# Investigating the Capabilities of Generative AI in Solving Data Structures, Algorithms, and Computability Problems

Nero Li
yimil31@uci.edu
University of California, Irvine
Irvine, USA

Shahar Broner
sbroner@uci.edu
University of California, Irvine
Irvine, USA

Yubin Kim
yubk1@uci.edu
University of California, Irvine
Irvine, USA

Katrina Mizuo
kmizuo@uci.edu
University of California, Irvine
Irvine, USA

Elijah Sauder
esauder@uci.edu
University of California, Irvine
Irvine, USA

Claire To
clairt2@uci.edu
University of California, Irvine
Irvine, USA

Albert Wang
alberw5@uci.edu
University of California, Irvine
Irvine, USA

Ofek Gila
ogila@uci.edu
University of California, Irvine
Irvine, CA, USA

Michael Shindler
mikes@uci.edu
University of California, Irvine
Irvine, USA

## Abstract

There is both great hope and concern about the future of Computer Science practice and education concerning the recent advent of large language models (LLMs).

We present the first study to extensively evaluate the ability of such a model to solve problems in Computer Science Theory. Specifically, we tested 165 exam-level problems across 16 specific topics related to computer science theory, ranging from preliminary data structures to algorithm design paradigms to theory of computation (automata and complexity). Our results use the recent popular models (GPT-4 and GPT-4o). This is a rapidly evolving field, with model performance continuously improving. We present our results primarily as an indication of what they can already achieve—equivalently how they can already be useful—today, fully expecting them to improve even further in the near future.

Our results show that what was very recently a state-of-the-art model (GPT-4) can solve 77% of free-response problems in data structures and algorithms with little to no guidance. The latest model, GPT-4o, can solve around 46% of the Theory of Computation problems we posed, with predictable categories for which problems it could not solve. When broken down by topic, the model can solve 80% of problems in 4 out of the 15 topics and at least half in 8 other topics. Other problems, namely more visual problems, either require more substantial coaching or seem to still be beyond the capabilities of the language model–for now.

By understanding the strengths and limitations of these models for solving theory problems, we can open the door to future work, ranging from human educational assessment on the topic to automated tutors for learners of the subject.

*SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA*
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701957

## CCS Concepts

• **Theory of computation** → *Algorithm design techniques*; • **Social and professional topics** → **Computational thinking**; • **Applied computing** → *Computer-assisted instruction.*

## Keywords

algorithm design techniques, data structures, computational thinking, computer-assisted instruction, ChatGPT, generative AI, large language models, GPT-4, GPT-4o

**ACM Reference Format:**
Nero Li, Shahar Broner, Yubin Kim, Katrina Mizuo, Elijah Sauder, Claire To, Albert Wang, Ofek Gila, and Michael Shindler. 2025. Investigating the Capabilities of Generative AI in Solving Data Structures, Algorithms, and Computability Problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701957

## 1 Introduction

The past few years have seen significant improvement in the ability of AI systems, trained on large amounts of data, to tackle a wide array of complex tasks and challenges, revolutionizing various domains across academia and industry. This progress is especially notable in problems that can be stated and solved primarily using words, as observed in domains like code completion and word problem-solving.

Given a natural language prompt for a problem, systems such as GitHub Copilot [14] can produce code that, in many cases, will either solve the problem or make significant progress toward doing so. Similarly, some systems, such as ChatGPT, can produce natural language solutions to problems with varying degrees of correctness, similar to what an undergraduate student might be expected to produce in an algorithms course. There has been significant recent work in evaluating these systems as they relate to Computer Science Education, particularly at the level of first-year undergraduate programming courses. The work related to CS1/CS2 is largely programming-related, as befits these classes. The only previous work we are aware of that examines the performance of LLMs in upper-division computer science courses in detail is by Golesteanu and Vowinkel and Dougherty [15], who determined that GPT-4 performs at an average level equivalent to a B- student on undergraduate theory questions. Our study differs from theirs in three main ways: (1) we emphasize open-ended, free-response questions; (2) we incorporate feedback, allowing GPT to refine its answers; and (3) we use the GPT-4o model in part of our experiment.

In this project, we extend the state of understanding of these systems by investigating their ability to solve more advanced problems in the CS Theory curricula. We summarize our work with the following guiding research question:

**RQ** How does ChatGPT perform on more advanced data structures, algorithms, and Theory of Computation free-response questions?

This is effectively 16 research questions. With GPT-4, we studied the performance on some simpler data structures (stacks, queues, binary trees), elementary and fundamental graph algorithms (traversals, Dijkstra's single-source shortest path algorithm, minimum spanning trees), and algorithm design techniques (graph reductions, divide-and-conquer, dynamic programming, greedy, network flow). After we investigated this, GPT-4o has been released, so we used GPT-4o to study the performance of more advanced topics, such as Theory of Computation, regular and context-free languages, and NP-completeness. In total, we evaluated GPT-4 on 62 free response questions and GPT-4o on 103 free response questions, with an emphasis on material that tends to appear in more advanced undergraduate Data Structures and Algorithms, along with Theory of Computation courses.

Unlike much previous work, the entirety of this project studied problems that require human expertise to grade at each step; unlike programming problems, there was no auto-grader option here.

## 2 Background and Literature Review

Large Language Models (LLMs) are a recent advent from the field of Artificial Intelligence / Machine Learning. Among them are ChatGPT [28], which interacts with the user in a manner similar to text communication with another human would appear. Another is GitHub Copilot [14], based on OpenAI's Codex [27], which is prompted by natural language and produces (typically usable, sometimes correct) program code as a result.

### 2.1 Generative AI Performance on Class Assignments

There has been much recent research towards the abilities of LLMs to solve programming assignments at various levels, ranging from CS1 to advanced courses. On one hand, this can raise academic integrity concerns if a student can "solve" a homework problem entirely by use of LLM; on the other hand, these can open opportunities in education that were not present until very recently.

The first research to investigate the use of LLMs to solve introductory course assignments was Finnie-Ansley et al. [11], who also showed that Codex could not only solve introductory course assignments in Python, but also summative exam questions. Their investigation showed that Codex out-performed 80% of the students in the class. Denny et al. [6] conducted a more exhaustive study and found that Copilot could get a very large number of such assignments fully correct either on the first try or with minimal natural-language prompting. Another exhaustive study by Piccolo et al. [32] observed similar results in an introductory-level Python course with an eye towards Bioinformatics, using 184 programming exercises. Savelka et al. [39] evaluated the davinci model, which was, at the time, the largest and most powerful model of GPT[1], on 599 exercises from three Python courses, consisting of 69 programming assignments and the remainder multiple-choice questions, some of which required reasoning about code. That model did not do well enough to pass the course. Shortly thereafter, Savelka et al. [38] demonstrated that the use of GPT-4 led to the models being able to pass, although some limitations still held.

LLMs have also been shown to be effective for solving assignments in introductory courses taught in Java by Ouh et al. [29] and Destefanis et al. [9].

Not all studies focused on artifacts that lend themselves to fast automatic grading. Malinka et al. [25] evaluated, with an eye towards information security, the ability of GPT to handle not only programming assignments, but also written artifacts including exams and even short research papers. Furthermore, this study was conducted in Czech, indicating that the models are not limited to the English language.

Perhaps surprisingly, for all the success at introductory level free-form programming problems, Codex had less success with Parsons problems [34]. Parsons problems, invented by Parsons and Haden [30], are a scaffolding method in teaching introduction to programming wherein students solve a puzzle whose solution is working code and where unsuccessful attempts indicate common errors. These avoid some difficulties for novice programmers while allowing some benefits that writing code for practice would provide. Tuning the difficulty via *faded Parsons problems* exists as well [13, 43]. It is interesting to note that while Parsons problems are easier than free-form code to write for a human learner, Reeves et al. [34] discovered that Codex was more successful with the latter than the former. Poulsen et al. [33] found that GPT-3.5 had a poor success rate with solving Proof Block Problems, which can be thought of as applying the concept of Parsons Problems to proofs. They also found that GPT-4 was significantly better.

There have been three previous investigations that included data structures and algorithms problems. The work of Wang et al. [42] found most of their sample of algorithms problems, of which 13 were short answer questions, to be either unsolvable or partially solvable by these models. However, the breakdown of which of these were solvable, and whether they came from the short answer section, from programming assignments (4), or from multiple choice questions (13), is neither available nor the point of their study. Similarly, most of their data structures problems (8PR, 4 SA, 18 MC) were solvable by the models, but the breakdown by category did not appear. However, the goal of that work was to broadly review capabilities across many courses, so this is reasonable.

Finnie-Ansley et al. [12] looked into Codex performance on CS2 questions, a topic that includes preliminary algorithms (sorting and searching) and simpler data structures (lists, stacks, queues, binary

[1] For a description of the various models of OpenAI's GPT, see https://platform.openai.com/docs/models. For older models, check "Deprecation history"

heaps, hashing, and binary search trees). They compared the Codex to students taking their class and found it performed in the upper quartile.

Golesteanu, Vowinkel and Dougherty [15] analyzed ChatGPT's performance on exams for their Theory of Computation course, noting its ability to write proofs was largely predictable by whether the problem, or one like it, had been present in the training cases.

Our work extends [12] by covering more advanced data structures and algorithms topics, as well as investigating the ability of GPT-4 to design algorithms using core paradigms that are common to algorithms courses [23]. We extend [15] with GPT-4o by exploring a larger number of ToC questions that were drawn from our university's class problem sets and exams.

## 2.2 Educational Opportunities Afforded By LLMs

There are many prospective benefits to LLMs in an educational setting. Ross et al. [36] used Copilot to develop a prototype Programmer's Assistant, making progress on an idea whose vision had been described in 1990 [35]. Other prospective benefits are in curricular design [40], course programming assistants [22], personalized assignments [8, 37], and code explanations to programming learners [20, 26].

There are risks of these in the educational setting too; for an excellent discussion of these risks, and more information about the opportunity they present, see [7]. For information about the trustworthiness of AI in these settings, see [4, 18]. With this in mind, for adoption plans to programming classes, see [2, 19]; for uses of GPT to provide feedback on programming assignments, see [1]; and [5, 16, 20, 21, 24, 31, 41] for other topics about LLMs in education.

In short, the effect on the educational setting of LLMs is that of a powerful tool. For an in-depth look at Copilot as an educational tool, both in what it can achieve in a CS1 classroom (beyond producing code for a programming assignment) and a discussion of the effect on the educational landscape, see the work of Wermelinger [44].

There is much more work about generative AI, but our focus in this project is as it relates to educational and classroom issues.

# 3 Method

As of the time of writing this paper, ChatGPT by OpenAI is one of the predominant general purpose large language models (LLMs). Since ChatGPT is continuously being updated, we performed our testing on its underlying model directly, invoking the GPT-4 API. Specifically, our results can be replicated by invoking the gpt-4-0613 version of the model, which we refer to simply as GPT-4 within this paper, for data structures and algorithms topics, and GPT-4o for Theory of Computation. As of this writing, GPT-4o (omni) is OpenAI's most advanced model, allowing multimodal input, such as PDF, which we extensively utilized.

At a high level, our methodology is as follows. We selected representative free response questions from sub-topics within data structures, algorithms, and Theory of Computation.

For each topic, we chose two "graders" to evaluate the system on that topic. For GPT-4 or older versions, the grader copied and pasted the questions into a new chat. For GPT-4o, the grader first sent the PDF file containing the questions to the model directly and then asked the model to give a brief answer for all the papers' problems. In both protocols, after the model responded, the grader may have needed to ask sub-questions or provide hints, acting like a student trying to guide the model to reach the correct answer through varying degrees of assistance. Graders also did this when the answers were ambiguous. This process was repeated several times per grader in order to reach an accurate assessment of the results. Since GPT-4 often was inconsistent in its responses, graders compared their scores for each problem with each other. When both verdicts matched, the result was accepted, and when they differed, the lower grade was picked.

The problems were selected from three of our institution's core theory courses: a second-year post-CS2 Data Structures course, an upper division course covering the Design and Analysis of Algorithms, and an upper division elective about Theory of Computation. Each of the selected problems appeared either in a homework assignment (as a required or suggested question) or on an exam (as an in-class exam given for credit or a practice one intended to help students prepare for the former). The problems are available at https://ics.uci.edu/~mikes/papers/TheoryGPT2025/.

The topics for which we provided questions are as follows:

- Stacks and Queues
- Binary Trees
- Graph Algorithms, such as Dijkstra's Single Source Shortest Path Algorithm and Minimum Spanning Trees
- Graph Reductions, wherein a problem is posed whose solution is to create a graph and use a traversal or fundamental algorithm upon that graph to solve the original problem.
- Network Flow, wherein a problem is posed whose solution is to create a graph and use maximum flow or minimum cut upon that graph to solve the original problem.
- Divide and Conquer, wherein a problem is posed to be solved via that algorithm design technique.
- Dynamic Programming, wherein a problem is posed to be solved via that algorithm design technique.
- Greedy Algorithms, wherein we either design an algorithm using this technique with a proof of correctness, adapt an existing greedy algorithm, or prove that a greedy algorithm is incorrect by providing a counter-example.
- Regular Languages, involving designing and analyzing finite-state machines and using proof techniques, such as closure properties and the pumping lemma, to verify membership and non-membership in the class of regular languages.
- Context-Free Languages, involving designing and analyzing pushdown automata and using proof techniques, such as closure properties and the pumping lemma, to verify membership or non-membership of context-free languages.
- Turing Machines, involving designing and analyzing Turing machines.
- Decidability, using proof techniques, such as reduction, to prove decidability and undecidability.
- Computational Complexity, using proof techniques, such as reduction, to prove a problem to be NP-complete.

With adequate help, GPT-4 was able to solve nearly all the problems in the topics related to data structures and algorithms. We define help as guidance provided to the model by the testers. This guidance became increasingly more significant if the model could not correctly solve the problem. For example, if the model provided an incorrect response, a tester might give it a small hint by highlighting a specific part of the problem or pointing out a minor mistake. Then, if the model was still incorrect, the tester could provide more assistance in the form of a more meaningful hint, such as a small part of the solution or the correct approach. Our goal was to distinguish the severity of "help" that GPT-4 required. Problems were then graded on a 5-point scale, with scores 4 to 5 corresponding to questions where a naive student could easily solve the problems using GPT-4, and scores 1 to 3 corresponding to questions where more experience and thoughtful guidance were required to lead GPT-4 to a correct answer.[2]

In contrast, the LLM performed relatively poorly on the more advanced topics, where we used GPT-4o instead of GPT-4. We believe this is primarily due to the inherent difficulty of these problems, rather than to differences of capabilities in these models.

A more detailed breakdown of these scores can be found in Table 1.

| Score | Meaning |
|---|---|
| 1 | Unable to solve the problem without explicitly being told the solution. |
| 2 | Solves the problem in a way that seems believable but has errors, requiring specific suggestions that could only be identified by an astute student or teaching assistant. |
| 3 | Solves the problem correctly after pointing out several problems that an average student would be able to identify. |
| 4 | Solves the problem correctly after pointing out a clear problem that even a naive student would be able to identify. |
| 5 | Solves the problem correctly and consistently with no dialogue. |

**Table 1: Scoring Criteria for responses**

## 4 Results and Discussion

As discussed in Section 3 (Method), our final results were grouped into questions with scores of 4 to 5, where a very naive student equipped with ChatGPT would be able to trivially solve, and into questions with scores from 1 to 3, where more experienced and thoughtful guidance was required. The results are in Table 2.

It is worth noting that some of GPT's answers are strikingly similar to online solutions. However, there are many ways to circumvent this, such as making minor changes to the API's system message[3], using a different version of the model, or simply asking GPT to reword its response. As such, we do not think storing a database of GPT responses or any similar method to identify cheating is viable long-term.

| Topic | # Trivially Solvable | Total # Questions |
|---|---|---|
| Binary Trees | 2 | 5 |
| Greedy Algorithms | 5 | 7 |
| Dynamic Programming | 7 | 9 |
| Network Flow | 7 | 9 |
| Graph Algorithms | 11 | 14 |
| Graph Reductions | 7 | 8 |
| Divide and Conquer | 7 | 8 |
| Stacks and Queues | 2 | 2 |
| Aggregated Results from GPT-4 | 48 | 62 |
| Turing Machines | 0 | 8 |
| Context-Free Languages | 1 | 28 |
| Regular Languages | 5 | 10 |
| Decidability | 6 | 10 |
| Proofs for Regular | 4 | 6 |
| Computational Complexity | 19 | 26 |
| Proofs for Context-Free | 12 | 15 |
| Aggregated Results from GPT-4o | 47 | 103 |

**Table 2: Results comparing the number of trivially solvable questions (obtaining scores of 4 or 5) versus the total number of questions for each topic. The results are ordered from worst to best performing topics for each model.**

Another observation is the relative ease for a human does not reflect in the relative difficulty for the system. Dynamic programming is reported to be a difficult topic for undergraduates [10], yet the system was able to solve most of these with ease. By contrast, binary tree questions are easier for students to solve, yet the system struggled with over half of the problems we posed to it. This is consistent with the findings from [17] that GPT performance and relative human difficulty are not necessarily correlated.

An interesting incident occurred with a binary tree question. We asked the system to count how many binary search trees have a particular property. The system answered correctly using the product of Catalan numbers. When asked to answer without using that method, as an undergraduate studying binary search trees for the first time is unlikely to know about these, it gave several incorrect answers involving similar combinatorics. Ultimately, we treated this as an incorrect answer, as it could not solve it using techniques we would expect a student to use, which limits its usefulness for many important applications.

The system can accurately respond to questions about the conceptual aspects of languages and automata. However, it frequently struggles with designing an automata or grammar to represent languages. While the system showed capability in creating graphs using LaTeX and the TikZ package, the system could ultimately only solve the simplest, most common problems. With more complicated questions, the generated graphs often contained logical, syntactical, and visualization errors.

When asked to provide different derivations of a string in a given language from a context-free grammar (CFG), a question students

[2] While grading, we added a score of 6 for questions where GPT-4 solved consistently without a normal hint that would be given to students, provided with the question. For simplicity, we consolidated the score of 5 and 6 within this paper.

[3] The API allows adding 'system' messages which dictate the 'personality' that GPT embodies in its responses. A slightly different system message can correspond to significant prompt rewording.

generally consider easy, the generated derivations do not adhere to the defined grammar, and the steps taken in the derivations are seemingly random. Interestingly, when producing context-free grammars (CFGs) and pushdown automata (PDAs) for a language, GPT often defaults to providing the representation for the language of palindromes, regardless of the requested language.

While GPT-4o has image recognition capabilities, there are still mistakes with reading graphs. When asked to find a Hamiltonian path in the graph, the initial reading of the edges was incorrect, but after providing feedback, the model generated a valid Hamiltonian path.

The system is capable of generating pumping lemma proofs for regular and context-free languages. Initially, the generated proofs were often incomplete, failing to demonstrate that the selected string would break the language when pumped. After receiving feedback, the system was able to provide fully correct and comprehensive proofs to straightforward problems. When faced with more complex problems, the selected string was not initially part of the language, rendering the proof invalid. Additionally, the system demonstrated the ability to prove the closure of a language under various operations through closure properties.

In the following two sub-sections, we will discuss successful and unsuccessful attempts by the model to solve a subset of our problems. While we are describing the problems for context, the de-anonymized version of the paper will include a link to the full set of questions we used to test this. That will allow for more context of the results and also be available for any desired replication or extension studies.

## 4.1 Successful Responses from the Model

An interesting success for the model is for a hard dynamic programming problem from an algorithm design class. This is an original question and does not, to our knowledge, appear in a textbook. While general dynamic programming problems only have one or two cases in their recurrence relations, the solution to this problem requires three—involving either one, two, or a variable number of terms. This problem is question 5 in the dynamic programming portion of our problem set.

The prompt was provided and the system provided a neatly arranged but incorrect answer involving a 2D array, complete with a reasonable attempt at a Python implementation. The system correctly determined that this solution has running time $O(n^3)$. The grader alerted the system that this is incorrect, that a 2D array is not needed, that it can be solved in $O(n)$ time, and emphasized key parts of the problem statement.

From this, the system was able to figure out the correct recurrence for each of the three cases and the order in which the iterative algorithm should memoize solutions.

## 4.2 Unsuccessful Responses from the Model

### 4.2.1 *Binary Tree Traversals.*

A surprising unsuccessful attempt by the model involved a classic binary tree problem, where a student must draw the unique binary tree described by a provided in-order and post-order traversal. In the instructor's experience, most undergraduates taking CS2 can and do successfully solve a problem like this during exams.

While the system successfully drew a binary tree, even containing all the requisite letters, the tree itself was very incorrect. The system provided a tree where a top to bottom, left to right reading of the labels matched the required in-order traversal, but of course this is a very simplistic and incorrect understanding of the layout of binary trees.

Further adding to our surprise is the model's explanation. The *narrative* was largely accurate: it correctly stated that the label of the root can be determined from the post-order traversal, and that each subtree's contents can then be identified based on the other traversal. However, the values used in these were not correct. One researcher on the team described the response as one we might expect from a student who knows what a correct explanation to how to solve this problem *looks like* but did not know how to solve this particular instance and wanted to try for partial credit.

Unfortunately, hints did not lead the model to a solution. Even when told the label of the root, the model gave a tree consistent with that hint and the set of labels, but still was not better than random guessing.

### 4.2.2 *Context-Free Grammars.*

As stated previously, the model significantly struggled when dealing with context-free grammars (CFGs), in stark contrast to humans who tend to find such questions easier. The model answered almost every question, including relatively simple ones, entirely incorrectly. What we found interesting, however, was one problem where the model answered a majority of the question correctly, while it provided entirely irrelevant explanations.

In one example, we asked the model to generate the CFG for a language that consist of some number of 'a's, followed by 'b's, and finally followed by 'c's, with the following condition—the number of 'b's must match either with the number of 'a's or the number of 'c's. Stated formally, the language $L$ can be defined as: $L = \{a^i b^j c^k \mid i == j \text{ or } j == k\}$. We also provided the model with the same hint that we provide the students, namely that it should start by splitting off into two disjoint paths, one for each possibility (either the 'b's match the 'a's or the 'c's).

The model surprised us by not only correctly splitting the problem into two parts ($S \longrightarrow AB \mid CD$), but also by correctly accounting for the second possibility, when the 'b's match the 'c's. This involved first allowing any number of 'a's ($C \longrightarrow aC \mid \epsilon$) and then only allowing an equal amount of 'b's and 'c's ($D \longrightarrow bDc \mid \epsilon$). For the first possibility, the model also correctly restricted the number of 'a's to match the number of 'b's ($A \longrightarrow aAb \mid \epsilon$). Quizzingly, however, the model failed to correctly allow an unrestricted number of 'c's, instead allowing more 'b's ($B \longrightarrow bBc \mid \epsilon$), essentially allowing the case where $j == i + k$.

Despite pointing this out, the model is unable to correct this seemingly simple error, opting to either break more parts of the solution or assert that the solution is correct as-is. Numerous attempts to re-prompt yielded similar or worse results. These responses further suggest that while the model is excellent at understanding the structure of answers, problems that involve, even simple, novel reasoning [3] prove elusive to the model.

### 4.2.3 *Dominating Set is NP-complete.*

A subset of vertices of a graph are a Dominating Set if every vertex is either included in that set or adjacent to one that is. This contrasts with a Vertex Cover, in which we seek a set of vertices such that each edge of the graph has at least one of its endpoints in the set. If our graph has no isolated vertices, then a valid vertex cover is trivially a dominating set, but not every dominating set constitutes a vertex cover. In both cases, the decision version of the problem is to determine if a set of vertices of size $k$ exists for the metric.

In the context of the class, students see that Vertex Cover is NP-complete and, on their homework, are asked to prove that this is also true of Dominating Set. The system correctly began with a proof that Dominating Set is in NP. It then correctly gets the direction of the reduction: we begin with an instance of Vertex Cover, and want to create an equivalent instance of Dominating Set. However, the system gives us a poor reduction: adding a single vertex to the graph and then seeking a Dominating Set of size $k+1$ in the resulting (modified) graph. The system's response completes by attempting to demonstrate that this produces neither false positives nor false negatives.

Like the binary tree traversals problem earlier, this has a familiar feel to it: the system knows the *shape* of what an answer looks like, but cannot fill in any particulars.

## 5 Threats to Validity

One potential threat to the validity of our study is the potential presence of solutions to the problems we used in the LLM's training data. To address this concern, a replication or extension of our work could encompass similar problems from various universities, as well as attempts to obfuscate the problem without modifying its solution. This approach would lead to a more comprehensive evaluation of the model's generalization abilities, providing insights into its proficiency across a broader array of problem variations and sources. [15] noted this also.

As we were working on this project, new versions of GPT, as well as other large language models, were regularly released. When we began collecting data for data structures and algorithm questions, GPT-4 was the latest model, and we were using the API version with text response only. When we moved to Theory of Computation, GPT-4o was released, and we also started using 4o's ability to read PDF files and images. These updates could impact the types of problems the models are capable of solving, and the result of accuracy between two models does not mean one model is necessarily better than another. Due to time constraints and the limitations of request availability, our team could not revise all prior questions with the newer model. Given the continuous evolution of large language models, evaluating the performance of the latest model remains a valuable subject for future research and discussion.

Additionally, there is a potential issue regarding variability among graders, especially for problems involving proofs. The results might have differed if different individuals had graded a particular problem. This variability is similar to the desire for vertical grading consistency with respect to student artifacts.

A final issue is that the problems utilized may not be sufficient to completely represent the performance of the model on a given topic. We attempted to test multiple types of questions within each general area of study, but it could be the case that the model would perform differently on different kinds of problems regarding the same topic. To address this concern, a larger variety of questions could be tested, and the general topics could be subdivided into types of problems so that they can all be experimented with.

## 6 Future Work

Replication work is an important endeavor in science, as are extensions of existing work. The de-anonymized version of this paper will include a link to the problems we used. Another research group should consider not only replicating this work, but extending it with similar problems of their own.

One of the motivations for conducting evaluations of the capacity of LLMs to tackle classroom-level problems is the potential to lay the groundwork for an AI-powered tutoring system. Considering the initial achievements demonstrated by GPT-4 in resolving advanced algorithms problems, there is an appealing prospect of the development of an AI tutorial system for algorithmic topics.

Several LLMs offer the capability of fine-tuning the model to address specific problem types or adjusting model parameters, such as 'system' messages in the case of GPT-4 and GPT-4o. These fine-tuning approaches fall beyond the purview of this paper, as our focus here is on gauging the extent to which a naive student can leverage these tools to solve problems, under the assumption that a naive student is less likely to engage in extensive model tuning. Nonetheless, evaluating the performance of a more optimized model remains an intriguing research question in its own regard.

## 7 Conclusion

The foundational large language model (LLM) driving OpenAI's widely used chatbot, ChatGPT, has demonstrated an impressive aptitude in the realm of Computer Science Theory. Notably, even an unseasoned student can leverage GPT-4 to successfully tackle over 77% of problems across nine distinct theory topics, but we are not yet at the stage where even the most advanced GPT model can guarantee a passing grade in these classes. As time progresses, this performance is anticipated to enhance further, showcasing its potential to become a pivotal resource for tutoring computer science students with an unprecedented level of accuracy.

These findings underscore the profound implications of such capabilities. They extend to the realm of educational tools, where GPT's prowess can potentially transform the landscape of Computer Science Education. Additionally, the insights drawn from this study prompt an exploration of adapting courses in related disciplines to accommodate the possibility of students utilizing such tools for assistance in assignments where general computer access is available. Such a consideration could lead to a thoughtful reevaluation of assignment structures to ensure equitable assessment while embracing the benefits of AI-powered support.

## References

[1] Rishabh Balse, Bharath Valaboju, Shreya Singhal, Jayakrishnan Madathil Warriem, and Prajish Prasad. 2023. Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 292–298.

[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*. 500–506.

[3] François Chollet. 2019. On the Measure of Intelligence. arXiv:1911.01547 [cs.AI]

[4] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources. *arXiv preprint arXiv:2306.10509* (2023).

[5] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. Human vs Machine: Comparison of Student-generated and AI-generated Educational Content. *arXiv preprint arXiv:2306.10509* (2023).

[6] 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)*. Association for Computing Machinery, New York, NY, USA, 1136–1142. https://doi.org/10.1145/3545945.3569823

[7] 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. 2023. Computing Education in the Era of Generative AI. *Commun. ACM* 51, 1 (2023).

[8] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Robosourcing Educational Resources–Leveraging Large Language Models for Learnersourcing. In *The Proceedings of The First annual workshop on Learnersourcing: Student-generated Content @ Scale*.

[9] Giuseppe Destefanis, Silvia Bartolucci, and Marco Ortu. 2023. A Preliminary Analysis on the Code Generation Capabilities of GPT-3.5 and Bard AI Models for Java Functions. *arXiv preprint arXiv:2305.09402* (2023).

[10] Emma Enström and Viggo Kann. 2017. Iteratively intervening with the "most difficult" topics of an algorithms and complexity course. *ACM Transactions on Computing Education (TOCE)* 17, 1 (2017), 1–38.

[11] 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*. 10–19.

[12] 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*. 97–104.

[13] Flynn Fromont, Hiruna Jayamanne, and Paul Denny. 2023. Exploring the Difficulty of Faded Parsons Problems for Programming Education. In *Proceedings of the 25th Australasian Computing Education Conference*. 113–122.

[14] GitHub. 2022. Github Copilot - Your AI pair programmer. https://github.com/features/copilot.

[15] Matei A. Golesteanu, Garrett B. Vowinkel, and Ryan E. Dougherty. 2024. Can ChatGPT Pass a Theory of Computing Course?. In *Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual 2024)* (Virtual Event). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3649165.3690116

[16] 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. In *Proceedings of the 2023 Conference on International Computing Education Research*.

[17] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*.

[18] Tyson Kendon, Leanne Wu, and John Aycock. 2023. AI-Generated Code Not Considered Harmful. In *Proceedings of the 25th Western Canadian Conference on Computing Education*. 1–7.

[19] Sam Lau and Philip J Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile" : How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 Conference on International Computing Education Research*.

[20] 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. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 2.

[21] 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*. 563–569.

[22] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2023. Understanding the Usability of AI Programming Assistants. In *Proceedings of the ACM/IEEE 46th International Conference on Software Engineering (ICSE)*.

[23] Michael Luu, Matthew Ferland, Varun Nagaraj Rao, Arushi Arora, Randy Huynh, Frederick Reiber, Jennifer Wong-Ma, and Michael Shindler. 2023. What is an Algorithms Course? Survey Results of Introductory Undergraduate Algorithms Courses in the US. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 284–290.

[24] Stephen MacNeil, Joanne Kim, Juho Leinonen, Paul Denny, Seth Bernstein, Brett A Becker, Michel Wermelinger, Arto Hellas, Andrew Tran, Sami Sarsa, et al. 2023. The Implications of Large Language Models for CS Teachers and Students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 2.

[25] Kamil Malinka, Martin Peresíni, Anton Firc, Ondrej Hujnák, and Filip Janus. 2023. On the educational impact of ChatGPT: Is Artificial Intelligence ready to obtain a university degree?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 47–53.

[26] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. *arXiv preprint arXiv:2307.08177* (2023).

[27] OpenAI. 2021. OpenAI Codex. https://openai.com/blog/openai-codex.

[28] OpenAI. 2023. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt.

[29] Eng Lieh Ouh, Benjamin Kok Siew Gan, Kyong Jin Shim, and Swavek Wlodkowski. 2023. ChatGPT, Can You Generate Solutions for my Coding Exercises? An Evaluation on its Effectiveness in an undergraduate Java Programming Course. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*.

[30] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. 157–163.

[31] Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. *International Journal of Management* 21, 2 (2023), 100790.

[32] Stephen R Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel Payne, and Perry G Ridge. 2023. Many bioinformatics programming tasks can be automated with ChatGPT. *arXiv preprint arXiv:2303.13528* (2023).

[33] 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* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1063–1069. https://doi.org/10.1145/3626252.3630928

[34] 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*. 299–305.

[35] Charles Rich and Richard C Waters. 1990. *The programmer's apprentice*. ACM.

[36] Steven I Ross, Michael Muller, Fernando Martinez, Stephanie Houde, and Justin D Weisz. 2023. A Case Study in Engineering a Conversational Programming Assistant's Persona. In *Joint Proceedings of the ACM IUI Workshops 2023, March 2023, Sydney*.

[37] 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*. 27–43.

[38] 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. In *Proceedings of the 2023 Conference on International Computing Education Research*.

[39] Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. 2023. Can Generative Pre-trained Transformers (GPT) Pass Assessments in Higher Education Programming Courses?. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 2.

[40] Pragnya Sridhar, Aidan Doyle, Arav Agarwal, Christopher Bogart, Jaromir Savelka, and Majd Sakr. 2023. Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives. *arXiv preprint arXiv:2306.17459* (2023).

[41] Tamara Tate, Shayan Doroudi, Daniel Ritchie, Ying Xu, and Mark Warschauer. 2023. Educational research and AI-generated writing: Confronting the coming tsunami. (2023).

[42] Tianjia Wang, Daniel Vargas-Diaz, Chris Brown, and Yan Chen. 2023. Towards Adapting Computer Science Courses to AI Assistants' Capabilities. *Visual Languages / Human-Centric Computing* (2023).

[43] Nathaniel Weinman, Armando Fox, and Marti A Hearst. 2021. Improving instruction of programming patterns with faded parsons problems. In *Proceedings of the 2021 chi conference on human factors in computing systems*. 1–4.

[44] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 172–178.