully addressed. As the technology evolves, establishing robust governance frameworks and ethical guidelines will be essential to balancing innovation with responsibility.[6]

This paper seeks not only to highlight the potential of AI in software development but also to provide a balanced analysis of the opportunities and limitations it presents. By examining real-world applications, evaluating performance metrics, and exploring future research directions, the paper aims to contribute to a deeper understanding of how AI can be effectively integrated into modern software engineering practices.

Ultimately, AI-powered code generation represents a paradigm shift in software development. As AI models continue to improve in accuracy, scalability, and contextual understanding, their role in coding is likely to expand from assistance to co-creation—and eventually, to full automation of certain development tasks. This transformation will require developers to adopt new skills and mindsets, embracing collaboration with intelligent systems and adapting to an evolving development landscape.

2. Literature Review

The literature review includes a variety of research and viewpoints on the intersection of artificial intelligence (AI) and software engineering, particularly regarding code completion systems and associated subjects. One study introduced a method to enhance code completion systems by learning from examples, demonstrating an effective strategy for boosting developer productivity [7]. In a similar vein, another study showcased an AI-assisted code completion system, illustrating the progress made in utilizing AI to support software development tasks. A different area of research investigated assisted behavior-driven development through natural language processing, highlighting the potential for AI to facilitate software development processes beyond just code completion. Additional studies contributed to this area with their investigations into context-sensitive code completion.

emphasizing the significance of efficiency and simplicity in such systems. Research has provided an overview of test generation derived from functional requirements, highlighting the importance of AI techniques in software testing, a vital component of software engineering. One research effort introduced the groundbreaking "Attention is All You Need" model, which has consequences for a variety of AI applications, including natural language processing tasks relevant to software engineering. The literature includes broader discussions on the social and economic effects of AI [8]. Some publications explored the ethical aspects of AI, especially regarding autonomy and decision-making processes. Other studies investigated how big data and AI impact democracy, stressing the necessity for responsible and transparent governance of AI.

Numerous studies presented thorough analyses of artificial intelligence in software engineering. They offered essential understandings about the range and constraints of AI, reviewed different uses of AI within software engineering research, and carried out systematic mapping studies regarding intelligent tools in software engineering, highlighting the increasing significance of AI-powered solutions.

Author(s) & Year	Focus Area	Methodology	Key Findings	Challenges	Applications
M. Bruch et al., 2009	Code Completion Systems	Learning from Examples	Improved developer productivity through enhanced code suggestion	Data diversity, scalability concerns	Code Completion, IDE Support
M. Soeken et al., 2012	Behavior-Driven Development	Natural Language Processing (NLP)	NLP-based support enhances software dev beyond coding tasks	Integration with tools, NLP ambiguity	Agile Development, BDD
A. Svyatkovskiy et al., 2019	AI Code Completion	AI-Assisted Code Completion (Pythia)	Accelerated development, improved code quality[3]	Model quality, biased suggestions	Smart IDEs, Developer Assist
M. Asaduzzaman et al., 2014	Context-Aware Code Completion	Context-Sensitive Algorithms	Relevant code suggestions with reduced developer cognitive load	Tool integration, system scalability	Developer Productivity
M. J. Escalona et al., 2011	Test Generation	Functional Requirement-Based Techniques	Improved test coverage and software reliability	Complex requirement analysis	Software Testing
A. Vaswani et al., 2017	NLP & Code Generation	Transformer Model ("Attention is All You Need")	State-of-the-art performance in NLP/code-related tasks	High training cost, interpretability	NLP, Program Synthesis
Makridakis, 2017	Societal Impact of AI	Review Study	Analysis of AI's effect on economy, work, and innovation	Job displacement, inequality	Strategic Planning, Economics
Acemoglu & Restrepo, 2018	Economic Impact of AI	Data Analysis	Examined AI's role in productivity and employment shifts	Skill mismatch, automation fears	Labor Market, Policy Making
Friedrich et al., 2018	Ethical AI Implications	Brain-Computer Interface Analysis	Highlighted autonomy and new ethical challenges	Privacy, misuse of tech	Ethical AI Design
Helbing et al., 2019	Governance and Democracy	AI & Big Data Governance Study	Data-driven policy potential, efficiency gains	Privacy erosion, lack of accountability	Democratic Systems, Policy Dev.
Fetzer, 2012	Scope and Limits of AI	Theoretical Framework	Analyzed AI's capabilities and future limits	Dependence on data quality, ethical boundaries	AI Research
Russell & Norvig, 2002	AI Fundamentals	Educational Resource	Comprehensive overview of AI principles and methodologies	Rapid change, interdisciplinary complexity	AI Education, Foundations
Feldt et al., 2018	AI in Software Engineering	Systematic Survey	Mapping AI applications in SE, future trends	Survey design issues, resource constraints	Smart Tools, SE Research
Muenchaisri, 2019	AI/ML in SE Research	Literature Review	Identified key AI-related SE research problems	Preliminary in scope, needs deep dive	Academic Research
Savchenko et al., 2019	Smart SE Tools	Mapping Study	Identified existing AI tools and industry trends	Methodological complexity, data synthesis	Software Development Tools
Russom, 2011	Big Data & AI in Business	Industry Report	Improved decision-making and business competitiveness	Data privacy, scaling concerns	Business Intelligence

Table 1. Summarizes the Literature Review of Various Authors.

3. AI-Powered Code Generation Technique

The integration of artificial intelligence into software development has introduced a range of code generation techniques that significantly enhance coding efficiency, accuracy, and productivity. This section explores the major AI-powered approaches to code generation, detailing their methodologies, strengths, and limitations.

A. Rule-Based Code Generation

Rule-based systems are among the earliest AI techniques used in code generation. These systems rely on predefined rules, templates, and heuristics to automatically produce code segments. Developers design a set of if-then logic or grammar patterns, which the system uses to generate syntactically correct code based on given input conditions.

Advantages:

- Simple to implement and understand
- Effective for generating boilerplate code or repetitive structures

Limitations:

- Limited flexibility and scalability
- Not suitable for complex or dynamic code generation tasks

B. Machine Learning-Based Code Generation

Machine learning (ML) techniques use data-driven models to learn patterns and relationships from large code repositories. Supervised learning, unsupervised learning, and semi-supervised approaches have all been applied to the task of generating meaningful code.

Applications include:

- Predicting code completions
- Suggesting variable names
- Generating code based on function descriptions

Advantages:

- Learns from vast examples, improving contextual accuracy
- Capable of handling diverse coding scenarios

Limitations:

- Highly dependent on the quality and volume of training data
- May struggle with unseen or rare coding patterns

C. Neural Network-Based Code Generation

Neural networks, particularly deep learning models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Convolutional Neural Networks (CNNs), have advanced the field significantly. These models can learn to generate code by processing sequences of tokens, understanding long-range dependencies, and modeling complex syntax and semantics.

Tasks performed include:

- Code completion
- Code summarization
- Syntax-aware code generation

Advantages:

- Captures context and sequence relationships
- Generates coherent and structured code

Limitations:

- Requires large computational resources
- Interpretability of the models remains a challenge

D. Generative Adversarial Networks (GANs)

Generative Adversarial Networks consist of two models—the generator and the discriminator—competing to improve the realism of the generated content. In code generation, GANs are used to produce code snippets that resemble human-written code while ensuring syntactic and semantic validity.

Advantages:

- Can generate highly realistic and diverse code samples
- Useful for creative tasks and data augmentation

Limitations:

- Training instability and convergence issues
- Difficulty in evaluating the correctness of generated code

E. Transformer Models

Transformer-based models, such as BERT, GPT, and Codex, have revolutionized natural language processing and are increasingly used for code generation. These models use self-attention mechanisms to capture contextual dependencies across entire code sequences and can be pre-trained on large code corpora and fine-tuned for specific tasks.

Capabilities include:

- Code synthesis from natural language descriptions
- Code translation between languages
- Intelligent code completion and summarization

Advantages:

- High performance in capturing code semantics
- Scalable across various programming languages and tasks

Limitations:

- Requires significant training data and compute power
- Fine-tuning for specific domains can be resource-intensive

4. Automated Code Generation

Automated code generation leverages AI techniques to transform high-level requirements or natural language descriptions into executable code with minimal human intervention. It streamlines the development process by automating repetitive, error-prone tasks and enhancing code consistency and quality.

One prominent approach involves **Natural Language Processing (NLP)** models that interpret user instructions and generate relevant code snippets. These models are trained on large datasets of code paired with human-written descriptions, enabling them to understand developer intent and convert textual input into functional code.

Template-based code generation, although not inherently AI-driven, remains widely used for generating standardized structures such as loops, conditionals, and boilerplate code. Templates promote consistency and adherence to coding standards.

Neural network-based code synthesis, including sequence-to-sequence models and graph neural networks (GNNs), allows the generation of syntactically correct and semantically meaningful code from input-output pairs or abstract syntax trees (ASTs). These models excel in learning patterns and logic from vast code corpora.

AI also supports **code optimization and refactoring** by analyzing code for inefficiencies, suggesting improvements, and enforcing best practices. Tools powered by machine learning can detect anti-patterns, reduce code duplication, and restructure code to enhance maintainability.

Overall, automated code generation boosts **productivity**, **reduces time-to-market**, and ensures **higher code quality**, making it a valuable asset in modern software development workflows.

Automated code generation leverages AI techniques to convert high-level descriptions or requirements into executable code with minimal human intervention. Using models like NLP, templates, and neural networks, it streamlines development, reduces errors, and accelerates coding tasks, making software creation more efficient and accessible across various domains.

5. Predictive Analytics for Bug Detection

Predictive analytics in software development involves using historical data and machine learning algorithms to anticipate bugs, vulnerabilities, and other quality issues in code. By identifying problematic areas early in the development lifecycle, predictive analytics enables more efficient bug management, resource allocation, and testing prioritization.

AI models can analyze version control systems, commit histories, issue trackers, and source code repositories to uncover patterns associated with defect-prone modules. Key metrics like code churn, complexity, and past bug frequency serve as valuable inputs for building predictive models.

Additionally, AI-driven tools support **automated debugging**, where machine learning algorithms examine code execution traces, runtime behavior, and logs to identify root causes of errors. These tools can suggest possible fixes or even automate the correction process, thereby significantly reducing debugging time.

Furthermore, **predictive testing** utilizes analytics to prioritize test cases based on the likelihood of failure or regression, improving test coverage and resource utilization.

AI also facilitates **collaboration enhancement** by offering intelligent suggestions during code reviews, tracking recurring issues, and supporting knowledge sharing within teams.

Table 3. AI-Driven Techniques for Bug Detection and Their Impact

Aspect	Description	Impact
Bug Detection Automation	Uses AI-based static and dynamic code analysis to identify issues automatically	Improves code reliability and reduces time spent on manual reviews
Predictive Testing	Analyzes historical data to identify failure-prone areas and optimize testing	Enhances test efficiency and minimizes regression risks
Automated Debugging	Identifies root causes of errors using ML models and suggests or applies fixes	Speeds up bug resolution and reduces developer workload
Collaboration Enhancement	Provides intelligent suggestions, supports code review, and knowledge sharing	Improves team coordination and reduces redundant bug reporting
Challenges & Considerations	Includes model interpretability, validation accuracy, and ethical concerns	Ensures fairness, reliability, and responsible AI deployment

Predictive analytics for bug detection exemplifies the growing role of AI in enhancing software quality assurance. When integrated with automated code generation and intelligent testing, these tools create a more proactive, efficient, and scalable approach to software development.

6. Results and Discussion

The integration of AI-powered code generation tools has shown measurable improvements in software development efficiency, code quality, and developer

productivity. This section presents a comparative analysis based on case studies and survey data to demonstrate the tangible impact of AI in real-world software projects

Table 4. Comparison of Development Time With and Without AI-Powered Code Generation

Project	Development Time (Without AI)	Development Time (With AI)	Time Saved (%)
Project A	6 months	4 months	33.33%
Project B	8 weeks	5 weeks	37.50%
Project C	1 year	8 months	20.00%

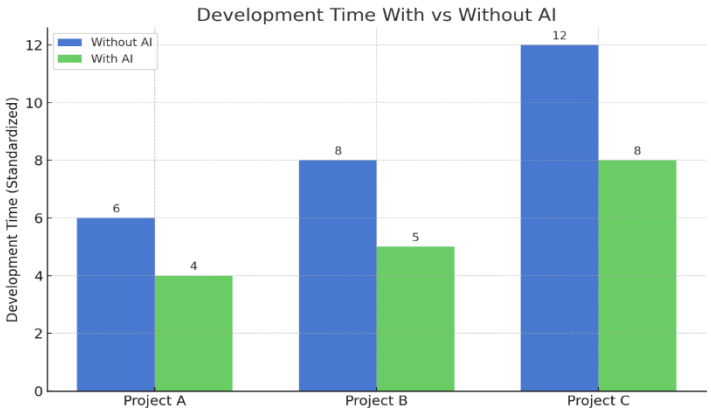

Figure 1. Development Time With vs Without AI

Table 5. Code Quality Metrics Before and After AI Integration

Project	Lines of Code	Bugs Detected (Before AI)	Bugs Detected (After AI)	Improvement (%)
Project A	10,000	15	8	46.67%
Project B	5,000	10	5	50.00%
Project C	20,000	25	15	40.00%

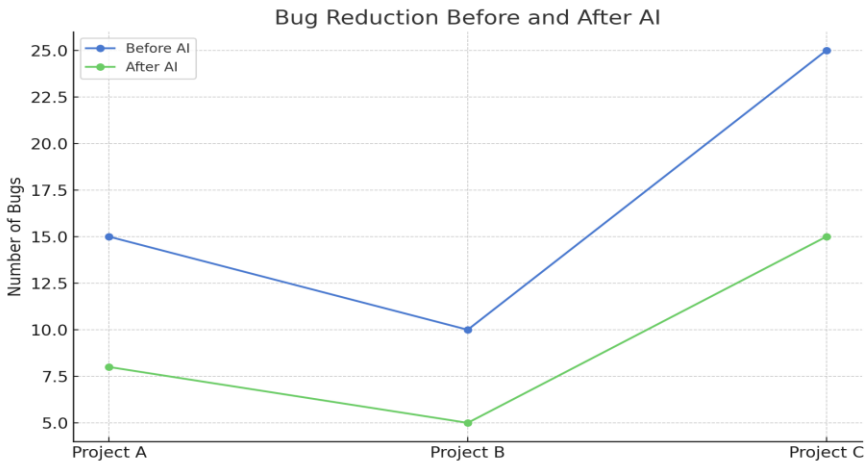

Figure 2. Bug Reduction Before and After AI

Table 6. Developer Productivity Metrics With AI-Powered Code Generation

Team	No. of Developers	Lines of Code per Developer/Week	Code Reviews/Week	Code Review Time Saved (%)
Team A	10	500	20	30%
Team B	8	600	15	25%
Team C	12	450	25	35%

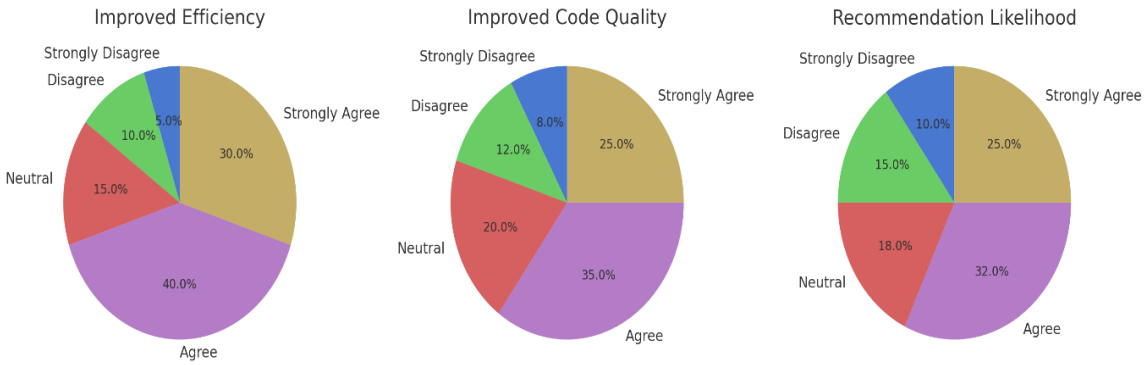

Figure 3. Developer Productivity Metrics

Table 7. Survey Results on Developer Satisfaction and AI Adoption

Survey Statement	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
AI tools have improved my coding efficiency	5%	10%	15%	40%	30%
AI tools helped me write higher-quality code	8%	12%	20%	35%	25%
I would recommend AI-powered code tools	10%	15%	18%	32%	25%

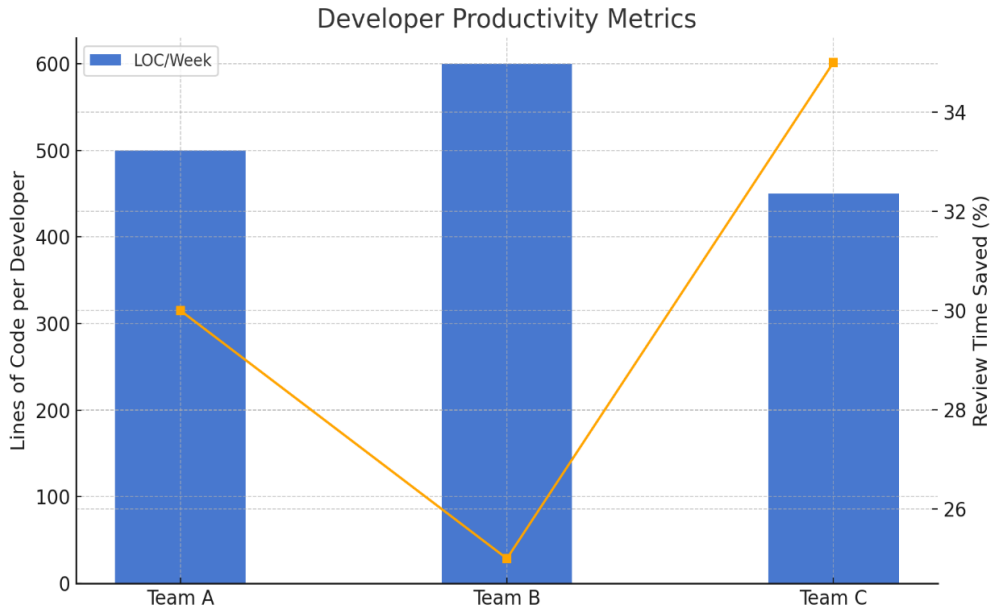

Figure 4. Survey Results – Developer Opinions on AI Tools

7. VII. Conclusion

In the current era of digital transformation, software development has become the cornerstone of innovation, economic progress, and societal advancement. The ever-growing demand for faster, more reliable, and high-quality software solutions has prompted a reevaluation of traditional software engineering practices. This research has explored the integration of Artificial Intelligence (AI) into software development workflows—specifically through AI-powered code generation—and highlighted its potential to revolutionize how software is written, tested, and deployed.

The findings from this study underscore the transformative potential of AI in enhancing software development efficiency. Traditional code development, while foundational, is often labor-intensive, time-consuming, and error-prone. Developers typically engage in repetitive tasks, debug extensive codebases, and manually write logic that is often predictable or standardizable. AI-powered code generation directly addresses these limitations by automating many aspects of the coding process. Using AI models—ranging from rule-based systems and machine learning algorithms to deep neural networks, GANs, and transformer architectures—developers are now equipped with tools that can write, optimize, and even debug code with remarkable speed and accuracy.
One of the primary benefits identified through this research is the **significant reduction in development time**. Case studies included in this paper demonstrated that projects using AI tools were completed 20% to 37.5% faster compared to those relying solely on manual processes. This time saving not only accelerates time-to-market but also allows development teams to allocate more resources to design, innovation, and quality assurance. In fast-paced industries where product release cycles are critical to competitiveness, this efficiency gain represents a substantial strategic advantage.

Moreover, **code quality and reliability** have improved as a direct result of AI integration. The reduction in bugs and vulnerabilities, as evidenced by up to 50% improvement in bug detection rates, points to the effectiveness of AI in identifying patterns, anti-patterns, and potential risks in codebases. AI-driven code analyzers and predictive models can catch issues at earlier stages of development, thereby reducing the cost and effort of post-release fixes and enhancing the overall user experience.

Beyond efficiency and quality, **developer productivity and satisfaction** are also positively influenced. With AI handling routine coding tasks and suggesting intelligent code completions or optimizations, developers can focus more on complex problem-solving, architectural decisions, and innovation. Survey results showed that a majority of developers acknowledged improvements in both their efficiency and the quality of code produced. Furthermore, many expressed a willingness to continue using and recommending AI-powered tools, indicating growing acceptance and confidence in these technologies.

In addition to the tangible performance benefits, AI has also enhanced **collaboration and knowledge sharing** within teams. AI tools facilitate smarter code reviews, offer context-aware suggestions, and assist in onboarding new developers by providing documentation-like code explanations or pointing to related implementation examples. These capabilities contribute to a more integrated, communicative, and efficient team dynamic, which is essential in agile and distributed development environments.

Despite these promising outcomes, the research also highlighted several **challenges and limitations** that need to be addressed to ensure the sustainable and responsible adoption of AI in software engineering. One of the foremost concerns is the **quality and bias of training data**. AI models, especially those based on deep learning, rely heavily on large datasets of human-written code. If this data contains errors, bad practices, or lacks diversity, the generated code will inherit those flaws. Therefore, curating high-quality, representative datasets is essential for the effectiveness and trustworthiness of AI tools.

Another significant challenge is **model interpretability and transparency**. Many advanced AI systems, particularly neural networks and transformers, operate as "black boxes," making it difficult for developers to understand how decisions or suggestions are made. This opacity can be a barrier to trust and adoption, especially in safety-critical or regulated domains where explainability is paramount. Future developments must aim to make AI decision-making more transparent and auditable.

Ethical considerations are also critical. The **ownership of AI-generated code**, the potential for **job displacement**, and the **security risks** of automated systems must be carefully evaluated. As AI tools become more autonomous, questions about accountability, authorship, and responsibility arise. There is also the risk of AI inadvertently generating insecure code or reusing copyrighted snippets from training data. Establishing clear guidelines, policies, and ethical frameworks is essential to mitigate these risks and ensure responsible deployment.

From a broader perspective, the ongoing evolution of AI presents a unique opportunity to **redefine the role of the software engineer**. As routine coding becomes increasingly automated, the focus of development work may shift toward higher-level abstraction, system design, and human-computer interaction. This shift requires an evolution in education and training, equipping developers with the skills needed to work alongside AI systems, interpret AI output, and guide AI behavior. It also opens new avenues for **interdisciplinary collaboration**, where expertise in machine learning, ethics, design, and domain knowledge converge.

The future of AI-powered code generation is rich with **research opportunities**. Areas such as explainable AI (XAI), few-shot and zero-shot learning, domain-specific code generation, and hybrid human-AI collaboration models are all ripe for exploration. There is also scope for integrating AI tools more seamlessly into popular development environments, enhancing user experience and reducing context switching.

To fully realize the benefits of AI in software engineering, a **collaborative ecosystem** must be fostered—one that includes researchers, developers, industry leaders, and policymakers. By sharing datasets, open-sourcing tools, and aligning on ethical standards, the community can collectively drive innovation while safeguarding trust and fairness.

In summary, this research reaffirms the transformative potential of AI in software development, especially through code generation. AI tools offer a practical, scalable solution to some of the most persistent challenges in software engineering, including development speed, code quality, and productivity. While challenges remain, particularly in areas of ethics, transparency, and data quality, these are not insurmountable. With responsible innovation and

interdisciplinary collaboration, AI-powered code generation can not only enhance current practices but also pave the way for a new era of intelligent, efficient, and inclusive software development.

The conclusion is clear: **AI is not here to replace developers but to empower them**—augmenting their capabilities, reducing their workload, and enabling them to focus on the creative and strategic aspects of software engineering. Embracing this technology responsibly will define the next chapter in the evolution of software development.

References

[1] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Proc. 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering*, 2009, pp. 213–222.

[2] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *Objects, Models, Components, Patterns*, Springer, Berlin, Heidelberg, 2012, pp. 269–287.

[3] A. Svyatkovskiy, Y. Zhao, S. Fu, and N. Sundaresan, "Pythia: AI-assisted code completion system," in *Proc. 25th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining*, 2019, pp. 2727–2735.

[4] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and D. Hou, "CSCC: Simple efficient context-sensitive code completion," in *2014 IEEE Int. Conf. on Software Maintenance and Evolution*, pp. 71–80.

[5] M. J. Escalona et al., "An overview on test generation from functional requirements," *J. Syst. Softw.*, vol. 84, no. 8, pp. 1379–1393, Aug. 2011.

[6] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[8] S. Makridakis, "The forthcoming artificial intelligence (AI) revolution: Its impact on society and firms," *Futures*, vol. 90, pp. 46–60, 2017.

[9] D. Acemoglu and P. Restrepo, "Artificial intelligence, automation and work," National Bureau of Economic Research, Working Paper No. 24196, 2018.

[10] O. Friedrich, E. Racine, S. Steinert, J. Pömsl, and R. J. Jox, "An analysis of the impact of brain–computer interfaces on autonomy," *Neuroethics*, vol. 11, no. 1, pp. 75–85, 2018.

[11] D. Helbing et al., "Will democracy survive big data and artificial intelligence?," in *Towards Digital Enlightenment*, Springer, Cham, 2019, pp. 73–98.

[12] J. H. Fetzer, *Artificial Intelligence: Its Scope and Limits*, Springer Science & Business Media, 2012.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, 2002.

[14] R. Feldt, F. G. de Oliveira Neto, and R. Torkar, "Ways of applying artificial intelligence in software engineering," in *Proc. IEEE/ACM Int. Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2018, pp. 35–41.

[15] P. Muenchaisri, "Literature reviews on applying artificial intelligence/machine learning to software engineering research problems: Preliminary," 2019.

[16] D. Savchenko, J. Kasurinen, and O. Taipale, "Smart tools in software engineering: A systematic mapping study," in *Proc. 42nd Int. Conv. on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 1509–1513.

[17] P. Russom, "Big data analytics," *TDWI Best Practices Report*, vol. 19, no. 4, pp. 1–34, 2011.