Environment. Technology. Resources. Rezekne, Latvia
Proceedings of the 16th International Scientific and Practical Conference. 2025. Volume II, 151-156

The Role of Generative Artificial Intelligence in Programmer Training: Opportunities and Challenges

Stanka Hadzhikoleva
Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
Plovdiv, Bulgaria
stankah@uni-plovdiv.bg

Emil Hadzhikolev
Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
Plovdiv, Bulgaria
hadjikolev@uni-plovdiv.bg

Todor Rachovski
Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
Plovdiv, Bulgaria
todormr@gmail.com

Ivan Ivanov
Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
Plovdiv, Bulgaria
ivanivanov1040@gmail.com

***Abstract*—Artificial intelligence has rapidly entered people's daily lives and has innovated many traditional models by integrating AI technologies. The need for changes in educational practices has come to the forefront. This article outlines various possibilities for using generative artificial intelligence in programming education. Different examples are presented, including code generation, debugging and optimization, documentation and commenting, automated testing, conversion between different programming languages, and more. Additionally, some risks and limitations are discussed.**

***Keywords*—AI in education, AI in programmer training, generative AI in programming education, programming with AI.**

I. INTRODUCTION

Generative artificial intelligence (GenAI) has rapidly entered the daily lives of part of society. It has provided many different opportunities for optimizing activities traditionally performed by highly qualified specialists. Many AI-powered chatbots, such as ChatGPT, Gemini, Llama, Claude, Mistral, Cohere, Reka, DeepSeek, and others, are easy to use by anyone and offer free versions. They have a wide range of applications and can be used for automated content creation, performing calculations and analyses, language translation, code generation, image creation, information retrieval, and much more.

GenAI has also entered the field of education, presenting educators with numerous challenges. Some have welcomed the unexpected opportunities to support learning and optimize their work, while others have expressed concerns about the potential risk of reducing independent thinking and the development of higher-order thinking skills.

Tools like GitHub Copilot, ChatGPT, and other AI-powered assistants are transforming the way software developers write code, test it, debug, and optimize their work. Software companies are now seeking professionals who can effectively use AI tools for software development and task automation. To remain competitive in the job market, programming students must not only master traditional coding methods but also learn to work efficiently with AI tools.

AI is not a replacement for human thinking but rather a tool that can optimize various activities. For this reason, the education of future developers and software engineers must be innovated to include the study of AI's applications in software production.

This article explores some ways GenAI can be used to solve various programming tasks. Its goal is to provide educators with ideas on how to enrich the curriculum to help students develop specific skills for working with AI.

Scientific literature describes numerous studies on how GenAI is used in programming education and its impact on learning and academic outcomes.

Becker and colleagues open a discussion among computer science educators on how to integrate AI tools into programming education [1]. They argue that the rapid

Online ISSN 2256-070X
https://doi.org/10.17770/etr2025vol2.8608

and easy access to these tools has caught educators unprepared and surprised by the significant impact GenAI has on the learning process. Therefore, it is an urgent need to reconsider educational practices and adapt teaching methods to align with new technologies.

Wermelinger evaluates Copilot's ability to generate code and conducts a qualitative analysis of the generated suggestions to identify its limitations [2]. He shares insights from using Copilot for various tasks, including code explanation, test generation, and debugging. He emphasizes that educators need to be aware of Copilot's capabilities to adapt their teaching methods to AI-powered programming assistants.

In [3], a study is presented on Copilot's ability to generate and reproduce correct and efficient solutions for fundamental algorithmic problems. The research evaluates its performance and functionality in solving core computer science problems, such as sorting and implementing data structures. The authors compare Copilot's solutions with those provided by human programmers across various programming tasks. The findings indicate that Copilot successfully generates solutions for nearly all fundamental algorithmic problems. However, a significant drawback is that the chatbot struggles to combine multiple methods to construct comprehensive solutions. The comparison between Copilot and human programmers reveals that human programmers produced more correct solutions than Copilot. Additionally, the incorrect solutions generated by Copilot required less effort to fix compared to human-made errors.

In [4], a study is presented on the use of ChatGPT in programming and programming education. As part of the Object-Oriented Programming II course, students were assigned weekly project tasks in which they were allowed to use ChatGPT. A survey was conducted to gather students' opinions on the experience. According to the students, the main benefits of using ChatGPT in programming education include providing quick and mostly accurate answers to questions, improving thinking skills, facilitating the debugging process, and increasing confidence in solving programming tasks. However, the students also identified key drawbacks, such as the risk of developing laziness, ChatGPT’s inability to answer certain questions, and the provision of incomplete or inaccurate responses. Additionally, they expressed professional concerns about the future of the programming profession.

Liu and colleagues share their experience using AI tools in programming education. They provided students with a personal AI-powered virtual assistant that guided them toward solutions rather than offering direct answers [5]. The assistant supported learning by explaining code fragments, suggesting more readable and efficient code, and responding to both academic and administrative inquiries. Students provided positive feedback on the course, reporting an improved learning experience. The authors conclude that carefully integrated AI tools in education can significantly enhance the learning process by offering personalized support to students while allowing educators to focus on other pedagogical activities.

A study on how Codex performs on typical tasks from introductory programming courses is presented in [6]. To assess its effectiveness, the authors used real exam questions and compared its results with those of students who took the same exams under standard conditions. The findings show that Codex outperformed most students in terms of success rate. The study also examines the variability of Codex-generated solutions, revealing that identical input prompts often result in significantly different solutions, both in terms of algorithmic approach and code length. The authors conclude that AI offers numerous opportunities, including use for learning, assistance in problem-solving, and code analysis. However, they emphasize the need to address challenges such as potential misuse and the necessity of adapting teaching methodologies.

Yilmaz investigates how the use of ChatGPT in programming education affects students' computational thinking skills, self-efficacy in programming, and motivation toward the subject [7]. The researchers conducted an experiment in which students were randomly divided into two groups: an experimental group, which used ChatGPT during practical sessions, and a control group, which did not use it. After the training period, a survey was conducted to assess the impact. The analysis of the results showed that students in the experimental group demonstrated significantly higher computational thinking skills, programming self-efficacy, and motivation toward the subject compared to those in the control group.

Ouh and colleagues conducted a study aimed at evaluating the effectiveness of using the ChatGPT language model for generating solutions to programming tasks in an undergraduate Java course [8]. They analyzed ChatGPT-generated solutions for 80 different programming problems and compared them with correct solutions. The findings indicate that ChatGPT generates correct Java programming solutions characterized by high readability and well-structured organization. Additionally, the model is capable of suggesting alternative, memory-optimized solutions. However, a notable drawback is that ChatGPT struggles with tasks containing non-textual descriptions or requiring work with class files, often leading to invalid solutions. The authors emphasize the importance of educators understanding ChatGPT's limitations to make full use of its capabilities while minimizing the potential for misuse and academic dishonesty among students.

Savelka and colleagues investigate the ability of large language models (LLMs) to pass assessments in postgraduate-level Python programming courses [9]. The study evaluates GPT-4 on assessments of varying complexity, ranging from simple multiple-choice questions (without code) to complex programming projects with codebases distributed across multiple files – a total of 599 tasks. The authors find that GPT models have evolved to the point where they can effortlessly pass typical course

assessments. They caution that by using GenAI, students may be able to complete current standard programming assessments and achieve high scores with little effort.

The study presented in [10] aims to determine how effectively LLM models can identify issues in student-written code and provide adequate explanations. Two LLM models – OpenAI Codex and GPT-3.5 – were examined, and their generated responses were evaluated both quantitatively and qualitatively. The findings indicate that the advice provided by the models is often reasonable but not always accurate. A key drawback is that the models generally fail to identify all errors and produce a high number of false positives. They perform better at detecting logical issues in code than at handling output formatting requirements. Additionally, they tend to generate complete solutions even when explicitly instructed not to. In conclusion, the authors highlight the benefits of using LLMs in programming education. However, they also emphasize the unreliability of these models, noting that they make some of the same mistakes as students – particularly in formatting output according to the requirements of automated grading systems.

Lau and Guo conducted a study among university instructors teaching introductory programming courses [11]. They gathered educators' perspectives on how they plan to adapt their courses in the new reality of easy access to GenAI, ensuring that students continue to acquire knowledge effectively. The researchers interviewed 20 instructors from nine countries – nine women and eleven men. Their findings indicate that, in the short term, most educators are primarily concerned with taking immediate measures to prevent academic dishonesty. However, their long-term intentions vary. One group prefers to continue teaching programming in the traditional way – without AI involvement – while the other aims to integrate AI innovations into courses to better prepare students for the future job market.

A study on the opinions of professional software developers regarding the use of GenAI in software development was conducted by Bull and Kharrufa [12]. Based on their analysis of the results, they propose a vision for the future of education. They argue that educators should integrate GenAI technologies into the learning process rather than resist them. The authors also formulate pedagogical recommendations for incorporating AI into programming education.

Perkel provides advice on the effective use of ChatGPT in programming [13]. He emphasizes that despite its powerful capabilities, tools like ChatGPT are not as intelligent as they may seem and should be used with caution.

Su and Yang propose a theoretical framework called “IDEE” for integrating generative artificial intelligence into education [14]. The framework consists of four key components: Identifying desired learning outcomes, Determining the appropriate level of automation, Ensuring ethical considerations, and Evaluating effectiveness. The authors highlight the main advantages of this approach, including the potential for more personalized and efficient learning experiences for students, as well as faster and easier feedback for educators.

In [15], a significant issue related to the use of AI in programming education is discussed. In many cases, AI models can generate complete solutions to assigned tasks, potentially preventing students from developing essential creative and critical thinking skills. To address this challenge, the authors propose the Self-Regulated Learning Framework, which integrates the use of generative AI tools into the problem-solving process.

II. Materials and Methods

LLMs are a type of artificial neural networks specifically trained to understand and generate text in natural languages. They are complex models with many parameters that enable the modeling and analysis of intricate relationships between words and phrases in various textual contexts. These models are most commonly based on transformer architecture. They have a large number of parameters, on the order of billions, allowing them to analyze complex dependencies in text. They are trained on vast amounts of textual data from the internet, books, scientific articles, and other sources. Their learning process involves predicting the next word or phrase based on the preceding context.

LLMs can be used by programmers to optimize various activities. This includes code generation, debugging and optimization, documentation, testing, conversion between different programming languages, and more. Below, we will explore example prompts.

A. Code Generation

LLMs can automatically generate code based on textual descriptions or given requirements. This saves time and effort for routine tasks such as creating templates, standard functions, or data structures. Below are specific examples and prompts:

Generating a function for input validation: "Create a Python function that validates whether the given input is a number and falls within the range of 1 to 100."

Generating a REST API template: "Create a REST API template for user management with methods for adding, deleting, and editing users in Python using Flask."

Generating a class template: "Create a Java class for managing a library with methods for adding, removing, and searching for a book by title."

Generating a web form template: "Generate an HTML login form with fields for username and password and a login button."

Function for data processing: "Create a Python function that reads a list of numbers and returns a list of their square roots."

Class for managing a bank account: "Create a JavaScript class for a bank account with methods for depositing, withdrawing, and checking the balance."

Class for managing users in a system: "Create a C# class that represents a user in a system, with a name, email, and methods for changing the password and validating the email address."

B. Code Debugging and Optimization

LLMs can detect errors in code and suggest corrections and improvements. They can identify logical, syntactic, or semantic errors in the code. They can also enhance code performance by suggesting modifications using more efficient built-in functions or methods, as well as replacing recursive functions with iterative ones to improve performance.

Fixing a code error: "Find and fix the error in the following Python function that sorts a list of numbers in descending order: …"

SQL query optimization: "Optimize the following SQL query for faster execution on large tables: …"

Fixing and optimizing a recursive function: "Fix and optimize the following recursive function for calculating the factorial of a number in Python: …"

JavaScript code optimization: "Optimize the following JavaScript code for finding the maximum value in an array: …"

Fixing a logical error in code: "Find and fix the logical error in the following Java code, which is supposed to check whether a number is prime: …"

C. Documentation and Comments

Models can automate the creation of documentation by generating comments and explanations for functions, classes, and methods. This facilitates code maintenance and helps other programmers quickly understand its purpose and logic. Learners can take advantage of AI-generated comments that include: Brief explanations of the logic and steps in the code; Detailed descriptions of functions, classes, or methods, including their parameters, possible exceptions, and return values; Descriptions of complex algorithms or functionalities to improve understanding and maintainability, etc.

Adding comments to code: "Add comments to the following Python code that checks if a given string is a palindrome: …"

Generating documentation for a function: "Create documentation for the following Java function that calculates the sum of numbers in an array: …"

Adding documentation to a class: "Add documentation to the following JavaScript class for managing inventory in a store: …"

Generating documentation for API methods: "Create documentation for the API methods of the following Python REST API for user management: …"

Adding comments for a complex algorithm: "Add comments to the following Python code that implements the QuickSort algorithm: …"

D. Maintenance and Debugging

Other important activities that can be optimized with the help of LLMs are maintenance and debugging. Learners can ask questions about errors in their code that cause unexpected behavior to identify syntactic, logical, or semantic errors. They can also receive suggestions for improving the performance, efficiency, or security of their code.

Finding an error in Python code: "Identify and fix the error in the following Python function that calculates the factorial of a number: …"

Debugging an SQL query: "Why does the following SQL query return the error 'Ambiguous column name': …"

Debugging a logical error in JavaScript: "Find the logical error in the following JavaScript code, which is supposed to check if a number is even: …"

Debugging an application with an API request error: "Help me debug a 404 error I receive when trying to send a POST request to my Python Flask API: …"

Debugging an infinite loop: "Find and fix the cause of the infinite iterations in the following Python loop: …"

Debugging a memory issue in Java: "Optimize the following Java code that throws an OutOfMemoryError: …"

E. Automated Testing

LLMs can support automated testing in various ways. They are capable of generating tests that cover basic and edge cases for a given function or method, as well as creating tests for different API methods to ensure they function correctly. Learners can use them to generate tests with tools like Selenium for automating functional testing of web applications. LLMs can also be useful in error handling and exception processing, verifying different possible outputs and the code's response to various inputs.

Generating test cases for a Python function: "Create automated tests for the following Python function that calculates the square root of a given number: …"

Generating JUnit tests for a Java method: "Create JUnit tests for the following Java method that checks whether a given string is a valid email: …"

Generating automated tests for a REST API using Postman or pytest: "Create Postman tests for the API methods of the following Python Flask REST API for product management: …"

Generating automated tests for a web application using Selenium: "Create Selenium tests for a web application that includes a login form with username and password fields."

Generating unit test cases in C#: "Create Unit tests for the following C# method that calculates the area of a rectangle: …"

F. Conversion Between Programming Languages

LLMs can convert code from one programming language to another while adhering to the syntax rules of the target language. They can also translate language-specific libraries and methods into their equivalents in the target language. This is particularly useful for project migration or working in a multi-platform environment.

Converting a Python function to JavaScript: "Convert the following Python function that calculates the factorial of a number into JavaScript: …"

Converting a C# method to Python: "Convert the following C# method for calculating the average value of an array of numbers into Python: …"

Converting a JavaScript function to Java: "Convert the following JavaScript function that checks whether a number is prime into Java: …"

Converting a Java method to Python: "Convert the following Java method that sorts an array of numbers using the insertion sort algorithm into Python: …"

Converting a Python script to Bash: "Convert the following Python script that copies all .txt files from one directory to another into a Bash script: …"

G. Best Practices Recommendations

LLMs are capable of suggesting best practices for software design, architecture, and security, helping programmers create more reliable and secure code. Learners can receive specific advice and guidelines related to programming languages, software tools, or technologies. They can also gain insights into established practices for writing readable code, improving code organization, and managing projects more effectively.

Best practices for writing Python code: "What are the best practices for writing Python code?"

Best practices for designing a REST API: "What are the best practices for designing a REST API?"

Best practices for SQL query optimization: "What are the best practices for optimizing SQL queries?"

Best practices for web application security: "What are the best practices for ensuring security in web application development?"

Best practices for memory management in Java: "What are the best practices for memory management in Java?"

Best practices for working with Git: "What are the best practices for using Git in a team environment?"

H. Learning Support and Development

LLMs can serve as interactive learning tools, providing personalized advice and explanations on new technologies, programming languages, or concepts, accelerating the learning process. They are capable of: Explaining basic and advanced topics in programming and software development; Providing instructions and examples for learning new technologies and concepts; Clarifying important development tools and industry best practices, etc.

Learning a new programming language: "Explain the basics of programming in Rust for beginners."

Learning a new technology or library: "Explain the fundamentals of Docker and how to create my own container."

Learning best coding practices: "What are the best practices for writing secure JavaScript code?"

Support for advanced concepts: "Explain the concept of 'Dependency Injection' in the context of .NET Core."

Support for technical interview preparation: "How can I prepare for a front-end developer interview?"

Learning new development tools: "What are the essential Git commands, and how do I use them?"

III. Results and Discussion

The use of GenAI to optimize various activities in software development provides significant advantages for programmers. The ability to generate program code saves time spent writing code from scratch and accelerates the software production process. LLMs can offer different approaches to solving a given problem, contributing to more efficient solutions. The ability to quickly identify errors and optimize code improves the quality of software development. Identifying potential areas for improvement, removing redundant code, reducing complexity, and enhancing efficiency by using better algorithms or data structures are tasks typically performed by highly experienced specialists. The automated creation of various types of tests — unit tests, integration tests, and performance tests — improves test coverage and reduces the likelihood of errors and omissions.

Despite all the mentioned advantages, the use of LLMs in software development carries some risks and challenges. The following are considered the most significant:

- LLMs can generate code that, at first glance, appears correct but contains logical errors or does not meet the specifications. Programmers must carefully review the code to avoid bugs;
- LLMs can introduce biases through the data they were trained on, such as social, racial, gender, or cultural biases. This can reflect on the generated code, proposed solutions, or violate best practices;
- Using public platforms or cloud services with AI to process sensitive data may lead to data confidentiality issues. Therefore, it is necessary to apply mechanisms for anonymizing sensitive information, develop or adhere to established standards for personal data protection, such as GDPR;
- AI can be used maliciously, e.g., to create viruses or to discover vulnerabilities in software for hacking attacks or financial fraud. This highlights

the need for security regulations to prevent abuses of AI;

- Some LLM platforms require significant resources for training and use, which can be expensive for small and medium-sized companies;
- Automation of various tasks in software production leads to a reduction in jobs, which indirectly affects the entire economy.

Here, we should also note another significant risk – the development of dependence on artificial intelligence. Programmers may begin to rely too much on LLM in their work, which could reduce their ability to think critically and solve problems independently.

IV. Conclusion

Programmers are increasingly using LLMs such as ChatGPT, Gemini, Llama, Claude, Mistral, Cohere, Reka, DeepSeek, and others in their work. With their help, they optimize various activities in the software development process, including code generation, debugging and optimization, documentation, testing, conversion between different programming languages, and more.

With the advancement of these technologies, it is becoming increasingly important for students to learn how to use them effectively for more efficient programming. In addition to other benefits, LLMs can support learning and skill development by providing examples, explanations, and guidance for solving complex problems. The examples presented in this article illustrate only a small portion of the various ways GenAI can assist programmers.

References

[1] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard-or at least it used to be: Educational opportunities and challenges of AI code generation," in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, Mar. 2023, pp. 500–506. https://doi.org/10.1145/3545945.3569759.

[2] M. Wermelinger, "Using GitHub Copilot to solve simple programming problems," in Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. V. 1, Mar. 2023, pp. 172–178. https://doi.org/10.1145/3545945.3569830.

[3] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "GitHub Copilot AI pair programmer: Asset or liability?" J. Syst. Softw., vol. 203, p. 111734, 2023. https://doi.org/10.1016/j.jss.2023.111734.

[4] R. Yilmaz and F. G. K. Yilmaz, "Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning," Comput. Hum. Behav.: Artif. Humans, vol. 1, no. 2, p. 100005, 2023. https://doi.org/10.1016/j.chbah.2023.100005.

[5] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, and D. J. Malan, "Teaching CS50 with AI: Leveraging generative artificial intelligence in computer science education," in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. V. 1, Mar. 2024, pp. 750–756. https://doi.org/10.1145/3626252.3630938.

[6] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The robots are coming: Exploring the implications of OpenAI Codex on introductory programming," in Proc. 24th Australas. Comput. Educ. Conf., 2022, pp. 10–19. https://doi.org/10.1145/3511861.3511863.

[7] R. Yilmaz and F. G. K. Yilmaz, "The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation," Comput. Educ.: Artif. Intell., vol. 4, p. 100147, 2023. https://doi.org/10.1016/j.caeai.2023.100147.

[8] E. L. Ouh, B. K. S. Gan, K. Jin Shim, and S. Wlodkowski, "ChatGPT, can you generate solutions for my coding exercises? An evaluation on its effectiveness in an undergraduate Java programming course," in Proc. 2023 Conf. Innov. Technol. Comput. Sci. Educ. V. 1, Jun. 2023, pp. 54–60. https://doi.org/10.1145/3587102.3588794.

[9] J. Savelka, A. Agarwal, M. An, C. Bogart, and M. Sakr, "Thrilled by your progress! Large language models (GPT-4) no longer struggle to pass assessments in higher education programming courses," in Proc. 2023 ACM Conf. Int. Comput. Educ. Res. - Vol. 1, Aug. 2023, pp. 78–92. https://doi.org/10.1145/3568813.3600142

[10] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva, "Exploring the responses of large language models to beginner programmers' help requests," in Proc. 2023 ACM Conf. Int. Comput. Educ. Res. - Vol. 1, Aug. 2023, pp. 93–105. https://doi.org/10.1145/3568813.3600139

[11] S. Lau and P. Guo, "From 'Ban it till we understand it' to 'Resistance is futile': How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot," in Proc. 2023 ACM Conf. Int. Comput. Educ. Res. - Vol. 1, Aug. 2023, pp. 106–121. https://doi.org/10.1145/3568813.3600138.

[12] C. Bull and A. Kharrufa, "Generative artificial intelligence assistants in software development education: A vision for integrating generative artificial intelligence into educational practice, not instinctively defending against it," IEEE Softw., vol. 41, no. 2, pp. 52–59, 2024. https://doi.org/10.1109/MS.2023.3300574

[13] J. M. Perkel, "Six tips for better coding with ChatGPT," Nature, vol. 618, no. 7964, pp. 422–423, 2023. https://www.nature.com/articles/d41586-023-01833-0

[14] J. Su and W. Yang, "Unlocking the power of ChatGPT: A framework for applying generative AI in education," ECNU Rev. Educ., vol. 6, no. 3, pp. 355–366, 2023. https://doi.org/10.1177/20965311231168423.

[15] P. Prasad and A. Sane, "A self-regulated learning framework using generative AI and its application in CS educational intervention design," in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. V. 1, Mar. 2024, pp. 1070–1076. https://doi.org/10.1145/3626252.363082