Unmasking the Giant: A Comprehensive Evaluation of ChatGPT's Proficiency in Coding Algorithms and Data Structures

Sayed Erfan Arefin, Tasnia Ashrafi Heya, Hasan Al-Qudah, Ynes Ineza and Abdul Serwadda
Texas Tech University, Lubbock, Texas, U.S.A.

Keywords: Large Language Models, ChatGPT, Code Smells, Algorithms.

Abstract: We conduct an extensive analysis of ChatGPT, a standout Large Language Model (LLM), particularly in coding within the Python language, focusing on data structures and algorithms. We assess ChatGPT's ability to accurately solve coding problems, its code quality, and the nature of run-time errors. Additionally, we examine how ChatGPT's code performs when it executes but doesn't solve the problem, identifying error patterns. We also explore whether ChatGPT has memorized training data through a structured experiment. Comparing with human performance where possible, our study encompasses both GPT-3.5 and GPT-4 models, various subtopics within the main areas, and problems of different complexities.

1 INTRODUCTION

Artificial Intelligence has made remarkable strides, showing extraordinary potential in automating a wide range of sectors. A significant contributor to this progress has been the advancement in Large Language Models (LLMs), which have essentially transformed the landscape of AI technology. Among these models, ChatGPT has emerged as a prominent example, particularly noted for its ability to engage in sustained, multi-turn conversations across a diverse range of problem domains.

With a specific focus on the Python programming language, this paper presents a rigorous evaluation of ChatGPT's coding capabilities. All challenges solved in our research are centered on data structures and algorithms, two topics at the very foundations of Computer Science. By virtue of the number and diversity of coding challenges posed to ChatGPT, variety of scenarios studied and attributes evaluated, *this paper is to our knowledge the most comprehensive evaluation of ChatGPT's coding proficiency* in the algorithms and data structures space to date.

We evaluate ChatGPT not only for its ability to generate correct solutions to the problems fed to it, but also for its code quality, and nature of run-time errors thrown by its code. Where ChatGPT code successfully executes, but fails to solve the problem at hand, we compile statistics, based on test cases evaluated, that provide some insights into how wrong ChatGPT code is in these kinds of situations. To gain some insights into whether ChatGPT might have directly memorized some of the data that was used to train it, we methodically design an experiment to investigate this phenomena. We investigate all these questions from the context of both its publicly available underlying models (GPT-3.5[1] and GPT-4), a vast array subtopics within the main topics, and varying degrees of difficulty of the problems.

The paper makes the following four primary contributions:

Evaluating Correctness of ChatGPT Coding Solutions Across a Diverse Spectrum of Algorithms and Data Structures Problems: Utilizing a comprehensive set of 2,792 coding prompts, we evaluate the accuracy of coding solutions produced by ChatGPT, spanning a diverse array of algorithms and data structures topics. Our analysis explores five key subtopics within algorithms: dynamic programming, greedy algorithms, depth-first search, divide and conquer, and topological sort. Concurrently, we investigate five fundamental areas within data structures, namely, priority queues, arrays, hash tables, stacks, and binary search trees. In addition, we conduct a detailed examination of string manipulation.

Evaluation of ChatGPT's Code Quality: Beyond the correctness of a coding solution, another key measure of coding proficiency is the code quality, a characterization of how well code is written in rela-

[1] In parts of the paper, we also refer to this by the base model number, GPT-3

Arefin, S., Heya, T., Al-Qudah, H., Ineza, Y. and Serwadda, A.
Unmasking the Giant: A Comprehensive Evaluation of ChatGPT's Proficiency in Coding Algorithms and Data Structures.
DOI: 10.5220/0012467100003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 1*, pages 412-419
ISBN: 978-989-758-680-4; ISSN: 2184-433X

tion to well-established good practices of programming. For this evaluation, we use PyLint, a widely-used tool in the Python programming language for checking a module for coding standards, and certain types of code smells.

Examining ChatGPT for Potential Memorization of Training Data: One of the fears surrounding LLMs is that they might memorize (potentially private) data that shows up in the training set. We design an experiment to provide some idea of how ChatGPT might be affected by this problem. While our experiment would not confirm 100% whether memorization has occurred, it can still provide some perspective to the end-user who is trying to make the decision on whether to allow their data to be used in the training set.

Assessing the Level of "wrongness" of Wrong ChatGPT Solutions: When ChatGPT generates a wrong solution to a problem, it is still insightful to gauge the extent to which this solution is wrong. For example, a marginally wrong solution might be fixable through minor tweaks. To study this phenomena, we evaluated the test cases passed by ChatGPT-generated programs which executed successfully yet failed to solve the problem at hand.

2 RELATED RESEARCH

Table 1 shows a summary of how our paper differs from a series of recent works that are closest to our work. We organize our variations from these works into five categories that we discuss in details below.

(1) Size of the experiment: Our research analyzes ChatGPT's coding abilities using a record 2,792 coding challenges, the largest number to date, compared to the previous high of 264 in related studies (see (Bubeck et al., 2023)). This scale is crucial for statistical robustness, allowing a thorough examination of the model's diverse strengths and weaknesses.

(2) Variety of coding tasks used in experiment: In terms of coding tasks, our study significantly extends the scope of previous research by Bubeck et al. and Tian et al., which focused on data structures, algorithms, and specific LeetCode problems. While Bubeck et al. used 264 questions including 100 from LeetCode without detailing their topics, and Tian et al. concentrated on arrays, hash tables, sorting, and string-related problems, our study covers more ground. We investigate five algorithmic areas: dynamic programming, greedy algorithms, depth-first search, divide and conquer, and topological sort. Additionally, we examine data structures, including priority queues, arrays, hash tables, stacks, and binary search trees, along with string-related problems.

(3) Coding quality evaluation: While most related studies mainly focus on the accuracy of coding solutions from language models, our approach additionally assess both GPT-3 and GPT-4 using a wide range of code quality metrics. This includes coding conventions, error intricacies, warnings, and refactoring needs. The research in (Feng et al., 2023) also examines coding errors using Flake8, but its primary aim is different, focusing on social media analysis to understand ChatGPT's code generation and overall usage.

(4) Language models under evaluation: Our study examines both GPT-4 and its predecessor, GPT-3.5, comparing their performance on identical coding challenges. Earlier works like (Noever and McKee, 2023), (Biswas, 2023), and (Tian et al., 2023) focused on GPT-3, contrasting it with older models like Codex and CodeGen, but did not include GPT-4, which was not available then. Our research is similar to (Bubeck et al., 2023), which also evaluates GPT-4's coding performance, but we differ in experiment size, the variety of computing problems addressed, and our inclusion of coding quality assessments.

(5) Training set memorization and assessment of wrong solutions: Finally, the evaluation of ChatGPT's memorization behavior and the assessment of the extent of "wrongness" of ChatGPT's wrong solutions (recall Section 1) are novelties in our work that have not been studied by any of the previous works on ChatGPT.

3 DATA COLLECTION EXPERIMENTS

Tools Used in Our Experiments: Our ChatGPT evaluations primarily utilized two tools: LeetCode and Pylint. LeetCode is an online platform that offers a vast array of coding challenges and interview preparation materials across various difficulty levels and topics, supporting numerous programming languages. Interview questions at major tech companies such as Google, Amazon, Microsoft, and so on are mostly directly drawn from LeetCode. LeetCode's built-in compiler not only assesses user-submitted code but also benchmarks it against other submissions using a comprehensive set of test cases. To evaluate the code quality of the programming solutions generated by ChatGPT, we used the Pylint Python library. PyLint conducts static analysis, checking Python code for adherence to coding standards, style guidelines, syntax, errors, unused code, and refactoring suggestions.

Data Collection Process: We manually input each LeetCode coding challenge into ChatGPT, al-

Table 1: Comparing our research with the sub-set of related works that are most similar to our work.

Research focus		Publications				
		(Noever and McKee, 2023)	(Biswas, 2023)	(Tian et al., 2023)	(Bubeck et al., 2023)	**This work**
Experiment details	**Number of coding prompts**	Not specified	Not specified	240	264	2,792
	Separate train and test problems			✓		✓
Coding problems solved	**Dynamic algorithms**			✓	Individual algorithms topics not specified	✓
	Greedy algorithms					✓
	Depth first search					✓
	Divide and conquer					✓
	Topological sort			✓		✓
	Priority queue					✓
	Arrays			✓		✓
	Hash tables			✓		✓
	Stacks					✓
	Binary search trees					✓
	Strings			✓		✓
Attributes of ChatGPT solutions evaluated	**Correctness of solutions**	✓		✓	✓	✓
	Runtime errors					✓
	Memorization of trainset			✓		✓
	Analysis of wrong solutions					✓
Code quality evaluation	**Error classification**					✓
	Warnings					✓
	Conventions					✓
	Refactoring					✓
Model evaluated	**GPT-3**	✓	✓	✓		✓
	GPT-4				✓	✓

lowing for visual inspection of its responses before further evaluation. This approach was chosen over using an API to identify incomplete responses. For each ChatGPT solution, we submitted the code to LeetCode and recorded: (1) a binary value indicating solution success, (2) the proportion of successful human submissions for that question, and (3) any error message generated. Additionally, each solution was analyzed by Pylint for code quality, returning problem type, ID, and a textual description of any issues.

Experiment Configurations: In our experiments, ChatGPT was given 2,792 coding prompts, resulting in 2,792 different Python programs. Approximately 52% (1,446) of these were *complete coding challenges*, where we provided ChatGPT with the full LeetCode question, including constraints, conditions, and examples. These 1,446 programs were equally divided between the two models: 723 were generated using GPT-3.5 and 723 with GPT-4. To ensure a fair comparison, the same challenges were presented to both GPT-3.5 and GPT-4. This implies that the total number of *unique* coding challenges posed to ChatGPT in this portion of our experiment were 723. The remaining 48% (1,346) of ChatGPT's tasks in our experiments were *incomplete coding challenges*, leading to an equal number of Python programs. These challenges were evenly split between GPT-3.5 and GPT-4, with each model processing 673 prompts. To maintain consistency with the "complete coding challenges," the same questions were presented to both GPT-3.5 and GPT-4. Therefore, the total number of unique coding questions in this part of the experiment was 673.

In these incomplete coding challenges, we presented ChatGPT with questions which had missing information. This approach is aimed to explore whether ChatGPT might be recalling parts of its training set. If ChatGPT correctly solves these problems despite missing information, it could indicate memorization of its training data. However, it might also suggest ChatGPT's advanced capability to intelligently infer the missing information and solve the problem. We investigated these possibilities by conducting this experiment on both train and test sets. For the data collection in this experiment, we entered each question into ChatGPT, omitting constraints and examples, which are crucial for clarifying the problem and the nature of the desired solution.

Selection of Coding Problems: The distribution of our coding problems across various topics and subtopics for both complete and incomplete challenges is shown in Table 2. Table 3 displays the breakdown of our questions by LeetCode's difficulty levels. Based on acceptance rates, Figure 1 provides some notion of a comparison of the difficulty levels of our problems with those in the entire LeetCode database, demonstrating that acceptance rates of our question selection (when attempted by humans) closely match the LeetCode database's overall acceptance rate distribution.

Why We Undertake Evaluations on Both the Training and Testing Sets: A significant proportion of the questions that will be posed by end-users to ChatGPT in real-life were possibly embedded in the

training set along with their answers. For instance, a student learning coding might ask ChatGPT about code snippets from forums like LeetCode that existed online before ChatGPT's training. These queries and their answers would probably have been included in its training data. Therefore, assessing ChatGPT's performance with questions and answers available on the internet during its training phase could more accurately reflect its capability in responding to such real-life queries.

On the other hand, assessments made using data (or queries) that showed up on the internet after its training date can better gauge its performance on novel queries. These require ChatGPT to synthesize various information fragments to formulate correct solutions. Both assessment approaches are vital for end-users and AI practitioners, as ChatGPT's real-world effectiveness, operating without live internet access, likely falls between these two scenarios. In our paper, we use 'train set' to refer to queries and solutions existing online before September 2021, the training cutoff for the ChatGPT model in our study, and 'test set' for those appearing after this date. We present and analyze results for each set separately.

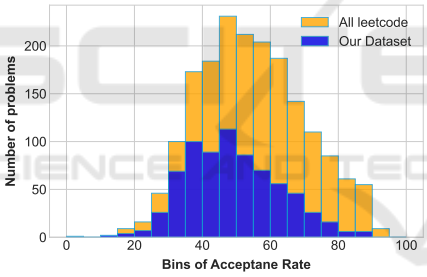


Figure 1: A histogram showing how the acceptance rates (tracked by LeetCode when humans solve problems) to LeetCode questions used in our experiments compare to the acceptance rates across all problems hosted on the entire LeetCode platform.

4 EXPERIMENTAL RESULTS

4.1 How Often Does ChatGPT Produce a Correct Coding Solution?

4.1.1 Complete Coding Challenges

For each of GPT-3, GPT-4 and human coders, Figure 2 displays the percentage of correct coding solutions by GPT-3, GPT-4, and human coders, aggregating results across all problems without topic or sub-topic differentiation. This is specifically for complete coding challenges, and the figure includes results for both train and test sets. For train set challenges, GPT-4 is notably superior, with GPT-3's performance comparable to that of humans. In the test set, humans surpass both models, while GPT-4's success rate is roughly double that of GPT-3. Comparing each model's performance between the train and test sets, the figure shows that GPT-3's performance on the test set is roughly 25% of its train set performance, while GPT-4 achieves about 50% of its train set effectiveness on the test set. This indicates GPT-4's superior generalization capability compared to GPT-3. The key insight from this data is that GPT-4 significantly outperforms GPT-3, and for coding solutions to algorithms and data structure problems not encountered during training, humans still perform much better than either ChatGPT model.

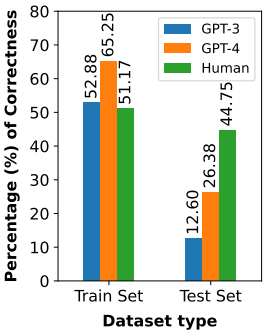
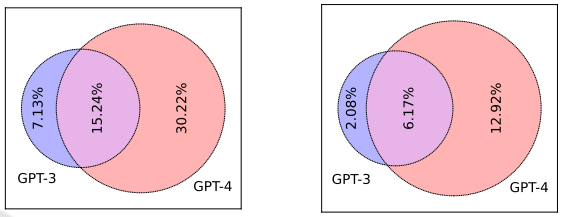


Figure 2: Correctness of GPT-3, GPT-4 and Humans for Train and Test sets.



(a) Train Set (b) Test Set

Figure 3: Venn diagram showing exclusive and inclusive correctness of GPT-3 and GPT-4 for all the problems in the train and test datasets.

Considering GPT-4's architectural advancements and its superior performance over GPT-3, it's relevant to ask if GPT-4 solved every problem GPT-3 did, plus additional ones GPT-3 couldn't. This query is explored in the Venn diagrams of Figure 3. For both train and test sets, these diagrams reveal that there are specific questions where GPT-3 succeeds but GPT-4 fails (7.13% in the train set and 2.08% in the test set). Human performance is not detailed in Figure 3 due to the lack of granular data; LeetCode provides only aggregate success rates, whereas our diagrams compare GPT-3 and GPT-4 as individual entities with distinct

Table 2: Percentage of LeetCode questions of different topics compared to the total no. and percentage of question no. of sub-topics compared to the topics they belong to in the dataset.

Topic	**No. of Questions (%)**		**Sub-topic**	**No. of Questions (%)**	
	Complete coding challenges	Incomplete coding challenges		Complete coding challenges	Incomplete coding challenges
Algorithm	422 (58.40%)	407 (60.48%)	Dynamic	132 (31.30%)	124 (30.47%)
			Greedy	136 (32.23%)	129 (31.70%)
			Depth first search	99 (23.46%)	99 (24.32%)
			Divide and conquer	33 (7.82%)	33 (8.11%)
			Topological sort	22 (5.21%)	22 (5.41 %)
Data Structure	248 (34.30%)	228 (33.88%)	Priority queue	82 (33.06%)	82 (35.96 %)
			Array	49 (19.76%)	45 (19.74 %)
			Hash table	43 (16.94%)	42 (18.42 %)
			Stack	38 (15.73%)	33 (14.47 %)
			Binary Search Tree	36 (14.52%)	26 (11.40 %)
Strings	53 (7.30%)	38 (5.65%)			
Total Questions	723	673			

Table 3: Dataset overview based on difficulty levels of Leetcode questions.

Difficulty Level	**Overall question share(%)**		**Topics**	**Number of questions**	
	Complete coding challenges	Incomplete coding challenges		Complete coding challenges	Incomplete coding challenges
Easy	96 (13.30%)	83 (12.31%)	Algorithm	40	38
			Data Structure	42	39
			Strings	14	6
Medium	374 (51.7%)	345 (51.18%)	Algorithm	230	217
			Data Structure	126	116
			Strings	18	12
Hard	253 (35.00%)	246 (36.49%)	Algorithm	152	152
			Data Structure	80	74
			Strings	21	20

pass or fail outcomes for each question.

Tables 4 and 5 provide a detailed breakdown of performance by topic and sub-topic. Table 4 confirms the trends observed in Figure 2 at the topic level, with GPT-4 leading on the train set and humans excelling on the test set across all topics. This pattern remains consistent at the sub-topic level, as shown in Table 5. Additionally, Table 5 indicates varying levels of difficulty across topics, regardless of the model. For instance, on the training set, accuracy ranges between 32% and 44% for dynamic programming, while depth-first search sees accuracies between 57% and 87%. This trend also appears in the test set and is attributed more to the question difficulty within a topic rather than the topic's nature, as discussed in the following paragraph.

Table 4: Correctness of GPT-3, GPT-4, and Humans for each problem topics in terms of Train and Test datasets.

	Topic	**Correct Solutions %**	
		Train Set	**Test Set**
GPT-3	Algorithms	51.90	10.62
	Data Structures	54.91	10.67
	Strings	50.00	36.84
GPT-4	Algorithms	66.79	23.12
	Data Structures	63.00	29.34
	Strings	64.70	42.10
Human	Algorithms	49.39	44.02
	Data Structures	54.76	44.80
	Strings	46.57	50.64

Table 6 illustrates performance variations across the difficulty levels of coding challenges. Notably, the percentage of correct solutions decreases from easy to hard questions, regardless of the topic. This indicates that question difficulty, rather than topic, is the primary factor influencing the correctness of solutions for both train and test sets. A notable observation is humans' relatively stable performance as question difficulty increases. For instance, on the test set, GPT-4's accuracy drops from 75% on easy questions to 0% on hard ones, while human accuracy declines more modestly, from 69.81% to 33.62%. On the training set, while the best-performing model outdoes humans on easy questions, the gap narrows significantly on harder questions.

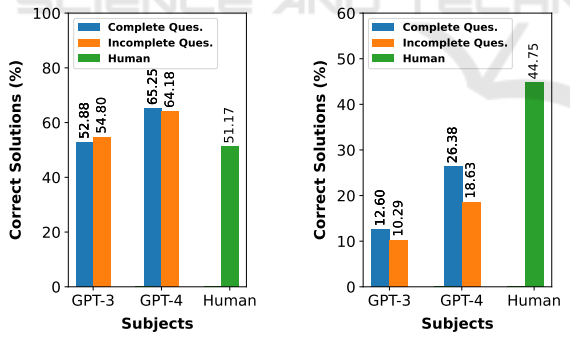
4.1.2 Incomplete Coding Challenges

Figure 4 displays results from the incomplete coding challenges alongside performance data from the complete challenges for comparison. On the training set (Figure 4a), both GPT-3 and GPT-4 show similar performance levels, regardless of whether the coding questions are complete or incomplete. Interestingly, humans, with access to complete information on all challenges, performed worse than both GPT models, which worked with incomplete information. This could suggest two possibilities: either GPT-3 and GPT-4 memorized these problems and solutions from

Table 5: Performance of GPT-3, GPT-4 and humans for each sub-topic across the Train and Test datasets.

Sub-topic	Subjects	Correct Solutions %	
		Train Set	Test Set
Dynamic Programming	GPT-3	32.43	6.90
	GPT-4	44.59	15.52
	Human	43.73	39.41
Greedy Algorithms	GPT-3	42.50	14.29
	GPT-4	66.25	23.21
	Human	46.55	42.66
Depth first search	GPT-3	77.94	16.13
	GPT-4	86.76	38.71
	Human	56.89	53.50
Divide and conquer	GPT-3	71.43	0
	GPT-4	78.57	40.0
	Human	54.30	43.24
Topological sort	GPT-3	41.67	0
	GPT-4	66.67	10.0
	Human	49.35	49.40
Priority queue	GPT-3	52.08	5.88
	GPT-4	70.83	35.29
	Human	51.46	46.89
Arrays	GPT-3	41.03	20.0
	GPT-4	43.59	20.0
	Human	54.11	38.37
Hash tables	GPT-3	59.38	9.09
	GPT-4	56.25	30.0
	Human	50.23	49.16
Stacks	GPT-3	57.14	20.0
	GPT-4	71.43	27.27
	Human	56.72	44.96
Binary Search Tree	GPT-3	73.08	10.0
	GPT-4	76.92	20.0
	Human	65.30	39.67

the training dataset, allowing consistent performance despite missing information, or these models have developed substantial domain knowledge from their extensive training data, enabling them to effectively fill in gaps in incomplete coding questions.



(a) Train Dataset (b) Test Dataset

Figure 4: Performance of GPT models on the incomplete coding challenges.

Figure 4b sheds light on the potential reasons behind the performance patterns observed in the incomplete coding challenges, particularly when sourced from the test set. Here, consistent with previous test set results, both GPT-3 and GPT-4 show a significant reduction in correct solutions compared to the training set. Notably, they still correctly solve 10% to 19% of incomplete coding questions. Since these are test set challenges not included in the training data, this performance cannot be due to memorization. It suggests that the correct solutions are likely a result of the models' ability to infer missing constraints, examples, and diagrams. Therefore, the overall behavior observed in Figures 4a and 4b could be attributed to a combination of some memorization and the robustness of the GPT models.

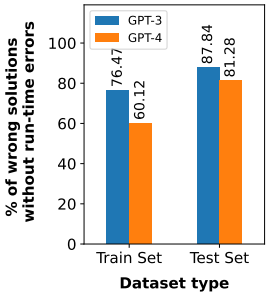


Figure 5: Percentage of non-runtime errors among total wrong answers. This figure shows the % of errors including mistakes and incorrect responses, excluding runtime errors.

4.2 Examining the Cases when ChatGPT Failed to Produce a Correct Solution

Figure 5 presents data on the frequency of cases where ChatGPT generated incorrect solutions that executed without run-time errors. The y-axis represents the percentage of such cases relative to all instances where ChatGPT failed to produce a correct solution (including both cases with and without run-time errors). On the training set, ChatGPT exhibited this behavior in 60% to 77% of cases, while on the test set, it occurred 80% of the time or more. Overall, this graph indicates that when ChatGPT couldn't provide a correct solution, the code often still executed successfully. Interestingly, GPT-4's wrong solutions were more likely to generate errors than GPT-3's wrong solutions, despite GPT-4's overall better performance in producing correct solutions. Further analysis of the errors will be discussed in Section 4.3.

Here, we delve into cases where ChatGPT's code executed successfully despite the solutions being incorrect. Table 7 addresses the question: *How wrong were ChatGPT's wrong solutions?* The importance of this question lies in determining whether the majority of incorrect solutions were only *marginally wrong*. If so, users in practice might need to make minor adjustments to these incorrect codes to solve the problems. To answer this, we rely on the test cases provided by LeetCode for each coding challenge. When a solution is flagged as wrong, LeetCode still provides the number of test cases passed. We use the fraction of test

Table 6: Performance of GPT-3, GPT-4 and humans across problems having varying difficulty levels.

Difficulty Level	Topics	Correct Solutions %					
		Train Set			Test Set		
		GPT-3	GPT-4	Humans	GPT-3	GPT-4	Humans
Easy	Algorithms	96.42	96.42	58.88	33.34	58.34	55.95
	Data Structures	72.72	81.81	68	33.34	66.67	55.64
	Strings	100	83.33	52.4	62.5	75	69.81
Medium	Algorithms	58.15	75.17	51.17	14.6	30.34	44.93
	Data Structures	64.04	64.04	54.5	8.1	35.13	45.7
	Strings	54.54	72.72	46.89	28.57	28.57	38.47
Hard	Algorithms	29.03	45.16	43.83	0	5.08	40.22
	Data Structures	27.45	49.01	46.64	6.45	10.34	40.31
	Strings	29.41	52.94	44.31	0	0	33.62

cases passed as a measure of how wrong a solution is—specifically, we classify a solution with a smaller proportion of test cases passed as *more wrong* than one with a larger proportion of test cases passed.

Table 7 presents the percentage of test cases passed by ChatGPT's incorrect solutions under various experiment conditions. The table is organized into bins of 10% width, ranging from 0-10% to 90-100% in the first column. For example, the value 42.57% in the second column (Algorithms - Train set - GPT-3) indicates that 42.57% of GPT-3's incorrect solutions passed between 0 to 10% of the test cases in the train set's algorithm problems. Overall, the table shows that when ChatGPT produced incorrect solutions, they often passed a very low percentage of test cases, indicating significant divergence from the correct solutions. This pattern remains consistent across different models(GPT-3 vs GPT-4), problem topic (algorithms vs data structures vs strings), and the source of the problems (i.e., train set vs test set).

4.3 Evaluation of ChatGPT's Code Quality

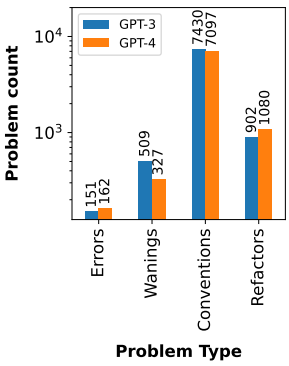


Figure 6: Code quality issues seen in ChatGPT code.

Figure 6 displays the total counts of issues observed in our Pylint-driven code quality experiments. We obtained a total of just over 300 errors, much lower in number than the other issues. On the other extreme, convention-related issues occurred over 15,000 times. In the following paragraphs, we break down each of these into their sub-types by percentage. The absolute numbers shown in Figure 6 should help provide context to the percentage numbers reported in the rest of the narrative.

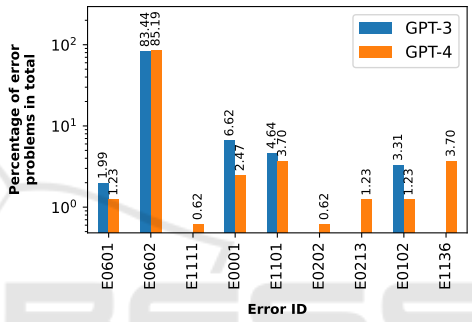


Figure 7: Percentage share of each of the error types seen in ChatGPT code.

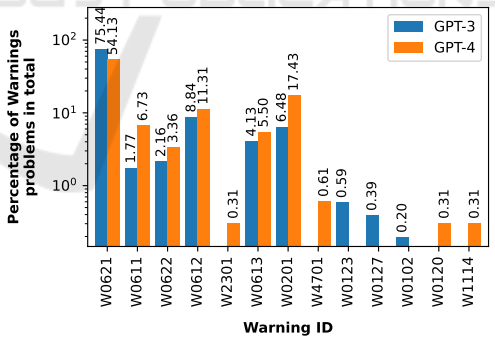


Figure 8: Percentage of the warning types in ChatGPT code.

Figure 7 illustrates the distribution of error types in our experiments. Notably, error E0602, denoted as "undefined-variable," occurred in over 80% of cases for both GPT-3 and GPT-4, significantly more frequently than other errors. The y-axis employs a logarithmic scale due to this disparity. This error indicates that accessing an *undefined-variable* caused the issue. For ChatGPT users, this finding suggests that when ChatGPT-generated code produces runtime errors, reviewing and correcting variable definitions may often

Table 7: Percentage of test cases passed when ChatGPT generated wrong solutions.

% of test cases passed	% of wrong solutions that passed the given % of test cases											
	Algorithms				Data structures				Strings			
	Train Set		Test Set		Train Set		Test Set		Train Set		Test Set	
	GPT-3	GPT-4	GPT-3	GPT-4	GPT-3	GPT-4	GPT-3	GPT-4	GPT-3	GPT-4	GPT-3	GPT-4
(0, 10]	42.57	45.45	54.40	49.51	35.19	37.84	43.33	37.78	50.00	66.67	44.44	25.00
(10, 20]	16.83	18.18	16.00	15.53	18.52	18.92	16.67	22.22	0.00	16.67	0.00	25.00
(20, 30]	8.91	7.27	7.20	6.80	1.85	16.22	5.00	2.22	0.00	0.00	0.00	0.00
(30, 40]	2.97	1.82	3.20	3.88	1.85	5.41	8.33	8.89	0.00	0.00	22.22	0.00
(40, 50]	5.94	3.64	1.60	6.80	3.70	8.11	5.00	8.89	8.33	0.00	11.11	0.00
(50, 60]	0.99	1.82	4.00	2.91	7.41	0.00	1.67	6.67	0.00	0.00	11.11	0.00
(60, 70]	3.96	3.64	4.80	2.91	3.70	5.41	3.33	2.22	0.00	0.00	0.00	50.00
(70, 80]	6.93	3.64	4.00	6.80	11.11	5.41	5.00	6.67	8.33	0.00	11.11	0.00
(80, 90]	3.96	1.82	3.20	1.94	9.26	0.00	5.00	2.22	16.67	0.00	0.00	0.00
(90, 100)	6.93	12.73	1.60	2.91	7.41	2.70	6.67	2.22	16.67	16.67	0.00	0.00

resolve many problems. Other error types occurred infrequently and are not discussed in detail.

Figure 8 displays the various warnings observed during our experiment, totaling 13 types. Similar to the error distribution, one warning type significantly outweighs the others by an entire order of magnitude. This prevalent warning is W0621, categorized by Pylint as *redefined-outer-name*. It triggers when a name from an outer scope is redefined.

Finally, we discovered 20 different refactor and 18 convention messages in our Pylint reports. Table 8 displays these messages and their occurrence percentages for both GPT-3 and GPT-4. Notably, two messages occurred significantly more frequently than the others. The first message, with ID *R0903*, pertains to *too-few-public-methods* according to Pylint documentation, highlighting cases where a class has an insufficient number of public methods. The dominant convention message bears ID *C0103* and is related to naming conventions, flagging instances where names do not align with their specific type's naming conventions.

Table 8: Percentage share of each of the refactor and convention issues seen in ChatGPT code.

Percentage of lint-problems occurring in terms of all lint-problems					
Refactor			Convention		
Message ID	GPT-3(%)	GPT-4(%)	Message ID	GPT-3(%)	GPT-4(%)
R0903	87.47	88.89	C0103	46.97	47.53
R1705	4.1	4.63	C0116	14.41	15.63
R0914	2.11	1.02	C0115	12.36	14.79
R1714	0.67	0.74	C0114	9.6	10.13
R0913	1.44	0.65	C0303	14.24	7.69
R1710	0.22	0.65	C0301	1.41	2.49
R1702	0.11	0.65	C0200	0.74	0.63
R1721	0.55	0.56	C0321	0	0.45
R1728	0.33	0.56	C0305	0.15	0.39
R0912	0.55	0.37	C0121	0	0.08
R1735	0.11	0.37	C0415	0	0.06
R1704	0.11	0.28	C0325	0	0.06
R0911	0	0.19	C0206	0.07	0.03
R1716	0.89	0.09	C1803	0	0.01
R0916	0.33	0.09	C0413	0	0.01
R1724	0.22	0.09	C0304	0.03	0
R1723	0.11	0.09	C0209	0.01	0
R1719	0.11	0.09	C0201	0.01	0
R1731	0.44	0			
R0205	0.11	0			

5 CONCLUSION

In our comprehensive study, we evaluated ChatGPT's proficiency in programming, focusing on algorithms and data structures with 2,792 coding prompts. Our findings show that ChatGPT's performance varies across topics and models. Humans outperformed ChatGPT in unseen problems but not in problems included in the training set. Interestingly, the older GPT-3 model sometimes outperformed the newer GPT-4, highlighting the complexity of learning algorithm improvements. Overall, our study highlights both the impressive capabilities and the areas for improvement in the latest LLMs, particularly ChatGPT.

REFERENCES

Biswas, S. (2023). Role of chatgpt in computer programming.: Chatgpt in computer programming. *Mesopotamian Journal of Computer Science*, 2023:8–16.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *Microsoft Research*.

Feng, Y., Vanam, S., Cherukupally, M., Zheng, W., Qiu, M., and Chen, H. (2023). Investigating code generation performance of chatgpt with crowdsourcing social data. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 876–885.

Noever, D. and McKee, F. (2023). Numeracy from literacy: Data science as an emergent skill from large language models. *arxiv*.

Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S.-C., Klein, J., and Bissyandé, T. F. (2023). Is chatgpt the ultimate programming assistant – how far is it? *arxiv*.