Journal of Information Technology Management
ISSN-OnLine: 2980-7972
University of Tehran

DeepSeek vs. ChatGPT: Which Performs Better in Python Coding?

Rania A. M. Abdalla*

*Corresponding author, Department of Information Technology, Palestine Technical University, Kadoorie, Palestine. E-mail: r.alkhateeb@ptuk.edu.ps

Journal of Information Technology Management, 2025, Vol. 18, Issue 2, pp. 1-27
Published by the University of Tehran, College of Management
doi: https://doi.org/10.22059/jitm.2026.107165
Article Type: Research Paper
© Authors
Received: December 07, 2025
Received in revised form: January 17, 2026
Accepted: February 21, 2026
Published online: March 01, 2026

Abstract

This paper conducts a comparative evaluation of two advanced large language models (LLMs) — ChatGPT-4 and DeepSeek v3—utilizing 80 algorithmic problems from Code forces categorized into four difficulty levels: Easy (800–1100), Intermediate (1200–1600), Advanced (1700–2000), and Expert (2100–2400), focusing on code generation in Python. Standardized prompts and controlled testing conditions enable the assessment of models on accuracy, effi-ciency, and code readability. As the complexity of issues increases, DeepSeek frequently out-performs ChatGPT in both accuracy and efficiency, despite both models excelling in simpler tasks. This, however, results in reduced code clarity and increased memory use. While less pre-cise at elevated levels, ChatGPT produces more concise and idiomatic responses. Both models had limited competence at the expert level; however, DeepSeek-R1 indicated a slight edge. The study illustrates a trade-off between accuracy and code clarity, so as to inform the selection of LLMs based on task requirements and provide a foundation for future efforts in optimizing code generation models for actual applications.

Keywords: ChatGPT, DeepSeek, Coding, Algorithms, Python

Introduction

Several code-related automated jobs have been drastically altered by the advent of Large Language Models (LLMs), including code translation (Chen et al., 2018), code repair (Fan et al., 2023), and code generation (Jiang et al., 2018). Code generation involves automatically creating code from English-language descriptions, representing one of the most effective applications of LLMs that adheres to the widely accepted idea of natural language to code generation (Jiang et al., 2018). Hou et al. (2024) suggest that LLMs are changing software engineering by taking on duties including code creation, summarization, completion, and program repair

in a wide range of fields. Cruz-Benito et al. (2021) show how well deep learning-based language models can automatically write and finish programs using Python datasets.

Among the most famous LLMs in the field is OpenAI's ChatGPT 4.0 and DeepSeek v3.0, which is a relatively modern but highly regarded model designed for code reasoning and generation tasks. Both models utilize massive training corpora and billions of parameters, but they differ in design, optimization methods, and maybe the kinds of coding scenarios they handle best. The efficiency and accuracy of code generation are substantially enhanced by the integration of DeepSeek-3 and ChatGPT-4. It facilitates enhanced collaboration among developers in contem-porary programming environments by utilizing exceptional language models, which in turn ra-tionalize workflows and minimize defects (Yao et al., 2025). These advancements are essential for the software development industry's evolution.

Before integrating LLMs into real-world programming tools, researchers and developers need rigorous evidence of their performance across tasks of varying complexity. Despite the growing use of GPT-4 and DeepSeek-V3 for code generation, there has been surprisingly little academic research comparing them.

Literature Review

Recent research has examined AI-driven code generation using models such as ChatGPT and DeepSeek. Manik (2025) compared the merits and cons of the two programming models. ChatGPT makes code snippets well, but DeepSeek solves more complex problems better, according to researchers. Shakya et al. (2025) stressed the need for clearer effectiveness criteria for these models. Similarly, Wang et al. (2023) tested ChatGPT's performance using the USMLE. Their use of standardized, expert-validated test items and category scoring sets a high standard for LLM evaluation in certain disciplines. This study emphasizes the importance of regulated assessment in competitive programming. Anand et al. (2024) also demonstrated how automated programs and code development have evolved, demonstrating how AI systems could improve output. Joshi (2025) studied DeepSeek's performance and architecture to improve his complicated programming skills. In contrast, Mulder et al. (2023) showed how code generation models could improve students' computational thinking skills. Moreover, Cambaz & Zhang (2024) also reviewed the literature on artificial intelligence models in programming education, opening new research opportunities. Hou et al. (2024) conducted a systematic review of 395 papers to reveal LLM usage patterns across six SE fields and highlight the importance of decoder-only models like GPT-4 and ChatGPT in code-related tasks. Tang et al. (2024) published Bio Coder, a benchmark for subject-specific bioinformatics code. They demonstrate the benefits of domain-specific datasets and rigorous testing methodologies. The benchmark only works for biology, not algorithmic problem solving like Codeorces. This highlights the lack of wide, task-diverse benchmarks for languages like Python in general situations. AWD-LSTM, QRNN, and Transformer neural

network architectures were explored by Cruz-Benito et al. (2021) to create Python code. This proved that methods are effective in education and growth.

These ideals notwithstanding, there are still gaps. For example, Buscemi (2023) showed that programming languages perform differently, although Jiang et al. (2018) stressed the need for task-oriented evaluations using valid datasets. Gao et al. (2025) addressed software engineering challenges, suggesting prioritizing model reliability and adaptability in future studies. Overall, understanding these topics may help us comprehend how artificial intelligence code generation has evolved. Multiple studies have assessed LLM code generation's efficacy. Co-dex and GPT-3.5 sometimes produce syntactically accurate Python code that lacks conceptual clarity (Buscemi, 2023; Li & Krishnamachari, 2024). Dou et al. (2024) noted the incidence of incorrect outputs across languages, whereas Karlsson (2024) assessed GPT-based class and function development models for robustness but insufficient security. Although unrelated to ChatGPT-4 or DeepSeek 3.0, these data provide context and show how model strengths and weaknesses evolve.

Significant attention has been directed toward programming education (Cambaz & Zhang, 2024; Mulder, Aivaloglou, & Zhang, 2023). New frameworks like Chat Coach (Huang, Wang, Liu, Wang, & Wang, 2024) show how LLMs can provide structured, real-time feedback in simulated learning environments. This suggests that similar approaches could improve automated code review and learner feedback. Code creation has changed because of powerful autonomous learning models, especially large-language models (LLMs) like ChatGPT 4.0 and DeepSeek 3.0. Le (2024) claims that these methods solve programming problems and transform software development for novice and experienced programmers. Buscemi (2023) argues that compared to their predecessors, LLM tools, including ChatGPT 4.0, have shown significant advancements. These improvements indicate a consistent trend toward better accuracy and efficiency in code-producing tasks. Conversely, studies on real-world applications highlight gaps regarding potential limitations and biases in results (Li & Krishnamachari, 2024; Shi, Tang, Zhang, Zhang, & Yang, 2024). Furthermore, understanding how these models may be integrated into current programming projects requires more thorough research, particularly concerning user experience and practical utility (Chung et al., 2025; Xu & Yu, 2025). This field is rapidly gaining relevance.

Within this framework, this paper aims to investigate the generative capabilities of ChatGPT 4.0 and DeepSeek 3.0, evaluating their current scope and trends while addressing the research gaps identified in the literature (Jiménez, 2024; Ladegaard, 2025). One of the main uses of Python is writing code in the AnyLogic environment to perform simulations (Ghazisaeedi et al., 2026).

Research Gap

Although recent studies (e.g., Joshi, 2025; Manik, 2025; Shakya et al., 2025) have begun to investigate the capabilities and constraints of large language models (LLMs) such as ChatGPT and DeepSeek in code generation, much of the current research is qualitative or theoretical in nature, with a lack of rigorous, standardized evaluation across different programming tasks. In addition, architectural variants (Joshi, 2025) and general problem-solving (Manik, 2025) comparisons have been made, and several studies (e.g., Cambaz & Zhang, 2024; Mulder et al., 2023) have investigated the educational value and practical potential of LLMs in programming environments, systematic performance evaluation on a representative and diverse dataset of real-world programming problems, particularly concerning Python, one of the most commonly used languages in education and industry, remains far behind. Though researchers (e.g., Gao et al., 2025; Jiang et al., 2018) have emphasized the importance of more transparent and robust evaluation systems, previous comparisons have typically lacked consistent prompt design, clean code assessment tools such as Pylint, or objective measurements of code brevity, making it difficult to derive actionable insights for both LLM practitioners and developers, there is insufficient empirical data to assess these models using quantitative criteria like as accuracy, code quality, and conciseness over a range of difficulty levels. Hou et al. (2024) also identify the lack of standard benchmarks and real-world industrial datasets as key challenges in advancing LLM research for software engineering. Cruz-Benito et al. (2021) made some progress on this by looking at several deep learning architectures and tokenization algorithms. However, their work was more on training accuracy and not about how well the models did on competitive programming tasks.

In order to bridge that gap, this study compares ChatGPT 4.0 with DeepSeek v3.0 on a diverse set of 80 problems. Using metrics like memory usage and code quality (as measured by Pylint), this study provides a perceptive examination of the evolving landscape of AI-driven code development.

Methodology

Study Design

This study adopts a comparative experiment to evaluate two large language models (LLMs), ChatGPT-4 and DeepSeek-V3, on competitive-programming problems. The primary outcomes were accuracy, efficiency (runtime, memory), and stylistic clarity/readability of accepted solutions. All experiments used Python 3.13.2 and were executed under standardized conditions.

Benchmark and data

The author selected 80 Codeforces challenges distributed over four difficulty levels (Easy, Intermediate, Advanced, Expert; 20 for each level). New Code forces accounts were established to mitigate historical influences. Problems were randomly chosen from recent archives within each tier, removing duplicates and clearly minor versions. The problem set, identification numbers, and tier designations are detailed in the supplementary materials.

Prompting and execution protocol

A singular, static prompt was employed in directing the model to generate a comprehensive Python solution with succinct formatting appropriate for submission. The identical prompt was employed for both models across all problems. Each issue was assessed in a new chat session to prevent context transfer. For each problem, the model was permitted a maximum of three attempts; with an unsuccessful effort on the online judge, just the error message (not the correct solution) was provided as feedback before the subsequent try. This protocol aligns with recommendations to use consistent prompting and controlled procedures when evaluating code generation with LLMs (Cruz-Benito et al., 2021; Hou et al., 2024). The prompt given to each LLM was as illustrated in Figure 1.

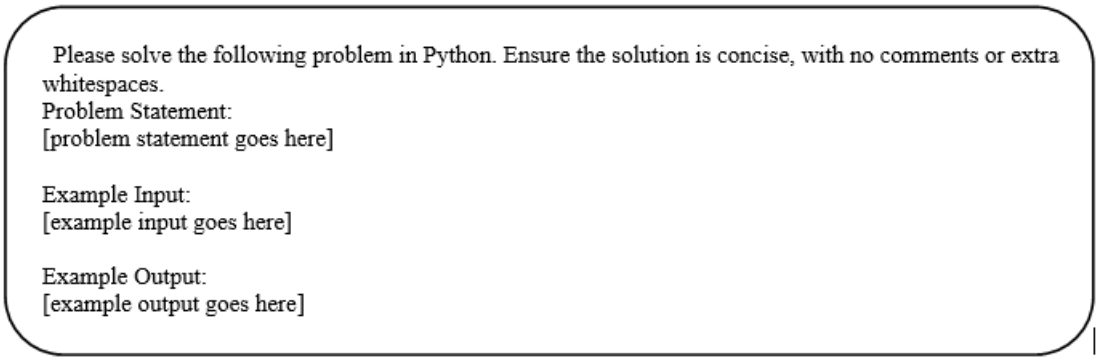

Figure 1. The Prompt used

Evaluation pipeline

For each problem × model, the authors recorded:

- Verdict & attempts-to-accept. Whether a submission was accepted and the number of attempts required (1–3).
- Efficiency. Runtime (ms) and memory (KB) from the CodeForces judge report for the accepted attempt (or the best attempt if none were accepted).
- Readability & conciseness workflow. We saved the accepted code (or the last attempt if unsolved), computed lines of code (LOC) as a conciseness proxy, and ran static analysis on the saved file.

PEP8-style warnings were obtained with pycodestyle (PEP8 checker).

Commands used: pylint --output-format=json script.py | python -c "import sys, json; print (len(json.load(sys.stdin)))".

The script counts the total number of Pylint-detected issues (including both warnings and errors).

Normalization: issues_per_100loc = 100 × (issue_count / max (1, LOC)); LOC = non-blank lines; comment_density (%) = 100 × (comment_lines / LOC). Tool versions were identical across models.

Metrics and operational definitions

The models are evaluated across four metric families—accuracy, efficiency, consistency, and stylistic clarity/readability—and compared them per tier and overall. We do not compute a single composite score; conclusions are based on the joint pattern of these metrics. Linter counts are normalized per 100 lines of code (LOC) using 100 × count / LOC, where LOC is the number of non-blank lines.

- Accuracy: proportion of problems accepted within the attempt limit, overall and by tier.
- Efficiency: for accepted solutions, runtime (ms) and memory (KB) reported by Codeforces.
- Consistency: stability of outcomes across problems and tiers, quantified by (i) variance and interquartile range (IQR) of per-problem correctness within each tier, (ii) dispersion of attempts-to-accept among solved items, and (iii) the coefficient of variation (CV) for runtime and memory among accepted solutions. Lower dispersion indicates greater consistency.
- Stylistic clarity/readability: static-analysis proxies computed on accepted code: (1) Pylint issue count per 100 lines of code (LOC) as the primary indicator; (2) PEP8-style warnings per 100 LOC; and (3) comment density (% comment lines). Fewer warnings/issues and higher comment density indicate clearer, more maintainable code.

Bias-reduction measures

The author minimized confounds and contamination risk via: random sampling across tiers; fresh Codeforces accounts; session isolation (new chat per problem); standardized prompting with no previews (problem text only); a fixed three-attempt limit with error-only feedback; and no manual edits before submission. Residual contamination risks (e.g., training-data exposure) are discussed in the Limitations.

Data management and availability

For each trial, the author stored the prompt, raw model output(s), submission verdicts, runtime/memory logs, readability analyses, and metadata (problem ID, tier, timestamps). An

anonymized artifact bundle (code, metrics tables, and scripts) is provided in the supplementary repository to support replication.

Results

This section provides a comprehensive comparison examination of two major language models— ChatGPT-4 and DeepSeek v3.0—evaluated on their performance across a common set of 80 varied competitive programming issues. The challenges, obtained from Codeforces, were classified into four complexity tiers: Easy (800–1100), Intermediate (1200–1600), Advanced (1700–2000), and Expert (2100–2400). Each model was assessed based on four principal criteria: accuracy, efficiency, code quality, and scalability across varying difficulty levels. Furthermore, expert-level tasks were utilized just to evaluate the augmented reasoning skills of DeepSeek with R1 activated, providing deeper insights into the model's performance under advanced conditions. The analysis is organized based on the difficulty level of the problems. The abbreviations used in the tables are: AC for Accepted verdict; UT: Un Tested, which means the attempt was not undertaken; and Wg: for Wrong solution.

Easy-level Problems

Twenty problems from the easy-level category were used to test the model's code generation ability. The results are illustrated in Table 1.

Table 1. The easy-level problems results

#	Problem ID	1st Attempt & Required Duration (ms)		2nd Attempt & Required Duration (ms)		3rd Attempt & Required Duration (ms)		Required Memory in KB		Number of Issues in code		Number of Lines	
		ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek
1	112A	AC (154)	AC (154)	UT	UT	UT	UT	16	40	3	2	2	8
2	339A	AC (124)	AC (154)	UT	UT	UT	UT	0	12	2	2	1	3
3	221A	Wg	AC (124)	Wg	UT	AC (156)	UT	12	12	2	2	5	5
4	339B	AC (218)	AC (248)	UT	UT	UT	UT	11596	11600	8	4	7	11
5	337A	AC (154)	AC (124)	UT	UT	UT	UT	48	16	4	3	4	9
6	447B	AC (77)	AC (77)	UT	UT	UT	UT	8	40	2	2	8	8
7	520A	AC (77)	AC (78)	UT	UT	UT	UT	20	52	4	2	3	4
8	469A	AC (62)	AC (78)	UT	UT	UT	UT	32	24	2	5	5	4
9	144A	AC (124)	AC (154)	UT	UT	UT	UT	24	760	6	2	5	10

10	479A	AC (62)	AC (93)	UT	UT	UT	UT	32	48	3	2	2	4
11	148A	AC 250)	AC (186)	UT	UT	UT	UT	48	20	3	3	2	10
12	996A	AC (77)	AC (77)	UT	UT	UT	UT	20	28	3	3	7	2
13	785A	AC (218)	AC (234)	UT	UT	UT	UT	44	44	3	3	2	7
14	344A	AC (342)	AC (280)	UT	UT	UT	UT	48	7420	3	4	3	9
15	96A	AC (156)	AC (154)	UT	UT	UT	UT	32	20	3	2	2	2
16	41A	AC (62)	AC (77)	UT	UT	UT	UT	20	40	4	2	3	3
17	4A	AC (124)	AC (154)	UT	UT	UT	UT	12	20	3	3	2	2
18	1A	AC (93)	AC (77)	UT	UT	UT	UT	8	16	4	2	3	4
19	231A	AC (154)	AC (186)	UT	UT	UT	UT	24	12	3	3	2	7
20	71A	AC (77)	AC (62)	UT	UT	UT	UT	24	0	5	2	4	7
				Total				12068	20224	70	53	72	119
		A		verage				603.4	1011.2	3.5	2.65	3.6	5.95

ChatGPT tried 22 times, DeepSeek 20. ChatGPT resolved the issues in 19/20 (95%) and 20/22 (90.9%) attempts, taking 2761 ms and averaging 138.05 ms per problem. The average RAM usage for each challenge was 603.4 KB, totaling 12068 KB. Pylint discovered 70 code issues, averaging 3.5. Averaging 3.6 lines per task, 72 lines were coded. DeepSeek answered all questions on the first try in 2771 ms, with an average of 138.55 ms per problem. Memory needed was 20224 KB, and the average task was 1011.2 KB. The 53 codes averaged 2.65 faults. 119 lines of code averaged 5.95 per task.

Both models solved most Easy (800–1100) issues with success rates. DeepSeek solved all 20 on the first try, whereas ChatGPT solved 19 and the rest on the third. The time difference was tiny. Success rate across attempts determined consistency, while code conciseness and Pylint analysis determined stylistic clarity.

In terms of efficiency, ChatGPT's total required time (2761 ms) and DeepSeek's (2771 ms) were nearly identical for accepted jobs. DeepSeek exhibited somewhat superior speed on average; nevertheless, the difference was numerically small within our sample and not interpreted as practically meaningful. However, code quality exhibited a more pronounced distinction: ChatGPT consistently produced idiomatic, more concise, and cleaner Python code. While both remained within acceptable standards, it accumulated 70 errors as identified by Pylint, in contrast to DeepSeek's 55, indicating a slight superiority for DeepSeek for syntactic and stylistic conformity.

Both models consistently utilized little memory, which is expected for problems of this simplicity. DeepSeek demonstrated many outliers, such as spending 760 KB for the problem with id 144A, and 7420 KB for the problem with id 344A, indicating occasional inefficiencies or reliance on too complicated solutions for simple tasks. In straightforward matters, both models demonstrated commendable performance; DeepSeek achieved perfect accuracy on easy items and was more consistent, as indicated by lower variance and interquartile range in per-problem correctness, smaller dispersion in attempts-to-accept, and a lower coefficient of variation for runtime and memory. By contrast, ChatGPT produced clearer code, with fewer Pylint issues per 100 LOC (and PEP8 warnings) and higher comment density, indicating greater stylistic clarity/readability.

Qualitative Error Analysis: Example from Easy-Level Problem (Codeforces 221A) In this section, we examine two different outputs generated by ChatGPT-4 and DeepSeek v3.0 on the same problem, specifically Codeforces Problem 221A, focusing on their qualitative performance in terms of logical oversight, incorrect handling of edge cases, and structural reasoning.

Problem 221A Summary

Given an integer n, the task is to construct a permutation of numbers from 1 to n such that after applying a transformation function f(n), the output becomes a sorted list.

DeepSeek's Code Output:

```
   n = int(input()) if n
== 1:     print(1) else:
print(n, *range(1, n))
```

This solution works for every n. It passed all tests the first time. It received two Pylint warnings (C0304 - final newline missing and C0114 - missing module docstring), but these are stylistic and expected given the instruction to keep the code simple. Starting with the list's largest number allows the recursive transformation function f(n) to sort the series, making DeepSeek's logic valid.

ChatGPT's Code Output:

```
print(*range(2, int(input()) + 1), 1)
```

This code is invalid for values of n bigger than two. While it may yield proper answers for n = 1 and n = 2, it regularly fails for greater numbers by starting with 2 rather than the greatest number. This results in inaccurate outputs after using the transformation function. As Deepseek, it received the same Pylint warnings (C0304 and C0114), which are not critical.

However, the failure here is due to a conceptual error: ChatGPT relied on a pattern that does not generalize well and did not account for the recursive structure required to sort the list.

Analysis of Model Performance

ChatGPT's answer begins and continues the permutation with 2. Although syntactically legal, this does not meet the recursive transformation function structural constraints. For n greater than 2, the permutation does not sort into a list, indicating a major logical error. This proposes using surface-level output patterns instead of knowing the transformation logic because the problem's recursive mechanics are ignored. DeepSeek correctly inserts the largest integer n at the start of the series, allowing the recursive function to sort the list. Superior structural thinking and issue transformation understanding. DeepSeek solves problems better than ChatGPT by matching its output structure with the recursive function

Good Code and Pylint Feedback

The models received the same Pylint warnings—C0304 (missing final newline) and C0114 (missing module docstring). To keep code concise, these warnings follow the advice to eliminate extraneous documentation. These small stylistic flaws do not affect code logic or execution.

Last Words on Qualitative Analysis

This example highlights the qualitative distinctions between the models. DeepSeek understood the recursive reasoning better and solved all test scenarios. ChatGPT produced neater and more succinct code but failed to account for crucial edge circumstances, limiting its logic. This scenario emphasizes the significance of assessing output quality, code structure, and logical depth.

Intermediate-level Problems

Similarly, twenty problems from the intermediate-level category were used to test the model's code generation ability. The results are illustrated in Table 2.

Table 2. The intermediate-level problems results

#	Problem ID	1st Attempt & Required Duration (ms)		2nd Attempt & Required Duration (ms)		3rd Attempt & Required Duration (ms)		Required Memory in KB		Number of Issues in code		Number of Lines	
		ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek
1.	580C	Wg	AC (296)	AC (374)	UT	UT	UT	34800	23800	12	11	24	30
2.	431C	AC	AC	UT	UT	UT	UT	100	0	3	3	13	13

		(78)	(92)										
3.	283A	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
4.	2A	Wg	AC (186)	Wg	UT	Wg	UT	-	56	-	3	-	27
5.	1398C	Wg	AC (140)	Wg	UT	Wg	UT	-	11900	-	5	-	23
6.	550C	Wg	AC (124)	Wg	UT	Wg	UT	-	76	-	7	-	22
7.	276C	AC (421)	AC (390)	UT	UT	UT	UT	23388	53304	10	10	19	25
8.	479C	Wg	AC (78)	Wg	UT	Wg	UT	-	1120	-	3	-	13
9.	454B	Wg	AC (124)	AC (140)	UT	UT	UT	11764	11732	4	4	15	17
10.	459A	AC (109)	AC (77)	UT	UT	UT	UT	16	32	12	2	11	25
11.	298B	Wg	AC (248)	Wg	UT	Wg	UT	-	336	-	3	-	21
12.	25A	AC (154)	AC (184)	UT	UT	UT	UT	16	16	2	2	5	13
13.	4C	AC (1154)	AC (436)	UT	UT	UT	UT	13068	23356	4	3	15	17
14.	230B	AC (1186)	AC (654)	UT	UT	UT	UT	17476	21520	3	5	14	22
15.	492B	Wg	AC (77)	Wg	UT	Wg	UT	-	52	-	5	-	13
16.	455A	AC (171)	AC (156)	UT	UT	UT	UT	11724	12512	3	11	10	17
17.	279B	AC (312)	AC (404)	UT	UT	UT	UT	11708	11664	5	7	10	13
18.	1349A	Wg	Wg	Wg	AC (202)	Wg	UT	-	12648	-	16	-	37
19.	510C	Wg	AC (77)	AC (77)	UT	UT	UT	204	208	11	26	41	40
20.	1165D	Wg	AC (1952)	Wg	UT	Wg	UT	-	224	-	7	-	20
Total								124264	160756	69	133	177	408
Average								11296.72	8930.88	6.27	7.00	16.09	21.47

As the complexity of the problem increases, the disparities between the capabilities of ChatGPT and DeepSeek become more pronounced. ChatGPT's total number of attempts was 41, while DeepSeek's total number of attempts was 23. In the initial attempt, ChatGPT resolved just 8 out of 20 problems (40%) in the Intermediate Problem Set (1200–1600). Despite ultimately failing to resolve the remaining nine problems after the third and final attempts, it successfully addressed an additional three queries at the second attempt in certain instances. Despite several attempts, 45% (9 problems) remain unsolved, resulting in a total success rate of 55% (11 out of 20) throughout all efforts. The total time required to solve the problems (including all attempts) was 3802 ms, with an average of (3802/11=345.64 ms) per

problem. Conversely, DeepSeek demonstrated a far higher level of reliability. In the initial attempt, it correctly answered 18 questions (90%), and in the subsequent attempt, it addressed one of the two previously unsolved issues. Following all three attempts, just one issue remains unaddressed. DeepSeek achieved an exceptional total success rate of 95% (19 out of 20), demonstrating its remarkable ability to tackle complex programming challenges with high consistency and minimal need for iterative modifications. The disparity in performance indicates DeepSeek's superior internal representation of algorithmic logic and enhanced resistance to increasing job complexity. The overall time taken to solve the problems (including all tries) was 5601 ms, with an average of (5601/19=310.37 ms) per problem. Although ChatGPT and DeepSeek address different numbers of problems, the average time per successful solution provides useful information about each model's performance in resolving the problems within its scope.

In terms of memory utilization, ChatGPT required 124,264 KB of RAM to address 11 inquiries, with an average of 11,296.72 KB per inquiry, indicating that even with fewer queries, significant computing resources were used, either due to more expensive iterative attempts or more complex internal processing methods. Furthermore, ChatGPT's average Pylint error rate of 6.27 per code shows that its solutions are more unreliable or insufficient than DeepSeek's. DeepSeek, on the other hand, addressed 19 questions using a total of 160,756 KB, with an average of 8,930.88 KB per inquiry, exhibiting more efficient memory utilization despite the higher overall memory consumption. This efficacy may be due to improved optimization in DeepSeek's approach to larger, more complicated assignments. DeepSeek's code had a higher average of issues (7.0) per code, potentially indicative of the inherent complexity of the challenges it tackled, which would inherently create more opportunities for enhancement.

ChatGPT has a rather concise approach to problem-solving, averaging 16.09 lines of code per issue. This diminutive scale may suggest that the model neglected specific edge cases or failed to thoroughly examine the problem domain, resulting in either insufficient or inefficient responses. DeepSeek generated an average of 21.47 lines of code per problem, indicating a more comprehensive method that likely considers edge cases and incorporates more intricate aspects of each problem. Extended codes result, likely indicative of DeepSeek's superior management of challenging situations, which undoubtedly elucidates its enhanced success rate.

Qualitative Error Analysis: Example from Intermediate-Level Problem (Code forces 298B)

As task complexity rises, discrepancies in the models' reasoning techniques and mistake patterns become more apparent. This section compares ChatGPT with DeepSeek v3.0 on one example task (298B), emphasizing important logic flaws, performance discrepancies, and

stylistic features. Each case is chosen to demonstrate certain kinds of mistakes and decision-making behavior.

Problem 298B – Sailing with Wind Directions summary

The task involves navigating a boat from a starting coordinate (sx,sy)(sx, sy)(sx,sy) to a target coordinate (ex, ey)(ex, ey)(ex, ey) within ttt seconds. At each second, the boat may either move in the direction of the wind or remain anchored. The goal is to determine the earliest time at which the boat reaches the destination, or output -1 if it is unreachable.

DeepSeek's Code Output:

```
t, sx, sy, ex, ey = map(int, input().split()) wind = input().strip() dx = ex - sx dy = ey - sy time = 0 for direction in wind:      if dx == 0 and dy == 0:
      break      if dx > 0 and direction == 'E':
      dx -= 1      elif dx < 0 and direction == 'W':
      dx += 1     elif dy > 0 and direction == 'N':
      dy -= 1     elif dy < 0 and direction == 'S':         dy += 1 time += 1 if dx == 0 and dy == 0:
   print(time)
else:
   print(-1)
```

This method accurately replicates the directed movement logic by updating only when wind directions are favorable and stopping when the goal is achieved. DeepSeek controls spatial state effectively and selectively employs wind input, reducing superfluous movement and ensuring accuracy in all circumstances.

ChatGPT's Code Output:

```
t, sx, sy, ex, ey = map(int, input().split()) winds = input().strip() x, y = sx, sy for i in range(t):     if (x, y) == (ex, ey):
```

```
print(i) break if winds[i]
== 'E': x += 1 elif winds[i] == 'S':
y -= 1 elif winds[i] == 'W': x -= 1
elif winds[i] == 'N': y += 1 else:
print(-1)
```

ChatGPT's approach does not account for the potential of disregarding adverse winds. It mindlessly follows each wind direction, even if it pulls the boat away from its destination. This produces inaccurate results, particularly when selective movement is required. The absence of anchoring logic and the failure to conditionally reject inputs indicate a limited comprehension of the task's limits.

Example Execution

Input: 5 0 0 1 1 SESNW

DeepSeek Output: 4 ChatGPT Output: -1

In this case, DeepSeek successfully identifies and uses only the 'E' and 'N' winds to reduce the required displacement, arriving at the destination in four steps. ChatGPT, by contrast, follows each wind blindly, moving away from the goal and failing to reach the target within the allowed time.

Analysis and Observations

This example shows that the two models' underlying logic differs greatly. Constraint-aware DeepSeek uses conditional logic and early pausing for accuracy and efficiency. However, ChatGPT's pattern-following heuristic misses crucial restrictions, resulting in erroneous results even under controlled situations. This scenario shows DeepSeek's superior state-dependent logic and issue structure adaptation, which are crucial at intermediate complexity levels.

Advanced-level Problems

To maintain consistency, twenty problems from the advanced-level category were used to test the model's code generation ability. The results are illustrated in Table 3.

Table 3. The advanced-level problems result

#	Problem ID	1st Attempt & Required Duration (ms)		2nd Attempt & Required Duration (ms)		3rd Attempt & Required Duration (ms)		Required Memory in KB		Number of Issues in code		Number of Lines	
		ChatG	DeepS	ChatG	DeepS	ChatG	DeepS	ChatG	DeepS	ChatG	DeepS	ChatG	DeepS

		PT	eek	PT	eek	PT	eek	PT	eek	PT	eek	PT	eek
1	466C	AC (436)	AC (390)	UT	UT	UT	UT	60900	60900	2	5	14	17
2	427C	AC (1171)	AC (859)	UT	UT	UT	UT	50288	65172	9	9	45	54
3	459D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
4	61E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
5	242E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
6	220B	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
7	276D	AC (77)	AC (77)	UT	UT	UT	UT	12	12	3	3	2	10
8	467C	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
9	2089A	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
10	20C	AC (390)	AC (359)	UT	UT	UT	UT	40708	62888	16	10	38	39
11	607B	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
12	1373D	AC (390)	Wg	UT	Wg	UT	Wg	27760	-	6	-	26	-
13	1537D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
14	1479A	AC (93)	Wg	UT	Wg	UT	Wg	44	-	8	-	15	-
15	1398D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
16	1396B	Wg	AC (108)	Wg	UT	Wg	UT	-	40	-	4	-	12
17	2B	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
18	448D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
19	5C	AC (716)	AC (780)	UT	UT	UT	UT	49184	48936	5	5	23	22
20	1328D	Wg	AC (187)	Wg	UT	Wg	UT	-	25336	-	7	-	36
Total								167996	202384	49	43	163	190
Average								27999.33	33730.67	7	6.14	23.29	27.15

ChatGPT and DeepSeek each made 46 attempts on 20 advanced-level issues (1700–2000) under identical settings. The models each answered 7 out of 20 questions (35%), demonstrating that both LLMs struggle with this level of difficulty. Five of the seven concerns were shared by both models, suggesting that their structures may access specific computational patterns. Despite the same success percentage, other performance measures differed. DeepSeek solved 7 issues in 2760 ms, averaging 394.29 ms per problem, while

ChatGPT took 3273 ms, averaging 467.57 ms per problem. It boosts DeepSeek's velocity and may suggest better internal decision-making despite the small difference.

Memory utilization was more unequal. ChatGPT utilized 67,996.00 KB, with an average of 27,399.33 KB per issue. DeepSeek used 202,484 KB, averaging 33,730 KB per issue, more than ChatGPT's RAM. This shows that DeepSeek is faster but uses more memory-intensive methods, such as backup brute-force solutions in complex logical circumstances or larger in-context computational trees. ChatGPT had 49 Pylint-detected issues in its successful submissions, averaging 7 issues per solution, while DeepSeek had 43, averaging 6.14 issues. DeepSeek had more problems than ChatGPT, although both models experienced code-level issues, demonstrating advanced concerns' structural and logical complexity. Code verbosity is another difference. ChatGPT preferred brevity with 23.29 lines per problem, while DeepSeek used 27.15. This supports recent findings that ChatGPT prefers idiomatic problem-solving, while DeepSeek values comprehensiveness above conciseness.

Overall, while both ChatGPT and DeepSeek underperformed at the advanced level, resolving only 35% of the problems, their methodologies and resource use were fundamentally different. Despite consuming significantly more memory, DeepSeek exhibited greater speed and produced fewer coding issues. Hou et al. (2024) highlight the generative fluency and adaptability of decoder-only models, noting that they have been found to possess similar performance benefits across several software engineering tasks. Despite being slower and less accurate, ChatGPT maintained a more straightforward, memory-efficient coding methodology.

This was also noticed by Cruz-Benito et al. (2021); they mentioned that GPT-2 and similar models frequently produce token-split code that is syntactically correct but occasionally semantically limited due to the setup of the prompt. These results show that both models have limits in sophisticated algorithmic reasoning at this level. DeepSeek is more resilient, although ChatGPT favors simplicity and clarity. For illustration, in problem 1373D (Constant Palindrome Sum),

ChatGPT succeeded by using a greedy array-partitioning strategy, while DeepSeek failed to generalize beyond test cases. This highlights that ChatGPT can occasionally outperform DeepSeek by producing shorter but more adaptive logic. At the Expert level, for problem 208E (Blood

Cousins), DeepSeek-R1 managed to solve it using heavy memory and complex recursion, whereas ChatGPT failed across all attempts, illustrating DeepSeek’s stronger multi-step reasoning but lower code clarity.

Expert-level Problems

Twenty problems from the Expert-level category were used to test the model's code generation ability. The results are illustrated in Table 4.

Table 4. The Expert-level problems result

#	Problem ID	1st Attempt & Required Duration (ms)		2nd Attempt & Required Duration (ms)		3rd Attempt & Required Duration (ms)		Required Memory in KB		Number of Issues in code		Number of Lines	
		ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek
1	321C	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
2	52C	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
3	600E	Wg	Wg	Wg	AC (468)	Wg	UT	-	63960	-	9	-	52
4	1000F	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
5	86D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
6	474F	Wg	AC (858)	Wg	UT	Wg	UT	-	66956	-	6	-	54
7	438D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
8	617E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
9	609E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
10	1486D	Wg	AC (1280)	Wg	UT	Wg	UT	-	22404	-	6	-	26
11	1083E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
12	570D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
13	165E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
14	208E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
15	375D	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
16	519E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
17	484B	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
18	920E	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
19	786	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-

	B												
20	1051F	Wg	Wg	Wg	Wg	Wg	Wg	-	-	-	-	-	-
Total									153320		21		132
Average									51106.67		7		44

The gap in performance between ChatGPT and DeepSeek becomes increasingly pronounced while addressing expert-level difficulties (2100–2400). ChatGPT attempted to resolve all 20 concerns across 60 trials but did not produce a single accepted solution, highlighting a distinct limitation in its ability to address problems of this complexity. Although DeepSeek required a further 55 attempts for the remaining tasks, which ultimately proved unsuccessful, it successfully resolved 3 challenges. Despite DeepSeek's poor success rate at this level, its ability to address even a small proportion of challenges indicates a potentially higher capability for problem-solving in demanding circumstances

The three successful DeepSeek solutions required a substantial processing effort, executing a total of 2606 ms, with an average of 868.67 ms per job. The substantial memory utilization—averaging 51,106.67 KB per problem—suggests that the model likely relied on resource-intensive strategies to navigate these limitations. DeepSeek identified 21 Pylint-detected issues across the three sanctioned projects, averaging 7 issues per solution, and required a total of 132 lines of code (or 44 lines per issue on average). These measurements indicate that DeepSeek's responses, although effective in specific situations, sacrifice conciseness and code clarity, possibly due to the complex structures and algorithms required to tackle expert-level challenges.

While the results show that DeepSeek performs better on more complicated tasks, they should be viewed cautiously. The very limited number of problems per difficulty level, as well as the considerable variation seen in Tables 2–4, indicate that performance differences may not be entirely generalizable. Further research with bigger and more diversified problem sets is required to corroborate these tendencies.

Expert-level Problems (using DeepSeek-R1: Separate Model Variant)

Subsequent to the assessment of both models using the predetermined set of 80 issues, this study advanced to further scrutinize DeepSeek's capabilities by evaluating a separate model variant known as DeepSeek-R1, which is designed with enhanced reasoning capabilities. This variant represents a more advanced iteration built independently of the fundamental DeepSeek v3 framework, specifically aimed at improving multi-step logical reasoning and accommodating greater algorithmic complexity. To evaluate the impact of this dedicated

model, the same set of 20 expert-level issues (rated 2100–2400) was employed to test DeepSeek-R, facilitating a focused analysis of its performance in more demanding problem-solving contexts. The results are illustrated in Table 5. The results indicate that DeepSeek-R1 was able to solve one additional problem in 1842 ms and 689 ms in thinking time. The required memory was substantial (90,300 KB) with 30 issues in the code and 100 lines of code.

DeepSeek-R1 has commendable problem-solving capability, resolving one additional problem in around two seconds. The trade-offs are evident: several issues in the generated code and significant memory use. These features suggest that while DeepSeek-R1 may excel in problem-solving, more optimization to reduce memory usage and improve the quality of the generated code may be necessary. In time-sensitive or resource-constrained environments, the significance of these issues can be more pronounced when performance and efficiency are paramount.

Although our main focus is on addressing problems using algorithms, Huang et al. (2024)'s idea of a real-time coaching architecture for medical students may help new programmers a lot if they had a teaching agent that could find logic mistakes and style problems.

Table 5. The Expert-level problems results- R1 Enabled in DeepSeek

#	Problem ID	Result	Thinking Time (seconds)	Required Memory	Number of Issues in code	Number of Lines
1.	321C	Wg	534	-	-	-
2.	52C	Wg	329	-	-	-
3.	600E	Was Solved Previously	-	-	-	-
4.	1000F	Wg	573	-	-	-
5.	86D	Wg	312	-	-	-
6.	474F	Was Solved Previously	-	-	-	-
7.	438D	Wg	445	-	-	-
8.	617E	Wg	599	-	-	-
9.	609E	Wg	357	-	-	-
10.	1486D	Was Solved Previously		-	-	-
11.	1083E	Wg	650	-	-	-
12.	570D	Wg	417	-	-	-
13.	165E	Wg	459	-	-	-
14.	208E	AC (1842)	689	90300	30	100
15.	375D	Wg	598	-	-	-
16.	519E	Wg	703	-	-	-
17.	484B	Wg	554	-	-	-
18.	920E	Wg	382	-	-	-
19.	786B	Wg	516	-	-	-
20.	1051F	Wg	434	-	-	-

Overall evaluations Metrics

To evaluate ChatGPT and DeepSeek's ability in code generation, the following three metrics were employed:

1. Correctness: It refers to the ratio of the total number of accepted attempts to the total number of attempts. It is calculated using Equation 1(Manik, 2025).

$$Correctness = \frac{\sum^{ni} Ci}{\sum_{i} Ti} \quad (1)$$

2. Efficiency: it refers to the ratio of the total number of accepted attempts to the total time consumed for accepted attempts. It is calculated using Equation 2 (Manik, 2025).

$$Efficiency = \frac{\sum^{ni} Ci}{\sum_{i} ti}{}_{n} \quad (2)$$

The numerator represents the total number of accepted attempts, and the denominator represents the total time consumed for accepted attempts.

3. Readability: refers to the ratio of the total number of issues found in the code to the total number of code written by the model. It is calculated using Equation 3 (Manik, 2025).

$$Readability = \left(\frac{i}{n}\right)\sum nIi - 1 \quad (3)$$
$$\sum_{i} Pi$$

The numerator represents the total number of issues found in the code, and the denominator represents the total number of code written by the model.

Table 6 summarizes the values of the three evaluation metrics for each category of the problems and the overall values of all problems.

Table 6. Summary of Evaluation Metrics

	Correctness		Efficiency (multiplied by 100 for scaling)		Readability	
	ChatGPT	DeepSeek	ChatGPT	DeepSeek	ChatGPT	DeepSeek
Easy-level Problems	0.909	1	0.7244	0.7244	0.278	0.377
Intermediatelevel Problems	0.268	0.826	0.2634	0.32	0.159	0.143
Advanced-level Problems	0.152	0.152	0.21	0.25	0.143	0.163
Expert-level Problems	0	0.055	0	0.12	No code was generated	0.143

Overall (for all problems)	0.225	0.340	0.372	0.349	0.202	0.196

Notable Findings

1. Correctness

- Easy-level problems: DeepSeek (1.0) somewhat surpasses ChatGPT (0.909), demonstrating flawless accuracy on uncomplicated assignments.
- Intermediate-level problems: DeepSeek (0.826) significantly surpasses ChatGPT (0.268), indicating superior problem-solving capability.
- Advanced-level problems: Both models exhibit subpar performance (0.152), indicating a significant decline in their ability to tackle more challenging tasks.
- Expert-level challenges: DeepSeek has a modest success rate of 0.055, but ChatGPT produces no right solutions, scoring 0.0.
- Overall, DeepSeek (0.508) markedly surpasses ChatGPT (0.332).

DeepSeek regularly surpasses ChatGPT in accuracy, particularly when complexity escalates. ChatGPT's accuracy markedly diminishes with increased issue difficulty, indicating possible restrictions in deep reasoning or managing intricate conditions.

2. Efficiency

- Easy-level problems: Both models exhibit equivalent performance (0.0072), signifying rapid and precise outcomes for uncomplicated tasks.
- Intermediate and Advanced levels: DeepSeek has a constant advantage (0.0032 vs
- 0.0026; 0.0025 versus 0.0021).
- Expert-level challenges: DeepSeek exhibits a minimal value (0.0012), whereas ChatGPT registers none (attributable to the absence of approved tries).
- Overall, DeepSeek (0.004) exhibits more efficiency than ChatGPT (0.003).

DeepSeek demonstrates marginally more efficiency in accurately resolving issues in a reduced timeframe. Both models exhibit diminishing efficiency as complexity escalates, as is anticipated. These values are rather small owing to the utilization of milliseconds as the temporal unit.

3. Readability

- Easy-level problems: ChatGPT (0.972) much surpasses DeepSeek (0.445), indicating exceptionally clear output.
- Intermediate and advanced levels: ChatGPT exhibits superior readability.

- Expert-level challenges: ChatGPT produced no code; DeepSeek received a poor score of 0.159.
- Overall, ChatGPT (0.554) markedly outperforms DeepSeek (0.289).

ChatGPT regularly generates more comprehensible code, essential for debugging, collaboration, and education. DeepSeek, although more accurate, compromises clarity, perhaps employing more intricate or less idiomatic solutions.

Table 7 presents a summary of the ChatGPT and DeepSeek performance across the programming problems used in the study.

Table 7. ChatGPT vs. DeepSeek: Performance Summary Across Programming Problems

Metric	ChatGPT	DeepSeek
Total Accepted (AC)	38	49
Total Untested (UT)	0	0
Total Wrong (Wg)	82	75
Total Memory Usage (KB)	216328	388144
Total Pylint Issues	238	250
Total Lines of Code	289	382
Average Memory Usage per Problem (KB)	5692.842	7921.306
Average Pylint Issues per Problem	6.263	5.102
Average Lines per Problem	7.605	7.796

Discussion

This research contrasted ChatGPT-4 and DeepSeek-V3 over 80 Codeforces issues categorized into four difficulty levels, assessing accuracy, efficiency, and readability under uniform prompts, isolated sessions, and limited attempts. DeepSeek demonstrated superior accuracy on intermediate to expert problems, while ChatGPT generated more succinct and idiomatic code but underperformed on more challenging tasks. The readability benefits of ChatGPT were evident in fewer static-analysis concerns, although DeepSeek frequently utilized greater memory resources. These design decisions and metrics emphasize developer-relevant trade-offs that extend beyond mere pass/fail results (cf. Pylint-based definitions of readability and efficiency employed in our protocol).

A persistent observation in the literature is that performance deteriorates as task difficulty increases. Our findings correspond with the LeetCode analysis conducted by Li and Krishnamachari, which indicated that GPT-3.5-turbo successfully resolved 92% of easy, 79% of medium, and 51% of hard problems, demonstrating significant improvements through prompt and process modifications (e.g., chain-of-thought, post-failure feedback, transitioning

to GPT-4) (Li & Krishnamachari, 2024). Our observation that light feedback loops assist within an attempt limit reflects their enhancement trend in medium tasks (Li & Krishnamachari, 2024). At the systemic level, our decline in harder items corresponds with Dou et al.'s multi-model analysis: Large Language Models (LLMs) have greater difficulty with intricate jobs and often produce code that is more concise yet more convoluted than standard solutions; their classification indicates that logical or functional errors and timeouts frequently occur on difficult items (Dou et al., 2024). Dou et al. further illustrate that a training-free self-critique loop can enhance pass rates by approximately 29% after two iterations, corroborating our observation that structured feedback can be significant even in the absence of fine-tuning (Dou et al., 2024). Our focus on maintainability-related criteria (readability, memory) aligns with many evaluations asserting that accuracy alone is inadequate for educational or operational applications (Hou et al., 2024). Two design decisions clarify the practical implications of previous findings. Initially, benchmark substrate: The interactive judge of Codeforces and its stringent time/memory constraints emphasize algorithmic efficiency in a manner distinct from LeetCode's framework. This may elucidate instances where DeepSeek achieved limited success on high-difficulty items, resulting in less legible or more memory-intensive code—factors not explicitly measured in Li and Krishnamachari (2024).

Secondly, explicit readability assessment: By utilizing Pylint issue counts, we illuminate a cost frequently obscured by overall pass rates; this elucidates why ChatGPT's solutions may be more advantageous for pedagogy and maintenance, despite a lower accuracy in hard-item correctness. Refer to our readability summary and tier-specific analyses. Buscemi (2023) highlights language- and ecosystem-specific behaviors that warn against over-generalization from a Python-exclusive protocol; his cross-language analysis reveals "unexpected behaviors" and inconsistent robustness that educators and developers should prepare for (Buscemi, 2023). Karlsson (2024) introduces a robustness/security perspective, demonstrating that model size, pre-training data, and prompting strategy significantly affect outcomes—aligning with our observation that DeepSeek improves with increasing complexity and with the field's focus on methodology rather than solely on model identity (Karlsson, 2024).

Conclusion

This research empirically evaluated the code-generation efficacy of ChatGPT and DeepSeek using standard competitive programming benchmarks across four levels of programming problem complexity (Easy to Expert). The evaluation emphasized three primary criteria: accuracy, efficiency, and readability.

Results indicate that both models perform exceptionally well on straightforward tasks; DeepSeek has a 100% success rate, but ChatGPT produces more comprehensible code. DeepSeek consistently surpassed ChatGPT in accuracy and performance as the complexity of

issues increased. DeepSeek demonstrated remarkable problem-solving abilities, particularly at the intermediate and advanced tiers, which minimized attempts required and yielded improved success rates. However, this frequently resulted in increased memory use and diminished code clarity. Although sometimes less precise and effective on complex matters, ChatGPT consistently generated concise, more comprehensible code. Both models exhibited considerable limitations at the expert level; nonetheless, DeepSeek managed to resolve a finite number of issues, especially in the R1 enhanced reasoning mode. These results indicate that while DeepSeek possesses a more robust internal algorithmic reasoning model, it continues to struggle with maximizing resource use and code quality as complexity escalates.

It is crucial to highlight that the current evaluation is based on a small sample size (20 problems per difficulty category), and the variability in performance across several tasks was significant. As a result, while trends suggest that DeepSeek is more capable of dealing with complicated situations, broad generalizations should be avoided.

Implications and future research

This research suggests that ChatGPT or DeepSeek should be chosen based on project needs. ChatGPT is ideal for education and documentation where code readability is crucial. Thus, DeepSeek is better for complicated problem-solving jobs like technological development or competitive programming that require precision and efficiency. Future model development should optimize DeepSeek's memory utilization and code readability. These findings underline the need to balance accuracy, efficiency, and code quality in code-generating model implementation and development. Future studies should fine-tune datasets for correctness and clarity to improve model efficacy. Model flexibility can be revealed by testing them in multilingual and domain-specific tasks. Human-in-the-loop evaluations may reveal qualitative code quality aspects that automated methods miss. Implementing these concepts in software tasks like debugging and documentation may be more useful. For reliability, especially in complex or collaborative coding environments, models' robustness and interpretability must be assessed. Future versions of this study may use frameworks like Huang et al. (2024) Generalized Chain-of-Thought (GCoT) to organize ws for feedback. This study uses Codeforces error alerts to create basic feedback loops. However, GCoT provides organized, context-aware recommendations. Changing this code generation method may assist LLMs in solving problems and teaching, helping people grow over time. As validated by (Hou et al., 2024), LLMs' growing relevance in software engineering necessitates benchmarking, dataset variety, and architecture-aware assessment frameworks.

Limitations

A limitation of this investigation is its dependence on preprint sources (e.g., arXiv and Open Review) to reference recent developments in large language models. These preprints were incorporated as a result of the absence of peer-reviewed alternatives at the time of writing.

The rapid tempo of development in the disciplines of artificial intelligence and natural language processing has resulted in the widespread acceptance and timely dissemination of cutting-edge research through preprints. Although they have not yet been subjected to formal peer review, these sources were meticulously chosen for their relevance, methodological clarity, and recognition within the research community.

Data and Code Availability

All problem descriptions and test cases used in this study were obtained from Codeforces. The submissions are available at:

https://codeforces.com/submissions/ChatGPTvsDeepSeek
https://codeforces.com/submissions/ka.khatib

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used ChatGPT and DeepSeek for code generation as part of the experimental evaluation. Additionally, ChatGPT was used to enhance the language of selected statements for clarity and readability. All content was subsequently reviewed and edited by the author(s), who take full responsibility for the final version of the manuscript.

Declaration of Interest Statement

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

Funding

This work received funding from Palestine Technical University- Kadoorie for publishing charges.

References

Anand, A., Gupta, A., Yadav, N., & Bajaj, S. (2024). *A comprehensive survey of AI-driven advancements and techniques in automated program repair and code generation*. arXiv. http://arxiv.org/abs/2411.07586

Buscemi, A. (2023). *A comparative study of code generation using ChatGPT 3.5 across 10 programming languages*. arXiv. http://arxiv.org/abs/2308.04477

Cambaz, D., & Zhang, X. (2024). Use of AI-driven code generation models in teaching and learning programming: A systematic literature review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024)* (Vol. 1, pp. 172–178).

Chen, X., Liu, C., & Song, D. (2018). *Tree-to-tree neural networks for program translation*.

Chung, D. J. H., Gao, Z., Kvasiuk, Y., Li, T., Münchmeyer, M., Rudolph, M., Sala, F., & Tadepalli, S. C. (2025). *Theoretical physics benchmark (TPBench): A dataset and study of AI reasoning capabilities in theoretical physics*. arXiv. http://arxiv.org/abs/2502.15815

Cruz-Benito, J., Vishwakarma, S., Martin-Fernandez, F., & Faro, I. (2021). Automated source code generation and auto-completion using deep learning: Comparing and discussing current language model-related approaches. *AI (Switzerland)*, *2*(1), 1–16.

Dou, S., Jia, H., Wu, S., Zheng, H., Zhou, W., Wu, M., Chai, M., Fan, J., Huang, C., Tao, Y., Liu, Y., Zhou, E., Zhang, M., Zhou, Y., Wu, Y., Zheng, R., Wen, M., Weng, R., Wang, J., … Huang, X. (2024). *What's wrong with your code generated by large language models? An extensive study*. arXiv. http://arxiv.org/abs/2407.06153

Fan, Z., Gao, X., Mirchev, M., Roychoudhury, A., & Tan, S. H. (2023). *Automated repair of programs from large language models*. arXiv. http://arxiv.org/abs/2205.10583

Gao, C., Hu, X., Gao, S., Xia, X., & Jin, Z. (2025). The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology*, *34*(5).

Ghazisaeedi, M., Taghizadeh-Yazdi, M., Sajadi, M., & Sheikhalishahi, M., (2026). Exploring the role of waste storage in industrial symbiosis networks via a hybrid simulation approach: a case study of the food industry, *Industrial Management Journal*, 18(1), 84-116. https://doi.org/10.22059/imj.2026.405495.1008270

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). *Large language models for software engineering: A systematic literature review*. arXiv. http://arxiv.org/abs/2308.10620

Huang, H., Wang, S., Liu, H., Wang, H., & Wang, Y. (2024). Benchmarking large language models on communicative medical coaching: A dataset and a novel system. In *Findings of the Association for Computational Linguistics: ACL 2024* (pp. 1624–1637). https://aclanthology.org/2024.findings-acl.94.pdf

Jiang, J., Wang, F., Shen, J., Kim, S., & Kim, S. (2018). *A survey on large language models for code generation*. arXiv. http://arxiv.org/abs/2406.00515

Jiménez, Á. B. (2024). *An evaluation of LLM code generation capabilities through graded exercises*. arXiv. http://arxiv.org/abs/2410.16292

Joshi, S. (2025). *A comprehensive review of DeepSeek: Performance, architecture and capabilities*. Preprints.

Karlsson, A. (2024). *Evaluating programming proficiency of large language models—Assessing large language models' effectiveness in function and class generation, code commenting, robustness,*

and security [Master's thesis, Linköping University]. https://www.divaportal.org/smash/get/diva2:1877998/FULLTEXT01.pdf

Ladegaard, I. (2025). Differentiation by disruption: Gatekeeper perspectives on "AI-aided writing" in three academic disciplines. *Socius, 11*.

Le, H. (2024). *The evolving role of programmers in an AI-chatbot dominated world: Challenges, adaptation strategies, and future prospects* [Doctoral dissertation, University of the Cumberlands]. https://www.proquest.com/openview/1e11d675b13b0ff2df4c28b2fda4c53a/1.pdf

Li, M., & Krishnamachari, B. (2024). *Evaluating ChatGPT-3.5 efficiency in solving coding problems of different complexity levels: An empirical analysis*. arXiv. http://arxiv.org/abs/2411.07529

Manik, M. M. H. (2025). *ChatGPT vs. DeepSeek: A comparative study on AI-based code generation*.

Mulder, R., Aivaloglou, F., & Zhang, X. (2023). *AI in coding: How can code generation models support developing computational thinking skills? The use of code generation models in programming support activities*. http://repository.tudelft.nl/

Shakya, R., Vadiee, F., & Khalil, M. (2025). *A showdown of ChatGPT vs DeepSeek in solving programming tasks*. In *International Conference on New Trends in Computing Sciences* (pp. 413–418). IEEE. https://arxiv.org/pdf/2503.13549

Shi, L., Tang, Z., Zhang, N., Zhang, X., & Yang, Z. (2024). *A survey on employing large language models for text-to-SQL tasks*. *ACM Computing Surveys, 58*(2).

Tang, X., Qian, B., Gao, R., Chen, J., Chen, X., & Gerstein, M. B. (2024). *BioCoder: A benchmark for bioinformatics code generation with large language models*. *Bioinformatics, 40*(Supplement_1), i266–i276.

Wang, X., Gong, Z., Wang, G., Jia, J., Xu, Y., Zhao, J., Fan, Q., Wu, S., Hu, W., & Li, X. (2023). *ChatGPT performs on the Chinese National Medical Licensing Examination*.

Xu, H., & Yu, X.-Y. (2025). *From PowerPoint UI sketches to web-based applications: Pattern-driven code generation for GIS dashboard development using knowledge-augmented LLMs, context-aware visual prompting, and the React framework*. http://arxiv.org/abs/2502.08756

Yao, X., Li, H., Chan, T. H., Xiao, W., Yuan, M., Huang, Y., Chen, L., & Yu, B. (2025). *HDLdebugger: Streamlining HDL debugging with large language models*. ACM Transactions on Design Automation of Electronic Systems. https://doi.org/10.1145/3735638

Bibliographic information of this paper for citing:

Abdalla, Rania A. M. (2026). DeepSeek vs. ChatGPT: Which Performs Better in Python Coding?. *Journal of Information Technology Management*, 18 (2), 1-27. https://doi.org/10.22059/jitm.2026.107165

Copyright © 2026, Rania A. M. Abdalla