Bachelor Degree Project

UNIVERSITY
OF SKÖVDE

Assessing the Effectiveness of ChatGPT in Generating Python Code

Bachelor's Degree Project in Information Technology
Basic level 30 ECTS
Spring 2023

Victor Adamson, Johan Bägerfeldt

Supervisor: Sören Richard Stahlschmidt
Examiner: Juhee Bae

Summary

This study investigated ChatGPT's Python code generation capabilities with a quasi-experiment and a case study, incorporating quantitative and qualitative methods respectively. The quantitative analysis compared ChatGPT-generated code to human-written solutions in terms of accuracy, quality, and readability, while the qualitative study interviewed participants with varying levels of programming experience about the usability of ChatGPT for code generation. The findings revealed significant differences in quality between AI-generated and human-written solutions but maintained overall similarities in accuracy and readability. The interviewees reported that ChatGPT showed potential for generating simple programs but struggled with complex problems and iterative development, though most participants were optimistic about its future capabilities. Future research could involve larger samples, more programming languages, and increased problem complexities.

Keywords: Artificial Intelligence, Code Generation, ChatGPT, Python

Contents

1 | Introduction

Automatic code generation is an emerging field, significantly advanced by ChatGPT's introduction on November 30, 2022 (OpenAI 2022). This language model has the capability of generating code from natural language descriptions, allowing users to create simple programs even without previous programming experience (McManus 2023). Due to the recency of this development, there is currently a lack of research on ChatGPT, especially concerning code generation. It is therefore important to determine how well it can solve programming problems and identify potential limitations, informing improvements and future research.

The aim of the study is to analyze the effectiveness of ChatGPT in generating Python code by employing both quantitative and qualitative methods. The quantitative study evaluates ChatGPT's ability to generate Python code by using questionnaires to score its effectiveness through three variables: accuracy, quality, and readability. In this context, accuracy is defined as the extent to which the generated code aligns with the task described in the prompt; quality measures how well-written the code is; readability refers to the ease with which users can read and understand the generated code. The qualitative study interviewed participants of different skill levels, who collaborated with ChatGPT in solving programming assignments, to measure the usability of ChatGPT for Python code generation. The participants' thoughts and opinions on the process provided qualitative results on the usefulness of natural language programming and its potential for simplifying programming for developers.

Through a thorough analysis of ChatGPT's capabilities in generating Python code, this study aims to expand on earlier research on AI code generation. The findings will offer guidance for the future development of AI models in the field of code generation and inform specifications for effective collaboration between human programmers and AI assistants. By combining quantitative and qualitative approaches, the study offers a comprehensive assessment of ChatGPT and the implications of AI-generated code on the landscape of software development.

2 | Background

2.1 Artifical Intelligence

Artificial intelligence (AI) is a broad term encompassing the use and development of computer systems that mimic human cognitive processes and actions to perform tasks typically requiring human intelligence, including decision-making, problem-solving, and intelligent behavior. Early AI models were primarily built with hard-coded statements, but advances in learning algorithms have allowed machine learning (ML) techniques to employ artificial neural networks, which mimic the connections of the neurons in the brain. This allows for iterative learning from training data, which gradually adjusts the connections based on the correctness of the outputs. The AI model can thereby learn to identify patterns and apply its knowledge to new problems that it has not been explicitly trained on. These improved learning methods offer superhuman performance in certain closed environments, outperforming human abilities. (Russell 2010, Janiesch et al. 2021)

2.2 Large Language Models

Large language models (LLMs) utilize ML to process and generate text. The first major success of very large-scale self-supervised neural network learning in 2018 propelled the field of natural language processing forward by allowing language models to learn by reading. The models could therefore be trained on texts totaling billions of words. To learn patterns, the AI models create their own prediction tasks by for example masking a word in a sentence and attempting to deduce the missing word. This iterative training process may repeat billions of times. Additionally, LLMs can be adapted to many different tasks related to language processing with extra guidance (Wiggers 2022, Manning 2022).

2.3 Natural Language to Code

LLMs can be used for predicting code, allowing the model to generate new source code from a natural language description of the problem. One of the largest initial models for code generation, OpenAI's Codex, was deployed for use in GitHub Copilot as an AI pair programmer (Xu, Alon, Neubig & Hellendoorn 2022, GitHub n.d.). This form of code synthesis goes beyond earlier forms of code completion, which involved the auto-completion of a single line of code, by generating new source code to comprehensively solve a given problem based on the provided context. This emerging capability of language models has the potential to offer significant benefits for programmers by streamlining the coding process, facilitating the breakdown of complex problems into smaller, more manageable parts, and enabling a more effective utilization of existing code libraries, APIs, and functions (OpenAI 2021*b*). As a result, language models not only enhance

the efficiency of software development tasks but also open up possibilities for further innovations in the programming landscape.

2.4 ChatGPT

ChatGPT, a derivative of OpenAI's GPT series using the GPT-3.5 model, is an advanced AI language model that exhibited a previously unparalleled ability to understand and generate human-like responses (OpenAI 2022). Since its launch on November 30, 2022, ChatGPT experienced a surge in popularity, capturing the attention of not only the tech community but also the general public. ChatGPT attracted 100 million unique visitors in January 2023 alone, with over 186 million unique visitors by March 2023 (Similarweb 2023).

An article by McManus (2023) describes how ChatGPT can be used to create simple games without any prerequisite programming knowledge. According to the author, ChatGPT can successfully create games such as Pong, Breakout, and Asteroids in 40 seconds, which would have taken an experienced developer half an hour. Furthermore, it was possible to recreate a game whose source code was not available online and therefore unlikely to be part of its training data.

2.4.1 Limitations of ChatGPT

While ChatGPT offers numerous benefits, it has certain limitations that affect its effectiveness: a tendency to guess user intent instead of seeking clarification, sensitivity to input phrasing and repetition, and plausible-sounding but incorrect or nonsensical answers. These incorrect answers are typically referred to as "hallucinations", which arise due to the lack of a source of truth (OpenAI 2022, Maynez et al. 2020). Hallucinations may be particularly problematic when generating code for precise requirements, as such inaccuracies would lead to code that is not functional or does not fulfill the intended purpose, and may necessitate additional development time correcting and validating the generated code.

Alkaissi & McFarlane (2023) found that ChatGPT could occasionally generate factually correct scientific content on certain medical topics; however, it also confidently wrote unverified information that could not be verified by the researchers and cited non-existent papers when prompted for sources. The authors also attempted to apply ChatGPT for checking recurrent references in a text and although it was unable to solve the task directly, it successfully wrote Python code to solve the same issue when prompted to do so.

A study by Frieder et al. (2023) found that ChatGPT performed poorly on most mathematical problems, with the score decreasing as the difficulty increased. While occasionally producing high-quality solutions, ChatGPT performed best on simple logic questions. The main difficulty lay in following the constraints of unusual puzzles, such as changing the color of the squares of a chessboard, where ChatGPT failed to cover the entire board and only ended up changing five squares. Additionally, in rare displays of lower confidence, ChatGPT would sometimes emphasize the difficulty of questions if it believed that they required complicated mathematics, even in cases where they were solvable by elementary techniques.

ChatGPT's effectiveness is further influenced by the phrasing of user input. It may provide different answers when presented with a question phrased differently or entered multiple times due to its non-deterministic nature. As a result, a user might be led to think that ChatGPT is incapable of providing the correct answer, even when it is possible. This inconsistency creates a need for users to modify their query's phrasing to receive the most accurate response. Due to this sensitivity in phrasing, users might need to experiment with various query styles to get code

that meets their requirements. This increases the time taken to obtain an accurate code snippet and reduces overall efficiency (OpenAI 2022).

When faced with ambiguous queries, an ideal model would ask clarifying questions to better understand the user's intent; however, ChatGPT tends to make assumptions about the user's intentions instead of seeking clarification. This limitation leads to potentially inaccurate generated code that does not align with user expectations (OpenAI 2022). In such cases, it would be necessary to alter the query or provide more context to achieve desired results, again increasing the time spent on obtaining relevant code.

2.5 Prompts

A prompt, in the context of language models and ChatGPT, refers to the input or query provided by a user, which serves as the starting point for the model's response generation process. Prompts can vary in complexity, ranging from simple words, phrases, or questions to more elaborate and detailed instructions. As these models have been trained on vast amounts of text data, they possess the ability to function as powerful language processors, adept at understanding and responding to a diverse array of user inputs. The manner in which prompts are constructed significantly influences the model's output, making it crucial to provide clear and contextually rich information to derive relevant and accurate responses (Jiang et al. 2022, OpenAI 2022, Manning 2022).

2.5.1 Types of Prompts

There are two general methods for writing prompts: few-shot and zero-shot. The fundamental difference between these methods lies in the number of provided examples. Few-shot prompting offers the AI multiple examples of similar outputs, while zero-shot provides no examples at all. An example of few-shot prompting would be "English: welcome French: bienvenue English: hello French: ", while a zero-shot prompt would be "The translation of "hello" into French is: " (Jiang et al. 2022).

In a study conducted by Reynolds & McDonell (2021), it was discovered that zero-shot prompts can outperform few-shot prompts. The authors argue that semantic contamination may result from the examples provided in few-shot prompts. Furthermore, they assert that few-shot prompts primarily serve the purpose of task location, rather than facilitating the learning of the task itself. A natural language description of a task, without any examples or prior knowledge, is considered a zero-shot prompt, as defined by Brown et al. (2020). This type of prompting encourages the AI to generate output based solely on its comprehension, without relying on previous examples of similar tasks.

2.6 HumanEval

The HumanEval data set (OpenAI 2021*a*) contains 164 few-shot prompts of original, hand-written programming problems, each containing a function signature and docstring explaining the intended functionality with examples of inputs and correct outputs. In addition, each problem has its corresponding human-written canonical solution as well as unit tests. It was created by Chen et al. (2021) for use in measuring functional correctness when generating code with Codex and other AI models, including GPT-3 and GPT-J.

2.7 Pair Programming

Pair programming involves a duo of programmers working together on a single computer, with one person taking the role of the driver who is writing code, and the other taking the role of the observer who identifies defects and aids in brainstorming and guiding the strategic direction of the work (Williams 2001). The benefit of this practice is the overall increase in quality, with greater design quality, fewer defects, and better communication within the team, at the cost of requiring two programmers to do the job of one; however, the increase in brainstorming capability and reduction in defects may save more time that would otherwise be required to debug issues (Cockburn & Williams 2001).

2.8 Grounded Theory

Grounded Theory, a research methodology developed by Glaser & Strauss (1967), focuses on conceptualizing data, such as qualitative data from interviews, to generate theories based on real-life observations. The framework, as described by (Tie et al. 2019), involves a recursive process starting with purposive sampling, where relevant participants are selected to address the research question.

Semi-structured interviews are then conducted to collect data. Each interview is analyzed before proceeding to the next participant. This process is repeated until all participants have been interviewed. The researcher continuously analyzes and codes the data from the start. In Grounded Theory, "coding" means extracting significant aspects of the data into categorizable elements called "codes". For example, "Better performance with less specific tasks" refers to ChatGPT's applicability. The first stage, "initial coding", identifies underlying features of the data using codes. The researcher then uses categories to comprehend the data's meaning.

Next, "theoretic sampling" adjusts the participant sampling based on codes, exploring relationships and gaps within the data. After categorizing all codes, "intermediate coding" determines which categories can be combined or refined. Eventually, a core category is chosen, and the data is abstracted into concepts. Finally, "advanced coding" integrates the developed theory into the final "grounded theory". This crucial stage yields a comprehensive grounded theory that explains a process or scheme associated with a phenomenon, serving as the conclusive outcome of the study (Birks & Mills 2015).

3 | Problem

Due to the recent introduction of ChatGPT, there is a lack of research on its code generation capabilities. Although automatic code generation has the potential to revolutionize the way developers interact with code in a more accessible approach, it yet remains largely untested and unverified in its effectiveness. As a result, conducting a thorough assessment of the performance and capabilities of code generation with ChatGPT is both timely and vital to understanding its true potential and identifying areas for improvement and further research in this promising domain. The aim of the study is therefore to analyze the effectiveness of ChatGPT in generating Python code, to determine its usability for automatic code generation by developers of varying programming experience levels.

The implementation of a dialogue-based query capability was previously suggested by Xu, Vasilescu & Neubig (2022), in order to allow users to provide feedback to the AI model and thereby iteratively refine the generated code. This capability could be considered fulfilled with the dialogue-based interface of ChatGPT, making it a relevant area to investigate. This type of pair programming with artificial intelligence could involve a dynamic synergy between a human programmer and an AI system in the process of generating code, where the AI system generates code as the driver, with the human programmer focusing on reviewing the code, locating bugs, and making high-level strategic decisions.

This study focuses on code generation with the Python programming language, as it has been the basis of earlier works by Chen et al. (2021) and Yetistiren et al. (2022) in combination with the HumanEval data set (OpenAI 2021*a*). It was also chosen for its simple syntax structure, which is particularly conducive to evaluating the effectiveness of ChatGPT in generating comprehensible code. The intention is therefore to expand on earlier research on Codex and GitHub Copilot by investigating ChatGPT in a similar approach.

3.1 Research Question

	Quasi-Experimental Study	Case Study
Independent variables	Code source Prompt	Participant expertise
Dependent variables	Accuracy Quality Readability	Usability

Table 3.1: Independent and dependent variables

Table 3.1 presents the selected variables for each study. Unlike previous studies, the decision was made to forego unit tests in favor of subjective assessment by humans to capture broader opinions on the effectiveness of the generated code. The variables are therefore not previously defined in earlier research, although the selection was motivated by earlier findings. The variables were defined by the authors to capture multiple aspects of code correctness and user experiences. The three dependent variables for the quasi-experimental study were defined as follows: accuracy reflects how closely the generated code corresponds to the task described in the prompt, quality measures how well-written the code is, and readability denotes the ease with which the reader can read and comprehend the produced code.

The case study investigates the usability of ChatGPT as perceived by participants of varying programming expertise. This variable covers the subjective experiences of solving programming assignments with the use of AI-generated code. Usability, together with the three earlier mentioned variables, make up the basis of the research question for this study.

RQ: How effective is ChatGPT in generating Python code in terms of accuracy, quality, readability, and usability?

3.1.1 Hypotheses

Hypothesis 1: There is a significant difference in accuracy between ChatGPT-generated Python code and the human-written solutions.

Hypothesis 2: There is a significant difference in quality between ChatGPT-generated Python code and the human-written solutions.

Hypothesis 3: There is a significant difference in readability between ChatGPT-generated Python code and the human-written solutions.

In addition to the hypotheses, a sub-question to the research question is posed to explore and assess the usability aspect of ChatGPT in the context of solving programming assignments. The sub-question for this study is: 'How do participants perceive the usability of ChatGPT for solving programming assignments?' This question aims to capture users' impressions, opinions, and reflections on the application of ChatGPT in this specific setting.

3.2 Objectives

A set of objectives was decided upon for each sub-study, detailing the planned procedure. As the sub-studies were conducted separately from each other, both contain independent objectives regarding sample and problem selection. Although both studies rely on assessments by people with programming experience, the required levels differ depending on the purpose, which is expanded on in the Methodology section. In the final step of each study, the results are compared to the other study and to previous work to relate the findings and identify differences and similarities.

3.2.1 Quasi-Experimental Study

1. Select prompts with human-written solutions
2. Generate code for each prompt
3. Write a questionnaire to rate the output code and the human-written solutions

4. Select a sample of recipients with Python experience
5. Share the questionnaire
6. Analyze the results
7. Compare findings

3.2.2 Case Study

1. Write programming assignments
2. Prepare semi-structured interview questions
3. Select a sample with varying programming experience
4. Send the assignment descriptions
5. Conduct the interviews
6. Transcribe the interviews
7. Analyze the results
8. Compare findings

4 | Related Work

4.1 Studies with HumanEval

Previous studies with the HumanEval data set include studies by Chen et al. (2021) and Yetistiren et al. (2022). Chen et al. (2021) evaluated the effectiveness of code generation with the Codex model by comparing it to previous models such as GPT-3 and GPT-J. The Codex model reached a 28.8% solve ratio, which increased to 70.2% with repeated sampling by generating 100 samples per problem.

The study by Yetistiren et al. (2022) used HumanEval to evaluate GitHub Copilot, which uses the Codex model to suggest code inside the editor (GitHub n.d.). They evaluated validity, correctness, and efficiency with automated tests, employing the Python 3.8 interpreter, HumanEval unit tests, and the OpenAI API as tools. Their findings show that 28.7% of solutions were generated accurately, which is nearly identical to the findings of Chen et al. (2021), as both use the same model. The authors also showed that including partially correct solutions increases the success rate to 91.5%, of which 51.2% were partially correct, 11.6% were "valid incorrect", and 8.5% were "invalid incorrect". They suggest that GitHub Copilot is a promising tool; however, they acknowledge that a more comprehensive evaluation is necessary for future studies. This motivated the choice of two different approaches, utilizing qualitative and quantitative data, to assess the generated code more comprehensively.

4.2 Other Approaches

A study on the Codex model by Finnie-Ansley et al. (2022) reported that its performance ranked competitively within the upper quartile among 71 CS-1 students when solving first-year assignments; however, it should be noted that computer science students have historically had high attrition rates (Beaubouef & Mason 2005). Of the 23 questions, 19 were solved in under 10 attempts, with 10 needing just one attempt. Despite exhibiting drawbacks related to compliance failure with certain code generation constraints, such as barring the use of specific functions, the authors posited that rapid advancements in the code generation model will likely overcome these limitations. It is therefore of interest to analyze the accuracy of code generated by ChatGPT, to determine how well it is able to identify the problem and respect such constraints.

Moreover, Codex's capability to understand and describe code has been demonstrated in various studies, such as the one conducted by Sarsa et al. (2022). In their research, the authors showcased the utility of Codex in generating programming exercises from code snippets. This level of perceived understanding emphasizes the relevance of investigating related factors, such as comment generation, motivating the inclusion of a readability variable in the quasi-experimental study to determine how comprehensible the generated code is.

A study conducted by Heyman et al. (2021) aimed to develop a model for natural language-guided programming focusing on Python, given its widespread utilization among data scientists, data engineers, and machine learning researchers. In their research, they identified several factors that warrant further investigation. These aspects include evaluation metrics, impact on productivity, code quality, and the learning curve involved. The authors suggest that the most effective approach to assessing generated code currently involves enlisting human experts to rate the code based on its relevance to the assigned task, motivating the relevance of human assessment by programmers in both studies.

Xu, Vasilescu & Neubig (2022) conducted a study in which they developed two natural language to code models that participants utilized when solving various Python programming tasks. While using the two tools proved to be an enjoyable experience for the participants, no statistically significant differences were observed in either completion speed or code correctness. The authors hypothesized that this result could be attributed to two factors: the crude approximations of the metrics used and the participants' reliance on search engines when writing code manually. The multifaceted dimensions of code quality pose challenges to measurement, potentially rendering the ratings less accurate. This motivated the choice of the quality variable for use in the quasi-experimental study, in order to determine how well-written the generated code was through human assessment.

In their study, Vaithilingam et al. (2022) compared the performance of automatic code generation with Copilot to writing code with Intellisense, the default code completion plugin in the IDE Visual Studio Code. Their findings revealed that users employing Copilot completed tasks more quickly than those using Intellisense, though they finished a fewer number of tasks. Participants reported a preference for Copilot over Intellisense and perceived the generated code as more helpful; however, notable drawbacks were identified, such as difficulties in understanding the generated code. When encountering issues, users often had to start over from scratch as the code failed to run properly. This situation led to increased debugging time, ultimately rendering the time difference between the two groups statistically insignificant. These findings motivated the usability variable for the case study to determine how participants perceive the effectiveness of ChatGPT for solving different programming assignments.

A noteworthy suggestion posited by Xu, Vasilescu & Neubig (2022) is the integration of a dialogue-based query capability, which allows users to iteratively refine and clarify their prompts by providing real-time feedback. This feature is of considerable relevance when examining the functionality and use of ChatGPT. Being a language model capable of engaging in dynamic conversations, ChatGPT possesses the potential to incorporate such a dialogue-based query system, allowing users to interact with the model iteratively. This would not only enhance prompt clarification and specificity, but it would also facilitate better code generation outcomes that more closely align with user requirements. This motivated the choice to investigate the effectiveness of code generation with ChatGPT.

In continuation of previous research, this study intends to expand upon the existing knowledge of automatic code generation by investigating ChatGPT, using human assessment and the newly defined variables accuracy, quality, readability, and usability to capture a broader perspective on code correctness beyond the functional correctness criteria seen in most previous research in this area. The intention is therefore to gain insights into the user experience and the strengths and weaknesses of the generated code.

5 | Methodology

The aim of this study was to assess the effectiveness of ChatGPT in generating Python code. To achieve a holistic understanding, our methodology combines both quantitative and qualitative methods in two distinct yet complementary studies. The two sub-studies explore different aspects of code generation with ChatGPT, using separate samples of tasks and participants, as the quantitative study delves into measurable aspects of code generation, while the qualitative study explores user experiences and perceptions. This multifaceted investigation allows us to provide a nuanced assessment of ChatGPT's code generation capabilities and its potential place as a valuable tool for developers.

5.1 Study 1: Quasi-Experimental Study

5.1.1 Approach

This sub-study combines a quasi-experimental design with quantitative data analysis to determine whether there is a significant difference between GPT-generated and human-written code in terms of accuracy, quality, and readability. Quasi-experiments resemble experiments but involve non-random allocation of subject and control groups, which in this study is marked by the comparison of ChatGPT with human-written solutions. As both groups are pre-selected, random assignment is infeasible; however, the experimental approach is highly replicable and allows the dependent variables to be effectively measured by generating code with ChatGPT in a controlled setting and in a defined order, then comparing outcomes to human-written solutions (Wohlin et al. 2012).

In line with the methodology used in the study by Heyman et al. (2021), code evaluation was conducted using human judgment. Quantitative data was collected using an online questionnaire, which offered several advantages such as standardized evaluation, visual presentation of code, and easy sharing, potentially gaining a more diverse sample of participants.

5.1.2 Selection of Prompts

Measuring the effectiveness of code generation with ChatGPT through experimentation required prompts with human-written solutions for comparison. This was obtained from the HumanEval data set (OpenAI 2021*a*). This data set was chosen for its abundance of original problems with 164 prompts in total, each with its own canonical solution written in Python. It was also relatively well-established for use in testing AI-generated code, as it had been used in studies by Chen et al. (2021) and Yetistiren et al. (2022).

```python
def correct_bracketing(brackets: str):
    """ brackets is a string of "<" and ">".
    return True if every opening bracket has a corresponding closing bracket.

    >>> correct_bracketing("<")
    False
    >>> correct_bracketing("<>")
    True
    >>> correct_bracketing("<<><>>")
    True
    >>> correct_bracketing("><<>")
    False
    """

    depth = 0
    for b in brackets:
        if b == "<":
            depth += 1
        else:
            depth -= 1
        if depth < 0:
            return False
    return depth == 0
```

Figure 5.1: Example prompt from HumanEval with canonical solution

As seen in Figure 5.1, the prompts are in the form of empty functions with docstrings that explain the intended functionality. These prompts are few-shot, as the docstrings each contain multiple examples of correct outputs for various inputs. Each prompt also has its respective human-written solution, which has been added after the prompt in Figure 5.1 as an example.

The data set contains a total of 164 prompts; however, the reliance on human assessment placed a limitation on how many prompts could feasibly be included in the questionnaire. Therefore, it was necessary to select a subset of prompts, for which eight of the total 164 prompts were randomly selected. An initial pilot test with ten prompts and multiple programming languages yielded low response rates, with just one response. To reduce completion time, the study dropped the other programming languages and reduced the number of prompts from ten to eight, resulting in an approximate 15-minute completion time (see Appendix D).

5.1.3 Generating Code

The code generation was conducted on a single day with the "Feb 13" version of ChatGPT. The process involved giving ChatGPT a prompt at a time, with the prefix "Complete this Python function". Only one attempt was allowed for each prompt due to the previously stated limitations of how much code could be included in the questionnaire. The output code was then retrieved and stored for later use in the questionnaire.

```python
def correct_bracketing(brackets: str) -> bool:
    """
    Given a string of "<" and ">", checks if every opening bracket has a corresponding closing bracket.

    Args:
        brackets: A string of "<" and ">" characters.

    Returns:
        True if every opening bracket has a corresponding closing bracket, False otherwise.
    """
    stack = []
    for bracket in brackets:
        if bracket == "<":
            stack.append("<")
        elif bracket == ">":
            if len(stack) == 0:
                return False
            stack.pop()
    return len(stack) == 0
```

Figure 5.2: Example solution generated by ChatGPT

Figure 5.2 shows an example of ChatGPT's code output after it was given the prompt in Figure 5.1. Note that the docstring was rewritten by ChatGPT. In this example, the code length is roughly the same as the canonical solution shown in Figure **??**; however, this was not the case for all prompts, as certain problems that were solved in a single line in the canonical solution were solved in multiple lines by ChatGPT.

5.1.4 Questionnaire Design

The design goal of the questionnaire was for it to be easy to understand, quick to answer, and unbiased. This was accomplished by masking certain information, providing a simple rating system, and keeping the questions concise. The masking involved titling and describing the questionnaire's purpose in a manner that avoids any mention of AI, in order to avoid AI sentiments affecting the scoring (see Appendix D).

The questions were made more concise by displaying the prompt independently, followed by the code for each solution, therefore excluding the docstring from the ratings. Not including the docstrings in the presented solutions may have affected the results as it could affect the code's readability; however, it was deemed more important that the participants are presented with both solutions in an identical manner, as it may cause confusion if the solutions have different descriptions. Ultimately, it was deemed acceptable given that the primary focus of this study was to evaluate only the generated code.

All generated code for the eight selected prompts, along with their respective human-written solutions, was included in the questionnaire. These solutions were presented without attribution and in a random order as "Solution 1" and "Solution 2". The participants were asked to evaluate each solution on an ordinal scale of 1-5, with 1 being the lowest score and 5 the highest. This scale was used for all three variables and included the following options: (1) Poor, (2) Fair, (3) Good, (4) Very Good, and (5) Excellent. This scale was intended to facilitate easier scoring by minimizing complexity; however, in hindsight, this scale may have inadvertently introduced bias as the options include more positive than negative options.

5.1.5 Participants

The intended sample group for the questionnaire included experienced programmers familiar with Python syntax to ensure accurate and reliable ratings, as they were required to comprehend the code in order to fairly evaluate it for each of the three variables. The questionnaire included a brief introductory section informing the respondents of the criteria that they must have programming experience and be familiar with Python syntax. To confirm that the participants belonged to the intended sample group, they were asked to provide a brief description of their background in software development as well as rate their perceived experience level with programming and familiarity with Python syntax. All collected data were stored anonymously to safeguard participant confidentiality.

An alternative would have been to include a programming test, either as part of the questionnaire or to be completed before being given access. This would have more reliably ensured that the participants match their reported experience level; however, it may also have lowered the response rate and was therefore not included in this study. The questionnaire received 17 responses, of which 16 were accepted, with one response rejected due to a lack of experience. All responses were gathered from the following forums and subforums related to programming or survey participation:

- thecodingforums.com
- codeforum.org
- reddit.com
  - /r/python
  - /r/programming
  - /r/samplesize
  - /r/takemysurvey
  - /r/surveyexchange
  - /r/surveycircle

5.1.6 Data Analysis

Each response from the questionnaire yielded a total of 48 data points, as there were eight prompts, each with two solutions, rated on three different variables. With 16 accepted responses, this resulted in a total of 768 data points (see Appendix A). To present the data in a comprehensive and easy-to-understand summary, a collection of statistical measures, encompassing mean, median, mode, and interquartile range (IQR), was calculated for each variable.

The mean value provided a general idea of the central tendency of the data, while the median offers a better representation of the central location of the data, especially when dealing with outliers. The mode was identified as the most frequent value to highlight the most common response. IQR, a measure of statistical dispersion well-suited for non-parametric data, shows the difference between the first quartile (25%) and the third quartile (75%) of the data (Marino 2014).

To test the hypotheses, the Wilcoxon Signed-Rank Test (Wilcoxon 1945) was chosen for its non-parametric statistical analysis technique, tailored for paired or related samples, such as the same group of respondents rating both ChatGPT-generated Python code and the human-written solutions. Since the data consists of ordinal ratings, the Wilcoxon Signed-Rank Test is

an appropriate choice for non-normally distributed data, making it suitable for evaluating the efficiency of ChatGPT in generating Python code (Marino 2014, Social Science Statistics n.d.).

The null hypotheses were tested using an online Wilcoxon Signed-Rank Test Calculator (Social Science Statistics n.d.). This tool provided the resulting p-value, which was compared to the 5% significance level, as it is a standard used for testing significance (Office for National Statistics n.d.), in order to determine with 95% confidence whether there was a significant difference for each variable between ChatGPT-generated Python code and human-written solutions.

5.2 Study 2: Case Study

The second study focused on examining the usability of code generation with ChatGPT with qualitative data. The study was conducted as a case study where participants of different skill levels were given two programming assignments that they were instructed to solve using ChatGPT. Their opinions were then gathered through a semi-structured interview to gain insights into the subjective user experiences of writing prompts for code generation as well as analyzing the generated code to identify different strengths and weaknesses of ChatGPT in terms of code generation.

5.2.1 Participants

The four participants were people with different levels of experience with Python programming so this could be compared to their impressions in the conclusion. The age or sex of the participants was not taken into account as this was not relevant to the study. This is because the experience of, and education in programming is a more determining factor than age on its own, as many years of experience do indicate an older age, but an older age does not necessarily indicate any more experience in the subject. Due to the time necessary to conduct, transcribe, and analyze semi-structured interviews, as well as a low response rate when searching for participants, four participants were included in this part of the study.

5.2.2 Materials

The interviews were transcribed by hand in a Microsoft Word document during the interview. This step could have been made easier by having two people work on the interview. One to ask the questions and one to write down what is said. Enabling a larger sample size. This, however, could not be accomplished in our small team.

Two programming assignments appropriate for beginner programmers were created to create interactions between the participants and ChatGPT. The assignments were tailored towards beginners so that any participant no matter their skill level could understand them. This was done in order to get comparable results, which in hindsight did narrow down the results and in future research, one should consider a wider set of assignments in order to be able to compare how ChatGPT handles assignments of different difficulties. Since the qualitative section of the study aims to measure user interaction with ChatGPT, the assignments are not the same as the ones in the quantitative section.

The "Lucky Card" game which was inspired by an assignment from a first-year programming course at the University of Skövde, had instructions with very clear requirements while the "Hangman" game had intentionally less restrictive requirements (see Appendix B). This was done so that we could conclude whether this had an impact on the outcome of code generation. Both assignments are games since this would mean that there are potential resources available for ChatGPT to train on. "Lucky Card" however is not as well-established as "Hangman" and

could potentially have less data to train from. A potential change to make in future research is to have assignments of vastly different levels of available solutions online in order to reduce bias in favor of ChatGPT. What is important to remember is that the purpose of the assignments was to give the participants something to experiment on ChatGPT with. This means that the exact contents of the assignments were not as important as the experience solving them gave.

5.2.3 Procedure

The first step of the study was to develop the aforementioned assignments for the participants to solve. The assignments were inspired by entry-level programming courses and were written in a clear and intuitive way to facilitate easier prompt writing by the participants, regardless of their prior programming experience.

The second step was to prepare the semi-structured interviews by writing a framework of interview questions that could be expanded upon during the interviews. The questions were formulated in a clear and concise way, with care taken to avoid leading or biased questions. The potential follow-up questions, however, could not be examined as thoroughly as they would come up "on the spot"(Transcriptions can be found in Appendix E).

The mode of data collection was as stated semi-structured interviews. Before the interviews could be conducted requests for participation were sent to a dozen people including students at the University of Skövde and employees of a local IT development company. The requests consisted of a short explanation that we were conducting a study and subsequent interviews on a programming tool for our bachelor's thesis. No further description of the purpose of the study was given to avoid selection bias. The requests also contained information regarding consent to the study (see Appendix C).

The interviews themselves were held both in-person and over video calls. This was simply done to facilitate the schedules of participants. The in-person interviews were however preferred as in that case it is easier to communicate to the participant that they should take a pause in their answer to give time for the interviewer to write down their answers before moving on. The semi-structured nature of the interviews allowed the interviewer to ask questions that were not in the initial script but could shine a light on something that the participant said in an answer.

After the interviews, the last step was to evaluate the answers given by the participants in the interviews. The answers were analyzed through the "Grounded Theory" (Glaser & Strauss 1967) methodology specifically, the framework by Tie et al. (2019), which is well suited for when little is known about a phenomenon. The processes of the grounded theory framework are as follows: purposive sampling, collecting data, initial coding, intermediate coding/conceptualizing, and advanced coding/categorization. The purpose is to find the relevant pieces from the data and from this, identify the concepts and then into categories, in order to create a theory.

5.3 Alternative Approaches

Several alternative approaches focused on collecting existing data and perspectives were considered, including surveys to gather people's thoughts and opinions, and performing literature reviews to collect and synthesize findings from prior research. This would have provided data on the effectiveness of Python code generation with ChatGPT; however, since code generation technology is relatively recent and not yet widely adopted, there is not a sufficient amount of real-world experience to gather meaningful data through these methods. As a result, we concluded that they would be inadequate for our study at this stage of the technology's development.

The lack of existing real-life data necessitated the creation of data through experimentation. As such, questionnaires for the quasi-experimental study, combined with the second approach of

inviting interviewees to solve coding problems with ChatGPT for the case study, is more suitable in the current stage. Nevertheless, the alternative approaches mentioned could be highly valuable in future studies when more data on natural language programming becomes available, and the technology has achieved a greater level of maturity and adoption.

Evaluating AI-generated code with automated tests, such as in the studies by Yetistiren et al. (2022) and Chen et al. (2021), would have allowed for a larger number of prompts to be tested; however, we opted not to use automated tests to evaluate the generated code, as the intention was to gain a deeper understanding of how well ChatGPT can generate Python code, investigating multiple factors as well as gaining qualitative insights, rather than only determining whether the solution is successful or not.

6 | Results

6.1 Quasi-Experimental Study

6.1.1 Accuracy

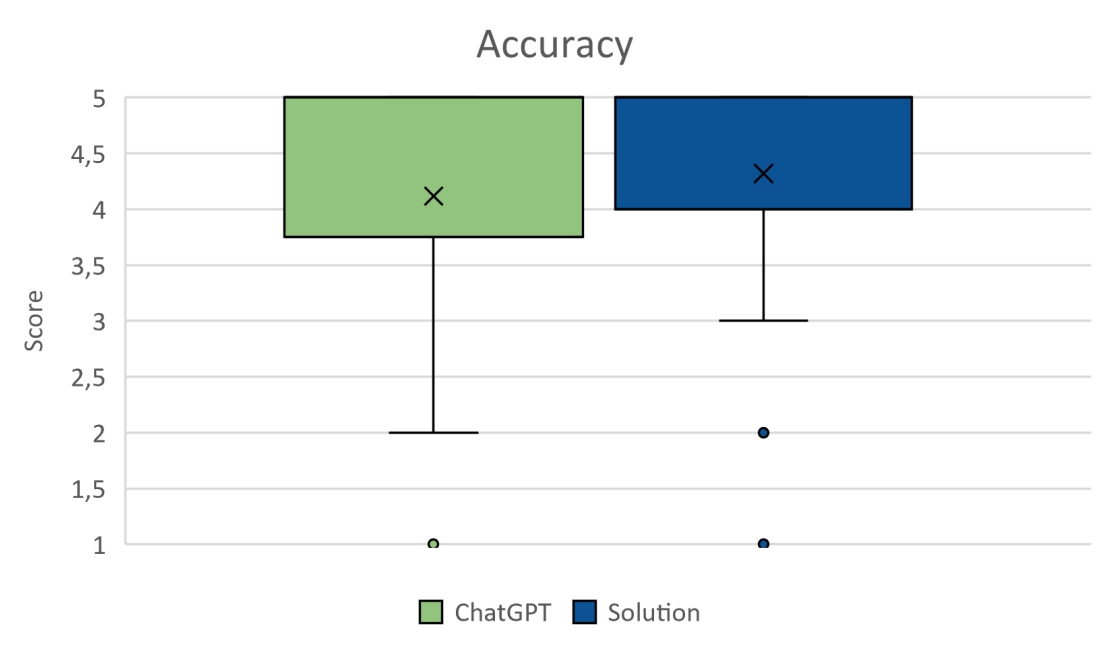


Figure 6.1: Boxplot of accuracy scores

	Mean	Median	Mode	IQR
ChatGPT	4.117	5	5	1.25
Human-written solutions	4.32	5	5	1

Table 6.1: Summary of accuracy scores

As shown in Figure 6.1 and Table 6.1, the mean accuracy score for ChatGPT-generated code is 4.117, while the mean score of human-written solutions is 4.32. The median and mode are the same for both methods, at a score of 5. The interquartile range (IQR) for ChatGPT-generated code is slightly higher at 1.25 compared to the human-written solutions' IQR of 1. The outcome of the hypothesis test yielded a p-value of 0.112. Therefore, we fail to reject the null hypothesis as the p-value is larger than the 0.05 significance level. This implies that there is no evidence of a significant difference in the accuracy of ChatGPT-generated and human-written Python code.

6.1.2 Quality

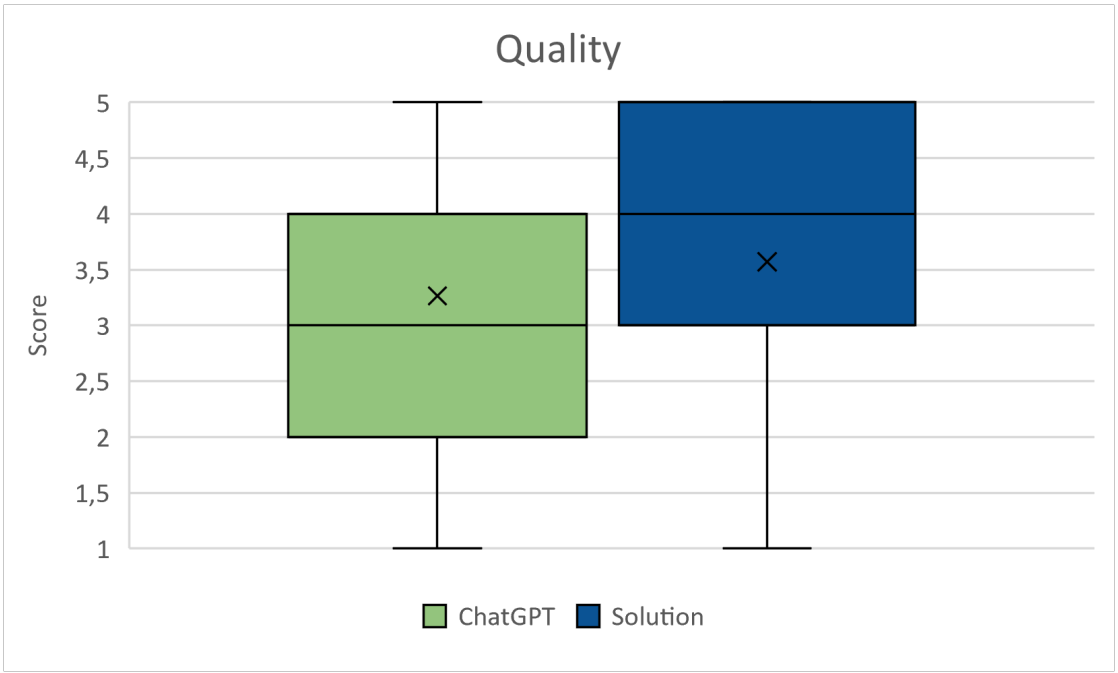


Figure 6.2: Boxplot of quality scores

	Mean	Median	Mode	IQR
ChatGPT	3.266	3	5	2
Human-written solutions	3.57	4	5	2

Table 6.2: Summary of quality scores

The mean quality score for ChatGPT-generated code, as seen in Figure 6.2 and Table 6.2, was 3.266, while the mean score for the human-written solutions was higher at 3.57. The median quality score for ChatGPT was 3, whereas the human-written solutions exhibited a higher median score of 4. Both the ChatGPT-generated code and the human-written solutions showed a mode quality score of 5. The IQR values of 2 for both ChatGPT and human-written solutions indicate similar variability in quality scores, suggesting a comparable range of performance levels within

both types of solutions. The hypothesis test calculated a p-value of 0.035, which is smaller than the chosen significance level of 0.05. Therefore, we reject the null hypothesis, indicating that there is evidence to suggest a significant difference between the median of the two paired groups.

6.1.3 Readability

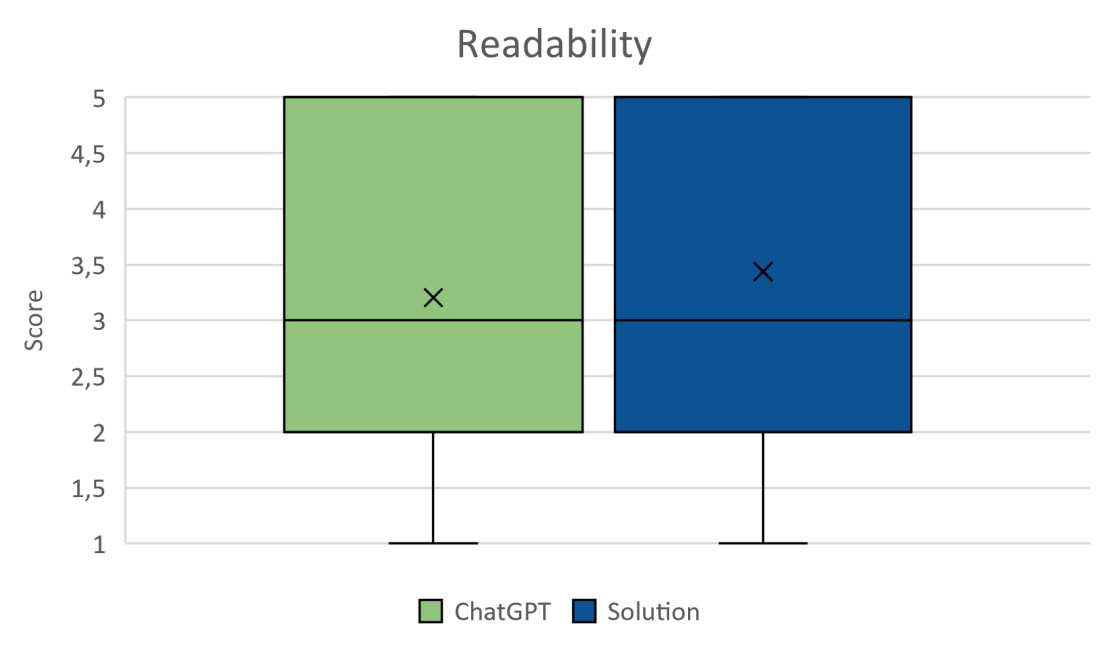


Figure 6.3: Boxplot of readability scores

	Mean	Median	Mode	IQR
ChatGPT	3.203	3	5	3
Human-written solutions	3.438	3	5	3

Table 6.3: Summary of readability scores

The mean readability score for ChatGPT-generated code, as seen in Figure 6.3 and Table 6.3 was 3.203, while the mean score for the human-written solutions was slightly higher at 3.438. The median readability score for both ChatGPT and the human-written solutions was identical at 3. The mode readability score for both the ChatGPT-generated code and the human-written solutions was 5. Lastly, the IQR for both sources of code was equal at 3. Based on the hypothesis test outcome with a p-value of 0.184, we fail to reject the null hypothesis. As the p-value is greater than the 0.05 significance level, it suggests that there is no significant difference between ChatGPT-generated and human-written Python code in terms of readability.

6.2 Case Study

The first step for formulating a grounded theory is to compare the cases of data collection for similarities and differences by identifying the words and sentences that provide the most information and holds the answer to the question. These codes were chosen so that concepts can be derived from the data. The codes can be found in Appendix F.

This report presents the results of a qualitative analysis of the usability of Python code generation using ChatGPT. The study involved interviews with four participants, each with varying levels of programming experience. The participant in the first interview ("one") had five years of work experience as a developer, the second one ("two") was a first-year student in a Computer Science program with no previous experience in programming. Participants of "three" and "four" were last-year students in two different three-year IT programs, "three" in Web Development, and "four" in Computer Science. The participants were recruited by contacting students of the University of Skövde, and Linköping University, as well as local IT-development companies. The report focuses on five categories that emerged during the end of the initial coding stage: experience with AI-generated code, personal connections to the code, potential in larger systems, human-AI cooperation, and future potential.

Although the answers to all questions asked during the interviews were analyzed during the coding process, the subsections in this part of the chapter come from the categories from this process. The full interviews can be found in Appendix E.

6.2.1 Experience With Generating the Code

Among the interviewees, opinions on the AI-generated code were both positive and critical, with some already using it for personal projects. Study participants noticed that ChatGPT performed well in creating code for the assignments, yet had trouble meeting specific guidelines such as those in the Lucky Card assignment. This issue became more prominent when participants attempted to modify the generated code by including certain criteria or implementing new features. They found that ChatGPT had a tendency to introduce unneeded changes to the existing code or add redundant code. In addition, providing ChatGPT with specific instructions and requirements in follow-up prompts caused it to generate code that rewrote the already functional part of the code and misusing or completely ignored existing functions and variables.

The readability was described as decreasing when new features were added, in the way a new developer writing new features for an unfamiliar system could. The relationship between quality and accuracy was explained by "two" like a scale: "The quality went down when I tried to make it more accurate"; however, "four" who had personal experience with AI-generated code found that their experience made it easier to write prompts that generated code that fulfilled all requirements in only two attempts.

6.2.2 Personal Connections to the Code

When the participants were asked whether they felt that they had written and were of the opinion that they were the true author of the code generated by ChatGPT, two of them answered no ("two" and "three"). "One", who was the only participant with work experience mentioned feeling more like a tester or manager than a developer; however, two of the participants ("one" and "four") noted that when a larger issue is divided into smaller, specific problems that ChatGPT can effectively tackle, they felt like they had indeed created the program as a whole.

6.2.3 Potential in Larger Systems

Participants unanimously expressed skepticism towards using code generation for larger systems, doubting that ChatGPT would be able to comprehend all essential components for proper implementation. "One" mentioned a concerning speculation about how ChatGPT probably could not understand the intricacies of a large system. This concern is intensified by the fact that ChatGPT cannot be provided with the entire code at once; however, "four" noted that this could potentially change with future development. "Two", the participant with the least amount of experience did not answer the question "Do you think that this technique could be used for larger systems such as a large game?", stating that they were: "not familiar with how larger systems work".

6.2.4 Human-AI Cooperation

Despite the limitations, interviewees emphasized the potential benefits of collaborating with ChatGPT. The generated code could serve as a foundation for further development, and large problems may be divided into manageable function-sized tasks like the size of the assignments used in the study. Additionally, "two" and "three" were of the opinion that ChatGPT can function as an educational tool, particularly for basic assignments, and "four" alleged that they already were using it for their projects. Nevertheless, it's vital to carefully integrate AI-generated code into educational settings and minimize dependency, as "one" considered its use in school assignments to be unethical and should be classified as cheating.

6.2.5 Future Potential

All interviewees acknowledged the need for continued development and refinement to address its current limitations and improve its ability to handle more complex tasks. The most experienced programmer, "One" said "it doesn't really solve any current issue and it would need more development. That is where it is interesting, it shows massive potential with AI." when asked what they thought about the future of generating code with ChatGPT. The rest, however, were hopeful that future improvements would make code generation a more viable option as developments continue in the field of AI language models.

7 | Discussion

7.1 Quasi-Experimental Study

7.1.1 Accuracy

The mean accuracy score for ChatGPT-generated code (4.117) is slightly lower than that of human-written solutions (4.32), however, The Wilcoxon Signed-Rank Test results support the hypothesis that there is no evidence of a significant difference in accuracy between ChatGPT-generated Python code and human-written solutions (p-value = $0.112 > \alpha = 0.05$). This finding underscores ChatGPT's impressive capability to produce accurate code in correspondence with the natural language prompts, often closely matching human performance. The results also show that both methods yield a median and mode of 5, suggesting that ChatGPT-generated code equals human-written solutions in many instances.

While the interquartile range (IQR) is marginally higher for ChatGPT-generated code (1.25) than for human-generated code (1), this difference is trivial and indicates only a minor disparity in the consistency of both methods' code accuracy. These results suggest that the ChatGPT model demonstrates considerable promise for generating highly accurate Python code in response to natural language prompts.

7.1.2 Quality

The hypothesis test for the quality variable resulted in a p-value of 0.035. Since the p-value is less than the significance level of 0.05, we reject the null hypothesis at a 95% confidence level. This indicates that there is a significant difference in code quality between ChatGPT-generated Python code and human-written solutions.

Despite the presence of a statistically significant difference in quality between the two samples, it is important to acknowledge that the ChatGPT-generated Python code had a mode value of 5, indicating that it frequently produced high-quality code. In conclusion, while the ChatGPT-generated Python code demonstrated potential in terms of frequently producing high-quality solutions, the overall quality of the code was found to be significantly different from, and lower than, human-written solutions. This suggests that ChatGPT, although promising, may still require further refinements and training to consistently produce code on par with human developers.

7.1.3 Readability

The null hypothesis states that there is no significant difference in readability between ChatGPT-generated Python code and human-written solutions. The p-value of 0.184 obtained from the statistical test indicates that there is no evidence of a significant difference in readability between

ChatGPT-generated code and human-written solutions at a 95% confidence level. As the p-value is greater than the commonly used alpha value of 0.05, we fail to reject the null hypothesis, implying that the differences in readability between the two groups can be attributed to random variation.

These findings suggest that ChatGPT-generated Python code is generally comparable to human-written solutions in terms of readability. It is worth noting that these results might not be generalizable for all programming languages and situations, and further research investigating the readability of generated code for a broader range of programming languages and problem types will potentially strengthen these findings. Moreover, as readability is a somewhat subjective measure, future investigations may benefit from incorporating multiple assessment methods to provide a more nuanced evaluation of the generated code's readability.

Notably, in certain instances, ChatGPT spontaneously restructured the docstring and function parameters to incorporate supplementary detail, such as clarifying argument and return data types, even though it was never explicitly directed to make these modifications; however, as the primary focus of the study was to evaluate the generated code rather than the function definition, these were ultimately not included in the questionnaire. While excluding docstrings might potentially have affected ChatGPT's readability score negatively compared to retaining them, including them risked introducing bias, since it is not clear whether ChatGPT made the alterations before or after it had generated the code. It is possible that the alterations were made to better suit the generated code, rather than the other way around. In addition, it was important to keep the questionnaire concise and to prevent potential confusion or bias arising from varying the docstrings.

7.2 Case Study

While ChatGPT-generated code demonstrates promise in handling simple tasks and offers potential benefits as an educational tool to the participants of this study, the opinion stated is that it currently struggles with more complex assignments and specific requirements. Participants expressed skepticism regarding the implementation of code generation in larger systems and stressed the importance of minimizing dependency, especially in educational settings. To optimize ChatGPT's effectiveness, the participants noted that continuous development and refinement is essential to address current limitations and enhance its ability to manage more advanced tasks. Future studies may examine the impact of iterative improvements in AI code assistance on developer productivity and its integration within software development ecosystems.

However, the AI's limitations in handling more complex assignments and adhering to specific guidelines cannot be overlooked. Its struggles with modifying generated code and the introduction of unnecessary or redundant elements raise concerns about its applicability in professional settings and larger-scale projects. Participants' skepticism towards incorporating AI-generated code in such contexts accentuates the need for ChatGPT's ongoing development and refinement. In order to work around the existing limitations of ChatGPT, developers can effectively tackle larger tasks by taking the approach of breaking them down into smaller, more manageable function-sized problems.

The results show no clear signs of a connection between the participants' programming experience and their opinions of ChatGPT for code generation; however, this may be due to the limited sample size and could be tested further in future studies with a larger sample group. Though the most experienced programmer of the participants ("one") was the most critical of ChatGPT's current usability for code generation. At the same time, the least experienced programmer ("two") also expressed doubt that it would be useful for larger systems, although they

were positive about the idea of using it for educational purposes.

In summary, while ChatGPT offers promising capabilities for code generation in particular scenarios, its current limitations highlight the importance of continued improvement to reliably address more intricate tasks and cater to a broader range of applications. The balance between AI-generated code and human expertise remains crucial for achieving the desired level of usability in code generation.

7.3 Comparative Analysis

The two sub-studies were conducted separately and achieved results independently of each other; however, the common goal and research area allow the findings to be compared to determine similarities regarding the effectiveness of generating Python code with ChatGPT. While the quasi-experimental study focused on quantitative data by comparing ratings of ChatGPT-generated code to human-written solutions, the case study investigated qualitative data, examining patterns derived from participant interviews who used ChatGPT for programming assignments.

The findings of both sub-studies indicate that ChatGPT is effective at solving simpler problems, though at a lower level than human-written solutions. The interviewees of the case study reported good impressions with simple problems. Although the quasi-experiment found a significant difference between ChatGPT-generated code and human-written solutions, the maximum value of the mode indicates that most participants considered the generated code to be well-written. Additionally, ChatGPT was found to perform well in terms of accuracy and readability, meaning that it was able to correctly identify the problem and generate well-structured and readable solutions that were relevant to the stated task. This indicates that while ChatGPT does not match the standard of human-written solutions, it is an effective tool for solving small programming problems.

In the case study, ChatGPT was found to struggle with more complex problems and iterative development, often generating redundant code, misusing variables, or making unnecessary changes to functional code; however, it achieved better results when the user divided the issue into smaller, function-sized problems. This would be similar to the prompts from the HumanEval dataset, used in the quasi-experiment, which are the size of a single function. Further research will be necessary to clearly define the limitations of ChatGPT in solving complex programming problems.

7.3.1 Comparison with Previous Work

Previous studies generally saw low solve ratios at one attempt. In a study on the Codex model by Finnie-Ansley et al. (2022), the model solved 10 out of 23 problems at the first attempt. Similar outcomes were observed in two HumanEval-based studies by Chen et al. (2021) and Yetistiren et al. (2022), presenting initial solve ratios of 28.8% and 28.7% respectively, without repeated sampling and only counting the correctly generated code.

In contrast with these earlier studies, the findings of the quasi-experimental study found no evidence of a significant difference in accuracy between the code generated by ChatGPT and the human-written solutions, though a significant difference was found regarding quality. This indicates that ChatGPT was able to successfully identify the task and generate relevant code for all prompts in a single attempt; however, the quality was lower than the human-written solutions. These findings indicate that the code generation capability has improved in the evolution from OpenAI's Codex model to ChatGPT.

7.4 Limitations

This study is constrained by several factors that affect the overall generalizability and applicability of its findings. Firstly, the scope was limited to simple programming challenges, enabling respondents to easily read and comprehend the provided code samples in order to evaluate them fairly. Consequently, the results might not extend to more complex programming tasks, which could necessitate further investigation.

Secondly, the study exclusively focused on Python as the programming language. This decision was made due to time constraints and limited resources available for conducting the research. While incorporating multiple programming languages would have allowed for a more comprehensive analysis, it would have also increased the number of generated code samples proportionally. This, in turn, would have complicated the first study, as all code samples needed to be incorporated into each questionnaire. The potential outcome would be a lower response rate due to respondents facing an increased workload. As such, the decision was made to prioritize a single language, leaving the possibility of extending the scope in future work. The results, therefore, may not be applicable to code generation using programming languages other than Python.

Additionally, the study focused only on ChatGPT (GPT-3.5) since it was the most popular and advanced model available at the time. This approach excluded less popular or emerging models that could potentially have comparable quality. It is important to note that during the course of this study, the AI landscape continued to evolve, and new models such as GPT-4 (OpenAI n.d.), Bing Chat (Microsoft n.d.), and Google Bard (Google n.d.) were becoming increasingly accessible. Nevertheless, these models remained either locked behind waitlists or accessible only through paywalls, rendering them unavailable for inclusion in the study.

A limitation of the Case Study is that the two games used as assignments were picked mostly at random. The use of these simple games have the stated reason that they should be understandable for all potential participants. But, this could skew results in favour of ChatGPT, providing a more positive impression that may not be earned.

Lastly, the non-deterministic nature of AI models introduces variance that inevitably influences the results. Depending on the variability in the randomness of the replies, the number of tasks used in the studies may not be sufficient to capture the reliability of code generation with ChatGPT. Future studies could examine this with a larger quantity of prompts and attempts, in order to determine the reliability of the generated code.

7.5 Threats to Validity

7.5.1 Internal Validity

The internal validity of the study can be analyzed by examining various factors that may have affected its outcomes. One such factor is history, which refers to events that may have occurred during the study, such as updates to the ChatGPT model, that could have influenced the results. In the quasi-experimental study, the experiment was conducted in a single day, using the "Feb 13" version of ChatGPT, minimizing the likelihood of model changes impacting the findings. The case study was conducted in March-April 2023 and the versions of ChatGPT used by the participants were "Feb 13" through "Mar 23".

In the case study, participant bias toward or against AI may skew the results if they do not provide an accurate representation of their experience with the AI-generated code. This threat was handled by asking the participants about their initial impressions and previous experience with AI-generated code since their answers to this question could be considered when drawing

conclusions from their other answers.

Another factor to consider is maturation, which in the quasi-experimental study pertains to the development of participants' knowledge or expertise in Python and ChatGPT, potentially affecting their performance in the programming assignments. The order of code samples presented in the questionnaire could have had an impact on the outcome. If the first code samples influenced the participants' opinion of the later samples, or if the respondents changed their earlier ratings after encountering the later code samples, this might introduce maturation-related biases in the study.

Another factor influencing maturation and internal validity is the respondents' attention span. If individuals lose focus or become fatigued while completing the questionnaire, it may lead to a less thorough analysis of the code samples, subsequently impacting the results. To mitigate this potential issue in the quasi-experimental study, the questionnaire length was designed to be as concise as necessary, while still capturing as much data as possible. The estimated time for completion was roughly 15 minutes, which skews towards the longer side. Although it may have been possible to further reduce the length by providing fewer code samples for each participant, either through a decrease in total samples or by distributing them across different individuals, such alterations could have complicated the data analysis process, rendering it less suitable for drawing meaningful conclusions.

Selection bias is another crucial factor in ensuring internal validity, as it concerns the representativeness of the sample and the sampling methods employed. In the quasi-experimental study, the target population included individuals with programming experience; however, given the absence of controls on who could respond to the questionnaire, it was necessary to filter answers based on the participants' self-reported programming backgrounds. Consequently, responses were removed if they fell outside the intended sample group. This approach may introduce selection bias towards people who are active on those types of forums, and furthermore, the absence of controlled programming tests for the participants means that it is not possible to ensure that the stated experience level matches their actual expertise.

Instrumentation is an additional aspect that warrants examination. It entails evaluating the design of questionnaires and interviews to ensure that they effectively capture the required data and that questions are clear and consistent for all participants. In the quasi-experimental study, it was crucial for participants to understand the different variables. To aid comprehension, the questionnaire offered descriptions of these variables and stressed their importance in each section; however, one limitation was the utilization of an ordinal scale to record ratings; the 1-5 scale simplified and expedited participant responses, prioritizing comprehensibility and speed over intricate analysis. Given the low response rate, this decision seems appropriate. Nevertheless, future studies might explore alternative techniques, such as follow-up interviews or free-written text responses, to capture more nuanced insights.

A threat to the internal validity in terms of the interviews held in the case study is the risk of misinterpretations when interpreting the answers, be it due to language barriers, the misuse of words, or personal research bias. The scope of this threat has been limited through the use of the framework for Grounded Theory methodology which aids in analyzing qualitative data. Additionally. Often when using surveys and interviews, different researchers conduct the interviews from those that analyze the data. The analysis is also often conducted by multiple researchers to avoid certain researcher biases through second opinions. This, however, would require a larger group of researchers and potentially also a larger time frame.

The assignments in the case study were based on well-established games such as Hangman and to an extent the Lucky Card game. This could potentially be a confounding factor as ChatGPT may have a pre-established idea of the necessary code, meaning that ChatGPT may add requirements that were not specified in the prompts. Inversely, the AI may ignore additional

requirements that are not part of its understanding of the game. The results may therefore be less applicable when creating new programs that the AI has no prior knowledge of. This is also seen in the interviews, where participants felt more involved when the criteria are more specific, as the prompts required more detail and attempts before the code fulfilled their expectations.

7.5.2 External Validity

Population validity concerns the generalizability of findings to broader populations, taking into account factors such as demographics, geography, and experience levels of participants. The low response rate in the quasi-experimental study poses challenges in terms of generalizing the results. The recruitment of participants through internet forums allows for a potentially diverse participant pool, covering various demographics; however, it may also be limited to the population group that is knowledgeable in programming and has an interest in participating in those forums. This may be a source of bias, as programmers who do not frequent these forums would not be included in the sample group. Since no demographic data was collected, it is impossible to ascertain the exact distribution among participants.

Selection bias is a validity threat for the case study, as the participants' willingness to participate could stem from their interest in AI. This may have resulted in a sample that is more interested in AI than the general population of interest, which may skew the results. Out of the eight people that were asked to participate, only half responded. Although the case study was limited to four participants, the qualitative approach allowed for a deeper analysis of their subjective opinions on AI-generated code. This reduces the need for a large sample to get generalizable results, though natural variation may still have had an impact. Future studies could conduct interviews with a larger sample group to get more opinions on the usability of AI-generated code.

Ecological validity assesses whether the programming assignments used in the study are representative of real-world problems and scenarios that ChatGPT could encounter. For the quasi-experimental study, the prompts were sourced from the HumanEval data set, which has been previously utilized to evaluate OpenAI Codex; however, due to time constraints, only a subset of HumanEval prompts could be employed in the study, possibly affecting the overall validity owing to the reduced task diversity. For instance, the randomly selected subset might consist predominantly of very easy or very difficult tasks relative to the full data set, potentially skewing the ratings in one direction.

7.5.3 Construct Validity

Construct validity evaluates the extent to which a study's methods and measures capture the underlying concepts they intend to measure. In this context, construct validity pertains to the conceptualization of the "effectiveness of generating Python code with ChatGPT". Construct validity is important when studying concepts that can not be measured directly such as the participants' opinions on prompt programming, as we did in the case study.

Rating code quality can be challenging due to the numerous variations in solutions for even the simplest problems. Previous studies have typically employed automated tests or human analysis to evaluate the effectiveness of generated code. In the quasi-experimental study, human assessment was chosen to rate the code's quality based on multiple factors, as this approach provided a more comprehensive understanding of the code itself, rather than simply examining the outcomes of running the code.

Keeping the tasks simple ensured that most programmers could determine the validity of the generated code by simply reading it; however, employing multiple participants introduced a natural variation in responses, as they had different levels of programming experience. Despite

this limitation, it was anticipated that a sufficiently large sample size would compensate for the variation in responses, leading to a more precise evaluation of ChatGPT's effectiveness in generating Python code.

7.5.4 Conclusion Validity

Conclusion validity evaluates the degree to which the findings and implications drawn from an analysis are accurate and reliable, particularly by assessing whether the assumptions of statistical tests have been violated. Violated assumptions occur when the chosen test is not appropriate for the type of data or design, potentially compromising the accuracy of the results. In the case of the quasi-experimental study, careful consideration of these concerns led to the selection of the Wilcoxon Signed-Rank Test for hypothesis testing, as it is a non-parametric test appropriate for the ordinal and paired nature of the gathered data. Paired data refers to the same respondents rating both ChatGPT and the solutions. By aligning the test with the data characteristics, the study aims to maintain transparency and enhance the credibility of the conclusions derived from the analysis.

Essential to our discussion is the potential implications of the response rates in both of the conducted studies. In the quasi-experimental study, data were collected from 16 participants through a questionnaire, while the case study utilized interviews with a total of four participants. Under normal circumstances, gathering information from a more sizable sample would contribute to the strength of the drawn conclusions; however, the employment of suitable statistical tests and research methodologies alleviates some of the constraints arising from a smaller sample size.

Still, it is necessary to recognize the limited generalizability of the findings, as the relatively small number of respondents in both studies poses a constraint. Future research endeavors could strive for larger sample sizes and incorporate diverse methods of data collection. Doing so would improve the validity of the conclusions and extend the applicability of the findings.

7.6 Ethical Considerations

With the involvement of human participants in both studies, it is important that these participants are aware of the studies' objectives and provide informed consent. The participants were informed that all questions were voluntary to answer. They were also informed of their right to end their participation at any point, with any answer given being redacted. The information regarding personal data provided to the participants of the interviews can be found in Appendix C.

The studies ensured adherence to ethical guidelines by prioritizing confidentiality, data protection, and anonymity for all parties involved. Confidentiality was maintained by ensuring that the participants remained anonymous, which was achieved by the absence of any personally identifying information while collecting, storing, and analyzing the data. By maintaining anonymity, the privacy of the participants was respected, reducing any potential risks or negative consequences of their involvement.

Furthermore, the questionnaire and interview methods employed in the studies presented minimal risks to the participants. For the questionnaires, individuals were asked to rank code samples, while the interview participants answered questions about their experience with ChatGPT for coding tasks. These task-oriented approaches centered around participants' expertise and experiences, without delving into personal or sensitive participant matters, thus maintaining a low-risk participation environment.

For the quasi-experimental study, there was a need to strike a balance between transparency and the need to minimize biases regarding participants' attitudes toward AI-generated code. Al-

though the true purpose of the questionnaire in the quasi-experimental study was not explicitly disclosed to the participants, they were informed that the study focused on evaluating and comparing Python code samples. By concealing the authorship of the code, we aimed to encourage a more objective evaluation without being influenced by the participants' attitudes toward AI-generated code. This approach deviates from full transparency; however, in this specific context, withholding certain details was necessary to maintain the integrity and validity of our study.

7.7 Potential Impact on Society

As AI continues to evolve, code generation by AI has the potential to drastically reshape various facets of society. Considering the social implications, the role of software developers might undergo a significant transformation. Instead of solely focusing on writing code, they may need to adapt their skills to emphasize a bigger-picture approach toward system design and managing AI-driven processes. Increased focus on continuous learning and human-AI collaboration may help maintain human developers' relevance in this evolving landscape. It may also be necessary to ensure that the AI code generation tools are ethically aligned by controlling for biased or harmful software.

7.7.1 Economical Impact

Economically, AI code generation can lead to a paradigm shift by reducing development costs and accelerating innovation. Increased efficiency would allow businesses to save on expenses and expedite production timelines. This economic boost could result in the emergence of new applications, faster market growth, and the entry of smaller companies and individuals formerly hindered by financial constraints.

7.7.2 Ecological Impact

From an ecological standpoint, the widespread use of AI-generated code presents concerns regarding energy consumption; the rise in technology proliferation may increase energy demand, consequently leading to higher carbon emissions and depletion of nonrenewable resources. To counter these negative environmental repercussions, adopting sustainable energy solutions will be of prime importance.

7.7.3 Educational Impact

AI-generated code presents both challenges and opportunities for education. The ease of generating code with AI assistance may encourage plagiarism and cheating, as students could rely on AI-generated solutions without fully understanding the underlying concepts. To mitigate these unethical practices, educators must develop strategies to identify and address instances of cheating and reinforce a strong emphasis on academic integrity; however, AI's potential to facilitate personalized coaching, clarify complex programming concepts, detect and resolve bugs, and assist in problem-solving could revolutionize the educational experience.

7.7.4 Social Impact

The widespread adoption of AI-generated code also holds the potential to influence the socio-economic landscape in terms of equity and accessibility. It could democratize programming by

lowering barriers to entry, granting individuals from diverse backgrounds and financial capabilities the opportunity to develop applications or bring innovative ideas to life. This accessibility could transform the tech industry by promoting a more inclusive environment and fortifying underrepresented perspectives.

However, the impact of these tools on accessibility could be constrained if they are prohibitively expensive or difficult to use, inadvertently contributing to a digital divide. To promote widespread access, it is crucial to ensure that AI tools, like ChatGPT, remain accessible and affordable to developers from diverse socio-economic backgrounds. By making AI code generation tools more accessible, the technological landscape can be made more inclusive and equitable.

7.7.5 Risks and Trustworthiness

The proliferation of AI-generated code may lead to an increased risk of malware and cybercrime. As AI technology advances, so does the sophistication of potential attacks, making it increasingly difficult for traditional security mechanisms to detect and prevent them. Furthermore, the trustworthiness of AI-generated code is a critical concern, as the reliance on an AI system could inadvertently introduce vulnerabilities or biases within the code that may lead to compromised security, ethical breaches, or other unintended consequences.

Mitigating these risks requires a multi-layered approach, encompassing the responsible development and deployment of AI systems, robust security measures, and constant vigilance towards potential threats. Ensuring AI-generated code is transparent and adhering to ethical guidelines will be vital in establishing trustworthiness. The potential advantages of AI-generated code must be balanced against these risks to maintain a safe and secure technological environment.

8 | Conclusion

The findings of the study indicate that ChatGPT can reliably solve small, function-sized, problems, albeit at a lower quality than human-written solutions. The code generation effectiveness appears to decrease with complexity, as interviewees reported issues with iterative development, with ChatGPT generating redundant code, misusing variables, or making unnecessary changes to functional code.

The findings of the quasi-experimental study demonstrated a significant difference between the effectiveness of ChatGPT-generated Python code and human-written code in terms of quality, using the Wilcoxon Signed-Rank Test. Most participants in the case study expressed a positive initial impression of ChatGPT-generated code, especially for simple problems; however, they found that ChatGPT often generated redundant code or misused existing functions and variables when attempting to add new features or when specifying criteria. The participants, therefore, expressed doubt that ChatGPT could be used to generate code for larger problems without first breaking it down into smaller function-sized parts. Further research could focus on analyzing these limitations and identifying ways to improve AI-generated code performance.

For accuracy, the findings of the quasi-experimental study found that there is no evidence of a significant difference between ChatGPT-generated Python code and human-written code. This suggests that, despite the observed significant difference in quality, the accuracy of the AI-generated code is not a key contributor to the identified discrepancy. The interviewees in the case study reported good accuracy for simple problems without clear specifications but found that the accuracy of the generated code suffered when attempting to add new functionality. ChatGPT would often rewrite previously written and functional code when attempting to incorporate new features. Further investigations could shed light on the specific factors that result in accuracy differences and present insights on how to enhance AI-generated code efficacy.

The readability scores in the quasi-experimental study suggest that there is no evidence of a significant difference in this attribute when comparing ChatGPT-generated Python code to human-written code. Though the study showed AI-generated code to be less effective in terms of quality, this analysis highlights that readability may not be a contributing factor. Further studies could delve into identifying the primary causes of this discrepancy and offering recommendations to improve future AI-generated code performance.

In the case study, the interviewees considered ChatGPT-generated code to be impressively readable when limited to programs with little complexity and already established solutions; however, the readability decreased when adding new features and requirements. Most participants stated that the Hangman game was more readable than the Lucky Card game. This was probably, according to one participant, because Hangman is a common game with well-established rules that almost everyone knows. This meant that ChatGPT could generate a mostly complete solution with the first prompt. Another participant thought this was due to the less specific instructions of the Hangman game.

8.1 Contributions

Our research builds upon existing knowledge and provides new insights into ChatGPT's effectiveness in generating Python code. By employing both quantitative and qualitative methods, we offer a comprehensive perspective on ChatGPT's performance and user experience. Consequently, our findings contribute to a deeper understanding of the current capabilities and areas of improvement for AI-generated code.

The results of our study can inform software developers, educators, and potential users about the advantages and limitations of utilizing ChatGPT for code generation, thereby influencing decision-making processes in its integration into workflows or educational programs. Beyond ChatGPT, our research contributes to a broader understanding and development of AI-assisted programming tools. The insights we provide might facilitate improvements in the design, implementation, and evaluation of similar technologies, paving the way for innovation and growth in this field. Tools like ChatGPT have the potential to bridge the gap between novice and experienced programmers, fostering a more inclusive environment within the programming community. By simplifying code generation and offering guidance, these tools may encourage more individuals to learn to program, diversify the field, and contribute to its advancement.

As a final remark, it is essential to acknowledge that AI-powered tools like ChatGPT are continuously evolving, and their dynamic nature warrants regular assessments and updates to ensure both optimal performance and responsible use. Therefore, we encourage the research community, practitioners, and policymakers to maintain an ongoing dialogue and collaboration regarding advancements in AI-generated code. By fostering open communication and regularly sharing findings, we can collectively work towards more robust, efficient, and ethical applications of AI-powered code generation technologies, ultimately driving progress and innovation in the field.

8.2 Future Work

In this report, we have analyzed the effectiveness of ChatGPT in generating Python code, considering aspects such as accuracy, quality, readability, and usability; however, there remains room for further research in this domain. In this section, we propose several areas of exploration that future researchers could investigate to better comprehend the potential of ChatGPT and other AI models for generating code across various circumstances.

8.2.1 Expanding Programming Languages

Our analysis was solely focused on Python, an immensely popular and widespread programming language; however, assessing the effectiveness of ChatGPT and other AI models in generating code for other programming languages is an essential avenue for future research. Investigating a broader set of programming languages such as JavaScript, Java, C++, or C# may yield insights into the strengths and weaknesses of AI-driven code generation across diverse programming environments.

8.2.2 Increased Prompt Instances

To improve the assessment of ChatGPT's code generation capabilities, future research could increase the number of prompts and chances provided to the AI, allowing it to generate appropriate solutions. By presenting the model with a wider variety of scenarios, researchers can better examine its effectiveness across diverse tasks. Additionally, this may reveal commonalities in the

AI's performance and help identify particular areas where further improvements could be made in code generation.

8.2.3 Complexity of Prompts

Our current analysis has primarily centered on relatively simple Python code generation tasks. Future work could examine ChatGPT's performance on more complex prompts, presenting intricately designed problems that require a deeper understanding of algorithms, data structures, and software architecture. Investigating the AI's ability to generate code for these complex tasks could reveal valuable insights into the model's limitations and the extent of its capabilities.

8.2.4 Chaining Prompts

Another potential area for future exploration is chaining prompts, where a series of tasks must be solved sequentially or interconnectedly. Chaining prompts may require an AI model to exhibit greater contextual understanding, problem-solving skills, and maintain state across multiple tasks. Investigating the capability of ChatGPT and other AI models to handle chained prompts could shed light on their performance across more sophisticated and interactive situations.

8.2.5 Examining Different Types of Prompts

Finally, future research could investigate the impact of prompt types on AI code generation, comparing the performance of AI models given zero-shot prompts, where no prior examples of the task are provided, and few-shot prompts, where the AI is given a small number of examples to learn from. This analysis could demonstrate how different levels of instruction and contextual information affect the accuracy, quality, and readability of the code produced by ChatGPT and similar AI models.

Another promising avenue for research pertains to the potential integration of visual prompts within the next generation of AI models, such as GPT-4 (OpenAI n.d.). This advancement could enable the generation of code based on visual descriptions, thus expanding AI capabilities considerably. For instance, a visual representation of a website layout could be converted into functional HTML, CSS, and JavaScript code. Furthermore, visual cues could also be employed to develop user-friendly interfaces or alter the graphical elements of an application. By examining these different prompt types, researchers can gain invaluable insights into the optimal techniques and strategies for AI code generation. This knowledge holds the potential to enhance the performance of AI models, ultimately making them more efficient, versatile, and intuitive to use.

Bibliography

Alkaissi, H. & McFarlane, S. I. (2023), 'Artificial hallucinations in chatgpt: implications in scientific writing', *Cureus* **15**(2).

Beaubouef, T. & Mason, J. (2005), 'Why the high attrition rate for computer science students: Some thoughts and observations', *SIGCSE Bull.* **37**(2), 103–106.
URL: *https://doi-org.libraryproxy.his.se/10.1145/1083431.1083474*

Birks, M. & Mills, J. (2015), *Grounded Theory A Practical Guide*, Sage Publications, Thousand Oaks, California, USA.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. (2020), Language models are few-shot learners, *in* H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan & H. Lin, eds, 'Advances in Neural Information Processing Systems', Vol. 33, Curran Associates, Inc., pp. 1877–1901.
URL: *https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf*

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I. & Zaremba, W. (2021), 'Evaluating large language models trained on code'.

Cockburn, A. & Williams, L. (2001), *The Costs and Benefits of Pair Programming*, Addison-Wesley Longman Publishing Co., Inc., USA, p. 223–243.

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, Association for Computing Machinery, New York, NY, USA, p. 10–19.
URL: *https://doi-org.libraryproxy.his.se/10.1145/3511861.3511863*

Frieder, S., Pinchetti, L., Griffiths, R.-R., Salvatori, T., Lukasiewicz, T., Petersen, P. C., Chevalier, A. & Berner, J. (2023), 'Mathematical capabilities of chatgpt', *arXiv preprint arXiv:2301.13867* .

GitHub (n.d.), 'Your ai pair programmer'. Accessed on: 24-April-2023.
URL: *https://github.com/features/copilot*

Glaser, B. G. & Strauss, A. L. (1967), *The Discovery of Grounded Theory: Strategies for Qualitative Research*, AldineTransaction (of Transaction Publishers), New Brunswick and London.

Google (n.d.), 'Meet bard'. Accessed on: 17-May-2023.
URL: *https://bard.google.com/*

Heyman, G., Huysegems, R., Justen, P. & Van Cutsem, T. (2021), Natural language-guided programming, *in* 'Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software', Onward! 2021, Association for Computing Machinery, New York, NY, USA, p. 39–55.
URL: *https://doi.org/10.1145/3486607.3486749*

Janiesch, C., Zschech, P. & Heinrich, K. (2021), 'Machine learning and deep learning', *Electronic Markets* **31**(3), 685–695.

Jiang, E., Olson, K., Toh, E., Molina, A., Donsbach, A., Terry, M. & Cai, C. J. (2022), Promptmaker: Prompt-based prototyping with large language models, *in* 'Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems', CHI EA '22, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3491101.3503564*

Manning, C. D. (2022), 'Human language understanding & reasoning', *Daedalus* **151**(2), 127–138.

Marino, M. J. (2014), 'The use and misuse of statistical methodologies in pharmacology research', *Biochemical pharmacology* **87**(1), 78–92.

Maynez, J., Narayan, S., Bohnet, B. & McDonald, R. (2020), 'On faithfulness and factuality in abstractive summarization'.

McManus, S. (2023), 'Friend or foe: Can computer coders trust chatgpt?'. Accessed on: 19-May-2023.
URL: *https://www.bbc.com/news/business-65086798*

Microsoft (n.d.), 'Bing chat'. Accessed on: 17-May-2023.
URL: *https://www.microsoft.com/en-us/edge/features/bing-chat?form=MT00D8*

Office for National Statistics (n.d.), 'Uncertainty and how we measure it for our surveys'. Accessed on: 18-May-2023.
URL: *https://www.ons.gov.uk/methodology/methodologytopicsandstatisticalconcepts/uncertaintyandhowwemeasure*

OpenAI (2021*a*), 'Humaneval: Hand-written evaluation set'. Accessed on: 24-April-2023.
URL: *https://github.com/openai/human-eval*

OpenAI (2021*b*), 'Openai codex'. Accessed on: 18-April-2023.
URL: *https://openai.com/blog/openai-codex*

OpenAI (2022), 'Introducing chatgpt'. Accessed on: 18-April-2023.
URL: *https://openai.com/blog/chatgpt*

OpenAI (n.d.), 'Gpt-4'. Accessed on: 17-May-2023.
URL: *https://openai.com/research/gpt-4*

Reynolds, L. & McDonell, K. (2021), Prompt programming for large language models: Beyond the few-shot paradigm, *in* 'Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems', CHI EA '21, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3411763.3451760*

Russell, S. J. (2010), *Artificial intelligence a modern approach*, Pearson Education, Inc.

Sarsa, S., Denny, P., Hellas, A. & Leinonen, J. (2022), Automatic generation of programming exercises and code explanations using large language models, *in* 'Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1', ICER '22, Association for Computing Machinery, New York, NY, USA, p. 27–43.
URL: *https://doi.org/10.1145/3501385.3543957*

Similarweb (2023), 'chat.openai.com'. Accessed on: 24-April-2023.
URL: *https://www.similarweb.com/website/chat.openai.com/#overview*

Social Science Statistics (n.d.), 'The wilcoxon signed-ranks test calculator'. Accessed on: 24-April-2023.
URL: *https://www.socscistatistics.com/tests/signedranks/default.aspx*

Tie, Y. C., Birks, M. & Francis, K. (2019), 'Grounded theory research: A design framework for novice researchers', *SAGE Open Medicine* **7**, 2050312118822927. PMID: 30637106.
URL: *https://doi.org/10.1177/2050312118822927*

Vaithilingam, P., Zhang, T. & Glassman, E. L. (2022), Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, *in* 'Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems', CHI EA '22, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3491101.3519665*

Wiggers, K. (2022), 'The emerging types of language models and why they matter'. Accessed on: 31-May-2023.
URL: *https://techcrunch.com/2022/04/28/the-emerging-types-of-language-models-and-why-they-matter/*

Wilcoxon, F. (1945), 'Individual comparisons by ranking methods', *Biometrics Bulletin* **1**(6), 80–83.
URL: *http://www.jstor.org/stable/3001968*

Williams, L. (2001), Integrating pair programming into a software development process, *in* 'Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)', pp. 27–36.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B. & Wesslén, A. (2012), *Experimentation in Software Engineering*, Springer, Germany.

Xu, F. F., Alon, U., Neubig, G. & Hellendoorn, V. J. (2022), A systematic evaluation of large language models of code, *in* 'Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming', MAPS 2022, Association for Computing Machinery, New York, NY, USA, p. 1–10.
URL: *https://doi.org/10.1145/3520312.3534862*

Xu, F. F., Vasilescu, B. & Neubig, G. (2022), 'In-ide code generation from natural language: Promise and challenges', *ACM Trans. Softw. Eng. Methodol.* **31**(2).
URL: *https://doi.org/10.1145/3487569*

Yetistiren, B., Ozsoy, I. & Tuzun, E. (2022), Assessing the quality of github copilot's code generation, *in* 'Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering', PROMISE 2022, Association for Computing Machinery, New York, NY, USA, p. 62–71.
URL: *https://doi.org/10.1145/3558489.3559072*

A | Appendix

	ChatGPT			Solution		
	Accuracy	Quality	Readability	Accuracy	Quality	Readability
	5	3	4	5	3	5
	5	5	5	5	5	5
	5	4	5	5	3	5
	5	3	5	5	3	3
	4	4	2	4	4	5
	5	5	1	5	5	5
	5	4	5	5	3	4
1	5	2	4	5	4	4
	5	4	3	3	4	4
	5	3	4	5	4	5
	3	3	3	1	3	3
	5	2	3	5	3	3
	3	2	2	4	3	3
	5	3	5	3	5	4
	5	4	5	5	3	2
	5	2	3	3	4	5

2

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
5	5	3	5	5	2
5	5	5	5	5	5
5	4	5	5	2	2
5	4	5	5	4	4
5	3	3	5	3	2
5	5	5	5	5	1
5	4	5	5	3	3
5	4	4	5	3	3
5	5	4	5	4	3
5	4	4	5	2	2
5	3	4	5	3	3
4	1	4	4	1	2
2	3	3	4	3	3
5	3	5	4	4	1
5	5	4	5	3	2
5	4	5	5	4	1

3

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
1	2	4	2	2	4
5	5	5	5	5	5
5	4	3	5	3	2
4	4	3	5	3	3
5	4	4	4	1	1
5	5	1	5	5	5
5	4	4	5	1	3
5	2	4	5	1	2
5	4	4	2	2	1
5	3	5	5	1	2
5	3	3	1	2	2
4	2	3	4	2	2
4	2	4	2	3	1
4	4	5	4	3	1
5	5	5	5	2	1
5	5	4	5	3	4

4

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
2	2	2	2	2	3
5	5	5	5	5	5
2	2	3	5	4	2
4	2	2	4	4	5
4	1	1	5	4	2
5	5	1	5	5	5
1	1	1	5	4	4
4	1	1	4	3	3
3	3	5	5	5	3
1	2	4	5	5	5
1	2	2	5	3	3
4	1	1	5	4	3
4	3	4	4	3	2
4	2	4	5	5	1
2	2	4	5	1	2
3	1	1	4	5	5

5

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
2	3	2	2	4	4
5	5	5	5	5	5
5	3	3	5	5	5
5	5	5	5	5	5
5	2	2	5	3	5
5	5	1	5	5	5
5	3	2	5	5	5
5	4	2	5	4	4
3	2	1	4	3	3
5	5	4	5	5	5
5	3	3	5	3	4
5	3	2	5	4	3
3	3	3	4	3	3
5	5	5	5	5	5
5	5	5	5	4	5
5	4	4	5	4	5

6

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
1	2	2	5	5	4
5	5	5	5	5	5
2	1	1	5	5	3
5	5	5	2	3	5
4	1	1	5	5	2
5	5	1	5	5	5
5	1	3	5	5	5
5	1	1	5	4	3
4	3	3	1	2	3
4	1	1	5	5	5
3	1	1	5	4	4
3	1	1	5	5	4
2	3	2	2	3	2
2	1	2	4	4	5
1	1	2	1	1	1
5	2	1	5	5	5

7

ChatGPT			**Solution**		
Accuracy	Quality	Readability	Accuracy	Quality	Readability
5	4	4	5	4	3
5	5	5	5	5	5
1	2	3	5	5	3
3	4	5	5	4	5
4	4	3	5	4	3
5	5	1	5	5	5
2	2	5	2	2	5
4	4	4	4	4	4
5	4	3	5	4	4
3	3	5	3	2	4
5	4	3	5	4	4
5	3	2	5	2	2
4	3	3	5	5	5
4	2	1	4	4	4
1	2	3	1	2	3
5	5	5	5	4	5

	ChatGPT			Solution		
	Accuracy	Quality	Readability	Accuracy	Quality	Readability
	2	2	1	1	1	1
	5	5	5	5	5	5
	5	5	3	5	4	3
	5	5	5	5	5	5
	3	1	1	3	1	1
	5	5	5	5	5	1
	5	3	2	5	4	4
8	5	4	2	5	2	3
	5	4	3	5	3	3
	5	4	3	5	3	4
	2	3	3	2	3	3
	4	4	1	5	4	3
	3	4	3	4	4	3
	5	5	4	1	1	4
	1	1	1	1	1	1
	5	5	3	5	4	4

B | Appendix

Intro:
These assignments should be written in Python, using ChatGPT. Remember to document your experience to make the interview questions easier to answer. When finished, hand in your solution to a20vicad@student.his.se and you will be contacted for the interview.

Assignment 1: Your assignment is to write a console application where the user plays "LuckyCard" game. This is a simple game, where a card deck is shuffled and placed in a heap. Each card has a value computed as follows: the card's value + bonus of the card, where the bonus is 4 for diamonds, 6 for clubs, 8 for hearts, and 10 for spades. In each round of the game, you draw three cards in sequence from the top of the deck. If the third card's value is between the first two card values, then you win! If the third card is equal to either of the first two cards or is outside of the set between the two card values, then you lose! Below is an illustration of the expected output.

```
Welcome to Lucky Card game!

——- Playing a game round
Card 1: Diamonds 7 ->Value = 11
Card 2: Hearts 11 ->Value = 19
Card 3: Hearts 12 ->Value = 20

You lose!

==========>Press ENTER to play again or "q" to quit:

——- Playing a game round
Card 1: Spades 12 ->Value = 22
Card 2: Clubs 5 ->Value = 11
Card 3: Diamonds 8 ->Value = 12

You win!

==========>Press ENTER to play again or "q" to quit: q

Thank you for playing and welcome back!
```

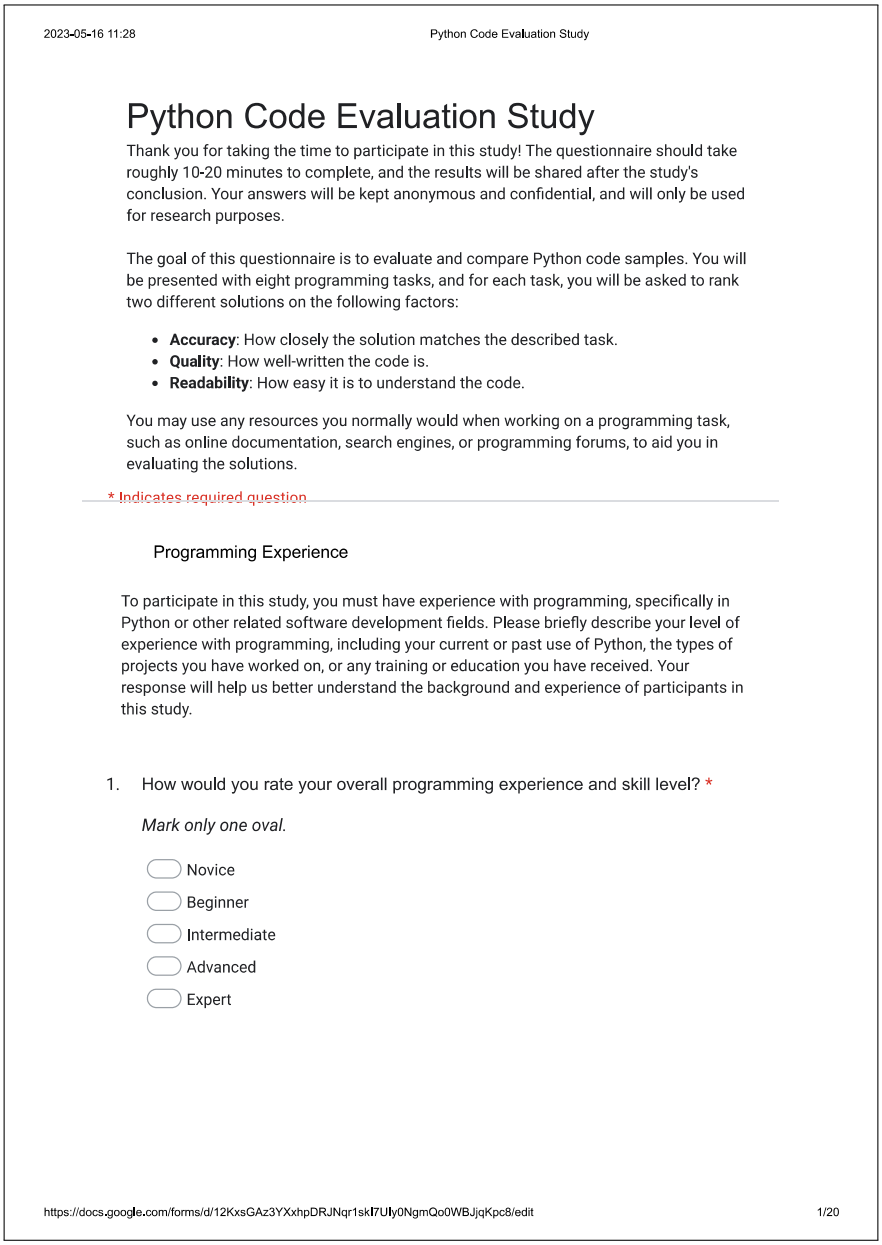
Assignment 2: Develop a game of Hangman in which a word is randomly chosen from storage by the 'computer'. Then the player must guess which letters the word contains. Each

wrong guess should remove a ‘life’ from the player until they either guess the word correctly or lose all their lives. Leading to a victory or defeat. The guessing progress and the incorrectly guessed letters should be displayed in some way.

C | Appendix

Be aware that none of your personal information will be published or saved and you have the right to leave the experiment at any point should you wish to do that. By participating in this study you accept these terms and conditions.

D | Appendix



2023-05-16 11:28 Python Code Evaluation Study

Python Code Evaluation Study

Thank you for taking the time to participate in this study! The questionnaire should take roughly 10-20 minutes to complete, and the results will be shared after the study's conclusion. Your answers will be kept anonymous and confidential, and will only be used for research purposes.

The goal of this questionnaire is to evaluate and compare Python code samples. You will be presented with eight programming tasks, and for each task, you will be asked to rank two different solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

You may use any resources you normally would when working on a programming task, such as online documentation, search engines, or programming forums, to aid you in evaluating the solutions.

* Indicates required question

Programming Experience

To participate in this study, you must have experience with programming, specifically in Python or other related software development fields. Please briefly describe your level of experience with programming, including your current or past use of Python, the types of projects you have worked on, or any training or education you have received. Your response will help us better understand the background and experience of participants in this study.

1. How would you rate your overall programming experience and skill level? *

Mark only one oval.

- ◯ Novice
- ◯ Beginner
- ◯ Intermediate
- ◯ Advanced
- ◯ Expert

https://docs.google.com/forms/d/12KxsGAz3YXxhpDRJNqr1skI7Uly0NgmQo0WBJjqKpc8/edit 1/20

2. How would you rate your familiarity with Python syntax and code structure? *

Mark only one oval.

Very Low

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Very High

3. What experience do you have with software development? *

1/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def correct_bracketing(brackets: str):
    """ brackets is a string of "<" and ">".
    return True if every opening bracket has a corresponding
closing bracket.

    >>> correct_bracketing("<")
    False
    >>> correct_bracketing("<>")
    True
    >>> correct_bracketing("<<><>>")
    True
    >>> correct_bracketing("><<>")
    False
    """
```

4. Solution 1 *

```
def correct_bracketing(brackets: str) -> bool:
    stack = []
    for bracket in brackets:
        if bracket == "<":
            stack.append("<")
        elif bracket == ">":
            if len(stack) == 0:
                return False
            stack.pop()
    return len(stack) == 0
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

5. Solution 2 *

```python
def correct_bracketing(brackets: str):
    depth = 0
    for b in brackets:
        if b == "<":
            depth += 1
        else:
            depth -= 1
        if depth < 0:
            return False
    return depth == 0
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

2/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def digitSum(s):
    """Task
    Write a function that takes a string as input and returns the
sum of the upper characters only'
    ASCII codes.

    Examples:
        digitSum("") => 0
        digitSum("abAB") => 131
        digitSum("abcCd") => 67
        digitSum("helloE") => 69
        digitSum("woArBld") => 131
        digitSum("aAaaaXa") => 153
    """
```

6. Solution 1 *

```
def digitSum(s):
    if s == "":
        return 0
    return sum(ord(char) if char.isupper() else 0 for char in s)
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

7. Solution 2 *

```
def digitSum(s: str) -> int:
    sum = 0
    for char in s:
        if char.isupper():
            sum += ord(char)
    return sum
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

3/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def strange_sort_list(lst):
    '''
    Given list of integers, return list in strange order.
    Strange sorting, is when you start with the minimum value,
    then maximum of the remaining integers, then minimum and so
on.

    Examples:
    strange_sort_list([1, 2, 3, 4]) == [1, 4, 2, 3]
    strange_sort_list([5, 5, 5, 5]) == [5, 5, 5, 5]
    strange_sort_list([]) == []
    '''
```

8. Solution 1 *

```python
def strange_sort_list(lst):
    sorted_lst = sorted(lst)
    result = []
    while sorted_lst:
        result.append(sorted_lst.pop(0))
        if sorted_lst:
            result.append(sorted_lst.pop(-1))
    return result
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

9. Solution 2 *

```python
def strange_sort_list(lst):
    res, switch = [], True
    while lst:
        res.append(min(lst) if switch else max(lst))
        lst.remove(res[-1])
        switch = not switch
    return res
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

4/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def max_fill(grid, capacity):
    import math
    """
    You are given a rectangular grid of wells. Each row represents
a single well, and each 1 in a row represents a single unit of
water.
    Each well has a corresponding bucket that can be used to
extract water from it, and all buckets have the same capacity.
    Your task is to use the buckets to empty the wells.
    Output the number of times you need to lower the buckets.

    Example 1:
        Input:
            grid : [[0,0,1,0], [0,1,0,0], [1,1,1,1]]
            bucket_capacity : 1
        Output: 6

    Example 2:
        Input:
            grid : [[0,0,1,1], [0,0,0,0], [1,1,1,1], [0,1,1,1]]
            bucket_capacity : 2
        Output: 5

    Example 3:
        Input:
            grid : [[0,0,0], [0,0,0]]
            bucket_capacity : 5
        Output: 0

    Constraints:
        * all wells have the same length
        * 1 <= grid.length <= 10^2
        * 1 <= grid[:,1].length <= 10^2
        * grid[i][j] -> 0 | 1
        * 1 <= capacity <= 10
    """
```

10. Solution 1 *

```python
def max_fill(grid, bucket_capacity):
    if not grid:
        return 0

    rows = len(grid)
    cols = len(grid[0])
    total_water = sum(sum(row) for row in grid)

    if total_water == 0:
        return 0

    buckets_needed = math.ceil(total_water / bucket_capacity)

    levels = []
    for j in range(cols):
        level = 0
        for i in range(rows):
            if grid[i][j] == 1:
                level += 1
        levels.append(level)

    buckets_per_level = [math.ceil(level / bucket_capacity)
                         for level in levels]

    total_buckets_needed = sum(buckets_per_level)

    return max(total_buckets_needed, buckets_needed)
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

11. Solution 2 *

```
def max_fill(grid, capacity):
    import math
    return sum([math.ceil(sum(arr)/capacity) for arr in grid])
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

5/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def eat(number, need, remaining):
    """
    You're a hungry rabbit, and you already have eaten a certain
number of carrots, but now you need to eat more carrots to
complete the day's meals.
    you should return an array of [ total number of eaten carrots
after your meals, the number of carrots left after your meals ]
    if there are not enough remaining carrots, you will eat all
remaining carrots, but will still be hungry.

    Example:
    * eat(5, 6, 10) -> [11, 4]
    * eat(4, 8, 9) -> [12, 1]
    * eat(1, 10, 10) -> [11, 0]
    * eat(2, 11, 5) -> [7, 0]

    Variables:
    @number : integer
        the number of carrots that you have eaten.
    @need : integer
        the number of carrots that you need to eat.
    @remaining : integer
        the number of remaining carrots thet exist in stock

    Constrain:
    * 0 <= number <= 1000
    * 0 <= need <= 1000
    * 0 <= remaining <= 1000

    Have fun :)
    """
```

12. Solution 1 *

```
def eat(number, need, remaining):
    if (need <= remaining):
        return [number + need, remaining-need]
    else:
        return [number + remaining, 0]
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

13. Solution 2 *

```
def eat(number, need, remaining):
    total_eaten = number + min(need, remaining)
    remaining = max(0, remaining - need)
    return [total_eaten, remaining]
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

6/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def is_equal_to_sum_even(n):
    """Evaluate whether the given number n can be written as the
sum of exactly 4 positive even numbers
    Example
    is_equal_to_sum_even(4) == False
    is_equal_to_sum_even(6) == False
    is_equal_to_sum_even(8) == True
    """
```

14. Solution 1 *

```
def is_equal_to_sum_even(n):
    return n % 2 == 0 and n >= 8
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

15. Solution 2 *

```python
def is_equal_to_sum_even(n):
    # Check if n is less than 8 or odd, since it can't be written
as the sum of 4 even numbers in that case
    if n < 8 or n % 2 == 1:
        return False

    # Check if n can be written as the sum of four even numbers
    for i in range(2, n//2+1, 2):
        for j in range(2, n//2+1, 2):
            for k in range(2, n//2+1, 2):
                m = n - i - j - k
                if m % 2 == 0 and m >= 2:
                    return True
    return False
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

7/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def multiply(a, b):
    """Complete the function that takes two integers and returns
    the product of their unit digits.
    Assume the input is always valid.
    Examples:
    multiply(148, 412) should return 16.
    multiply(19, 28) should return 72.
    multiply(2020, 1851) should return 0.
    multiply(14,-15) should return 20.
    """
```

16. Solution 1 *

```
def multiply(a, b):
    return abs(a % 10 * b % 10)
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

17. Solution 2 *

```
def multiply(a, b):
    return abs(a % 10) * abs(b % 10)
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

8/8

Based on the task description below, rate the given solutions on the following factors:

- **Accuracy**: How closely the solution matches the described task.
- **Quality**: How well-written the code is.
- **Readability**: How easy it is to understand the code.

Task Description

```
def fib4(n: int):
    """The Fib4 number sequence is a sequence similar to the
Fibbonacci sequnece that's defined as follows:
    fib4(0) -> 0
    fib4(1) -> 0
    fib4(2) -> 2
    fib4(3) -> 0
    fib4(n) -> fib4(n-1) + fib4(n-2) + fib4(n-3) + fib4(n-4).
    Please write a function to efficiently compute the n-th
element of the fib4 number sequence.  Do not use recursion.
    >>> fib4(5)
    4
    >>> fib4(6)
    8
    >>> fib4(7)
    14
    """
```

18. Solution 1 *

```
def fib4(n: int):
    results = [0, 0, 2, 0]
    if n < 4:
        return results[n]

    for _ in range(4, n + 1):
        results.append(results[-1] + results[-2] + results[-3] +
results[-4])
        results.pop(0)

    return results[-1]
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

19. Solution 2 *

```python
def fib4(n: int) -> int:
    if n < 2:
        return 0
    sequence = [0, 0, 2, 0]
    for i in range(4, n+1):
        # equivalent to sequence[(i-1) % 4] + sequence[(i-2) % 4]
+ sequence[(i-3) % 4] + sequence[(i-4) % 4]
        sequence[i % 4] = sum(sequence)
    return sequence[n % 4]
```

Mark only one oval per row.

	Poor	Fair	Good	Very Good	Excellent
Accuracy	◯	◯	◯	◯	◯
Quality	◯	◯	◯	◯	◯
Readability	◯	◯	◯	◯	◯

Thank you for participating!

If you wish to be informed of the results of the study, you can enter your e-mail here to receive an update after the study has concluded. Your e-mail will not be linked to your answers in this questionnaire.

If you have additional thoughts or feedback about the code or the questionnaire itself, you may share them below.

20. Thoughts

E | Appendix

VA: Victor Adamson (interviewer)

E.0.1 First interview:

VA: How much experience do you have with programming?
I have a bachelor's degree in information technology and worked for 5 years as a developer for two different firms.
VA: So you are familiar with the rise of AI-generated code?
Yes, but I have not used ChatGPT in that way before.
VA: Well, then that answers my next question which was if you have used it before.
Well like I said, not for generating code but for formulating certain points better.
VA: That seems to be a good use for it. What was your first impression of the concept of prompt programming?
It seemed interesting at first, with good potential. ChatGPT is overall a very impressive feat of AI engineering but I quickly found that it is still limited.
VA: In what ways?
The code generation for example. Sure it may look good for beginners but you really have to rewrite a lot of the generated code if you want it to fit into any existing code or just have it be efficient.
VA: Were the assignments in any way challenging?
Not at all, but I can see why they were at that level.
VA: And why is that?
Well, I guess that is a limitation of the current state of the AI.
VA: That is a fair assessment they were on the level of beginner programming courses. What did you think of the generated code regarding readability, quality, and accuracy?
The readability was better on the first iteration and only decreased when adding new features. ChatGpt seems to often add redundant or "intersecting" code and variables when adding new features to a program if you understand.
VA: Yes, I think I do. Could that not be an issue for a human developer as well?
Well, I guess. But it seems a lot less likely than with the AI. I mean, in cases of real developers adding new features that don't connect well with the original code it is usually in cases where the new code is added by a developer that didn't work on the original code. Or at least did it a long time ago.
VA: But the AI did it in just one "session"?
Yes, exactly.
VA: Do you think this form of programming could be used in the future?

Not really, it doesn't really solve any current issue and it would need more development. That is where it is interesting, it shows massive potential with AI. But right now from a user's standpoint, it can only really be used for cheating by students. In terms of code generation that is.
VA: Should it be classified as cheating?
Yes, I think so, you are not going to learn anything from being lazy.
VA: How many times did you have to generate the code before you were satisfied with the results?
With the more specific instructions on the first assignment, it took a few more iterations to cover all criteria. It was difficult sometimes to get the AI to add or change things without messing with the already functional part of the code.
VA: Messing in what ways?
mostly by adding unnecessary variables and not properly using functions that were already present. Like ignoring certain features altogether.
VA: I see, but the second assignment?
That one only took three iterations but could probably have been done in two. Probably because it was less specific.
VA: Well, do you feel like you have developed this program?
I mean, I gave the instructions to ChatGpt, and then made sure it functioned correctly. So I was more like a manager and tester instead of a developer.
VA: So?
So, no.
VA: Do you think that this technique could be used for larger systems such as a game or web service?
I guess it could be used to get a basic idea for certain features. But I don't think that these additions would play well with a large existing system since there is no way that ChatGpt could understand all necessary parts of the system to properly implement anything.
VA: What do you think of using it in a sort of "pair programming" way?
That could work as I said, it could give some tips for what sort of features to add and how they might look. But I still think that another human person would do the same job but better.
VA: All right, thank you very much for your time.

E.0.2 Second interview:

VA: How much experience do you have with programming?
I have taken two programming courses, but no Python. I can recognize code.
VA: What was your first impression of the concept of prompt programming?
I have used ChatGpt before and am familiar with how it works. Yeah, in what ways? For help with school assignments, like double-checking solutions.
VA: Were the assignments challenging?
Not really, but ChatGpt "thought" so. Because of the specific nature of the assignments. ChatGpt changed the wrong things. The second one worked better because it was less specific.
VA: What did you think of the generated code regarding readability, quality, and accuracy?
The generated code started off as readable but when adding more prompts and functions. Making it more complicated, the code got less readable. For the first assignments, unnecessary code was added when new parameters were given.
VA: And in terms of quality and accuracy? How well was the code written and did it accurately do what you asked it to do?

In the first assignment again, the quality went down when I tried to make it more accurate I guess.
VA: What could this form of programming be used for in the future?
Yes, for learning to program.
VA: In what way?
It's very good at writing every step. If you want to learn to program it would be useful. But not for specific assignments.
VA: How many times did you have to generate the assignment code before being satisfied with the results?
Many times, but only for the first assignment, it did not quite understand what to 'change'. Things a human would understand.
VA: Do you feel like you have developed this program?
No, I would not say that.
VA: Do you think that this technique could be used for larger systems such as a large game?
I'm not familiar with how larger systems work.
VA: That is a perfectly fine answer.
But even for simple but longer code like simple games. Not all code can be given to ChatGpt in one go and when doing it in more steps the code will be unreadable and just worse.
VA: Thank you very much for your time.

E.0.3 Third interview:

VA: How much experience do you have with programming?
Bachelors program in IT, web dev. Worked front end and full stack for 2 months, side projects in studies.
VA: What was your first impression of the concept of prompt programming?
Very impressed, but it has "gone down". I felt the hype, but after using it for projects it feels like it can confuse you.
VA: What do you mean by "confuse you"?
It's not very good at problem-solving. And if you want to use it for projects it often tries to solve the issue in the wrong way, and if you get into that mindset it will be hard to solve it yourself. I have had that happen. You can't see other paths and get stuck in that. And I got help from someone who hadn't gotten stuck.
VA: I haven't thought about that. Were the assignments challenging?
Not these, pretty straightforward. Easy for AI to generate code.
VA: What did you think of the generated code regarding readability, quality, and accuracy?
Impressively readable.
VA: Did you feel like you got the code you asked for?
Yes.
VA: How many times did you have to generate the assignment code before being satisfied with the results?
Not many times.
VA: What could this form of programming be used for in the future?
I think that it is good for specific functions to do a specific thing. Instead of larger code. The developer has to divide the project into smaller parts. If you give the whole project it will 100 percent of the time get the wrong answer.

VA: Do you feel like you have developed this program?
Not for these assignments.
VA: But something else?
Yes, because I have solved the problem first, and then I would use it for solving more complex parts. So syntax-wise, and that's what I mean with problem-solving. Good for answering small problems.
VA: So if it is in your opinion, better for smaller problems. Do you still think it could be used for larger systems such as a web service or a game?
If you are a developer then yes, but not for everyone AI can help to make it faster. But someone who doesn't know and just asks the AI for everything. It would only work for simple systems.
VA: So, other people have mentioned the use in schools. What do you think about using ChatGpt in school?
In school, it has been useful due to the basic assignments and that's where it is good. Like a diet plan. But not for complex tasks as I said. Maybe in the future?
VA: For larger projects in school?
Could be used there for solving specific issues. But you have to find where the problem lies and specify what function you need.
VA: Thank you for your time.

E.0.4 Fourth interview:

VA: How much experience do you have with programming?
Three years of educational experience (bachelour IT), but no work experience.
VA: What was your first impression of the concept of prompt programming?
"Holy *****, this actually works"
VA: Was this your first experience?
I often use it for programming.
VA: like in what ways?
Simpler scripts.
VA: Big projects?
No.
VA: How many times did you have to generate the assignment code before being satisfied? The first one took only two tries. It helped that I have experience in using it for this purpose. The second one took two as well.
VA: Were the assignments challenging?
Not at all.
VA: Do you think that some problems would be more difficult?
Yes, the more complex system ex. ChatGPT looses track of itself in larger programs. Like hangman could be done in only one attempt if I wanted to. Actually I could probably do the first one-in-one prompt as well, I only had to specify that you could play again and for hangman it only took two to fix a small issue in the end. 'Lucky card' is more complex though and I had to specify the rules.
VA: What did you think of the generated code regarding readability, quality, and accuracy?
I would say that if you are familiar with python it is readable. The quality is good as well, it works modularly.
VA: Accuracy refers to if the AI generated what you asked it to. Would you say that it did?
Yes it did, and even more.

VA: Like what?
In hangman, if you said "A" multiple times it did not punish you, same with invalid inputs. Probably because it recognises hangman, it might not do this for custom projects.
VA: Do you think this form of programming could be used in the future?
Yes, definitely.
VA: Any specific area?
In my experience for very simple programs, python has been very accurate, using libraries. It has a future, it's already good and will only improve.
VA: Do you feel like you have developed this program?
Not for hangman, but for lucky card I had to specify the rules more and the program simply made my work go faster.
VA: Do you think that this technique could be used for larger systems such as a game or web service?
Not ChatGPT in its current state.
VA: Thank you for participating.

F | Appendix

F.1 Codes for Grounded Theory:

F.1.1 Interview 1

Bachelor's degree in IT
5 years developer experience
Familiar with AI-generated code
Code generation has potential
Code rewriting necessary
Limited to simple problems
Readability issues when adding features
Redundant code
Less efficient than human developers
Current use limited to cheating by students
More development needed
Multiple attempts are needed to cover all criteria
AI tampering with existing code when adding features
Manager/tester role rather than developer
Usable for basic features
AI unlikely to understand larger systems
Usable for pair programming, but less so than a human

F.1.2 Interview 2

Limited programming experience
Familiar with ChatGPT
Used for school assignments
Difficulty understanding specific assignment criteria
Better performance with less-specific tasks
Readability decreases when adding features
Redundant code when adding new parameters
Unnecessary code additions
Quality impacted by 'accuracy instructions'
Potential as programming learning aid
Limitations in specific assignments
Multiple code generation attempts
Difficulty in understanding what to change
No personal connection to program development

Limited knowledge of larger systems
Unsuitable for larger, multi-step projects
Issues with readability and quality in complex tasks

F.1.3 Interview 3

Bachelor's degree in IT
Initially impressed, decreasing after usage
Difficulty with problem-solving
Attempts to solve the problem in the wrong way
Lack of perspective
Easy assignments
Impressively readable.
Code met expectations
Few code generation attempts required
Suitable for specific functions
Necessity to divide projects into smaller parts
Unsuitable for whole projects
Effective for small problem-solving
Applicable for large systems with developer guidance
AI-reliant users would only be able to solve simple problems
Potentially useful in educational settings for basic assignments
Future potential for complex tasks in education
Important to identify specific problems

F.1.4 Interview 4

Three years of CS
Initial good impression of code generation
Uses it often for simple scripts
Not useful for big projects
Experienced with code generation
Few attempts needed with experience
ChatGPT loses track in larger programs
Readable if familiar with Python
Good code quality
ChatGPT may add unspecified details for well-known concepts (Hangman)
High future potential as it improves
Does not feel like a developer, except when giving feedback
Not useful for larger projects in its current state