digital

MDPI

*Article*

# Effectiveness of ChatGPT in Coding: A Comparative Analysis of Popular Large Language Models

**Carlos Eduardo Andino Coello, Mohammed Nazeh Alimam and Rand Kouatly ***

Faculty of Tech and Software Engineering, University of Europe of Applied Sciences, 14469 Potsdam, Germany; carloseduardo.andinocoello@ue-germany.de (C.E.A.C.); mohammednazeh.alimam@ue-germany.de (M.N.A.)
* Correspondence: rand.kouatly@ue-germany.de

**Abstract:** This study explores the effectiveness and efficiency of the popular OpenAI model ChatGPT, powered by GPT-3.5 and GPT-4, in programming tasks to understand its impact on programming and potentially software development. To measure the performance of these models, a quantitative approach was employed using the Mostly Basic Python Problems (MBPP) dataset. In addition to the direct assessment of GPT-3.5 and GPT-4, a comparative analysis involving other popular large language models in the AI landscape, notably Google's Bard and Anthropic's Claude, was conducted to measure and compare their proficiency in the same tasks. The results highlight the strengths of ChatGPT models in programming tasks, offering valuable insights for the AI community, specifically for developers and researchers. As the popularity of artificial intelligence increases, this study serves as an early look into the field of AI-assisted programming.

**Keywords:** artificial intelligence; ChatGPT; GPT-3.5; GPT-4; Python programming; OpenAI; Google's Bard; Anthropic's Claude



**Citation:** Coello, C.E.A.; Alimam, M.N.; Kouatly, R. Effectiveness of ChatGPT in Coding: A Comparative Analysis of Popular Large Language Models. *Digital* **2024**, *4*, 114–125. https://doi.org/10.3390/digital4010005

Academic Editor: Jorge Bernardino

Received: 3 October 2023
Revised: 27 December 2023
Accepted: 2 January 2024
Published: 8 January 2024



## 1. Introduction

In recent years, artificial intelligence (AI) has experienced exponential growth, marked by significant advancements in natural language processing and machine learning. This surge has brought about a transformation in various industries and applications. One specific area that has garnered considerable attention is AI-assisted programming. Advanced language models have the potential to revolutionize the way developers create, maintain, optimize, and test code. The primary objective of this study is to evaluate the effectiveness of the most popular large language model publicly available, ChatGPT, encompassing both the GPT-4 and GPT-3.5 models. These two models undergo testing across a variety of code generation tasks in this study, with the aim of understanding their potential and overall performance.

OpenAI's release of the GPT models and the widely available ChatGPT represents a substantial breakthrough in the advancement of AI capabilities [1,2]. With each iteration, the models have demonstrated improved performance and versatility, generating increased interest in their potential uses and applications across multiple fields. In programming alone, these models have shown significant promise, particularly in automating tasks, improving code, and providing insights to developers.

The impact of AI in programming cannot be underestimated. AI has been proven to have the potential to enhance productivity, reduce human error, and automate tasks. Examples of such tasks include code generation, documentation, and bug detection, effectively streamlining the programming process. This, in turn, allows programmers to focus on more complex and creative aspects of their work.

To further analyze the subject, this study will employ quantitative testing. The primary objectives of this study are as follows:

- To evaluate the efficiency of ChatGPT in various code generation tasks;

- To compare the performance of GPT-4 and GPT-3.5 to that of other popular LLMs in said tasks;
- To identify challenges and limitations of using large language models in programming.

The breakthrough in automated code generation has been significantly propelled [3] and greatly boosted by recent advancements in large language models like GPT-3 [4], surpassing the capabilities of earlier state-of-the-art deep learning methods [5].

As an illustration, OpenAI Codex [6], a refined iteration of GPT-3, can produce entirely accurate code for 29% of unfamiliar programming tasks using just one sample of generated programs. It was found that when testing 100 samples, 72% of them are correct. In [7], the authors evaluate the GPT Python code-writing capabilities and the correctness of the code generated. The results in this paper are based on only a small number of samples, which shows that the model can solve only 28% of the problems. Hammond et al. [8] investigated the possibility of using OpenAI Codex and other large language models (LLMs) to fix software security bugs. The results show that 67% of vulnerabilities in a selection of historical bugs in real-world open source projects can be fixed and discovered by LLMs. Meanwhile, Refs. [9,10] tested the usability of the code generated by LLMs and not the accuracy of the codes.

Xu and colleagues [11] compared the performance of code generated by GPT-Neo, GPT-J, and GPT-NeoX—all large language models (LLMs)—when trained with a substantial number of parameters derived from ready codes in 12 different languages. Zan et al. [12] investigated the existing large language models for NL2Code and summarized them from diverse perspectives. However, neither of these research studies investigated the accuracy and the quality of the code generated by LLMs.

This study serves as an early look into the field of AI-assisted programming, at a time when artificial intelligence is experiencing exponential growth. The goal is to provide useful insights for developers, researchers, and the broader technology community to help shape the future of AI-assisted programming.

In Section 2 of this paper, a short concept about the history of ChatGPT will be introduced, followed by an explanation of how generative AI works. Section 4 of this paper will present the experimental results, and the conclusion will be summarized in Section 5.

## 2. Generative AI History

Modern generative AI development began in the 1940s after the conception of the first artificial neural networks (ANNs). However, due to constraints such as limited computational capabilities and insufficient knowledge of the brain's biological workings, ANNs failed to draw significant interest until the 1980s. During this period, parallel advances in hardware and neuroscience, along with the emergence of the backpropagation algorithm, eased the training process of ANNs. Previously, training NNs was a demanding task, as there was no effective method to compute the error's gradient relating to each neuron's parameters or weights. However, backpropagation automated the training procedure, unlocking the potential usage of ANNs [13].

In 2013, Kingma and Welling presented a novel model structure named variational autoencoders (VAEs) in their papers entitled "Auto-Encoding Variational Bayes". VAEs are generative models grounded in the principle of variational inference. They offer a mechanism for learning via a condensed representation of data, where the data are transformed into a lower-dimensional area called the latent space through an encoding process. Then, the decoder component reconstructs the data back into their original data space [14].

In 2017, Google researchers introduced a pivotal development in their research titled "Attention Is All You Need". This new architecture, called Transformer, was a revolution in language generation [15]. Unlike previous language models based on long short-term memory (LSTM) [16] or recurrent neural networks (RNN) frameworks [17], Transformer allowed for parallel processing while retaining context memory, leading to superior performance [17].

In 2021, OpenAI released a fine-tuned version of GPT, Codex, which was trained on code publicly available on GitHub. Early results showed that the fine-tuned model was able to solve around 30% of the Python problems used, compared to the 0% that the current GPT version (GPT-3) was able to achieve. This served as an early look into how large language models (LLMs) [10] can learn and generate code. Codex then served as the basis for GitHub Copilot [10].

GitHub Copilot is an AI programming tool that can be installed in the most popular code editors and is powered by GPT-4. It reads the code and can generate suggestions and even write code instantly. In a controlled test environment, researchers found that programmers who used Copilot finished tasks approximately 55.8% quicker than those who did not, speaking to the potential of AI tools in programming [18,19].

In another research work, GPT's Python code generation is deemed remarkable, showing that it can help novice programmers to solve complex coding problems using only a few prompts. However, both studies have shown that human input is almost always required to steer ChatGPT in the correct direction [20].

## 3. What Is Generative AI

Generative AI models harness the capabilities of neural networks to discern patterns and structures within existing datasets and create original content [21]. These AI models draw inspiration from human neuronal processes, learning from data inputs to create new output that matches learned patterns. This involves advanced techniques that range from generative adversarial networks (GANs) [21], large language models (LLMs), variational autoencoders (VAEs), and transformers to create content across a dynamic range of domains [22].

Numerous methodologies, such as unsupervised or semi-supervised learning, have empowered organizations to utilize abundant unlabeled data for training and laying foundations for more complex AI systems. Referred to as foundation models, these systems, which comprise models like GPT-3 and Stable Diffusion, serve as a base that can be proficient in multiple tasks. They enable users to maximize the potency of language, such as constructing essays from brief text prompts using applications like ChatGPT or creating remarkably realistic images from text inputs with Stable Diffusion [23].

Generative AI models can refine their outputs through repeated training processes by studying the relationships within the data. They can adapt parameters and diminish the gap between the intended and created outputs, continually enhancing their capacity to produce high-quality and contextually appropriate content. The utilization of this technology is often initiated with a prompt, followed by iterative exploration and refining of variations to guide content generation [24].

## 4. Testing Methodology

In this section, the methodological approach used to evaluate ChatGPT will be presented, along with the subsequent comparison with other reputable large language models (LLMs). Next, the description of the dataset used for this study will be explained. Finally, the evaluation strategy, detailing how the tests were performed and how the language model scores were calculated, will be provided.

### *4.1. Selection of LLMs*

After the massive popularity of ChatGPT emerged in the market, it was only natural that other companies would release their own large language models (LLMs) to compete with OpenAI. In order to paint a more complete picture, it was decided to choose three other popular and easily accessible LLMs and compare their performance. These models were selected based on the company's popularity as well as potential impact.

The initial LLM chosen for testing was Google's Bard. Bard was introduced in February 2023 as Google's next significant step into the AI space [25]. Powered by Google's PaLM2

model [26], Bard is said to excel at advanced reasoning tasks, including math and coding. Bard is currently available for free on the Bard website [27].

The second LLM to be tested was Microsoft's Bing (technically known as Bing Chat, but in this paper, it will be referred to as Bing only). Released in February 2023, Bing is the "copilot for the web" and is an AI chatbot powered by GPT-4, considering that Microsoft is one of the top investors in OpenAI. Bing Chat is also generally available on the Bing website and should have similar performance to ChatGPT [27,28].

The third LLM to be tested was Anthropic's Claude. While not as popular as Google and Microsoft, Anthropic's Claude was founded by a research company established in 2021 by former OpenAI employees. Claude was initially released as the next-generation AI assistant in March 2023. The new version, Claude v2, was quickly released in July 2023. According to Anthropic, Claude has similar use cases to ChatGPT and Bard, including coding capabilities [29,30].

### *4.2. Dataset*

This research mostly used basic Python problems based on a well-known and tested dataset called Basic Python Programming (MPPP) [7]. This dataset was employed to measure the code generation capabilities of the AI models. It was created by Google researchers and consists of approximately 1000 crowd-sourced Python programming problems. These problems cover various programming fundamentals, functionalities, etc., and are designed to be solved by entry-level programmers [7].

From the main dataset, a subset of around 460 problems has been hand-verified by Google, and this was the dataset used in this research. Each problem consists of:

- Task_id: a number from 1–1000.
- Prompt: the instructions for the LLM, explaining what the code should do.
- Code: a proposed solution for the code.
- Test_imports: libraries that should be imported.
- Test_list: usually 3 test cases to verify if the code works as expected.

Figure 1 shows an example of coding problem number 162 from the dataset [7], along with the test code used.

```
Task_id = "162"
prompt = """Write a function to calculate the sum (n - 2*i) from i=0 to n // 2,
        for instance n + (n-2) + (n-4)... (until n-x =< 0)."""
test_list = [
    "assert sum_series(6) == 12",
    "assert sum_series(10) == 30",
    "assert sum_series(9) == 25",
]
def sum_series(n):
  """Calculates the sum of the series (n - 2*i) from i=0 to n // 2.
  Args:
    n: The upper limit of the series.
  Returns:
    The sum of the series.
  """
  sum = 0
  for i in range(n // 2):
    sum += n - 2 * i
  return sum
```

(**a**)

**Figure 1.** *Cont.*

```
### INSERT LLM-GEBNERATED CODE HERE ###

score = 0
for test in test_list:
    try:
        exec(test)
        score += 1
        print("Test passed\n")
    except AssertionError:
        # Extracting the code from the assert statement
        test_code = test.split("assert ")[1].split("==")[0]
        # Evaluate the extracted code and capture the computed result
        computed_result = eval(test_code)

        # Extracting the expected result from the assert statement
        expected_result = (test.split("==")[1])

        print("Test failed:")
        print("Test:", test)
        print("Expected:", expected_result)
        print("Computed:", computed_result)
        print()

print("Total score:", score, "/", len(test_list))
```

(**b**)

**Figure 1.** Example of the used Python problems [7]. (**a**) The code problem number 162. (**b**) The testing code.

### *4.3. Evaluation Strategy*

Even though OpenAI offers paid API access to its GPT-3.5 and GPT-4 models per token used, it was decided to use the ChatGPT web interface for all of the tests. The reasoning behind this is that it is the most accessible way to access these models and is probably what most people are going to use due to its ease of use. All models are free to use via their web interfaces, except GPT-4, which is locked behind a paid subscription, "ChatGPT Plus". Bing and Bard do not have API access, while Claude does. However, there is currently a waitlist for Claude's API access. This was also another reason behind not using the APIs in the tests, so as to test all models based on their web interfaces.

This study used a purely quantitative approach. Each model was provided with the programming prompts from the MBPP dataset. For these tests, only the "prompt" was provided, as well as the name of the function, in order to match the function name that was used in the test cases. Figure 2 shows the prompt that served as the input to the LLM systems.

```
I will give you a prompt, follow the prompt to write the python code that was
instructed. Do this in one code block, not in parts. If you need to import
something, do it in the code.


"prompt": "Write a function to find the shared elements from the given two
lists.",
name the function: similar_elements


Write ONLY the function
```

**Figure 2.** Example of the prompt used as an input to the LLM systems.

The resulting generated code by the LLM systems was pasted and tested using the code shown in Figure 2, which includes the assertion tests and the test lists to check if the AI-generated code passes or fails the assertion tests. The assertion test was designed to generate the total results of the tests with a score of between 3 and 6 points, where one point was given if the test succeeded and 0 if it failed. The final score for all 100 of the tested programs was 305 points if a 100% pass rate for the tests was recorded.

This process was repeated for all five LLMs with the same prompt. Every test passed was scored one point for each test in the list for the 100 prompts, and then the final score was calculated and compared.

At the second stage of the evaluation process, the lowest scoring LLMs were retested on an equal number of tasks that they did not complete successfully or did not complete 100% of the tests correctly. The goal here was to confirm whether the system was capable of generating the correct code when provided enough feedback from the human user.

At the third stage, the same prompt was provided to the models in a new conversation to remove any history. Some of the tasks were completed on the first retry, as LLMs tend to be creative and generate different code, while others needed extra feedback. After every response, the code was tested, and feedback was provided to the model, including the results of the test. The number of messages input into the models was also counted until the correct code was provided by the AI system. The maximum number of attempts used was set to 10 attempts; after that, testing would stop, and the task would be marked as failed.

## 5. Experimental Results

The test was performed using 460 Python problems that were certified by Google [7], employing all five of the above-mentioned LLMs, which added up to a total possible score of 1225. It can be considered that a small number of tests would be enough to provide a clear trend in performance.

The obtained results are shown in Figure 3; the Claude model displayed the worst performance, scoring 875 points, equivalent to 71.43%. Google's Bard model performed similarly to Claude, scoring 933 points, or 76.16%. While these models are generally good at understanding the tasks, they have a lot of room for improvement.

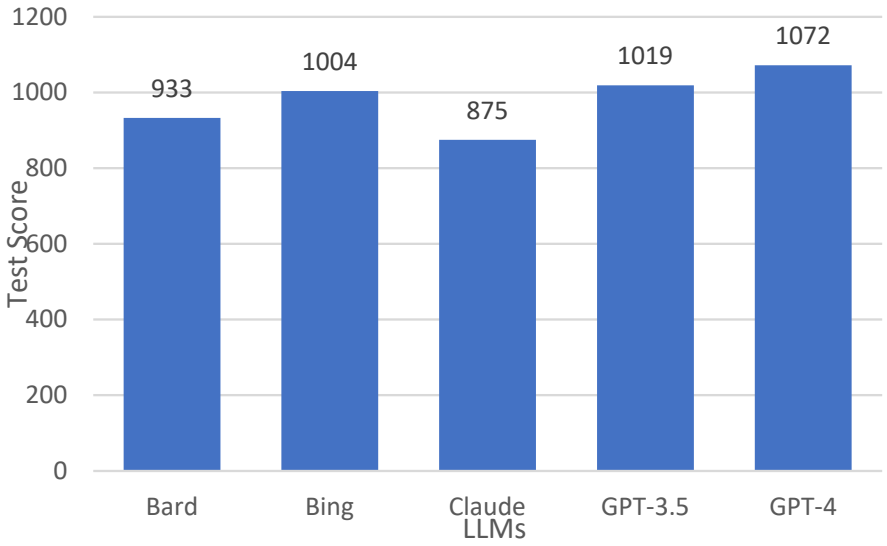


**Figure 3.** Performance comparison between the different LLMs.

As for the GPT-based models, the GPT-3.5 model scored an impressive 1019 points, equivalent to 83.18%. Similarly, Bing only scored 15 points lower, with 1004, which is equivalent to 81.96%. Bing scoring similarly to GPT-3.5 was definitely not expected, as Bing is based on the GPT-4 model. GPT-4, however, scored an impressive 1072, or 87.51%.

These results show that while each model has impressive code generation capabilities, GPT-4 particularly stands out with its highest success rate, closely followed by Bing and GPT-3.5.

### *5.1. Code Quality*

One thing that stood out during testing was that the non GPT-based models typically generate longer code, while GPT-based models tend to generate much shorter, more concise code. There were instances of Bard outputting code that when run, entered an infinite loop and the process had to be aborted, while GPT-based models were much more efficient.

One example is task−id 45. The prompt was "Write a python function to find the maximum difference between any two elements in a given array". Bard's generated output was more complicated and less efficient, as it used the basic programming skills which were used by nonprofessional basic programming skills; the results are shown in Figure 4a. Meanwhile, ChatGPT-4 generated correct compact code using readily available methods in Python, as shown in Figure 4b.

```python
# Bard Generated Code
def max_Abs_Diff(arr):
  """Finds the maximum difference between any
     two elements in a given array."""
  max_diff = -float("inf")
  min_val = float("inf")
  for num in arr:
    if num < min_val:
      min_val = num
    elif num - min_val > max_diff:
      max_diff = num - min_val
  return max_diff
```

(**a**)

```python
# ChatGPT-4 code genegaretd
def max_Abs_Diff(arr):
  """Finds the maximum difference between any two elements
     in a given array."""

    return max(arr) - min(arr)
```

(**b**)

**Figure 4.** Task−id 45 code generated by LLM. (**a**) Bard; (**b**) ChaGPT 4.

Another example is task−id 101. The prompt was "Write a function to find the *k*th element in the given array using based indexing". Bard's output was relatively long and resulted in a complex loop. On the other hand, GPT-4's responses are much more concise and made better use of built-in functions, as shown in Figure 5.

In order to compare the quality of the generated code for the five LLMs, the average number of lines of code (LOC) was calculated for the whole of the tested dataset, and the results are shown in Figure 6. As noticed from the results, across all of the LLMs, Bard returned the highest average number of LOC, which indicates lower code quality, while Bing produced a higher quality of code compared to Chat GPT-4 and Claude.

```python
# Bard Generated Code
def kth_element(array, k):
  """Finds the kth element in the given array using 1-based indexing."""
  if k > len(array):
    raise ValueError("k is out of bounds")
  def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
      if array[j] <= pivot:
        i += 1
        (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1
  i = 0
  while i < len(array):
    j = partition(array, i, len(array) - 1)
    if j == k - 1:
      return array[j]
    elif j < k - 1:
      i = j + 1
    else:
      k = k - j - 1
```

(**a**)

```python
# ChatGPT-4 Generated Code
def kth_element(arr, k):
    """
    Return the kth element in the given array using 1-based indexing.
    """
    if k <= 0 or k > len(arr):
        raise ValueError("Invalid index value")
    return arr[k-1]
```

(**b**)

**Figure 5.** Task−id 101 code generated by LLM. (**a**) Bard; (**b**) ChaGPT 4.

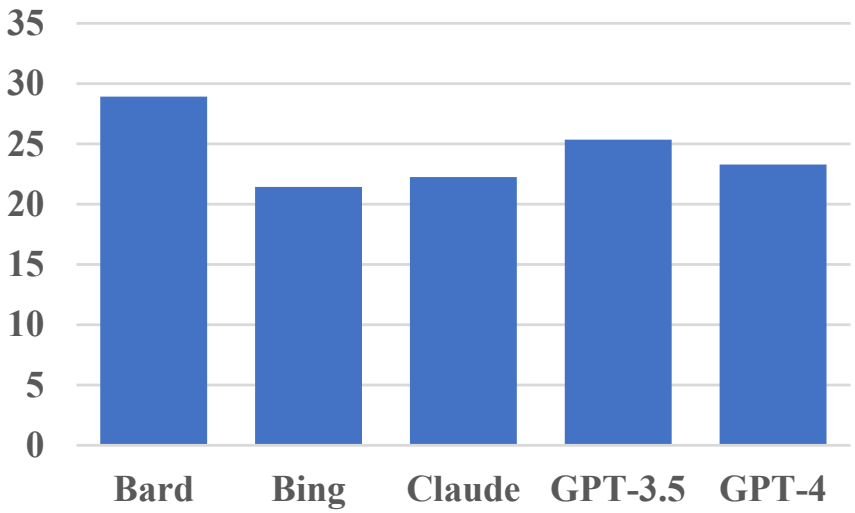


**Figure 6.** Average number of lines of code for each model, excluding blank lines, tested for the whole dataset.

### *5.2. Providing Feedback to the Model*

In the second phase of the tests, the highest-scoring model, GPT-4, and the lowest-scoring model, Bard, were tested again on previously failed tasks. This time, feedback was provided to the LLM models to comprehend how well they could understand the errors and try to correct them. It was found that out of the 16 tasks that GPT-4 did not complete on the first try in the previous phase, GPT-4 was able to complete 14, while Bard was only able to complete 5 tasks after feedback was provided. Figure 7 compares the results of both GPT-4 and Bard.

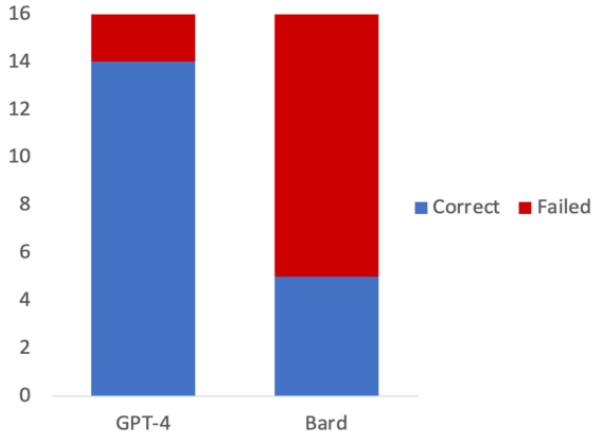


**Figure 7.** Comparison of GPT-4 and Bard's score when providing feedback.

These tests showed that there is a significant difference in the way both models processed and applied feedback to the generated code. GPT-4 demonstrated an impressive understanding of the provided feedback and effectively incorporated fixes into the code, meaning that with each output, the code improved until it performed the desired task with no errors.

On the other hand, Google's Bard, while sometimes able to generate code effectively, did not exhibit the same level of comprehension of the provided feedback as GPT-4. Bard's responses typically acknowledged the error; however, when generating a "fixed" version of the code, it generally just rewrote the exact same code again.

Overall, the results indicate that the AI language models, especially GPT-4, demonstrate proficient abilities for code generation. Additionally, it showed adaptability and the capability of self-improvement when faced with real-world coding scenarios, including bug fixing. In the case of GPT-4, when properly used, it could solve almost 100% of all tasks, showcasing its outstanding ability and potential to be a coding assistant. On the other hand, the second phase of the testing shows that LLMs are best as coding assistants and not as replacements for programmers. These models benefit greatly from feedback from humans and significantly improve when proper feedback is provided.

These results also suggest that LLMs have transformative potential in programming and software development, offering researchers and developers insights into the ever-growing field of AI-assisted programming.

### *5.3. Limitations*

While LLMs have demonstrated overall effectiveness at writing code, they also have their limitations. Firstly, LLMs sometimes tend to output inconsistent answers [31]. For example, for task_id 11, "Write a Python function to remove the first and last occurrence of a given character from the string", GPT-4 generated two different outputs when the test was repeated with small changes in the prompt. The resulting generated codes are shown in Figure 8.

```python
# ChatGPT-4 Generated Code Task Id -11
# First Solution
def remove_first_and_last_occurrence(s, char):
    # Remove the first occurrence
    s = s.replace(char, '', 1)

    # Reverse the string, remove the first occurrence (which is the last occurrence in the original string)
    # and then reverse it back
    s = s[::-1].replace(char, '', 1)[::-1]

    return s
```

(**a**)

```python
# ChatGPT-4 Generated Code Task Id -11
# Second Solution
def remove_first_last_occurrence(s, char):
    first_index = s.find(char)
    last_index = s.rfind(char)

    # If the character is not in the string or only appears once, return the original string.
    if first_index == -1 or first_index == last_index:
        return s

    # Remove the first occurrence.
    s = s[:first_index] + s[first_index+1:]

    # Adjust the last index because we removed a character.
    last_index -= 1

    # Remove the last occurrence.
    s = s[:last_index] + s[last_index+1:]

    return s
```

(**b**)

**Figure 8.** Task−id 11 code generated by GhatGPT-4. (**a**) Solution 1. (**b**) Solution 2.

In this example, both functions work as expected and pass the tests. However, this will not always be the case. This introduces an element of randomness, not knowing if the code will be correct or not. This can mean that the user has to spend valuable time testing and correcting the code if it is not correct.

This brings us to another limitation: in order to use ChatGPT and LLMs efficiently, it is very likely that human feedback will play a key role. Users have to be able to effectively communicate feedback to the model in a way that the model will understand and learn from, which might also be time- and resource-consuming.

Third, the programming style of the models can differ from that of the users. Every programmer has their own preferences when writing code, and the model may not always follow the same format. This impacts the readability of the code, as well as potentially introducing bugs into the software and affecting future maintainability.

Fourth, the provided code by the models may have compatibility issues with the rest of the code that will be used. The models generate code based on what they were trained on and do not know the entire context in which it will be implemented later on; therefore, the code may lead to bugs and compatibility issues, which in turn will consume resources in order to refactor the provided code.

## 6. Conclusions and Future Work

This research work, evaluating the GPT (GPT-3.5 and GPT-4) models, as well as other large language Models (LLMs) from competitors (Bard, Bing, and Claude) using the Mostly Basic Python Problems dataset for code generation tasks, has provided interesting results. These results show some insights into the strengths of these models, as well as areas in which they could improve. Hopefully, this research serves as an empirical demonstration of the current state-of-the-art (SOTA) models in software development. Of all the models tested, GPT-4 exhibited the highest proficiency in code generation tasks, achieving a success rate of 86.23% on the small subset that was tested. GPT-based models

performed the best compared to the other two models, Bard and Claude. These two models performed the worst, indicating a need for improvement in terms of coding and code generation, considering they are the two most recently released models. It was also found that LLMs had some difficulties concerning code generation, which led to bugs and errors and multiple solutions when a slight change or wrong query was provided. This can lead to the inefficient use of time by programmers. Overall, it was found that ChatGPT and other LLMs, most of the time, are able to generate code effectively and can be used as programming assistant tools, but they are not replacements for human software developers since they require constant feedback and monitoring by a human.

While the results were promising, there is room for future research. More studies can focus on evaluating the generated code itself. New metrics can be considered, such as the cleanliness of the code, execution time, resource usage, and more. Besides the MBPP, more complex tasks can be used to test the LLMs and see how they handle more complicated tasks and code. In addition, future research can also focus on real-world applications. They can examine the use of these LLMs in professional software development processes besides coding, to identify any other practical uses that can essentially enhance productivity in the industry and potentially revolutionize it.

**Author Contributions:** Conceptualization, R.K. and C.E.A.C.; methodology, R.K., C.E.A.C. and M.N.A.; validation, R.K.; formal analysis, C.E.A.C.; investigation, R.K. and C.E.A.C.; data curation, C.E.A.C.; writing—original draft preparation, C.E.A.C.; writing—review and editing, R.K.; visualization; R.K. and M.N.A.; project administration, R.K. and M.N.A.; funding acquisition, C.E.A.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** The research presented in this paper "Effectiveness of ChatGPT in Coding: A Comparative Analysis of Popular Large Language Models.", has not received any external funding or financial support. The authors have independently conducted and completed this work without the assistance of grants, scholarships, or other financial contributions. The absence of funding does not diminish the rigor or validity of the research findings presented herein.

**Data Availability Statement:** The data sets used by this article are available for download for GitHub the following link: https://github.com/google-research/google-research/tree/master/mbpp.

**Conflicts of Interest:** The authors declare no conflict of interest regarding the publication of this paper, "Effectiveness of ChatGPT in Coding: A Comparative Analysis of Popular Large Language Models". There are no financial, personal, or professional relationships that could influence the objectivity or integrity of the research conducted or the conclusions drawn in this study.

## References

1. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
2. Fan, L.; Li, L.; Ma, Z.; Lee, S.; Yu, H.; Hemphill, L. A bibliometric review of large language models research from 2017 to 2023. *arXiv* **2023**, arXiv:2304.02020.
3. Ni, A.; Iyer, S.; Radev, D.; Stoyanov, V.; Yih, W.T.; Wang, S.; Lin, X.V. Lever: Learning to verify language-to-code generation with execution. In Proceedings of the International Conference on Machine Learning 2023, Honolulu, HI, USA, 23–29 July 2023; PMLR: Westminster, UK; pp. 26106–26128.
4. OpenAI; Pilipiszyn, A. GPT-3 Powers the Next Generation of Apps. 2021. Available online: https://openai.com/blog/gpt-3-apps/ (accessed on 26 November 2023).
5. Hardesty, L. Explained: Neural Networks, MIT News. Massachusetts Institute of Technology. 2017. Available online: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (accessed on 25 July 2023).
6. Zaremba, W.; Brockman, G.; OpenAI. OpenAI Codex. 2021. Available online: https://openai.com/blog/openai-codex/ (accessed on 23 November 2023).
7. Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. Program Synthesis with Large Language Models. *arXiv* **2021**, arXiv:2108.07732.
8. Pearce, H.; Tan, B.; Ahmad, B.; Karri, R.; Dolan-Gavitt, B. Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs? *arXiv* **2021**, arXiv:2112.02125.
9. Vaithilingam, P.; Zhang, T.; Glassman, E.L. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In Proceedings of the Chi Conference on Human Factors in Computing Systems Extended Abstracts, New Orleans, LA, USA, 29 April–5 May 2022; pp. 1–7.

10. 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. Evaluating large language models trained on code. *arXiv* **2021**, arXiv:2107.03374.
11. Xu, F.F.; Alon, U.; Neubig, G.; Hellendoorn, V.J. A systematic evaluation of large language models of code. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022; pp. 1–10.
12. Zan, D.; Chen, B.; Zhang, F.; Lu, D.; Wu, B.; Guan, B.; Yongji, W.; Lou, J.G. Large language models meet NL2Code: A survey. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics 2023, Toronto, ON, Canada, 9–14 July 2023; Volume 1, pp. 7443–7464.
13. Basheer, I.A.; Hajmeer, M. Artificial neural networks: Fundamentals, computing, design, and application. *J. Microbiol. Methods* **2000**, *43*, 3–31. [CrossRef] [PubMed]
14. Kingma, D.P.; Welling, M. Auto-encoding variational bayes. *arXiv* **2013**, arXiv:1312.6114.
15. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is All You Need. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2017; p. 30.
16. Azzouni, A.; Pujolle, G. A long short-term memory recurrent neural network framework for network traffic matrix prediction. *arXiv* **2017**, arXiv:1705.05690.
17. Sherstinsky, A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Phys. D Nonlinear Phenom.* **2020**, *404*, 132306. [CrossRef]
18. Peng, S.; Kalliamvakou, E.; Cihon, P.; Demirer, M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv* **2023**, arXiv:2302.06590. Available online: http://arxiv.org/abs/2302.06590 (accessed on 20 August 2023).
19. GitHub. Copilot Your AI Pair Programmer. Available online: https://github.com/features/copilot (accessed on 13 August 2023).
20. Poldrack, R.A.; Lu, T.; Beguš, G. AI-assisted coding: Experiments with GPT-4. *arXiv* **2023**, arXiv:2304.13187. Available online: http://arxiv.org/abs/2304.13187 (accessed on 23 July 2023).
21. NVIDIA. What Is Generative AI? 2023. Available online: https://www.nvidia.com/en-us/glossary/data-science/generative-ai/ (accessed on 23 July 2023).
22. Shanahan, M. Talking about large language models. *arXiv* **2022**, arXiv:2212.03551.
23. Wang, K.; Gou, C.; Duan, Y.; Lin, Y.; Zheng, X.; Wang, F.Y. Generative adversarial networks: Introduction and outlook. *IEEE/CAA J. Autom. Sin.* **2017**, *4*, 588–598. [CrossRef]
24. Elastic. What Is Generative AI? | A Comprehensive Generative AI Guide. 2023. Available online: https://www.elastic.co/what-is/generative-ai (accessed on 23 July 2023).
25. Pichai, S. An Important Next Step on Our AI journey, Google. 2023. Available online: https://blog.google/technology/ai/bard-google-ai-search-updates/ (accessed on 12 August 2023).
26. Google AI. Google AI PaLM 2. Available online: https://ai.google/discover/palm2/ (accessed on 12 August 2023).
27. Microsoft. Microsoft and OpenAI Extend Partnership. *The Official Microsoft Blog*. 23 January 2023. Available online: https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/ (accessed on 12 August 2023).
28. Mehdi, Y. Reinventing Search with a New AI-Powered Microsoft Bing and Edge, Your Copilot for the Web. *Official Microsoft Blog*. 7 February 2023. Available online: https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/ (accessed on 12 August 2023).
29. Anthropic. Introducing Claude. 2023. Available online: https://www.anthropic.com/index/introducing-claude (accessed on 12 August 2023).
30. Broadway, M. Who Is Anthropic? The Company behind Claude AI. PC Guide. 17 July 2023. Available online: https://www.pcguide.com/apps/who-is-anthropic/ (accessed on 12 August 2023).
31. Koga, S. *Exploring the Pitfalls of Large Language Models: Inconsistency and Inaccuracy in Answering Pathology Board Examination-Style Questions*; published in medrxiv; Cold Spring Harbor Laboratory: Cold Spring Harbor, NY, USA, 2023.