TOOLS FOR AUTOMATED SOFTWARE CODE GENERATION WITH PROMPT ENGINEERING, EFFICIENCY AND THE EDUCATIONAL UTILIZATION POTENTIALS: A BRIEF REVIEW

DOI: 10.5937/EMC26297R

Branislava Radu
University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Republic of Serbia
E-mail: branislava.radu@tfzr.rs
Ljubica Kazi
University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Republic of Serbia

ABSTRACT

The development of generative artificial intelligence and large language models (LLMs) has enabled significant progress in automatic code generation and the application of AI systems in education and software engineering. This paper presents fundamental concepts of generative artificial intelligence, the importance of prompt engineering, prompting techniques and strategies for improving the quality of responses generated by large language models. Special attention is given to AI tools for code generation, their characteristics, operating principles and application in software development. The paper also analyzes studies that evaluate the performance, accuracy and quality of AI-generated code, as well as the potential applications of these tools in programming education. In addition to the advantages, challenges such as inaccurate responses, model bias, academic ethics and the need for human supervision are also discussed.

Key words: Generative AI, prompt engineering, large language models, AI code generation

INTRODUCTION

Generative artificial intelligence (Generative AI) and Large Language Models (LLMs) have become key technologies that are transforming the way digital content is created and used. Their ability to automatically generate text, images and source code has enabled greater work efficiency, automation of complex tasks, and the application of AI systems in education, medicine and finance (Yazdani et al., 2025; Aydın & Karaarslan, 2023). At the same time, these systems improve collaboration between humans and artificial intelligence, while also raising concerns related to bias, reliability and copyright issues (Feuerriegel et al., 2024). The development of generative AI has also led to the emergence of prompt engineering as an important approach for controlling the behavior of large language models. Prompt engineering involves carefully designing instructions in order to obtain higher-quality responses, using techniques such as zero-shot and few-shot prompting, as well as structured prompt templates (Sahoo et al., 2024; Cook, 2026).

This paper presents an overview and analysis of generative artificial intelligence and large language models, with a particular focus on prompt engineering, its techniques and prompt templates. Furthermore, the fundamental characteristics of generative AI systems and prompt engineering will be explained, together with their capabilities and limitations in practical applications. The paper will also present the architecture and workflow of AI systems for code generation, along with an overview of code generation tools. In addition, the efficiency and quality of AI-generated code will be analyzed through a comparison of different model performances. Finally, the application of these tools in programming education will be discussed, with particular attention given to their advantages and the challenges associated with their use.

GENERATIVE ARTIFICIAL INTELLIGENCE

Generative artificial intelligence (Generative AI) represents a field of artificial intelligence focused on generating new content based on data on which the model has previously been trained. Generative models are capable of creating original outputs such as text, images, videos, music and source code. These systems learn patterns and structures from large datasets and then use them to generate content similar to existing examples. Owing to their capability for automated content creation, generative AI systems are now widely applied in various fields, including education, medicine and finance(Aydın & Karaarslan, 2023).

The most significant approaches in the implementation of generative artificial intelligence include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), diffusion models, models based on the transformer architecture and hybrid architectures. Transformers form the foundation of Large Language Models (LLMs), including GPT models and ChatGPT. (Feuerriegel et al., 2024, Bandi et al., 2023) Generative AI also requires adequate hardware support, including GPUs, TPUs, cameras and microphones, as well as software tools such as PyTorch and TensorFlow, together with a high-quality user experience characterized by accuracy, speed and performance (Bandi et al., 2023).

Recent research results related to generative AI are focused on evaluation of Generative AI tools. One of the results present conclusion that evaluation results differ depending on the application domain. For image generation, realism is evaluated; for text generation, accuracy and informativeness are assessed; for source code generation, functionality is examined; and for audio generation, the naturalness of audio signals is analyzed (Bandi et al., 2023). Although generative models have achieved significant progress and widespread adoption, they still face numerous challenges, including high demands for data and computational resources, training instability, difficulties in evaluating the quality and creativity of outputs and limited generalization capabilities (Bandi et al., 2023). Furthermore, models often rely on statistical patterns rather than causal relationships and may contain biases and hidden assumptions (Manduchi et al., 2024). In addition, important ethical challenges remain, including hallucinations, lack of transparency and the potential misuse of these systems through the generation of convincing but inaccurate information (Banh & Strobel, 2023).

PROMPT ENGINEERING

The development of large language models has led to the emergence of prompt engineering, a technique that leverages the capabilities of pre-trained Large Language Models (LLMs). It is based on designing instructions that guide the model's output without modifying its parameters. Prompts may take the form of natural language instructions that provide context and help the model generate appropriate responses. The quality of outputs generated by large language models depends on carefully designed prompts used to direct the model's behavior (Sahoo et al., 2024). Within prompt engineering, various techniques have been developed to help models produce more accurate and higher-quality responses. (Sahoo et al., 2024):

- Zero-shot prompting is a fundamental technique in which a pre-trained large language model is asked to solve a task without being provided with prior examples, relying solely on instructions and knowledge acquired during training. Performance can be further improved through various sub-techniques, including additional instructions provided in interaction with a user.
- Few-shot prompting involves including one or more input–output examples directly within the prompt, thereby demonstrating the expected method of solving the task and enabling in-context learning without additional training. The effectiveness of this technique depends on the careful selection, number, arrangement, and formatting of examples, while research shows that properly structured and representative examples can significantly improve response quality.

Quality of generative AI results is improved by utilization of prompt templates, with structured approach to prompting. They provide key components that enable more precise control over generated

responses (Cook, 2026): Persona – defines the role and style of the model, determining the perspective from which the model responds; Instructions – describes the task and the expected result, including specific actions, response style and priorities; Context – additional information about the topic, target audience and environment in which the response will be used. If the few-shot technique is applied, examples may also be included; Constraints – limitations and rules that the model must follow, such as text length, prohibited expressions and safety parameters; Output – format of the final response, which may be presented as a paragraph, table, FAQ section, structured data or another standardized format suitable for further use. Prompt engineering includes structured prompt patterns and sometimes particular patterns are named upon the particular prompt part or prompting interaction type that is emphasized: Persona Pattern (role assignment), Question Refinement Pattern (question clarification), Template Pattern (format-based interaction), Flipped Interaction Pattern (where the model asks questions), Cognitive Verifier and Reflection patterns that support problem decomposition and explanation of responses, Meta Language Creation Pattern, Output Automater Pattern, Alternative Approaches Pattern, Recipe Pattern (White et al., 2023)

Research related to prompt engineering and generative AI are related to improvement of AI generated results. In the implementation of Generative AI based on prompting, particularly important are reasoning methods such as Chain-of-Thought, Tree-of-Thoughts and Graph-of-Thoughts, which enable step-by-step and multi-path logical reasoning while generating the result. To improve accuracy, methods such as Retrieval-Augmented Generation (RAG), ReAct and Chain-of-Verification are also used, reducing errors and hallucinations through the combination of generation and information verification (Giray, 2023). Limitations of prompt engineering are addressed in the context of generated results quality, such as ambiguous prompts, bias, lack of context and potential ethical concerns, which may lead to inaccurate or overly generic responses (Sahoo et al., 2024; Giray, 2023).

GENERATIVE AI TOOLS FOR SOFTWARE CODE GENERATION

In the research of (Becker et al., 2023), comparative analysis of Generative AI tools for code generation and their characteristics is presented and key results are provided in Table 1.

Table 1: Examples of generative AI tools for software code generation and their characteristics (Becker et al., 2023)

Tool	Characteristics
OpenAI Codex	generates code from natural language descriptions; supports multiple programming languages (Python, JavaScript, Go, PHP, Ruby, Swift, TypeScript, etc.); capable of translating code between languages; explains code functionality; supports working with APIs.
GitHub Copilot	AI programming assistant based on the Codex model; marketed as an "AI pair programmer"; provides code suggestions during programming; designed to increase developer productivity; free for verified students and teachers.
DeepMind AlphaCode	Transformer-based model for program generation; trained on more than 715 GB of GitHub code; supports C++, C#, Go, Java, JavaScript, Lua, PHP, Python, Ruby, Rust, Scala and TypeScript; solves complex algorithmic problems; achieves performance comparable to competitive programmers.
Amazon CodeWhisperer	AI tool for code recommendation and generation; uses machine learning; generates code based on natural language comments and existing code; recommends cloud services and libraries; integrates with IDE environments; aims to increase developer productivity.

Efficiency of Generative AI Tools in Software Code Creation

Recent research results are related to development of methods and systems for evaluation of generated software code. (Yetiştiren et al., 2023) performed evaluation of large language models trained on code. (Yetiştiren et al., 2023) performed experimental study with a system that includes creating prompts based on problem formulation, utilizing automated code generation tools and verification of the generated code quality. GitHub Copilot and Amazon CodeWhisperer were used within the Visual Studio Code environment, while ChatGPT was accessed through the OpenAI web interface in a

browser. (Yetiştiren et al., 2023) Once the code is generated, test cases are executed to verify whether the solutions are correct and executable. Afterwards, SonarQube is used for code quality analysis, including the evaluation of security, reliability, maintainability, the number of bugs and code smell issues. (Yetiştiren et al., 2023)

Manik (2025) compares the performance of large language models (LLMs) utilized for Python programming, where their efficiency is evaluated through accuracy, code quality, execution speed and code conciseness. The tools were tested on standardized tasks, and the results showed differences in performance: DeepSeek achieved higher accuracy and more frequently produced correct solutions, while ChatGPT generated more readable and concise code, but with a higher number of initial errors. Muñoz et al. (2024), compared different AI tools with tasks of varying complexity, ranging from basic algorithms to complex systems such as CRUD applications and REST APIs. Analysis using SonarQube and Jenkins tools showed that no single tool is universally superior; instead, performance depends on the programming language and task type. Simpler tasks produced better results, particularly in Python, Go and Ruby, while Java, PHP and C# proved to be more problematic. In more complex tasks, differences between tools became more pronounced, with Bing and Replit standing out as more stable solutions, whereas some tools exhibited more errors and code quality issues.

Li et al. (2024) explains the causes of performance variations and shows that Copilot-generated code often contains inefficient functions, poorly designed loops, and suboptimal algorithms, affecting performance and resource consumption. The study also highlights that prompt engineering can improve results, particularly through the use of specific prompts, but that limitations still exist, especially regarding memory optimization and more complex tasks.

Software Code Generation Tools in the Educational Context

Educational utilization of generative AI tools for software code creation has been evaluated in recent research results. Becker et al. (2023) point to the broader educational implications of these tools, emphasizing that systems such as Codex, AlphaCode and CodeWhisperer provide advantages including faster access to solutions, easier understanding of code, and the development of critical thinking through analysis. At the same time, they shift the educational focus from writing code toward understanding and evaluating it. Sheard et al. (2024) analyze computer science teachers' perceptions of the use of large language models in education. Assessment of generative AI tools was focused on their ability for explaining concepts and providing interactive learning support. It has been concluded that these tools have limitations, such as dependence on prompt quality, unclear explanations and their "black-box" nature which complicates full understanding and raises reliability concerns in educational contexts.

Building on these findings, Agbo et al. (2025) show that Generative AI is most commonly used in programming education at the university level in areas such as algorithms and software engineering, using tools such as ChatGPT, GitHub Copilot, and Codex. Although Generative AI enables personalized learning, code generation and explanation, error identification, and the development of creativity, limitations still exist regarding accuracy, consistency and the quality of explanations, as well as the risk of dependency on AI systems. The study concludes that Generative AI significantly improves computer science education but requires controlled and responsible use in order to preserve fundamental skills and independent problem-solving abilities.

CONCLUSION

The aim of this paper was to analyze the role of generative artificial intelligence-based tools for automatic code generation with a particular focus on prompt engineering and its impact on the quality, accuracy and efficiency of generated code. In addition, the study aimed to present the fundamental characteristics of AI tools, provide an overview of modern code generation tools and examine their application in programming education.

The research results indicate that generative artificial intelligence and large language models significantly contribute to the automation of code generation processes, increased productivity and the improvement of learning processes. However, the efficiency of these tools depends on the application context, task type, prompting quality and programming language, which means that no universally most effective solution can be identified. The analysis shows that the quality of generated code largely depends on the model and the way prompts are formulated, with prompt engineering representing an important factor in improving performance. In addition to these technical aspects, challenges related to system reliability, ethical implications, academic integrity and the risk of excessive reliance on AI tools were also identified, indicating the necessity of continuous human supervision and evaluation of generated results.In the educational context, the application of these tools contributes to improving the learning process through personalized support, faster understanding of programming concepts and the development of analytical skills. At the same time, challenges remain, including the reduced development of fundamental programming competencies, the possibility of academic dishonesty and difficulties in knowledge assessment due to the “black-box” nature of the models.

Future directions of development involve improving the accuracy and reliability of models through advanced prompt engineering techniques, hybrid approaches that combine code generation and verification, as well as the development of models with improved understanding of the semantic aspects of problems. Furthermore, it is necessary to establish standardized methods for evaluating the quality of AI-generated code and appropriate verification mechanisms, together with the definition of ethical and pedagogical guidelines for their application in education.

REFERENCES

Agbo, F. J., Olivia, C., Oguibe, G., Sanusi, I. T., & Sani, G. (2025). Computing education using generative artificial intelligence tools: A systematic literature review. Computers and Education Open, 9, 100266. https://doi.org/10.1016/j.caeo.2025.100266

Aydın, Ö., & Karaarslan, E. (2023). Is ChatGPT leading generative AI? What is beyond expectations?. Academic Platform Journal of Engineering and Smart Systems, 11(3), 118–134. https://doi.org/10.21541/apjess.1293702

Bandi, A., Adapa, P. V. S. R., & Kuchi, Y. E. V. P. K. (2023). The power of generative ai: A review of requirements, models, input–output formats, evaluation metrics, and challenges. Future Internet, 15(8), 260. https://doi.org/10.3390/fi15080260

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 500–506. https://doi.org/10.1145/3545945.3569759

Banh, L., & Strobel, G. (2023). Generative artificial intelligence. Electronic Markets, 33(1), 63. https://doi.org/10.1007/s12525-023-00680-1

Cook, D. A. (2026). The PICCO Framework for Large Language Model Prompting: A Taxonomy and Reference Architecture for Prompt Structure. arXiv preprint arXiv:2604.14197.

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2024). Generative AI: S. Feuerriegel et al. Business & Information Systems Engineering, 66(1), 111–126. https://doi.org/10.1007/s12599-023-00834-7

Giray, L. (2023). Prompt engineering with ChatGPT: a guide for academic writers. Annals of Biomedical Engineering, 51(12), 2629–2633. Springer Nature.

Li, S., Cheng, Y., Chen, J., Xuan, J., He, S., & Shang, W. (2024). Assessing the performance of ai-generated code: A case study on github copilot. In 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), 216–227. IEEE.

Manik, M. M. H. (2025). ChatGPT vs. DeepSeek: A comparative study on AI-based code generation. arXiv preprint arXiv:2502.18467.

Manduchi, L., Meister, C., Pandey, K., Bamler, R., Cotterell, R., Däubener, S., … & Fortuin, V. (2024). On the challenges and opportunities in generative ai. arXiv preprint arXiv:2403.00025.

Muñoz, M. A. F., De La Torre, J. C. J., López, S. P., Herrera, S., & Uribe, C. A. C. (2024). Comparative study of ai code generation tools: quality assessment and performance analysis. LatIA, (2), 21. https://doi.org/10.62486/latia2024104

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927.

Sheard, J., Denny, P., Hellas, A., Leinonen, J., Malmi, L., & Simon. (2024). Instructor perceptions of ai code generation tools-a multi-institutional interview study. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, 1223–1229. https://doi.org/10.1145/3626252.3630880

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., … & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. arXiv preprint arXiv:2302.11382.

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

Yazdani, S., Singh, A., Saxena, N., Wang, Z., Palikhe, A., Pan, D., … & Zhang, W. (2025). Generative AI in depth: A survey of recent advances, model variants, and real-world applications. Journal of Big Data, 12(1), 230. https://doi.org/10.1186/s40537-025-01247-x