World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Effective prompt engineering for generative AI in C++ programming tasks

Ramona Markoska [1] * and Aleksandar Markoski [2]

[1] *Department of Software Engineering and Information systems, Faculty of ICT, UKLO, Bitola, N. Macedonia.*
[2] *Department of Intelligent Systems, Faculty of ICT, UKLO, Bitola, N. Macedonia.*

World Journal of Advanced Research and Reviews, 2025, 25(02), 1390-1397

Publication history: Received on 26 December 2024; revised on 11 February 2025; accepted on 14 February 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.25.2.0516

Abstract

The rise of Generative AI, propelled by Large Language Models (LLMs), has opened new opportunities to streamline programming tasks across various domains. In C++ programming, renowned for its intricate syntax, memory management complexities, and performance-critical applications, Generative AI offers invaluable support for code generation, optimization, and debugging. However, the effectiveness and accuracy of these AI models rely heavily on the application of prompt engineering—a technique that involves crafting precise, contextually relevant queries to guide the AI's response.This paper delves into the methodology and best practices for effective prompt engineering within the context of a cloud-based C++ training ecosystem. Here, developers and students can leverage AI tools to enhance productivity and learning outcomes. By utilizing advanced AI models such as GPT-4 and Jdroid, integrated within JDoodle, the ecosystem offers an interactive platform for generating, analyzing, and refining C++ code in real time. The study emphasizes strategies for optimizing prompts, including specificity, task segmentation, and iterative refinement, to overcome common challenges in C++ programming. Furthermore, it evaluates the integration of prompt engineering techniques with the cloud C++ training ecosystem, highlighting the scalability and accessibility of this approach for educational purposes. The results demonstrate that well-structured prompts significantly improve the accuracy and relevance of AI-generated solutions, enabling users to tackle complex C++ problems with greater efficiency and reliability. This work lays the groundwork for advancing AI-driven programming methodologies and underscores the critical role of prompt engineering in maximizing the potential of Generative AI tools.

Keywords: Prompt Engineering; Generative AI; LLMs; Cloud training ecosystem; C++

1. Introduction

In recent years, Generative AI has gained substantial attention across various domains, particularly in programming and software development. By leveraging models such as Large Language Models (LLMs), Generative AI has increasingly been used to assist developers in writing code, automating repetitive tasks, and generating optimized solutions for complex problems [1]. The parallel development of Generative AI, LLMs, and Natural Language Processing (NLP) has been transformative, enabling these capabilities. NLP provides the foundational framework for teaching machines to understand, process, and generate human language. Over time, its evolution from rule-based systems and statistical models to deep learning techniques culminated in the development of transformer architectures, a pivotal innovation in AI.

Simultaneously, the emergence of the Transformer architecture revolutionized NLP and Generative AI by introducing the self-attention mechanism [1]. This mechanism allows models to evaluate the relationships between all tokens (words or characters) in a sequence simultaneously, rather than processing them sequentially. As a result, transformers can efficiently handle long-range dependencies, making them highly effective for complex language tasks. Furthermore,

* Corresponding author: Ramona Markoska

their scalability enables training on massive datasets, laying the groundwork for modern LLMs like GPT-3, BERT, and Codex, which now dominate the NLP landscape [1, 5].

LLMs, built on transformers, capitalize on their ability to understand context and generate coherent text. By training on extensive datasets, these models achieve remarkable performance across diverse applications, from text summarization and translation to programming assistance [2, 6]. The process of interacting with Generative AI tools to obtain meaningful results requires a well-structured approach known as Prompt Engineering.

Efficient prompt engineering can significantly enhance a model's performance by framing tasks in ways that align with its underlying capabilities, often utilizing specialized techniques such as fine-tuning [3, 4]. Prompt engineering involves designing and structuring input queries to guide AI models, such as GPTs, toward generating the most accurate and relevant outputs [3, 7]. In the realm of C++ programming, transformers have enabled LLMs to generate syntactically accurate and contextually appropriate code while also assisting in debugging and optimization tasks. These advancements are particularly beneficial in languages like C++, which is known for its complexity and wide range of applications, including system software, game development, and performance-critical systems [2].

The focus of this paper is to demonstrate how understanding and applying the nuances of prompt engineering is essential for achieving effective outcomes in C++ programming tasks.

2. Interdependence of Prompt Engineering and Generative AI

Prompt engineering is a critical aspect of interacting with AI models, especially large language models (LLMs) like GPT and Generative AI. It involves crafting effective and precise input prompts to achieve desired outputs from an AI system. Although they overlap in principle, it should be emphasized that LLMs (Large Language Models) are a type of generative AI focused on processing and generating human-like text, while generative AI encompasses a broader range of technologies designed to create various types of content, including text, images, music, and more [3,8].

2.1. Using AI Tools in C++ Programming: Practical Insights

Instructional activities in programming technologies and programming, by their practical nature, are considered essential and are supported by additional resources such as forums for experience exchange, educational and practice platforms, online repositories, and more. Over the past two years, AI-enabled tools in programming, particularly generative AI, have gained significant popularity. These tools have surpassed traditional educational resources in both scope and applicability to programming. Among all AI-powered tools, generative AI stands out as the dominant force. Surveys among student groups reveal that generative AI is perceived as highly trustworthy, interactive, and adaptable for personalized learning experiences, whereas integrated AI tools often offer limited customization options.

Practical experience highlights several key concerns in the interaction with generative AI that require attention and further action:

- High confidence in the resulting AI solutions and intent to use them without being logically evaluated
- Imprecision and carelessness in user request formulation.
- Lack of feedback on whether the problem's logic is accurately described when querying AI.
- No validation mechanisms to ensure the solution is feasible and correct.
- Unclear task objectives and lack of contextual guidance.
- Absence of code analysis and applied solutions to facilitate deep understanding and long-term retention of core programming principles.

In this context, eliminating inaccuracies and preventing future ones is achievable through the application of prompt engineering techniques.

2.2. Prompt Engineering and Generative AI- general recommendations

In the context of programming, prompt engineering serves as both a skill and a technique to optimize the use of AI tools for code generation, debugging, optimization, and more. Generating effective and precise prompts is essential for clearly defining programming tasks[9]. Well-structured prompts help in setting clear requirements, ensuring accurate and high-quality responses when using automated tools or coding assistance. The Table1 outlines best practices for constructing well-defined programming prompts. Each entry highlights a key principle, such as defining the problem, setting context, providing examples, requesting documentation, and optimizing solutions. Additionally, practical examples illustrate each step to enhance clarity and understanding.

Table 1 Prompt engineering in generative AI - step by step C++ examples

Step	Description	Example
Be specific	Clearly define the problem or task. Include details like language, framework, libraries, or constraints.	❌ "Write a sorting algorithm." ✅ "Write a C++ function to implement merge sort for a vector of integers."
Set the context	Provide background or relevant details, such as input/output requirements or dependencies.	"I am developing a command-line tool in C++. Create a function saveToFile that saves a vector of integers to a .txt file, with one number per line."
Use clear instructions	Break the task into steps and specify constraints. You can request comments, optimization, or specific standards.	"Generate a C++ function that calculates the factorial of a number using recursion. Add detailed comments for each step."
Request modular code	Ask for modular and reusable code, or add requests for test cases.	"Write a C++ class Matrix for basic matrix operations (addition, multiplication, transpose). Include a test program for a 2x2 and a 3x3 matrix."
Refine through iteration	If the output is not accurate, provide feedback and ask for improvement, such as optimization or clarification.	"The generated solution works, but it uses unnecessary global variables. Can you rewrite it to use local variables instead for better encapsulation?"
Provide examples	Include input-output examples to clarify expectations.	"Write a C++ function fibonacci(int n) that uses dynamic programming. Example: fibonacci(5) should return 5, and fibonacci(10) should return 55."
Specify constraints	Mention specific tools, platforms, or versions to avoid compatibility issues.	"Write a C++ program that uses the C++17 std::filesystem library to list all files in a directory."
Request explanations	Ask for an explanation of the code, logic, or specific decisions to better understand the output.	"Provide a C++ implementation of a linked list and explain how the insert and delete methods work."
Include error handling	Request code with error handling and edge case management.	"Write a C++ program that reads integers from a file and calculates their average. Include error handling for missing files and invalid data formats."
Request comments and documentation	Request well-documented code with comments to clarify complex sections.	"Generate a commented C++ implementation of Dijkstra's algorithm for finding the shortest path in a graph represented as an adjacency list."
Limit or expand scope	Specify the level of detail or abstraction (smaller or larger tasks).	"Create a detailed C++ project structure for a command-line tool that parses and analyzes log files, providing modularized components for input, parsing, and output."

The recommendations provided are general and not limited to C++, as their applicability extends to other contexts. Their order is also flexible. The implementation of each step depends on the specific example being analyzed. For beginners in programming, it is always advisable to start with a pre-existing example and seek an explanation of its functionality. Following that, exploring code optimization is beneficial, with a clear focus on specifying which tools to use to avoid overly streamlined or highly functional tools that may require further evaluation. Additionally, the following section explains the workflow and practical application of generative AI within an open-source software ecosystem designed for hands-on C++ learning. In the development phase of smart technologies, this ecosystem also integrates generative AI as part of the embedded compiler itself.

3. Experiential C++ Learning: Generative AI and Prompt Engineering in JDoodle ecosystem

The study of programming languages and technologies is most effective through shared, interactive theoretical and practical experiences, which enable both the practice and evaluation of the concepts learned. Over time, these concepts have consistently aligned with emerging trends in programming, as well as with the evolving IT knowledge and skills demanded by the industry. This evolution requires continuous changes in the available IT education resources, which are constantly transforming. One of the most notable and beneficial trends is the creation and growth of cloud software ecosystems, offering great opportunities for customization. These ecosystems allow for their integration into educational cloud-based open-source ecosystems, enabling content to be accessed and viewed from a single, integrated platform [10]. Simultaneously, the availability of free options for integration and implementation in education has a positively reinforcing effect, helping future IT professionals develop the habits necessary to understand, use, and recognize the power and importance of new technologies as essential skills for their careers.

Initially, these resources consisted of websites supported by forums for sharing experiences [11] . Over time, they developed into cloud-based software ecosystems, enriched with examples, exercises, theoretical questions, certified tests, and the ability to track individual progress [12].

By last year, the most technologically advanced cloud software ecosystems with integrated IDE environments began incorporating AI tools, and by the end of 2024, generative AI was also included [13]. The availability of generative AI in its full capacity, even in education-friendly, free-to-use versions, demonstrates its superiority. Generative AI is no longer a distant concept; it is now an efficient and practical reality for education. As mentioned earlier, in order for the use of AI tools, with an emphasis on generative AI, to be effective, it is necessary to think well and adhere to the principles of prompt engineering, at all levels of programming learning and in all programming activities. The following is an explanation of the working steps and programming activities in JDoodle's cloud ecosystem, which is supported by a generative AI tool, JDroid, through a concrete real-world example.

3.1. Integrating Generative AI into JDoodle IDE – A Practical Case Study

To illustrate AI-assisted programming, this study examines a C++ chess program where a logical error in diagonal movement was detected. Using effective prompt engineering, JDroid AI was prompted to process, demonstrating how well-structured prompts lead to improved AI-generated solutions.The rules for formulating certain requirements to generative AI are general, however each specific case requires functional planning, thinking, and strategy and is related to the user's intentions and level of knowledge. Therefore, successful use of generative AI requires prior knowledge of both the syntax and semantics of the programming language, an assessment of one's own level of knowledge, and mandatory, and most importantly, how to formulate questions to generative AI, at what stages to ask for help once or more times, and feedback knowledge to evaluate the resulting modification. Experience has shown that while generative AI consistently produces syntactically correct and executable C++ code, the real challenge often lies in the user's ability to accurately convey the intended logic. Misalignment between the user's request and the actual problem requirements can lead to logically incorrect or suboptimal solutions. Smarter education that includes generative AI requires more training and smarter users, and qualitatively articulated demands on generative AI. The following is a step-by-step analysis and review of the real-education example, according to the screenshots numeration from Figure.1:

3.1.1. Initial preparation

All fields that rely on the practical application of knowledge, skills, and abilities,(particularly programming and programming technologies) require a strong foundational level of theoretical preparation. Moreover, successful practices emerge from identifying and addressing gaps between theory and practice by continuously monitoring, adopting, and integrating new technologies, environments, tools, and concepts at a deeper theoretical level. Many IT companies recognize this need and, as part of their business strategy, invest in developing and training resources for both initial and, when necessary, more advanced theoretical preparation. In programming education, specifically in the case of C++, theory and practice cannot be strictly separated, as theoretical understanding is evaluated and reinforced through practical activities. Recently, JDoodle has introduced its own C++ resource, structured into three levels of complexity, organized thematically and functionally. At the initial stages of education, it is highly recommended to start by reviewing and analyzing C++ thematic resources.

Figure 1 Prompt engineering supported code development- JDoodle, generative AI

3.1.2. Generative AI chat via main JDoodle page

Dubbed JDroid chat, the front page features generative AI functionality, among a host of others. All of the standard features of generative AI chat are available on this functionality, including the ability to request more general informational content that is not specifically related to programming. Although it is possible to directly request a programming example and get a solution, it is recommended to use the JDroid AI tool that is integrated into JDoodle's IDE for such purposes. to follow is to use keywords from your title in first few sentences.

3.1.3. JDoodle IDE- working window

The classic work environment, with the standard and pre-existing options, allows you to open an existing project, or create a new one. At any stage of the work, it is possible to invoke JDroid AI, via prompt instructions: Generate a prompt, or ask a question to JDroid, however, it is necessary for effective use and optimal results, to first check that a state in the code that requires intervention can be resolved through the options provided as an integral part of standardized requirements, such as:

- Optimize code: Modify code for better performance.
- Debug code: Fix error and bug in your code.
- Explain code: Get detailed explanation of how your code works
- Inline comments: Improve code readability by adding comments

Core workload capabilities, including compilation, debugging, and execution, continue to be available as a core integral part of the environment.

3.1.4. A Working Case Study of Applying Prompt Engineering with Generative AI

In order to demonstrate the process of effective prompt engineering using generative AI at the IDE JDooodle level, screenshots 4, 5 and 6 of Figure 1 will be considered.

Initial analysis, which shows an executable screen of a previously successful program, which, for a given queen position in chess, gives all the possible positions that the queen can hit. The code is compiled and a graphical representation is obtained, which shows that one diagonal is missing. This indicates a logical error and the need for further correction. This correction cannot be requested and corrected through the standard options given at the JDroid AI level such as optimization, debugging, nor detected through a detailed explanation.

Effective prompt engineering in JDoodle's generative AI in this case aims to address, concisely clearly and in a few sequential clarifications, precisely the type of problem. The generative dynamics in the case is seen in the possibility, colloquially through prompt, to explain that the graphical representation lacks a single direction of diagonal motion, without writing or commenting on the code. When formulating prompt requirements, it is recommended to use the recommendations in Table 1. If necessary, it is possible to apply prompt engineering multiple times when fine-tuning a single code.

JDroid's generative AI component understands the request, comments on requested changes, analyzes the code, updates it backwards and clarifies the changes, and the last screenshot of the image shows that the positions of the previously missing diagonal have been taken into account.

Effective prompt engineering when making requests to generative AI gives the best results if the programmer/coder user has knowledge of the code they are working on and knows what expected results they should get. At the basic level, it is possible to use it to generate a solution from scratch, but here is the very real problem of not recognizing whether the code meets some of the more specific requirements defined by the problem to be encoded.

4. Results and discussion

In the study of programming technologies and the practice of coding, generative AI extends beyond the role of an innovative tool—it represents a new digital intelligent entity capable of analytically processing requests, understanding conversational contexts, and translating them into specific programming code. This positioning, particularly within software and educational ecosystems, establishes generative AI as a paradigm that offers only strengths and opportunities. Any potential concerns, such as threats or weaknesses, arise solely from how users interact with and apply the technology, as systematized in Table 2.

Table 2 SWOT analysis of AI driven prompt engineering

Strengths (S)	**Weaknesses (W)**
Enhances learning efficiency and accelerates code generation. Provides instant feedback and debugging assistance. Supports personalized learning and adapts to different skill levels. Encourages creativity and exploration in coding. Saves time on repetitive coding tasks, allowing more focus on logic and design.	Over-reliance on AI may hinder students' problem-solving skills. AI-generated code may contain inefficiencies or subtle errors. Students may struggle to understand underlying logic if they rely too much on AI. AI-generated explanations might lack depth or clarity for complex concepts.
Opportunities (O)	**Threats (T)**
Learned rules of application have a general significance as a concept Can be integrated into existing IDEs and educational platforms. Helps bridge the gap for beginners by providing structured assistance. Encourages interdisciplinary applications (e.g., AI + software development). Can facilitate automated assessment and feedback in programming courses. Supports lifelong learning and upskilling for professionals.	Risk of generating biased or insecure code. AI-generated code may not always follow best practices or standards. Dependence on proprietary AI models may create accessibility issues. Potential misuse by students to bypass learning objectives.

The notion that generative AI can independently build a fully integrated solution without the active involvement of programmers or coders—hereinafter referred to as users—is partially correct. This applies to the simplest initial tasks that can be requested solely through prompts. However, there is a risk of using tools or solutions that exceed the user's level of understanding.

The skill of seeking and utilizing AI assistance involves both analytical and programming knowledge, meaning the ability to assess results and iteratively guide generative AI as needed. This does not imply obtaining software solutions or ready-made code without prior knowledge, but rather requires a solid understanding of syntax and semantics, along with the ability to articulate the problem effectively in program-based conversational logic—an essential component of effective prompt engineering. When approached in this way, potential risks and weaknesses become negligible.

5. Conclusion

Generative AI, when effectively integrated into cloud-based programming ecosystems, significantly enhances learning and productivity. However, its full potential is realized only when combined with structured prompt engineering techniques. Well-structured prompts enable users to harness AI tools efficiently while maintaining control over code quality and logic. Furthermore, effective prompt engineering fosters a more analytical approach to coding, encouraging developers and learners to refine their problem-solving skills while leveraging AI assistance. As AI continues to evolve, refining prompt engineering strategies will be critical in maximizing the effectiveness of AI-assisted programming workflows. The study demonstrates that users who adopt structured and iterative prompting techniques achieve better coding outcomes, reducing errors and improving efficiency. This highlights the broader implications of AI-driven learning, where human intelligence and machine learning models collaborate to produce optimal results. Moving forward, expanding research into adaptive AI-driven programming tools and refining prompt engineering methodologies will be essential in shaping the future of software development and education. By embracing these innovations, the programming community can cultivate an ecosystem where AI serves as a complementary asset—enhancing, rather than replacing, human ingenuity in coding practices.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A, Kaiser Ł, Polosukhin I. Attention is all you need. arXiv. 2017. Available from: https://arxiv.org/abs/1706.03762

[2] Stroustrup B. The C++ programming language. 4th ed. Boston: Addison-Wesley; 2013.

[3] Brown T, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language models are few-shot learners. In: NeurIPS 2020. Available from: https://arxiv.org/abs/2005.14165

[4] Reynolds L, McDonell K. Prompt programming for large language models: Beyond the basics. In: Proceedings of the 1st Workshop on Large-Scale Generative Models. 2021. Available from: https://arxiv.org/abs/2109.09693

[5] Wolf T, Debut L, Sanh V, et al. Hugging Face's transformers: State-of-the-art NLP for everyone. In: Proceedings of EMNLP 2020. Available from: https://arxiv.org/abs/1910.03771

[6] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT 2019. Available from: https://arxiv.org/abs/1810.04805

[7] OpenAI. Best practices for prompt engineering with the OpenAI API. OpenAI Help Center. Available from: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api

[8] Radford A, Kim J, Hallacy C, Ramesh A, Goh G, Agarwal S, et al. Learning transferable visual models from natural language supervision. Proceedings of the International Conference on Machine Learning. 2021. Available from: https://arxiv.org/abs/2103.00020

[9] Ihantola P, et al. AI-Tutoring in Software Engineering Education. IEEE Transactions on Education, vol. 65, no. 3, 2022, pp. 345-356. Available from: https://ieeexplore.ieee.org/document/10554752

[10] Markoska R, "Managing ICT solutions for training and evaluation of C++ programming skills in e-learning ecosystem". New Trends and Issues Proceedings on Humanities and Social Sciences, 6(7), 33–41. 2019. Available from: https://doi.org/10.18844/prosoc.v6i7.4509

[11] Cplusplus.com. C++ Programming. Available from: https://cplusplus.com/ Accessed: 8 February 2025.

[12] W3Schools. C++ Tutorial. Available from: https://www.w3schools.com/ Accessed: 8 February 2025.

[13] JDoodle. Online C++ Compiler. Available from: https://www.jdoodle.com/ Accessed: 8 February 2025.