Machine Learning with Applications 15 (2024) 100526

Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

Programming with ChatGPT: How far can we go?

Alessio Bucaioni [a], Hampus Ekedahl [a], Vilma Helander [a], Phuong T. Nguyen [b,*]

[a] Mälardalen University, Västerås, Sweden
[b] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila L'Aquila, Italy

ARTICLE INFO

Keywords:
ChatGPT
Large language models
Programming

ABSTRACT

Artificial intelligence (AI) has made remarkable strides, giving rise to the development of large language models such as ChatGPT. The chatbot has garnered significant attention from academia, industry, and the general public, marking the beginning of a new era in AI applications. This work explores how well ChatGPT can write source code. To this end, we performed a series of experiments to assess the extent to which ChatGPT is capable of solving general programming problems. Our objective is to assess ChatGPT's capabilities in two different programming languages, namely C++ and Java, by providing it with a set of programming problem, encompassing various types and difficulty levels. We focus on evaluating ChatGPT's performance in terms of code correctness, run-time efficiency, and memory usage. The experimental results show that, while ChatGPT is good at solving easy and medium programming problems written in C++ and Java, it encounters some difficulties with more complicated tasks in the two languages. Compared to code written by humans, the one generated by ChatGPT is of lower quality, with respect to runtime and memory usage.

1. Introduction

Artificial Intelligence (AI) is a rapidly advancing field that gains momentum in several domains, including finance, healthcare, entertainment, transportation (Littman et al., 2021), to name but a few. Notably, AI has shown remarkable advancements in medical diagnosis, speech recognition, game playing, and autonomous vehicles (Littman et al., 2021). Natural Language Processing (NLP) is a sub-field of AI (Aker et al., 2019), playing a pivotal role in the advancement of various applications, including machine translation (Arnold et al., 1994). Among these applications, chatbots have witnessed a surge in popularity and find utility in several domains, including customer service, financial applications and mental health counseling (Arnold et al., 1994; Fuscaldo, 2023). Code generation is one area of significant interest in chatbots, as they have the potential to generate code based on natural language inputs.

ChatGPT,[1] a chatbot developed by OpenAI, sparked significant attention following its release in November 2022, primarily due to its advanced natural language processing capabilities and proficiency in code generation (OpenAI, 2022). The potential of ChatGPT to enhance the software development process holds profound implications for the future of software engineering and programming roles, particularly considering the ongoing growth in demand for skilled programmers (Marr, 2023; Phillips, 2022). Nevertheless, concerns remain regarding the reliability of chatbots in code generation, necessitating further investigation into the ethical implications associated with their use (Müller, 2021). Under these circumstances, it is crucial to delve into the exploration and evaluation of chatbots' capabilities in code generation.

The main objective of this work is to investigate the ability of ChatGPT in writing code, compared to human programmers. We conducted a series of experiments where ChatGPT was prompted with a set of 240 programming problems curated from the well-known coding website LeetCode.[2] In our experiments, we targeted two different programming languages, i.e., C++ and Java, and used publicly available data provided by LeetCode, which serves as a reliable benchmark for assessing the accuracy and efficiency of the generated code. The set of programming problems was selected so as to encompass various type of programming tasks as well as difficulty levels. The code generated by ChatGPT was submitted to LeetCode, and the results were compared with those performed by human programmers. To assess the accuracy, we measured the number of attempts required to successfully complete the given programming problems. We studied the efficiency by considering both runtime (in milliseconds), and memory usage (in megabytes).

Through the experiments, we see that ChatGPT exhibits proficiency in solving programming problems at lower and medium difficulty

* Corresponding author.
E-mail addresses: alessio.bucaioni@mdu.se (A. Bucaioni), phuong.nguyen@univaq.it (P.T. Nguyen).

[1] https://chat.openai.com/chat
[2] https://leetcode.com/

https://doi.org/10.1016/j.mlwa.2024.100526
Received 15 July 2023; Received in revised form 27 December 2023; Accepted 6 January 2024
Available online 8 January 2024

levels. However, its accuracy in generating correct code decreases when being prompted with more challenging problems. Similarly, concerning runtime performance and memory usage, ChatGPT obtains above-average results for problems of lower and medium difficulty, while its performance worsens when solving more complicated programming problems. The experimental results suggest that while ChatGPT cannot produce good source code as human programmers do, it demonstrates a promising potential as a programming assistant marking the beginning of a new era in the utilization of machine learning and large language modeling based approaches for code generation.

The main contributions of our work are summarized as follows.

- By means of a dataset collected from LeetCode, we evaluate *(i)* how well can ChatGPT solve general programming problems; and *(ii)* how ChatGPT's solutions compare to those by humans in terms of runtime and memory usage.
- The dataset and the paper corpus curated in this paper have been published online to allow for future research.[3]

The remainder of this paper is organized as follows. Section 2 introduces key concepts and terminology essential for understanding the context of our work. Section 3 details the pipeline that we built for evaluating the code generated by ChatGPT. In Section 4, we describe the research methodology, including research goal and questions, experimental design, and threats to validity. Section 5 presents a comprehensive summary of the results of our experiment, while Section 6 delves into a discussion of the implications of our results. Finally, Section 8 concludes the work with final remarks and future works.

2. Background

In this section, we introduce key concepts and terminology essential for understanding the context of our work.

2.1. Natural language processing and ChatGPT

Natural Language Processing (NLP) is a sub-field of AI (Aker et al., 2019) that aims at enabling computers to understand, process, and generate human language. It encompasses various techniques, including statistical analysis, and computational linguistics, to develop models capable of interpreting the meaning, emotional tone, and intent behind human language, hence facilitating effective interaction between computers and humans (Arnold et al., 1994). The field of NLP originated in the 1950s and 1960s, as researchers began exploring the possibility of using computers to understand human language (Nadkarni et al., 2011). Early NLP systems used a set of grammatical rules to analyze and create sentences, but they could not handle complex sentence structures and the nuances of human language. In the 1980s and 1990s, researchers started to explore statistical approaches for NLP, as described by Nadkarni et al. (2011). Models such as Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) were developed, which were more flexible and could handle a wider range of sentence structures. However, they still faced challenges with the nuances of language, such as detecting sarcasm. In the early 2000s, researchers applied ML techniques, such as Support Vector Machines (SVMs) and neural networks, to NLP (Nadkarni et al., 2011). These techniques needed large amounts of labeled data to train models and were able to capture more subtle patterns in language with greater accuracy. Today, NLP approaches include techniques such as Transformers and Recurrent Neural Networks (RNNs) (Gillioz et al., 2020). These approaches can process massive amounts of text and generate human-like responses, making applications such as chat-bots, language translation, and voice assistants possible (Natural Language Processing, 2023).

As NLP techniques evolve, they have enabled the development of sophisticated chat-bots like ChatGPT. ChatGPT is developed by OpenAI and built upon the advanced GPT-3.5 and GPT-4 architectures, which allow the model to understand and process complex human language effectively, as well as tackle complex tasks such as programming (OpenAI, 2023). ChatGPT has been trained on a large amount of text and code, which includes various user questions and their appropriate responses. In addition, the extensive data-set has equipped ChatGPT with a broad knowledge base that encompasses various programming languages and concepts (Israelsen, 2023; OpenAI, 2023). Given these capabilities, ChatGPT is a promising candidate for investigating the potential of chat-bots as a replacement for human programmers. Therefore, we believe that it is a suitable candidate for exploring the potential of chat-bots replacing human programmers.

2.2. LeetCode

LeetCode is an online platform offering a collection of programming problems and challenges that is widely used by programmers to practice and enhance their coding skills. LeetCode provides a reliable repository of programming challenges categorized based on their difficulty levels and topics. These difficulty levels encompass "easy", "medium", and "hard", while the topics span a wide range, including algorithms, databases, shell scripting, concurrency, and more. It is worth noting that our contribution does not include the categorization of problems based on their difficulty level or topic. However, readers – who are interested in exploring the specific categorization of problems – can refer to the LeetCode platform for greater details.[4] Typically, each programming problem comes with a problem statement, sample inputs and outputs. LeetCode includes a built-in editor and compiler, which allows users to test their code against a set of predefined test cases for evaluating its correctness and efficiency. In addition, LeetCode tracks submission status, provides error messages, and places users in distinct performance percentiles based on their submission scores. In this work, we leverage LeetCode as a source of diverse programming problems and challenges for our experiment. We prompt ChatGPT with programming problems and challenges of varying topics and difficulty available on the platform. We then submit the solutions generated by ChatGPT to LeetCode to compare the accuracy and efficiency of ChatGPT generated code against the code generated by human programmers.

2.3. Code quality

Avizienis et al. emphasized the significance of dependability and security in computing systems, which can only be achieved through the development of high-quality code (Avizienis et al., 2004). Poor code quality can lead to bugs, errors, and vulnerabilities, jeopardizing the reliability and security of a system (Avizienis et al., 2004). Error detection and recovery techniques also rely on properly functioning code. Thus, developing good code is crucial for dependable and secure computing systems (Avizienis et al., 2004). In software engineering, code quality is evaluated based on several attributes including memory efficiency, and runtime performance (Sharma et al., 2022). This work focuses on both these two metrics. In this study, runtime performance refers to the total duration required for a solution to execute. Memory utilization refers to the overall amount of memory resources used by the solution during its execution. For the measurement and evaluation of runtime performance and memory utilization, we rely on the LeetCode's publicly available data that serves as a reliable benchmark for assessing these properties of the generated code.

[3] https://github.com/hampusekedahl/ChatGPT_Experiment_Extended

[4] https://leetcode.com/discuss/interview-question?currentPage=1&orderBy=most_relevant&query=levels

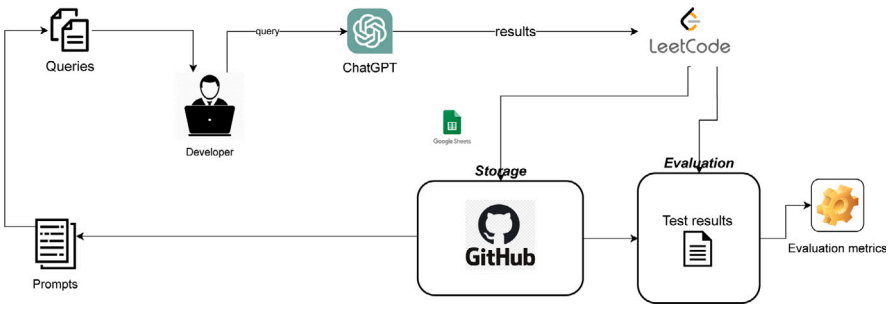

Fig. 1. The pipeline.

3. The pipeline for evaluating code generated by ChatGPT

In our study, we employed key tools for data collection and analysis, including the GPT-4.0 language model, LeetCode, Git and GitHub, Google Sheets, a Python GUI, and various Python frameworks. We accessed and executed the GPT-4.0 language model through the OpenAI website.[5] To utilize the ChatGPT-4.0 API, we subscribed to a monthly plan that incurred a cost of $20. This subscription was crucial for conducting our experiments and generating the required solutions for our research. An overview of the proposed pipeline is shown in Fig. 1. Data was collected based on correctness, runtime, and memory usage using LeetCode's built-in code submission system. The system compiles and executes submitted code on a range of test cases for each programming question, and evaluates the output against the expected output for each test case.

We stored all data in an Excel spreadsheet and linked it with relevant code solutions and errors in GitHub[6] by means of unique IDs. A dedicated GitHub repository was created to store the solutions and errors generated during each iteration of the experiment. Each experiment was assigned a Google Sheets document, and the data was organized in a structured format, facilitating efficient analysis. Essentially, the use of Google Sheets minimizes the potential for errors and inconsistencies, and ensures that the data remains well-organized and easily accessible for further analysis. The utilization of LeetCode is threefold as follows:

- We leveraged the extensive collection of programming questions available on LeetCode to construct our dataset.
- LeetCode is used to collect statistical data on the runtime, memory usage, and acceptance rates of human-generated solutions for programming tasks.
- Eventually, LeetCode provides a validation and bench-marking mechanism for the solutions generated by ChatGPT.

To ensure consistency and standardization in presenting the problems to ChatGPT, we developed a graphical user interface (GUI) tool – shown in Fig. 2 – using the PySimpleGUI library for generating standardized prompts. This GUI plays a crucial role in our experiments by facilitating prompt generation. The Python GUI offers a user-friendly interface, allowing us to seamlessly generate prompts. It accepts as input each question in its corresponding starting template, and automatically generates prompts in a specific format, described in Section 4.2. We could select programming problems from our data-set and generate prompts based on predefined templates. Additionally, the GUI offers the functionality to incorporate solutions based on errors encountered in previous iterations. This feature enhances the efficiency of our experiments, enabling us to iterate and improve the prompts based on the feedback received.

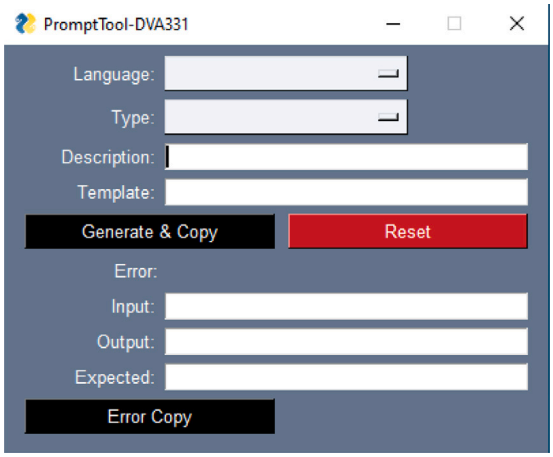

Fig. 2. The GUI for generating standardized prompts.

It is worth mentioning that the prompt generator is a graphical user interface tool developed as an effective means to generate standardized prompts. Essentially, the tool is conceived solely as a supporting element, helping users to generate prompts, nevertheless it does not have a mere impact on the final results.

Once the prompt was generated using the GUI tool, it was stored in a GitHub repository. The prompt was then passed as input to ChatGPT to produce an output response, which was then documented in the same GitHub repository for further reference and analysis. Subsequently, the output response was fed as input to LeetCode's built-in editor to submit it as a solution to the corresponding problem. If the solution generated by ChatGPT passes the test cases on LeetCode, the corresponding feedback, including metrics and statistics, was documented and collected using Google Sheets. In contrast, if an error occurs during the submission process, it is documented, and passed back to the GUI tool. The error message was then formatted into our template for error occurrences, and the modified prompt was once again passed to ChatGPT for another iteration of generating a response. This process was repeated for up to a maximum of three iterations. After the third iteration, if the solution still fails, then this will be counted as a negative score, to be used for computing the final results.

When deciding on the prompt format for the experiments, our primary focus was to create clear and concise prompts. Clarity and conciseness were crucial to ensure that ChatGPT accurately understands our instructions and to minimize any potential misunderstandings. In addition, this helps us ensure that the responses produced were meaningful and relevant to our research aim.

For data analysis and visualization, we employed four Python libraries, i.e., *matplotlib*, *numpy*, *seaborn*, and *pandas*. These frameworks are used to analyze and visualize the data collected from our experiments, allowing for insightful interpretations and representations of the results. To analyze the collected data, we performed a descriptive

[5] https://openai.com/
[6] https://github.com/

statistical analysis following existing work (Fisher & Marshall, 2009). Our analysis primarily focuses on key metrics, including runtime, and memory usage, and success rate per iteration.

4. Research methodology and study material

In this section, we describe the Research Questions (RQs), the research methods, including the experimental design, employed in our study that evaluated the possibility of ChatGPT writing correct programs.

4.1. Research questions

The goal of our work is to *understand whether ChatGPT can replace humans in programming tasks*. We break down this goal into the following RQs that provide unique and complementary answers to this investigation.

- **RQ$_1$**: *How well can ChatGPT solve general programming problems?* We perform an empirical evaluation using ChatGPT and LeetCode to determine the extent to which ChatGPT is capable of solving general programming problems.
- **RQ$_2$**: *How do ChatGPT's solutions compare to those by humans in terms of runtime and memory usage?* We look for insights into the performance of ChatGPT's solutions compared to those created by human programmers in terms of runtime and memory usage. This can be used to evaluate the feasibility and effectiveness of using ChatGPT as a tool for automating programming tasks.

To answer the aforementioned RQs, we used an iterative research process that draws upon the Action Design Research (ADR) process (Sein et al., 2011). We complemented the ADR process with several empirical research methods including a systematic literature review and experiments (Wohlin et al., 2012). In particular, we employ the former to gain an understanding of the current state-of-the-art ChatGPT for programming tasks, and the latter for answering RQ$_1$ and RQ$_2$. In the following sections, we provide details on the experiment processes.

4.2. Quantitative evaluation

This section first describes the settings and tools used in the experimental approach and then delve on the design and execution of the experiments used to answer RQ$_1$ and RQ$_2$.

The overall experimental approach is depicted in Fig. 3. A diverse and versatile data-set was curated by selecting 120 programming problems for each language, i.e., C++ and Java from LeetCode, encompassing **nine** distinct categories and **three** difficulty levels, aiming to reflect a natural learning progression for computer science students. A random selection process was applied to choose questions from each category and level of difficulty. We designed our experiment to ensure comprehensiveness and representativeness across various programming concepts and difficulty levels. The categories are array, string, sorting, math, hash table, binary search, dynamic programming, greedy, stack and queue, covering **easy**, **medium**, and **hard** tasks.

With RQ$_1$, we assess the ability of ChatGPT to solve a programming problem using C++ and Java. To this end, we chose a straightforward prompt: *"Solve this programming problem with C++"*, or *"Solve this programming problem with Java"*. This prompt focuses solely on solving the problem without any specific requirements regarding, e.g., runtime performance or memory utilization. By keeping the prompt generic, we wanted to evaluate ChatGPT's problem-solving capabilities without biasing it towards any particular performance metric. An example of a prompt for this task is shown in Fig. 4, and the corresponding code generated by ChatGPT is in Fig. 5.

With RQ$_2$, we further evaluated ChatGPT's runtime and memory utilization. To this end, we provided specific instructions regarding performance expectations by modifying the prompt format for RQ$_2$ to include a clear directive for optimizing runtime and memory utilization, i.e., *"Solve this programming problem with C++ as runtime and memory efficient as possible"*. By explicitly stating the performance requirements, we ensured that ChatGPT would focus on generating solutions that were not only correct but also efficient in terms of runtime and memory usage. The source code generated by ChatGPT was then downloaded to our local computers, and then uploaded to LeetCode, which performs further analysis. In particular, thanks to its internal design, LeetCode is able to measure both the running time and memory consumption for any input code snippet.

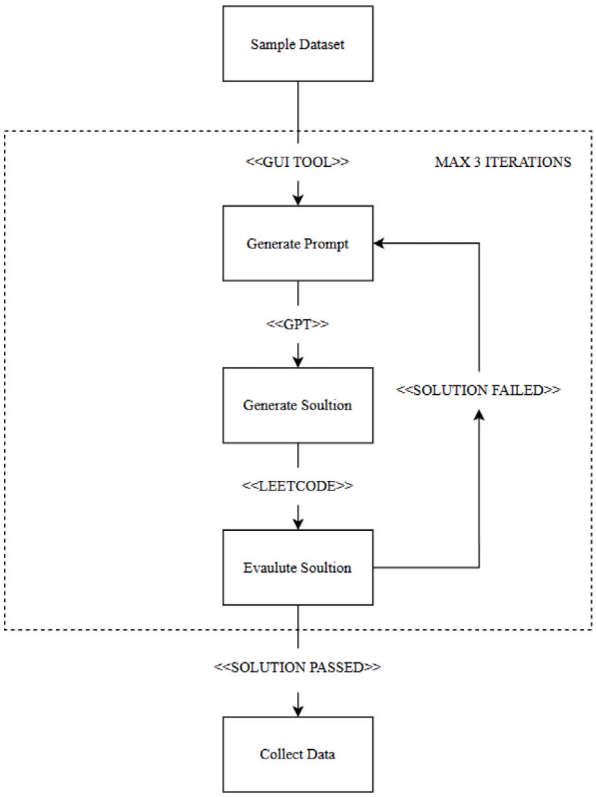

Fig. 3. The experiment process.

When a solution was not accepted as correct, the next iteration was based on the error message from previous solution. The format for prompts generated from error message was also straightforward and followed this structure: *With input "x" output was "y" but expected result was "z"*. If the error was a runtime error the complete error message was used as prompt to be fed to ChatGPT. Given the time constraints of our work, it was necessary to establish a threshold for the maximum number of iterations allowed for prompt one and prompt two. We determined that a maximum of three iterations would be allocated to each research question. This decision was based on our investigation into the model's behavior when faced with programming problems. During this investigation, we noticed that the majority of problems were typically resolved within the first attempt. However, we also identified a consistent pattern: if the model failed to produce a solution after three consecutive attempts, we deemed the question either unsolvable or the process overly demanding. By setting this threshold, we were able to efficiently allocate our time and resources while examining the limitations of the model and making necessary adjustments to our approach.

5. Results

In this section, we present the findings of our study organized and aligned according to the identified research questions.

```
Solve this programming problem with C++

Given an array of integers nums and an integer target,
return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6
Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6
Output: [0,1]

Constraints:

2 <= nums.length <= 10^4

-10^9 <= nums[i] <= 10^9

-10^9 <= target <= 10^9

Only one valid answer exists.
Code starting template:

class Solution {
 public:
 vector<int> twoSum(vector<int>& nums, int target) {

 }
};
```

Fig. 4. Example of prompt one of the Two Sum problem from LeetCode.

Certainly! Below is the solution to the problem you described, implemented in C++:

cpp Copy code

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> numMap;

        for (int i = 0; i < nums.size(); i++) {
            int complement = target - nums[i];

            // Check if the complement exists in the map
            if (numMap.find(complement) != numMap.end()) {
                // Return the indices of the two numbers
                return {numMap[complement], i};
            }

            // Add the current number and its index to the map
            numMap[nums[i]] = i;
        }

        // No solution found, return an empty vector
        return {};
    }
};
```

Fig. 5. An example of code solution generated by ChatGPT.

5.1. RQ_1: *How well can ChatGPT solve general programming problems?*

In this research question, we evaluate the capability of ChatGPT in solving various programming problems following three levels of difficulty defined by LeetCode. To this end, we used success rate to study how well ChatGPT is successful by different attempts. We measured the cumulative success rate across each of the three iterative attempts. We found out that 90% of the solutions passed the test of LeetCode, on the first iteration. On the second iteration the success rate increases to 97.78%. However, by the third iteration, ChatGPT was incapable of solving the programming problems, i.e., success rate remains the same of the second iteration, 97.78%. This suggests that while ChatGPT had an initial success rate of 90% on the first, and further solved some problems on the second iteration, increasing the success rate to 97.78%, ChatGPT was unable to solve any of the remaining unsolved problems during the third iteration.

Table 1
RQ_1: Success rate per iteration corresponding to each difficulty level (%).

Easy iteration			Medium iteration			Hard iteration		
1st	2nd	3rd	1st	2nd	3rd	1st	2nd	3rd
96.67	100.00	100.00	96.67	100.00	100.00	76.67	93.33	93.33

In our examination of ChatGPT's problem solving abilities per level of difficulty, we utilized the difficulty of the programming problems and the corresponding cumulative success rate, as shown in Table 1. The figure illustrates the success rate across iterative attempts, segmented by the problem's difficulty level. It is evident that ChatGPT obtains a higher success rate by easy and medium programming problems compared to that obtained by hard problems. For the Easy category, we get 96.67%, 100%, and 100% as success rate by the first, second, and third attempt, respectively. This is also the result obtained by the Medium category. This demonstrates that ChatGPT obtains a comparable outcome for the first two categories, i.e., Easy and Medium. However, with hard programming problems, ChatGPT suffers a worse performance compared to the previous two categories. In particular, by the first iteration, only 76.67% of the programming problems are successful, and the remaining are not. Even by the second and third attempts, though the overall performance is improved, not all the programming problems get passed, as demonstrated by a success rate of 93.33% for both cases. This essentially means that the level of difficulty of the programming problems poses a challenge for ChatGPT, i.e., it becomes less effective with complex tasks. For Java tasks, we also witnessed a similar trend in the obtained success rate. Thus, the results are not shown for the sake of clarity.

Answer to RQ_1. The experimental results suggest that while ChatGPT performs well with easy and medium programming problems, it encounters difficulties when solving more complicated tasks. In this respect, we conclude that ChatGPT needs to be further enhanced, so as to boost up its ability in programming.

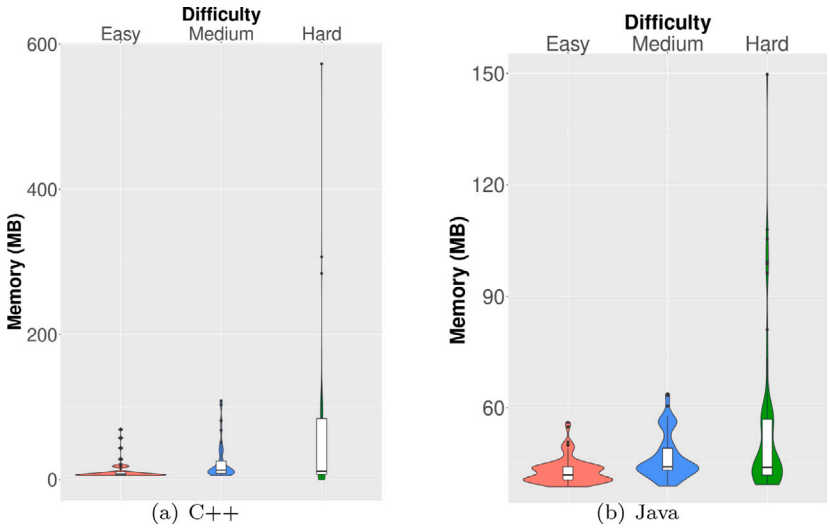

Fig. 6. RQ_2: Memory usage for C++ and Java tasks.

5.2. *RQ_2: How do ChatGPT's solutions compare to those by humans in terms of runtime and memory usage?*

In this research question, we further evaluate the performance of ChatGPT on programming problems in comparison to human programmers, focusing on two performance characteristics, i.e., runtime and memory usage. The runtime for C++ solutions is quite small, especially for those that are classified as easy. By most of the easy programming problems, ChatGPT requires a tiny fraction of time to finish, i.e., less than 100 ms. When ChatGPT deals with medium programming tasks, it needs a bit more time to complete, nearly reaching 500 ms in some extreme cases. With hard programming problems, the execution of ChatGPT may last for several milliseconds. Apart from some outliers starting from 1000 ms, and spanning up to 3000 ms, most of the tasks require almost 500 ms to complete. For Java tasks, we can see that there is a similar trend in the execution time according to the level of difficulty, compared to that of tasks written in C++. In general, ChatGPT requires more runtime when it comes to dealing with complicated programming problems.

The memory usage for the tasks written in C++ and Java is depicted in Fig. 6. To display the results, we make use of violin boxplots, a combination of boxplot and density traces to yield a more informative indication of the distribution, as well as the magnitude of the density (Hintze & Nelson, 1998). The memory usage for tasks written in C++ varies according to the level of difficulty, as shown in Fig. 6(a). In particular, with Easy tasks, ChatGPT consumes a small amount of memory, i.e., less than 30MB. With respect to an increase in the level of difficulty, ChatGPT needs additional memory, i.e., up to 100 MB. When the tasks become harder, the memory usage increases accordingly. Looking at Fig. 6(b), we witness a similar trend for the memory with Java tasks. With easy programming problems, ChatGPT usually finishes after using less than 45 MB, and it needs a bit more memory with medium tasks, i.e., ranging from 45 MB to 60 MB. Interestingly, for more difficult tasks, the runtime is not increased too much, i.e., apart from outliers, most of the tasks require less than 60MB of memory.

The results in Fig. 6 show that ChatGPT is faster and more memory efficient when running on Java tasks, compared to running on C++ tasks. We assume that this happens possibly due to the fact that ChatGPT has been trained with several Java snippets, which have been optimized to save execution time and memory. This, however, is a pure assumption, and in order to reach a solid conclusion we need empirical evidence, which can only be obtained from additional evaluations. This is out of scope of this paper, and we consider the issue as our future work.

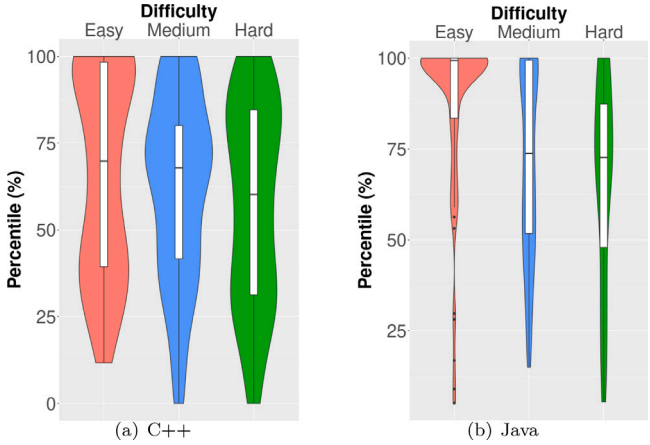

Fig. 7. RQ_2: Percentile runtime.

When the code generated by ChatGPT was submitted to LeetCode, it was evaluated by LeetCode, which eventually produced a report concerning the percentile of runtime and memory usage, in comparison with the code written and submitted by several humans through the platform. These metrics are used as a means to compare the overall performance of code written by ChatGPT and that of humans.

Fig. 7 shows the percentile of runtime for the solutions generated by ChatGPT, in comparison with those by humans. By most of the programming tasks, ChatGPT ranks from the 25% up to the 100% percentile. For easy and medium programming problems, ChatGPT obtains an encouraging performance with respect to those implemented by human developers. Especially by the medium tasks, by most of the solutions, ChatGPT ranks in the 75% percentile, i.e., compared to code by humans. Looking at the average scores, we can see that ChatGPT is ranked in the 58th, 59th, and 56th for easy, medium, and hard problems, respectively.

With respect to the Java programming problems, as shown in Fig. 7(b), ChatGPT yields a promising performance with easy tasks, i.e., most of the solutions produced by ChatGPT reside in the upper part of the corresponding diagram, close to the 100% percentile. However, by medium and hard tasks, the percentile by the solutions of ChatGPT is considerably low compared to that for solutions written by humans, i.e., most of the points are distributed across a low threshold. *These rankings suggest that with respect to timing efficiency, the code generated by ChatGPT does not earn a good merit compared to that by real developers.*

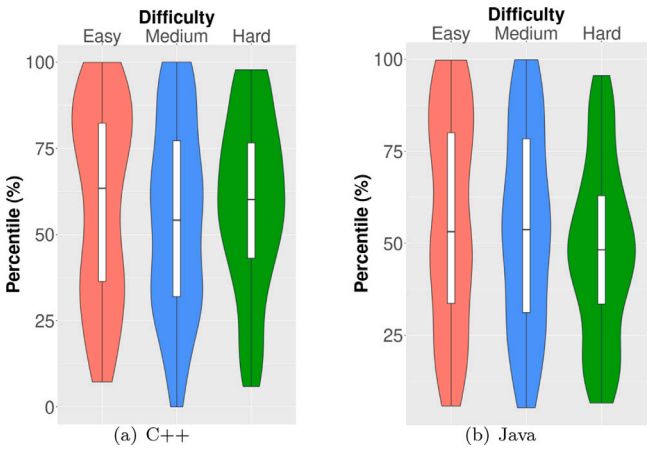

Fig. 8. RQ_2: Percentile memory.

In Fig. 8, we show the distribution of percentiles of memory for the solutions generated by ChatGPT, in comparison with those implemented by humans. For C++ tasks, ChatGPT's percentiles are distributed in a wide range of values, starting from 10% to 100%, with most of the instances residing from the 60% to 87% thresholds. When considering the mean values, ChatGPT is ranked in the 58th percentile for easy problems, 47th percentile for medium problems, and 56th percentile for hard problems. These results suggest that ChatGPT obtains a comparable outcome with respect to the majority of human programmers on easy and medium problems. However, when it comes to more complex problems, ChatGPT does not get a better result compared to a significant proportion of human programmers.

When we compared the result of RQ_1 and RQ_2, we made an interesting observation regarding the impact of context information on ChatGPT's performance in terms of runtime and memory utilization. ChatGPT's overall performance in terms of runtime and memory utilization was negatively affected when using prompt two. However, despite the decrease in performance metrics, we observed that adding more contextual information actually improved ChatGPT's ability to solve a problem on the first iteration.

Answer to RQ_2. Despite getting an encouraging performance with respect to timing and memory usage, ChatGPT is far from optimal compared to human developers by the submitted solutions for programming problems written in C++ and Java.

6. Discussion

The primary objective of our work was to investigate the potential of ChatGPT as a replacement for human programmers in programming tasks. This section discusses the implications, and the threats to validity of our findings.

6.1. Implications

While the metrics we selected offer valuable insights into the capabilities of ChatGPT in programming tasks, code quality encompasses various aspects that extend beyond the metrics we focused on. Aspects as readability and scalability are also crucial for assessing the quality of code. Although our study has limitations in capturing these aspects, it provides valuable insights into ChatGPT's ability to generate accurate and efficient solutions to a diverse range of programming problems.

The conclusion drawn from these results may seem counterintuitive at first. It suggests that when prioritizing performance, providing a simple prompt to ChatGPT may be more effective. However, if the primary goal is to increase the chances of solving the programming problem, providing a more detailed and explicit prompt is recommended. This implies that there is a trade-off between performance and problem-solving capabilities. The relation between ChatGPT's performance and more contextual information raises important questions about how to effectively prompt ChatGPT and prioritize the user's needs. In real-world programming scenarios, developers must consider multiple factors when determining the best solution, and the ability of ChatGPT to handle such complexities becomes crucial.

A possible extension is to investigate the ability of ChatGPT on code summarization tasks, i.e., explaining a program first by reading the source code, and then giving a summary written in natural language. This has been recently studied by Sun et al. in their research (Sun et al., 2023), and we suppose that further attention should be paid in order to explore how ChatGPT can summarize code written in different languages.

It is worth remarking that our study focuses on small programming problems, and does not consider the performance of ChatGPT on larger and more comprehensive programming problems. The question of how ChatGPT performs with such problems is indeed interesting and warrants further investigation. We expect that larger and more complex problems may pose challenges for ChatGPT's performance. Future research could delve into the performance of ChatGPT on larger problems, explore methods to improve its performance in more complex scenarios, and investigate how developers can effectively leverage ChatGPT while considering various real-world constraints and requirements.

Our study suggests that ChatGPT has the potential to partially replace humans in programming tasks. For instance, ChatGPT can be valuable as a programming assistant, particularly in automating repetitive tasks and providing assistance with simpler problems. However, when it comes to more challenging and complex programming problems frequently encountered in real-world scenarios, our study suggest that ChatGPT's performance are still limited. Indeed, as AI technologies, including chatbots like ChatGPT, continue to evolve rapidly, their impact on software engineering and other fields is expected to grow. However, along with their potential benefits, the increasing use of AI raises ethical concerns that must be carefully considered.

Considering the time limit and the nature of our process, alternative choices could have yielded different results. If we had opted for a larger number of iterations for each research question, it might have allowed the model to find a better solution, potentially leading to a higher success rate in solving the programming problems. However, this would have required a greater investment of time and resources. On the other hand, if we had set a lower maximum number of iterations, the model's ability to tackle challenging problems could have been compromised, potentially limiting the insights we could gather regarding its limitations. Moreover, our decision to establish the threshold based on the observation that most problems were resolved within the initial attempt meant that we prioritized efficiency and time management.

It is evident that different choices in terms of the maximum number of iterations and prompt format could have influenced the outcomes and implications of our study. The data from our study suggest that the majority of the problems were resolved during the second iteration, and subsequent iterations did not exhibit a significantly higher success rate. Employing prompts that were more specific and explanatory could have yielded different outcomes, not necessarily better ones.

The evaluation shows that ChatGPT performs well with easy and medium tasks, however when the exercises are at the hard level (classified by LeetCode), then it encounters some difficulties in fulfilling the tasks, and thus it does not pass the tests. This implies that ChatGPT has some limitations with respect to hard tasks. Such limitations happen possibly due to the training, i.e., ChatGPT might not have been well trained with this type of task. Moreover, we assume that if ChatGPT had been trained with data from LeetCode, then the code generated by ChatGPT would be successful when being tested with LeetCode. In

our work, we did not attempt to investigate ChatGPT's superiority over other systems or methods. Instead, the ultimate aim is to provide a reproducible assessment of ChatGPT's code generation capabilities.

ChatGPT has been built on top of GPT (Generative Pre-trained Transformer), which traces its roots back to Transformers, Encoder–Decoder, consisting of two sides, one for encoding the input data, and the other for decoding the output data. In this way, we can think of querying ChatGPT to get source code as putting a sentence written in natural language on one side (Encoder), and getting the source code on the other (Decoder). The main focus of our work is to evaluate ChatGPT's code generation capabilities. Thus, we conducted a practical and empirical assessment of its utility as a code generation tool, rather than running into an exhaustive analysis of ChatGPT's underlying architecture and reasoning mechanisms. A comprehensive examination of its architecture and reasoning mechanisms could reveal additional insights into its strengths and weaknesses, especially in the context of general programming problems. This deserves an independent investigation, which can be investigated in another paper.

6.2. Threats to validity

In this section, we discuss different types of validity threats that could affect the results of our study along with the strategies adopted to mitigate them.

- Threats to *external validity* refer to the factors that could impact the generalization of the findings of an experiment to real-world applications (Wohlin et al., 2012). In our study, one of the potential threats to conclusion validity may be related to the chosen programming language and to the problem set collected from LeetCode, i.e., tasks related to C++ and Java, two popular programming languages. For future work, we plan to extend our experiments with other languages, including Python. For the evaluation, we had a dataset of 240 programming exercises, and such a number might not be large enough to be generalizable. In fact, curating a dataset for the experiments by involving both ChatGPT and LeetCode is a strenuous process, requiring a lot of time and effort, as the samples first were collected from ChatGPT, and then uploaded to LeetCode to run other measurements. Altogether, this is a prolonged procedure, preventing us from increasing the number of code samples. To mitigate this threat, we attempted to cover three main levels of difficulty, including Easy, Medium, and Hard. Moreover, the tasks spread over 10 different categories in programming, including: 'Array', 'Dynamic Programming', 'Hash Table', 'Queue Stack', 'Binary Search', 'Greedy', 'Math', 'Sorting', and 'String'.
- Threats to *internal validity* concern the factors that can potentially affect the causal relationship between the independent variable and the outcome. In the experiments, we evaluated the solutions by ChatGPT and humans using the same metrics. This is to make sure that we performed a fair comparison.
- Threats to *conclusion validity* are factors that could impact the ability to correctly draw conclusions about the relationship between the treatment and the outcome of an experiment. The chosen metrics may not capture all aspects of code, such as readability or scalability. Eventually, the data from LeetCode may not be representative of the overall performance of human programmers, which could also limit the validity of our conclusions
- Threats to *construct validity* are related to the experimental design, as well as to social factors. In our study, one potential threat is that the programming tasks given to ChatGPT may not fully represent the complexity of tasks typically performed by human programmers, which could affect the validity of our evaluation. Additionally, the competition among participants may impact on the performance of human programmers on the LeetCode programming tasks. As ChatGPT learns from previous prompts within a conversation, this could also impact the construct validity of our study.

7. Related work

To review related work, we conducted a systematic literature review. We started with an automatic search in software engineering (Kitchenham & Brereton, 2013; Petersen et al., 2008) on the following four scientific databases and indexing systems: IEEE Xplore Digital Library,[7] ACM Digital Library,[8] Scopus,[9] and Web of Science.[10] Aiming at a reasonable trade-off between efficiency and the coverage of state-of-the-art studies, we adhered to the existing guidelines for systematic literature studies in software engineering. We queried the aforementioned databases using concise and well-constructed search strings, collecting as many studies as possible. Given the novelty of the topic, the following query strings were utilized:

"ChatGPT" AND "programming"

The initial search produced a set of 27 peer-reviewed publications.[11] From this set, we removed impurities and duplicates and obtained a new set of 22 publications. We used the guidelines by Ali and Petersen (2014) to define the following selection criteria for filtering the primary studies.

- **Inclusion criteria.** We considered studies that are: *(i)* subject to peer review; *(ii)* written in English; *(iii)* available as full-text; and *(iv)* focusing on ChatGPT and programming.
- **Exclusion criteria.** To ensure the inclusion of as many relevant studies as possible and minimize threats to validity, we excluded studies that are shorter than 4 pages.

Apart from the studies obtained by conducting the literature review, we also took into account some additional papers that are relevant to the topic being under consideration, and eventually got a set of relevant papers to be reviewed as follows.

Jacques (2023) investigated the potential of using ChatGPT to help instructors in enhancing the critical thinking skills of novice programmers. Inspired by the historical developments in mathematical education, the author hypothesized that fluency and basic skills in programming are important regardless of the availability of advanced tools. Based on this hypothesis, Jacques proposed various tasks aimed at enhancing the critical thinking skills of novice programmers. One task involved instructing novices to create flowcharts that depict the logical structure of programs generated by ChatGPT. In addition, the author suggested a task where learners were prompted to extend or modify a program that was generated by ChatGPT. While the work utilized ChatGPT to generate programs, the primary focus was not on assessing the correctness, runtime performance, or memory usage of the generated solutions.

Similar to Jacques (2023), Kazemitabaar et al. tried to understand whether novice developers were able to understand the code generated by ChatGPT and to modify or extend the generated code (Kazemitabaar et al., 2023). In addition, the authors also studied if using such tools would form a reliance, or help learners write code without such tools being present. To investigate these aspects, they developed a web-based application called Coding Steps. This application was specifically designed to facilitate the learning of basic Python programming. Coding Steps provides learners with a progressive set of programming tasks that introduce new concepts gradually. It offers a submission functionality that allows learners to submit their code to remote instructors for grading and feedback, which is provided through a dedicated

[7] https://ieeexplore.ieee.org/Xplore/home.jsp
[8] https://dl.acm.org/
[9] https://www.scopus.com/
[10] http://webofscience.com/
[11] It is important to remark that we performed the automatic search in June 2023.

grading dashboard. Additionally, Coding Steps incorporates code generation capabilities using the *code-davinci-002* model from OpenAI's code completion API. This feature generates code to assist learners in completing their programming tasks. The study's results demonstrated that ChatGPT can be a valuable asset for computer science educators and students. Novice programmers who used Coding Steps environment exhibited improved performance, increased efficiency, and reduced frustration when writing code. Importantly, the use of Coding Steps and ChatGPT did not negatively impact their ability to manually modify code or perform tasks without its support.

Yilmaz et al. investigated the effect of using ChatGPT on students' computational thinking skills, programming self-efficacy, and motivation towards the lesson (Yilmaz & Karaoglan Yilmaz, 2023). They conducted the research on 45 undergraduate students who took a university-level programming course using the experimental design with the pretest-posttest control group. Students were randomly divided into the experimental group that could use ChatGPT during the weekly programming practices, and the control group that could not use it. Their findings revealed that the experimental group students' computational thinking skills, programming self-efficacy, and motivation for the lesson were significantly higher than the control group students. Hence, it can be said that ChatGPT may be useful in programming training.

In a broader context, Lertbanjongngam et al. compared the performance of the code generation tool AlphaCode[12] to that of human programmers (Lertbanjongngam et al., 2022). The study discovered that AlphaCode was capable of generating code that was often similar to human-generated code, and performed equally or worse in terms of execution-time and memory utilization. While the authors measured the execution-time themselves to evaluate solutions (Lertbanjongngam et al., 2022), we relied on the publicly available data provided by the LeetCode platform, which serves as a reliable benchmark for assessing the accuracy and efficiency of the generated code. Additionally, the authors used a popular competitive programming platform, namely Codeforces,[13] to retrieve code produced by humans. Similar to Lertbanjongngam et al. (2022), we compared the performance of AI-generated code to human programmers. However, we did not focus on competitive programming, which is an important distinction. Competitive programming platforms like Codeforces and technical interview preparation platforms like LeetCode serve different purposes and require different skills and approaches. For example, Codeforces places more emphasis on time and memory limits (Mirzayanov, 2012), whereas LeetCode focuses on optimizing solutions and improving performance relative to other users.

Finnie-Ansley et al. investigated the accuracy of an AI model designed for code writing named Codex on a range of first-year college-level programming problems (Finnie-Ansley et al., 2022). Their study found that Codex outperformed most first-year programming students by ranking in the top quartile of a typical first-year programming exam. Their study evaluated the accuracy of Codex on 23 basic computer science programming problems, where the authors presented the assignments exactly as they were given to the students. They also assessed Codex through variations of the problem wording for the Rainfall Problem. Unlike Finnie-Ansley et al. (2022), we prompted ChatGPT with programming problems of varying difficulty levels and a custom-designed question and compared ChatGPT's performance to public data available on LeetCode, rather than using student-generated data. Moreover, we used predetermined prompts, whereas Finnie-Ansley et al. used randomly varied problem wordings (Finnie-Ansley et al., 2022).

[12] https://alphacode.deepmind.com/

[13] https://codeforces.com/

8. Conclusions and future work

In this paper, we have evaluated the potential of ChatGPT in replacing humans as programmers. To achieve this, we have started with a systematic literature review to gain an understanding of the current state-of-the-art in the use of ChatGPT for programming tasks. Then we have designed and conducted experiments to assess the extent to which ChatGPT is capable of solving general programming problems. The objective is to assess ChatGPT's capabilities in two different programming languages, namely C++ and Java. To evaluate the quality of the generated code, we leveraged LeetCode's automated submission system, which assessed and provided feedback on the code correctness and performance with respect to runtime and memory usage.

Future work may encompass different directions. One direction may investigate the impact of dedicated training and its potential to enhance performance. Building upon the insights gained from our experiment, we observed that a more detailed prompt yielded more effective results in terms of finding solutions. We suppose that the architecture, the reasoning mechanisms, as well as the training of ChatGPT have a certain effect on general programming problems. This, however, needs to be investigated in a separate paper with concrete empirical evidence, and thus we consider it as our future work. A possible direction for future work is to explore the use of different programming languages than C++ and Java, and of a larger and more complex data-set of problems to gain a more comprehensive understanding of ChatGPT's capabilities. Finally, in our future work, we will also explore additional code quality metrics than runtime and memory usage, as well as involve revisiting the experiment with a focus on utilizing different parameters to assess code quality.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The work in this paper has been supported by the Swedish Knowledge Foundation (KKS) through the Modev project, by the Excellence in Production Research (XPRES) Framework and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the iSecure project. The authors would like to thank the anonymous reviewers for their valuable comments.

References

Aker, A., Ceausu, A., Feng, Y., Gaizauskas, R. J., Hunsicker, S., Ion, R., Irimia, E., Stefanescu, D., & Tufis, D. (2019). Mapping and aligning units from comparable corpora. In I. Skadina, R. J. Gaizauskas, B. Babych, N. Ljubesic, D. Tufis, & A. Vasiljevs (Eds.), *Using Comparable Corpora for under-Resourced Areas of Machine Translation, Theory and Applications of Natural Language Processing* (pp. 141–188). Springer, http://dx.doi.org/10.1007/978-3-319-99004-0_5.

Ali, N. B., & Petersen, K. (2014). Evaluating strategies for study selection in systematic literature studies. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement* (pp. 1–4).

Arnold, D., Balkan, L., Humphreys, R., Meijer, S., & Sadler, L. (1994). *Machine Translation: an Introductory Guide*. London: NCC Blackwell, URL http://www.essex.ac.uk/linguistics/external/clmt/MTbook/PostScript/.

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing, 1*(1), 11–33.

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference* ACE '22, (pp. 10–19). New York, NY, USA: Association for Computing Machinery.

Fisher, M. J., & Marshall, A. P. (2009). Understanding descriptive statistics. *Australian Critical Care*, *22*(2), 93–97.

Fuscaldo, D. (2023). How chatbots can help grow your small business. URL https://www.businessnewsdaily.com/16018-chatbots-for-growth.html.

Gillioz, A., Casas, J., Mugellini, E., & Khaled, O. A. (2020). Overview of the transformer-based models for nlp tasks. In *2020 15th Conference on Computer Science and Information Systems* FedCSIS, (pp. 179–183).

Hintze, J. L., & Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. *The American Statistician*, *52*(2), 181–184. http://dx.doi.org/10.1080/00031305.1998.10480559, URL https://amstat.tandfonline.com/doi/abs/10.1080/00031305.1998.10480559.

Israelsen, A. (2023). How to use ChatGPT to write code. URL https://www.pluralsight.com/blog/software-development/how-use-chatgpt-programming-coding.

Jacques, L. (2023). Teaching cs-101 at the dawn of chatgpt. *ACM Inroads*, [ISSN: 2153-2184] *14*(2), 40–46. http://dx.doi.org/10.1145/3595634.

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23, New York, NY, USA: Association for Computing Machinery., ISBN: 9781450394215, http://dx.doi.org/10.1145/3544548.3580919.

Kitchenham, B., & Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information and Software Technology*, [ISSN: 0950-5849] *55*(12), 2049–2075. http://dx.doi.org/10.1016/j.infsof.2013.07.010, https://www.sciencedirect.com/science/article/pii/S0950584913001560.

Lertbanjongngam, S., Chinthanet, B., Ishio, T., Kula, R. G., Leelaprute, P., Manaskasemsak, B., Rungsawang, A., & Matsumoto, K. (2022). An empirical evaluation of competitive programming ai: A case study of alphacode. arXiv preprint arXiv:2208.08603.

Littman, M. L., Ajunwa, I., Berger, G., Boutilier, C., Currie, M., Doshi-Velez, F., Hadfield, G., Horowitz, M. C., Isbell, C., Kitano, H., Levy, K., Lyons, T., Mitchell, M., Shah, J., Sloman, S., Vallor, S., & Walsh, T. (2021). *Gathering strength, gathering storms: The one hundred year study on artificial intelligence (ai100) 2021 study. sq2: What are the most important advances in ai? Technical report*, Stanford University, URL https://ai100.stanford.edu/2021-report/standing-questions-and-responses/sq2-what-are-most-important-advances-ai.

Marr, B. (2023). How ChatGPT and natural language technology might affect your job if you are A computer programmer. URL http://bit.ly/44fpw95.

Mirzayanov, M. (2012). Codeforces contest rules. URL https://codeforces.com/blog/entry/4088.

Müller, V. C. (2021). Ethics of artificial intelligence and robotics. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2021 ed.). Metaphysics Research Lab, Stanford University.

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551.

Natural Language Processing 2023. URL https://www.ibm.com/topics/natural-language-processing.

OpenAI (2022). ChatGPT. URL https://openai.com/blog/chatgpt.

OpenAI (2023). GPT-4 research. URL https://openai.com/research/gpt-4.

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. In G. Visaggio, M. T. Baldassarre, S. G. Linkman, & M. Turner (Eds.), *12th International Conference on Evaluation and Assessment in Software Engineering*. Italy: University of Bari, 26-27 2008, Workshops in Computing. BCS. URL http://ewic.bcs.org/content/ConWebDoc/19543.

Phillips, T. (2022). *Is There a Shortage of Developers? Developer Shortage Statistics in 2022*. https://codesubmit.io/blog/shortage-of-developers/, 5.

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly*, 37–56.

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H., & Sarro, F. (2022). A survey on machine learning techniques for source code analysis. arXiv preprint arXiv:2110.09610.

Sun, W., Fang, C., You, Y., Miao, Y., Liu, Y., Li, Y., Deng, G., Huang, S., Chen, Y., Zhang, Q., Qian, H., Liu, Y., & Chen, Z. (2023). Automatic code summarization via chatGPT: How far are we?.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering* (1st ed.). Berlin, Heidelberg: Springer, http://dx.doi.org/10.1007/978-3-642-29044-2.

Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023). The effect of generative artificial intelligence (ai)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, [ISSN: 2666-920X] *4*, Article 100147. http://dx.doi.org/10.1016/j.caeai.2023.100147, URL https://www.sciencedirect.com/science/article/pii/S2666920X23000267.