An Extended Investigation of Large Language Models Employing the Socratic Method

1st Westin Musser
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
ORCID 0009-0000-6467-4547

2nd Sudipto Ghosh*
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
ORCID 0000-0001-6000-9646
*Corresponding author

***Abstract*—Employing the Socratic method of teaching can be time consuming and cognitively demanding. We seek to understand how capable a number of popular large language models (LLMs) are as Socratic tutors in helping novice programmers with debugging their solutions to uncomplicated computing problems. This research is an extension of prior work that both contributes a dataset of multi-turn Socratic advice and benchmarks LLMs in their Socratic debugging capabilities. Our objectives are twofold; expand this dataset and evaluate an additional LLM with the same intrinsic metrics, and employ a reward model to gain further insight. We created a Java counterpart to the original Python-only dataset and assigned scores to the LLM generations with a reward model trained on good and bad Socratic utterances. The results from the extended evaluation with the NLP metrics adopted from the prior work are consistent with their findings. The scores given by our reward model agree with human preference.**

***Index Terms*—Large Language Models, Socratic questioning, reward models, debugging, benchmark**

I. Introduction

Teaching introductory computer science presents unique challenges. Students, especially first year undergraduates, taking their first steps into programming may not have the critical thinking skills necessary to succeed without additional support. Well suited for tackling this problem is the Socratic method of teaching, defined as a shared dialogue wherein the teacher formulates and asks probing questions meant to elicit certain thought processes in the student. By probing the student's understanding of their work and the task itself, the Socratic method promotes critical thinking. Providing such individualized help necessitates hiring many teaching assistants, whose availability through office hours is not a given, especially in large classes. Students seeking help and feedback may be unable to attend scheduled office hours. Therein lies an issue that large language models (LLMs) have the potential to solve.

The LLMs in education remains under a microscope. Individualized AI tutors have been shown to help students learn spoken languages [1]. Institutions have furthermore sought to simulate the one-to-one tutoring experience, using generative AI (Gen-AI) to achieve timely feedback at scale [2]. Certainly, there is much potential for personalized and collaborative learning offered by AI [3].

In a study motivated by the difficulty of meeting the demand for teaching assistants who are effective in providing tailored assistance, Al-Hossami et al. [4] contribute a dataset of multi-turn Socratic advice intended to aid a novice programmer in resolving bugs in their solutions to uncomplicated problems. This dataset consists of problem descriptions common to introductory computer science courses, test cases, buggy implementations in Python, and dialogue between the first year undergraduate student and a teaching assistant. In the dialogue, the teaching assistant employs the Socratic method to guide the student to understand their mistake. Using this method with Gen-AI, the student interacts with an LLM that generates the probing questions after considering the conversation so far. The authors then use this to benchmark how well GPT-3.5 and GPT-4 can provide Socratic debugging help. The prompt used to do so includes partial dialogue and the full problem and buggy code for context. Each LLM generates candidate Socratic utterances (questions designed to shift focus in a certain direction and provoke critical thinking) befitting the conversation so far, and they are scored using common Natural Language Processing (NLP) metrics. We find that one limitation of their paper is exclusively evaluating Python. While Python is the most commonly taught language in introductory courses, it is closely followed by or tied with Java [5]. Another limitation is that generated Socratic utterances are only evaluated on how close they are to good human utterances and not how they contrast with bad ones. We distinguish between the two types of utterances in Section III.

This paper is an extension of the research done by Al-Hossami et al. [4]. We involve another LLM not in the GPT family (Llama) in the experiments and introduce a reward model for new insight. While these LLMs' capabilities as Socratic tutors are evaluated using the same metrics as Al-Hossami et al. [4], we recognize that such metrics do not address the negative case. Bad Socratic utterances are well defined by the previous authors, but none are used in any empirical contrasting to the generated utterances. Therefore, we create a reward model that is trained with both good and bad Socratic utterances toward assigning scores which are less intrinsic than those of the NLP metrics. To support these extensions, we expand Al-Hossami et al.'s [4] dataset to also

include Socratic dialogues around Java bugs.[1]

This research aims to understand the efficacy that can be achieved by LLMs when acting as Socratic tutors. To this end, the following research questions are studied:

- **RQ**$_1$: Do the LLMs perform better or worse when benchmarked using Java bugs compared to Python bugs?
- **RQ**$_2$: How does Llama's ability to generate high-quality Socratic utterances compare to those in the GPT family when evaluated using the same metrics?
- **RQ**$_3$: What insights are gained by scoring with a reward model in the evaluation of LLMs as Socratic tutors?

Understanding **RQ**$_1$ informs how comparatively beneficial LLM tutors may be in courses taught in Java or Python. Answering **RQ**$_2$ is important because Llama is an open-source LLM that can save on subscription costs. The significance of answering **RQ**$_3$ comes from the reward model's ability to indicate that some LLM-generated Socratic utterances are closely aligned to bad examples, rather than only knowing how close or far they are from good ones.

Section III provides context for some of the machine learning topics referenced in this paper. Section IV describes the methods used to carry out the study. The data gathered is used to answer our RQs in Section V. We review related work in Section II, and offer closing thoughts and opportunities for future work in Section VI.

II. Literature Review

In this section, we discuss existing work that reinforces our inclusion of Llama in the evaluation, complements our research with findings on LLM use in computer science education, and achieves similar results.

Motivation for Evaluating Llama. In a literature review, Raihan et al. [6] find that most works employed ChatGPT, in part due to its popularity among students and the comparative cost of using GPT-4. Their finding bolsters our added evaluation of Llama, which can be hosted locally for completely free use. Koutcheme et al. [7] evaluate the efficiency of open-source LLMs in providing high-quality feedback for programming assignments. They find that Llama-3.1-70B performs better than or on par with GPT-3.5-turbo and competitively with GPT-4o-mini and GPT-4o in explaining feedback and creating perfect fixes. Such results support our motivation for including Llama.

LLMs for Fundamental Understandings. To understand how different guidance impacts interactions between learners and LLMs, Kumar et al. [8] work with 500 students, implementing four pedagogically-informed guidance strategies. They found the *solve then refine* strategy, where students attempt problems before using LLMs to refine their answers, reduced likelihood of students immediately using AI to generate their assignments. This complements our research, which assumes that a student has written a solution to a programming problem before seeking help. Yeh et al. [9] research the impact of an interactive LLM used for code generation by novice programmers and report the need for special care when the primary learning goal is fundamental understanding. Our work in this paper is underpinned by this conclusion because we evaluate LLMs in their ability to improve understanding.

Similar Results. Hassan et al. [10] interview undergraduates who solved ten *Explain in Plain English* problems and five Python-code-writing questions using a chat bot, built on GPT-4o. Students appreciated that the chat bot stepped through tasks and made them think computationally. Among their conclusions is that the chat bot was effective in breaking down programs and guiding students to think critically. Our paper reports similar results, that multiple LLMs are capable of breaking down programs to the benefit of students.

III. Background

Bad Socratic Utterances. Al-Hossami et al. [4] categorize bad Socratic utterances into four groups. *Irrelevant utterances* shift focus from the actual bug and may cause confusion. *Repeated utterances* have previously been asked and answered. *Overly direct utterances* reduce learning opportunities by disclosing the bug too early. Lastly, utterances may be premature and guide students to change their code before the issue is identified. Good utterances properly direct student attention, do not repeat previous ones, progressively disclose information about the bug at a rate commensurate with the student's comprehension of relevant paradigms, and exhibit patience by waiting until the student identifies the issue.

NLP metrics. Evaluation of the LLMs in this paper and the prior research is conducted using the following common natural language processing metrics. BLEU (Bilingual Evaluation Understudy) [11] compares generated translations to reference texts towards calculating n-gram precision, which may be modified depending on differences in length. It claims a high correlation with human judgments of quality. A score close to 1 indicates much correspondence between the LLM-generated text and that of a human. ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation - Longest common subsequence) [12] informs on the semantic similarity of texts by calculating the recall, precision, and F1 scores using the length of the longest common subsequences between reference and generated texts. This metric is included to capture structural similarity. BERTScore (Bidirectional Encoder Representations from Transformers) [13] matches words in reference and generated sentences using cosine similarity. It can handle paraphrases better than the other metrics. Higher scores in this metric indicate closer semantic similarities.

Reward models. A core component of reinforcement learning with human feedback, reward models are language models trained to align with human preferences [14]. Commonly, this is achieved with dyadic data consisting of chosen examples and their rejected counterparts.

LLM characteristics. The LLMs we evaluate differ in some important facets. Llama, a smaller model than those in the GPT family, is more efficient and accessible per its non-commercial license. GPT models, while very powerful,

[1]The dataset is available at https://github.com/wstnmssr/extended-socratic-debugging-benchmark.

may be difficult to use for developers and researchers because they are large and resource intensive. Furthermore, all GPT models are closed-source while Llama is open-source. This enables leveraging Llama as a foundation for specialized tasks. Critical to either model's capabilities is training data, and the GPT models' training data is unknown to anyone outside of OpenAI. Meanwhile, Llama's training data, albeit undocumented, is available to view and even modify.

Chain of Thought. To help LLMs reason with more accuracy, the prompting technique Chain of Thought (CoT) instructs models to decompose problems and solve each small step before giving an answer [15]. Al-Hossami et al. [4] use it to split utterance generation into two steps: reasoning about why the student wrote the buggy code and what stops them fixing it, and using the dialogue so far with the results from the first step to generate a list of Socratic utterances.

IV. Approach

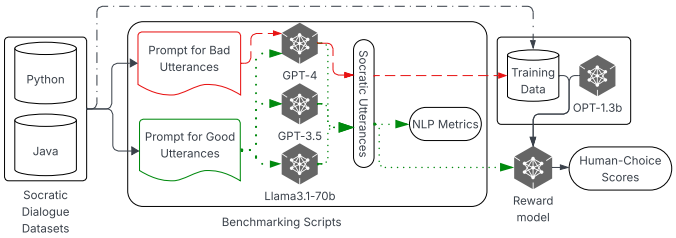

Fig. 1. Approach Overview

Figure 1 shows our overall approach for the full study, in which the datasets are used to construct prompts for good and bad Socratic utterances. The bad utterances are generated solely by GPT-4 to ensure consistency and quality that may be lost by using multiple models. They are paired with the original dataset to train a fork of OPT-1.3b, the result being our reward model. The good utterances, generated by all three models, are evaluated using both the NLP metrics and the reward model.

Llama. We add Llama3.1-70b, a member of the family of models by Meta [16], to the list of LLMs used in the previous study (GPT-3.5, GPT-4). We adopted the same approach as the previous authors to handle the subtle differences in interacting with different LLM API. We wrap the LLM API using a class, which is instantiated using the name of the LLM. To alleviate difficulties with using a cloud-based instance, we hosted Llama3.1-70b on our own high-performance cluster.

Java Dataset. We observed that the existing dataset of multi-turn Socratic advice for Python code has conversations using Python terms. We duplicated the dataset and translated this clone to Java. This approach allows us to limit our scope and focus on extending the evaluations. However, there are some paradigms and bugs that cannot be replicated in Java, or require much effort. Creating entire new Socratic dialogues because of large differences in languages is beyond the scope of this work. Thus, some items were omitted from the dataset. The following lists the differences between Python and Java that we found would necessitate creating completely new dialogues.

- **Absence of a `return` statement in non-void methods**. This can cause a compile-time error in Java, preventing the programmer from running test cases entirely.
- **Division using one or two backslashes**. While such a difference in Python yields a different numerical value, two backslashes denote a comment in Java.
- **Capitalization of boolean values**. For boolean values, Python uses `True` and `False` while Java uses `true` and `false`. Mistaking the capitalization in Java would again result in a compile-time error and prevent test cases from being run.

Differences in results using the Java and Python datasets may arise from the ways they can linguistically contrast. Detailed guidance can vary for bugs arising from misconceptions, unfamiliarity, or simple syntax errors in either language because of how they contrast structurally and syntactically.

An example scenario involves the task of writing code that returns the product of two numbers and solutions in Java and Python have the same bug: a missing *return* keyword. Tables I and II shows conversations for Python and Java respectively:

TABLE I
Conversation for Debugging in Python

Student	Hello, I'm having trouble with 'product()'. I'm failing even the first assertion and I can't figure out why.
Teacher	Okay, can you explain your 'product()' method in detail?
Student	I take in two inputs, which I have named 'a' and 'b', and then I multiply the two numbers together.
Teacher (GOOD)	Okay, can you modify your code such that on line 3, you execute the line 'print(Product(1,2))'?
Teacher (BAD)	Have you considered what happens when you multiply 'a' and 'b' but don't use the 'return' keyword?

In Table I, the teacher's first response is a good example because it allows the teacher to spot any gaps in the student's explanation. Their good second response gives the student a small hint toward understanding what the bug is, while the bad second response reveals the bug too early, at the cost of the student's learning.

TABLE II
Conversation for Debugging in Java

Student	Hello, I'm having trouble with 'product()'. I can't run the first assertion and I can't figure out why.
Teacher	Okay, do you see any errors when you try to run the assertions?
Student	Yes, it says "Syntax error on token '*', InvalidAssignment-Operator".
Teacher (GOOD)	Okay, can you explain your 'product()' method and why it should run?
Teacher (BAD)	Why do you think the multiplication operator is producing that error?

Since the same bug must be fixed differently in each language, the focus is immediately different. The conversation in Python would involve determining why the function returns 'None', while in Java it is a matter of proper syntax. In

Table II, the teacher's good second response stays focused on probing the student for their understanding while the bad second response diverts attention to the misleading error, which will likely confuse the student further.

Because of this, there may also be different trends in results for CoT. For example, there could a bug in a student's Python code that requires the teacher to direct attention toward proper indentation. In Java, where whitespace does not affect correctness, a similar bug may require the teacher to direct attention to where the student closed a scope with braces and whether or not a specific line was included in said scope. Because CoT decomposes problems into intermediate steps, which are likely to vary between these two languages, the results from evaluation are unlikely to match.

NLP Metrics. For consistency with Al-Hossami et al. [4], we use the same metrics BLEU4, BERTScore, and ROUGE-L for evaluation with the Java-enriched dataset. We append to the existing results for Python the scores earned by Llama.

Reward Model. To gain a more wholistic view of how well the models in consideration can employ the Socratic method, we fine-tune an existing pre-trained model (OPT-1.3b) to use as a reward model [17]. This model assigns scores to the generated Socratic utterances that can indicate how well they reflect human preference, thereby distinguishing itself from the NLP metrics. This approach has been shown to yield better evaluative performance and address issues around flexibility and understandability [18] [19]. We create a fork of OPT-1.3b [20] and configure it to properly tokenize the elements of the implicit prompt preference dataset used to train it.

The training data consists of over 21,000 pairs of *chosen* and *rejected* samples. Each sample has a prompt, which is described in [4], and its completion, a Socratic utterance befitting the conversation so far in the prompt. We use the human-generated utterances from the previous authors as the *chosen* half, regarding them as the *gold standard*. For the *rejected* half of the dataset, we direct GPT-4 to generate bad Socratic utterances by altering the steering prompt that tells LLMs to respond as a Socratic tutor by requesting bad Socratic utterances as defined by the previous authors.

V. Experimental Results

In this section we present our experimental results, answer our **RQs**, discuss our findings, and identify threats to validity.

*a) **RQ**$_1$: LLM performance in Java and Python:* In Tables III and IV, we display the results of evaluating all three LLMs with the Python and Java datasets, respectively. All scores are in percentages. The *+CoT* row contains scores achieved using the Chain of Thought (CoT) prompting technique. The models earn higher scores nearly comprehensively when evaluated with the Java dataset. Yet, these increases are by a small margin; F1 scores increase by an average of 6.6% with an even smaller increase of 1.3% for CoT.

While GPT-3.5 sees the largest increase, 18.9%, in average F1 scores when going from Python to Java, with CoT it is the only model to perform worse on average, doing so by -19.7%. GPT-4 sees the lowest increase overall with increases of 2.7% and 10.5% with CoT. Achieving the highest increase when evaluated with Java over Python and using Cot, Llama's average F1 scores without CoT increase by 4.7% and by 14.4% with it.

TABLE III
Evaluation of LLMs on the benchmark dataset in Python.

Language Model	BLEU-4			BERTScore_F1			ROUGE-L		
	P	R	F1	P	R	F1	P	R	F1
GPT-3.5	2.9	1.2	1.5	61.7	36.0	41.6	20.7	10.4	12.6
+CoT	2.3	0.8	1.1	62.2	35.6	41.9	20.5	10.1	12.5
GPT-4	1.5	5.8	2.2	14.6	65.6	22.3	6.4	27.5	9.7
+CoT	0.9	4.5	1.4	13.4	64.6	20.5	5.5	26.1	8.4
Llama	2.4	3.8	1.9	32.3	51.0	27.3	11.9	19.4	9.9
+CoT	1.8	3.6	1.7	29.2	53.9	27.5	10.6	20.2	10.1

TABLE IV
Evaluation of LLMs on benchmark dataset in Java.

Language Model	BLEU-4			BERTScore_F1			ROUGE-L		
	P	R	F1	P	R	F1	P	R	F1
GPT-3.5	3.8	2.0	2.3	61.5	36.5	41.6	22.3	11.2	13.2
+CoT	1.6	0.5	0.7	63.3	30.0	37.3	20.0	8.5	11.0
GPT-4	1.6	6.6	2.4	14.5	66.0	22.1	6.4	28.1	9.7
+CoT	0.9	4.2	1.5	16.1	65.5	23.8	6.5	26.5	9.6
Llama	2.8	4.7	2.1	34.7	49.6	27.8	12.7	19.5	10.1
+CoT	2.6	4.5	2.2	31.0	53.9	28.1	12.1	22.5	11.3

Tables V and VI display the descriptive statistics of scores earned by each LLM using the Python and Java datasets, respectively. Mean scores increase by just 2.1% from Python to Java. This percent change does not consider Llama+CoT because this score sees the only decrease, -11.8%, and it is influential enough on the average percent change to make it negative, despite the trend of growth in the other models. Standard deviation increases for GPT-3.5 by 17.6% and 24.5% with CoT. The other models see improved consistency with changes in standard deviation of -7.8% and -7.7% with CoT for GPT-4 and -3.8% and -6.9% for Llama.

We find that the LLMs in question do *not* perform much better or worse when benchmarked using Java bugs. They subtly increase in ability to act as Socratic tutors in debugging tasks that deviate across multiple programming languages.

TABLE V
Evaluation of LLMs by Reward Model in Python.

Language Model	mean	std	min	25%	50%	75%	max
GPT-3.5	5.72	0.68	3.55	5.35	5.73	6.21	7.57
+CoT	4.53	0.61	2.41	4.17	4.63	4.95	6.33
GPT-4	4.56	1.14	-6.15	4.07	4.69	5.22	7.84
+CoT	3.34	0.90	-2.36	2.82	3.37	3.85	6.80
Llama	4.86	1.06	-2.69	4.33	4.96	5.54	7.49
+CoT	4.37	1.01	-3.14	3.89	4.42	4.98	7.61

*b) **RQ**$_2$: Llama's Ability To Generate Utterances:* Throughout our results in Tables III and IV, Llama appears to perform similarly to GPT-4, surpassing it in some scores, namely Bleu-4 precision, BERTScore precision and F1, and Rouge-L precision and F1. In all of these cases, both models

TABLE VI
EVALUATION OF LLMS BY REWARD MODEL IN JAVA.

Language Model	mean	std	min	25%	50%	75%	max
GPT-3.5	5.73	0.80	2.01	5.18	5.83	6.33	7.77
+CoT	4.62	0.76	2.07	4.01	4.85	5.19	6.21
GPT-4	4.70	1.05	-3.81	4.17	4.81	5.36	7.59
+CoT	3.43	0.83	0.16	2.94	3.40	3.90	6.32
Llama	4.98	1.02	-2.90	4.45	5.04	5.62	7.54
+CoT	3.85	0.94	0.50	3.30	3.83	4.40	6.38

are outdone by GPT-3.5 and this pattern is regardless of the programming language or use of CoT.

Indeed, this trend also exists in the reward model data of Tables V and VI. Going beyond mean scores, Llama sees minimums and standard deviations situated between the other models. However, there are some disruptions to this observation. When using either dataset, Llama achieves the lowest maximum score without CoT and the highest with it.

Llama finds itself in the middle ground. The NLP metrics never indicate it to be a superlative, and while our reward model tells us that Llama may be able to outperform by using CoT in the best case scenario, it is outdone on average.

*c) **RQ$_3$**: Insights Gained After Using a Reward Model:* Figure 2 visualizes our reward model scores. We are able to glean that the use of CoT with Java reliably results in positive scores, with a greater spread with Python. Llama and GPT-4 maintain similarly-sized inter-quartile ranges with many outliers while GPT-3.5's scores are much more closely situated. This difference in spread illustrates GPT3.5's greater consistency with achieving high human preference. GPT-4 without CoT appears to earn the widest range of values. Its outliers go far into the negative, indicating that the model generated utterances that closely resembled bad examples more so than the other models.

We gather from Table V and Figure 2 that GPT-3.5 performs the most consistently with the highest mean scores and lowest standard deviations. It is the only model with all positive minimums. This suggests that the worst Socratic utterances generated by GPT-3.5 still aligns well with human preference. Informed by Table VI, we find that GPT-3.5 scores consistently higher, and posts the highest maximum. With a higher mean and lower standard deviation, Llama produces more favorable results than GPT-4 by thin margins.

Although the use of CoT does not yield higher reward model scores, standard deviation does decrease for each of the LLMs. These scores nearly suggest a trade-off between alignment to human preference and consistency. However, GPT-4 and Llama see their minimum scores turn positive, a vast improvement. With Java, these two models are signified to produce Socratic utterances that much better agree with human choice when CoT is employed.

A. Discussion

The scores earned by the GPT models vary from those reported in Al-Hossami et al. [4], offering new insights that potentially reflect changes made to the GPT models since the previous paper's data collection. Because details of these models such as training data and number of parameters are not available, it is difficult to attribute these variations in results to any particular change(s) made to them. Llama being open-source enables easier analysis if scores change much in any later iterations of this work.

Al-Hossami et al. [4] reported GPT-4 sacrificing precision for gains in recall when using CoT. We observe a new trend wherein the use of CoT somewhat reliably lowers all scores. The exceptions to this are Llama's BERTScore and Rouge-L results, where the open-source model saw improved performance. As this expresses a divergence from the human-generated utterances rather than greater similarity, we hypothesize that the additional detail and depth of explanation yielded by CoT prompting can cause the LLM-generated Socratic utterances to poorly cater to a student's level of understanding. We leave this for future investigation.

Moreover, GPT-4 still achieves the highest maximum score when considering the Python dataset. Simultaneously, GPT-4 consistently scores the lowest minimums. Al-Hossami et al. [4] remark on GPT-4 generating more Socratic utterances "focused on addressing various possible reasons or misconceptions typically while generating utterances that have already been asked or answered, or too early where the student is not yet aware of the issue." Highlighted by this result is the usefulness of the approach taken by the previous authors, and us, of prompting the LLMs for multiple candidate Socratic utterances and choosing the best among them. As such, we submit that further investigation is needed to ensure the LLM upon which a proposed Socratic debugging tutor is based always yields the best utterance in a robust manner.

Apropos Tables V and VI, save for Llama with CoT, the models consistently earn slightly higher mean scores when evaluated using the Java dataset. Although this is likely influenced by training data and model architecture, we conjecture that debugging in Java is perhaps more direct and succinct than in Python because of its strongly, statically typed nature and explicit syntax. Novices starting with Python may struggle with its dynamic types and whitespace-reliant syntax.

B. Threats to Validity

Internal Validity. The Java dataset is a translation of the Python version. If it were produced manually and by a team of expert teachers using genuine student code, it may differ and result in different evaluations. We mitigate this by excluding bugs and dialogues that do not translate well or at all.

External Validity. Our results are specific to the LLMs evaluated in this paper and cannot necessarily be used to generalize about other models. To understand how well emerging models such as Anthropic's Claude or DeepSeek-R1 would perform as Socratic tutors, analysis like ours is necessary. Similarly, our findings cannot inform LLM performance when benchmarked with a dataset of any other programming language.

VI. CONCLUSIONS AND FUTURE WORK

This paper extends previous work that introduces a dataset of expertly curated Socratic dialogues where teachers assist

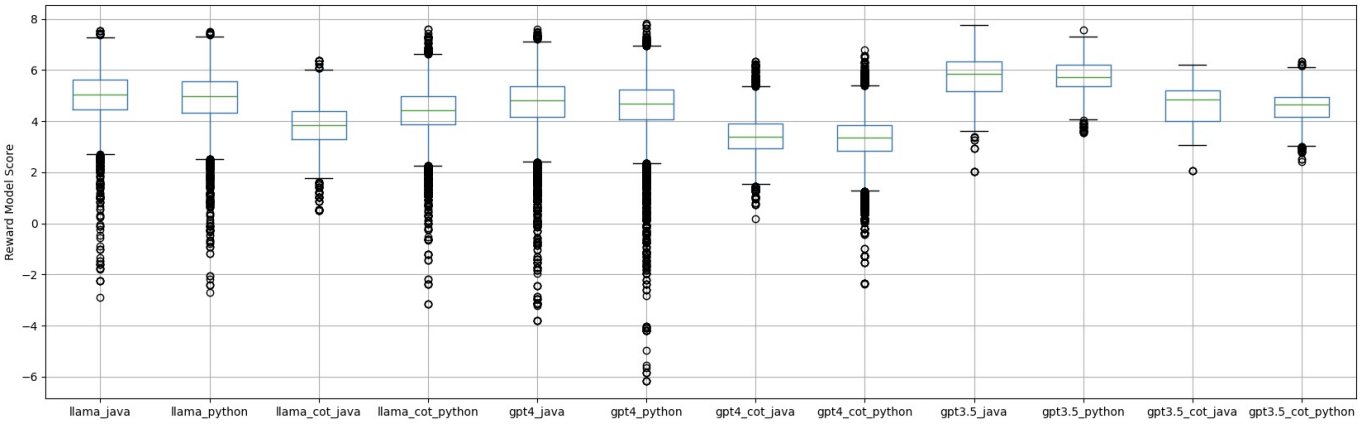

Fig. 2. Reward model scores distinguished by model, use of chain of thought, and programming language of dataset.

students in debugging programs and uses it in evaluating popular LLMs. We grow the original dataset of Python bugs translating the code to Java and revising the dialogues. Additionally, we build upon the previous authors' benchmarking of GPT-3.5 and 4 by considering Llama and using a reward model to provide insight.

Across the different evaluation methods that this paper reports on, we see that the LLMs considered perform somewhat better when benchmarked using the Java dataset over the Python dataset. Though this finding is promising, important future work entails the same benchmarking with less popular yet still commonly used programming languages.

We find that our data mostly agrees with that of the previous paper, and we glean that Llama holds much potential as a Socratic debugging tutor. Without any fine-tuning, we observe it out-performing GPT-3.5. Specialization of Llama for this task through efforts such as manipulation of training data and fine tuning are left as future work. Also left for future work is statistical significance testing on our findings.

Acknowledgment

This project was supported in part by the US National Science Foundation under award number OAC 1931363.

References

[1] W.-H. Kim and J.-H. Kim, "Individualized AI tutor based on developmental learning networks," *IEEE Access*, vol. 8, pp. 27927–27937, 2020.

[2] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, and D. J. Malan, "Teaching cs50 with AI: Leveraging generative artificial intelligence in computer science education," in *Proc. of 55th ACM SIGCSE TS V. 1*, (New York, NY, USA), p. 750–756, ACM, 2024.

[3] O. Joseph and C. Nwankwo, "Integrating AI and machine learning in stem education: Challenges and opportunities," *Computer Science and IT Research Journal*, vol. 5, pp. 1732–1750, 08 2024.

[4] E. Al-Hossami, R. Bunescu, J. Smith, and R. Teehan, "Can language models employ the socratic method? experiments with code debugging," in *Proc. 55th ACM SIGCSE TS, V. 1*, (New York, NY, USA), p. 53–59, ACM, 2024.

[5] R. Mason, Simon, B. A. Becker, T. Crick, and J. H. Davenport, "A global survey of introductory programming courses," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, (New York, NY, USA), p. 799–805, ACM, 2024.

[6] N. Raihan, M. L. Siddiq, J. C. Santos, and M. Zampieri, "Large language models in computer science education: A systematic literature review," in *Proc. of 56th ACM SIGCSE TS V. 1*, (New York, NY, USA), p. 938–944, ACM, 2025.

[7] C. Koutcheme, N. Dainese, S. Sarsa, A. Hellas, J. Leinonen, S. Ashraf, and P. Denny, "Evaluating language models for generating and judging programming feedback," in *Proc. of ACM SIGCSE TS V. 1*, (New York, NY, USA), p. 624–630, ACM, 2025.

[8] H. Kumar, I. Musabirov, M. Reza, J. Shi, X. Wang, and J. J. e. a. Williams, "Guiding students in using llms in supported learning environments: Effects on interaction dynamics, learner performance, confidence, and trust," *Proc. ACM Hum.-Comput. Interact.*, vol. 8, Nov. 2024.

[9] T. Y. Yeh, K. Tran, G. Gao, T. Yu, W. O. Fong, and T.-Y. Chen, "Bridging novice programmers and llms with interactivity," in *Proc. of 56th ACM SIGCSE TS V. 1*, (New York, NY, USA), p. 1295–1301, ACM, 2025.

[10] M. Hassan, Y. Chen, P. Denny, and C. Zilles, "On teaching novices computational thinking by utilizing large language models within assessments," in *Pro. of 56th ACM SIGCSE TS V. 1*, (New York, NY, USA), p. 471–477, ACM, 2025.

[11] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, (USA), p. 311–318, ACL, 2002.

[12] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, (Barcelona, Spain), pp. 74–81, ACL, July 2004.

[13] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," in *Proc. of ICLR*, 2020.

[14] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," in *Advances in Neural Information Processing Systems*, vol. 36, pp. 53728–53741, Curran Associates, Inc., 2023.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, and F. e. a. Xia, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, Curran Associates, Inc., 2022.

[16] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, and A. A.-D. et al., "The llama 3 herd of models," 2024.

[17] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, and D. A. et al., "Fine-tuning language models from human preferences," *CoRR*, vol. abs/1909.08593, 2019.

[18] M. Cao, A. Lam, H. Duan, H. Liu, S. Zhang, and K. Chen, "Compassjudger-1: All-in-one judge model helps model evaluation and evolution," 2024.

[19] J. Li, S. Sun, W. Yuan, R.-Z. Fan, H. Zhao, and P. Liu, "Generative judge for evaluating alignment," 2023.

[20] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, and S. C. et al., "Opt: Open pre-trained transformer language models," 2022.