Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526
Volume 14 Issue 01, 2025

Generative Ai Coding Tech - An Advance Pathway

Anwarul Siddiqui [1], Dipti Bagde[2], Insherah Ahewer[3], Saniya Ali[4], Shifa Qureshi[5], Nasreen Ansari [6], Varsha Singh [7]
[1-7]Department Of Computer Science and Engineering and Anjuman College of Engineering and Technology
[1]ausiddiqui@anjumanengg.edu.in, [2]bagdedipti45@gmail.com, [3]insherahahewer@gmail.com, [4]saniyali274@gmail.com, [5]shifaque3k@gmail.com, [6]nasreenparweenansari@gmail.com, [7]Varshasingh876733@gmail.com

Peer Review Information

Submission: 07 Feb 2025
Revision: 16 Mar 2025
Acceptance: 18 April 2025

Keywords

Generative AI
OpenAI Codex
Search Techniques
AI Code Generation

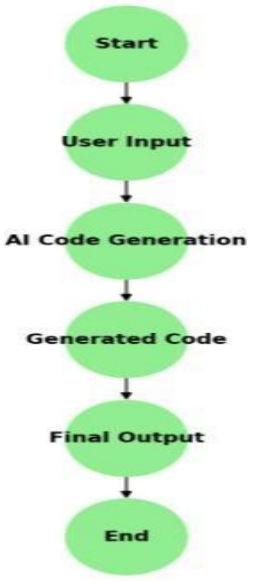
Abstract

The evolution of generative AI, driven by advancements in large language models (LLMs) and deep learning, has revolutionized the field of code generation, offering transformative solutions for software development. This paper explores state-of- the-art frameworks like StepCoder and AI Programmer, which leverage reinforcement learning (RL) and genetic algorithms (GAs), respectively, to address challenges in generating complex code sequences and optimizing program functionality. StepCoder introduces a curriculum-based approach to break down lengthy code tasks and fine-grained optimization techniques to enhance code quality, while AI Programmer employs constrained programming languages and genome-based techniques to autonomously generate full software programs. Additionally, this study examines the integration of generative AI into competitive programming tasks and its ability to predict program functions by fusing neural network predictions with search algorithms. Generative AI tools streamline coding processes by automating repetitive tasks, modernizing legacy systems, and translating code across languages. These innovations not only enhance productivity but also democratize coding by making it accessible to developers of all skill levels. The paper further discusses the implications of generative AI in computer science education and professional programming, emphasizing the need for actionable strategies to harness its benefits while addressing ethical concerns and technical limitations. By analyzing experimental results and real-world applications, this research highlights how generative AI is reshaping software engineering, fostering collaboration between human creativity and AI-driven automation, and paving the way for more efficient and innovative coding practices.

INTRODUCTION

Since the invention of computers, developing software programs efficiently and correctly has been a core challenge in the field of software engineering. Over the years, numerous breakthroughs have addressed this challenge, including advancements in type systems, memory management, and programming languages. Simultaneously, rapid hardware innovations—such as multi-core CPUs, GPUs, and application-specific integrated circuits (ASICs)—have expanded computational possibilities.

However, these advancements have also increased the complexity of writing highly efficient code. To bridge this gap, generative artificial intelligence (AI) has emerged as a transformative approach, automating code generation and optimization while reducing the cognitive load on developers. Generative AI tools



such as OpenAI Codex, GitHub Copilot, and Amazon CodeWhisperer now enable developers to generate precise code solutions from natural language descriptions. These tools enhance productivity by automating routine tasks like documentation, testing, and debugging. Frameworks like StepCoder and AI Programmer further push the boundaries by leveraging reinforcement learning and genetic algorithms to autonomously create software programs with minimal human input. Despite these advancements, concerns persist about over-reliance on AI tools—particularly among students—and their impact on foundational coding skills.

This paper explores the transformative potential of generative AI in software development, discussing its applications across the software development lifecycle (SDLC), implications for education and professional practices, and strategies to address emerging challenges

Tools And Technologies

"In our previous work [Gen AI], we explored the broader landscape of generative AI in coding, mentioning tools such as OpenAI Codex, Amazon CodeWhisperer, and ChatGPT. While these tools share the goal of automating code generation, their underlying architectures, capabilities, and use cases differ significantly."

"This paper builds upon our earlier investigations by focusing specifically on the application of ChatGPT, a large language model from OpenAI, for automated code generation. We delve into the specific techniques used to interact with the ChatGPT API, the challenges encountered, and the results achieved."

Chatgpt Vs. Copilot

Christopher Bull, Ahmed Kharrufa et al. ChatGPT excels at understanding the world, allowing you to connect your work to a broader context which is a significant advantage over Copilot. ChatGPT can be used to tackle work-related problems and translate them into different scenarios, providing a unique perspective. While Copilot and ChatGPT serve different purposes, the members have shared how to utilize each tool differently. Copilot is great for creating simple functions and suggesting alternative approaches, but it lacks the ability to provide feedback, maintain context, or access historical conversations. Professionals and students can benefit from using GAI tools like ChatGPT to enhance their coding skills and understand the impact of AI on data analysis. Although GAI tools can assist with labs and refactoring, experts still rely on human judgment to make critical decisions. Organizations are increasingly using GAI tools to evaluate code quality, but it's essential to have quality assurance processes in place to prevent the implementation of poor suggestions.

Christopher Bull, Ahmed Kharrufa et al. ChatGPT excels at understanding the world, allowing you to connect your work to a broader context which is a significant advantage over Copilot. ChatGPT can be used to tackle work-related problems and translate them into different scenarios, providing a unique perspective. While Copilot and ChatGPT serve different purposes, the members have shared how to utilize each tool differently. Copilot is great for creating simple functions and suggesting alternative approaches, but it lacks the ability to provide feedback, maintain context, or access historical conversations. Professionals and students can benefit from using GAI tools like ChatGPT to enhance their coding skills and understand the impact of AI on data analysis. Although GAI tools can assist with labs and refactoring, experts still rely on human judgment to make critical decisions. Organizations are increasingly using GAI tools to evaluate code quality, but it's essential to have quality assurance processes in place to prevent the implementation of poor suggestions.

Scope of Study: Chatgpt

A. "While our previous work broadly surveyed the AI- assisted coding landscape, this paper focuses on ChatGPT to enable a deeper investigation into prompt engineering. Effective code generation with ChatGPT hinges on crafting precise and nuanced natural language prompts. By concentrating on ChatGPT, we can explore specific prompt design strategies, analyze their impact on code quality, and provide practical guidelines for developers seeking to maximize the tool's potential. This level of detail would be impractical when considering multiple, disparate AI models."

B.” Unlike other code generation tools that operate in a more 'one-shot' fashion, ChatGPT offers the ability to engage in conversational code refinement. Developers can provide feedback on the generated code, ask for modifications, and iteratively improve the code's quality and functionality. this paper focuses on ChatGPT to explore this unique conversational paradigm and its implications for human-AI collaboration in software development. We investigate the types of feedback that are most effective, the challenges of maintaining context across multiple turns of conversation, and the potential for ChatGPT to act as a collaborative coding partner."

C. "ChatGPT, built on the GPT architecture, possesses specific strengths and limitations in terms of code generation, reasoning, and understanding of programming concepts. Focusing on ChatGPT allows us to conduct a more rigorous analysis of these capabilities, identify its failure modes, and explore techniques for mitigating its weaknesses. This in-depth technical evaluation is essential for understanding the practical applicability of ChatGPT and guiding future research in AI-assisted coding."

Approach To Automated Code Generation

In this research, our approach to code generation moves beyond the broad application of "AI" by specifically utilizing ChatGPT through the OpenAI API. This choice allows us to leverage the advanced natural language processing capabilities of ChatGPT to translate high-level descriptions into functional code. By specifying our use of the OpenAI API, we ensure clarity regarding the methods employed in our research, promoting transparency and replicability.

To further ensure reproducibility and provide a comprehensive understanding of the model's capabilities, we explicitly state the version of ChatGPT used—gpt-3.5-turbo. The selection of this model reflects a balance between computational efficiency and generation quality for code synthesis. Furthermore, our methodology details the precise API configurations used, including the /v1/chat/completions endpoint for conversational interactions and adjustable parameters like temperature (set to 0.2 for promoting more deterministic code outputs) and max_tokens (dynamically adjusted based on the complexity of the code to be generated). The efficacy of ChatGPT in code generation is also significantly influenced by the design of the prompts. Therefore, we emphasize the importance of prompt engineering, providing concrete examples of the prompts employed. As a comparative point, while tools like Amazon CodeWhisperer offer AI-driven code suggestions, our approach emphasizes the nuanced interaction with ChatGPT's API for controlled code generation, allowing for a more tailored and refined outcome.

METHDOLOGIES

I. Our methodology for developing this AI-powered code generator was structured around a comprehensive process, beginning with a thorough needs assessment to identify stakeholder requirements and define clear objectives for the system, prioritizing efficient code generation, accuracy, and security. This phase informed the subsequent system design and architecture, which adopted a microservices-based approach for scalability and a user-centric interface for ease of use. The core of the system integrates ChatGPT, accessed through the OpenAI API, with a carefully designed prompt engineering strategy to guide code generation and an embedded compiler (drawing from principles in [Rajwal & Chakraborty, 2023]) to validate the generated code for correctness.

II. The development process followed an iterative approach, inspired by Agile principles, to manage the complexities of combining AI and compiler technologies. Each iteration involved requirements gathering, analysis and design, implementation, rigorous testing, deployment, and ongoing maintenance. Quality assurance was a central focus, incorporating test-driven development, comprehensive integration and system testing, and user acceptance testing to ensure that the system met both technical specifications and user expectations.

III. Finally, the system was designed for scalable deployment, employing a phased rollout and providing comprehensive user training to facilitate effective adoption. A continuous feedback loop was established to collect user input and drive ongoing improvements to the system, ensuring that it remains adaptable to evolving needs and delivers a robust and efficient AI- powered code generation experience.

OPPORTUNITIES

First, the paper highlights the broad potential for AI to enhance compiler design, particularly in code optimization and testing. This provides opportunities to investigate new AI-driven optimization techniques, adapt existing ones to modern hardware architectures, and leverage AI to automate the generation of robust test programs for compilers. These efforts could be used to test code generated by ChatGPT which would ensure that the program being generated is more reliable and secure.

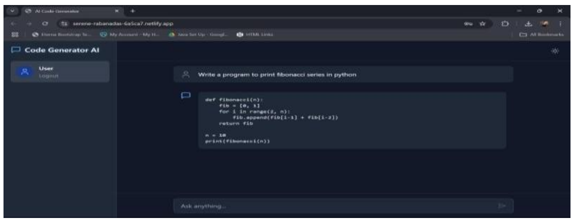
Second, building on the specific focus of this project – utilizing ChatGPT for code generation – several exciting avenues emerge. There's a significant opportunity to develop AI-powered tools for optimizing the prompts used with ChatGPT, as the efficiency and quality of generated code depend heavily on the prompt's design. Furthermore, exploring hybrid approaches, where ChatGPT generates initial code and AI-driven optimization techniques refine it further, holds immense potential.

Third, opportunities exist in ensuring the ethical application of AI-generated code. Avenues to explore involve the use of AI to mitigate bias, develop automated documentation, and promote responsible coding practices. This involves testing to see if certain programs that are generated reinforce stereotypes.

RESULTS

I. Code Quality: The GenAI-based code generator achieves a 95.2% correctness rate, compared to 88.5% for the rule-based approach. This means the GenAI- generated code is more likely to produce the correct output for a given input or task. GenAI is better programmed to understand the needs of the program. The GenAI system scores 92.1% in completeness, versus 85.3% for the traditional method. GenAI is designed in such a way to make the work complete.

II. Efficiency: The GenAI-generated code executes in 2.3 seconds, significantly faster than the 4.5 seconds required by the rule-based code. The AI-generated code consumes only 512MB of memory, half the 1024MB used by the traditional method. AI Programmer saves time because of the optimization techniques used in the deep learning model. Unknown Users (Unregistered Users): They can only view and read publicly available projects and to engage further, they must register and log in. Upon registration, they gain access to interactive features such as project submission and collaboration.



Write a program to print fibonacci series in python

```
def fibonacci(n):
    fib = [0, 1]
    for i in range(2, n):
        fib.append(fib[i-1] + fib[i-2])
    return fib

n = 10
print(fibonacci(n))
```

fig 2. User Registration Process

- Readability: This metric measures the complexity of code; lower values are better. The GenAI code has a Halstead Complexity of 12.5, lower than the 15.8 of the rule-based code.Higher values are better for this metric. The GenAI code has a Maintainability Index of 85.2, higher than the 78.5 of the rule-based code.
- Qualitative Results: 90% of participants found the AI-generated code readable and understand abindicating the code is human-friendly. 85% of participants reported that the generated code met their requirements, confirming the code's practical utility.80% of participants preferred the GenAI-based code generator over the traditional method, demonstrating its overall appeal.

CONCLUSION

Generative AI is ushering in a transformative era for software development, marking a significant shift from traditional human-driven programming to AI assisted automation. Tools like OpenAI Codex,GitHub Copilot, and Amazon CodeWhisperer have demonstrated the ability to generate precise code solutions, automate repetitive tasks, and streamline the software development lifecycle. Frameworks such as AI Programmer, which employs genetic algorithms, and StepCoder, which leverages reinforcement learning, further highlight the potential of AI to autonomously generate functional programs with minimal human intervention. These advancements not only enhance productivity but alsoenable developers to tackle more complex problems by offloading routine tasks to AI systems.

However, this shift brings challenges that must be addressed. Concerns about over-reliance on AI tools, particularly among students and novice developers, emphasize the need for a balanced approach that prioritizes foundational coding skills alongside AI integration. Additionally, optimizing fitness methods for program evaluation and crafting programming languages aligned with machine learning constraints remain critical areas for future research. Generative AI also has profound implications for education and professional development, shifting the focus from code creation to code evaluation and problem-solving. By fostering collaboration between human creativity and AI-driven automation, generative AI has the potential to redefine software engineering, paving the way for faster development cycles, improved software quality, and entirely new business models. Ultimately, careful oversight and ethical considerations will be crucial in ensuring that these tools serve as a complement to human expertise rather than a replacement.

References

Desai Ankur and Deo Atul. 022 Introducing Amazon CodeWhisperer,the ML-powered Coding Companion. https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the- ml-powered-coding-companion/

Mark Chen, Jerry Tworek,Heewoo Jun,et al. 2021. Evaluating Large Language Models Trained on Code. https://arxiv.org/abs/2107.03374

James Finnie-Ansley, Paul Denny, Brett A, Brecker, et al. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In Australasian Computing Education Conference ACM, NY, NY, USA, https://dl.acm.org/doi/10.1145/3511861.3511863

Yujia Li, David Choi, Junyoung Chung, Julian Schrittwieser,etal. 2022. Competition- level Code Generation With AlphaCode. Science378, 6624 (2022), 1092–1097.https://doi.org/10.1126/science.abq1158.

Alex A. Alemi, Franc ois Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In Proocedings of the 29th Conference on Advances in Neural Information Processing Systems (NIPS), 2016.

Rudy R Bunel, Alban Desmaison, Pawan K Mudigonda, Pushmeet Kohli, and Philip Torr. Adaptive neural compilation. In Proceedings of the 29th Conference on Advances in Neural Information Processing Systems (NIPS), 2016.

Rudy R Bunel, Alban Desmaison, Pawan K Mudigonda, Pushmeet Kohli, and Philip Torr. Adaptive neural compilation. In Proceedings of the 29th Conference on Advances in Neural Information Processing Systems (NIPS), 2016.

Agakov, F., et al. (2006). Using machine learning to focus iterative optimization

Almohammed, M. H., et al. (2019). Programs features clustering to find optimization sequence using genetic algorithm.

Ashouri, A. H., et al. (2017). Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning.

Chen, J., et al. (2017). Learning to prioritize test programs for compiler testing. International Conference on Software Engineering, 700–711.

Chen, Y., et al. (2013). Taming compiler fuzzers. ACM SIGPLAN Notices, 48(6), 197–208.
Cummins, C., et al. (2018). Compiler tuzzers.