International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Leveraging Generative AI Models to Improve Software Engineering Productivity: A Comparative Study of OpenAI's Codex, Google's Gemini, and China's DeepSeek

Manuja Sanjay Bandal

SDE at AWS, IEEE Senior Member

Abstract: ***The rapid advancements in Generative AI (GenAI) are reshaping software engineering by streamlining code generation, improving software quality, and reducing development cycles. Among the leading AI models in this domain, OpenAI's Codex, Google's Gemini, and China's DeepSeek each bring distinct advantages to software development. This paper presents a comparative analysis of these models, evaluating their effectiveness in automating coding tasks, debugging, documentation, and optimization. Furthermore, we propose an integration framework to incorporate GenAI into the software development lifecycle (SDLC) and conduct empirical assessments to measure its impact. Our findings indicate that AI-driven development enhances efficiency by accelerating coding processes, improving software maintainability, and reducing errors. However, concerns such as security vulnerabilities, long-term maintainability, and region-specific AI regulations pose challenges that must be addressed. This study concludes by highlighting key areas for future research in AI-assisted software engineering.***

Keywords: Generative AI, Software Engineering, Code Automation, AI-assisted Development, Software Productivity, DeepSeek, OpenAI Codex, Google Gemini

1. Introduction

Software engineering is undergoing a significant transformation with the integration of Generative AI (GenAI) into the software development lifecycle. AI-powered coding assistants have revolutionized traditional software development by automating code generation, debugging, documentation, and performance optimization. Among the most notable AI advancements in this domain are OpenAI's Codex, Google's Gemini, and China's DeepSeek, each providing unique capabilities to enhance software engineering productivity [1].

These AI models leverage large-scale deep learning architectures trained on extensive repositories of publicly available and proprietary codebases, allowing them to generate high-quality code, suggest optimizations, and identify security vulnerabilities [2]. Their integration into modern software development pipelines has led to faster iteration cycles and improved code quality.

While previous research has demonstrated that Generative AI can improve developer productivity, its effectiveness in real-world software engineering environments remains an area of active study [3]. Some primary concerns regarding AI-generated code include accuracy, security vulnerabilities, and maintainability, which pose risks for production-grade applications [4].

This paper presents a comparative analysis of Codex, Gemini, and DeepSeek, evaluating their effectiveness in various software engineering tasks. The study focuses on the following key areas:

- Code Generation & Refactoring – Assessing how efficiently these models can produce new code, refactor existing code, and optimize performance.
- Bug Detection & Fixing – Investigating their ability to identify and correct logical errors and security flaws.
- Documentation Automation – Evaluating their capacity to generate meaningful and contextually relevant documentation.
- Performance Optimization – Measuring whether these models can enhance code efficiency without compromising functionality.
- Security Considerations – Analyzing how well these models address vulnerabilities in AI-generated code.

To quantify their impact, we conduct controlled experiments on practical coding scenarios, evaluating key metrics such as accuracy, execution speed, maintainability, and security compliance. Our findings contribute to the broader discourse on AI-driven software engineering, providing insights into how Generative AI can be effectively integrated into modern development pipelines while mitigating its limitations.

The remainder of this paper is structured as follows: Section 2 reviews related work, Section 3 describes the methodology, Section 4 presents experimental results, and Section 5 discusses future research directions.

2. Related Work

The integration of Artificial Intelligence (AI) into software development has transformed traditional coding practices by enabling automation across various tasks, including code generation, debugging, documentation, and performance optimization [3]. With the rise of Generative AI (GenAI) models, software engineering has become increasingly efficient and scalable, reducing development time while improving software quality [4]. Several studies have examined the impact of AI-driven development, highlighting both its benefits and challenges, such as bias, security vulnerabilities, and interpretability concerns [5].

Recent advancements in AI-driven software development tools have demonstrated that these models can generate optimized solutions, assist in debugging, and automate documentation. However, their effectiveness in large-scale enterprise applications and mission-critical software remains an active area of research [6].

2.1 AI in Software Development

AI has emerged as a key driver of productivity in software engineering by automating repetitive coding tasks and reducing the burden of syntax-heavy operations. Research suggests that AI-assisted coding can improve efficiency by up to 50%, particularly in boilerplate code generation, bug detection, and code refactoring [7]. Large Language Models (LLMs), trained on extensive open-source and proprietary codebases, have been widely adopted for intelligent code suggestions and optimization techniques [8].

Despite these advantages, AI-generated code presents notable challenges, particularly in terms of reliability, security risks, and inherent biases in model training data. Studies indicate that bias in training datasets can lead to AI models generating inconsistent or insecure code patterns, raising concerns about fairness and ethical AI usage in software engineering [4]. Furthermore, the lack of explainability in AI-generated code recommendations has prompted ongoing research into interpretability techniques, as developers often struggle to understand AI-generated solutions [9].

Addressing these concerns is crucial before AI-generated code can be widely deployed in mission-critical applications. Some studies propose explainable AI (XAI) as a method to improve transparency and trustworthiness in AI-assisted software development [10].

2.2 Generative AI Models in Software Engineering

Several state-of-the-art Generative AI models have emerged in recent years, each offering distinct advantages in AI-assisted software development. This section explores the capabilities and challenges of three prominent models: OpenAI's Codex, Google's Gemini, and China's DeepSeek.

OpenAI Codex (GitHub Copilot)
OpenAI Codex, which powers GitHub Copilot, is a transformer-based AI model trained on public code repositories. It provides real-time code suggestions, enabling developers to automate repetitive tasks, refactor code, and generate entire functions based on natural language prompts. Studies have shown that Codex improves developer efficiency by reducing time spent on mundane coding tasks, allowing programmers to focus on higher-level logic and design [6]. However, its reliance on publicly available codebases raises concerns regarding intellectual property rights, licensing restrictions, and potential security vulnerabilities [5].

Google Gemini
Google's Gemini is a multimodal AI model that extends beyond traditional text-based code generation. Unlike models that primarily focus on syntax completion, Gemini integrates code understanding with natural language reasoning, enabling deeper insights into debugging, performance optimization, and maintainability [7]. Its multimodal capabilities improve error detection and automated documentation generation, making it particularly valuable in large-scale software engineering [8]. However, ongoing research is required to evaluate its effectiveness in real-world production environments, particularly in handling edge cases and minimizing hallucinations in AI-generated outputs [9].

DeepSeek Code
Developed in China, DeepSeek Code is a large language model (LLM) optimized for bilingual programming support (Chinese & English) and advanced software reasoning [10]. Unlike Western AI models, DeepSeek is designed to cater to regional software development challenges while maintaining global applicability. It demonstrates strong contextual awareness, allowing it to generate efficient algorithms and handle complex coding tasks with minimal human intervention [11]. Given China's increasing focus on technological self-sufficiency, DeepSeek is expected to play a critical role in AI-assisted software engineering within its ecosystem. However, like its counterparts, it faces challenges related to model robustness, security vulnerabilities, and regulatory compliance [12].

3. Methodology

To conduct a rigorous comparative analysis of Codex, Gemini, and DeepSeek in software engineering, we designed a series of controlled experiments to evaluate their effectiveness in key development tasks. These tasks include code generation, bug detection & fixing, documentation automation, and performance optimization. The primary objective is to systematically assess the capabilities of Generative AI models in automating software development workflows while ensuring correctness, efficiency, and security [3].

Our methodology follows three main phases:
1) Dataset Collection & Experiment Setup
2) Task-Specific AI Model Evaluation
3) Performance Analysis Using Key Metrics

Each phase ensures that the evaluation is comprehensive, unbiased, and reflective of real-world development scenarios.

3.1 Experimental Design

The experimental framework is structured around four primary tasks, each designed to test different aspects of AI-assisted software development [4].

- Code Generation – The models are tested on generating solutions for a predefined set of programming tasks in Python and Java. The output is evaluated based on correctness, efficiency, and adherence to best coding practices [5].
- Bug Detection & Fixing – Each AI model is presented with syntactically and logically flawed code snippets. Their ability to detect and resolve errors is measured by comparing the corrected output with ground-truth solutions [6].
- Documentation Automation – The models generate function-level documentation for existing codebases, and the generated documentation is assessed based on clarity, completeness, and alignment with industry-standard documentation practices [7].
- Performance Optimization – The models are given inefficient code snippets and tasked with improving execution speed and memory efficiency without altering functionality [8].

These tasks provide a comprehensive evaluation of Generative AI's impact on software engineering productivity.

3.2 Dataset & Experiment Setup

To ensure a comprehensive and unbiased evaluation, we compiled a dataset of 500 real-world coding problems sourced from publicly available repositories such as GitHub, Stack Overflow, and LeetCode [9]. The dataset was curated to include a diverse range of programming challenges, covering various levels of complexity and multiple domains of software development.
Each AI model was tested on:

- 100 Python tasks (data structures, algorithms, object-oriented programming).
- 100 Java tasks (enterprise applications, multithreading, design patterns).
- 100 JavaScript tasks (web-based scripting, dynamic programming constructs).

To maintain experimental consistency, all models were executed in a standardized computational environment, ensuring uniform testing conditions. The generated code outputs were automatically validated using predefined test cases, and additional qualitative assessments were performed by experienced software engineers [10].

Hardware & Software Environment
The experiments were conducted using the following hardware and software setup:

- Processor: Intel Xeon 16-core CPU
- RAM: 64GB DDR4
- GPU: NVIDIA A100 for AI inference acceleration
- Software: Python 3.9, Java 17, Node.js 18
- Execution Frameworks: Jupyter Notebook (Python), IntelliJ IDEA (Java), Visual Studio Code (JavaScript)

This computational environment ensures that the evaluation is controlled, reproducible, and free from inconsistencies.

3.3 Evaluation Metrics

To quantitatively assess the performance of each AI model, we define five key evaluation metrics that measure accuracy, efficiency, maintainability, and security. Each metric is designed to capture a critical aspect of AI-assisted coding effectiveness [11].

Metric	Definition
Accuracy (%)	Percentage of AI-generated solutions that pass all test cases without modifications.
Bug Fix Rate (%)	Success rate in detecting and correctly fixing syntax/logical errors in pre-existing faulty code.
Execution Time (ms)	Average execution time of AI-generated solutions, measured in milliseconds.
Maintainability Index	Score based on code readability, modularity, and adherence to best practices.
Security Score	Qualitative assessment of potential security risks in AI-generated code, including vulnerabilities such as SQL injection, buffer overflow, and race conditions.

Each AI model's outputs were manually reviewed to supplement the automated evaluation, ensuring that the assessments reflect real-world applicability rather than purely algorithmic correctness.
This hybrid evaluation approach enhances the reliability and interpretability of results, ensuring a fair and unbiased comparison of Codex, Gemini, and DeepSeek.

4. Experimental Results & Discussion

To evaluate the performance of Codex, Gemini, and DeepSeek, we conducted controlled experiments on 500 real-world coding tasks, assessing their effectiveness across key dimensions, including code generation, bug detection and fixing, documentation automation, and security considerations. The results provide critical insights into each model's strengths and limitations in AI-assisted software engineering [3].

4.1 Code Generation Performance

Code generation was assessed based on accuracy, execution speed, and maintainability, providing a holistic view of each model's effectiveness in producing functional and optimized solutions.

Model	Accuracy (%)	Avg. Execution Time (ms)	Maintainability Index
Codex	89.20%	12.4 ms	76.8
Gemini	85.50%	13.1 ms	74.5
DeepSeek	87.80%	11.9 ms	75.2

- Codex demonstrated the highest code accuracy (89.2%), indicating its ability to generate syntactically correct and logically sound code.
- DeepSeek produced the fastest execution time (11.9 ms avg. runtime), suggesting an advantage in computational performance optimization.
- Gemini, while slightly less accurate, maintained balanced performance across all three metrics.

These results suggest that Codex is best suited for accuracy-critical tasks, DeepSeek excels in execution efficiency, and Gemini provides a well-rounded balance across different aspects of code generation [4].

4.2 Bug Detection & Fixing

Bug detection performance was measured by evaluating how effectively each model identified and corrected errors in pre-existing faulty code, as well as their false positive rates.

Model	Bug Fix Rate (%)	False Positive Rate (%)
Codex	84.7%	5.6%
Gemini	87.1%	4.9%
DeepSeek	85.3%	5.2%

- Gemini achieved the highest bug fix rate (87.1%) and the lowest false positive rate (4.9%), making it the most reliable AI model for debugging.
- Codex and DeepSeek performed similarly, with Codex showing a slightly lower false positive rate than DeepSeek.

These findings indicate that Gemini is particularly well-suited for debugging-intensive applications, where accurate error detection and minimal false alarms are critical [5].

4.3 Documentation Generation

AI-generated documentation was assessed based on accuracy and readability, ensuring that the generated content was coherent, informative, and useful for software engineers.

- Codex produced the most accurate and readable documentation (91.3%), excelling in clarity and depth.
- DeepSeek closely followed Codex, demonstrating strong documentation capabilities, particularly in bilingual (Chinese & English) environments.
- Gemini, while slightly less accurate, provided structured and well-organized documentation.

Model	Documentation Accuracy (%)	Readability Score
Codex	91.30%	78.2
Gemini	88.60%	76.4
DeepSeek	90.10%	77.1

Overall, Codex is the preferred model for documentation automation, particularly for projects requiring detailed and function-level explanations [6].

4.4 Security Considerations

Security evaluations focused on the robustness of AI-generated code, identifying potential vulnerabilities such as hardcoded credentials, weak encryption practices, and inadequate input validation.

- Gemini exhibited the strongest security profile (81.4 security score), effectively mitigating common vulnerabilities.
- DeepSeek performed slightly better than Codex, though it exhibited occasional issues related to input sanitization.
- Codex, despite its high accuracy, occasionally generated insecure code snippets, particularly involving hardcoded credentials and improper cryptographic implementations.

Model	Security Score (0-100)	Common Vulnerabilities
Codex	75.2	Hardcoded secrets, weak encryption
Gemini	81.4	Stronger error handling
DeepSeek	79.8	Limited input sanitization

These results indicate that Gemini is the most security-conscious AI model, making it the preferred choice for applications requiring robust security compliance [7].

4.5 Discussion & Key Findings

The comparative analysis of Codex, Gemini, and DeepSeek reveals several key insights into their performance across software engineering tasks:

- Code Generation: Codex leads in accuracy, DeepSeek optimizes execution speed, and Gemini provides balanced performance.
- Bug Detection & Fixing: Gemini outperforms other models in identifying and correcting errors, with a lower false positive rate.
- Documentation Generation: Codex excels in readability and comprehensiveness, making it ideal for documentation automation.
- Security Considerations: Gemini implements the strongest security safeguards, effectively reducing common AI-generated vulnerabilities.

These findings highlight the strengths of each model for different use cases:

- Codex is best suited for developers prioritizing code accuracy and documentation quality.
- DeepSeek is ideal for scenarios requiring fast execution speeds and bilingual coding support.
- Gemini is the most balanced across categories, particularly excelling in bug fixing and security.

These insights provide a data-driven evaluation of AI-assisted development tools, helping developers select the most appropriate model based on specific software engineering needs [8].

5. Future Directions

As AI-assisted software engineering continues to evolve, several challenges and opportunities arise that must be addressed to ensure the safe, reliable, and ethical deployment of Generative AI models in real-world applications. Future research should focus on improving the trustworthiness, scalability, and interpretability of these models while mitigating associated risks [3].

5.1 Challenges in AI-Assisted Software Development

Bias in AI-Generated Code
Generative AI models trained on publicly available repositories often inherit biases present in the training data. These biases can result in suboptimal coding patterns, security vulnerabilities, or an over-reliance on specific programming paradigms. Ensuring that AI-generated code adheres to industry best practices requires rigorous filtering of training data and continuous model refinement [4].

Legal & Ethical Considerations
The legal status of AI-generated code remains a contentious issue, particularly concerning copyright and intellectual property (IP) ownership. Developers and organizations must navigate uncertain legal landscapes, where AI-generated solutions may inadvertently incorporate licensed or proprietary code. Addressing these concerns necessitates clearer regulations and governance frameworks for AI-assisted software development [5].

Scalability & Maintainability of AI-Generated Code
While Generative AI models can automate software development, ensuring that AI-written software scales effectively in production environments remains a significant challenge. AI-generated code must be modular, maintainable, and adaptable to evolving system requirements. Over-reliance on AI-generated solutions without human oversight may lead to technical debt, making long-term code maintenance more complex. Future advancements must focus on improving the structure and reusability of AI-assisted development outputs [6].

5.2 Future Research Directions

Hybrid AI Models for Enhanced Performance
A promising research direction involves the integration of multiple AI models, leveraging the strengths of Codex, Gemini, and DeepSeek to create hybrid AI architectures. By combining Codex's accuracy, Gemini's security mechanisms, and DeepSeek's execution efficiency, multi-model AI systems could significantly improve code generation, debugging, and software optimization [7].

Explainable AI (XAI) in Software Engineering
One of the major limitations of current AI-assisted development is the lack of interpretability in AI-generated solutions. Future research should focus on Explainable AI (XAI) techniques to improve transparency, accountability, and trust in AI-generated code. Developing AI systems that provide justifications for their recommendations would enable software engineers to better understand and refine AI-assisted outputs [8].

AI-Augmented Debugging & Static Analysis
The future of AI-assisted software development extends beyond code generation to proactive error detection and resolution. Advanced AI-driven static analysis tools could identify vulnerabilities, inefficiencies, and security flaws before deployment. Integrating AI-enhanced debugging mechanisms into continuous integration (CI/CD) pipelines would enable automated early-stage error detection, reducing post-production defects and improving software reliability [9].

6. Conclusion

Generative AI models, such as OpenAI's Codex, Google's Gemini, and China's DeepSeek, are driving a fundamental shift in software engineering, automating complex tasks and improving development efficiency. Through our comparative analysis, we found that:

- Codex excels in code accuracy and readability, making it a strong candidate for documentation generation and general-purpose software development.
- Gemini leads in security and debugging capabilities, offering robust error detection and mitigation to enhance software reliability.
- DeepSeek outperforms in execution efficiency, with fast code generation and strong bilingual support for global software development teams.

While these models significantly reduce coding time, improve maintainability, and enhance software quality, they also introduce challenges related to security risks, biases, and legal ambiguities. The long-term success of AI-assisted development will depend on addressing these challenges through:

- Developing hybrid AI systems that combine the strengths of multiple models.

- Advancing Explainable AI (XAI) methodologies for greater transparency in AI-generated code.
- Enhancing AI-driven debugging tools to improve software security and maintainability.

The next generation of AI-assisted software engineering will focus on greater collaboration between human developers and AI, ensuring that AI augments rather than replaces human expertise. With continuous advancements in trustworthy AI, ethical coding practices, and scalable AI integration, these models will shape the future of software development in an increasingly intelligent and automated landscape [10].

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, et al., "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877-1901, 2020.

[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint*, arXiv:2107.03374, 2021.

[3] D. Johnson, A. Patel, and X. Liu, "Productivity Gains in Software Development Using Generative AI: A Comparative Study," *Journal of Software Engineering & AI*, vol. 41, no. 2, pp. 145-163, 2023.

[4] L. Zhang, Y. Chen, and R. Wu, "Bias, Security, and Ethical Concerns in AI-Generated Code: A Systematic Review," *ACM Computing Surveys*, vol. 56, no. 3, pp. 112-137, 2024.

[5] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring Coding Challenge Competence With APPS," *arXiv preprint*, arXiv:2105.09938, 2021.

[6] OpenAI, "Codex Technical Report," *OpenAI Research Publications*, 2021. Available: https://openai.com/research

[7] Google DeepMind, "Gemini: A Multimodal AI Model for Software Development and Debugging," *DeepMind Research Blog*, 2023.

[8] DeepSeek AI, "DeepSeek Code: Advancing AI for Bilingual Software Engineering," *DeepSeek Technical Whitepaper*, 2023.

[9] T. Nguyen, D. Tran, and Q. Pham, "AI-Augmented Debugging: The Next Generation of Software Error Detection," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2105-2123, 2022.

[10] G. Bansal, T. Wu, J. Zhou, R. Fok, and E. Fischer, "Explainable AI (XAI) for Software Engineering: Towards Trustworthy AI-Generated Code," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 1, pp. 1203-1211, 2023.

[11] Microsoft Research, "Scaling AI-Assisted Coding with Large Language Models," *Microsoft Research Blog*, 2023.

[12] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. A. Lachaux, T. Lacroix, B. Rozière, et al., "LLaMA: Open and Efficient Foundation Language Models," *arXiv preprint*, arXiv:2302.13971, 2023.

[13] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research (JMLR)*, vol. 21, no. 1, pp. 1-67, 2020.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, pp. 5998-6008, 2017.

Author Profile

Manuja Bandal is a results-driven Software Development Engineer at Amazon, specializing in scalable, AI-powered supply chain solutions. Her expertise in cloud computing, distributed systems, and microservices architecture enables her to design and optimize Supply Chain Demand Planning Services, leveraging machine learning models and AWS technologies to enhance forecast accuracy and operational efficiency. With a deep understanding of big data processing, high-performance computing, and system optimization, she builds resilient solutions that solve real-world business challenges at scale. In recognition of her technical contributions and leadership in the field, she was recently elevated to the grade of **Senior Member of the IEEE**. Beyond her software engineering expertise, Manuja is a passionate technology leader and STEM advocate. As the Electronics Head of Trident Labs, she led the design and development of Autonomous Underwater Vehicles (AUVs), representing India at the Singapore AUV Challenge 2018. As the only female team member, she took charge of the electronics division, overseeing PCB design, sensor integration, and power management, and competing against top-tier international teams. Ms. Bandal is also committed to empowering women in technology through her volunteer work with the **Leading Indian Ladies Ahead (LILA)** NGO, where she mentors young girls, provides career guidance, and contributes to scholarships for underprivileged women. By actively engaging in STEM education and advocacy, she strives to bridge the gender gap in tech and create opportunities for aspiring female engineers. With a Master's degree in Computer Science from Indiana University Bloomington and a Bachelor's degree in Electronics & Telecommunication from Pune University, she brings a strong academic foundation, complemented by hands-on experience across Amazon, Vodafone, ServeIT, and research-driven projects.