On the Performance of Large Language Models on Introductory Programming Assignments

Nishat Raihan
mraihan2@gmu.edu

George Mason University
Dhiman Goswami
George Mason University
Sadiya Sayara Chowdhury Puspo
George Mason University
Mohammed Latif Siddiq
University of Notre Dame
Christian Newman
Rochester Institute of Technology
Tharindu Ranasinghe
Lancaster University
Joanna C.S. Santos
University of Notre Dame
Marcos Zampieri
George Mason University

Research Article

Keywords: Benchmark Dataset, Code LLM, Prompting

Posted Date: November 13th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-5348871/v1

Additional Declarations: No competing interests reported.

On the Performance of Large Language Models on Introductory Programming Assignments

Nishat Raihan[1*], Dhiman Goswami[1], Sadiya Sayara Chowdhury Puspo[1], Mohammed Latif Siddiq[2], Christian Newman[3], Tharindu Ranasinghe[4], Joanna C.S. Santos[2], Marcos Zampieri[1]

[1]George Mason University, Fairfax, VA, USA.
[2]University of Notre Dame, Notre Dame, IN, USA.
[3]Rochester Institute of Technology, Rochester, NY, USA.
[4]Lancaster University, Lancaster, UK.

*Corresponding author(s). E-mail(s): mraihan2@gmu.edu;

Abstract

Recent advances in artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) have led to the development of a new generation of Large Language Models (LLMs) trained on massive amounts of data. Commercial applications (e.g., ChatGPT) have made this available to the general public, enabling the use of LLMs to produce high-quality texts for academic and professional purposes. Educational institutions are increasingly aware of students' use of AI-generated content and are researching its impact and potential misuse. Computer Science (CS) and related fields are particularly affected, as LLMs can also generate programming code in various languages. To understand the potential impact of publicly available LLMs in CS education, we extend our previously introduced `CSEPrompts` [1], a framework comprising hundreds of programming exercise prompts and multiple-choice questions from introductory CS and programming courses. We provide experimental results on `CSEPrompts`, evaluating the performance of several LLMs in generating Python code and answering basic computer science and programming questions, offering insights into the implications of this technology for CS education.

Keywords: Benchmark Dataset, Code LLM, Prompting

1 Introduction

The past decade has witnessed a remarkable evolution in natural language processing (NLP) models. We have progressed from n-gram and word embedding models, such as word2vec [2] and GloVe [3], to sophisticated context-aware models like ELMo [4], and BERT [5]. These advancements have significantly enhanced performance across various NLP tasks [6]. More recently, Large Language Models (LLMs) such as GPT [7, 8] have further revolutionized the field. The impact of the latest generation of LLMs has been explored in a variety of domains. Nori et al. [9] explore their use in healthcare, while Tack and Piech [10] investigate their impact on education, heralding a new era in generative AI.

The impact of GPT models on education has also been the subject of several recent studies, which include studies conducted by Lo [7], Sok and Heng [11], Halaweh [12] among others. While these models offer numerous opportunities in educational technology, such as enhanced writing assistants, intelligent tutoring systems, and automatic assessment tools, they also raise concerns about potential misuse, particularly in coding tasks. Savelka et al. [13] find that while GPT scores may not meet course completion criteria, the model exhibits notable capabilities, including the ability to correct solutions based on auto-grader feedback. This capability raises concerns about students potentially exploiting this technology to generate complete essays and programming assignments, thereby artificially inflating their grades. Furthermore, Surameery and Shakor [14] demonstrate that ChatGPT excels in debugging, bug prediction, and explanation, although it has limitations in reasoning and integration.

Recent studies explore the use of LLMs for assessment in various domains. Katz et al. [15] examine its application in law, Zhang et al. [16] in mathematics and computer science, Haruna-Cooper and Rashid [17] in medicine and Raihan et al. [18] in Computer Science Education. These studies evidence the high quality of the models' output, with some suggesting that these models could even "pass the bar exam" Katz et al. [15]. Such findings underscore the potential impact of LLMs on educational assessment and the need for further investigation.

In this paper, we build upon our original `CSEPrompts` benchmark [1] to introduce `CSEPrompts 2.0`. This extension encompasses three key areas: an increased number of multiple-choice question (MCQ) prompts, evaluations of newer and more recent models, and a detailed error analysis. We present a comprehensive evaluation that goes beyond GPT, examining the performance of eight models capable of generating both English text and Python code on introductory CS and programming course assignments. To facilitate reproducibility and ensure further research in this area, we develop `CSEPrompts 2.0` as a robust framework. It comprises 219 programming prompts and 100 MCQs, carefully collected from coding websites and massive open online courses (MOOCs). This diverse set of prompts allows for a thorough assessment of LLM capabilities in the context of CS education. Our investigation addresses the following research questions:

- RQ1: How well do state-of-the-art LLMs perform on introductory CS assignments compared to existing Benchmarks?

- RQ2: Is there a significant difference in the performance of LLMs when completing assignments from coding websites compared to academic MOOCs?
- RQ3: Are state-of-the-art LLMs better at generating code or answering MCQs?
- RQ4: Are Code LLMs better at generating code and/or answering MCQs than raw LLMs?

We aim to provide some key insights into the capabilities and limitations of LLMs in CS education, informing both educators and researchers about the potential implications of these powerful tools in academic settings.

2 Related Work

2.1 Code Generation Models

Early automated code generation methods concentrated on inferring user intent from high-level specifications or input-output examples [19–21]. These methods convert task specifications into constraints, and a program is generated once it demonstrates compliance with those constraints [19]. With the advent of attention-based transformer models [22], code generation has evolved into a sequence-to-sequence task, where user intent is expressed through natural language. Most coding tasks involved code completion, code infilling, comment generation, and similar tasks that were often handled using encoder-only models like BERT [5]. Models such as CodeBERT [23], GraphCodeBERT [24], and SynCoBERT [25] are pre-trained on text-code pairs, including Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs) to capture syntactic and semantic code structures. However, encoder-only models are not primarily designed for generative tasks and exhibit subpar performance in code generation [25].

The emergence of generative models based on encoder-decoder architectures, such as CodeT5 [26], and decoder-only architectures, like CodeGen [27], CodeLLaMA [28], and StarCoder [29], has significantly improved code generation capabilities. Zan et al. [30] conduct a comprehensive survey, highlighting the superior performance of these models in code generation tasks. With these advancements, the need for unified benchmarks to evaluate and compare code generation models has become more pronounced.

2.2 Code Generation Benchmark

Several benchmarks have been introduced to assess the performance of code generation models. HumanEval, introduced alongside OpenAI's Codex model [31], and MBPP (Mostly Basic Python Problems) [32] are among the most widely used. These datasets contain coding prompts paired with human-generated solutions and three test cases for each task. Other benchmarks include CONCODE [33], DS-1000 [34], and extensions like HumanEval+ [35] and MBPP+ [36].

Recently, Large Language Models (LLMs) like GPT-3 [37], GPT-4 [8], and fine-tuned code models like CodeLLaMA [28] and StarCoder [29] have demonstrated remarkable code generation abilities. These models are evaluated on benchmarks like HumanEval and MBPP, consistently outperforming previous models. For instance, Roziere et al. Roziere et al. [28] show that CodeLLaMA achieves state-of-the-art results on HumanEval, demonstrating the effectiveness of decoder-only architectures for code generation. Siddiq et al. [38] analyzed these benchmark datasets and found several quality issues, such as insufficient contextual information.

In addition to code generation, other related tasks, such as code completion, which involves predicting the next token or sequence of tokens in code, have been extensively explored. Models like GPT-J [39], GPT-NeoX [40], and PaLM-Coder [41] are applied to code completion tasks using prompts longer than one sentence. Svyatkovskiy et al. [42] introduce IntelliCode Compose, a transformer-based model for real-time code completion, emphasizing the importance of handling multi-line code completions.

Despite these advancements, existing datasets and benchmarks primarily focus on general-purpose coding tasks relevant to software development but do not adequately address educational coding tasks. Educational coding tasks often require a deep understanding of specific programming language syntax and semantics, evaluating the learner's comprehension of fundamental concepts. These tasks differ significantly from the prompts included in existing benchmarks.

To bridge this gap, we introduce `CSEPrompts 2.0`, an extension of our previous work [1]. Our framework provides diverse programming exercise prompts and multiple-choice questions retrieved from introductory CS and programming courses. Each programming prompt is paired with five test cases, compared to three in most benchmarks, offering a more rigorous evaluation of code correctness.

While significant progress has been made in code generation and related tasks using LLMs, benchmarks focusing on educational coding tasks remain needed. `CSEPrompts 2.0` addresses this need by providing a comprehensive framework for evaluating LLMs on introductory CS assignments, thereby contributing to the understanding of LLMs' potential in educational contexts.

3 CSEPrompts 2.0

We introduce `CSEPrompts 2.0`[1], an enhanced evaluation framework consisting of coding prompts from coding websites and academic MOOCs (Table A1). `CSEPrompts 2.0` features a total of 319 exercise prompts, comprising 219 programming prompts and 100 multiple-choice questions, as shown in Table 1. The framework represents a significant evolution from its predecessor, incorporating a more diverse range of programming challenges and assessment formats. The programming prompts span various difficulty levels and conceptual areas, from basic syntax and control structures to more complex algorithmic problems. The multiple-choice questions are carefully curated to test theoretical understanding and practical knowledge of programming concepts. Each programming prompt is accompanied by comprehensive test cases, ensuring thorough validation of LLM-generated solutions.

Coding Websites

We select five leading online resources for introductory Python learning, detailed in Table 1. These platforms are chosen based on their interactivity, diversity of challenges, and structured learning approaches. They offer a variety of coding exercises,

[1] https://github.com/mraihan-gmu/CSEPrompts

Table 1: Summary of Coding Prompts from Various Sources

Coding Websites		MOOCs - Coding Prompts			MOOCs - MCQs		
Platform	Prompts	University	Course	Prompts	Uni./Org.	Course	Prompts
CodingBat	24	Harvard	CS50	29	GT	CS1301xI	20
LearnPython	16	UMich	PforE	7	GT	CS1301xII	20
Edabit	29	GT	CS1301xI	11	GT	CS1301xIII	16
Python Principles	26	GT	CS1301xII	20	GT	CS1301xIV	16
HackerRank	23	GT	CS1301xIII	17	Meta	Programming in Python	28
Total	**118**		**Total**	**101**		**Total**	**100**

interactive tutorials, and cater to a broad spectrum of skill levels. The strong communities and instant feedback mechanisms provided by these platforms are crucial for effective programming education.

MOOCs

Our study also incorporates programming assignments and multiple-choice questions from several MOOCs offered by prestigious institutions such as Harvard University, the University of Michigan, and the Georgia Institute of Technology, as detailed in Table 1. These courses, available on platforms like edX and Coursera, focus on introductory Python programming for beginners and those with some prior experience. We select courses emphasizing practical programming exercises while excluding tasks involving file I/O or command-line operations. Additionally, we gather 100 multiple-choice questions from various sources, each containing 5 to 10 options with one correct answer and multiple distractors.

Table 2: Statistics for Prompts

Metric	CodingSites	Academic	MCQ
Total Prompts	118	101	100
Max. No. of Tokens	101	372	221
Min. No. of Tokens	5	17	15
Mean No. of Tokens	28	158	106
Standard Deviation	16	72	51

Dataset Statistics

The prompts from coding sites are generally shorter than those from MOOCs, as shown in Table 2. For each prompt, we collect a minimum of 5 test cases, primarily from the source platforms. When necessary, we supplement with additional test cases generated using Pynguin [43], an open-source unit test generator for Python. To ensure the quality and relevance of Pynguin-generated tests, we manually review them, focusing on edge cases and comprehensive code coverage. For MCQs, we obtain correct answers from the original platforms. We collect LLM-generated responses for each prompt, manually clean them to isolate code snippets, and label them based on the number of passed test cases. We present a few prompts fro each subset of `CSEPrompts 2.0` in Appendix 2.

Data Collection Strategy

Unlike benchmarks such as HumanEval [44] or MBPP [32], which are created specifically to test how well LLMs generate code, our approach reflects actual classroom programming assignments. We carefully gather prompts from real academic courses and coding websites, checking for overlap and maintaining the original difficulty level of these tasks. This method ensures our dataset represents the types of problems students actually encounter in their programming education, rather than artificially constructed test cases. By using authentic educational content, we provide a more realistic assessment of how LLMs perform on the kind of programming challenges that form the foundation of computer science education.

4 Experiments

Large Langauge Models (LLMs)

We experiment with eight different LLMs that represent diverse architectural designs and implementations. Our selection is based on their exceptional performance across established leaderboards maintained by respected research communities: the EvalPlus LeaderBoard [45], AllenAI's WildBench[2] and HuggingFace's BigCode LLM[3]. In our evaluation of `CSEPrompts 2.0`, we include two proprietary models from OpenAI: GPT-4o [46] and GPT3.5 [44], both of which demonstrate state-of-the-art performance across various NLP tasks. For open-source base models, we test Llama-3 [47] and Mistral [48], each offering unique approaches to model architecture and optimization. We complete our evaluation with four code-specific models that have been fine-tuned for programming tasks: Code-Llama [28], StarCoder [29], MagiCoder [49], and Wizard-Coder [50]. This diverse selection of models allows us to comprehensively assess both general-purpose language models and specialized code generation systems, providing insights into their relative strengths in handling educational programming tasks.

'You are a helpful AI assistant. You are given the following problem: '

Write a function named capital_indexes. The function takes a single parameter, which is a string. Your function should return a list of all the indexes in the string that have capital letters.

'Please write a Python code snippet to solve the problem. Thanks.'

Fig. 1: Sample Prompt for Coding Tasks; includes 3 parts - a system prompt, the task description, and the instruction.

[2] https://huggingface.co/spaces/allenai/WildBench
[3] https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard

Code Generation

We prepare prompts and test each model on tasks from `CSEPrompts 2.0`. Figure 1 and 2 show the simple prompt format used for the models. The generated responses, including code, pseudo-code, and explanations, are manually cleaned to isolate the code. We then evaluate these codes using pytest [51], which enables efficient creation of readable Python unit tests.

'You are a helpful AI assistant. You are given a Multiple Choice Question. '

(False and True) or (False or True)

Is this statement resolved to True or False?

- **True**
- **False**
- **Statement will not compile**

'You need to pick one or multiple correct answers from the given ones. Thanks.'

Fig. 2: Sample Prompt for MCQs; includes 3 parts - a system prompt, the MCQ with multiple choices, and the instruction.

Evaluation Metric

In our work, we employ the `pass@1` metric, a variant of *pass@k* [31]. *Pass@k* metric evaluates the probability that *at least one* out of k generated samples are *functionally correct* (i.e., passed all *functional* test cases). To evaluate the `pass@k`, we generate n samples per prompt ($n \geq k$), count the number of samples c that are functionally correct ($c \leq n$), and calculate the unbiased estimator $\mathbb{E}$ by Kulal et al. [52]:

$$pass@k = \mathbb{E}_{prompts}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{1}$$

We use `pass@1` in this work, which measures how often a model passes all test cases on its first attempt.

5 Results

5.1 Coding Tasks

For our analysis, we employ the `pass@1` metric, a variant of `pass@k`, which measures how often a model passes all test cases on its first attempt. This metric provides a rigorous assessment of model performance in real-world programming scenarios where immediate correctness is crucial. Figure 3 illustrates the performance across different models and prompt sources. Proprietary models (GPT-4o and GPT3.5) demonstrate superior performance on both MOOC and CodingSite prompts, followed by code-finetuned models. Notably, all models consistently perform better on CodingSite

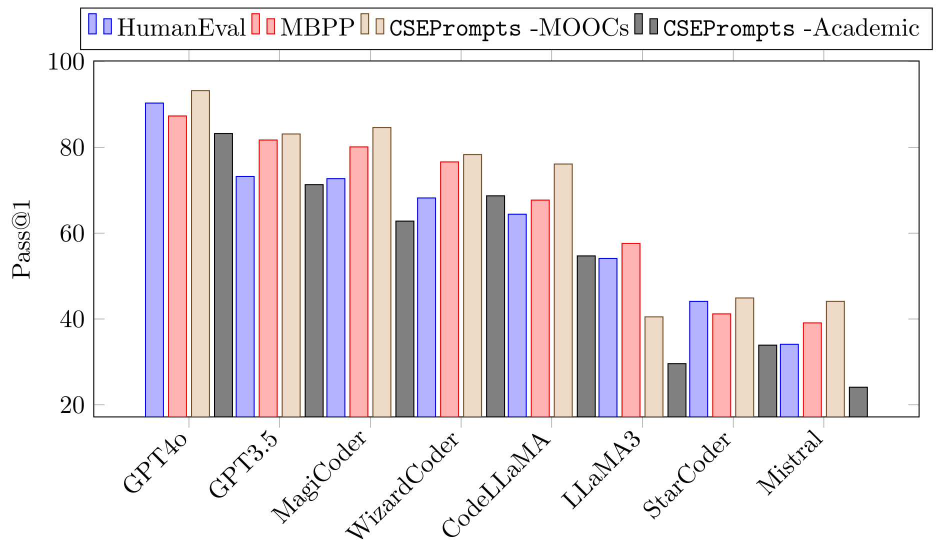

Fig. 3: Comparing `CSEPrompts` with HumanEval and MBPP based on **Pass@1**.

prompts compared to MOOC prompts, suggesting a difference in prompt complexity or structure between these sources. This performance gap may be attributed to the more structured nature of coding website problems compared to the potentially broader, more conceptual requirements of academic assignments.

To contextualize our results, we compare `CSEPrompts 2.0` with other widely used benchmarks. Our analysis reveals that the difficulty level of `CSEPrompts 2.0` falls between that of HumanEval and MBPP, providing a balanced challenge for evaluating code generation capabilities. This intermediate positioning makes our benchmark particularly suitable for assessing both the basic and advanced capabilities of code generation models in educational contexts.

5.2 MCQ Tasks

We compare our Multiple Choice Question (MCQ) task results with the MathQA-Python benchmark [32], which contains coding-related question-answer pairs. The MCQ subset of `CSEPrompts 2.0` introduces the first Code-MCQ benchmark in this domain, addressing a significant gap in the evaluation of LLMs' comprehension of programming concepts. Figure 4 illustrates LLM performance on both datasets. Our analysis reveals that models generally find MCQ tasks easier than open-ended QA tasks, likely due to the additional context and limited answer set guiding responses. Notably, while proprietary models excel in this task, code-finetuned models underperform, possibly due to their specialization in structured code generation rather than MCQ-style prompts. This performance disparity suggests that the ability to generate code does not necessarily translate to strong performance in understanding and answering questions about programming concepts, highlighting an important distinction in model capabilities.

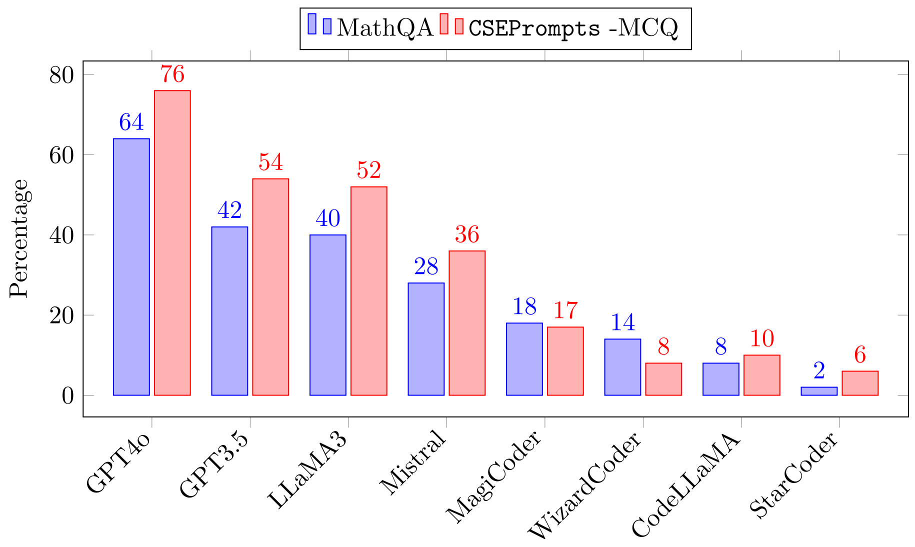

Fig. 4: Comparing `CSEPrompts` -MCQ with MathQA based on Zero Shot Prompting (in percentage).

6 Error Analysis

In this section, we present an error analysis of different code generation models in two distinct environments: coding sites and academic settings. Our analysis reveals four major categories of errors consistently encountered by these models: (1) syntax-related issues (*Indentation* and general *Syntax* errors), (2) naming and referencing problems (*Name*, *Attribute*, and *Key* errors), (3) data handling errors (*Value* and *Type* errors), and (4) runtime issues (*Recursion*, *Import*, and *SystemExit* errors). Tables 3 and 4 show the percentage distribution of these error types for each model in coding sites and academic tasks, respectively. This comprehensive categorization enables a detailed assessment of model performance across different programming contexts.

Table 3: Error Analysis (in percentage): Coding Sites

Error Type	**GPT4o**	**GPT 3.5**	**Magi Coder**	**Wizard Coder**	**Code LLaMA**	**Star Coder**	**LLaMA3**	**Mistral**
Name	2.00	2.50	4.00	4.50	5.00	4.50	–	–
Indentation	1.00	3.00	5.00	–	7.00	7.00	8.00	9.00
Value	1.50	4.00	–	–	8.00	8.00	9.00	10.00
Type	2.00	4.50	5.00	6.00	7.00	8.50	–	–
Syntax	–	–	0.50	0.50	8.50	9.50	10.00	11.00
UnboundLocal	–	2.00	2.50	4.00	4.50	5.50	7.00	8.00
Attribute	–	–	–	3.50	4.50	5.00	6.00	7.00
Recursion	–	1.00	1.50	3.50	–	6.00	7.00	8.00
ModuleNotFound	–	1.00	1.50	2.50	3.50	4.50	6.00	7.00
Index	–	1.00	2.00	3.50	4.50	5.50	–	8.00
Key	–	–	2.00	–	–	4.50	6.00	7.00
Import	–	–	2.50	2.00	–	4.00	6.00	7.00
Tab	–	0.50	–	–	2.50	–	6.00	–
System Exit	–	0.50	–	2.00	2.50	3.00	4.00	5.00
Infinite Loop	–	0.50	–	–	2.00	–	4.00	5.00

In the coding sites environment (Table 3), we observe a relatively even distribution of errors across categories, with *Name*, *Indentation*, and *Type* errors emerging as the most prevalent across models. Notably, Magi Coder shows a significant proportion of *Name* errors (4.00%), suggesting challenges with variable naming and scope management. Similarly, GPT 3.5 exhibits notable *Name* errors (2.50%). The more recent models - Star Coder, LLaMA3, and Mistral - demonstrate higher rates of *Syntax* errors, indicating fundamental challenges with code structure and formatting requirements.

Table 4: Error Analysis (in percentage): Academic

Error Type	GPT4o	GPT 3.5	Magi Coder	Wizard Coder	Code LLaMA	Star Coder	LLaMA3	Mistral
Name	1.00	5.25	7.25	3.00	–	3.50	2.00	4.00
Indentation	1.00	2.50	3.75	2.25	2.00	5.50	3.00	2.50
Value	0.50	3.00	2.00	1.00	1.00	3.50	2.50	1.00
Type	2.50	3.50	3.00	3.50	1.50	5.00	4.50	2.00
Syntax	–	–	–	–	–	5.50	6.00	5.75
Attribute	0.50	3.00	2.50	2.25	2.00	3.75	3.25	3.00
Recursion	0.25	1.50	1.75	2.50	2.50	2.00	2.75	2.25
ModuleNotFound	0.25	2.00	1.25	1.75	2.00	3.00	2.25	2.50
Index	0.50	1.50	2.50	1.50	1.00	2.25	3.00	2.75
Key	–	0.50	1.00	1.00	1.00	1.75	1.25	1.50
Import	–	0.75	0.50	1.25	1.50	1.50	2.00	1.75
ZeroDivision	1.00	1.00	1.00	1.00	1.00	–	–	–
FileNotFound	0.50	1.50	0.75	1.00	0.75	–	–	–
Exception	–	0.25	0.50	0.75	0.75	0.25	0.50	0.75

The academic environment (Table 4) presents a markedly different error distribution pattern. Models like Code LLaMA, Star Coder, LLaMA3, and Mistral show elevated rates of *Syntax*, *Value*, and *Type* errors. For instance, Mistral's high *Syntax* error rate (5.75%) and consistent *Value* error occurrences point to difficulties in handling academic code complexity. These patterns suggest that these models struggle with both advanced programming concepts and the more rigorous syntax requirements typical in academic assignments, particularly when dealing with complex data structures and algorithmic implementations.

Cross-environmental comparison reveals GPT4o's consistent superior performance, maintaining low error rates across all major categories in both settings. This consistency suggests a robust capability in generating correct code regardless of the context. The generally higher error rates in academic settings likely stem from several factors: the presence of more complex algorithms, requirements for specialized libraries, and stricter formatting standards. These elements may be underrepresented in the training data of models like Code LLaMA and Mistral.

Furthermore, academic tasks often demand a deeper grasp of theoretical concepts, potentially exceeding these models' current capabilities. These findings underscore the importance of careful model selection based on the intended application context, particularly in educational settings where code quality and conceptual understanding are equally important.

7 Conclusion and Future Work

In this work, we evaluated the performance of various Large Language Models (LLMs) on introductory computer science tasks, focusing on Multiple Choice Questions (MCQs) and Python programming assignments. We compiled `CSEPrompts 2.0`, a diverse evaluation framework comprising prompts from online coding platforms, academic resources, and programming courses. By analyzing eight state-of-the-art LLMs, we provided detailed performance metrics and error analyses to address four key research questions.

RQ1: How do LLMs perform on introductory CS assignments?

Our results show that all evaluated models can generate high-quality outputs on `CSEPrompts 2.0`, with GPT-based models demonstrating superior performance. GPT4o and GPT 3.5 consistently outperformed other models in both MCQs and coding tasks. The models performed better on `CSEPrompts 2.0` [MOOCs] compared to existing benchmarks but faced challenges with `CSEPrompts 2.0` [Academic], indicating a need for improvement in handling academic-style prompts. This performance pattern suggests that while LLMs have achieved significant capabilities in programming tasks, there remains room for enhancement in managing academic-specific requirements.

RQ2: Is there a performance difference between coding websites and academic MOOCs?

We observed a performance variance between prompts from coding websites and those from academic MOOCs. Most LLMs found prompts from coding websites more manageable, achieving higher accuracy and exhibiting fewer errors. In contrast, prompts from academic MOOCs posed greater challenges, leading to increased error rates. This suggests that academic MOOC content presents more complex or abstract concepts that require deeper understanding. The disparity highlights the unique challenges posed by educational content, where problems often integrate theoretical concepts with practical implementation requirements.

RQ3: How do LLMs perform in code generation tasks compared to MCQs?

Contrary to our initial assumption that text-focused LLMs would excel in MCQs, the evaluation showed that the tested LLMs generally performed better in code generation tasks. The models were able to produce syntactically correct and logically coherent code more consistently than selecting correct answers in MCQs. This outcome suggests that LLMs may benefit from the structured nature of programming languages, which provides clear syntax and semantics, whereas MCQs often require nuanced comprehension and reasoning. This finding indicates that despite their natural language capabilities, LLMs might be more adept at handling well-structured programming tasks than interpreting and responding to conceptual questions.

RQ4: How do Code LLMs compare to general-purpose LLMs in CS tasks?

While larger models like GPT4o and GPT 3.5 achieved the highest performance across all tasks, we observed that general-purpose LLMs typically performed better on MCQs, whereas Code LLMs showed stronger performance in coding tasks. This reflects the specialized training of Code LLMs on programming code repositories, enhancing their ability to generate code that adheres to programming standards. The performance distinction suggests that model specialization plays a crucial role in task-specific capabilities, though general-purpose models with sufficient scale can maintain competitive performance across diverse tasks. This insight has important implications for choosing appropriate models for different educational applications.

These findings highlight the importance of selecting appropriate models based on specific task requirements in educational contexts. Understanding each model's strengths and limitations is crucial for effective deployment in different coding environments.

Our future research directions encompass several key areas. First, we plan to expand `CSEPrompts 2.0` by incorporating a more diverse set of coding prompts and MCQs that span multiple programming languages and advanced computer science topics. Second, we aim to evaluate emerging LLMs and conduct in-depth analyses of code characteristics that are particularly relevant to specific course types, including code comprehensibility, security considerations, and algorithmic complexity. This targeted approach would enable us to assess LLMs' capabilities in generating course-specific code examples—for instance, emphasizing security features in cybersecurity courses or code readability in introductory programming classes.

Additionally, we intend to investigate the underlying factors that contribute to performance disparities between academic and non-academic prompts, as understanding these differences could yield valuable insights for both model development and educational applications. The application of our findings to computer science education and automated assessment systems offers promising opportunities to enhance pedagogical experiences and customize LLM deployment according to specific educational objectives.

In conclusion, this analysis establishes a comprehensive foundation for understanding LLM performance in introductory computer science contexts. By systematically identifying both the capabilities and limitations of current models, we contribute to the ongoing development of AI technologies in computer science education. These insights will guide the evolution of more effective and pedagogically sound integration of LLMs in educational assessment and instruction, ultimately supporting both educators and students in the learning process.

Funding None

References

[1] Raihan, N., Goswami, D., Puspo, S.S.C., Newman, C., Ranasinghe, T., Zampieri, M.: Cseprompts: A benchmark of introductory computer science prompts. In: International Symposium on Methodologies for Intelligent Systems (2024)

[2] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of NIPS (2013)

[3] Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of EMNLP (2014)

[4] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proceedings of ACL (2018)

[5] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL (2018)

[6] Rogers, A., Kovaleva, O., Rumshisky, A.: A primer in bertology: What we know about how bert works. Transactions of the Association for Computational Linguistics **8**, 842–866 (2020)

[7] Lo, C.K.: What is the impact of chatgpt on education? a rapid review of the literature. Education Sciences **13**(4), 410 (2023)

[8] Achiam, J., Adler, S., Agarwal, S., et al.: Gpt-4 technical report. arXiv:2303.08774 (2023)

[9] Nori, H., King, N., McKinney, S.M., Carignan, D., Horvitz, E.: Capabilities of gpt-4 on medical challenge problems. arXiv preprint arXiv:2303.13375 (2023)

[10] Tack, A., Piech, C.: The ai teacher test: Measuring the pedagogical ability of blender and gpt-3 in educational dialogues. In: Proceedings of EDM (2022)

[11] Sok, S., Heng, K.: Chatgpt for education and research: A review of benefits and risks. Available at SSRN 4378735 (2023)

[12] Halaweh, M.: Chatgpt in education: Strategies for responsible implementation. Contemporary Educational Technology **15**(2) (2023)

[13] Savelka, J., Agarwal, A., Bogart, C., Song, Y., Sakr, M.: Can generative pretrained transformers (gpt) pass assessments in higher education programming courses? arXiv preprint arXiv:2303.09325 (2023)

[14] Surameery, N.M.S., Shakor, M.Y.: Use chat gpt to solve programming bugs. International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290 **3**(01), 17–22 (2023)

[15] Katz, D.M., Bommarito, M.J., Gao, S., Arredondo, P.: Gpt-4 passes the bar exam. SSRN (2023)

[16] Zhang, S.J., Florin, S., Lee, et al.: Exploring the mit mathematics and eecs curriculum using large language models. arXiv preprint arXiv:2306.08997 (2023)

[17] Haruna-Cooper, L., Rashid, M.A.: Gpt-4: the future of artificial intelligence in medical school assessments. Journal of the Royal Society of Medicine, 01410768231181251 (2023)

[18] Raihan, N., Siddiq, M.L., Santos, J., Zampieri, M.: Large language models in computer science education: A systematic literature review. arXiv preprint arXiv:2410.16349 (2024)

[19] Gulwani, S., Polozov, O., Singh, R., *et al.*: Program synthesis. Foundations and Trends® in Programming Languages **4**(1-2), 1–119 (2017)

[20] Green, C.: Application of theorem proving to problem solving. In: Proc. of the 1st Intl. Joint Conf. on Artificial Intelligence. IJCAI'69, pp. 219–239. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1969)

[21] Manna, Z., Waldinger, R.J.: Toward automatic program synthesis. Commun. ACM **14**(3), 151–165 (1971) https://doi.org/10.1145/362566.362568

[22] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc., ??? (2017)

[23] Feng, Z., Guo, D., Tang, D., Duan, N., *et al.*: Codebert: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020 (2020)

[24] Guo, D., Ren, S., Lu, S., Feng, Z., al., T.: Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366 (2020)

[25] Wang, X., Wang, Y., Mi, F., Zhou, P., Wan, Y., Liu, X., Li, L., Wu, H., Liu, J., Jiang, X.: Syncobert: Syntax-guided multi-modal contrastive pre-training for code representation. arXiv preprint arXiv:2108.04556 (2021)

[26] Wang, Y., Wang, W., Joty, S., Hoi, S.C.: Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In:

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (2021)

[27] Nijkamp, E., Lee, J., Touvron, H., et al.: Codegen: An open large language model for code with multi-turn program synthesis. arXiv:2203.13474 (2022)

[28] Roziere, B., Gehring, J., Gloeckle, F., al., S.: Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)

[29] Li, R., Allal, L.B., Zi, Y., Muennighoff, N., Kocetkov, D., et al.: Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161 (2023)

[30] Zan, X.V., Deng, M., Yang, D., *et al.*: A survey of benchmarks for natural language to code generation. In: ACL (2022)

[31] Chen, M., Tworek, J., Jun, H., et al.: Evaluating large language models trained on code. arXiv:2107.03374 (2021)

[32] Austin, J., Odena, A., Nye, M., al., B.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021)

[33] Iyer, S., Konstas, I., Cheung, A., Zettlemoyer, L.: Mapping language to code in programmatic context. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (2018)

[34] Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R.: Ds-1000: A natural and reliable benchmark for data science code generation. In: International Conference on Machine Learning (2023). PMLR

[35] Liu, J., Xia, C.S., Wang, Y., Zhang, L.: Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. arXiv preprint arXiv:2305.01210 (2023)

[36] Guo, W., Yang, J., Yang, K., al., L.: Instruction fusion: Advancing prompt evolution through hybridization. arXiv preprint arXiv:2312.15692 (2023)

[37] Brown, T.B., Mann, B., Ryder, N., et al.: Language models are few-shot learners. Advances in neural information processing systems (2020)

[38] Siddiq, M.L., Dristi, S.B., Saha, J., Santos, J.C.S.: The fault in our stars: Quality assessment of code generation benchmarksn. In: 24th IEEE International Conference on Source Code Analysis and Manipulation (SCAM) (2024)

[39] Wang, B., Komatsuzaki, A.: GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model (2021)

[40] Black, S., Gao, L., Wang, P., et al.: Gpt-neox-20b: An open-source autoregressive language model. arXiv:2204.06745 (2022)

[41] Chowdhery, A., Narang, S., Devlin, J., et al.: Palm: Scaling language modeling with pathways. arXiv:2204.02311 (2022)

[42] Svyatkovskiy, A., Zhao, S.K., Fu, S., *et al.*: Fast and memory-efficient neural code completion. In: ICML (2021)

[43] Lukasczyk, S., Fraser, G.: Pynguin: Automated unit test generation for python. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, pp. 168–172 (2022)

[44] OpenAI: Gpt-4 technical report. ArXiv **abs/2303.08774** (2023)

[45] Liu, J., Xia, C.S., Wang, Y., Zhang, L.: Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems **36** (2024)

[46] OpenAI: Gpt-4 omni: A comprehensive multimodal model for language, vision, and beyond. arXiv preprint arXiv:2408.01234 (2024)

[47] Dubey, A., Jauhri, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)

[48] Jiang, A.Q., Sablayrolles, A., Mensch, A., et al.: Mistral 7b. arXiv preprint arXiv:2310.06825 (2023)

[49] Wei, Y., Wang, Z., Liu, J., Ding, Y., Zhang, L.: Magicoder: Source code is all you need. arXiv preprint arXiv:2312.02120 (2023)

[50] Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., Jiang, D.: Wizardcoder: Empowering code large language models with evol-instruct. arXiv preprint arXiv:2306.08568 (2023)

[51] krekel, team: pytest: helps you write better programs (2023). https://docs.pytest.org/en/7.4.x/

[52] Kulal, S., Pasupat, P., Chandra, K., Lee, M., Padon, O., Aiken, A., Liang, P.S.: Spoc: Search-based pseudocode to code. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., ??? (2019)

Appendix A Data Sources

Table A1: List of Coding Websites & MOOCs

Name	**Link**
CodingBat	https://codingbat.com/python
Learn Python	https://www.learnpython.org
Edabit	https://edabit.com/challenges/python3
Python Principles	https://pythonprinciples.com/challenges/
Hacker Rank	https://www.hackerrank.com/domains/python
Edx	https://www.edx.org
Coursera	https://www.coursera.org
CS50 (Harvard)	https://learning.edx.org/course/course-v1:HarvardX+CS50S+Scratch/home
PforE (UMich)	https://www.coursera.org/learn/python/home
CS1301xI (GT)	https://learning.edx.org/course/course-v1:GTx+CS1301xI+1T2023/home
CS1301xII (GT)	https://learning.edx.org/course/course-v1:GTx+CS1301xII+1T2023/home
CS1301xIII (GT)	https://learning.edx.org/course/course-v1:GTx+CS1301xIII+1T2023/home
CS1301xIV (GT)	https://learning.edx.org/course/course-v1:GTx+CS1301xIV+1T2023/home
Programming in Python (Meta)	https://www.coursera.org/learn/programming-in-python

2 Sample Prompts

```
Prompt1.
    You are given the coefficients of a polynomial P.
    Your task is to find the value of P at point x.
Prompt2.
    You are given a square matrix A with dimensions N x N.
    Your task is to find the determinant.
Prompt3.
    ...
```

(a) Sample Prompts from the Coding Sites.

```
Prompt1.
    implement a program that prompts the user for the answer to the Great
    Question of Life, the Universe and Everything, outputting Yes if the
    user inputs 42 or (case-insensitively) forty-two or forty two.
    Otherwise output No.
Prompt2.
    implement a program that prompts the user for a greeting. If the
    greeting starts with "hello", output $0. If the greeting starts with
    an "h" (but not "hello"), output $20. Otherwise, output $100. Ignore
    any leading whitespace in the user's greeting, and treat the user's
    greeting case-insensitively.
Prompt3.
    ...
```

(b) Sample Prompts from the MOOCs.

```
Prompt1.
    def func(x): return x * 2
    print(func(3))
    What is the output?
    6 3 9 None
Prompt2.
    print(float(5))
    What will be the output?
    5 5.0 None Error
Prompt3.
    ...
```

(c) Sample MCQ Prompts.

Fig. B1: Sample prompts from `CSEPrompts 2.0`