Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/131/2024.20795

GPThelper: A Prompt that You Can Discover LLM's Full Potential

Peixin Han
Nanjing University of Posts and Telecommunications

1679802132@qq.com

Abstract. It is widely acknowledged that LLMs(Large Language Models)assist us in our daily lives and it both save our times and energy,but it still fails in some specific area like DSL(Domain Specific Language). In response to this, the GPThelper was invented, which includes basic knowledge of the language, concepts, and commands used by the language, as well as tips for improvement to correct mistakes.With the Brainf language, issues were first identified in the program calculator regarding the location of the pointer, and prompts were improved to correct this. In the second experiment, errors occurred during the final section where parameters were to be added together, and these were subsequently corrected.In the evaluation section, various LLMs, including Kimi and Doubao, were tested using the GPThelper to handle tasks involving DSLs, and the results were excellent.It was found that this model works well with most LLMs, although there are still specific occasions where it fails.

Keywords: DSL,LLMs,Prompts

1. Introduction

Large Language Models (LLMs) have become an integral part of various fields today, as they possess the ability to assist in a wide range of tasks, including but not limited to, coding, data analysis, and even academic paper writing.There are already papers demonstrating applications in tasks such as code completion and translation[1]. Research on the potential of LLMs for programming language generation and understanding has also shown that LLMs have a great advantage in dealing with programming language problems.[2]The versatility and adaptability of LLMs have thus cemented their role as a critical asset in both professional and educational settings.

Nevertheless, despite their widespread application and usefulness, LLMs encounter significant challenges in certain specialized areas where user requests may not be fully comprehended, leading to inaccurate or incomplete responses.For example, this paper discusses the limitations of large language models in terms of code generation, including specific challenges when dealing with DSLs, such as syntax complexity and context understanding[3]. These shortcomings can have a considerable impact on users, particularly in niche fields, where precise and contextually relevant information is crucial for effective learning and work performance. Studies on the effectiveness of large language models in domain-specific code generation have shown the limitations of LLMs in dealing with this type of problem[4].The limitations arise primarily due to the insufficient availability of training data and the lack of user feedback in these specific areas, which hinders the LLMs' ability to learn and adapt to less commonly used languages or specialized terminologies.

DOI: 10.54254/2755-2721/131/2024.20795

```
+++ > +++++ [ < + > - ] ++++ ++++ [ < +++
+++ > - ] < .
```

Figure 1: Input of Brainf code

Figure 2: Output of ChatGPT

To address challenges in interpreting minority languages and specialized jargon, GPThelper was developed as a supplementary tool to enhance the performance of large language models (LLMs). By providing essential examples, guidance, and foundational knowledge about these languages—including key commands and concepts—GPThelper significantly reduces errors and improves the accuracy of LLM responses. Its introduction not only mitigates understanding difficulties but also enhances practical usability, leading to increased efficiency in users' work and learning activities. This improvement highlights the importance of tailored tools like GPThelper in expanding LLM capabilities to meet the diverse needs of users across specialized domains.

2. Related Work

The closely related prior work is the following.

A similar model is proposed in Using Grammar Masking to Ensure Syntactic Validity in LLM-based Modeling Tasks [5] .It demonstrates its own way to improve and show that frameworks that enable constrained decoding enable smaller, less performant LLMs to produce syntactically correct models at a reasonable rate.This part is highly related to my work at using GPThelper to improve the accuracy and readability.However,although their research goal is to develop a domain-independent DSL-specific modeling approach that is not optimized for a specific use case or domain,but this model has limitations and requires pretraining.In this case the GPThelper saves more time and cost less computing power.

An advanced idea was raised in A Comparative Study of DSL Code Generation: Fine-Tuning vs.Optimized Retrieval Augmentation [6].This paper discuss about two types of grounding information for RAG based DSL generation. they defined 3 key metrics to focus on code generation quality as well as syntactic accuracy and hallucination rate.they have a compiler to test the syntax and validate the functions against a database of API names as well as parameter keys. It is also interesting to note that this benefit does not transfer to hallucinated API names and their parameters keys where the fine-tuned model holds the advantage.The model has been built still need to be improved.

3. Example

So in the previous experiment,we already know that ChatGPT can handle the coding tasks perfectly and even for some DSLs like GDscript or haskell,but we do find a language it can not understand properly which is Brainf. Brainfuck is an archetypically minimalist language, providing merely eight commands[7].Researchers have conducted studies on the design and implementation of a 256-core Brainfuck computer[8].This example find out how ChatGPT react to such a minority language and how does GPT perform under the direction of my prompts. Dialogue showed as Figure 1.

```
+++  > +++++  [ < + > - ] ++++ ++++ [ < +++ +++ > - ] < .
```

This is a kind of language called brainfuck, the meaning of the character lists below, try to find out the right output. > Increment the data pointer by one (to point to the next cell to the right). < Decrement the data pointer by one (to point to the next cell to the left). + Increment the byte at the data pointer by one. - Decrement the byte at the data pointer by one. . Output the byte at the data pointer. , Accept one byte of input, storing its value in the byte at the data pointer. [If the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching command.] If the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching command. (Tip: After the loop ends, the pointer will stay at the cell position that determines whether the loop continues.)

Figure 3: Requests with prompts

Copy code

```
H
```

Figure 4: Output of ChatGPT

ChatGPT encountered significant difficulties in accurately interpreting the instructions' meaning (Figure 2) and in fully understanding the functional mechanics of the language itself. Detailed analysis revealed that the model unexpectedly altered the pointer's position after the loop, which was not intended or correct. To address this, a more detailed and explicit prompt could be developed, explaining the role of each character in the sequence and specifying the intended pointer location to guide the model effectively. This clarification is likely to enhance the model's performance, as demonstrated in Figure 3.

Unexpectedly, ChatGPT still struggled to fully grasp the intended concept, as seen in Figure 4, often overlooking or misinterpreting the critical instruction about the pointer's position. This highlights a major challenge with niche languages: ChatGPT's training likely lacks sufficient exposure to such specialized content. Although the prompt provided clear instructions, the model still struggled to execute the task correctly, revealing its limited understanding of domain-specific languages (DSLs). This limitation arises from its superficial comprehension and the scarcity of substantial training data for these languages. As

DOI: 10.54254/2755-2721/131/2024.20795

a result, even explicit guidance can lead to persistent errors due to the model's restricted foundational knowledge. This emphasizes GPT's limited adaptability to niche languages, mainly due to insufficient, domain-specific training data and the lack of extensive user feedback in these areas.

4. Background

4.1. LLM

A large language model (LLM) is a sophisticated language model using an artificial neural network with billions or trillions of parameters. Trained on extensive unlabeled text data through self-supervised or semi-supervised learning, LLMs can generate and understand natural language with high sophistication. The two important models and methods used in large language models are BERT[9] (Bidirectional Encoder Representations from Transformers) and GPT[10](Generative Pre-trained Transformer).Since their emergence around 2018, large language models have demonstrated exceptional performance across a wide range of tasks, including natural language processing, text generation, translation, and more. Their ability to generalize knowledge from diverse datasets has made them invaluable tools in both research and practical applications. Among the most well-known and widely used LLMs today are ChatGPT, Kimi.ai, and Doubao, each of which has garnered significant attention for their capabilities in various domains, from everyday conversational assistance to specialized tasks in different industries. The rapid evolution and deployment of these models continue to shape the landscape of artificial intelligence and its applications.Further information on website LLM.

4.2. DSL

A domain-specific language (DSL) is tailored for a specific application domain, unlike general-purpose languages (GPLs), which are versatile across various fields. An example of a DSL is HTML, designed for web development. Some DSLs are restricted to particular software, such as the code in MUSH software. DSLs can be categorized by purpose, including markup languages, modeling languages, and programming languages. While the idea of specialized languages has existed since computers began, the term "domain-specific language" gained traction with the rise of domain-specific modeling. Simple DSLs for single applications are often called mini-languages. Further information on DSL or Domain-specific languages[11] written by Martin Fowler.

4.3. Interpreter

An interpreter is a program that reads and executes instructions from an interpreted language, acting as a "middleman" between source code and the machine. It processes code line by line, resulting in slower execution compared to compiled programs. However, interpreters allow code execution without recompilation after updates, simplifying the development process. In contrast, a compiler processes the entire source code at once, converting it into a standalone binary file that can be executed independently without further interpretation.Further information on interpreterand Dynamic interpretation for dynamic scripting languages[12].

5. Prompt Design

the figure 5 shows the procedure of designing the GPThelper

5.1. Design of the GPThelper

Our research begins when a line of Brainf code was input to the llms to see how llms response.Although most LLMs like ChatGPT or kimi knows what kind of language it gets,sitll,there are some LLMS like doubao that can not recognize the category of the language.The figures 6 shows a example that a LLM can not define the type of the language.

In this context, a clear and effective prompt is essential for the language model (LLM) to accurately identify the specified programming language and its domain. The first component of GPThelper includes a brief introduction that provides foundational knowledge about the language's functional scope and characteristics. This aids the LLM in recognizing the language and optimizing response time, particularly for familiar models.

However, even with this introduction, LLMs may struggle to fully comprehend the code. Identifying the language is just the first step; the real challenge lies in accurately interpreting and processing the code. As shown in Figure 7, LLMs can still have difficulty parsing its nuances. This highlights the need for further refinement of prompts or additional support mechanisms to enhance the LLMs' code comprehension abilities.

Upon evaluating the performance of the LLMs(figure 7), it became evident that the results were suboptimal. While the models were able to accurately identify the specific programming language being used, they struggled with properly utilizing the language or responding appropriately to the given tasks. This deficiency likely stems from insufficient pre-training in the specific language or a lack of relevant commands and examples in the training data. To address this, the second component of GPThelper, "language design," offers detailed information about the language's commands and usage patterns. This aims to enhance the LLM's understanding and operational capabilities. Code lines were then input alongside the brief introduction and language design, with results showing the positive impact of these enhancements on performance.

In the version shown in Figure 8, we added a language design that helps LLMs fully understand the functions and commands of Brainf.

However, as shown in Figure 9, the result was incorrect: rather than producing the correct output of 8, it returned 'H'. Analyzing the process reveals where ChatGPT's response faltered.

The red part labeled in Figure 10 highlights where the model went wrong, indicating a failure to understand how the pointer moves during the loop. To address this, we propose "tips for improvement" to correct the LLM's errors and ensure adherence to rules during analysis. Experiments revealed that ChatGPT can be confused by these tips; for instance, it may only update the pointer's location before the loop, forgetting to do so afterward. Thus, it's crucial that these suggestions are comprehensive and applicable to all scenarios. Providing a related example can also enhance the model's task handling. Below is the final version of GPThelper and its impact on LLM responses.

Figure 11 presents the final version of GPThelper. A request formatted according to the GPThelper guidelines was submitted to ChatGPT to evaluate its response capabilities in this particular context. This assessment aims to determine how effectively the model leverages the enhancements provided by GPThelper to interpret and execute the given instructions.

According to figure 12,this time ChatGPT output the right answer and none of its steps went wrong.In the next part we will discuss about how is its performs improved, does this take more time or less time, and most importantly, how much progress did it make in improving the results' Correctness and Accuracy.

5.2. Technique core

To address the challenge of extracting relevant information from dynamic responses generated by large language models (LLMs) and crafting tailored prompts, initial experiments showed the LLM did not assimilate input information as effectively as anticipated. Despite clearly defined requirements, the LLM often failed to deliver satisfactory results. Consequently, GPThelper modules were developed incrementally to address scenarios where the LLM's superficial understanding of Domain-Specific Language (DSL) limited its comprehension of inputs. Providing detailed explanations, supplemented with examples, proved essential to enabling the LLM to produce accurate, coherent responses.

6. Evaluation

Although there is interpreters that can output the right answer and tells if the LLMs have output the right answer,but the key point is to see how much progress it make.So the specific two dialogue (one in calculation and one in words printing) will be analyzed in details.Some specific results should be taken into account according to evaluating Large Language Models Trained on Code[13]. Although there have

DOI: 10.54254/2755-2721/131/2024.20795

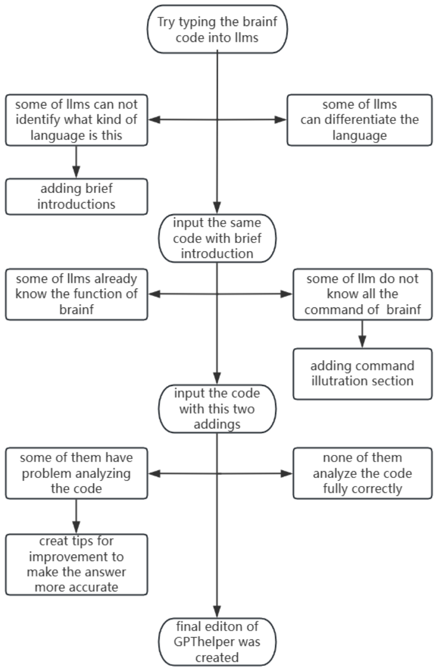


Figure 5: Flow chart of design

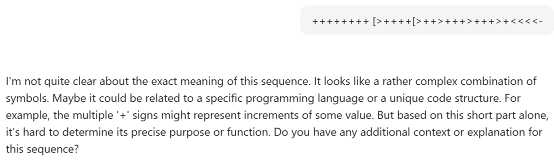


Figure 6: LLM fail to define the language

Figure 7: LLM can not read the code

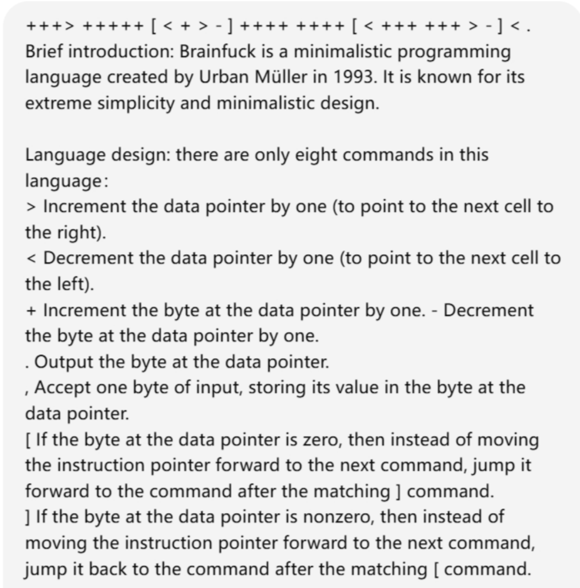


Figure 8: Design without tips

Figure 9: Result of ChatGPT

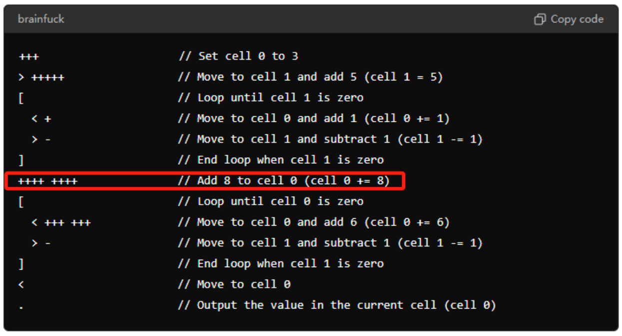


Figure 10: Error section

been a lot of researches such as An Evaluation on ChatGPT's Effectiveness in an undergraduate Java Programming Course[14]or a comparative analysis of popular large language models in coding[15],but few of them compare various large language models in parallel.In this section several kinds of LLMs will be analyzed and more elements will be taken into account. With the adding of the GPThelper,how much progress it make?Did the output be more accurate? Did the illustration be more clear?

Two lines of Brainf code, using GPThelper, were input into various LLMs, including ChatGPT and Doubao, with results shown below. Judging success based solely on output correctness overlooks the progress these LLMs make, as outputs remain incorrect. However, comparing these results to prior attempts without GPThelper allows us to observe measurable progress, increased efficiency, and the number of correct steps achieved. Comparative experiments were also conducted across different LLMs to evaluate GPThelper's effectiveness in enhancing each model's performance.

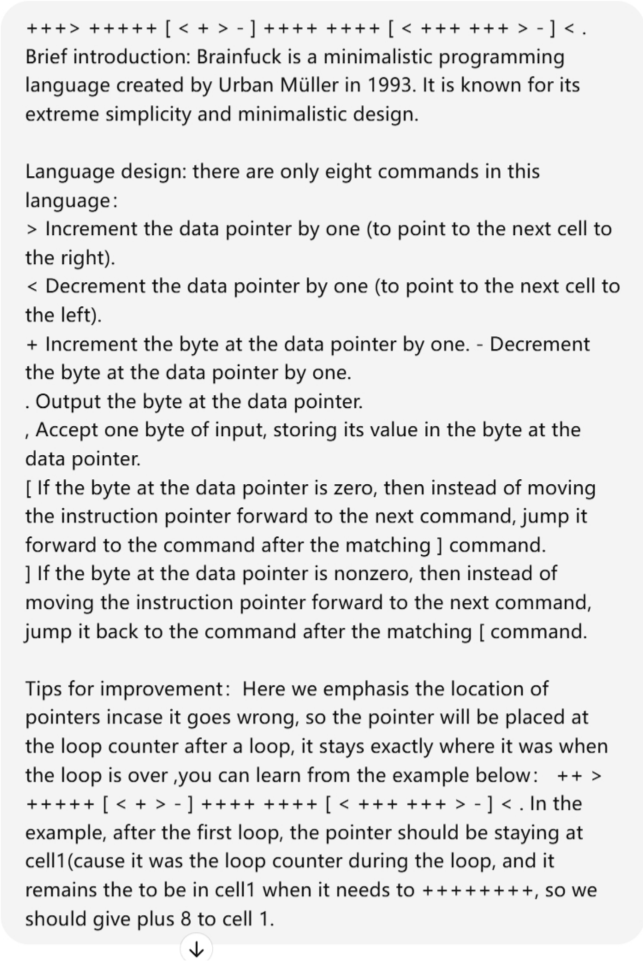
Test apps: ChatGPT, Kimi.ai and doubao

Brainf interpreter: EI brainfuck

6.1. Did LLMs answer more correct and accurate?

To assess the accuracy of various language models, we used the number of correctly interpreted code lines as a primary metric, posing the same coding question to each LLM for comparison. Figure 13

DOI: 10.54254/2755-2721/131/2024.20795

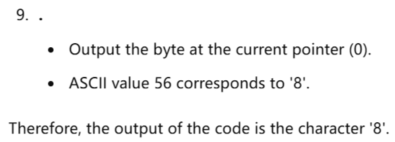


+++> +++++ [< + > -] ++++ ++++ [< +++ +++ > -] < .
Brief introduction: Brainfuck is a minimalistic programming language created by Urban Müller in 1993. It is known for its extreme simplicity and minimalistic design.

Language design: there are only eight commands in this language:
> Increment the data pointer by one (to point to the next cell to the right).
< Decrement the data pointer by one (to point to the next cell to the left).
+ Increment the byte at the data pointer by one. - Decrement the byte at the data pointer by one.
. Output the byte at the data pointer.
, Accept one byte of input, storing its value in the byte at the data pointer.
[If the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command.
] If the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command.

Tips for improvement: Here we emphasis the location of pointers incase it goes wrong, so the pointer will be placed at the loop counter after a loop, it stays exactly where it was when the loop is over ,you can learn from the example below: ++ > +++++ [< + > -] ++++ ++++ [< +++ +++ > -] < . In the example, after the first loop, the pointer should be staying at cell1(cause it was the loop counter during the loop, and it remains the to be in cell1 when it needs to ++++++++, so we should give plus 8 to cell 1.

Figure 11: Final version of GPThelper



9. .
- Output the byte at the current pointer (0).
- ASCII value 56 corresponds to '8'.

Therefore, the output of the code is the character '8'.

Figure 12: Output of ChatGPT

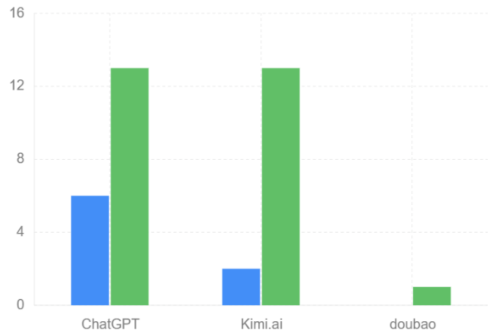


Figure 13: LLM's correctness evaluation

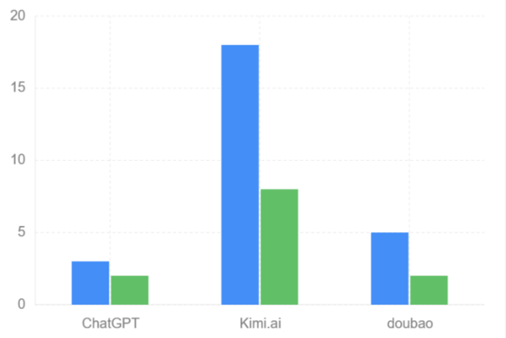


Figure 14: LLM's readability evaluation

illustrates the results, showing each model's effectiveness with and without GPThelper. Blue bars represent performance without GPThelper, while green bars indicate improvement with its support, highlighting the tool's positive impact on interpretation accuracy. Notably, ChatGPT and Kimi.ai analyzed all 13 lines accurately with GPThelper, with Kimi.ai's accuracy rising from 2 to 13 lines—a sevenfold increase, the highest among the tested LLMs. Though Doubao still misinterpreted the second line, it showed marked progress in Brainf language understanding with GPThelper.

6.2. Did llms illustrate the steps clearly?

When we are using LLMs we always find it generate a lot of unnecessary information before we get the final result,this is because LLMs need to read system prompts before it generate the key part of the answer.In this case,we compared both with or without the GPThelper to see how many useless lines was output before it generate the code analyzing part.According to the bar chart(Figure 14),it was clear that only ChatGPT benefit a little from the GPThelper,while Kimi.ai and doubao all have a at least 50 percent useless lines reduction.This could highly improve user's experience as it takes less time to generate and the answer is even more clear and readable.

7. Conclusion

This study focuses on the performance of large language models (LLMs) in the domain-specific language (DSL) field and explores strategies to enhance their effectiveness while minimizing computational resources and time. By employing GPThelper, LLMs can analyze the input language type more efficiently, which reduces unnecessary outputs and significantly enhances user experience by saving time and

DOI: 10.54254/2755-2721/131/2024.20795

improving overall efficiency. Furthermore, GPThelper has been shown to markedly increase the accuracy of responses across various LLMs, thereby ensuring that LLMs provide more precise answers to DSL-related queries. These findings underscore the potential of GPThelper to optimize LLM performance, highlighting its scientific significance in advancing practical applications in the DSL domain.

References

[1] Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, Qin B, Liu T, Jiang D *et al.* 2020 *arXiv preprint arXiv:2002.08155*

[2] Brown T B 2020 *arXiv preprint arXiv:2005.14165*

[3] Hadi M U, Al Tashi Q, Shah A, Qureshi R, Muneer A, Irfan M, Zafar A, Shaikh M B, Akhtar N, Wu J *et al.* 2024 *Authorea Preprints*

[4] Gu X, Chen M, Lin Y, Hu Y, Zhang H, Wan C, Wei Z, Xu Y and Wang J 2024 *ACM Transactions on Software Engineering and Methodology*

[5] Netz L, Reimar J and Rumpe B 2024 *arXiv preprint arXiv:2407.06146*

[6] Bassamzadeh N and Methani C 2024 *arXiv preprint arXiv:2407.02742*

[7] Mateas M 2008 *Software Studies\A lexicon. London: MIT* 267–276

[8] Jun S W 2016 *Computer Science and AI Laboratory, MIT*

[9] Devlin J 2018 *arXiv preprint arXiv:1810.04805*

[10] Yenduri G, Ramalingam M, Selvi G C, Supriya Y, Srivastava G, Maddikunta P K R, Raj G D, Jhaveri R H, Prabadevi B, Wang W *et al.* 2024 *IEEE Access*

[11] Fowler M 2010 *Domain-specific languages* (Pearson Education)

[12] Williams K, McCandless J and Gregg D 2010 Dynamic interpretation for dynamic scripting languages *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization* pp 278–287

[13] Chen M, Tworek J, Jun H, Yuan Q, Pinto H P D O, Kaplan J, Edwards H, Burda Y, Joseph N, Brockman G *et al.* 2021 *arXiv preprint arXiv:2107.03374*

[14] Ouh E L, Gan B K S, Jin Shim K and Wlodkowski S 2023 Chatgpt, can you generate solutions for my coding exercises? an evaluation on its effectiveness in an undergraduate java programming course. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* pp 54–60

[15] Coello C E A, Alimam M N and Kouatly R 2024 *Digital* **4** 114–125