
Benchmarking Generative Models on Computational Thinking Tests in Elementary Visual Programming

Victor-Alexandru Pădurean
MPI-SWS
vpadurea@mpi-sws.org

Adish Singla
MPI-SWS
adishs@mpi-sws.org

Abstract

Generative models have demonstrated human-level proficiency in various benchmarks across domains like programming, natural sciences, and general knowledge. Despite these promising results on competitive benchmarks, they still struggle with seemingly simple problem-solving tasks typically carried out by elementary-level students. How do state-of-the-art models perform on standardized programming-related tests designed to assess computational thinking and problem-solving skills at schools? In this paper, we curate a novel benchmark involving computational thinking tests grounded in elementary visual programming domains. Our initial results show that state-of-the-art models like GPT-4o and Llama3 barely match the performance of an average school student. To further boost the performance of these models, we fine-tune them using a novel synthetic data generation methodology. The key idea is to develop a comprehensive dataset using symbolic methods that capture different skill levels, ranging from recognition of visual elements to multi-choice quizzes to synthesis-style tasks. We showcase how various aspects of symbolic information in synthetic data help improve fine-tuned models' performance. We will release the full implementation and datasets to facilitate further research on enhancing computational thinking in generative models.

1 Introduction

The recent advances in generative models and large language models (LLMs) have the potential to positively impact a wide variety of domains, such as medicine [1, 2, 3], arts [4, 5], and education [6, 7, 8, 9]. This potential is reflected by their success on a wide range of popular competitive benchmarks assessing their knowledge of natural sciences and day-to-day facts [10, 11, 12, 13, 14] and their skills in programming. For example, GPT-4o [10] is capable of obtaining a high accuracy on two popular programming benchmarks: 90.2% on HumanEval [15] and 87.5% on MBPP [16]. Previous studies also showed that GPT-4 [17] is capable of passing assessments in higher education programming courses, achieving course totals greater than 79% [18].

Despite these promising results, state-of-the-art models struggle with seemingly simple tasks. These models often underperform in tasks requiring mathematical reasoning, planning, and problem-solving [19, 20, 21, 22]. For example, they fail to solve planning tasks involving stacking of colored blocks [23]. Moreover, generative models often face problems with basic algebra and counting [19], or coming up with correct codes in visual programming domains [24], tasks which can successfully be carried out by elementary-level school students. These weaknesses seem to contradict the generative models' impressive performance in complex programming tasks. Based on these observations, we aim to study how generative models tackle programming tasks specifically designed to foster computational thinking and problem-solving skills in elementary-level students. This leads to our main research question: *How do state-of-the-art models perform on standardized programming-related tests designed to assess computational thinking and problem-solving skills at schools?*

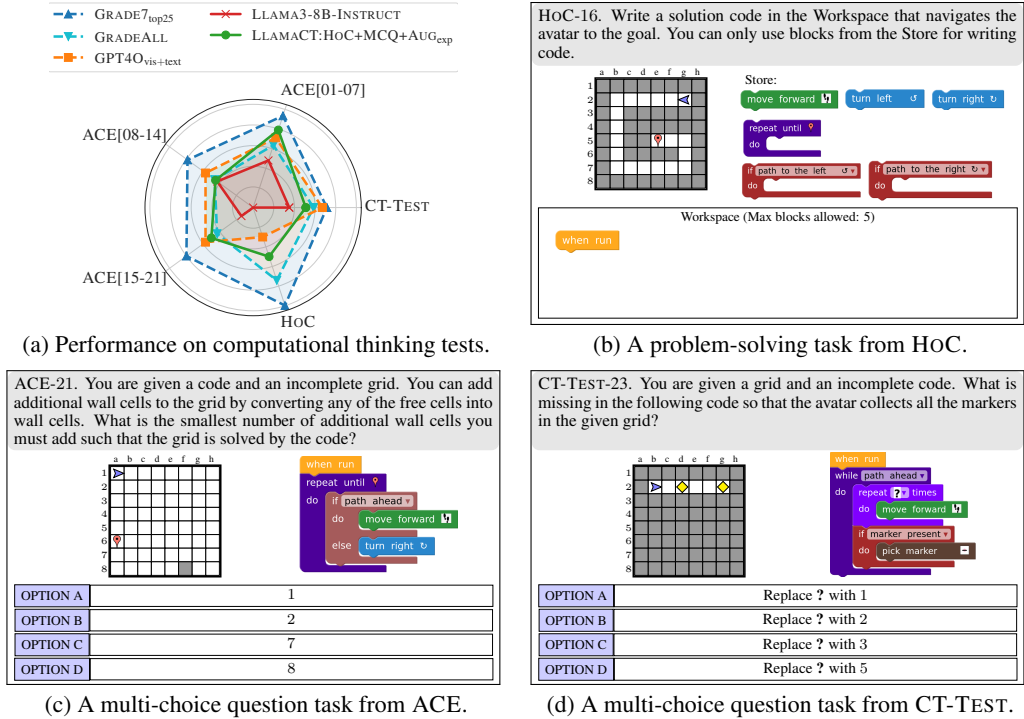


Figure 1: **(a)** shows the performance of school students compared to various models on a scale of 0 to 100. We break down the ACE [25] test into its constituent parts: *Analyzing* (ACE[01-07]), *Evaluating* (ACE[08-14]), and *Creating* (ACE[15-21]). **(b)** shows MAZE16 from *Hour of Code:Maze Challenge* (HOC) [26, 27], an example of a solution synthesis task. **(c)** shows a *Creating* multi-choice question task from the ACE test. **(d)** shows a multi-choice question task from the CT-TEST [28, 29].

In this paper, we introduce a novel benchmark for assessing generative models’ computational thinking and problem-solving capabilities. We conduct extensive experiments with various models and our results show that state-of-the-art models struggle with the computational thinking tests in our benchmark. Figure 1a illustrates how GPT-4o barely matches the performance of an average school student, with Llama3 [11] performing even worse.

We make the following contributions towards improving the models’ performance on computational thinking tests: (1) We introduce a novel data generation methodology based on symbolic methods. An important aspect of the generated dataset is that it captures different skill levels, ranging from recognition of visual elements to multi-choice questions to synthesis-style tasks. (2) We fine-tune Llama3-8B and obtain the LLAMACT family of models, the best of which achieves an accuracy on par with GPT-4o (see Figure 1a). We further analyze how various aspects of symbolic information in our synthetic dataset help improve the fine-tuned models’ performance. (3) We will release the data and implementation to promote further research on enhancing computational thinking in generative models.¹

2 Related Work

We identify two key research themes in the literature: one focuses on the programming capabilities of generative models, and the other focuses on their general reasoning capabilities. To our knowledge, this paper is the first to evaluate numerous generative models on a comprehensive set of computational thinking tests grounded in elementary visual programming. Figure 2 presents a comparison between our work and the most relevant benchmarks in the literature.

Benchmarks assessing programming capabilities. Several works benchmark the programming capabilities of models, with popular examples of benchmarks including HumanEval [15], MBPP [16], and APPS [33]. For example, HumanEval [15] focuses on Python code generation but lacks mul-

¹GitHub repo: <https://github.com/machine-teaching-group/neurips2024-benchmark-ct>.

Work	Domain	Evaluation tasks	Multimodal	Benchmark evaluation size	Trained model	Human comparison
Our Work	visual programming	code synthesis, multiple choice questions	✓	65	✓	✓
HumanEval [15]	programming	Python code writing	✗	164	✓	✗
MMCode [30]	programming	Python code writing	✓	3,548	✗	✗
MathVista [31]	mathematics	multiple choice questions, integer free form questions	✓	6,141	✗	✓
MMMU [32]	diverse	multiple choice questions, open answer questions	✓	11,500	✗	✓
HoC+Karel [24]	visual programming	code synthesis, tracing, grid synthesis	✗	30	✗	✗

Figure 2: Comparison of our work with related benchmarks. The first column shows the name of the benchmark and the work it was introduced in. “Domain” specifies the domain for which the benchmark was designed, and “Evaluation tasks” outlines the tasks involved. “Multimodal” indicates if the benchmark includes both visual and textual data. “Benchmark evaluation size” shows the number of samples in each benchmark. “Trained model” notes whether any model is trained in the work, and “Human comparison” indicates if model performance was compared to that of humans.

timodal elements and human comparisons, as shown in Figure 2. A more recent benchmark, MMCode [30], includes visual information in traditional coding tasks to assess multimodal model capabilities. However, this benchmark does not include comparisons between model and human performance, nor does it explore potential improvements in model performance through fine-tuning or other methods. Besides program generation, other benchmarks handle code completion, translation, summarization, debugging or explanation generation [34, 35, 36, 37], thus analyzing numerous programming-related tasks. However, these works typically focus on generating code or explanations and do not evaluate the core computational thinking and problem-solving skills of models. In contrast, our paper seeks to provide a deeper understanding of these capabilities. Additionally, we train models on our dataset and compare their performance to human counterparts, addressing gaps in previous studies.

Benchmarks assessing reasoning capabilities. For general reasoning, benchmarks like MathVista [31] and MMMU [32] assess models on tasks involving multiple-choice and free-form questions, with a focus on multimodal data. These, along with other benchmarks, evaluate reasoning in fields such as mathematics and the natural sciences [14, 20, 32, 31], planning [22, 23, 38], and causal reasoning [39, 40]. Our benchmark goes beyond these by including a variety of tasks that assess computational thinking through visual programming. This relatively underexplored area can offer intriguing insights into the reasoning capabilities of generative models. Previous efforts [24] address visual tasks without the multimodal component and do not include human comparisons. In contrast, our benchmark integrates multimodal tasks combining both programming and visual reasoning, while also comparing models directly against human performance, offering a more comprehensive evaluation of reasoning abilities in generative models.

3 Computational Thinking Tests in Elementary Visual Programming

This section first provides a background on visual programming, and then introduces the sources we use for curating our benchmark and as the basis for our synthetic dataset generation methodology.

3.1 Preliminaries and Definitions for Elementary Visual Programming

The space of grids. A visual grid, denoted as G , includes an avatar with an initial position (row, column) and orientation (north, east, south, west), alongside free cells, wall cells, and a goal or multiple markers. The avatar is required to reach the goal or interact with the markers. The resulting grid space includes visual grids based on HOC by Code.org [26, 27], such as the grids in Figures 1b and 1c, and Karel [41], such as the grid in Figure 1d.

The space of codes. The set of valid codes C is defined via a domain-specific language (DSL). We adopt DSLs previously used in literature for visual programming [42, 43, 44]. A code $C \in C$ is characterized by its size C_{size} , utilized constructs C_{blocks} , and programming concepts exercised in terms of its nesting structure C_{sketch} . For example, the code in Figure 1c uses $C_{\text{size}} = 5$

Concepts	HoC	ACE			CT-TEST
		Analyzing ACE[01-07]	Evaluating ACE[08-14]	Creating ACE[15-21]	
Basic actions	H01–H05	Q01	Q08	Q15	P01–P04
REPEAT{}	H06–H09	Q02, Q05	Q12	Q16	P05, P06
REPEATUNTIL{}	H10–H13	Q06	Q09	Q17, Q18	P09, P10
REPEATUNTIL{IF}	H14–H17	Q07	Q10	Q19	P13, P14
REPEATUNTIL{IFELSE}	H18, H19	Q04	Q11, Q14	Q20, Q21	P17, P18
REPEATUNTIL{IFELSE{IFELSE}}	H20				P19, P20
REPEAT{REPEAT}			Q13		P08, P27, P28
REPEAT{IF}		Q03			
REPEATUNTIL{IF; IF}					P16
REPEATUNTIL{REPEAT}					P11
REPEATUNTIL{IF{REPEAT}}					P15
WHILE{ }; REPEAT{ }					P21
WHILE{REPEAT; REPEAT}					P22
WHILE{REPEAT; IF}					P23
WHILE{IF{WHILE}}					P24

Figure 3: Programming concepts C_{sketch} required for solving tasks in HoC [26], ACE [25], and CT-TEST [28, 29]. HoC comprises code-writing tasks. ACE and CT-TEST comprise multi-choice question tasks. ACE is further split according to the higher cognitive levels of Bloom’s taxonomy [45, 46].

blocks, with constructs $C_{\text{blocks}} = \{\text{move}, \text{turnRight}, \text{REPEATUNTIL}, \text{IFELSE}\}$, and is structured as $C_{\text{sketch}} = \text{REPEATUNTIL}\{\text{IFELSE}\}$. Executing a code on a grid generates a sequence of avatar locations, referred to as trace, along with a sequence of basic actions executed i.e., constructs from $\{\text{move}, \text{turnLeft}, \text{turnRight}\}$. A code is considered to solve a grid if it successfully navigates the avatar to the goal or interacts correctly with the markers (e.g., collects them when intended).

Solution synthesis tasks. A solution synthesis task is defined by the following elements: a grid G , an allowed set of constructs called Store , and a maximum code size maxSize . The objective is to write a solution code C that successfully solves G while respecting $C_{\text{blocks}} \subseteq \text{Store}$ and $C_{\text{size}} \leq \text{maxSize}$. Figure 1b exemplifies a solution synthesis task, where a solution code C should solve G , have $C_{\text{blocks}} \subseteq \{\text{move}, \text{turnRight}, \text{turnLeft}, \text{REPEATUNTIL}, \text{IFELSE}\}$ and $C_{\text{size}} \leq 5$.

Multi-choice question tasks. A multi-choice question (MCQ) task is defined by the following elements: a text description, a set of grids or codes, one correct option, and three distractor options. The objective is to choose the correct option out of four options. For example, Figures 1c and 1d have a text description inside the gray area, a given grid and a given code, and four options. The correct option for Figure 1c is Option A, and the correct option for Figures 1d is Option A as well.

3.2 Three Different Computational Thinking Tests

Our benchmark is based on two pedagogically validated computational thinking tests comprising multiple-choice question tasks [25, 28, 29] and a popular curriculum comprising code-writing tasks [26]. Henceforth, we refer to these three as tests, and we will use them throughout the paper to measure the performance of generative models. These tests have been carefully designed by educational experts to assess or teach a diverse set of skills in elementary visual programming within the duration of a typical one-hour school lesson. They are representative of computational thinking in this domain, providing valuable data on student performance, which we can use as a basis to benchmark the performance of generative models. Figure 3 gives an overview of the programming concepts C_{sketch} utilized by tasks in each test. Next, we provide details for each test.

HoC. This test includes 20 code-writing tasks from Code.org’s popular block-based visual programming lesson *Hour of Code:Maze Challenge* [26]. The tasks mainly cover concepts such as basic actions, REPEAT and REPEATUNTIL loops, as well as IF and IFELSE branching (see Figure 3). This curriculum has been used by millions of learners to get acquainted with programming and to assess students’ programming background [25, 26, 27].

ACE. This test includes 21 multi-choice question tasks from the ACE test, which was designed to evaluate higher cognitive levels of Bloom’s taxonomy: *Analyzing*, *Evaluating*, and *Creating* [25, 45, 46]. These tasks were selected from a larger pool to ensure balanced coverage of cognitive levels and programming concepts, being validated using standardized pedagogical tools. Figure 3 categorizes each task by cognitive level and programming concepts covered.

Synthetic data	Original Size	Selected Size	Percentage
Solution synthesis	7,576	7,576	6.77%
Multi-choice questions (MCQ)	9,223	9,223	8.25%
Analyzing MCQ (A)	2,779	2,779	2.49%
Evaluating MCQ (E)	2,072	2,072	1.85%
Creating MCQ (C)	4,372	4,372	3.91%
Fine-grained: Basics	586,341	11,726	10.48%
Locate avatar (LoA)	65,149	1,336	1.19%
Locate goal (LoG)	65,149	1,273	1.14%
Apply action (Act)	195,447	3,930	3.51%
Sense condition (Sense)	260,596	5,187	4.64%
Fine-grained: Tracing	15,152	15,152	13.54%
Sequence trace	7,576	7,576	6.77%
Code trace	7,576	7,576	6.77%
Fine-grained: Grid synthesis	68,184	68,184	60.95%
Place avatar	7,576	7,576	6.77%
Place goal	7,576	7,576	6.77%
Place avatar+goal	7,576	7,576	6.77%
Place walls	37,880	37,880	33.87%
Design all	7,576	7,576	6.77%
Total	586,341	111,861	100%



(a) Distributions for synthetically generated data.

(b) Treemap of selected data distribution.

Figure 4: Our synthetically generated training dataset. Subsampling is done only in the case of basics.

CT-Test. This test is based on CT-TEST, one of the earliest and most popular computational thinking tests in block-based visual programming [28, 29]. Out of 28 tasks in the original set, we curate 24 tasks compatible with our definitions and representation. Figure 3 shows the programming concepts covered, with the original task numbering: if its number is not in the table, we have not included the task.

4 Synthetic Data Generation to Fine-tune Models for Computational Thinking

In this section, we introduce our novel data generation methodology for computational thinking and problem-solving skills. With the resulting data (see Figure 4), we aim to fine-tune models to increase performance on all three tests. Next, we present our three main methods for generating data.

4.1 Synthetic Data for Solution Synthesis

We first generate data for solution synthesis tasks. Our process will start with generating a dataset of pairs $(C, \{G\})$, where C is a code and $\{G\}$ is a set of grids solved by C . To obtain $(C, \{G\})$, we employ existing techniques for synthesizing code C and grid G [42, 43, 44]. We then split the sets into pairs of one solution code and one grid (C, G) . We extract `Store` and `maxSize` from code C . Then, we treat $(G, \text{Store}, \text{maxSize})$ as input for the task, and keep C as target output.

To enhance the fine-tuning process, we aim to train the model to first produce a trace and sequence of basic actions that the avatar should execute to reach the goal, and then to produce the solution code. We refer to the trace and sequence of basic actions as an explanation for the produced answer. This method is grounded in previous research, which has shown that smaller models benefit from richer signals while being fine-tuned, leading to more careful reasoning at inference [47, 48]. However, unlike literature, we cannot rely on more powerful models like GPT-4 to produce these explanations, as state-of-the-art models struggle with computational thinking (see Figure 1a). So, we rely on symbolic methods such as executing codes on grids via an emulator to produce correct traces and basic action sequences as explanations.

4.2 Synthetic Data for Multi-choice Programming Questions

We now focus on generating MCQ tasks similar to those in ACE and CT-TEST [25, 28, 29]. We generate MCQs starting from the same $(C, \{G\})$ used for generating solution synthesis tasks, using a template-based approach, with manually written text descriptions for each task type. Next, we present our task types covering all the higher cognitive levels in Bloom’s taxonomy – *Analyzing*,

Evaluating, Creating [25, 45, 46]. We also augment MCQs with explanations similar to the ones we use for solution synthesis tasks.

Analyzing. First, we describe the process of generating *Analyzing* cognitive level tasks. For this level, we generate three task types: tasks that require selecting a solution code for a given grid, tasks that require indicating which given grids are solved by a given code, and tasks that require reasoning about the trace of a given code on a given grid and selecting the cells visited by the avatar. To offer an overview of our method, we explain the generation process of one task type, namely reasoning about the trace. We start from a pair (C,G) and the text description specific to this type of task. Then, we generate the correct option by executing C on G and selecting random cells visited by the avatar. We generate distractor options by randomly picking free cells that were not visited by the avatar. Note that this task is correct by construction, unlike some more complex task types below that need validation. Finally, we have the task containing text description, C, G, the correct option, and three distractor options.

Evaluating. Second, we describe the process of generating *Evaluating* cognitive level tasks. For this level, we generate four task types: tasks that require identifying bugs, tasks that require repairing bugs, tasks that require evaluating code equivalence with no given grid, and tasks that require evaluating code equivalence given a grid. We explain generation process for the task type that requires repairing bugs. We start from pair (C,G) and corresponding text description. We generate a mutation and apply it to code C to obtain C_{mut} [44, 49]. The correct option is obtained as the reverse mutation that would transform C_{mut} back to C. Distractor options are obtained by generating three other mutations. We validate the task by applying reverse mutation on C_{mut} and checking whether resulting code solves G. We also apply distractor mutations on C_{mut} and make sure that resulting codes do not solve G. Finally, we have the task containing text description, C_{mut} , G, the correct option, and three distractor options.

Creating. Third, we describe the process of generating *Creating* cognitive level tasks. For this level, we generate six types of tasks that require reasoning about modifying an incomplete grid such that the given code solves the modified grid. We generate: tasks that require placing the avatar, tasks that require reasoning about the number of possible initial avatar locations, tasks that require placing the goal, tasks that require counting possible goal positions, tasks that require placing walls, and tasks that require counting the minimum number of walls needed. For example, a task similar to Figure 1c can be synthesized by starting from a pair (C,G) and the text description. We set the correct option by randomly picking the number of walls to remove from $\{1, 2, 3\}$, in this case 1. We remove one wall from G, obtaining G_{mut} . We generate three distractor options by applying arithmetic operations to the correct option. To validate the task correctness, we check whether C solves any grid obtained via adding all possible combinations of walls less than the correct option. For this specific example, as the correct option for the example is 1, we just need to check if the grid will be solved with no added walls. Finally, we have the task containing text description, C, G_{mut} , the correct option, and three distractor options.

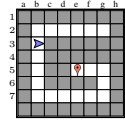
4.3 Synthetic Data at Fine-grained Skills

We now introduce new kinds of tasks, aimed at improving fine-grained skills fundamental to making the models better understand the domains of our computational thinking tests. The main intuition for using various fine-grained skills is due to inter-task transfer observed during instruction-tuning [50, 51, 52], which can enhance performance on solution synthesis tasks and MCQ tasks. We now give details about generating three kinds of tasks for improving fine-grained skills.

Basics. We describe the process of generating tasks aimed at familiarizing models with the fundamental aspects of visual programming. We generate four types of basic tasks: tasks that require locating the avatar in a given grid, tasks that require locating the goal in a given grid, tasks that require specifying the new location of the avatar after executing a given basic action on a given grid, and tasks which require specifying the outcome of applying a given condition to a given grid. For example, we generate the input for the task in Figure 5a starting from a grid G, a randomly selected condition present in the DSL, and a fixed text description. The target output is obtained as the outcome of applying the condition on G, in this case True as the avatar has a free cell to its right. As the number of obtainable basic tasks is very large, we subsample to 2% of the original size (see Figure 4a). We have empirically chosen this percentage, analyzing the performance on a validation segment corresponding to basic tasks.

Tracing. Next, we describe the process of generating tasks aimed at enhancing the model’s understanding of the interaction between the basic actions, conditions, and grids, crucial for answering the MCQs corresponding to the *Analyzing* cognitive level in tests. We generate two types of tasks: tasks requiring to produce the trace obtained by applying a sequence of basic actions to a given grid, and

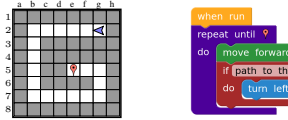
You are given a grid and a condition. Respond with either True or False, as returned by the condition when executed on the grid.



Target output: True

(a) Example for sense condition in basics.

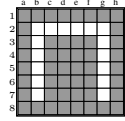
You are given a grid and a code. Trace the locations of the avatar when the code is executed on the grid. If at any time the avatar crashes, print crash and stop the trace.



Target output: g2:west → f2:west → e2:west → d2:west → c2:west → b2:west → b2:south → b3:south → b4:south → b5:south → b6:south → b7:south → b7:east → c7:east → d7:east → e7:east → f7:east → g7:east → g7:north → g6:north → g5:north → g5:west → f5:west → e5:west → goal

(b) Example for code trace in tracing.

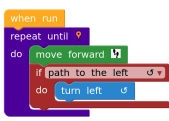
You are given a code and an incomplete grid without the avatar and the goal. Pick the avatar's initial location and the location of the goal to create a complete grid that can be solved by the given code.



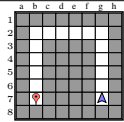
Target output:
avatar at g7 facing north
goal at b7

(c) Example for place avatar+goal in grid synthesis.

You are given a code. Pick the avatar's initial location, the location of the goal, and the wall locations to create a complete grid that can be solved by the given code.



Target output:



(d) Example for design all in grid synthesis.

Figure 5: Illustrative examples for synthetically generated tasks with target outputs for the fine-grained skills. The tasks have been adapted for readability. For example, in (d), we illustrate the target output visually, but the actual target output is in textual form.

tasks requiring to produce the trace of a given code on a given grid. For example, in Figure 5b, we use a pair (C,G) and a fixed text description as input and treat the trace of C on G as the target output.

Grid synthesis. Finally, we describe the generation of tasks aimed at boosting the model’s understanding of the role of each grid element and how it can influence the execution of a code, crucial for answering the MCQs corresponding to the *Creating* cognitive level in tests. We generate five types of tasks: tasks that require placing the avatar, tasks that require placing the goal, tasks that require placing both the avatar and the goal, tasks that require placing walls, and tasks that require designing a full grid. For example, we generate the task in Figure 5c starting from a pair (C,G) and a fixed text description, removing the avatar and the goal from G, and giving the incomplete grid and C as input. The target output is the avatar and goal positions from G. Similarly, in Figure 5d, we keep only C as input, and require as output a natural language description of G. We also include tracing information and sequences of basic actions as explanations during fine-tuning in a style similar to solution synthesis and MCQ tasks.

5 Experiments

In this section, we compare performance of open-access models, OpenAI’s GPT family of models, and our fine-tuned LLAMACT. We also include school students’ performance based on studies in existing literature [25, 29]. We evaluate LLAMACT variants to assess the impact of training on different data segments and the use of explanations. We also provide insights into models’ reasoning process.

5.1 Techniques Evaluated

We start with NAIVE technique, a baseline that generates random tokens for HOC tasks and selects most frequent answer from four options for ACE and CT-TEST tasks. Note that because of non-uniform distribution of options, NAIVE yields better results for ACE and CT-TEST than randomly choosing an option. Next, we present techniques based on generative models and performance of school students. Figure 6 shows a summary of our model-based techniques.

Open-access models. We select smaller, instruction-tuned models from the Llama family, such as the 7B parameter version of CodeLlama [53] and the 8B parameter version of Llama3 [11], alongside the 7B parameter version of Llava [54]. These are referred to as CODELLAMA-7B-INSTRUCT,

Technique	Base model	Modality		Fine-tuning data	Explanation
		Visual	Text		
CODELLAMA-7B-INSTRUCT	CodeLlama-7B [53]	✗	✓	n/a	n/a
LLAVA1.5-7B	LLaVA-v1.5-7B [54]	✓	✓	n/a	n/a
LLAMA3-8B-INSTRUCT	Llama3-8B [11]	✗	✓	n/a	n/a
GPT3.5, GPT4 _{text} , GPT4O _{text}	GPT-3.5 [56], 4o [17], 4o [10]	✗	✓	n/a	n/a
GPT4 _{vis} , GPT4O _{vis}	GPT-4V [57], 4o [10]	✓	✗	n/a	n/a
GPT4 _{vis+text} , GPT4O _{vis+text}	GPT-4V [57], 4o [10]	✓	✓	n/a	n/a
LLAMACT:HOc	Llama3-8B [11]	✗	✓	Solution syn	None
LLAMACT:HOc+MCQ	Llama3-8B [11]	✗	✓	Solution syn+MCQ	None
LLAMACT:HOc _{exp}	Llama3-8B [11]	✗	✓	Solution syn	Train
LLAMACT:HOc+MCQ _{exp}	Llama3-8B [11]	✗	✓	Solution syn+MCQ	Train
LLAMACT:HOc+MCQ+AUG _{exp}	Llama3-8B [11]	✗	✓	Full data	Train
LLAMACT:HOc+MCQ+AUG _{exp} *	Llama3-8B [11]	✗	✓	Full data	Train+infer

Figure 6: Table summarizing techniques based on generative models, showing the base model and whether the input grid is represented visually or in text (modality). For fine-tuned models (e.g., LLAMACT), the table specifies the data segment used for training and whether models were trained with no explanations, to generate explanations, or to receive explanations during inference.

LLAMA3-8B-INSTRUCT, and LLAVA1.5-7B, respectively. For LLAVA1.5-7B, we incorporate both natural language and visual representations of grids to utilize its vision capabilities. All techniques are prompted to use chain-of-thought (CoT) [55].

GPT family. This group includes techniques based on GPT-3.5 [56] and GPT-4 [10, 17]. We start with GPT3.5 technique which processes tasks, including grids, only in natural language, as it has no vision capabilities. Similarly, GPT4_{text} is solely based on natural language. Next, for the GPT4_{vis}, we input the grids solely as visual representation, while the rest of the task is represented through natural language. GPT4_{vis+text} technique combines textual and visual representations for grids, with the rest of the task in natural language. We also include similar techniques based on the newer GPT-4o [10], namely GPT4O_{text}, GPT4O_{vis}, and GPT4O_{vis+text}. All techniques are prompted to use CoT.²

Fine-tuned models. We fine-tune the instruction-tuned 8B parameter version of the Llama3 model using LoRA [58] and obtain the LLAMACT family. LLAMACT:HOc and LLAMACT:HOc_{exp} are fine-tuned using only the generated solution synthesis tasks, LLAMACT:HOc+MCQ and LLAMACT:HOc+MCQ_{exp} are trained using both generated solution synthesis tasks and generated MCQ tasks, and LLAMACT:HOc+MCQ+AUG_{exp} is trained on the full synthetic dataset. LLAMACT:HOc_{exp}, LLAMACT:HOc+MCQ_{exp}, and LLAMACT:HOc+MCQ+AUG_{exp} are trained on target outputs enriched with explanations. Additionally, LLAMACT:HOc+MCQ+AUG_{exp}* simulates an ideal scenario where the correct reasoning process is known at inference time.

Human students. We benchmark these models against the performance of students observed in literature, reporting results for one group of students for HOc and ACE [25], and for a different group of students for CT-TEST [29]. GRADEALL comprises the average performance of students across grades 3-7 for HOc and ACE, and the average performance of students across grades 5-10 for CT-TEST. GRADE7_{top25} represents the top 25% of grade 7 students for HOc and ACE, and the top 25% of grade 7-8 students for CT-TEST, showing the performance of the best students.

5.2 Performance on Computational Thinking Tests

We evaluate techniques on HOc, ACE, and CT-TEST, introduced in Section 3.2. Figure 7 shows results, with accuracy computed as percentage of correctly answered tasks in one trial out of total tasks per test. We set temperature to 0 and assess over three seeds, reporting average results as mean (stderr).

Combining language and vision enhances performance. Providing input in both text and visual modality leads to better results for GPT4O_{vis+text} when compared with GPT4O_{vis} and GPT4O_{text}. Similar results hold for GPT4_{vis+text} when compared with GPT4_{vis} and GPT4_{text}.

Symbolic information-based explanations improve outcomes. Template-based explanations derived from execution information used while training enhance reasoning at inference and boost model

²Few-shot prompting did not improve results. All results are based on zero-shot CoT prompting.

Technique	HoC	ACE	CT-TEST	Overall
NAIVE	0.0	33.0	33.0	22.0
CODELLAMA-7B-INSTRUCT	0.0 (0.0)	14.3 (0.0)	29.2 (0.0)	14.3 (0.0)
LLAVA1.5-7B	0.0 (0.0)	28.6 (0.0)	20.8 (0.0)	16.7 (0.0)
LLAMA3-8B-INSTRUCT	0.0 (0.0)	34.9 (2.0)	34.7 (5.0)	22.9 (1.3)
GPT3.5	25.0 (0.0)	31.7 (4.0)	36.1 (5.0)	31.1 (0.5)
GPT4 _{vis}	18.3 (2.0)	31.7 (8.0)	44.4 (3.0)	31.6 (1.3)
GPT4 _{text}	21.7 (2.0)	52.4 (6.0)	56.9 (5.0)	43.7 (2.9)
GPT4 _{vis+text}	28.3 (2.0)	57.1 (3.0)	58.3 (5.0)	48.0 (0.4)
GPT4O _{vis}	20.0 (0.0)	38.1 (3.0)	52.8 (3.0)	36.9 (1.6)
GPT4O _{text}	30.0 (0.0)	61.9 (3.0)	59.7 (3.0)	50.7 (1.7)
GPT4O _{vis+text}	30.0 (0.0)	61.9 (0.0)	66.7 (0.0)	53.0 (0.0)
LLAMACT:HoC	10.0 (0.0)	30.5 (2.0)	25.0 (0.0)	21.9 (0.7)
LLAMACT:HoC+MCQ	11.7 (4.0)	44.4 (5.0)	33.3 (5.0)	29.8 (1.2)
LLAMACT:HoC _{exp}	55.0 (4.0)	27.6 (3.0)	23.1 (2.0)	35.3 (0.9)
LLAMACT:HoC+MCQ _{exp}	40.0 (9.0)	43.5 (3.0)	36.1 (2.0)	40.0 (3.6)
LLAMACT:HoC+MCQ+AUG _{exp}	50.0 (4.0)	57.8 (1.0)	51.4 (3.0)	53.0 (1.7)
LLAMACT:HoC+MCQ+AUG _{exp} *	76.7 (7.0)	74.6 (4.0)	65.3 (0.0)	72.2 (1.3)
GRADEALL	74.1	50.9	58.5	61.2
GRADE7 _{top25}	99.8	84.0	71.4	85.1

Figure 7: Results on HoC, ACE, CT-TEST, and overall performance.

Technique	HoC	HoC reasoning	ACE	ACE reasoning	CT-TEST	CT-TEST reasoning
LLAMA3-8B-INSTRUCT	0.0 (0.0)	4.2 (1.0)	34.9 (2.0)	4.8 (0.0)	34.7 (5.0)	0.0 (0.0)
GPT4O _{vis+text}	30.0 (0.0)	35.0 (0.0)	61.9 (0.0)	28.6 (0.0)	66.7 (0.0)	39.6 (0.0)
LLAMACT:HoC+MCQ+AUG _{exp}	50.0 (4.0)	73.3 (8.0)	57.8 (1.0)	32.5 (3.0)	51.4 (3.0)	23.6 (3.0)

Figure 8: Comparison of accuracy in correctly answered tasks and reasoning correctness across domains for representative models in HoC, ACE, and CT-TEST tasks, reported as mean (stderr). Reasoning correctness results are based on manual annotations done by two independent annotators.

performance. Specifically, LLAMACT:HoC_{exp} and LLAMACT:HoC+MCQ_{exp}, which are trained with explanations, outperform their counterparts LLAMACT:HoC and LLAMACT:HoC+MCQ trained only for generating an answer with no explanation.

Fine-grained skills make LLAMACT comparable to GPT-4. We notice an increase of at least 10% in performance on HoC, ACE, and CT-TEST for the model fine-tuned with fine-grained skills data. Fine-tuning with explanations and across the full dataset allows LLAMACT:HoC+MCQ+AUG_{exp} to achieve overall results comparable to those of GPT4O_{vis+text}. This shows that a better understanding of the visual domain is key to better performance on all three tests.

Reasoning for MCQ tasks is harder than for solution synthesis tasks. We analyze the reasoning capabilities of three selected models, LLAMA3-8B-INSTRUCT, GPT4O_{vis+text}, and LLAMACT:HoC+MCQ+AUG_{exp}, through manual annotations of their reasoning process. A model’s reasoning is considered correct if it respects grid constraints, correctly maps codes to traces and sequences, and avoids introducing unnecessary details. We follow a strict binary metric, where the reasoning process is marked as correct (i.e., 1) if the entire reasoning process is correct and incorrect (i.e., 0) otherwise. The reasoning processes were reviewed by two independent annotators³. Figure 8 compares accuracy in correctly answered tasks with reasoning correctness averaged across annotators and aggregated over three seeds, for HoC, ACE, and CT-TEST tasks. LLAMA3-8B-INSTRUCT often tries guessing answers without providing any reasoning, while GPT4O_{vis+text} struggles with grid layouts, sometimes missing walls. LLAMACT:HoC+MCQ+AUG_{exp} traces tasks well in HoC but faces challenges with converting sequences to minimal codes and tracing in ACE and CT-TEST.

Symbolic information at inference leads to human-level performance. Including explanations with a correct reasoning process in the input prompts increases performance, bringing it closer to that of school students. However, LLAMACT:HoC+MCQ+AUG_{exp}* simulates an ideal scenario, as correct explanations are usually not available at inference as input.

³The annotators obtained a Cohen’s kappa score of 0.84, indicating high agreement [59].

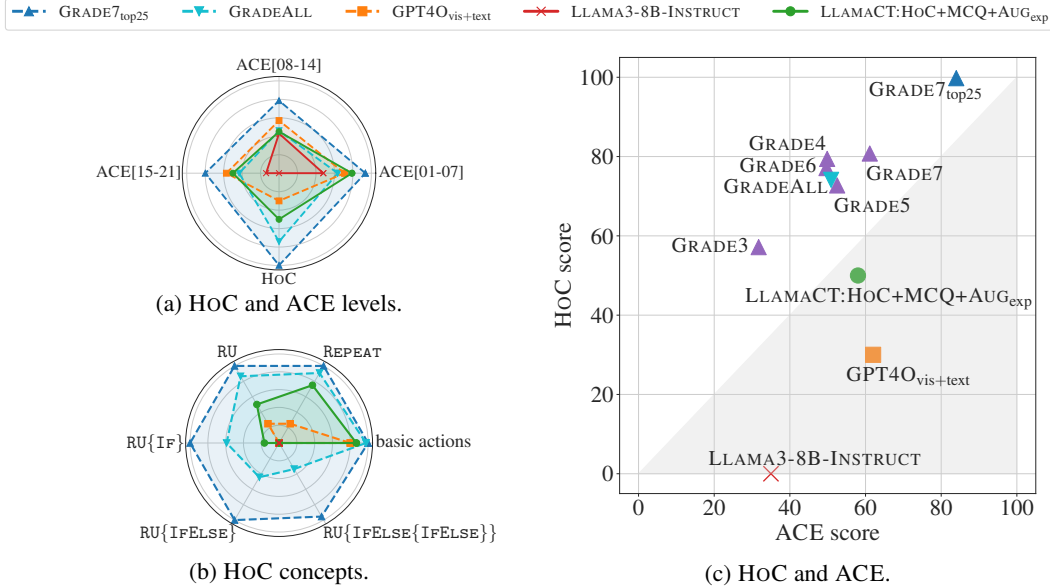


Figure 9: Comparison between the performance of the best techniques and school students (grade 3-7) on a scale of 0 to 100. For better visualization and comparison, we present results only for HOC and ACE. (a) shows that state-of-the-art and fine-tuned models have a similar performance to an average grade 3-7 student on the ACE, but lag behind for HOC. (b) shows that fine-tuning can help models’ problem-solving skills get closer to an average grade 3-7 student for simpler concepts. (RU stands for REPEATUNTIL). (c) shows how every grade is dominating models for HOC. It also shows that state-of-the-art and fine-tuned models are close to the average grade 7 students’ performance on ACE. However, the performance of the best 25% grade 7 students is still far from reach for generative models.

Human students are better at solution synthesis. Figure 9a showcases that state-of-the-art and fine-tuned models have slightly better performance than the average grade 3-7 student across three analyzed levels of Bloom’s taxonomy, and that state-of-the-art models struggle with solution synthesis. Figure 9b shows a deeper analysis of performance on HOC, breaking down the performance per concept. It shows that by fine-tuning, a model’s understanding of programming concepts grows similarly to that of an average student. Finally, Figure 9c compares models’ performance on HOC and ACE tests with that of students from various grades. It reveals that models have not yet reached the problem-solving capabilities of grade 3 students on HOC tasks. Besides spatial reasoning, adhering to constraints such as the required size and constructs is another reason for this weak performance. Interestingly, models can match the performance of grade 7 students on ACE tests, where answer options are available.

6 Concluding Discussion

In this paper, we introduced a new benchmark for assessing generative models on computational thinking tests grounded in elementary visual programming. We made a detailed analysis of the performance of open-access models such as Llama3 and the GPT family of models, comparing it to that of school students. To boost performance of Llama3-8B, we fine-tuned it using our novel synthetic generation methodology based on symbolic information. The best fine-tuned model has a performance similar to state-of-the-art models, even though it is much smaller and does not use vision capabilities.

While our analysis gives a deep insight into the computational thinking and problem-solving capabilities of generative models, there are some limitations of our current work and directions to tackle them in future work. First, we assess multi-modal models on our benchmark but do not fine-tune them to improve performance. An interesting direction for future work is fine-tuning multi-modal models for solving computational thinking and problem-solving tasks. Second, one of our techniques naively uses correct explanations provided at inference time to help it reach an answer. An interesting direction for future work is developing techniques where generative models interact with symbolic tools to obtain this kind of information at inference time, possibly via multiple rounds of interaction.

Acknowledgements

Funded/Co-funded by the European Union (ERC, TOPS, 101039090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

- [1] Karan Singhal et al. Large Language Models Encode Clinical Knowledge. *CoRR*, abs/2212.13138, 2022.
- [2] Chunyuan Li, Cliff Wong, Sheng Zhang, Naoto Usuyama, Haotian Liu, Jianwei Yang, Tristan Naumann, Hoifung Poon, and Jianfeng Gao. LLaVA-Med: Training a Large Language-and-Vision Assistant for Biomedicine in One Day. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2023.
- [3] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. BioGPT: Generative Pre-trained Transformer for Biomedical Text Generation and Mining. *Briefings in Bioinformatics*, 2022.
- [4] Anqi Wang, Zhizhuo Yin, Yulu Hu, Yuanyuan Mao, and Pan Hui. Exploring the Potential of Large Language Models in Artistic Creation: Collaboration and Reflection on Creative Programming. *CoRR*, abs/2402.09750, 2024.
- [5] Tyler Angert, Miroslav Ivan Suzara, Jenny Han, Christopher Lawrence Pondoc, and Hariharan Subramonyam. Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts. In *Proceedings of the Annual Symposium on User Interface Software and Technology (UIST)*, 2023.
- [6] Khan Academy. Khanmigo. <https://www.khanmigo.ai/>, 2023.
- [7] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Large Language Models for In-Context Student Modeling: Synthesizing Student’s Behavior in Visual Programming from One-Shot Observation. In *Proceedings of the International Conference on Educational Data Mining (EDM)*, 2023.
- [8] Paul Denny, Sumit Gulwani, Neil T. Heffernan, Tanja Käser, Steven Moore, Anna N. Rafferty, and Adish Singla. Generative AI for Education (GAIED): Advances, Opportunities, and Challenges. *CoRR*, abs/2402.01580, 2024.
- [9] Tung Phung, Victor-Alexandru Padurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint Validation. In *Proceedings of the Learning Analytics and Knowledge Conference (LAK)*, 2024.
- [10] OpenAI. Hello GPT-4o. <https://openai.com/index/hello-gpt-4o/>, 2024.
- [11] Meta. Llama 3. <https://llama.meta.com/llama3/>, 2024.
- [12] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations (ICLR)*, 2021.
- [13] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the Conference of the Association for Computational Linguistics (ACL) - Volume 1*, 2019.
- [14] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *CoRR*, abs/1803.05457, 2018.

- [15] Mark Chen et al. Evaluating Large Language Models Trained on Code. *CoRR*, abs/2107-03374, 2021.
- [16] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program Synthesis with Large Language Models. *CoRR*, abs/2108.07732, 2021.
- [17] OpenAI. GPT-4 Technical Report. 2023.
- [18] Jaromír Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. In *Proceedings of the Conference on International Computing Education Research (ICER) - Volume 1*, 2023.
- [19] Sébastien Bubeck et al. Sparks of Artificial General Intelligence: Early Experiments with GPT-4. *CoRR*, abs/2303.12712, 2023.
- [20] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.
- [21] Boshi Wang, Xiang Yue, and Huan Sun. Can ChatGPT Defend the Truth? Automatic Dialectical Evaluation Elicits LLMs’ Deficiencies in Reasoning. *CoRR*, abs/2305.13160, 2023.
- [22] Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2023.
- [23] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [24] Adish Singla. Evaluating ChatGPT and GPT-4 for Visual Programming. In *Proceedings of the Conference on International Computing Education Research (ICER) - Volume 2*, 2023.
- [25] Ahana Ghosh, Liina Malva, and Adish Singla. Analyzing-Evaluating-Creating: Assessing Computational Thinking and Problem Solving in Visual Programming Domains. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE)*, 2024.
- [26] Code.org. Hour of Code: Classic Maze Challenge. <https://studio.code.org/s/hourofcode>, 2013.
- [27] Code.org. Code.org: Learn Computer Science. <https://code.org/>, 2013.
- [28] Marcos Román González. Computational Thinking Test: Design Guidelines and Content Validation. In *Proceedings of the International Conference on Education and New Learning Technologies (EDULEARN)*, 2015.
- [29] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test. *Computers in Human Behavior*, 72, 2017.
- [30] Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, and Jing Ma. MMCode: Evaluating Multi-Modal Code Large Language Models with Visually Rich Programming Problems. *CoRR*, abs/2404.09486, 2024.
- [31] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. MathVista: Evaluating Math Reasoning in Visual Contexts with GPT-4V, Bard, and Other Large Multimodal Models. *CoRR*, abs/2310.02255, 2023.
- [32] Xiang Yue et al. MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI. *CoRR*, abs/2311.16502, 2023.

- [33] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring Coding Challenge Competence with APPS. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.
- [34] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. OctoPack: Instruction Tuning Code Large Language Models. *CoRR*, abs/2308.07124, 2023.
- [35] Tung Phung, Victor-Alexandru Padurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. In *Proceedings of the Conference on International Computing Education Research (ICER) - Volume 2*, 2023.
- [36] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Zhiyuan Liu, and Maosong Sun. DebugBench: Evaluating Debugging Capability of Large Language Models. *CoRR*, abs/2401.04621, 2024.
- [37] Tianyang Liu, Canwen Xu, and Julian J. McAuley. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems. *CoRR*, abs/2306.03091, 2023.
- [38] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.
- [39] Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng Lyu, Kevin Blin, Fernando Gonzalez Aduato, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Schölkopf. CLadder: A Benchmark to Assess Causal Reasoning Capabilities of Language Models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [40] Zhijing Jin, Jiarui Liu, Zhiheng LYU, Spencer Poff, Mrinmaya Sachan, Rada Mihalcea, Mona T. Diab, and Bernhard Schölkopf. Can Large Language Models Infer Causation from Correlation? In *International Conference on Learning Representations (ICLR)*, 2024.
- [41] Richard E. Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons, Inc., 1981.
- [42] Victor-Alexandru Pădurean, Georgios Tzannetos, and Adish Singla. Neural Task Synthesis for Visual Programming. *Transactions on Machine Learning Research (TMLR)*, 2024.
- [43] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. In *International Conference on Learning Representations (ICLR)*, 2018.
- [44] Umair Z. Ahmed, Maria Christakis, Aleksandr Efremov, Nigel Fernandez, Ahana Ghosh, Abhik Roychoudhury, and Adish Singla. Synthesizing Tasks for Block-based Programming. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- [45] Benjamin S Bloom, Max D Engelhart, Edward J Furst, Walker H Hill, and David R Krathwohl. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook 1: Cognitive Domain*. Longman New York, 1956.
- [46] David R Krathwohl. A Revision of Bloom’s Taxonomy: An Overview. *Theory into Practice*, 2002.
- [47] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive Learning from Complex Explanation Traces of GPT-4. *CoRR*, abs/2306.02707, 2023.
- [48] Arindam Mitra et al. Orca 2: Teaching Small Language Models How to Reason. *CoRR*, abs/2311.11045, 2023.

- [49] Ahana Ghosh, Sebastian Tschiatschek, Sam Devlin, and Adish Singla. Adaptive Scaffolding in Block-Based Programming via Synthesizing New Tasks as Pop Quizzes. In *Proceeding of the International Conference on Artificial Intelligence in Education AIED*, 2022.
- [50] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations (ICLR)*, 2022.
- [51] Wenpeng Yin, Jia Li, and Caiming Xiong. ConTinTin: Continual Learning from Task Instructions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL) - Volume 1*, 2022.
- [52] Jisoo Mok, Jaeyoung Do, Sungjin Lee, Tara Taghavi, Seunghak Yu, and Sungroh Yoon. Large-scale Lifelong Learning of In-context Instructions and How to Tackle It. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL) - Volume 1*, 2023.
- [53] Baptiste Rozière et al. Code Llama: Open Foundation Models for Code. *CoRR*, abs/2308.12950, 2023.
- [54] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [55] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [56] OpenAI. ChatGPT. <https://openai.com/blog/chatgpt>, 2023.
- [57] OpenAI. GPT-4V(ision) System Card. <https://openai.com/blog/chatgpt>, 2023.
- [58] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2022.
- [59] Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20:37 – 46, 1960.

Checklist

1. For all authors...
 - (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [\[Yes\]](#) See Sections 4 and 5.
 - (b) Did you describe the limitations of your work? [\[Yes\]](#) See Section 6.
 - (c) Did you discuss any potential negative societal impacts of your work? [\[No\]](#) We do not foresee any potential negative societal impacts of this work.
 - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [\[Yes\]](#)
2. If you are including theoretical results...
 - (a) Did you state the full set of assumptions of all theoretical results? [\[N/A\]](#)
 - (b) Did you include complete proofs of all theoretical results? [\[N/A\]](#)
3. If you ran experiments (e.g. for benchmarks)...
 - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [\[Yes\]](#) See the GitHub repository.
 - (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [\[Yes\]](#) See the supplemental material.
 - (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [\[Yes\]](#) See Section 5.
 - (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [\[Yes\]](#) See the supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
 - (a) If your work uses existing assets, did you cite the creators? [\[Yes\]](#) See Section 3 and the supplemental material.
 - (b) Did you mention the license of the assets? [\[Yes\]](#) See the supplemental material.
 - (c) Did you include any new assets either in the supplemental material or as a URL? [\[Yes\]](#) See the GitHub repository.
 - (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [\[N/A\]](#)
 - (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [\[N/A\]](#)
5. If you used crowdsourcing or conducted research with human subjects...
 - (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [\[N/A\]](#)
 - (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [\[N/A\]](#)
 - (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [\[N/A\]](#)