Journal of Information Systems and Informatics

Vol. 6, No. 3, September 2024 e-ISSN: **2656-4882** p-ISSN: **2656-5935**

DOI: 10.51519/journalisi.v6i3.783 Published By DRPM-UBD

An Experimental Study of The Efficacy of Prompting Strategies In Guiding ChatGPT for A Computer Programming Task

Nompilo Makhosi Mnguni[1], Nkululeko Nkomo[2], Kudakwashe Maguraushe[3], Murimo Bethel Mutanga[4]

[1,2,3,4]Department of Information and Communication Technology, Mangosuthu University of Technology, Umlazi, Durban, South Africa
Email: [1]mnguninompilo969@gmail.com, [2]nkomonkululeko021@gmail.com, [3]maguraushe.kuda@mut.ac.za, [4]mutangamb@mut.ac.za

Abstract

In the rapidly advancing artificial intelligence (AI) era, optimizing language models such as Chatbot Generative Pretrained Transformer (ChatGPT) for specialised tasks like computer programming remains a mystery. There are numerous inconsistencies in the quality and correctness of code generated by ChatGPT in programming. This study aims to analyse how the different prompting strategies; text-to-code and code-to-code, impact the output of ChatGPT's responses in programming tasks. The study adopted an experimental design that presented ChatGPT with a diverse set of programming tasks and prompts, spanning various programming languages, difficulty levels, and problem domains. The generated outputs were rigorously tested and evaluated for accuracy, latency, and qualitative aspects. The findings indicated that code-to-code prompting significantly improved accuracy, achieving a 93.55% success rate compared to 29.03% for text-to-code. Code-to-code prompts were particularly effective across all difficulty levels, while text-to-code struggled, especially with harder tasks. Based on these findings, computer programming students need to appreciate and comprehend that ChatGPT prompting is essential for getting the desired output. Using optimised prompting methods, students can achieve more accurate and efficient code generation, enhancing the quality of their code. Future research should explore the balance between prompt specificity and code efficiency, investigate additional prompting strategies, and develop best practices for prompt design to optimize the use of AI in software development.

Keywords: artificial intelligence; ChatGPT; prompting strategies; quality; latency; difficulty level

1. INTRODUCTION

Students generally perceive programming as one of the most challenging courses [1], [2]. Firstly, novice programmers without any exposure and prior understanding of programming concepts find it difficult [3], [4]. Although perceived as demanding, programming requires strong analytical and reasoning skills to understand its syntax, debug and write algorithms [5]–[7]. Not only does this challenge lie with the lack of students' ability to solve problems, but it also involves the ineffective use of teaching and learning material, including the effective use of technology [8], [9]. In extreme instances, some students even develop negativity about the programming course [10], creating a mental block towards effective learning. Given the plethora of challenges students face when learning programming, there is considerable interest in leveraging emerging technological developments [11]. In recent years, large-scale language models (LLMs) have garnered significant attention within the software development community for their potential to automate various aspects of programming tasks [12]. They exhibit several properties, including the ability to answer questions and generate text, which makes them powerful tools [13].

Interacting with these LLMs, such as Chatbot Generative Pretrained Transformer (ChatGPT), involves providing "prompts"—instructions used to provide context to the LLM and guide its generation of textual responses—presenting a promising avenue for aiding common software development and engineering tasks [14]. A prompt is a sentence or short paragraph that initiates a task or text to be completed by a language model through artificial intelligence (AI) technology [15]. Prompting first emerged in NLP tasks in 2018 using a single dataset [16] and was fine-tuned in 2021 with multiple datasets [17]. Text-to-code and code-to-code techniques were proposed in 2022 and continue to be fine-tuned to this day [18]. The text-to-code strategy involves providing natural language descriptions of programming problems, while the code-to-code strategy involves providing partial code snippets or hints to guide ChatGPT in generating code solutions [19].

Prompting strategies have emerged as crucial techniques influencing the performance of ChatGPT, particularly in programming-related tasks such as code generation, testing, and validation [20], [21]. They impact ChatGPT's performance, especially in programming tasks like code generation, testing, and validation. The importance of prompt design is underscored by several studies [15], [21]. These studies emphasize the substantial influence of prompt design on the performance of ChatGPT in code generation tasks, finding that carefully crafted prompts can significantly improve the quality of output [19]. To this end, [15] even advocated that teachers integrating AI language models like ChatGPT need to teach students "prompt literacy" skills. Similarly, research has explored the application of standard prompting techniques in an educational context and found that assigning specific roles to ChatGPT, such as "teacher" or "instructor," improved the relevance of the generated lesson plans[22], [23] Detailed instructions and seed words enhanced the clarity and focus of responses, resulting in more coherent, logically structured, and engaging content [22].

Further contributions to understanding prompt design come from studies that demonstrated the effectiveness of prompting strategies in enhancing ChatGPT's performance in natural language processing (NLP) tasks [24]. Their findings suggest that the principles applied in NLP could also benefit programming assistance. Additionally, an evaluation of ChatGPT's capabilities as a fully automated programming assistant focused on code generation, program repair, and code summarization [19]. It was found that while ChatGPT excels at generating accurate code for common tasks, it encounters difficulties with medium to hard problems and longer prompts. These findings suggest that ChatGPT's effectiveness is influenced by problem complexity and prompt design. Studies show that ChatGPT achieves a high accuracy rate when relying on its internal knowledge of health-related questions [25]–[27]. However, when prompted with external evidence, accuracy drops, indicating the impact of prompt knowledge [28]. It is important to note that despite GPT's impressive capabilities, it has the potential to generate believable yet erroneous data, causing a blending of reality and fiction [29].

Although numerous studies have presented significant insights and stressed the impact of prompt strategies on accuracy, particularly in health-related situations, there is little research focusing on prompt techniques in computer programming. This study seeks to fill this void by exploring the impact of different prompting strategies on the output of ChatGPT's responses in computer programming tasks. Therefore, this research seeks to answer the question: How do different prompting strategies influence the accuracy, difficulty and qualitative aspects of ChatGPT's code generation in computer programming tasks?

The rest of the paper is structured as follows: Section 2 outlines the methodology used, section 3 presents the findings and discussion, and the conclusions are given in section 4.

2. METHODS

The study aimed to explore the efficacy of ChatGPT in using different prompting strategies for programming-related tasks. This study adopted an experimental research design to investigate the impact of various prompting strategies on the performance and output of ChatGPT in programming tasks. The study employed two main prompting strategies: text-to-code and code-to-code. To ensure a diverse range of challenges, programming problems were sourced from HackerRank, a platform known for its wide range of challenges that vary in difficulty, making it an ideal source for a comprehensive assessment of ChatGPT's performance [30]. The research will be guided by the six-step research design in Figure 1.

Step A: The first step is to have a clear definition of the objectives. This study examines how different prompting strategies impact ChatGPT's accuracy and efficiency in programming tasks. The goals are to assess accuracy, handle difficulties, and qualitatively analyze generated code. The specific objectives are to evaluate the accuracy, handle difficulty levels, and conduct a qualitative analysis of the generated code.

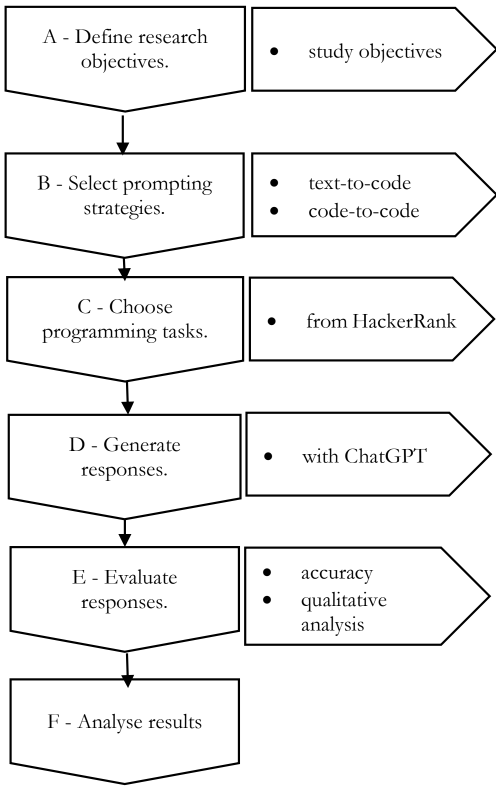

Figure 1: Study research design

Step B: Two prompting strategies were analysed for comparison. Both the text-to-code and code-to-code generation tasks were comprehensively evaluated for the effectiveness of different prompting strategies [19]. By doing so, the method enabled controlled manipulation of prompting strategies (independent variables) to measure their impact on the accuracy of generated code (dependent variables).

a) Text-to-code Prompting: Provide the problem statement in plain English and ask ChatGPT to generate the code.
b) Code-to-code Prompting: Presenting structured or partially completed code to ChatGPT for completion or correction.

Step C: For our experiments, we used a dataset from HackerRank, an online platform widely used for improving coding skills and preparing for technical interviews through a vast array of programming challenges. The dataset comprises problems categorized by difficulty levels—Easy, Medium, and Hard—covering topics such as algorithms, data structures, mathematics, artificial intelligence, databases, and functional programming. Each problem includes a detailed description, input and output formats, and sample test cases. Solutions are validated against sample and hidden test cases to ensure correctness (accuracy).

Step D: Create ChatGPT Responses
ChatGPT received text-to-code and code-to-code prompts for each task and gathered its responses for evaluation. ChatGPT was tasked with generating responses to the prompts for each programming problem obtained from HackerRank, following the specific prompting approach. The generated responses were then evaluated using test cases from the datasets to assess the accuracy. The responses were carefully recorded alongside the prompting strategy for analysis. Each produced response was tested against the labelled 1, 2, or 3 test cases from HackerRank, depending on the available tests on the dataset.

Step E: The generated code responses were analyzed according to two primary criteria i.e., accuracy and qualitative analysis:

a) Accuracy is measured by the degree of correctness displayed by the code during testing against the given sample test cases [31]. Every response was run to verify if it generated accurate outputs. We used the accuracy metric to evaluate the accuracy of ChatGPT's responses. Accuracy was calculated based on the performance of ChatGPT's generated solutions against the provided test cases. The Equstion 1 is for accuracy [32]:

$$Accuracy = \left(\frac{Number\ of\ Correct\ Test\ Cases}{Total\ number\ of\ Test\ cases}\right) X\ \ 100\% \tag{1}$$

b) Qualitative analysis involved an exhaustive review of the code to uncover recurring patterns, errors, and areas where the prompting technique had an impact on the quality of the code.

During qualitative analysis, the given code snippets were examined to identify ChatGPT trends. Based on the level of difficulty of the questions, it was determined which levels of ChatGPT yield the best results. It also determines whether the errors in the code snippets are common or uncommon. This determination provides insights into the qualitative aspects of ChatGPT's code generation capabilities, complementing the quantitative metrics of accuracy and latency.

Step F: The findings were evaluated to derive conclusions on the efficacy of each prompting strategy. The analysis encompassed a comparison of the accuracy rates and qualitative aspects of the responses generated through the use of text-to-code and code-to-code prompting strategies.

3. RESULTS AND DISCUSSION

The study aimed to explore the efficacy of ChatGPT in using different prompting strategies in programming-related tasks. The study found that the code-to-code prompting strategy significantly improved accuracy compared to the text-to-code strategy. The section below details the findings in terms of the efficacy of ChatGPT in using the two different prompting strategies, text-to-code and code-to-code, in terms of accuracy.

3.1 Accuracy

ChatGPT was given problem descriptions and was required to generate the code from scratch. The results were populated for both the text-to-code and code-to-code. Firstly, out of a total of 31 test cases, the text-to-code responses generated by ChatGPT succeeded in passing only nine test cases, yielding an accuracy rate of 29.03%. This relatively low accuracy rate can be attributed to several factors. Firstly, generating code from textual problem descriptions involves complex natural language understanding (NLU) tasks. The model must accurately interpret the requirements, constraints, and other key aspects of the problem statement. Any misunderstanding or ambiguity in the problem description can lead to incorrect or incomplete code. Secondly, text-to-code generation requires the model to comprehend the broader context and specifics of the problem. Large language models, while powerful, may struggle with understanding the full context and translating it into precise code logic. This process of converting natural language into structured code requires robust knowledge representation. While ChatGPT is trained on vast amounts of data, its training data may not always fully capture the intricacies of specific coding problems. Additionally, small errors in interpreting the problem description can propagate through the code generation process, leading to incorrect outputs. Debugging such code requires iterative refinement, which is challenging in a single interaction with the model. These factors combined contribute to the lower success rate observed in the text-to-code strategy.

In the code-to-code strategy, ChatGPT was provided with existing code snippets and was tasked with modifying or completing them to meet the requirements. In this regard, ChatGPT passed 29 out of the 31 test cases, resulting in a significantly higher accuracy rate of 93.55%. This high accuracy rate can be attributed to several factors. Firstly, with an existing code snippet, ChatGPT has a concrete starting point, reducing the complexity of the task as the model focuses on making specific changes rather than generating code from scratch. Moreover, the presence of existing code also reduces the likelihood of fundamental errors, as the model can leverage the structure and logic already present, making it easier to make accurate modifications. Furthermore, large language models are adept at recognizing patterns.

Comparatively, the model can identify and replicate coding patterns and styles from the existing snippet in code-to-code tasks, ensuring consistency and correctness. Modifying code is a more constrained task than generating code from scratch, allowing the model to focus on specific areas that need change and leading to more precise and effective solutions. Lastly, the model has likely been trained on numerous examples of code modifications, as these are common tasks in programming resources and repositories. This extensive exposure improves its capability in code-to-code tasks, contributing to the high success rate observed in this strategy as compared to text-to-code tasks.

3.2 Difficulty Level Analysis

Accuracy was also calculated for each difficulty level individually, as shown in Figure 2. The findings illustrate the differences in ChatGPT's responses between the text-to-code and code-to-code prompting strategies across various difficulty levels of programming tasks, codified as "easy", "medium" and "hard".

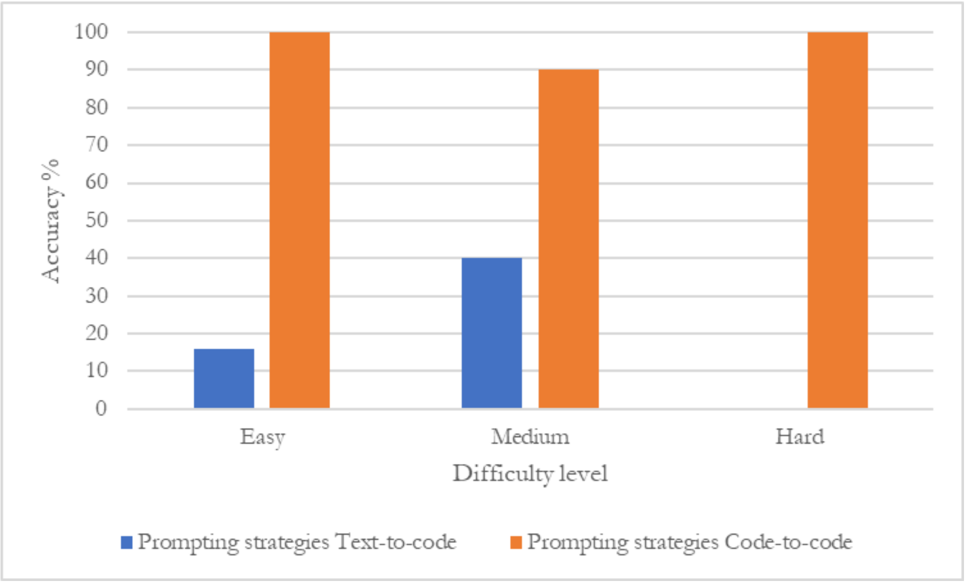

Figure 2. Results of the difficulty level

A detailed analysis of the difficulty levels was done for the efficacy of ChatGPT in using the two prompting strategies. Firstly, in the text-to-code responses, ChatGPT's code outputs were evaluated on a total of six test cases for easy tasks. Only one test case was successfully passed, resulting in an accuracy rate of 16.67%. This small percentage shows that ChatGPT had difficulty generating accurate answers when only provided with natural language problem descriptions.

Secondly, for medium tasks, ChatGPT's code outputs were evaluated on a total of 20 test cases. A total of eight test cases successfully passed, resulting in an accuracy rate of 40%. This demonstrates that ChatGPT moderately encountered difficulties in accurately producing solutions solely based on natural language descriptions. Lastly, for hard tasks, ChatGPT's code outputs were evaluated on a total of five test cases. However, none of the test cases passed, resulting in an accuracy rate of 0%. This highlights the major challenge ChatGPT faced while attempting to resolve intricate issues with only natural language explanations. The relatively low accuracy rates across all difficulty levels for the text-to-code strategy can be attributed to the challenges in natural language understanding, contextual comprehension, and error propagation. Generating code from textual descriptions requires accurate interpretation of problem requirements, which is inherently complex and prone to misunderstandings.

Code-to-code tasks were also evaluated. Firstly, ChatGPT's code outputs were calculated on a total of six test cases for easy tasks. All test cases passed, resulting in an accuracy rate of 100%. This indicates that giving ChatGPT partial code snippets or clues greatly aided in producing accurate solutions for easier problems. Secondly, ChatGPT's code outputs were evaluated on a total of 20 test cases for medium tasks. A total of 18 cases successfully passed, resulting in an accuracy rate of 90%. This continues to validate the usefulness of offering code suggestions or incomplete code samples, even when the tasks become more complicated. Lastly, for hard tasks, ChatGPT's code outputs were evaluated on a total of five test cases. All test cases successfully passed, resulting in an accuracy rate of 100% for hard tasks. This emphasizes the significant advantage of using partial code to direct ChatGPT, allowing it to address and navigate through difficult tasks effectively. The model's ability to leverage existing code structures can explain the high accuracy rates across all difficulty levels for the code-to-code strategy. Starting with partial code reduces complexity and the likelihood of fundamental errors, allowing the model to make accurate modifications. Existing patterns and structures significantly enhance the model's performance in producing correct solutions.

3.3 Qualitative Analysis

The study also evaluated the efficacy of ChatGPT in using different prompting strategies from a qualitative perspective. Some studies suggest that ChatGPT might return only partial code when addressing bugs or filling in missing parts, indicating a need for precise and comprehensive initial instructions to ensure the output meets the full requirements [33]. However, crafting input prompts thoughtfully enables researchers and developers to elicit more accurate, relevant, and useful responses from these models [34]. When given code-to-code prompts, ChatGPT sometimes returned only the portion of the code that needed fixing, contained a bug, or was missing. This often necessitated a follow-up prompt, such as "give me full code," to receive the complete code.

3.4 Discussion

The study results underscore significant variations in the efficacy of text-to-code and code-to-code prompting strategies in influencing ChatGPT's ability to generate programming code. The code-to-code prompting strategy consistently yielded higher precision and better-quality code in comparison to text-to-code prompting. These findings can be attributed to several factors.

The code-to-code prompting strategy furnished structured guidance by providing partial code snippets, which likely enhanced ChatGPT's interpretation and understanding of the programming task context. These findings are consistent with previous research, confirming that giving structured and specific prompts can improve the performance of AI models like ChatGPT by restricting ambiguity and directing the model's focus towards relevant aspects of the task [35], [36].

In the context of utilizing code-to-code prompting with ChatGPT for programming tasks, additional guidance may be necessary to achieve the desired comprehensive outcome, as suggested by [19]. This augments the desire to derive maximum value from adopting AI technologies like ChatGPT within academia, wherein a conceptual framework for guidance that espouses learning, research, and study was developed [37]. ChatGPT tends to focus on the specific instructions provided in the initial prompt, often prioritizing the delivery of an exact solution for the bug or missing segment. Consequently, it might return only partial code, addressing the issue directly without including the surrounding context or related code.

It can also be noted that various prompting strategies significantly impact the output of ChatGPT's responses in programming tasks. When provided with existing code snippets (code-to-code prompting), ChatGPT is more effective at generating accurate solutions compared to solely interpreting textual descriptions of problems (text-to-code prompting). However, code generated from text-to-code instructions compiles quicker than that from code-to-code scenarios, with ChatGPT's text-to-code compilation being faster than HackerRank's, while HackerRank's compilation is faster than code-to-code compilation. According to the Khan Academy [38], the code with the least runtime is typically the most efficient. Quality is ensured when the code does not produce errors, as a quick runtime for faulty code would not count as high quality. Thus, error-free code is essential for verifying code quality. Despite the advantages of code-to-code prompting in understanding and modifying code, it may require additional guidance to achieve the complete desired outcome. This research has observed that ChatGPT might return only partial code when addressing bugs or filling in missing parts, indicating a need for precise and comprehensive initial instructions to ensure the output meets the full requirements.

The qualitative examination of the generated code indicated that the use of code-to-code prompts resulted in a code that was more logically structured and efficient. This approach reduced the frequency of common errors, such as syntactical errors and logical contradictions, which are frequently more prevalent in responses generated by text-to-code prompts. The use of code-to-code prompts assists in reducing these issues by presenting a more explicit framework for the model to adhere to [39], [40].

The findings of this study have significant implications for the application of AI such as ChatGPT in both the fields of education and programming. Particularly in educational environments, giving students structured pieces of code as prompts can improve their learning experience by providing clearer direction and lowering the frustration that comes with vague instructions. Similarly, in professional coding environments, the use of code-to-code prompting can enhance the accuracy and efficacy of AI-assisted coding tools, potentially boosting productivity and reducing rates of errors [41], [42].

4. CONCLUSION

This study examined the impact of text-to-code and code-to-code prompting strategies on ChatGPT's performance in programming tasks. The findings indicated that code-to-code prompting significantly improved accuracy, achieving a 93.55% success rate compared to 29.03% for text-to-code. Code-to-code prompts were particularly effective across all difficulty levels, while text-to-code struggled, especially with harder tasks. Although text-to-code compilation was faster, code-to-code prompts yielded more accurate and comprehensive solutions, highlighting a trade-off between speed and accuracy. The study underscores the importance of prompt design in optimizing ChatGPT's utility for programming, suggesting that detailed and specific prompts enhance performance. This study presented evidence that code-to-code prompting significantly increases the accuracy and quality of ChatGPT's code generation as compared to text-to-code prompting, as illustrated by a higher rate of accuracy in test scenarios and the production of more logically structured, efficient code with fewer errors. This study contributes to understanding how different prompting strategies affect ChatGPT's performance in programming tasks. It highlights the importance of prompt design in achieving accurate and efficient code generation, providing valuable insights for optimizing the use of LLMs in software development.

Future research could further explore the balance between prompt specificity and code efficiency, investigate other prompting strategies, and assess their applicability across a broader range of programming challenges. Developing best practices for prompt design could also enhance the utility of LLMs like ChatGPT in various programming contexts. By shedding light on the effectiveness of different prompting strategies, this study provides a foundation for optimizing the use of ChatGPT and similar LLMs in programming assistance, contributing to more efficient and accurate software development processes.

REFERENCES

[1] J. Msane, M. B. Mutanga, and T. Chani, 'Students' Perception of the effect of cognitive factors in determining student's success in computer programming', *J. Theor. Appl. Inf. Technol.*, vol. 98, no. 17, pp. 3607–3618, 2020.

[2] M. Thuné and A. Eckerdal, 'Analysis of Students' learning of computer programming in a computer laboratory context', *Eur. J. Eng. Educ.*, vol. 44, no. 5, pp. 769–786, 2019, doi: 10.1080/03043797.2018.1544609.

[3] B. S. Javier, 'Understanding their Voices from Within: Difficulties and Code Comprehension of Life-Long Novice Programmers', *Int. J. Arts*, vol. 1, no. 1, pp. 53–76, 2021.

[4] M. B. Mutanga, P. X. Piyose, and L. S. Ndovela, 'Factors Affecting Career Preferences and Pathways : Insights from IT Students', *J. Inf. Syst. Informatics*, vol. 5, no. 3, pp. 1111–1122, 2023, doi: 10.51519/journalisi.v5i3.556.

[5] N. Islam, G. Shafi Sheikh, R. Fatima, and F. Alvi, 'A Study of Difficulties of Students in Learning Programming', *J. Educ. Soc. Sci.*, vol. 7, no. 2, pp. 38–46, 2019, doi: 10.20547/jess0721907203.

[6] E. Y. İnce, 'Students' Perceptions on Learning Programming with CodinGame', *Int. J. Technol. Teach. Learn.*, vol. 17, no. 1, pp. 38–46, 2021, doi: 10.37120/ijttl.2021.17.1.03.

[7] D. A. Egbe, B. M. Mutanga, and T. Chani, 'Combating Digital Academic Dishonesty: A Scoping Review of Approaches', no. April, 2021, doi: 10.35940/ijeat.F1268.089620.

[8] C. S. Cheah, 'Factors contributing to the difficulties in teaching and learning of computer programming: A literature review', *Contemp. Educ. Technol.*, vol. 12, no. 2, pp. 1–14, 2020, doi: 10.30935/cedtech/8247.

[9] M. B. Mutanga, 'The effect of cognitive factors in determining studens' success in computer programming', *J. Theor. Appl. Inf. Technol.*, vol. 98, no. 17, pp. 3607–3618, 2020.

[10] C.-H. ; Lai, Y.-K. ; Chen, and Y. Wang, 'Learning Computer Programming for Students with Medical Fields of', *Int. J. Environ. Reasearch Public Heal.*, vol. 19, pp. 1–17, 2022, doi: 10.3390/ijerph19106005

[11] W. Takerngsaksiri, C. Warusavitarne, C. Yaacoub, M. H. K. Hou, and C. Tantithamthavorn, 'Students' Perspective on AI Code Completion: Benefits and Challenges', *arXiv.org*, pp. 1–6, 2023.

[12] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, 'ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design', *arXiv:2303.07839*, pp. 1–35, 2023.

[13] R. Jagerman, H. Zhuang, Z. Qin, X. Wang, and M. Bendersky, *Query*

Expansion by Prompting Large Language Models, vol. 1, no. 1. Association for Computing Machinery, 2023.
[14] E. Kadir, T. Rahman, and S. Barman, 'Exploring the Competency of ChatGPT in Solving Competitive Programming Challenges', *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 13, no. 1, pp. 13–23, 2023, doi: 10.30534/ijatcse/2024/031312024.
[15] R. Yilmaz and F. G. Karaoglan 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, doi: 10.1016/j.chbah.2023.100005.
[16] B. McCann, N. S. Keskar, C. Xiong, and R. Socher, 'The Natural Language Decathlon: Multitask Learning as Question Answering', *arXiv*, pp. 1–23, 2018.
[17] V. Sanh *et al.*, 'Multitask Prompted Training Enables Zero-Shot Task Generalization', in *ICLR 2022 - 10th International Conference on Learning Representations*, 2022.
[18] S. H. Bach *et al.*, 'PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts', in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2022, pp. 93–104. doi: 10.18653/v1/2022.acl-demo.9.
[19] C. Liu *et al.*, 'Improving ChatGPT Prompt for Code Generation', *arXiv*, pp. 1–12, 2023.
[20] S. Wang and P. Jin, 'A Brief Summary of Prompting in Using GPT Models', *Qeios*, no. April, 2023, doi: 10.32388/IMZI2Q.
[21] T. Lehtinen, C. Koutcheme, and A. Hellas, *Let's Ask AI About Their Programs: Exploring ChatGPT's Answers To Program Comprehension Questions*, vol. 1, no. 1. Association for Computing Machinery, 2024. doi: 10.1145/3639474.3640058.
[22] A. J. Spasic and D. S. Jankovic, 'Using ChatGPT Standard Prompt Engineering Techniques in Lesson Preparation: Role, Instructions and Seed-Word Prompts', in *2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST 2023 - Proceedings*, IEEE, 2023, pp. 47–50. doi: 10.1109/ICEST58410.2023.10187269.
[23] G. van den Berg and E. du Plessis, 'ChatGPT and Generative AI: Possibilities for Its Contribution to Lesson Planning, Critical Thinking and Openness in Teacher Education', *Educ. Sci.*, vol. 13, no. 10, 2023, doi: 10.3390/educsci13100998.
[24] X. Sun *et al.*, 'Pushing the Limits of ChatGPT on NLP Tasks', *ArXiv*, pp. 1–26, 2023.
[25] A. Suárez, V. Díaz-Flores García, J. Algar, M. Gómez Sánchez, M. Llorente de Pedro, and Y. Freire, 'Unveiling the ChatGPT phenomenon: Evaluating the consistency and accuracy of endodontic question answers', *Int. Endod. J.*, vol. 57, no. 1, pp. 108–113, 2024, doi: 10.1111/iej.13985.

[26] K. Kusunose, S. Kashima, and M. Sata, 'Evaluation of the Accuracy of ChatGPT in Answering Clinical Questions on the Japanese Society of Hypertension Guidelines', *Circ. J.*, vol. 87, no. 7, pp. 1030–1033, 2023, doi: 10.1253/circj.CJ-23-0308.
[27] Y. Kaneda *et al.*, 'Evaluating ChatGPT's effectiveness and tendencies in Japanese internal medicine', *J. Eval. Clin. Pract.*, no. April, pp. 1–7, 2024, doi: 10.1111/jep.14011.
[28] G. Zuccon and B. Koopman, *Dr ChatGPT, tell me what I want to hear: How prompt knowledge impacts health answer correctness*, vol. 1, no. 1. Association for Computing Machinery, 2023.
[29] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, 'On the dangers of stochastic parrots: Can language models be too big?', in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623. doi: 10.1145/3442188.3445922.
[30] HackerRank, 'HackerRank', 2024. https://www.hackerrank.com/ (accessed Jun. 26, 2024).
[31] K. Buffardi and P. Valdivia, 'Measuring Unit Test Accuracy', in *50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 578–584. doi: 10.1145/3287324.3287351.
[32] Tricentis, '64 essential testing metrics for measuring quality assurance success.', 2016. https://www.tricentis.com/blog/64-essential-testing-metrics-for-measuring-quality-assurance-success (accessed Jun. 04, 2024).
[33] J. Cao, M. Li, M. Wen, and S. Cheung, *A study on Prompt Design, Advantages and Limitations of ChatGPT for Deep Learning Program Repair*, vol. 1, no. 1. Association for Computing Machinery, 2023.
[34] N. Mungoli, 'Exploring the Synergy of Prompt Engineering and Reinforcement Learning for Enhanced Control and Responsiveness in Chat GPT', *J. Electr. Electron. Eng.*, vol. 2, no. 3, pp. 201–205, 2023, doi: 10.33140/jeee.02.03.02.
[35] T. B. Brown *et al.*, 'Language Models are Few-Shot Learners', no. NeurIPS, 2020.
[36] X. Liu *et al.*, 'GPT Understands, Too', *AI Open*, 2023, doi: 10.1016/j.aiopen.2023.08.012.
[37] S. Dube *et al.*, *Students' Perceptions of ChatGPT in Education: A Rapid Systematic Literature Review*, vol. 4, no. 1. Springer Nature, 2024. doi: 10.3390/proceedings47010031.
[38] Khan Academy, 'Categorizing run time efficiency', 2024. https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/evaluating-algorithms/a/comparing-run-time-efficiency (accessed Jun. 26, 2024).
[39] S. Sivarajkumar, M. Kelley, A. Samolyk-mazzanti, S. Visweswaran, and Y. Wang, 'An Empirical Evaluation of Prompting Strategies for Large Language Models in Zero-Shot Clinical Natural Language Processing', *EJMIR Med. Informatics*, vol. 12, 2023, doi: 10.2196/55318.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

[40] M. Chen *et al.*, 'Evaluating Large Language Models Trained on Code', *arXiv Prepr.*, 2021.

[41] R. Yilmaz, F. Gizem, and 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, pp. 1–14, 2023, doi: 10.1016/j.caeai.2023.100147.

[42] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, 'The Impact of AI on Developer Productivity : Evidence from GitHub Copilot arXiv : 2302 . 06590v1 [cs . SE] 13 Feb 2023', *arXiv Prepr.*, pp. 1–19, 2023.