Solving Proof Block Problems Using Large Language Models

Seth Poulsen
Utah State University
Logan, Utah, USA
seth.poulsen@usu.edu

Sami Sarsa
Aalto University
Espoo, Finland
sami.sarsa@aalto.fi

James Prather
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Juho Leinonen
University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.n

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Arto Hellas
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Brent N. Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

ABSTRACT

Large language models (LLMs) have recently taken many fields, including computer science, by storm. Most recent work on LLMs in computing education has shown that they are capable of solving most introductory programming (CS1) exercises, exam questions, Parsons problems, and several other types of exercises and questions. Some work has investigated the ability of LLMs to solve CS2 problems as well. However, it remains unclear how well LLMs fare against more advanced upper-division coursework, such as proofs in algorithms courses. After all, while known to be proficient in many programming tasks, LLMs have been shown to have more difficulties in forming mathematical proofs.

In this paper, we investigate the ability of LLMs to solve mathematical proofs by using Proof Blocks, a tool previously shown to efficaciously teach proofs to students. Our results show that GPT-3.5 is almost completely unable to provide correct solutions (11.4%), while GPT-4 shows a significant increase in correctness (64.8%). However, even given this improvement, current models still struggle to correctly order lines in a proof. It remains an open question whether this is a temporary situation or if LLMs will continue to struggle to solve these types of exercises in the future.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **CS1**;
• **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

AI; algorithms; artificial intelligence; ChatGPT; code generation; generative AI; GPT-3; GPT-4; large language models; OpenAI; Proofs; Proof Blocks

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03.
https://doi.org/10.1145/3626252.3630928

ACM Reference Format:
Seth Poulsen, Sami Sarsa, James Prather, Juho Leinonen, Brett A. Becker, Arto Hellas, Paul Denny, and Brent N. Reeves. 2024. Solving Proof Block Problems Using Large Language Models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630928

1 INTRODUCTION

Large language models have quickly revolutionized computing education [10, 36]. Tools such as GPT-4, Codex, and Github Copilot can write code from English language prompts [9], solve first and second semester programming assignments [15, 16], generate assignments [41], interpret programming error messages [24], and more. Students are already using them to write code and help solve their programming homework assignments in both helpful and unhelpful ways [37]. Researchers have been quick to point out the risks in student utilization of these tools, such as over-reliance and not recognizing inherent biases [2, 10].

Although it is clear that LLMs can easily solve much of the programming curriculum, one key area not explored yet is that of proofs. As a core part of a discrete mathematics course, proof writing remains a critical piece of computer science curriculum [45], forming a foundation for many upper-level algorithms courses. When GPT-4 was introduced, one of the key results researchers at OpenAI touted was its performance on math problems compared to GPT-3.5 [29]. This continued a trend of the latest model's increase in capability in solving math equations, similar to the increase seen from GPT-3 to GPT-3.5 [21]. How this performance increase can be applied to a computer science curriculum remains unseen. One recent (static) attempt to provide automated feedback for proof-writing by Poulsen et al. implemented Proof Blocks [35]. Proof Blocks, similar to Parsons Problems [14], is a drag-and-drop interface for arranging lines in a mathematical proof [35]. Recent work has shown that LLMs can be used to solve Parsons problems [38]. In this paper, we extend previous work on both Proof Blocks and solving Parsons problems via LLMs by benchmarking GPT-3.5 and GPT-4 against 128 Proof Blocks questions.

In our work we are guided by the following research questions:

RQ1 How do GPT-3.5 and GPT-4 perform in solving Proof Blocks problems?
RQ2 Are there Proof Blocks problems that are challenging to solve for GPT-3.5 and GPT-4?

This article is organized as follows. In Section 2, we discuss related work. Section 3 presents our methodology, including the data used in the study and how solutions generated by LLMs were evaluated. We present the results of the study in Section 4, which are then discussed in Section 5. Section 5 also outlines the limitations of our work. Section 6 concludes the work, presenting answers to our research questions and outlining avenues for future work.

2 RELATED WORK

In this section, we discuss related work on the use of generative AI coding tools, proof blocks in general, and using AI for proofs.

2.1 Use of Generative AI Coding Tools

Generative AI tools such as ChatGPT and Copilot are widely expected to drastically change the landscape of programming [1, 4, 8, 10] and software engineering [7] education. Such tools have proven proficient in producing code for CS1 [15], can work with Parsons problems [38], and can also deal with more advanced CS2 problems [16] and object-oriented concepts [6]. Despite the fact that these tools are not perfect, their progress in the last two years has been rapid and this improvement shows no signs of abating. A new book aimed at university-level introductory programming even advocates learning AI-assisted programming from day one [31]. These tools are also capable of more advanced tasks such as code competition problems [25], are being used by professional developers [3], and are expected to add trillions of dollars to global GDP [11].

These tools raise several challenges and opportunities for computing education beyond simply allowing students to generate code and the obvious academic integrity concerns [2, 10]. In particular, they can help instructors with many tasks [26]. For instance, they can generate programming questions including test cases and solutions [41], provide feedback to students [22], be used for grading [27] and for answering help requests [19], and classify and answer student questions on discussion boards [48].

As students begin to use these tools, several of the threats identified by researchers have come into clearer focus. For instance, Prather et al. observed students using Github Copilot, which is a generative AI coding tool that can produce highly accurate code suggestions [37]. They found that students often do not understand the code automatically generated by the tool simply because they did not write it. They also found that students will quickly accept incorrect code suggestions and tinker with that code before discovering they don't need it and deleting it, only to start the process over again. Their final finding was that some students used the AI tool to help them toward their goal, discovering new ways to achieve it, and doing so more quickly. Other user studies have found similar results [20, 46]. Although Proof Blocks are a different kind of exercise than open-ended code writing, there are similar concerns. Any kind of LLM-based tool for Proof Blocks would need to ensure that students do not become over-reliant on it and that it does not overwhelm them with feedback they don't understand (this is especially possible with proofs). The present work focuses on benchmarking Proof Blocks against LLMs, but future work that utilizes LLMs to provide feedback, hint generation, or even exploratory features, must keep these concerns in mind.

2.2 Proof Blocks

Understanding and constructing mathematical proofs is an essential aspect of the discrete mathematics curriculum, yet it is a very difficult topic for many students [18, 28]. Although there are many individual aspects that can be challenging for learners [43], even when all prerequisite knowledge is known, students are often unable to put the different elements together to correctly construct a proof without appropriate scaffolding [47]. To address this, Poulsen et al. introduced the idea of 'Proof Blocks' in 2021, which was a novel software tool leveraging a drag-and-drop mechanism to enable learners to assemble proofs from pre-written lines [34]. Inspired by the idea of Parsons problems [12, 14], the goal of Proof Blocks was to provide scaffolding to students learning to write mathematical proofs, in much the same way as Parsons problems provided scaffolding to students learning to write code. However, unlike a Parsons problem in which code lines typically must be arranged into a unique order, Proof Blocks problems are more flexible in the sense that only those lines in the proof that depend on other lines must appear before them.

Research exploring Proof Blocks has shown their significant potential for improving learners' proof comprehension and for fostering efficient learning [32, 35]. Students in the early phases of learning about *proof by induction* learned just as much from reading lecture notes and using Proof Blocks as they did by reading lecture notes and writing proofs from scratch, and did so while saving significant time [32]. Not only do students believe that Proof Blocks accurately represent their ability to write proofs, but when used as test questions they provide approximately the same amount of information about student knowledge as do written proofs [35].

To facilitate efficient grading of Proof Blocks, Poulsen et al. describe an auto-grader that uses a dependency graph to capture the relationships between subsets of blocks, and which grades as correct any topological sort of the graph [35]. The autograder allows for swift feedback to students and provides extensive opportunities for problem generation. Work on autograding Proof Blocks problems has been expanded to include partial credit grading, an aspect that poses computational challenges due to the vast solution space, and the expense of calculating the difference between an incorrect solution and a model solution. One novel algorithm for computing such an edit distance exhibited enormous performance improvements, up to two orders of magnitude, when compared to a naïve approach. This algorithm can also be used to provide feedback to students when solving Proof Blocks problems, and could be applied to other problems such as Parsons problems [33].

Figure 1 shows an example of a proof blocks problem taken from https://www.proofblocks.org/. The figure also gives an idea of how the Proof Blocks user interface looks like for students when they are solving Proof Blocks problems.

2.3 AI for Mathematical Proofs

Mathematicians and computer scientists have chased the promise of automating the construction of mathematical proofs for years.

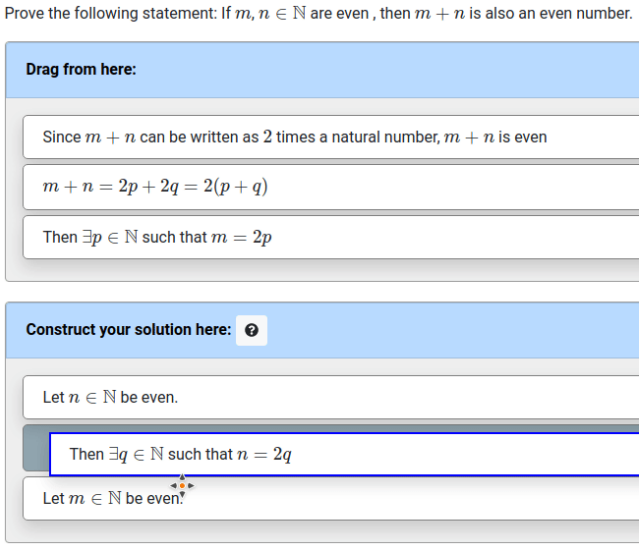

Figure 1: An example Proof Blocks problem and the user interface of the Proof Blocks tool.

The earliest history starts with heuristic search algorithms over a search space limited by a formal theorem proving language [39]. More recently, these search algorithms have been supplanted by reinforcement learning approaches, but even the best attempts have less than a 50% success rate in proving the lemmas and theorems in benchmark data sets [17, 40]. Researchers have sought improvements to these heuristic search algorithms using LLMs [49].

In parallel, researchers have worked on creating language models to solve math problems formulated as the informal mathematical language typically used by mathematicians rather than in formal theorem proving languages, an approach that has proved extremely difficult for many years [30, 44]. Most recently, GPT-4 has posted surprisingly good metrics in this area with huge improvements from GPT-3.5 to GPT-4 in all its math related benchmarks, which included AP Calculus exams, the quantitative portion of the GRE examination, and problems from the AMC series of high school mathematics competitions [29]. None of the above present benchmarks on questions like the mathematical proofs used for introductory discrete mathematics courses that are taught by the math and/or computer science departments of most universities. Thus, it remains an open question what the success rate of any AI powered system would be on such questions.

3 METHODOLOGY

3.1 Data

3.1.1 Proof Blocks problems. As our data set, we obtained from the authors all Proof Blocks problems that have been used in the data sets for prior publications [32–34]. Questions were originally written for Discrete Mathematics courses in computer science departments at two large research universities in the midwestern United States, one public and one private, as well as for a research study designed to measure learning gains of students using Proof Blocks. The dataset had in total 128 Proof Blocks questions, of which 91 did not have distractors (additional lines).

3.1.2 Proof Blocks problem solutions. We explored a variety of prompts to identify ones that could be used to produce Proof Blocks solutions with GPT-3.5 and GPT-4. As the final prompt, we used the format outlined in Listing 1, where the prompts start with the theorem, a prompt to reorder the lines to form a proof for the theorem, an additional instruction to disallow altering the text in lines, and the scrambled lines. Because the input to LLMs is ascii text, notice the example prompt forgoes mathematical symbols and uses textual descriptions such as `\frac{a+b}{2}` to encode

$$\frac{a+b}{2}$$

.

For each of the 128 Proof Blocks questions in the dataset, we generated five sets of scrambled lines and prompted both GPT-3.5 and GPT-4 to generate solutions for them[1]. The prompting was done with temperatures 0.0 and 0.7, where the temperature is used to control the degree of randomness in the outputs. In total, this yielded $n = 2560$ Proof Blocks problem solutions.

Listing 1: Example prompt

```
Consider the following theorem:
For any non-negative real numbers $a,b$ (i.e., $a,b \geq
    0$), $\frac{a+b}{2} \geq \sqrt{ab}$.
Reorder the following lines to form a proof of the
    theorem.
Do not alter the text in the lines.

$\frac{a+b}{2} \geq \sqrt{ab}$.
Since $a,b \geq 0$, $\sqrt{a}$ and $\sqrt{b}$ are real.
$(\sqrt{a} - \sqrt{b})^2 \geq 0$, since the square of any
     real number is non-negative.
$a+b - 2\sqrt{ab} \geq 0$.
$a+b \geq 2\sqrt{ab}$.
Let $a,b$ be arbitrary non-negative real numbers.
```

3.2 Evaluating Solutions

The Proof Blocks problem solutions were programmatically extracted from the LLM output. The resulting data were analyzed using the automated Proof Blocks grading algorithm outlined in [33], which relies on determining the edit distance from a given solution to a correct solution. For the present evaluation of Proof Blocks problem solutions, we focused on absolute correctness. That is, if the distance from a given solution to the model solution was greater than zero, it was considered incorrect.

When outlining the results, we first provide an overall model performance in Section 4.1. The subsequent results, outlined in Sections 4.2 and 4.3, focus on $n = 640$ solutions created with the best-performing model and temperature, which was in our case GPT-4 with the temperature 0.0.

4 RESULTS

[1]The data was generated using OpenAI APIs in August 2023. GPT-4 corresponds to the `gpt-4` model, while for GPT-3.5, we used the most recent `gpt-3.5-turbo`.

Listing 2: Example solutions from GPT-3.5 and GPT-4

```
SCRAMBLED BLOCKS:
We are trying to match each student to the sum of eight
     numeric digits.
Since each digit ranges from 0 to 9, the possible sums
     range from 0 to $8 \times 9 = 72$.
Therefore, there are 73 possible sums.
Since there are 39 students enrolled, each student can
     have a unique sum.
To guarantee that there are two students who have the
     same sum, we need strictly more students than
     possible sums.
Since there are 73 possible sums, we require 74 students.

GPT-3.5 SOLUTION: (score 33%)
We are trying to match each student to the sum of eight
     numeric digits.
Since each digit ranges from 0 to 9, the possible sums
     range from 0 to $8 \times 9 = 72$.
Since there are 73 possible sums, we require 74 students.
To guarantee that there are two students who have the
     same sum, we need strictly more students than
     possible sums.
Therefore, there are 73 possible sums.
Since there are 39 students enrolled, each student can
     have a unique sum.

GPT-4 SOLUTION: (score 100%)
We are trying to match each student to the sum of eight
     numeric digits.
Since each digit ranges from 0 to 9, the possible sums
     range from 0 to $8 \times 9 = 72$.
Therefore, there are 73 possible sums.
Since there are 39 students enrolled, each student can
     have a unique sum.
To guarantee that there are two students who have the
     same sum, we need strictly more students than
     possible sums.
Since there are 73 possible sums, we require 74 students.
```

4.1 Overall Model Performance

The model performance for GPT-4 and GPT-3.5 is outlined in Table 1 for temperatures 0.0 and 0.7. The overall success rate for GPT-3.5 is 11.4% and 11.7% for temperatures 0.0 and 0.7 respectively, while for GPT-4 the overall success rate was 64.8% and 61.9% for the same temperatures. This suggests that lower temperatures work slightly better for solving Proof Blocks problems, but the effect is not large (+3% points for GPT-4). More interestingly, GPT-4 vastly outperforms GPT-3.5, correctly solving Proof Blocks problems over half of the time compared to GPT-3.5's ~11%. An example of how GPT-3.5 and GPT-4 solves a combinatorics Proof Blocks problem is shown in Listing 2. In this case, GPT-4 got the problem fully correct, while GPT-3.5 was partially correct, scoring 33%.

4.2 Proof Blocks without Distractors

We continued the analysis by focusing on the performance of GPT-4 in solving a variety of Proof blocks without distractors. The overview of the results, including the topics of the problems and the correctness of the GPT-4 produced solutions is outlined in Table 2. From the table, we can see that overall, GPT-4 can solve Proof Blocks problems quite accurately when there are no distractors, being able to solve them on average 73% of the time. There are differences between topics in how well GPT-4 can solve Proof Blocks, ranging from 54% for questions related to 'combinatorics' (see Listing 2) to 100% for questions on 'algorithm analysis' and 'sets, functions'.

Model	Temperature	Overall Success Rate
GPT-3.5	0.0	11.4%
	0.7	11.7%
GPT-4	0.0	64.8%
	0.7	61.9%

Table 1: Comparison of performance of models. This is consistent with prior results that lower temperature is better for more technical and less creative tasks, and that GPT-4 vastly outperforms prior models on mathematical tasks.

4.3 Proof Blocks with Distractors

Finally, we studied the performance of GPT-4 in solving a variety of Proof blocks with distractors. The overview of the questions and topics, as well as correctness of the GPT-4 produced solutions is outlined in Table 3. Altogether, having distractors seems to hurt GPT-4's performance in solving Proof Blocks problems as the overall performance for questions without distractors was 73%, but only 44% for questions that had distractors. From the table, we can see that there are again differences between topics. Similar to the problems without distractors, 'combinatorics' problems are the hardest for GPT-4 to solve, with performance at 20%. When distractors are used, the easiest topic for GPT-4 is 'Pigeonhole Principle' with a success rate of 77%. Interestingly, GPT-4's performance on problems on this topic was actually better compared to performance without distractors (70%). Also interestingly, out of the topics that had questions with distractors, 'Cardinality' had the best performance without distractors (93%) but this was not reflected on the problems with distractors.

5 DISCUSSION

5.1 LLMs and Solving Proof Blocks

Our results highlight that state-of-the-art LLMs such as GPT-4 are rather capable of solving Proof blocks. Distractors make the problems harder to solve, as Proof blocks with distractors were solved only 44% of the time, while those without distractors were solved 73% of the time. This is in line with the performance of LLMs in solving Parsons problems, where problems with distractors were in general harder to solve than problems without distractors [38].

There were considerable differences in the performance between topics. For problems without distractors, the worst performance was for combinatorics (54% correctness), while two topics were solved perfectly (Algorithm analysis and Sets and functions). For problems with distractors, the correctness ranged from 20% for the combinatorics problems to 77% for the pigeonhole principle. Similar observations of the performance of LLMs varying by problem have also been observed when using LLMs to solve programming-related help requests [19], where the ability to address the help requests depended heavily on the problem and the manner in which the help request was phrased.

One reason for this difference in performance between topics could be the training data for the model. It could be that GPT-4 used more training data related to algorithms and sets and functions than it did combinatorics. Another explanation could be that GPT-4,

Topic	Questions	Attempts	Attempts Correct	Percentage Correct
Algorithm analysis	4	20	20	100%
Cardinality	9	45	42	93%
Combinatorics	7	35	19	54%
Graph Theory	8	40	26	65%
Logic	12	60	50	83%
Number Theory	14	70	41	59%
Pigeonhole Principle	6	30	21	70%
Probability	9	45	33	73%
Proof by Induction	18	90	62	69%
Sets, functions	4	20	20	100%
Total	91	455	334	73%

Table 2: GPT-4 Performance on Proof Blocks problems without distractors, grouped by topic.

Topic	Questions	Attempts	Attempts Correct	Percentage Correct
Cardinality	8	40	20	50%
Combinatorics	4	20	4	20%
Logic	4	20	10	50%
Number Theory	1	5	2	40%
Pigeonhole Principle	6	30	23	77%
Proof by Induction	14	70	22	31%
Total	37	185	81	44%

Table 3: GPT-4 Performance on Proof Blocks problems with distractors, grouped by topic.

as a next-token-predictor, is not actually searching the state space for these problems and therefore naturally lends itself better to certain types of Proof Blocks problems.

One implication of the results is that, for the topics where performance is good, LLMs could most likely be used to support students who are solving Proof Blocks problems. If the LLM can solve the problem, it might be able to generate a hint for students on what block to move next and where the block should be moved.

Although GPT-4 provided the best results, it is possible that others may not observe similar results even with the same prompts. Recent research has shown that the performance of GPT-3.5 and GPT-4 has changed over time, and the change has not always been an improvement [5]. This highlights a worrisome issue in relying on closed LLMs for research and practice; in effect, these observations call for further developments and research into open LLMs.

5.2 Comparison to Student Performance

In general, Proof Blocks problems are harder for students than typical multiple choice questions, but not as difficult as written proof questions [34]. The exam data from the evaluation of Proof Blocks questions as test questions from Poulsen et al. [34] contained 22 questions across 6 different topics. On these problems, students got the problem correct 61% of the time on their first attempt, and had gotten the problem correct by their third attempt 85% of the time [34]. Student data is only publicly available for the some of the questions, and which questions students were given with distractors is confounded with the topics. Thus, while we cannot draw any particular comparisons between GPT-4 and students on particular topics, we see that the performance of GPT-4 is roughly similar to the performance of students who have been taught the material–slightly better on problems without distractors, and worse on problems with distractors.

The finding that the performance of GPT-4 is similar to students in solving Proof Blocks problems suggests that they are more difficult for GPT-4 compared to, for example, code writing tasks and creating code explanations. Prior work has found that even Codex, which is an earlier, less capable model compared to GPT-4, performed better than the average student in code writing tasks [15]. Similarly, GPT-3 seems to outperform students in its ability to explain code in natural language as the explanations generated by GPT-3 were rated as being easier to understand and being better summaries of the code compared to explanations created by students in prior work [23].

5.3 Evolution of LLMs

Our results also highlight the impact of the evolution of LLMs. In our case, GPT-3.5 had an average success rate of 11.4%, while GPT-4 had an average success rate of 64.8% (both for temperature 0.0). Such an improvement is considerable, and suggests that future improvements to LLMs might also yield improvements in their capability of solving Proof blocks. In the broader CER literature, the improvement and evolution of LLMs has been highlighted in multiple areas. As an example, there is a considerable difference in the performance of Codex and GPT-3.5 in solving students' programming-related help requests [19]. Similarly, the performance of GPT-4 in passing various programming assessments is better compared to earlier models [42]. Using data from three Python courses with varying assessments including multiple-choice questions, programming

exercises, and large projects, Savelka et al. found that GPT-3's performance was such that it would have failed the courses, while GPT-4 would have passed the courses easily.

While for some tasks such as Proof blocks, the performance of GPT-4 is vastly better compared to earlier models, when considering code generation, GPT-4 has shown comparatively moderate improvements over Codex in tasks such as generating code. This might be due to earlier models such as Codex already having quite impressive performance in code generation [15, 16], so there is less room to improve for GPT-4.

5.4 Limitations

Our study comes with a number of limitations, which we address here. First, although we explored a number of prompts, i.e. did prompt engineering, we cannot state that the prompts that we used are the best possible ones for solving Proof Blocks. This is an inherent problem of Large Language Models where, due to the vast parameter space, finding an optimal prompt is practically impossible – the problem is exacerbated by semantically similar prompts potentially leading to different outcomes [13]. Second, due to how the performance of GPT-3.5 and GPT-4 can change in the same tasks over time due to updates on the side of the model developers [5], it is possible that our results could not be replicated even with the same dataset using the same models in the future. There is a need for open LLMs that can be versioned and studied in more detail. Third, we used so-called 'zero-shot' prompting (i.e., we did not provide any examples of solving Proof blocks to the model) and did not engage in dialogue with the model for producing the answers, and thus, it is possible that students who were to use ChatGPT or a similar system that keeps track of the conversation history could observe better performance in the tasks.

6 CONCLUSION

In this study, we explored the potential of LLMs for solving Proof Blocks problems. Proof Blocks problems are problems where students are given a theorem and a set of scrambled lines that need to be ordered to form the proof for the theorem. To summarize, our research questions and their answers are as follows.

Question: *How do GPT-3.5 and GPT-4 perform in solving Proof Blocks problems?* **Answer**: Both GPT-3.5 and GPT-4 can solve Proof Blocks problems, although they have vast differences in performance. In our study, GPT-3.5 was able to solve approximately 11% of the given Proof Blocks, while GPT-4 achieved an almost 65% success rate in solving the problems.

Question: *Are there Proof Blocks problems that are challenging to solve for GPT-3.5 and GPT-4?* **Answer**: In short, yes. Focusing on GPT-4, as its performance is considerably better than the performance of GPT-3.5, we observed that problems with distractors were considerably more difficult than problems without distractors. For problems without distractors, GPT-4 was able to solve approximately 73% of the given problems, while for problems with distractors, the corresponding number was 44%. Furthermore, we also observed that there are considerable differences in performance depending on the topic of the Proof Blocks problem.

Our results highlight the possibility of using LLMs such as GPT-4 for solving Proof Blocks problems. The practical implications of this include the possibility of using GPT-4 and future LLMs as an additional tutor for solving the Proof Blocks, which can help students in improving proof comprehension [32, 35].

This work opens up multiple research directions. As GPT-4 was quite successful in solving Proof Blocks problems, a natural next step would be analyzing its performance in solving free-form proofs. These could be manually graded using rubrics currently in use for student work. If GPT-4 were able to solve free-form proofs, then it could be used by instructors to create example solutions for free-form proof problems. Another avenue for future research is studying in more detail the types of mistakes that GPT-4 does, and whether prompt engineering could help it solve problems where it failed using our current prompts.

ACKNOWLEDGMENTS

We are grateful for the grant from the Ulla Tuominen Foundation to Juho Leinonen.

REFERENCES

[1] Brett A. Becker, Michelle Craig, Paul Denny, Hieke Keuning, Natalie Kiesler, Juho Leinonen, Andrew Luxton-Reilly, Lauri Malmi, James Prather, and Keith Quille. 2023. Generative AI in Introductory Programming Education. https://csed.acm.org/large-language-models-in-introductory-programming/ CS2023 Curricular Practices Volume.

[2] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, NY, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[3] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot. *Commun. ACM* 66, 6 (may 2023), 56–62. https://doi.org/10.1145/3589996

[4] Peter Brusilovsky, Barbara J. Ericson, Cay S. Horstmann, Christian Servin, Frank Vahid, and Craig . 2023. Significant Trends in CS Educational Material: Current and Future. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 1253. https://doi.org/10.1145/3545947.3573353 Draft report: https://csed.acm.org/the-future-of-cs-educational-materials/.

[5] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? *arXiv preprint arXiv:2307.09009* (2023).

[6] Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 61–67. https://doi.org/10.1145/3587102.3588814

[7] Marian Daun and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 110–116. https://doi.org/10.1145/3587102.3588815

[8] Paul Denny, Brett A. Becker, Juho Leinonen, and James Prather. 2023. Chat Overflow: Artificially Intelligent Models for Computing Education - RenAIssance or ApocAIypse?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 3–4. https://doi.org/10.1145/3587102.3588773

[9] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 1136–1142. https://doi.org/10.1145/3545945.3569823

[10] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* (2024). https://doi.org/10.1145/3624720

[11] Thomas Dohmke, Marco Iansiti, and Greg Richards. 2023. Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle. arXiv:2306.15033 [econ.GN]

[12] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20)*. ACM, NY, NY, USA, 195–202. https://doi.org/10.1145/3373165.3373187

[13] Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics* 9 (2021), 1012–1031.

[14] Barbara J. Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S. Miller, Briana B. Morrison, Janice L. Pearce, and Susan H. Rodger. 2022. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. ACM, NY, NY, USA, 191–234. https://doi.org/10.1145/3571785.3574127

[15] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. ACM, NY, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

[16] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE '23)*. ACM, NY, NY, USA, 97–104. https://doi.org/10.1145/3576123.3576134

[17] Emily First, Yuriy Brun, and Arjun Guha. 2020. TacTok: Semantics-Aware Proof Synthesis. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 231 (nov 2020), 31 pages. https://doi.org/10.1145/3428299

[18] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process. *SIGCSE Bull.* 40, 1 (mar 2008), 256–260. https://doi.org/10.1145/1352322.1352226

[19] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *arXiv preprint arXiv:2306.05715* (2023).

[20] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. ACM, NY, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[21] Anis Koubaa. 2023. GPT-4 vs. GPT-3.5: A Concise Showdown. *Preprints* (March 2023). https://doi.org/10.20944/preprints202303.0422.v1

[22] Charles Koutcheme. 2022. Towards Open Natural Language Feedback Generation for Novice Programmers Using Large Language Models. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '22)*. ACM, NY, NY, USA, Article 29, 2 pages. https://doi.org/10.1145/3564721.3565955

[23] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. arXiv:2304.03938 [cs.CY]

[24] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

[25] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[26] Stephen MacNeil, Joanne Kim, Juho Leinonen, Paul Denny, Seth Bernstein, Brett A. Becker, Michel Wermelinger, Arto Hellas, Andrew Tran, Sami Sarsa, James Prather, and Viraj Kumar. 2023. The Implications of Large Language Models for CS Teachers and Students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 1255. https://doi.org/10.1145/3545947.3573358

[27] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2023. Machine Learning-Based Automated Grading and Feedback Tools for Programming: A Meta-Analysis. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 491–497. https://doi.org/10.1145/3587102.3588822

[28] Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013*. ACM/Association for Computing Machinery.

[29] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[30] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP Models really able to Solve Simple Math Word Problems? arXiv:2103.07191 [cs.CL]

[31] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming with GitHub Copilot and ChatGPT*. Manning: Shelter Island, NY, USA.

[32] Seth Poulsen, Yael Gertner, Benjamin Cosman, Matthew West, and Geoffrey L. Herman. 2023. Efficiency of Learning from Proof Blocks Versus Writing Proofs. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 472–478. https://doi.org/10.1145/3545945.3569797

[33] Seth Poulsen, Shubhang Kulkarni, Geoffrey Herman, and Matthew West. 2023. Efficient Feedback and Partial Credit Grading for Proof Blocks Problems. In *International Conference on Artificial Intelligence in Education*. Springer, 502–514.

[34] Seth Poulsen, Mahesh Viswanathan, Geoffrey L Herman, and Matthew West. 2021. Evaluating Proof Blocks Problems as Exam Questions. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. 157–168.

[35] Seth Poulsen, Mahesh Viswanathan, Geoffrey L. Herman, and Matthew West. 2022. Proof Blocks: Autogradable Scaffolding Activities for Learning to Write Proofs. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. ACM, NY, NY, USA, 428–434. https://doi.org/10.1145/3502718.3524774

[36] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Peterson, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. https://doi.org/10.48550/arXiv.2310.00658 arXiv:2310.00658 [cs.CY] To appear in Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2023), July 7–12, 2023, Turku, Finland, 10.1145/3623762.3633499.

[37] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That It Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (nov 2023), 31 pages. https://doi.org/10.1145/3617367

[38] Brent Reeves, Sami Sarsa, James Prather, Paul Denny, Brett A. Becker, Arto Hellas, Bailey Kimmel, Garrett Powell, and Juho Leinonen. 2023. Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 299–305. https://doi.org/10.1145/3587102.3588805

[39] Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric, and Zachary Tatlock. 2019. QED at Large: A Survey of Engineering of Formally Verified Software. *Foundations and Trends® in Programming Languages* 5, 2-3 (2019), 102–281. https://doi.org/10.1561/2500000045

[40] Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. 2020. Generating Correctness Proofs with Neural Networks. (2020), 14.

[41] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, NY, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[42] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. *arXiv preprint arXiv:2306.10073* (2023).

[43] GJ Stylianides, AJ Stylianides, and K Weber. 2017. Research on the teaching and learning of proof: Taking stock and moving forward. In *Compendium for Research in Mathematics Education*, Jinfa Cai (Ed.). National Council of Teachers of Mathematics, Chapter 10, 237–266.

[44] Sowmya S Sundaram, Sairam Gurajada, Marco Fisichella, Deepak P, and Savitha Sam Abraham. 2022. Why are NLP Models Fumbling at Elementary Math? A Survey of Deep Learning based Word Problem Solvers. arXiv:2205.15683 [cs.CL]

[45] Association for Computing Machinery (ACM) The Joint Task Force on Computing Curricula and IEEE Computer Society. 2016. *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. Technical Report. NY, NY, USA.

[46] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665

[47] Keith Weber. 2001. Student difficulty in constructing proofs: The need for strategic knowledge. *Educational Studies in Mathematics* 48, 1 (Oct. 2001), 101–119. https://doi.org/10.1023/A:1015535614355

[48] Paul Zhang, Brandon Jaipersaud, Jimmy Ba, Andrew Petersen, Lisa Zhang, and Michael R. Zhang. 2023. Classifying Course Discussion Board Questions Using LLMs. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 658. https://doi.org/10.1145/3587103.3594202

[49] Shizhuo Dylan Zhang, Talia Ringer, and Emily First. 2023. Getting More out of Large Language Models for Proofs. arXiv:2305.04369 [cs.FL]