VFAST Transactions on Software Engineering
https://vfast.org/journals/index.php/VTSE@ 2025, ISSN(e): 2309-3978, ISSN(p): 2411-6246
Volume 13, Number 4, October-December 2025 pp:68-81

Complexity Analysis of LLM-Generated Recursive Code: A Systematic Evaluation

Khalida Shaheen [1*], **Muhammad Shumail Naveed** [1], **Anwar Ali Sanjrani** [1], **Shafaque Saira Malik** [1], **Samina Azeem** [2]

[1]Department of Computer Science & Information Technology, University of Balochistan, Quetta, Pakistan; [2]Department of Computer Science, Sardar Bahadur Khan Women's University, Quetta, Pakistan

Keywords:
Programming, Recursion, ChatGPT, DeepSeek, Gemini, Halstead Complexity Metrics, Cyclomatic Complexity.

Journal Info:
Submitted: October 25, 2025
Accepted: November 29, 2025
Published: December 14, 2025

Abstract

Programming is an essential skill, but it can be difficult for beginners, especially when it comes to logical concepts like recursion. Despite the development of many computational and pedagogical methods to simplify programming, recursion remains a challenging topic to understand, implement, and debug. Artificial intelligence has led to the development of large language models (LLMs), such as ChatGPT, Gemini, and DeepSeek that can generate programming source code. Various studies have analyzed the quality of code produced by LLM. However, the complexity of the recursive code generated by these models has not been studied. This study compared and analyzed recursive Python programs generated by Gemini (2.5 Pro), DeepSeek (V3.1) and ChatGPT (GPT-5) in an attempt to fill this gap. For the study, 250 programs generated by each model were examined using Halsted and cyclomatic complexity metrics. The results showed that ChatGPT produced less complex code, indicating easier recursion, while DeepSeek produced more complex programs due to higher Halstead and cyclomatic complexity scores. Gemini programs have a medium level of difficulty. The Kruskal-Wallis test was used to further analyze the data, and it revealed significant differences between the recursive code generated by ChatGPT, DeepSeek, and Gemini. Overall, the study found that each LLM has a distinct pattern: ChatGPT emphasizes simplicity, Gemini takes a balanced approach, and DeepSeek's generated code promotes clarity but suffers from complexity. More comprehensive analysis will be conducted in the future by expanding the dataset and including larger language models.

***Correspondence author email address:** kzkhalida786@gmail.com
DOI: 10.21015/vtse.v13i4.2269

This work is licensed under a Creative Commons Attribution 3.0 License.

1 Introduction

Programming is the basic and most important skill of computer science [1] and the cornerstone of the digital world [2]. It is considered a skill that is essential for the development of reasoning and problem-solving skills [3].

The importance of programming becomes evident because software, which is developed through a process called programming, is essential to virtually every field in computer science. Despite its importance, programming is often considered a complex profession that requires a deep understanding of data structures, algorithms, and programming languages [4].

Programming is difficult because it requires knowledge of syntax, semantics, and logical structures that support algorithmic thinking. Writing functional code requires the ability to abstract, debug, and understand control flow, all of which are often difficult for beginners.

A number of pedagogical and computational strategies have been developed to solve these problems by simplifying programming and reducing cognitive stress. They consist of collaborative coding methods [5, 6], programming precursors [7], and visualization tools [8] that help illustrate programming ideas.

Recursion is an essential concept in programming that is important to both programming theory and code development [9]. Recursion a programming technique in which a problem is decomposed into smaller or self-similar subproblems [10]. Recursive programming was first introduced with ALGOL 60 [11]. The concept of recursion is important, although it is often misunderstood [12]. Gal-Ezer and Harel [13] describe recursion as one of the most universally difficult concepts to teach. Teaching and learning recursion have been the focus of hundreds of published articles[14].

The advancement of machine learning, natural language processing, and artificial intelligence has led to the emergence of a new generation of Large Language Models (LLMs) trained on vast datasets [15].

These models have demonstrated strong performance in code generation tasks [16, 17] and hold great potential for automating various aspects of software development [18]. LLMs capable of writing code are expected to significantly transform the landscape of software engineering [19].

The latest and most popular large language models, such as ChatGPT [20], DeepSeek [21], and Gemini [22], are capable of generating programming code across a wide range of programming languages, and numerous studies have analyzed and compared their coding performance. Large language models can generate recursive code. However, no existing study has systematically analyzed and compared the complexity of recursive code generated by popular large language models.

The present study aims to fill this gap by systematically analyzing recursive programs generated by modern large language models. By using quantitative measurements to analyze the complexity of generated code, it also offers an objective basis for assessing how well LLMs generate recursive code, an important but difficult programming technique.

This study is important and novel because it bridges the gap between the developing field of AI-generated code evaluation and the well-established field of code complexity analysis. As a result, it advances knowledge about the quality of code produced by contemporary AI systems from both a methodological and practical perspective. The results of the study can help developers, educators, and academics understand the advantages and disadvantages of using LLMs to create sophisticated programming code.

The major contributions of the present study are stated below:

- It systematically analyzes and compares the recursive programming code generated by modern and widely used large language models.
- It evaluates the generated code using quantitative metrics to measure the complexity of recursive code.
- It establishes a dataset of recursive programs and makes it publicly available so it can be used in other studies.
- It provides empirical evidence on how different large language models conceptualize recursion, which can aid developers, researchers, and peda-

gogists in selecting models for recursion-related tasks.
- It offers a useful and meaningful dimension for the analysis of programming code generated by large language models.

This article has the following structure: the literature review is presented in Section 2, the design and methodology are described in Section 3, the results are included in Section 4, Section 5 contains the discussion of the results, and finally, the last section provides the conclusion.

2 Related Work

Since the development and public release of large language models, there have been several studies that have analyzed AI-generated code from different perspectives.

Shahzad and Iqbal [23], conducted a study to compare Python programming code generated by ChatGPT, DeepSeek, and Gemini to assess their performance in terms of accuracy, code quality, and computational efficiency. The study used a dataset of programming tasks ranging in complexity from 800 to 2000. The study used a thorough evaluation procedure that included runtime profiling, static code analysis, and online judge validation. Even when DeepSeek reasoning time increased with increasing problem complexity, the results indicated that it consistently produced correct solutions in fewer trials, achieving significantly higher accuracy. Gemini, on the other hand, completed tasks faster, but his accuracy decreased with more difficult tasks. Although ChatGPT occasionally exhibited lower code quality, it demonstrated balanced performance with moderate efficiency and correctness. The study emphasized general problem types and did not analyze the complexity of specific code such as recursion-specific code, which is essential and widely used in commercial programming.

Wang et al. [24] conducted a study to analyze the differences between LLM-generated code and human-developed code, focusing on the accuracy of code generation and variations in coding style. For the study, CodeLlama-7B, StarCoder2-7B, DeepSeekCoder-1.3B, DeepSeekCoder-6.7B, and GPT-4 were selected, and the experiment was conducted on CoderEval. To compare the LLM-generated code with human-generated code, the study manually analyzed the coding styles and found that the styles of large language models differ from those of human developers. Although the study made a notable contribution, it manually analyzed the coding style and did not use any quantitative scale, leaving a gap that the present study addresses.

Maharani et al. [25] conducted a study to analyze and compare Python code generated by ChatGPT and Gemini. For the study, a sample of 157 coding tasks was used, and the generated code was evaluated for correctness and complexity using the Python Concept Extraction and Representation Framework. The results showed that ChatGPT generated slightly more complex code and achieved a higher success rate (98.7%) than Gemini (94.9%). While ChatGPT provided tuple operations and the "range" function, Gemini focused more on loop structures and arithmetic assignments. Although the study made a notable contribution, it compared only two models using a very small dataset.

Coignion et al. investigated the efficacy of LLM-generated code using a dataset from LeetCode and contrasted their output with human-generated solutions. The study evaluated 18 LLMs, including model temperature and success rates, as well as their impact on code performance. The findings showed that the performance of the generated code is mostly consistent across various models, regardless of their size or training data. Moreover, increasing the temperature parameter during code generation results in higher performance variance, but this does not imply that solutions are better or worse on average. The study also found that code produced by large language models is, on average, more efficient than that created by humans. The findings and contribution of the study are extremely useful. However, the study did not focus on the complexity of LLM-generated code, which is important and essential in real-world situations.

Du et al. [27] conducted a study to examine large language models in class-level code generation. In this

study, a benchmark called ClassEval was manually created for class-level code generation tasks. It consists of 100 class-level Python code generation tasks and required approximately 500 person-hours to develop. Using these benchmarks, 11 LLMs were evaluated for class-level code generation. The results revealed that all large language models exhibit significantly lower performance on class-level code generation than on method-level code generation. Although GPT models continue to outperform other LLMs in class-level code generation, the performance rankings of other models in method-level code generation do not apply to class-level tasks. Moreover, most models (except for GPT models) generate class methods more effectively when approached method by method. Additionally, their capability to produce dependent code is limited. The study, overall, provides useful information. However, the dataset is solely based on class-level code, which naturally restricts the generalization of the results.

The study conducted by Almanasra and Suwais [28], examined ChatGPT's ability to produce high-quality, functional code in several programming languages. By analyzing code generated for 300 data structure problems and 300 LeetCode problems, the performance of Python and Java was compared in terms of error management, code quality and efficiency. The study found that Python had better memory efficiency at all complexity levels, while Java had better runtime speed, especially for medium and demanding task. There were also significant differences in code quality, with both languages showing flaws in documentation and exception handling. The study concluded that while ChatGPT-4o can effectively handle smaller tasks, its ability to produce optimal code decreases with task complexity, especially regarding runtime in Python-generated solutions. The findings of the study highlight useful insights on LLM-generated code, but the study used only one large language model.

A study by Tosi [29] examined the source code developed by GPT-3, GPT-4, and Bard. Three difficult code assignments from a Java programming test administered by Insubria University were selected for the study. Quality metrics and test suites were used to assess the correctness and quality of the resulting code. The study concluded that while all three models can attempt this task, they require constant professional supervision.

The performance and energy efficiency of the code produced by large language models were examined in a study by Solovyeva et al. [30]. Three LLMs—GitHub Copilot, GPT-4o, and o1-mini—were utilized in the study, and their Python, Java, and C++ code was analyzed for difficult challenges on two platforms. According to the analysis, the models produced Python and Java code far more successfully than C++ code. In a similar vein, LLM-generated code occasionally performs better than effective human-crafted solutions. Python produces the best outcomes, albeit this varies according on the programming language used. Furthermore, a significant correlation between the created code's performance on the two systems was discovered, indicating that the outcomes might be transferable between both. The study analyzed performance and efficiency well, but did not focus on complexity analysis.

The studies included in this section clearly identify that LLM-generated code has been analyzed from different perspectives, but none has analyzed recursive code. Similarly, no study has examined or compared the complexity of LLM-generated code, leaving a gap that is addressed in the present study.

3 Design & Methods

3.1 Research Design

The central objective of the present study is to analyze the recursive programming code generated by large language models. The methodology used in the study is illustrated in Figure 1.

3.2 Data Collection

There are several large language models capable of generating programming code. For this study, three LLMs, namely ChatGPT (GPT-5), DeepSeek (V3.1), and Gemini (2.5 Pro), were selected because they are popular large language models widely used for code generation [31].

The study began with the preparation of a programming dataset, for which 287 recursive problems were

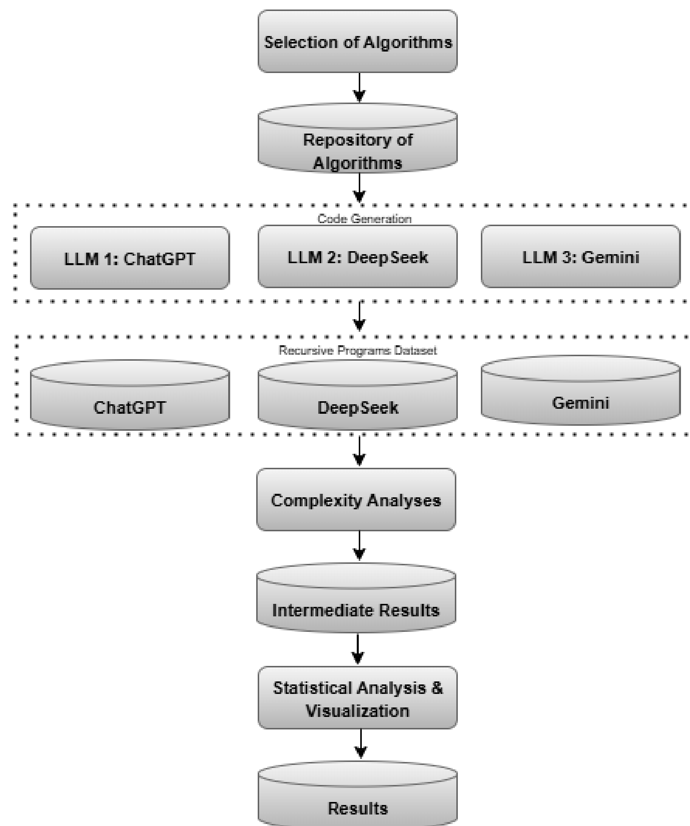

Figure 1. Research Methodology

initially identified, out of which 250 were selected in consultation with two programming experts. From 28 August 2025 to 3 September 2025, equivalent code for the selected algorithms was generated by each LLM, and the obtained code was organized as a dataset of recursive programs.

The criteria used during the selection of recursive problems are as follows:

- The problem must either allow recursion as a logical technique or require a recursive solution.
- The problem should demonstrate a variety of recursion techniques, including nested, structural, divide-and-conquer, and linear recursion.
- Excessive domain-specific requirements were removed to maintain universality.
- Problems solvable within classical programming levels were included.

3.3 Prompting Method

The same prompt template was utilized for every problem to ensure consistency across the three large language models: ChatGPT (GPT-5), DeepSeek (V3.1), and Gemini (2.5 Pro). The problems were presented to each model under zero-shot conditions and in the same order.

The code generation process used the following prompt template:

Prompt Template:

"Write a complete recursive Python code to solve the following problem: [problem statement]. The solution must use recursion."

No further cues, examples, or follow-up instructions were provided to ensure impartiality. No regeneration or refinement was performed, and each model was allowed to produce only one solution per problem.

3.4 Data Availability

The recursive programs generated by ChatGPT, DeepSeek, and Gemini were organized into a dataset for this study and have been made publicly available for reproducibility:

Data link: https://doi.org/10.5281/zenodo.17738426

3.5 Evaluation Metrics & Tools

The code generated from ChatGPT, DeepSeek, and Gemini was further quantitatively analyzed using Halstead complexity metrics and Cyclomatic complexity metrics.

Halstead complexity metrics (HCM) are a suite of metrics used evaluate the complexity of a program code [32]. It provides line-level coverage and has been widely used in a variety of fields, including similarity approximation, fault prediction, and quality evaluation [33]. HCM is based on the following four key variables [34], which represent the fundamental aspects of operators and operands [35, 36]. Halstead complexity is widely used for comparing programming code written in different programming languages [37].

Similarly, Halstead complexity is also used in many other areas, including software fault prediction [38]. The details of HCM are stated below.

N_1 = Total number of operators
N_2 = Total number of operands
η_1 = Number of distinct operators
η_2 = Number of distinct operands

A variety of metrics are available to evaluate code complexity using the Halstead Complexity Metrics. The recursive code in this study was analyzed using Halstead's Volume, Difficulty, and Effort measures.

A program's volume indicates the extent of its algorithmic implementation and represents the information it contains. The program's volume is calculated using the formula below.

$$\text{Volume} = (N_1 + N_2) \times \log_2(\eta_1 + \eta_2)$$

A program's difficulty level is directly related to the number of distinct operators it contains, which reflects how difficult it is to write or comprehend the program. The following formula is used to calculate the difficulty.

$$\text{Difficulty} = \left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right)$$

The mental effort required to translate an algorithm into its implementation in a specific programming language is measured by the effort. The following formula is used to calculate the effort.

$$\text{Effort} = \left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right) \times \left((N_1 + N_2) \times \log_2(\eta_1 + \eta_2)\right)$$

The second metric used in the study to analyze the recursive code of Python generated by ChatGPT, DeepSeek, and Gemini was the Cyclomatic Complexity metric. Cyclomatic Complexity is a common measure of software complexity that assesses the number of linearly independent paths [39, 40].

The Halstead and Cyclomatic complexity of all generated programs were calculated using Python (ver. 3.9). The popular package *radon* was used for complexity analysis, and *pandas* was used for storing the results. The results were further statistically analyzed using SPSS (ver. 25), and the visualization of the results was carried out in Python.

3.6 Code Verification

The correctness of the dataset (programming code) is essential for ensuring the quality and reliability of the analysis results. Therefore, a two-phase verification process was conducted before the actual analysis to ensure the accuracy of the generated recursive programs. In the first phase, each program was manually reviewed by three human experts.

In the second phase, each program was tested through execution in Python 3.9. Only those recursive codes that were verified by human experts, executed successfully, and produced the expected output were included in the computational analysis. Necessary preprocessing, such as fixing indentation errors, was applied only when essential to ensure code executability without altering the logic.

The complete algorithm used for recursive code generation, verification, and complexity analysis is shown in Algorithm 1.

4 Results

The study presented in this article analyzed the recursive programming code generated by ChatGPT, DeepSeek and Gemini. During analysis, initially the operators, distinct operators, operands and distinct operands were counted from the programs generated

Algorithm 1. Algorithm for Recursive Code Analysis

Require: Set of all recursive problems P_{all}
Ensure: Complexity metrics for ChatGPT, DeepSeek, and Gemini
1: Select $P \subseteq P_{\text{all}}$ such that $|P| = 250$
2: MODELS $\leftarrow \{M_1$ = ChatGPT, M_2 = DeepSeek, M_3 = Gemini$\}$
3: **for** each problem $p \in P$ **do**
4: **for** each model $m \in$ MODELS **do**
5: PROMPT $\leftarrow$ "Write a complete recursive Python code for: p. The solution must use recursion."
6: $C[m, p] \leftarrow m$.generate(PROMPT)
7: **end for**
8: **end for**
9: **for** each model $m \in$ MODELS **do**
10: **for** each code $C[m, p]$ **do**
11: **if** HumanReview($C[m, p]$) = $\times$ **then**
12: Remove $C[m, p]$
13: **continue**
14: **end if**
15: **if** Run($C[m, p]$) = error **or** Output($C[m, p]$) $\neq$ Expected(p) **then**
16: Remove $C[m, p]$
17: **continue**
18: **end if**
19: **end for**
20: **end for**
21: **for** each verified code $C[m, p]$ **do**
22: $N_1 \leftarrow$ number of operators
23: $N_2 \leftarrow$ number of operands
24: $\eta_1 \leftarrow$ distinct operators
25: $\eta_2 \leftarrow$ distinct operands
26: $V \leftarrow (N_1 + N_2)\log_2(\eta_1 + \eta_2)$
27: $D \leftarrow (\eta_1/2) \cdot (N_2/\eta_2)$
28: $E \leftarrow D \cdot V$
29: Cyclomatic Complexity:
30: $CC \leftarrow e - n + 2$
31: Store $\{V, D, E, CC\}$
32: **end for**
33: **for** each metric $X \in \{V, D, E, CC\}$ **do**
34: Perform normality tests: KS(X), SW(X)
35: **if** X is non-normal **then**
36: Apply Kruskal–Wallis test on X
37: **end if**
38: **end for**
39: **return** Comparative results for all three models

by selected models and the summarized results are shown in Table 1.

The recursive programs produced by ChatGPT, DeepSeek, and Gemini showed discernible differences according to the descriptive analysis (Table 1) of the elementary Halstead parameters. All four parameters—total operators, distinct operators, total operands, and distinct operands—exhibited higher mean values and greater variability in DeepSeek's generated programs than in those of ChatGPT and Gemini. These trends suggest that DeepSeek generated more diverse and complex code structures, demonstrating a broader utilization of operators and operands.

For further illustration of elementary results, the group bar charts are created and are shown in Figure 2.

The group bar charts (Figure 2) provide a clear view of the elementary results and illustrate the patterns and differences within and between the models. For a clearer and more comparative view of the elementary results of the programs, the radar chart is created and shown in Figure 3.

To statistically analyze the results of the elementary parameters obtained from the program analysis, normality tests were conducted, and the results are presented in Table 2.

The Shapiro-Wilk and Kolmogorov-Smirnov normality tests were performed on the elementary Halstead parameters, and the results indicated non-normality. To determine whether the recursive programs created by ChatGPT, DeepSeek, and Gemini differed significantly from one another, these parameters were then analyzed using the non-parametric Kruskal-Wallis test. The results showed that DeepSeek's mean ranks were consistently higher for total operators, distinct operators, total operands, and distinct operands, suggesting that the code it generated was more sophisticated and contained more operators and operands. For all four elementary parameters, the Kruskal–Wallis test confirmed that these differences were statistically significant ($p < 0.05$).

After the initial and elementary analysis of the programs, a more detailed analysis was conducted on all three datasets of Python programs using the Halstead complexity metrics and Cyclomatic complexity metrics. The summarized results are shown in Table 3.

Table 3 provides a complexity analysis of the recursive Python programs generated by ChatGPT, DeepSeek, and Gemini. Halstead and Cyclomatic complexity metrics were used to compare logical and structural complexity. The mean value for the Halstead volume of DeepSeek's programs (261.27) was higher than Gemini's (172.30) and ChatGPT's (136.68), suggesting that programs generated bt DeepSeek were longer and contained more operands and operations. The high variance and standard deviation (88202.12 and 296.99, respectively) suggest

Table 1. Results of Elementary Analysis

Metric	Model	Mean	Median	Variance	Std. Dev.	Min	Max	Range	IQR	Skewness	Kurtosis
Operators	ChatGPT	10.36	9	45.437	6.741	1	43	42	6	1.827	4.516
	DeepSeek	17.09	12	227.481	15.082	0	107	107	14	2.494	8.545
	Gemini	12.68	11	52.097	7.218	1	44	43	8	1.346	2.665
Distinct Operators	ChatGPT	5.42	5	4.782	2.187	1	14	13	3	0.598	0.698
	DeepSeek	6.58	6	8.814	2.969	0	16	16	4	0.680	0.430
	Gemini	6.10	6	5.869	2.423	1	14	13	4	0.467	-0.037
Operands	ChatGPT	19.83	17	167.806	12.954	1	81	80	12	1.853	4.865
	DeepSeek	32.47	24	820.041	28.636	0	208	208	28	2.514	9.046
	Gemini	23.99	22	194.430	13.944	1	88	87	16	1.359	2.751
Distinct Operands	ChatGPT	14.23	12	84.369	9.185	1	60	59	9	1.925	5.252
	DeepSeek	22.66	17	368.320	19.192	0	145	145	18	2.588	9.477
	Gemini	16.56	14	86.247	9.287	1	58	57	11	1.424	2.712

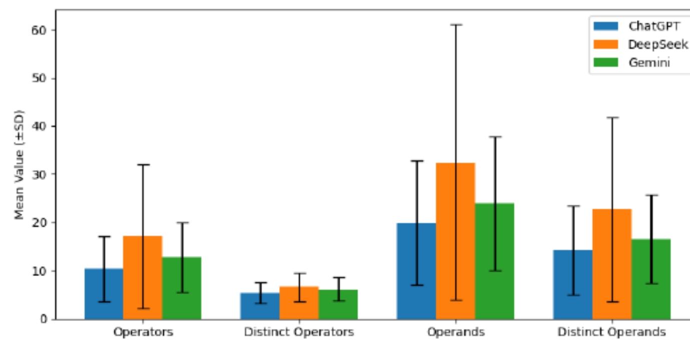

Figure 2. Grouped Bar Charts

that DeepSeek programs are more inconsistent. In contrast, the programs generated by ChatGPT had least mean and variance, which suggest than their programs are more concise and consistent. Programs produced by Gemini demonstrated a moderate degree of verbosity and conciseness.

Halstead Difficulty results showed that in terms of logical structure, DeepSeek's programs (mean = 4.72) were more sophisticated than ChatGPT's (mean = 3.85) and Gemini's (mean = 4.46) programs. This suggests that more mental effort is required to understand and change the recurrent programs of the DeepSeek.

In terms of Halstead Effort, DeepSeek had the highest average (1736.95), significantly higher than Gemini (938.09) and ChatGPT (655.46). This means that DeepSeek programs will be more difficult to understand and maintain. Furthermore, the large standard deviation (2913.24) reflects the difference in complexity in DeepSeek's programs. While ChatGPT's low effort score indicates effective and concise programs, Gemini's results are still reasonable.

Overall, the results showed a clear pattern: Gemini exhibits intermediate complexity, balancing elaboration and clarity, ChatGPT emphasizes simplicity and structural efficiency, while DeepSeek produces longer and more complex recursive code. DeepSeek's metrics also show higher skewness and kurtosis values, implying that its data distributions are more asymmetrical and contain more extreme outliers, supporting the conclusion that its code is more variable and unpredictable. For a clearer and more visible illustration of the complexity analysis results, the strip plots are created and shown in Figure 4.

To further analyze the results of the complexity analysis, the normality tests are performed, and the results are shown in Table 4.

The Kruskal-Wallis test was also applied to the higher-level Halstead and Cyclomatic metrics—Volume, Difficulty, Effort, and Cyclomatic Complexity—to examine differences in the structural

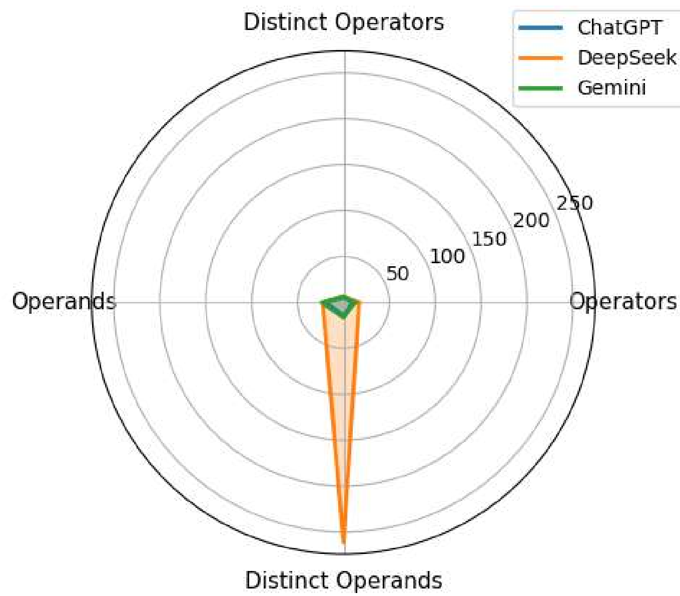

Figure 3. Radar Charts

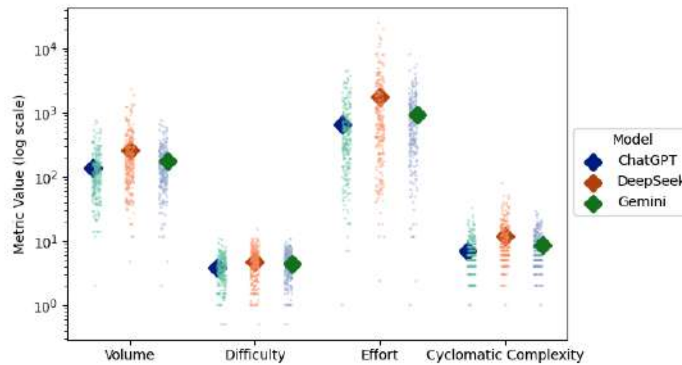

Figure 4. Strip Plot of Complexity Scores

and logical complexity of the recursive code generated by the three models, as the results of the normality tests on complexity scores indicated non-normality in most cases.

The mean rankings again showed that DeepSeek consistently produced the most intricate recursive programs. DeepSeek achieved the highest mean rank in Volume (428.01), followed by Gemini (388.11) and ChatGPT (310.38).

A similar pattern was observed for Cyclomatic Complexity (DeepSeek = 452.12; Gemini = 371.20; ChatGPT = 303.18) and Effort (DeepSeek = 422.59; Gemini = 389.90; ChatGPT = 314.01). For Difficulty, Gemini (392.65) slightly outperformed ChatGPT (326.61), while DeepSeek remained at the top (407.24), though with a smaller margin. The Kruskal–Wallis H-values for all four metrics were statistically significant (ranging from 19.66 to 59.57, $p < 0.05$), indicating that the variations in code complexity among the models were significant and consistent.

5 Discussion

The ability of three popular language models—ChatGPT, DeepSeek, and Gemini—to generate recursive programming code is evaluated in this study. From each model, suitable Python code was produced for the 250 algorithms chosen for analysis. The results showed that each LLM's recursive code generation methods

Table 2. Normality Tests on Elementary Parameters

Metric	Model	Kolmogorov–Smirnov			Shapiro–Wilk		
		Statistic	df	Sig.	Statistic	df	Sig.
Total Operators	ChatGPT	0.156	250	< .05	0.846	250	< .05
	DeepSeek	0.179	250	< .05	0.760	250	< .05
	Gemini	0.111	250	< .05	0.910	250	< .05
Distinct Operators	ChatGPT	0.127	250	< .05	0.960	250	< .05
	DeepSeek	0.115	250	< .05	0.959	250	< .05
	Gemini	0.128	250	< .05	0.966	250	< .05
Total Operands	ChatGPT	0.146	250	< .05	0.850	250	< .05
	DeepSeek	0.182	250	< .05	0.767	250	< .05
	Gemini	0.101	250	< .05	0.910	250	< .05
Distinct Operands	ChatGPT	0.151	250	< .05	0.843	250	< .05
	DeepSeek	0.180	250	< .05	0.755	250	< .05
	Gemini	0.127	250	< .05	0.898	250	< .05

Table 3. Results of Complexity Analyses

Metric	Model	Mean	Median	Variance	Std. Dev.	Min	Max	Range	IQR	Skewness	Kurtosis
Volume	ChatGPT	136.68	108	13465.727	116.042	2	749.5	747.5	98.49	2.278	6.973
	DeepSeek	261.27	161.69	88202.115	296.988	0	2306.41	2306.41	250	3.104	13.469
	Gemini	172.30	144.95	15642.615	125.070	2	748.76	746.76	134.34	1.694	3.775
Difficulty	ChatGPT	3.85	3.63	3.146	1.774	0.5	10.73	10.23	2.35	0.746	1.057
	DeepSeek	4.72	4.50	5.764	2.401	0	15.22	15.22	3.41	0.679	0.809
	Gemini	4.46	4.45	3.966	1.991	0.5	10.73	10.23	3.01	0.354	-0.277
Effort	ChatGPT	655.46	372.76	603751.402	777.014	1	4483.81	4482.81	616.65	2.343	6.268
	DeepSeek	1736.95	700.62	8486958.849	2913.239	0	24813.76	24813.76	1772.85	4.265	24.313
	Gemini	938.09	641.83	1012825.271	1006.392	1	8035.47	8034.47	991.66	2.742	12.206
Cyclomatic Complexity	ChatGPT	6.96	6	20.605	4.539	1	32	31	5	1.771	4.796
	DeepSeek	11.95	9	93.468	9.668	1	79	78	9	2.667	11.128
	Gemini	8.48	8	25.945	5.094	1	29	28	6	1.190	1.524

varied significantly. Different patterns in the recursive code generation behavior of ChatGPT, DeepSeek, and Gemini are shown by the observed discrepancies in Halstead and Cyclomatic complexity. DeepSeek's higher complexity metrics in terms of Halstead's volume, effort, and difficulty as well as Cyclomatic complexity indicate that it produces code with more complex control structures, nested conditions, and more operators and operands.

The pattern identified in the results implies that DeepSeek supports verbose and modular program structures. These characteristics likely stem from its training bias toward concise and unambiguous code representations, which can be optimized for functional completeness and robustness rather than brevity. While this can improve interpretability and error handling in certain situations, it also increases cognitive stress and makes code harder to maintain or debug.

On the other hand, programs generated by ChatGPT appear to be more concise, structured, and syntactically better iterative solutions based on their lower Halsted and cyclomatic complexity scores. In general, its recursion patterns are conceptually simple, indicating an inherent optimization for readability and simplicity—qualities that are beneficial to developers and learners looking for clarity. This is in line with ChatGPT's training focus on language accuracy and code optimization.

Gemini falls between ChatGPT and DeepSeek in terms of complexity, balancing and functionality. Its recursive code preserves the appropriate structural information without going deep. This synergy may come from its multimodal architecture, which integrates language and code comprehension to produce balanced, human-like coding patterns. Overall, these

Table 4. Normality Tests on Complexity Scores

Metric	Model	Kolmogorov–Smirnov			Shapiro–Wilk		
		Statistic	df	Sig.	Statistic	df	Sig.
Volume	ChatGPT	0.175	250	< .05	0.790	250	< .05
	DeepSeek	0.193	250	< .05	0.687	250	< .05
	Gemini	0.113	250	< .05	0.864	250	< .05
Difficulty	ChatGPT	0.063	250	0.016	0.967	250	< .05
	DeepSeek	0.053	250	0.089	0.969	250	< .05
	Gemini	0.065	250	0.013	0.984	250	< .05
Effort	ChatGPT	0.209	250	< .05	0.729	250	< .05
	DeepSeek	0.276	250	< .05	0.556	250	< .05
	Gemini	0.176	250	< .05	0.759	250	< .05
Cyclomatic Complexity	ChatGPT	0.176	250	< .05	0.855	250	< .05
	DeepSeek	0.174	250	< .05	0.760	250	< .05
	Gemini	0.137	250	< .05	0.906	250	< .05

differences show that the three LLMs represent distinct design philosophies: DeepSeek emphasizes clarity, ChatGPT emphasizes simplicity, and Gemini combines both approaches.

The valid results of the complexity analyses were confirmed with statistical analysis. Kruskal–Wallis test results ($p < 0.05$), which indicated statistically significant in complexity between ChatGPT, DeepSeek and Gemini for all measured parameters. When combined, the Halstead and cyclomatic complexity measures provide a solid basis for evaluating the code produced by LLMs. These results provide compelling evidence that different LLMs react differently to repetition and that code size, structure, and readability are clearly affected by training objectives and token generation strategies.

According to the programming dataset utilized and examined in this study, DeepSeek would be better suited for investigating algorithmic diversity or intricate recursive formulations, whereas ChatGPT might be more appropriate for producing recursive solutions that prioritize clarity. When moderate complexity and readability are both desired, Gemini offers a well-balanced choice.

The present study made a novel contribution in the analysis and comparison of LLM-generated code; however, there are several threats to the validity of the results: 1) only one programming language was used in the study, so the results of Python-based code generation cannot be generalized to other programming languages, 2) the identical prompt was used to generate every code, although the quality of the code produced may be impacted by variations in the input prompt, 3) only 250 programs per model were generated and used in the study, which may not fully represent the diversity of recursive problems or programming scenarios, 4) only one version of each model was used in the study, and differences in model versions can affect the quality of programming code, 5) only two metrics (Halstead and Cyclomatic complexity) were used in the study, which may overlook other important aspects of code quality, 6) the study does not evaluate the functional correctness or runtime behavior of the generated programs, 6) the selection of recursive problems may not cover all recursion types, reducing generalizability, 7) the long-term validity of findings may be impacted by future versions of LLMs acting differently as a result of periodic updates and enhancements, 8) only three large language models were used in the study; and 9) the study lacks human-intervened quality analysis of LLM-generated programming code.

In future, these limitations will be addressed, and a more comprehensive study shall be conducted to analyze the quality of recursive code generated by different large language models.

6 Conclusion

Programming is the core of computer science but is challenging, especially when dealing with concepts like

recursion. Recursion is a useful technique for solving problems and allows for elegant and compact solutions, but it is often difficult to understand and apply. Large language models are capable of generating recursive programming code. This study examined how the latest large language models, notably ChatGPT, DeepSeek, and Gemini, generate recursive Python code. In the study, the complexity of 250 recursive programs generated by these models was analyzed using Halstead and cyclomatic complexity metrics. The results identified that ChatGPT, DeepSeek, and Gemini are capable of generating Python code. The results also showed that ChatGPT generated less complex recursive code, while programs generated by DeepSeek were more complex due to higher Halstead and cyclomatic complexity values. The recursive programs created by Gemini were of medium complexity. The results highlight LLMs' ability to generate iterative code while identifying areas that need improvement to increase code quality. To better align model design, code optimization, and LLM-generated code with human programming standards, this study emphasizes the growing intersection between AI and software engineering and lays the groundwork for future work on incorporating human-based evaluation measures.

Author Contributions

Khalida Shaheen: Conceptualization, Methodology, Writing-Original draft preparation. **Muhammad Shumail Naveed**: Supervision, Validation, Investigation. **Anwar Ali Sanjrani**: Data curation. **Shafaque Saira Malik**: Visualization. **Samina Azeem**: Software.

Compliance with Ethical Standards

It is declare that all authors don't have any conflict of interest. It is also declare that this article does not contain any studies with human participants or animals performed by any of the authors. Furthermore, informed consent was obtained from all individual participants included in the study.

Funding Information

The study did not receive any funding from any institution.

References

[1] L. C. Cheng, W. Li, and J. C. Tseng, "Effects of an automated programming assessment system on the learning performances of experienced and novice learners," *Interactive Learning Environments*, vol. 31, no. 8, pp.5347-5363, 2023.

[2] R. Yilmaz, and F. G. K. Yilmaz, "The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation," *Computers and Education: Artificial Intelligence*, vol. 4, 2023.

[3] F. Kalelioglu, and Y. Gülbahar, "The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective," *Informatics in education*, vol. 13, no. 1, pp.33-50, 2014.

[4] S. Biswas, "Role of ChatGPT in computer programming," *Mesopotamian Journal of Computer Science,* vol. 2023, pp.9-15, 2023.

[5] A. Hawlitschek, S. Berndt, S., and S. Schulz, "Empirical research on pair programming in higher education: a literature review," *Computer science education,* vol. 33, no. 3, pp.400-428, 2023.

[6] J. Denner, E. Green, and S. Campe, "Learning to program in middle school: How pair programming helps and hinders intrepid exploration," *Journal of the Learning Sciences,* vol. 30, no. 4-5, pp.611-645, 2021.

[7] M. S. Naveed, and M. Sarim, "Two-phase CS0 for introductory programming: CS0 for CS1," *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences,* vol. 59, no. 1, pp.59-70, 2022.

[8] M. Mladenović, Ž. Žanko, M. A. Čuvić, "The impact of using program visualization techniques on learning basic programming concepts at the K–12 level," *Computer Applications in Engineering Education,* vol. 29, no. 1, pp.145-159, 2021.

[9] A. Baron, and D. Feitelson, Why is recursion hard to comprehend? An experiment with experienced programmers in python. Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, 115–121, 2024.

[10] S. Thorgeirsson, L. C. Lais, T. B. Weidmann and Z. Su, "Recursion in secondary computer science education: A comparative study of visual programming approaches,"

In Proceedings of the 55th ACM Technical Symposium on Computer Science Education, vol. 1, pp. 1321-1327, 2024.

[11] E. G. Daylight, "Dijkstra's rallying cry for generalization: The advent of the recursive procedure, late 1950s–early 1960s," *The Computer Journal*, Vol. 54, no. 11, pp.1756-1772, 2011.

[12] P. N. Johnson-Laird, M. Bucciarelli, R. Mackiewicz, and S. S. Khemlani, "Recursion in programs, thought, and language," *Psychonomic bulletin review*, vol. 29, pp.430-454, 2022.

[13] J. Gal-Ezer, and D. Harel, "What (else) should CS educators know?," *Communications of the ACM*, Vol. 41, no. 9, pp.77-84, 1998.

[14] R. McCauley, S. Grissom, S. Fitzgerald, and L. Murphy, "Teaching and learning recursive programming: a review of the research literature," *Computer Science Education*, Vol. 25, no. 1, pp.37-66, 2015.

[15] N. Raihan, D. Goswami, S. S. C. Puspo, M. L. Siddiq, C. Newman, T. Ranasinghe, J. Santos, and M. Zampieri, "On the performance of large language models on introductory programming assignments," *Journal of Intelligent Information Systems*, pp. 1-25, 2025.

[16] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, and J. Wang., "On the effectiveness of large language models in domain-specific code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 3, pp.1-22, 2025.

[17] K. Jin, C. Wang, H. V. Pham, and H. Hemmati, "Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation," *In 21st International Conference on Mining Software Repositories*, 2024.

[18] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty and S. K. Lahiri, "LLM-Based Test-Driven Interactive Code Generation: User Study and Empirical Evaluation," *IEEE Transactions on Software Engineering*, vol. 50, no. 9, pp. 2254-2268, 2024.

[19] S. L. France, "Navigating software development in the ChatGPT and GitHub Copilot era," *Business horizons*, vol. 67, no. 5, pp.649-661, 2024.

[20] H. Hassani, and E. S. Silva, "The role of ChatGPT in data science: how ai-assisted conversational interfaces are revolutionizing the field," *Big Data and Cognitive Computing*, vol. 7, no. 2, pp. 62, 2023.

[21] Z. Deng, W. Ma, Q. L. Han, W. Zhou, X. Zhu, S. Wen, and Y. Xiang, "Exploring DeepSeek: A Survey on Advances, Applications, Challenges and Future Directions," *IEEE/CAA Journal of Automatica Sinica*, vol. 12, no. 5, pp. 872-893, 2025.

[22] P. C. Nair, D. Gupta, and B. I. Devi, "Extracting Clinical Relationships from Discharge Summaries of Supra Sellar Lesion Patients using Gemini LLM," *Procedia Computer Science,* vol. 258, pp. 2391-2404, 2025.

[23] K. Shahzad, and S. Iqbal, "Comparative Analysis of ChatGPT, DeepSeek, and Gemini for Automated Code Generation," *In 2025 18th International Conference on Engineering of Modern Electric Systems, IEEE*, pp. 1-4, 2025.

[24] Y. Wang, T. Jiang, M. Liu, J. Chen, M. Mao, X. Liu, Y. Ma, and Z. Zheng, "Beyond functional correctness: Investigating coding style inconsistencies in large language models," *In proceedings of the ACM on Software Engineering*, vol. 2, pp. 690-712, 2025.

[25] A. L. Maharani, Y. S. Nugroho ,and S. Islam, "Unlocking AI Potential: An Investigation of Python Coding Capabilities of ChatGPT and Gemini," *In proceedings of the International Conference on Smart Computing, IoT and Machine Learning (SIML)*, pp. 1-6, 2025.

[26] T. Coignion, C. Quinton, and R. Rouvoy, "A performance study of LLM-generated code on leetcode, " *In Proceedings of the 28th international conference on evaluation and assessment in software engineering*, pp. 79-89, 2024.

[27] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Evaluating Large Language Models in Class-Level Code Generation," *In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1-13, 2024.

[28] S. Almanasra, and K. Suwais, "Analysis of ChatGPT-generated codes across multiple programming languages," *IEEE Access*, vol. 13, pp. 23580-23596, 2025.

[29] D. Tosi, "Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation," *Future Internet*, vol. 16, no. 6, pp. 188, 2024.

[30] L. Solovyeva, S. Weidmann and F. Castor, "AI-Powered, But Power-Hungry? Energy Efficiency of LLM-Generated

Code," *In proceedings of the IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering*, pp. 49-60, 2025.

[31] P. Lanzi, D. Loiacono, "Chatgpt and other large language models as evolutionary engines for online interactive collaborative game design," in Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1383–1390, 2023.

[32] M. S. Naveed, "Measuring the programming complexity of C and C++ using Halstead metric," *University of Sindh Journal of Information and Communication Technology*, vol. 5, no. 4, pp. 2521-5582, 2021.

[33] B. Khan, and A. Nadeem, "Evaluating the effectiveness of decomposed Halstead Metrics in software fault prediction," *PeerJ Computer Science*, vol. 9, pp. e1647, 2023.

[34] S. Azeem, M. S. Naveed, M. Sajid, and I. Ali, "AI vs. Human Programmers: Complexity and Performance in Code Generation," *VAWKUM Transactions on Computer Sciences*, vol. 13, no. 1, pp.201-216, 2025.

[35] M. S. Naveed, "Pedagogical suitability: A software metrics-based analysis of Java and Python," *International Journal of Innovations in Science & Technology*, vol. 6, no. 4, pp.1956-1967, 2024.

[36] G. Hao, H. Hijazi, J. Medeiros, J. Duraes, C. T. Lam, P. D Carvalho, and H. Madeira, "Complementarity in Software Code Complexity Metrics," *Journal of Systems and Software*, vol. 232, pp. 112679, 2025.

[37] M. S. Naveed, "Comparison of C++ and Java in implementing introductory programming algorithms," *QUEST Research Journal*, vol. 19, no. 1, pp.95-103, 2021.

[38] B. Khan, and A. Nadeem, "Evaluating the effectiveness of decomposed Halstead Metrics in software fault prediction," *PeerJ Computer Science*, vol. 9, pp. e1647, 2023.

[39] D. R. Wijendra, and K. P. Hewagamage, "Analysis of cognitive complexity with cyclomatic complexity metric of software," *International Journal of Computer Applications*, vol. 174, pp.14-19, 2021.

[40] K. Munawar, and M. S. Naveed, "The impact of language syntax on the complexity of programs: A case study of Java and Python," *International Journal of Innovations in Science & Technology*, vol. 4, no. 3, pp. 683-695, 2022.