Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses

Jaromir Savelka
Carnegie Mellon University
Pittsburgh, PA, USA
jsavelka@cs.cmu.edu

Arav Agarwal
Carnegie Mellon University
Pittsburgh, PA, USA
arava@andrew.cmu.edu

Marshall An
Carnegie Mellon University
Pittsburgh, PA, USA
haokanga@andrew.cmu.edu

Chris Bogart
Carnegie Mellon University
Pittsburgh, PA, USA
cbogart@andrew.cmu.edu

Majd Sakr
Carnegie Mellon University
Pittsburgh, PA, USA
msakr@cs.cmu.edu

ABSTRACT

This paper studies recent developments in large language models' (LLM) abilities to pass assessments in introductory and intermediate Python programming courses at the postsecondary level. The emergence of ChatGPT resulted in heated debates of its potential uses (e.g., exercise generation, code explanation) as well as misuses in programming classes (e.g., cheating). Recent studies show that while the technology performs surprisingly well on diverse sets of assessment instruments employed in typical programming classes the performance is usually not sufficient to pass the courses. The release of GPT-4 largely emphasized notable improvements in the capabilities related to handling assessments originally designed for human test-takers. This study is the necessary analysis in the context of this ongoing transition towards mature generative AI systems. Specifically, we report the performance of GPT-4, comparing it to the previous generations of GPT models, on three Python courses with assessments ranging from simple multiple-choice questions (no code involved) to complex programming projects with code bases distributed into multiple files (599 exercises overall). Additionally, we analyze the assessments that were not handled well by GPT-4 to understand the current limitations of the model, as well as its capabilities to leverage feedback provided by an autograder. We found that the GPT models evolved from completely failing the typical programming class' assessments (the original GPT-3) to confidently passing the courses with no human involvement (GPT-4). While we identified certain limitations in GPT-4's handling of MCQs and coding exercises, the rate of improvement across the recent generations of GPT models strongly suggests their potential to handle almost any type of assessment widely used in higher education programming courses. These findings could be leveraged by educators and institutions to adapt the design of programming assessments as well as to fuel the necessary discussions into how programming classes should be updated to reflect the recent technological developments. This study provides evidence that programming instructors need to prepare for a world in which there is an easy-to-use widely accessible technology that can be utilized by learners to collect passing scores, with no effort whatsoever, on what today counts as viable programming knowledge and skills assessments.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; *Student assessment*; • **Computing methodologies** → **Artificial intelligence**; *Natural language processing*.

KEYWORDS

AI code generation, introductory and intermediate programming, Multiple-choice question answering, MCQ, coding exercises, generative pre-trained transformers, GPT, Python course, programming knowledge assessment, ChatGPT, Codex, GitHub Copilot, AlphaCode

ACM Reference Format:
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 ACM Conference on International Computing Education Research V.1 (ICER '23 V1), August 07–11, 2023, Chicago, IL, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3568813.3600142

1 INTRODUCTION

Rapidly increasing capabilities of large language models (LLM) keep challenging established practices in various contexts, including computer science and information technology (CS/IT) education. There are important unanswered questions related to (i) curricular changes needed to accommodate the new reality, (ii) excessive learners' reliance on LLMs in engaging with learning materials, assignments, and assessments, as well as (iii) considerable uncertainty as to how the future of CS/IT profession(al)s look like. Hence, the all-important concern shared by many CS/IT educators as to what are the skills and knowledge the learners in CS/IT programs need in order to have successful and meaningful careers. Perhaps, a more immediate question that likely occupies minds of many instructors

ICER '23 V1, August 07–11, 2023, Chicago, IL, USA

ACM ISBN 978-1-4503-9976-0/23/08.
https://doi.org/10.1145/3568813.3600142

is how to assess learners' skills and knowledge in the presence of ubiquitous tools (e.g., ChatGPT,[1] GitHub Copilot[2]) that could be easily utilized to pass the assessments (at least partially).

While it is difficult to provide definitive or even satisfactory answers to questions posed above it is of utmost importance to build and maintain a body of empirically validated knowledge that would facilitate deep and meaningful discussions on these topics. Indeed, there has been a growing body of scholarship focused on understanding the capabilities of LLMs, as well as their limitations, in the context of programming education (see Section 2). The recent release of GPT-4 poses a challenge for the existing work that needs to be confirmed or updated to account for this seemingly more powerful technology. The issue is especially pressing when it comes to what we know about the capabilities of GPT models to handle assessments that were originally designed for a human test-taker. This is because GPT-4 appears to perform much better on academic and professional exams when compared to the preceding GPT-3.5 generation. The technical report [35] made available with the GPT-4 release lists 34 such exams, including various Graduate Record Examination (GRE) tests,[3] SAT Math,[4] or a Uniform Bar Exam.[5] Several of the listed exams involve programming tasks (e.g., Leetcode,[6] Codeforces Rating[7]) and those too show notably improved performance. Hence, it appears that the current knowledge of the capabilities of the GPT models to handle assessments in programming courses might be outdated.

This paper analyzes the capabilities of the newest state of the art generative pre-trained transformer (GPT-4) to pass typical assessments, i.e., multiple-choice question (MCQ) tests and coding exercises, in introductory and intermediate programming courses. The aim of this paper is to quickly react to the recent release of GPT-4 and assess if and to what extent do the findings presented by similar past studies performed with GPT-3 and 3.5 models still stand. Hence, the focus is not only on the performance of GPT-4 but also on the comparison of its performance to that of the earlier GPT models. To that end we employ a data set comprising of 599 assessments from three currently running Python courses. We asses the outputs of the GPT models as if they were coming from a human learner. This means that we also expose the models to a feedback generated by an auto-grader and provide them with an opportunity to iterate on the solution. This is all done in a manner ensuring that there is no human intervention that could contribute to models successfully passing the assessments. This approach allows us to accurately gauge if the automatically generated solutions would enable a human learner to pass a course provided it would have been their own work. The immediate insight that this study offers is that the danger of learners' over-reliance on GPT models when completing their programming courses' assignments and assessments is a real concern that has to be taken seriously.

To investigate if and how GPT-4 challenges the findings of the prior works related to the capabilities of LLMs to handle diverse types of assessments typically employed in real-world introductory and intermediate Python programming courses at the post-secondary level, we analyzed the following research questions from the prior work in light of the newly released model:

(RQ1) To what degree can GPT-4 generate correct answers to MCQs in order to pass an introductory or intermediate course in Python in higher education? [45, 46]
(RQ2) Does GPT-4 struggle with programming MCQs containing code snippets that require multi-hop reasoning? [45]
(RQ3) To what degree can GPT-4 produce solutions to complex coding tasks from instructions in order to pass an introductory or intermediate course in Python in higher education? [7, 10, 11, 38, 46]
(RQ4) Can GPT-4 successfully utilize feedback to fix solutions of coding tasks? [46]

By carrying out this work, we provide the following contributions to the CS education research community. To our best knowledge, this is the first comprehensive study that:

(C1) Measures performance of the GPT-4 model on diverse assessment instruments from Python programming courses, updating and extending the current body of knowledge that has been developed on experiments with GPT-3 models.
(C2) Offers a detailed in-depth analysis of common properties of MCQs and coding tasks that are answered incorrectly by GPT-4.

2 RELATED WORK

GPT-4. Given the recent arrival of GPT-4, there have been few studies of the implications of the new model in education as of the writing of this paper. OpenAI's technical report states performance of GPT-4 on numerous tasks across diverse domains. Of particular importance are the 92.0% success rate of GPT-4 on grade-school mathematics questions using 5-shot examples and chain-of-thought prompting, solving 31/41 Leetcode easy and 21/80 Leetcode medium exercises, and significant success across several quantitative AP and competitive mathematics exams [35]. Katz et al. demonstrate that GPT-4 achieves 297 points on the Uniform Bar Exam (UBE), passing the bar exam and, in the authors words "by a significant margin" [18]. Lastly, Jiao et al. consider the performance of GPT-4 on academic translation tasks, demonstrating that the ChatGPT service achieves significantly better performance compared to existing commercial translation products [16]. This paper falls in line with such work, conducting a rigorous evaluation of the recently-released GPT-4 model when applied to typical introductory and intermediate programming assessments. We demonstrate that the gains in performance observed in other domains extend to the programming education as well.

GPT Performance on Programming MCQs. Savelka et al. evaluated the capability of `text-davinci-003`, to pass a diverse set of assessment instruments, including MCQs, in the realistic context of full-fledged programming courses [46]. They found that the

[1] OpenAI: ChatGPT. Available at: https://chat.openai.com/ [Accessed 2023-03-20]
[2] GitHub Copilot: Your AI pair programmer. Available at: https://github.com/features/copilot [Accessed 2023-03-20]
[3] ETS: The GRE General Test. Available at: https://www.ets.org/gre/test-takers/general-test/about.html [Accessed 2023-03-20]
[4] SAT Suite of Assessments. Available at: https://satsuite.collegeboard.org/sat/whats-on-the-test/math [Accessed 2023-03-20]
[5] NCBE: Uniform Bar Examination. Available at: https://www.ncbex.org/exams/ube/ [Accessed 2023-03-20]
[6] LeetCode. Available at: https://leetcode.com/ [Accessed 2023-03-22]
[7] Codeforces. Available at: https://codeforces.com/contests [Accessed 2023-03-22]

then current GPT models were not capable of passing the full spectrum of assessments typically involved in a Python programming course (below 70% on even entry-level modules); but a straightforward application of these models could enable a learner to obtain a non-trivial portion of the overall available score (over 55%) in introductory and intermediate courses alike. They also observed that an important limitation of the GPT models was their apparent struggle with activities that required multi-hop reasoning, and that there appeared to be a difference in success rate between MCQs that contained a code snippet and those that did not [45, 46]. In this work, we re-examine those findings in the light of the more powerful model released since their publication. We find that the conclusions about the models not being able to pass the courses no longer hold. However, some of the limitations identified in the prior work still hold.

GPT Performance on MCQs in Other Domains. Robinson et al. apply InstructGPT [36] and Codex to OpenBookQA [31], StoryCloze [33], and RACE-m [23] data sets which focus on multi-hop reasoning, recall, and reading comprehension, reporting 77.4-89.2% accuracy [43]. In some cases, GPT can generate code when applied to programming assignments in higher education courses. Drori and Verma used Codex to write Python programs to solve 60 computational linear algebra MCQs, reporting 100% accuracy [9]. Others have used GPT models to solve various MCQ-based exams, including the United States Medical Licensing Examination (USMLE), with accuracy around 50% [13, 22, 27], the Multistate Bar Examination (MBE) [3, 18], and the American Institute of Certified Public Accountants' (AICPA) Regulation (REG) exam [2]. Although, GPT can often answer questions *about* systems and rules, it is especially challenged by tasks that involve *applying* them and reasoning about their implications in novel examples. Hendryks et al. created data set that includes a wide variety of MCQs across STEM, humanities and arts, with GPT-3 performing at levels above 50% for subjects such as marketing and foreign policy, but below 30% for topics such as formal logic [14]. They found that the model performed particularly poorly in quantitative subjects. For example, in Elementary Mathematics they note that GPT can answer questions *about* arithmetic order of operations (e.g., that multiplications are performed before additions), but it cannot correctly answer questions that require *applying* this concept. They also note that GPT performance is not necessarily correlated with how advanced the topic is for humans, doing better at College Mathematics than Elementary Mathematics. Finally, they noted that GPT does poorly on tests of legal and moral reasoning [14]. Lu et al. studied GPT models' performance on a large data set consisting of 21,208 MCQs on topics in natural science, social science, and language [28]. They prompted the models to produce an explanation along with its answer and reported 1-3% improvement in accuracy (74.04%). In this work, we do not adopt the approach and, hence, leave space for future work as it appears quite promising and definitely applicable in the context of programming MCQs.

GPT Performance on Coding Assessments. There is a growing body of related work on GPT models' capabilities in solving educational programming tasks by generating code and text. Finnie-Ansley et al. evaluated Codex on 23 programming tasks used as summative assessments in a CS1 [10] and CS2 [11] programming courses. Denny et al. focused on the effects of prompt engineering when applying Copilot to a set of 166 exercises from the publicly available CodeCheck repository [7]. Jalil et al. evaluated the performance of ChatGPT on content from five chapters of software testing curricula, reporting a 55.6% accuracy in their assessment [15]. Our paper extends the existing body of work, most importantly by using the more powerful GPT-4 model. Piccolo et al. used 184 programming exercises from an introductory bioinformatics course to evaluate the extent to which ChatGPT can successfully complete basic to moderate level programming tasks, reporting the success rate of 75.5% on the first attempt and 97.3% when provided with feedback [38]. Several studies focus on collaboration between a human learner and GPT-based assisting tools (e.g., Copilot). Kazemitabaar et al. studied learners using OpenAI's Codex during traditional code creation tasks, and demonstrated the use of Codex did not harm their performance, with experienced students performing significantly better [19]. Leinonen et al. used Codex to generate more readable error messages for learners to use for project-level debugging, suggesting that model-created explanations can serve as effective scaffolding for students learning to program [25]. Prather et al. examined how novices interact with these tools, observing that novices struggle to understand and use Copilot [39].

GPT and Computing Education. Besides the work focused on how well LLMs do in various tasks meant to be performed by human learners, there is also a growing body of work on using LLMs to support computing education. Sarsa et al. used `code-davinci-001` to generate 240 introductory programming exercises, along with tests, sample solutions and explanations, reporting that over 75% of the generated exercises were novel and suitable for use in a university setting [44]. Macneil et al. integrated LLM-generated code explanations into an interactive e-book and compared several different explanation types, such as line-by-line or summarization-oriented explanations, reporting students using line-by-line explanations the most despite them being considered the least useful according to the students [29]. Leinonen et al. compared learner-authored code explanations with those generated by GPT-3, showing that students perceived GPT-3 generated explanations as more readable of the two [24].

GPT Performance on Coding Tasks in Professional Settings. Outside of the educational context, there have been studies exploring GPT's capabilities on competitive and interview programming tasks. Chen et al. released the HumanEval data set where Codex achieved 28.8% success rate on the first attempt and 72.3% when allowed 100 attempts [6]. Li et al. report Deepmind's AlphaCode performance on Codeforces competitions,[8] achieving a 54.3% ranking amongst 5,000 participants [26]. Karmakar et al. reported 96% pass rate for Codex on a data set of 115 programming problems from HackerRank[9] [17]. Nguyen and Nadi reported Copilot's effectiveness on LeetCode[10] problems, achieving 42% accuracy [34]. Perry et al. explored the security implications of using Copilot [37].

Program code does more than control computer execution; it also, some argue primarily, serves as communication among developers [20]. Since GPT is a text prediction model trained on code in

[8] Codeforces. Available at: https://codeforces.com/contests [Accessed 2023-01-22]
[9] HackerRank. Available at: https://www.hackerrank.com/ [Accessed 2023-01-22]
[10] LeetCode. Available at: https://leetcode.com/ [Accessed 2023-01-22]

the context of human discussions about it, the model's representation of code is likely to capture code's *design intent* more strongly than code's *formal properties*. For example, work from multiple studies suggest that models that interpret code depend heavily on function names and input variables [32, 49]. Although, models like GPT are not trained to simulate code execution, they can in many cases generate code based on natural language description of the code's intent. Researchers have reported varying success at generating code in response to programming assignments, ranging from Codex's 100% success generating Python computational linear algebra programs [9], to 78.3% on some CS1 programming problems [10], to 79% on the CodeCheck[11] repository of Python programming problems [7].

Prompt Engineering. It has been well established that LLMs are few-shot learners, capable of answering questions without additional fine-tuning in a zero-shot fashion [4]. In general, finding the best prompt for a specific task is challenging, with prompts that are semantically similar sometimes providing large differences in performance [51]. Despite this difficulty, there have been several advancements in developing techniques for prompt engineering to improve the performance of LLMs. Numerous studies have explored prompts which include a number of examples to demonstrate what the desired output should be [4, 12, 50]. However, the current research literature remains inconclusive as to the efficacy of adding examples to natural language prompts, with multiple studies suggesting that the order and number of examples can dramatically influence the performance of LLMs across various tasks [42, 50].

In introductory CS context, there has been an inquiry into explainable prompt-engineering practices. Denny et al. explored prompting Copilot for CS1 exercises, demonstrating that while prompt engineering can significantly improve the performance of Copilot on CS1 problems, verbose prompts can lead to decreases in model performance [7]. Similar study has been performed on CS2 coding tasks [11].

More recently, there has been significant interest in chain of thought prompting, a technique where an LLM is asked to provide both the answer and the reasoning that lead to the answer in question. This has lead to significant performance gains in symbolic and quantitative reasoning tasks, by forcing the LLM to emulate human reasoning in addition to the answer itself [48]. Recently, researchers have also explored the so called "least-to-most" prompting, where a task is decomposed into several sub-problems, which are then answered all at once by the model [52].

3 DATA

For the purpose of this study, we obtained the data set that was originally used in [45, 46]. The researchers collected assessment exercises from three real-world currently running Python programming courses.

Python Essentials - Part 1 (Basics)[12] (**PE1**) transitions learners from a state of complete programming illiteracy to a level of programming knowledge which allows them to design, write, debug, and run Python programs. There are four units in the course, and one completion (summary) test. The units include:

(1) Introduction to Python and computer programming,
(2) Data types, variables, basic input-output operations and basic operators
(3) Boolean values, conditional loops, lists, logical and bitwise operators
(4) Functions, tuples, dictionaries and data processing.

PE1 employs MCQ assessments. Formative assessments are called quizzes and summative assessments are called tests. Qualitatively, the test MCQs appear to be considerably more challenging than quiz MCQs. The MCQs often include small snippets of code and ask learners to reason about them.

There are 149 questions in PE1. An MCQ may involve a snippet of Python code (*with code*) or it may be expressed fully in natural language (*no code*). For an MCQ, to be considered as *with code* there either is at least one line fully dedicated to computer code, and/or the choices are computer code expressions. Inline mentions of names of functions or variables were not considered as sufficient for an MCQ to be considered *with code*. Out of the 149 MCQs in PE1, 96 have code snippets. The MCQs are further distinguished into the following categories:

- *True/False* – The learner is asked to assess the truthfulness of a single statement.
- *Identify True/False Statement* – The learner is asked to pick one or more choices as either true or false.
- *Finish Statement.* – The learner is asked to complete a statement.
- *Output* – The learner is asked to identify the choice that corresponds to the output of a given snippet of code.
- *Fill-in Blanks* – The learner is asked to fill in a code snippet by selecting the appropriate choice as an answer.
- *Other* – Any MCQ that does not fall into any of the above categories.

Table 1 provides additional details, including the categorization of questions according to their type. Example questions for all the types are shown in Appendix A.

Python Essentials - Part 2 (Intermediate) (**PE2**)[13] covers advanced aspects of Python programming, such as modules, packages, exceptions, file processing, or object-oriented programming. Similarly to PE1, the course is organized into four units and it is also equipped with a completion (summary) test. The course units are:

(1) Modules, packages, and PIP
(2) Strings, String and List Methods, Exceptions
(3) Object-Oriented Programming
(4) Miscellaneous

Just like PE1, PE2 also employs MCQ assessments exclusively (quizzes and tests). There are 148 questions in PE2 out of which 83 have code snippets. Table 1 has additional details.

Finally, *Practical Programming with Python*[14] (**PPP**) centers around hands-on projects focused on fundamental Python constructs and

[11] CodeCheck: Python Exercises. Available at: https://horstmann.com/codecheck/python-questions.html [Accessed 2022-01-22]
[12] OpenEDG: Python Essentials - Part 1 (Basics). Available at: https://edube.org/study/pe1 [Accessed 2023-03-20]
[13] OpenEDG: Python Essentials - Part 2 (Intermediate). Available at: https://edube.org/study/pe2 [Accessed 2023-01-15]
[14] Sail(): Social and Interactive Learning Platform. Available at: https://sailplatform.org/courses. [Accessed 2023-03-20]

Table 1: MCQ Data Set. Each row provides information about the MCQ assessments each of the courses employ. Each column reports on the distribution of the MCQ types across the courses.

Course	Units	MCQ (no code)				MCQ (with code)						Course
	(topics)	T/F	Id. T/F	Fin. S.	Other	T/F	Id. T/F	Fin. S.	Out	Fill-in	Other	Overall
PE1	4	0	5	23	18	0	5	6	51	0	41	**149**
PE2	4	0	7	31	10	0	0	21	27	0	52	**148**
PPP	8	25	32	2	19	23	21	0	32	13	66	**233**
Type Overall	16	**25**	**44**	**56**	**47**	**23**	**26**	**27**	**110**	**13**	**159**	**530**

Table 2: Coding Activities Data Set. Each row provides information about each project's focus area(s) and the number of tasks and activities they contain. Fund. stands for Python language fundamentals; SW Dev. stands for software development practices (e.g., Test-driven development); Data stands for data processing and analysis (e.g., file formats, databases).

Project Topic	Fund.	SW Dev.	Data	Tasks	Acts.
Types, variables, functions	✓	✓		4	6
Iteration, conditionals, strings, basic I/O	✓			4	6
Lists, sets, tuples and dictionaries	✓		✓	2	11
Classes, objects, attributes and methods	✓	✓		6	8
Debugging, refactoring, testing and packaging		✓		6	15
Files and datastores			✓	3	7
Web scraping and office document processing			✓	4	7
Data analysis			✓	3	9
				32	**69**

exposure to software development tools, practices, and real-world applications. The course consists of eight units which include:

(1) Python basics and introduction to functions
(2) Control flow, strings, input and output
(3) Python data structures,
(4) Object-oriented programming
(5) Software development
(6) Data manipulation
(7) Web scraping and office document processing
(8) Data analysis

PPP also uses MCQs extensively. However, their influence on learners' passing the course is limited compared to PE1 and PE2. In PPP MCQs are used as inline gating activities meant as formative assessments and graded tests as summative assessments. The contribution of the tests to the overall grade would vary across the PPP offerings but it would rarely exceed 20%. There are 233 MCQs in PPP (144 with code snippets). Table 1 has additional details about MCQs in PPP.

In comparison to PE1 and PE2, PPP mostly employs the project-based education model [21]. Learners individually work on larger programming projects that are subdivided into tasks. The projects

```
[15 lines ...]
popular_books[book_id] = {                    ①
    'info': { [5 lines ...]
        'num_pages': row['num_pages'], [2 lines ...]
    },
    'rating': {
        'average_rating': row['average_rating'],
        'text_reviews_count': row['text_reviews_count']
[5 lines ...]
```

```
[FEEDBACK] The value of [...] book's `info` and `num_pages`    ②
keys [...] should be of type `<class 'int'>`. [...]
```

③

```
[22 lines ...]
'num_pages': int(row['num_pages']), [6 lines ...]
'average_rating': row['average_rating'],      ⑤
'text_reviews_count': int(row['text_reviews_count'])    ④
[5 lines ...]
```

```
[FEEDBACK] The value of [...] book's `rating` and `average_rating`
keys [...] should be of type `<class 'float'>`. [...]    ⑥
```

```
[29 lines ...]
'average_rating': float(row['average_rating']),    ⑦
[7 lines ...]
```

Figure 1: An example interaction with auto-grader which recognizes that the num_pages field (1) should not be of type str and produces corresponding feedback (2). After the flaw is corrected (3), as well as similar one (4) not mentioned in the feedback, another issue with the average_rating field (5) is fixed (7) based on the additional feedback (6).

require sustained effort that often extends over several days, depending on the proficiency of the learner. All the eight projects are auto-graded. The auto-grader provides learners with feedback that can be utilized for improving the solutions until a project deadline. The score from the projects, tests, and reflections (discussion posts) determines if a learner successfully completes the course. The projects typically contribute around 80% towards the grade. There are 69 coding activities in PPP (elements of the 32 project tasks). Further details about the projects and their coding activities are reported in Table 2.

Each project activity is associated with one or more assessments which are high-level rules that need to be met in order for a learner to be awarded with score points. For example, an assessment could require the output JSON file to have specific fields in terms of their names and data types. The auto-grader then uses an extensive battery of detailed tests to ensure the high-level assessment rule is met. The test cases are dynamically generated during the evaluation of

each submission. In case one or more tests fail, the learner receives a feedback the aim of which is to (1) clearly explain as to why the assessment rule is not met, and, hence, the score cannot be awarded, and (2) provide a hint on how to iterate on the solution towards a successful outcome. The feedback would typically not provide an extensive enumeration of the failed test cases. Instead, it often focuses on the most prominent one or the first one encountered. The feedback usually does not expose the exact nature of the test. While the feedback varies greatly across the activities the most common pattern is the contrast between the expectation and the actual state of the submitted solution. In most of the cases, the focus of the auto-grader is on the correctness of the solution. However, there are several activities focusing on code style and quality. In those activities, the grader goes beyond correctness and evaluates the compliance of the submitted solution. Figure 1 shows an example interaction with the auto-grader, illustrating how the feedback facilitates iterative improvement of the solution.

4 EXPERIMENTAL DESIGN

4.1 Models

The original GPT model [40] is a 12-layer decoder-only transformer [47] with masked self-attention heads. Its core capability is fine-tuning on a downstream task. The GPT-2 model [41] largely follows the details of the original GPT model with a few modifications, such as layer normalization moved to the input of each sub-block, additional layer-normalization after the first self-attention block, and a modified initialization. Compared to the original model it displays remarkable multi-task learning capabilities [41]. The third generation of GPT models [4] uses almost the same architecture as GPT-2. The only difference is that it alternates dense and locally banded sparse attention patterns in the layers of the transformer. The main focus of Brown et al. was to study the dependence of performance and model size where eight differently sized models were trained (from 125 million to 175 billion parameters). The largest of the models is commonly referred to as GPT-3. The interesting property of these models is that they appear to be very strong zero- and few-shot learners. This ability appears to improve with the increasing size of the model [4]. The technical details about the recently released GPT-4 model have not been disclosed due to (alleged) concerns about potential misuses of the technology as well as highly competitive market with generative AI [35].

We are primarily interested in the performance of the `gpt-4` (GPT-4) model as compared to `text-davinci-003` (GPT-3.5). As of writing of this paper, GPT-4 is by far the most advanced model released by OpenAI. The model is focused on dialog between a user and a system. On the other hand, GPT-3.5 is a more general model focused on text completion. It builds on top of previous `text-davinci-002`, which in turn is based on `code-davinci-002` (focused on code-completion tasks) which is sometimes referred to as codex. To gauge the rate of improvement over the several recent years, we compare the performance of GPT-4 to GPT-3.5 as well as to the previous generation's InstructGPT `text-davinci-001` model (GPT-3)[15] on the MCQ answering task. For coding exercises, we benchmark GPT-4 to GPT-3.5 only. This is because GPT-3 is mostly focused on text completion, and is not capable of producing (decent) solutions to coding exercises; this ability only emerged with `code-davinci-002` and later models.

We set the `temperature` of all the models to 0.0, which corresponds to no randomness. The higher the `temperature` the more creative the output but it can also be less factual. As the temperature approaches 0.0, the model becomes more deterministic, which we deem as important for reproducibility. Given that existing literature does use different temperatures for testing, we did initially test a variety of temperatures, but found that setting temperature to 0.0 worked well for our setting, which falls inline with the findings and precedence of existing work regarding multiple-choice questions [2, 27, 28]. Additionally, given that we were largely evaluating questions automatically a single-time per question, setting temperature to 0.0 provided us with the most likely completion of GPT-4, allowing us to be more confident in our resulting analysis. We set `max_tokens` to 500 (a token roughly corresponds to a word) for MCQ answering, and to 2,000 (GPT-3.5) or 4,000 (GPT-4) for coding activities. This parameter controls the maximum length of the completion (i.e., the output). Note that each model has a length limit on the prompt, and the completion counts towards that limit. While GPT-4 allows for 8,192 tokens[16] the GPT-3.5 can only accept up to 4,097 tokens. We set `top_p` to 1, as is recommended when `temperature` is set to 0.0. This parameter is related to `temperature` and also influences creativeness of the output. We set `frequency_penalty` to 0, which allows repetition by ensuring no penalty is applied to repetitions. Finally, we set `presence_penalty` to 0, ensuring no penalty is applied to tokens appearing multiple times in the output.

4.2 Experimental Design

To test the performance on MCQs, we submit questions one by one using the `openai` Python library[17] which is a wrapper for the OpenAI's REST API. For GPT-3 and GPT-3.5, we embed each question in the prompt templates shown in Figure 2. Since GPT-4 is a model optimized for dialogue, we use different prompts—the ones shown in Figure 3. Note that the prompt for GPT-4 is designed with the intent to prevent the model from explaining the answer to a user as we are only interested in the answer(s) themselves. Each model returns one or more of the choices as the prompt completion (response), which is then compared to the reference answer. Following the approach adopted by PE1 and PE2, partially correct answers are considered to be incorrect.

In coding tasks, we submit the instructions using the prompt templates shown in Figure 4 for GPT-3.5. Again, we use a different prompt for GPT-4 which is shown in Figure 5. While we needed to embed the coding activity instructions into the GPT-3.5's prompt (as shown in Figure 4) these are passed to GPT-4 more naturally as a message coming from a user.

To each submission, the auto-grader assigns a score and generates detailed actionable feedback. If the full score was not achieved we amended the GPT-3.5's prompt with the addendum (shown in Figure 4). For GPT-4 we simply continued in the dialogue where the

[15] OpenAI: Model index for researchers. Available at: https://beta.openai.com/docs/model-index-for-researchers/instructgpt-models [Accessed 2023-01-15]

[16] There is also a variant of the model that supports up to 32,768 tokens.

[17] GitHub: OpenAI Python Library. Available at: https://github.com/openai/openai-python [Accessed 2023-01-16]

```
I am a highly intelligent bot that can easily handle answering
multiple-choice questions on introductory Python topics.        ①
Given a question and choices I can always pick the right ones.

Question: {{question}}    ②

Choices:
{{choices}}    ③

The correct answer:
```

Figure 2: MCQ Prompt Template for GPT-3 and GPT-3.5. The text of the preamble (1) is inspired by OpenAI's QA example. The {{question}} token (2) is replaced with the question text. The {{choices}} token (3) is replaced with the candidate answers where each one is placed on a single line preceded by a capital letter.

```
You are a highly intelligent bot that can easily handle answering
multiple-choice questions on introductory Python topics. Given a
question and choices you can always pick the right ones. You are
not expected to explain the answers.  ①

Example user question:
What function in Python is typically used to display text to the
terminal?
A. input
B. print
C. len      ②
D. int

Example bot response:
B. print
```

```
Question: {{question}}  ③

Choices:
{{choices}}  ④
```

Figure 3: MCQ Prompt Templates for GPT-4. The outer frame shows the system's prompt which is used to set the context of the dialogue. The text of the preamble (1) is inspired by OpenAI's QA example. The example user question and bot response (2) primes the model to return the answer(s) only (no explanations). The inner frame depicts the user's message sent to the dialogue system. The {{question}} token (3) is replaced with the question text. The {{choices}} token (4) is replaced with the candidate answers where each one is placed on a single line preceded by a capital letter.

solution it generated was followed by the auto-grader's feedback. Then, we submitted the revised solution to the auto-grader and repeated the process until either the full score was achieved or the solution remained unchanged from the preceding one (impasse).

When dealing with coding activities, we encountered the models' limitation on the prompt length (4,097 tokens for GPT-3.5 and 8,192 or 32,768 for GPT-4). Within this limit, it was necessary to fit: (i) the prompt boilerplate; (ii) the instructions; (iii) the contents of the handout files (usually starter code) distributed to learners; and (iv) the solution generated by the model (i.e., the prompt completion). Instead of full project (8) instructions we submitted the individual project tasks (32). If a task could not be fitted into a prompt, we decreased the `max_tokens` parameter (space for solution) to <2,000 for GPT-3.5 or <4,000 for GPT-4. If this did not resolve the issue we edited the instructions leaving out pieces that could be reasonably expected as not being useful for the GPT models. As the last resort,

```
TASK
Implement a Python program to print "Hello, World!" in hello.py.
=== hello.py ===
# TODO 1
===              ①

SOLUTION
=== hello.py ===
print("Hello, World!")
===

TASK
{{instructions}}  ②
=== {{file_name}} ===  ③
{{handout}}  ④
===

SOLUTION
```

Figure 4: Coding Task Prompt Templates for GPT-3.5. Outer frame is the first submission template. The preamble (1) primes the model to generate a solution code as completion. The {{instructions}} token (2) is replaced with the coding task instructions. A starter code is injected into {{file_name}} (3) and {{handout}} (4) tokens. The inner frame shows the template for re-submission (appended to the original). The {{file_name}} (5) and {{solution}} (6) were replaced with the GPT's solution and the {{feedback}} (7) with the auto-grader's feedback.

```
You are a highly intelligent coding bot that can easily handle any
Python programming task. Given a natural language instructions you
can always implement the correct Python solution. Your focus is the
solution code only. You are not allowed to provide explanations.
                                                          ①
Example (toy) instructions:
Implement a Python program to print "Hello, World!" in the  ②
hello.py.

Example bot solution:
=== hello.py ===        ③
print("Hello, World!")
===
```

Figure 5: Coding Task Prompt Template for GPT-4. The preamble (1) primes the model to generate the code of the solution only (no explanations). The example instructions (2) and solution (3) are used to further clarify the expectations on the output.

we would split the task into several smaller coding activities (69 overall) if the task was to develop several loosely coupled elements. The GPT models' solutions were then submitted to the auto-grader.

5 RESULTS AND DISCUSSION

5.1 (RQ1) To what degree can GPT-4 generate correct answers to MCQs?

The results of applying the GPT models to the MCQ exercises are reported in Tables 3 (PE1), 4 (PE2), and 5 (PE3). While the original GPT-3 model correctly answered only 199 out of the 530 questions (37.5%), the GPT-3.5 and GPT-4 models were much more successful. GPT-3.5 correctly answered 341 MCQs (64.3%). GPT-4

successfully handled 446 questions (84.1%). Hence, we observe a sizeable improvements across the successive generations of the GPT models.

The results from PE1 are reported in Table 3. In order to pass the course the score of 70% or better is required from all the 5 tests. While the GPT-3 model could not pass any of the tests, and the more successful GPT-3.5 model passed only the first course module, as already reported in prior work [45, 46], the GPT-4 model passed all the four module tests as well as the summary test (i.e., passing the course with the overall score of 85%).

The performance of the models on PE2 is presented in Table 4. The assessment scheme of PE2 is the same as that of PE1. Here, the GPT-3.5 model was somewhat more successful and passed 3/4 module tests. However, it also failed the Summary Test (65.0%). The original GPT-3 model still could not pass a single test. These findings were also reported in prior work [45, 46] The GPT-4 model again passed all the five graded assignments (89.6% overall score).

Table 5 reports the results of applying the models to the MCQs in PPP. Again, we observe similar progression from the weakest GPT-3 model (30.9% on the tests), through the better performing GPT-3.5 (65.4%), as already reported in prior work [45, 46], to the best performing GPT-4 model (77.8%). Note, that passing of PPP is not solely determined by the MCQ assessments, and largely depends on the performance on the coding activities (projects).

Overall, the findings reported in prior work [45, 46] no longer hold. While GPT-3 and GPT-3.5 models' performance on the programming MCQs is not sufficient for passing the three courses, GPT-4 handles the MCQs well enough to reliably pass the course MCQ assessments. Note, that in some countries much lower passing scores may be required. Hence, our finding of GPT-4 passing the assessments the prior models fail to pass might not hold in those contexts.

5.2 (RQ2) Does GPT-4 struggle with programming MCQs containing code?

Table 6 reports the results of our experiments on how GPT models handle MCQs of various types. The GPT-4 model performs the best (84.5% overall) with quite a noticeable margin over the GPT-3.5 (65.5% overall). The performance of the original GPT-3 appears to be much lower compared to the other two models. This is to be expected, as the major breakthrough in OpenAI GPT models' capabilities in handling computer code was Codex (`code-davinci-002`) [6] which is the predecessor of `text-davinci-003` (i.e., GPT-3.5).[18]

There appears to be a clear difference between the performance of the most capable GPT-4 on the MCQs that contain code snippets (81.0% overall) compared to those that do not (90.7% overall). This is to be expected as the combination of code and natural language likely constitutes (on average) more complex input than natural language alone. Additionally, it is quite possible that in our particular context the questions with code are (on average) more difficult than questions with no code. However, notice that the gap appears to be much wider in the preceding generations of the GPT models (29.9% vs 53.3% for GPT-3 and 59.5% vs 77.9% for GPT-3.5). Hence, it appears that GPT-4's capabilities in handling MCQs with code are much improved compared to its predecessors. However, the observable difference between the performance on MCQs with natural language only and MCQs with code remains. There also appears to be clear difference between the performance of GPT-4 on the completion-oriented MCQs (96.9%), i.e., *Finish Statement* and *Fill-in* and the rest (81.3%). Since GPT models are primarily focused on prompt completion, be it text or computer code, this finding is also as expected. Hence, the findings from prior work [45] still hold in this regard.

To investigate further GPT-4's code handling limitations, we analyzed the 67 MCQs with code that GPT-4 answered incorrectly, manually inspecting the full answers, and sometimes altering the prompt and requerying, to hypothesize the reasons for the errors. We found that the model's mistakes fell into five main categories, listed in Table 7.

Problems in the question (WRONG-Q) GPT's explanations in some cases exposed problems in the original questions. For example, one question asks, in part, "Which of the potential solutions would load the data into a `DataFrame` object where the index is set to the `month` column?". The correct answer was "`df = pd.read_csv("tickets_monthly.csv", index_col=0)`", using the Pandas package to load a CSV file into a variable `df`. However GPT-4, answered "`pd.read_csv( "tickets_monthly.csv", index_col=0)`", explaining that the actual correct answer "... is correct in terms of functionality, but it assigns the `DataFrame` to a variable `df`. The question asked for a solution that loads the data into a `DataFrame` object, not a variable assignment." Note that the answer preferred byt the GPT-4 model would be defensible since it does in fact load a `DataFrame`, albeit in an impractical way. Realistically, we might have granted a student the point if they provided similar explanation . Also note that GPT-4's explanation is not completely correct. The correct answer *does* load a `DataFrame` as well as assign it. It is then the better answer of the two (because more pragmatic).

Multi-hop reasoning (MULTI-HOP) GPT is known to have difficulty with answering questions that require multiple hops of reasoning [48]. Adding an instruction to show the reasoning steps sometimes improves the answers, or even leads to revising a wrong answer after explaining it. We re-queried GPT4 for some questions that appeared to use multi-hop reasoning, changing the prompt to encourage explanation (we removed the prompt text discouraging explanations, and added, "Give your answer, then provide your reasoning:" at the end of the prompt.) This indeed corrected some, but not all, of the errors. Interestingly, even when GPT-4 lists out the steps, its analysis of a step may be incorrect in the context of a complex, multi-hop answer, even though it is correct when answering a similar question in isolation. Sometimes this mimics *motivated reasoning*. For example, one question involved predicting the effect of two apparently opposite string replacement commands:

```
quote = '"The things that make me different are the ' +\
        'things that make me, me."'
new_sentence = quote.replace(" different", ", me")
new_sentence = new_sentence.replace(", me",
                                    " different")
```

Although the second replacement does not restore the original sentence, GPT's step claims that it does:

[18] OpenAI: Model index for researchers. Available at: https://beta.openai.com/docs/model-index-for-researchers/instructgpt-models [Accessed 2023-01-15]

Table 3: PE1 results. The graded assignments are colored; green and check mark indicate passing while red means failing.

	Quizzes			Tests		
Module Topic	GPT-3	GPT-3.5	GPT-4	GPT-3	GPT-3.5	GPT-4
Introduction to Python and programming	8/10 (80.0%)	10/10 (100%)	10/10 (100%)	6/10 (60.0%)	✓ 9/10 (90.0%)	✓ 10/10 (100%)
Data types, variables, I/O, operators	6/10 (60.0%)	10/10 (100%)	10/10 (100%)	6/20 (30.0%)	10/20 (50.0%)	✓ 18/20 (90.0%)
Booleans, conditionals, loops, operators	3/10 (30.0%)	7/10 (70.0%)	10/10 (100%)	6/20 (30.0%)	12/20 (60.0%)	✓ 16/20 (80.0%)
Functions, data structures, exceptions	6/12 (50.0%)	9/12 (75.0%)	9/12 (75.0%)	7/22 (31.8%)	12/22 (54.5%)	✓ 20/22 (90.9%)
Completion (Summary Test)	-	-	-	7/35 (20.0%)	17/35 (48.6%)	✓ 27/35 (77.1%)
Course Total	**23/42 (54.8%)**	**36/42 (85.7%)**	**39/42 (92.4%)**	**32/107 (29.9%)**	**60/107 (56.1%)**	**91/107 (85.0%)**

Table 4: PE2 results. The graded assignments are colored; green and check mark indicates passing while red means failing.

	Quizzes			Tests		
Module Topic	GPT-3	GPT-3.5	GPT-4	GPT-3	GPT-3.5	GPT-4
Modules, packages, and PIP	3/10 (30.0%)	6/10 (60.0%)	10/10 (100%)	10/18 (55.6%)	✓ 14/18 (77.8%)	✓ 17/18 (94.4%)
Strings, string list methods, exceptions	7/10 (70.0%)	6/10 (60.0%)	9/10 (90.0%)	4/15 (26.7%)	✓ 11/15 (73.3%)	✓ 13/15 (86.7%)
Object-oriented programming	7/10 (70.0%)	8/10 (80.0%)	9/10 (90.0%)	4/17 (23.5%)	✓ 12/17 (70.6%)	✓ 15/17 (82.4%)
Miscellaneous	8/12 (66.7%)	9/12 (75.0%)	11/12 (91.7%)	4/16 (25.0%)	9/16 (56.2%)	✓ 15/16 (93.8%)
Completion (Summary Test)	-	-	-	11/40 (27.5%)	26/40 (65.0%)	✓ 35/40 (87.5%)
Course Total	**25/42 (59.5%)**	**29/42 (69.0%)**	**40/42 (95.2%)**	**33/106 (31.1%)**	**72/106 (67.9%)**	**95/106 (89.6%)**

Table 5: PPP results. The tests contribute to the grade, typically by no more than 20%. Since in PPP tests themselves do not determine pass or fail no colors are used.

	Quizzes			Tests		
Module Topic	GPT-3	GPT-3.5	GPT-4	GPT-3	GPT-3.5	GPT-4
Python basics and introduction to functions	12/30 (40.0%)	21/30 (70.0%)	27/30 (90.0%)	4/12 (33.3%)	9/12 (75.0%)	10/12 (83.3%)
Control flow, strings, input and output	8/22 (36.4%)	10/22 (45.5%)	16/22 (72.7%)	3/11 (27.3%)	8/11 (72.7%)	10/11 (90.9%)
Python data structures	9/18 (50.0%)	10/18 (55.6%)	14/18 (77.8%)	4/14 (28.6%)	9/14 (64.3%)	10/14 (71.4%)
Object-oriented programming	6/14 (42.9%)	7/14 (50.0%)	11/14 (77.6%)	4/11 (36.4%)	10/11 (90.9%)	11/11 (100%)
Software development	9/19 (47.4%)	12/19 (63.2%)	16/19 (84.2%)	5/10 (50.0%)	7/10 (70.0%)	10/10 (100%)
Data manipulation	6/17 (35.3%)	9/17 (52.9%)	13/17 (76.5%)	5/13 (38.5%)	5/13 (35.5%)	8/13 (61.5%)
Web scraping and office document processing	5/10 (50.0%)	5/10 (50.0%)	5/10 (50.0%)	0/5 (0.0%)	3/5 (60.0%)	3/5 (60.0%)
Data analysis	6/22 (27.3%)	17/22 (77.3%)	18/22 (81.8%)	0/5 (0.0%)	2/5 (40.0%)	2/5 (20.0%)
Course Total	**61/152 (40.1%)**	**91/152 (59.9%)**	**120/152 (78.9%)**	**25/81 (30.9%)**	**53/81 (65.4%)**	**64/81 (79.0%)**

> The second `replace()` function call replaces the first occurrence of ", me" with " different". The new string becomes: "The things that make me different are the things that make me, me."

This is not true as the call to the `replace` function replaces all matching strings. GPT-4 explains this aspect of the question correctly when asked in isolation:

> This code will output a modified version of the quote string where every occurrence of the substring ", me" is replaced with the string " different". [...] "The things that make me different are the things that make me different."

Another commonly occurring failure due to multi-hop reasoning seems to be related to interference between subparts of a question. For example, one question defines a function called `mystery` that does a series of edits and permutations on a list. The choices list four calls to this function, each with a proposed output, and asks the student to identify which of the pairs is correct. Unlike the true "multi-hop" questions, these do not build on each other, but they nonetheless seem to interact. GPT-4 can explain the function in isolation and identify the correct result. However, when the four choices are posed together for verification, it identifies all four of them as correct, even though only one is in fact correct.

Unlikely failure (FLUKE) For MCQs, we set `temperature` to 0 as we were explicitly focused on running a reproducible experiment, capturing the performance of the most likely output of the model.

Table 6: Performance of the GPT models across MCQs of different types.

Question Type	GPT-3	GPT-3.5	GPT-4
No Code			
True/False	13/25	20/25	23/25
	(52.0%)	(80.0%)	(92.0%)
Identify True/False Statement	12/44	27/44	35/44
	(27.3%)	(61.4%)	(79.5%)
Finish Statement	42/56	50/56	56/56
	(75.0%)	(89.3%)	(1.0%)
Other	25/47	37/47	43/47
	(53.2%)	(78.7%)	(91.5%)
Total	**92/172**	**134/172**	**157/172**
	(53.5%)	**(77.9%)**	**(91.3%)**
With Code			
True/False	12/23	10/23	13/23
	(52.2%)	(43.5%)	(56.5%)
Identify True/False Statement	10/26	11/26	16/26
	(38.5%)	(42.3%)	(61.5%)
Output	28/110	53/110	86/110
	(25.4%)	(48.2%)	(78.2%)
Fill-in	5/13	11/13	12/13
	(38.5%)	(84.6%)	(92.3%)
Finish Statement	10/27	22/27	25/27
	(37.0%)	(81.5%)	(92.6%)
Other	42/159	106/159	139/159
	(26.4%)	(66.7%)	(87.4%)
Total	**107/358**	**213/358**	**291/358**
	(29.9%)	**(59.5%)**	**(81.3%)**
Overall	199/530	347/530	448/530
	(37.5%)	(65.5%)	(84.5%)

Table 7: Qualitative coding of wrong GPT-4 code MCQ answers

Code	Definition	# Q
WRONG-Q	GPT's explanation revealed a valid interpretation suggesting that the question itself was ambiguous	13
MULTI-HOP	Model was confused by multi-hop reasoning.	19
FLUKE	GPT only gets the question wrong when temperature=0	8
DUAL-INTERPRETATION	The text of the code did not match its intent, and GPT sometimes relied on the intent	13
INCOHERENT	GPT answers are variable and inconsistent	14

Setting a higher `temperature` and taking the most common of several answers could be a way to achieve more reliably correct performance. We marked as FLUKE any questions that GPT-4 got right with `temperature` being set higher. This is because in this case the incorrect handling is associated with a particular parametrization of the model.

Reasoning biased by inferred code intent (DUAL-INTERP) GPT-4 sometimes provided answers that focused on the intent rather than the exact nature of the code itself. For example, in one question, a calculation is performed but the last line, printing out the result, is commented out. GPT-4 answered as if the line were not commented out. If asked to explain the reasoning step-by-step, GPT-4 sometimes caught its mistake. GPT-4 is optimized for dialog, in which humans do the best to make sense of inconsistent inputs; we infer the most plausible coherent interpretation of an interlocutor. A debugging-focused question that asks about unintended behavior from subtly wrong code is interpreted as if the code were "correct". The same robust ability to interpret intent that lets it answer poorly-worded English questions apparently trips it up in questions about purposefully misleading code.

Inconsistent reasoning (INCOHERENT) GPT-4 sometimes gives different answers when queried repeatedly with `temperature > 0`. When providing reasons, it may give contradictory answers within the same response. For example, one question asked if the expression `not (mag < 5)` would be correct in a program with multiple blanks, at a point where values >= 6 have already been ruled out. GPT-4 incorrectly responds "The suggested expression `not (mag < 5)` is not accurate because it will also include magnitudes of 6.0 and higher." However, it then gives the completed code, using the logically equivalent expression `mag > 5.0`. It explains that "should be mag >= 5.0, as this accurately identifies the `"Moderate"` category without including higher magnitude levels." Other responses contained similar contradictions.

5.3 (RQ3) To what degree can GPT-4 produce solutions to complex coding tasks?

The results of applying the GPT models to the coding activities are reported in Table 8. While the GPT-3.5 model obtained 407 points from the available 760 (53.6%) after a single submission to each activity GPT-4 collected 545 points (71.7%). While there is no stable grading scheme for PPP it is fair to anticipate that the performance of GPT-3.5 on MCQ tests (65.4%) and projects (54.6%) would typically not be deemed sufficient for passing the course. On the other hand, the performance of GPT-4, i.e., 79% on the MCQ tests and 71.7% on the projects, comes dangerously close if not all the way towards actually passing the course. Hence, the findings reported in [46] appear to be challenged by the more capable GPT-4 model as well as the findings reported in [7, 10, 11, 38].

We observe that the performance of GPT-4 across the tasks appears to be related to the performance of GPT-3.5. That is to say, the tasks that were challenging for GPT-3.5 appear to be challenging even for GPT-4. GPT-3.5 collected very low scores on the first submission from projects 2 (40 points), 5 (4 points), 7 (20 points), and 8 (11 points). While GPT-4 obtained somewhat higher scores (56, 14, 53 and 44 points respectively) these four projects remained the most challenging. Hence, it appears that the workings of GPT-3.5 and GPT-4 with respect to the coding tasks are not fundamentally different, despite GPT-4 being noticeably better performing. Hence, the strengths and limitations reported in prior work [7, 10, 11, 38, 46] still hold to a certain degree.

Table 8: Coding tasks results. Max score is the maximum score achievable from a task. First score is the score after first submission. Resubs is the number of re-submissions after the first submission before the full score or no-change impasse were reached. Final score is the score after feedback.

			GPT-3.5			GPT-4		
Project Topic	Tasks (skills)	Max	1st	Resubs	Final	1st	Resubs	Final
Types, variables, functions	Variable assignment	13	13	0	13	13	0	13
	User-defined functions, imports, return vs print	20	16	1	16	20	0	20
	Simple scripts, comments	50	50	0	50	50	0	50
	Testing	12	12	0	12	12	0	12
	Project 1 Total	**95**	**91**	**1**	**91**	**95**	**0**	**95**
Iteration, conditionals, strings, basic I/O	Conditional statements	10	10	0	10	10	0	10
	Strings, while loop, for loop, complex printing	35	5	4	9	6	11	18
	Read and write files	15	0	2	3	15	0	15
	Complex script	35	25	1	25	25	1	25
	Project 2 Total	**95**	**40**	**7**	**47**	**56**	**12**	**68**
Lists, sets, tuples and dictionaries	Create container, access, add, remove and update elements, and convert containers across types	40	38	1	40	40	0	40
	Nested data structure, transformation, sorting, export to file, complex report	55	30	3	40	55	0	55
	Project 3 Total	**95**	**68**	**4**	**80**	**95**	**0**	**95**
Classes, objects, attributes and methods	Implement classes	13	13	0	13	13	0	13
	Define, access and set private attributes	19	19	0	19	19	0	19
	Inheritance	20	20	0	20	20	0	20
	Implement re-usable utility functions	16	16	0	16	16	0	16
	Composition, object instantiation	16	16	0	16	16	0	16
	Override special methods (repr, eq)	11	0	3	0	9	1	11
	Project 4 Total	**95**	**84**	**3**	**84**	**93**	**1**	**95**
Debugging, refactoring, testing and packaging	Identify and fix errors in code	10	0	2	0	5	2	5
	Refactor larger code base	14	-	-	-	-	-	-
	Exception handling	13	4	2	4	4	2	13
	Analyze and fix code on style correctness	28	0	10	0	0	23	14
	Test-driven development	20	0	5	5	5	7	10
	Package Python application using pip	10	-	-	-	-	-	-
	Project 5 Total	**95**	**4**	**19**	**9**	**14**	**34**	**42**
Files and datastores	Load and store data in files	45	30	3	45	45	0	45
	Create SQL objects, load and query data in SQL	30	30	0	30	30	0	30
	Load and query data in MongoDB	20	20	0	20	20	0	20
	Project 6 Total	**95**	**80**	**3**	**95**	**95**	**0**	**95**
Web scraping and office document processing	Get HTML, extract information from HTML, handle multiple HTML files	35	5	10	18	33	1	35
	Manipulate Excel files programatically	25	5	4	5	5	4	5
	Authenticate and utilize public API	15	0	1	0	15	0	15
	Manipulate Word files programmatically	20	10	3	20	0	1	20
	Project 7 Total	**95**	**20**	**18**	**43**	**53**	**6**	**75**
Data analysis	Load data to pandas, merge pandas DataFrames, persist pandas DataFrame	35	11	3	11	24	2	24
	Assess data quality, examine descriptive statistics	40	0	3	25	0	1	25
	Utilize regular expressions	20	0	1	20	20	0	20
	Project 8 Total	**95**	**11**	**7**	**56**	**44**	**3**	**69**
	Course Total	**760**	**407**	**59**	**505**	**545**	**56**	**634**
			53.6%		**66.4%**	**71.7%**		**83.4%**

5.4 (RQ4) Can GPT-4 successfully utilize feedback to fix solutions of coding tasks?

The coding tasks results after providing the models with feedback are also reported in Table 8. The overall score achieved by the GPT-3.5 model improved from 53.6% to 66.4%. The score would still likely be too low for passing the course. GPT-4's score increased from 71.7% to 83.4%. This score would almost certainly enable a human learner to pass the course. Hence, the ability of GPT-4 to utilize feedback seems to be even stronger than the ability of the GPT-3.5 model evaluated in [38, 46]. That is, of course, if there is no requirement related to passing some minimal threshold for all of the projects. Even after 34 feedback iterations on the project 6 coding activities (debugging, refactoring, testing and packaging) GPT-4 obtained only 42 of the available 95 points. The low performance on this particular project could be explained in terms of several closely related factors. First, certain activities in this project could not be performed at all because they involved use of external tools beyond writing code. For example, the refactoring activities require creating new files and directories as well as renaming and moving files. The packaging activities involve interaction with the command line. Note that it would be possible to equip LLMs, including GPT models, with the ability to manipulate such external tools [30]. However, we did not consider the possibility in this study, and we have simply refrained from attempting such activities. Similarly, the remaining activities related to fixing errors, style correctness, and testing heavily rely on external tools (i.e., the debugger, `pylint`, `pycodestyle`, `pytest`). These activities were attempted since these tools are not strictly required. However, this is problematic because the feedback would often not contain all the necessary information. For example, in the testing task the feedback contains the high-level information about the test coverage (%). The models did not have access to the full coverage report that would show the lines not covered by the tests accessible to human learners.

Some of the coding activities were challenging for GPT-4 because they involved artifacts, beyond the task instructions, that were crucial for generating correct solution. Often, human learners would not necessarily find such tasks particularly challenging. One example of such an artifact is an input data set the size of which exceeds the maximum length of the model's prompt. While human learners can simply inspect the large input data set and identify the appropriate methods to parse the data as required by the task specifications, GPT-4 was unable to solve the task based on the instructions. By extracting sample records from the input data set and providing the sample input along with the expected output, GPT-4 was able to successfully implement the required data transformation.

Finally, we observed that GPT-4, despite being more successful than GPT-3.5, still struggles with fine-grained formatting requirements related to both, the output as well as the code itself. For example, GPT-4 was able to utilize the feedback based on the output of the style-checker that the code contained long lines over 100 characters, and modified the code to shorten the lines. At the same time, the model completely failed to utilize similar feedback from a more strict style-checking tool complaining about the lines that were over 79 characters long. In this particular case, GPT-4 was not able to break up a string that made the line in question 81 characters long. Similarly, when the provided feedback complained about a missing white-space between the | character and subsequent number in the task focused on printing a tabular report to a terminal, the GPT-4 model was not able to correct the solution accordingly.

6 IMPLICATIONS FOR TEACHING PRACTICE

This study provides evidence that programming instructors need to prepare for a world in which there is an easy-to-use widely accessible technology that can be utilized by learners to collect passing scores, with no effort whatsoever, on what today counts as viable programming knowledge and skills assessments. While this development has been apparent from the growing body of prior work [7, 10, 11, 38, 46] this paper is the strongest evidence reported so far in the context of programming education, and it is consistent with the OpenAI's GPT-4 release report [35].

In consequence, the instructors may consider shifting the focus from assessment to learning, i.e., they should prioritize the learning experience and skills development, rather than merely preparing learners for assessments. The learners should be encouraged to focus on learning and growth, rather than on always coming up with the right answers. The importance of academic honesty and ethical behavior in the classroom should be emphasized. Ideally, a culture that values original work and personal effort should be promoted. The instructors may need to move away from traditional assessments, such as multiple-choice exams. Instead, they may consider using more complex assessments such as code reviews, pair programming, and oral examinations that require students to demonstrate their understanding in real-time.

While it may be appealing to the instructors to understand the limitations of GPT models when it comes to handling MCQs to design tests that are difficult to be answered automatically we argue that this is likely not a viable approach. Given the rate of improvement over the past several years we document in this study, it appears quite likely the existing limitations will be overcome rather soon, reducing the effectiveness of tests designed to exploit the identified weaknesses. Instead of trying to create "GPT-proof" tests, it may be more productive to focus on developing assessments focused on higher-order thinking skills, such as critical thinking, problem-solving, and creativity as these are more difficult for the GPT models to replicate.

In programming activities, we identified several use cases where the application of GPT models does not (yet) appear to be straightforward. While similar argument as the one used for MCQs may be employed against potential hardening of the coding tasks by insisting on unusual fine-grained nuances of the code itself or its output, re-designing the tasks to rely on artifacts beyond the instructions and/or employment of external tools may be promising. This is because the ability to extract, consolidate, and express essential information from multiple sources may become an important skill critical for success of future learners. Instructors may consider incorporating such complex (but not necessary complicated) programming tasks requiring learners to consolidate problem context from multiple sources. This approach could reduce the misuse while promoting beneficial use of GPT models.

7 LIMITATIONS ANT THREATS TO VALIDITY

Although, the results of our study provide important insights into the evolving capabilities of the GPT models in passing typical assessments employed in introductory and intermediate Python classes, limitations in several areas must be acknowledged.

Generalizability. While Python is a widely used and representative language, it is one of many languages used in programming courses at the postsecondary level. GPT models have been shown to handle a number of programming languages [5, 8]. Nevertheless, our findings may not generalize to those languages with different structures, syntax, and conventions. Secondly, while the programming assessments used in this study are typical of those found in many Python programming courses, there may be other types of assessments (e.g., open questions, oral exams). Finally, our study was conducted using assessments in English, limiting the extent to which the findings apply to programming assessments in other languages. This poses a limitation as programming education is a global endeavour, and a significant proportion of programming courses and resources are available in languages other than English.

Prompt Engineering. The prompts employed in this study were carefully crafted following the best practices. However, it is important to acknowledge that our research did not explore the effects of prompt engineering, i.e., further fine-tuning the initial prompts. Engaging in prompt engineering could potentially lead to even stronger performance of the models. This unexplored area could limit the implications of our findings which should rather be interpreted as lower bound of the performance.

Information on GPT Models. It is not well-known what data have been used during the models' training. This is important because LLMs such as the ones evaluated in this study have capacity to memorize the data seen during their training. Hence, in case the assessments would have been seen during the training our experiments would not be able to show the models' capabilities to pass the assessments. They would rather be a testament to their memorization abilities. While we can be reasonably sure that the assessments employed in this study were not seen during the training as they are not part of any publicly available data set, this is an important limitation one has to be aware of when evaluating the OpenaAI's GPT models. Additional limitation is the lack of publicized technical details about GPT-4. The rapid development and evolution of GPT models, coupled with a lack of available technical details, makes it challenging for researchers to reproduce our study. Therefore, the results should be interpreted with this lack of full transparency and reproducibility in mind.

8 CONCLUSIONS AND FUTURE WORK

We analyzed the capabilities of the GPT-4 model in passing typical assessments, such as MCQ tests and coding exercises, in introductory and intermediate programming courses. The analysis is the needed response to the recent release of GPT-4 evaluating the extent to which previous findings regarding GPT-3 and 3.5 models are still relevant. The study highlights that the risk of learners becoming overly reliant on GPT models when completing programming course assignments and assessments is a genuine concern that must be taken seriously which is consistent with [1]. In light of these findings, it is crucial to develop strategies to address this growing challenge and maintain the relevance and integrity of programming education.

The future work should focus on development of innovative assessment techniques resilient to automatically generated solutions. This could include incorporating real-time problem-solving components, group projects, or other collaborative activities that require human interaction. Additionally, further studies of potential benefits and risks associated with LLMs are needed to enable educators to harness their power while mitigating potential drawbacks. Finally, as more capable LLMs continue to emerge, it is crucial to conduct ongoing evaluations of their capabilities in the context of programming education. This will ensure that educators and institutions remain informed and prepared to adapt their teaching methodologies and assessment strategies in response to the rapid advancements.

REFERENCES

[1] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2022. Programming Is Hard–Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation. *arXiv preprint arXiv:abs/2212.01020* (2022).

[2] Jillian Bommarito, Michael Bommarito, Daniel Martin Katz, and Jessica Katz. 2023. GPT as Knowledge Worker: A Zero-Shot Evaluation of (AI) CPA Capabilities. *arXiv preprint arXiv:abs/2301.04408* (2023).

[3] Michael Bommarito II and Daniel Martin Katz. 2022. GPT Takes the Bar Exam. *arXiv preprint arXiv:abs/2212.14402* (2022).

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712 [cs.CL]

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:abs/2107.03374* (2021). https://doi.org/10.48550/ARXIV.2107.03374

[7] 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.* 1136–1142.

[8] 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).

[9] Iddo Drori and Nakul Verma. 2021. Solving Linear Algebra by Program Synthesis. *arXiv preprint arXiv:2111.08171* (2021). https://doi.org/10.48550/ARXIV.2111.08171

[10] 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 *Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

[11] 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.

[12] Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. arXiv:2012.15723 [cs.CL]

[13] Amelia Gilson, Conrad W. Safranek, Tao Huang, Vimig Socrates, Lim Sze Chi, Roderick A. Taylor, and David Chartash. 2022. How Well Does ChatGPT Do

When Taking the Medical Licensing Exams? The Implications of Large Language Models for Medical Education and Knowledge Assessment. In *medRxiv*. https://doi.org/10.1101/2022.12.23.22283901.

[14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring Massive Multitask Language Understanding. *arXiv preprint arXiv:abs/2009.03300* (2020). https://doi.org/10.48550/ARXIV.2009.03300

[15] Sajed Jalil, Suzzana Rafi, Thomas D. LaToza, Kevin Moran, and Wing Lam. 2023. ChatGPT and Software Testing Education: Promises & Perils. https://doi.org/10.48550/arXiv.2302.03287 arXiv:2302.03287 [cs.SE]

[16] Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Xing Wang, and Zhaopeng Tu. 2023. Is ChatGPT A Good Translator? Yes With GPT-4 As The Engine. arXiv:2301.08745 [cs.CL]

[17] Anjan Karmakar, Julian Aron Prenner, Marco D'Ambros, and Romain Robbes. 2022. Codex Hacks HackerRank: Memorization Issues and a Framework for Code Synthesis Evaluation. *ArXiv* abs/2212.02684 (2022).

[18] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. 2023. GPT-4 Passes the Bar Exam. *Available at SSRN 4389233* (2023).

[19] 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)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[20] Donald Ervin Knuth. 1984. Literate programming. *The computer journal* 27, 2 (1984), 97–111.

[21] Dimitra Kokotsaki, Victoria Menzies, and Andy Wiggins. 2016. Project-based learning: A review of the literature. *Improving schools* 19, 3 (2016), 267–277.

[22] Tiffany H Kung, Morgan Cheatham, Arielle Medinilla, Czarina Sillos, Lorie De Leon, Camille Elepano, Marie Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. 2022. Performance of ChatGPT on USMLE: Potential for AI-Assisted Medical Education Using Large Language Models. *medRxiv preprint* (2022). https://doi.org/10.1101/2022.12.19.22283643.

[23] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:abs/1704.04683* (2017).

[24] 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. https://doi.org/10.48550/arXiv.2304.03938 arXiv:2304.03938 [cs.CY]

[25] 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)*. Association for Computing Machinery, New York, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

[26] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158 arXiv:https://www.science.org/doi/pdf/10.1126/science.abq1158

[27] Valentin Liévin, Christoffer Egeberg Hother, and Ole Winther. 2022. Can large language models reason about medical questions? *ArXiv preprint arXiv:abs/2207.08143* (2022).

[28] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. https://doi.org/10.48550/ARXIV.2209.09513

[29] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. 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, 931–937. https://doi.org/10.1145/3545945.3569785

[30] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented Language Models: a Survey. *arXiv preprint arXiv:2302.07842* (2023).

[31] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? A new dataset for open book question answering. *arXiv preprint arXiv:abs/1809.02789* (2018).

[32] Ahmad Haji Mohammadkhani, Chakkrit Kla Tantithamthavorn, and Hadi Hemmati. 2022. Explainable AI for Pre-Trained Code Models: What Do They Learn? When They Do Not Work? *ArXiv preprint* arXiv:abs/2211.12821 (2022).

[33] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 839–849.

[34] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. 1–5. https://doi.org/10.1145/3524842.3528470

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

[36] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:abs/2203.02155* (2022).

[37] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2022. Do Users Write More Insecure Code with AI Assistants? *arXiv preprint arXiv:2211.03622* (2022).

[38] 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).

[39] 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. *arXiv preprint arXiv:2304.02491* (2023).

[40] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. (2019).

[42] Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. arXiv:2102.07350 [cs.CL]

[43] Joshua Robinson, Christopher Michael Rytting, and David Wingate. 2022. Leveraging Large Language Models for Multiple Choice Question Answering. *arXiv preprint arXiv:abs/2210.12353* (2022). https://doi.org/10.48550/ARXIV.2210.12353

[44] 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)*. Association for Computing Machinery, New York, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[45] Jaromir Savelka, Arav Agarwal, Christopher Bogart, and Majd Sakr. 2023. Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions about Code. *arXiv preprint arXiv:2303.08033* (2023).

[46] 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 28th Annual ACM Conference on Innovation and Technology in Computer Science Education*.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *ArXiv* abs/2201.11903 (2022).

[49] Guang Yang, Yu Zhou, Wenhua Yang, Tao Yue, Xiang Chen, and Taolue Chen. 2022. How Important are Good Method Names in Neural Code Generation? A Model Robustness Perspective. *ArXiv* abs/2211.15844 (2022).

[50] Biao Zhang, Barry Haddow, and Alexandra Birch. 2023. Prompting Large Language Model for Machine Translation: A Case Study. *ArXiv* abs/2301.07069 (2023).

[51] Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate Before Use: Improving Few-Shot Performance of Language Models. arXiv:2102.09690 [cs.CL]

[52] Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Huai hsin Chi. 2022. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *ArXiv* abs/2205.10625 (2022).

A MCQ EXAMPLES PER TYPES

True/False

Developers that write code individually are not expected to apply code standards.
A. True
B. False

Evaluate the following expression and determine whether it is True or False.

```
2 + 2 != 2 * 2
```

A. True
B. False

Identify True/False Statement

Which of the following statements is false?
A. The pandas module provides some CSV-related methods.
B. Python has a built-in XML package with several modules for XML parsing.
C. JSON data format has syntax to represent all Python data structure types.
D. Python has a built-in `csv` module containing methods for reading and writing into CSV files.

Take a look at the snippet and choose one of the following statements which is true:

```
nums = []
vals = nums[:]
vals.append(1)
```

A. `nums` is longer than 'vals'
B. `vals` *is longer than* `nums`
C. `nums` and `vals` are of the same length

Finish Statement

The '**' operator:
A. performs duplicated multiplication
B. does not exist
C. performs exponentiation

Right-sided binding means that the following expression:

```
1 ** 2 ** 3
```

will be evaluated:
A. from right to left
B. in random order
C. from left to right

Output

What is the output of the following snippet if the user enters two lines containing `2` *and* `4` *respectively?*

```
x = int(input())
y = int(input())
print(x + y)
```

A. 2
B. 24
C. 6

What is the output of the following snippet?

```
my_list_1 = [1, 2, 3]
my_list_2 = []
for v in my_list_1:
    my_list_2.insert(0, v)
print(my_list_2)
```

A. `[1, 2, 3]`
B. `[1, 1, 1]`
C. `[3, 3, 3]`
D. `[3, 2, 1]`

Fill-in Blanks

Fill in the blank of the `is_negative` function definition shown below, so that the function returns `True` when the argument provided to `num` is a negative number and returns `False` otherwise.

```
def is_negative(num):
    return ________________
```

A. `not (num > 0)`
B. `num > 0`
C. `num <= 0`
D. `num < 0`

The following code snippet should open the `myfile` file and assign the lines to the `all_lines` variable. Which of the options below should be used to fill in the blanks?

```
with ___________________________
    all_lines = file.readlines()
```

A. `open("myfile",'r') as file:`
B. `"myfile" in open as file:`
C. `with open "myfile" as file:`

Other

How many times will the code snippet below print 'X'?

```
for i in range(1, 7):
    for j in range(2, 6):
    print('X')
```

A. 24
B. 28
C. 35

Given the piece of code presented in the code snippet below, what is the value of `palindromes[1]`?

```
palindromes = ['pop', 'noon', 'madam']
```

A. `'pop'`
B. `'noon'`
C. `'p'`
D. `'madam'`
E. `'o'`