TurtleBench: A Visual Programming Benchmark in Turtle Geometry

Sina Rismanchian, Yasaman Razeghi, Sameer Singh, Shayan Doroudi
University of California, Irvine
{srismanc,yrazeghi,sameer,doroudis}@uci.edu

Abstract

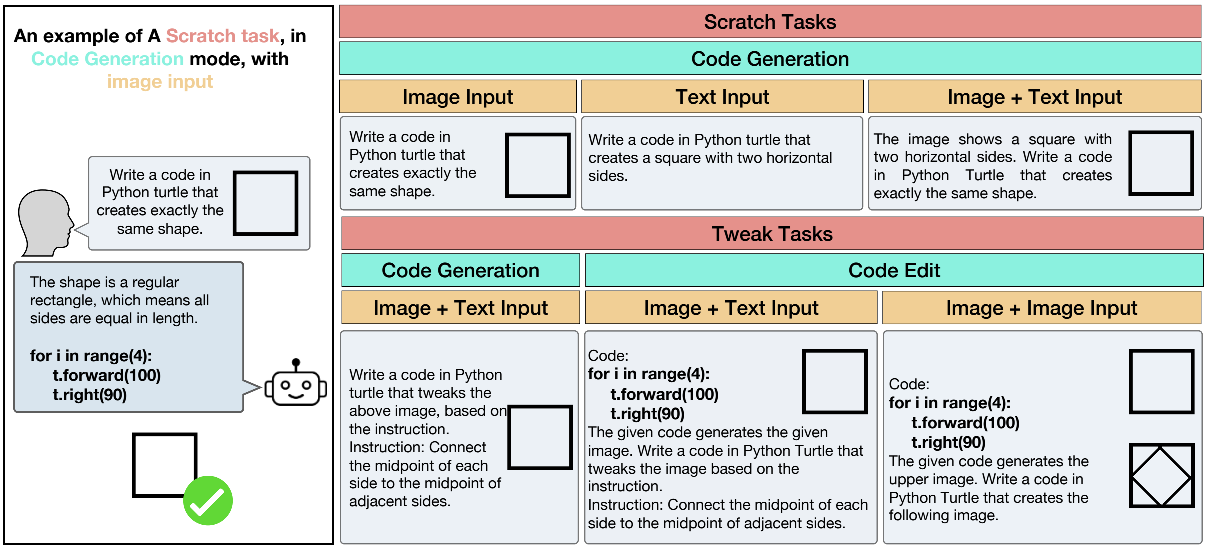
Humans have the ability to reason about geometric patterns in images and scenes from a young age. However, developing large multimodal models (LMMs) capable of similar reasoning remains a challenge, highlighting the need for robust evaluation methods to assess these capabilities. We introduce TurtleBench, a benchmark designed to evaluate LMMs' capacity to interpret geometric patterns—given visual examples, textual instructions, or both—and generate precise code outputs. Inspired by turtle geometry, a notion used to teach children foundational coding and geometric concepts, TurtleBench features tasks with patterned shapes that have underlying algorithmic logic. Our evaluation reveals that leading LMMs struggle significantly with these tasks, with GPT-4o achieving only 19% accuracy on the simplest tasks and few-shot prompting only marginally improves their performance ($< 2\%$). TurtleBench highlights the gap between human and AI performance in intuitive and visual geometrical understanding, setting the stage for future research in this area. TurtleBench stands as one of the few benchmarks to evaluate the integration of visual understanding and code generation capabilities in LMMs, setting the stage for future research. Code and Dataset for this paper is provided here: https://github.com/sinaris76/TurtleBench

1 Introduction

Large Multimodal Models (LMMs) have the potential to handle tasks that combine visual, linguistic, and reasoning abilities, previously achievable only by humans. Indeed, LMMs such as GPT4-V (Yang et al., 2023) and Gemini 1.5 flash (Team et al., 2023b; Fu et al., 2023) have been shown to be state-of-the-art models in solving multi-modal tasks such as visual question answering (Goyal et al., 2017; Liu et al., 2024), coding for visual multimodal questions (Li et al., 2024a), visual mathematical questions (Lu et al., 2023), chart question answering (Masry et al., 2022), etc. Despite these successes, there remains the question of how LMMs perform in tasks that intertwine visual reasoning and programming knowledge. That is, given an image of a geometric pattern (and/or a verbal description of the pattern) can LMMs generate code that would be able to procedurally generate that pattern? Indeed, Bubeck et al. (2023) showed that the large language model (LLM) with no visual training data was able to create a unicorn in TikZ—the LaTeX-based graphics drawing library. This feat amazed many and provoked many discussions on the intelligence of large language models—but how general is this ability?

In this work, we introduce TurtleBench, a set of manually crafted image/text to code tasks in turtle geometry (Papert, 1972; Abelson and diSessa, 1986) to evaluate the abilities of these models to combine visual pattern recognition, mathematical reasoning, Python programming, and abstract geometrical reasoning. To ensure the visual inputs and the programming language remain straightforward, TurtleBench harnesses turtle geometry, a concept widely recognized for its effectiveness in introducing programming concepts to children within the K-12 education system. In turtle geometry, a turtle acts as a programmable object that navigates the screen, drawing as it goes and turning at specified angles, to create simple visual patterns. The primary objective within this framework is to generate code capable of producing simple visual inputs. These visual inputs consist of basic geometric shapes, and the programming syntax required is intentionally limited and straightforward. An example of such a task is presented in the left side of Figure 1. As illustrated, the input image is the shape of a simple square and the corresponding code only uses two simple turtle functions (`forward` and `right`) along with a simple for loop. This simplicity makes TurtleBench an effective benchmark for evaluating the capabilities of Large Multimodal

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 12170–12188
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics



Write a code in Python turtle that creates exactly the same shape.

The shape is a regular rectangle, which means all sides are equal in length.

```
for i in range(4):
    t.forward(100)
    t.right(90)
```

Write a code in Python turtle that creates exactly the same shape.

Write a code in Python turtle that creates a square with two horizontal sides.

The image shows a square with two horizontal sides. Write a code in Python Turtle that creates exactly the same shape.

Write a code in Python turtle that tweaks the above image, based on the instruction. Instruction: Connect the midpoint of each side to the midpoint of adjacent sides.

Code:

```
for i in range(4):
    t.forward(100)
    t.right(90)
```

The given code generates the given image. Write a code in Python Turtle that tweaks the image based on the instruction. Instruction: Connect the midpoint of each side to the midpoint of adjacent sides.

Code:

```
for i in range(4):
    t.forward(100)
    t.right(90)
```

The given code generates the upper image. Write a code in Python Turtle that creates the following image.

Figure 1: An illustration of existing types and modes in TurtleBench.

Models (LMMs). To reflect different real-world use cases of an LMM in the domain of Turtle and also cover the broad range of underlying reasoning abilities, TurtleBench includes 260 tasks with a variety of types and modalities. The different types and input/output modalities are presented in Figure 1. We define two types of tasks scratch and tweak in TurtleBench. Scratch tasks challenge models to *generate* Python code for a specified shape using the Turtle library based on inputs that could be an image of the shape, a textual description, or both. This subset evaluates model's proficiency in recognizing patterns within the shape and accurately translating these into executable code. Conversely, tweak tasks are designed to probe deeper into a model's understanding, examining its ability to comprehend the implications of described modifications to shapes—such as connecting midpoints (example in Figure 1)—and their representation in the image. Here, models are provided with a base shape and are *instructed* to create the desired alteration in shape. Instructions for these modifications may be provided visually or textually. To simplify, in a subset of the tweak tasks, we adopt a *code editing* approach, supplying the original shape's code and directing the model to *edit* this code to generate the target shape.

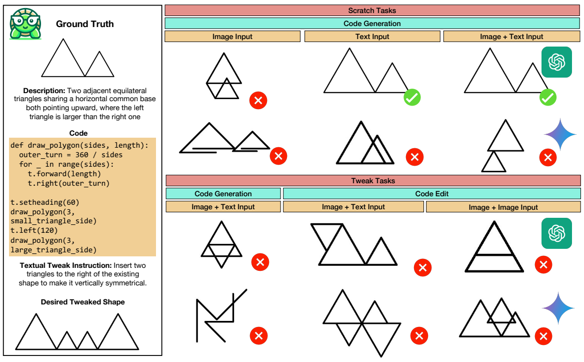
We conduct an evaluation of leading LMMs on TurtleBench code generation and code editing tasks, utilizing zero-shot and visual chain-of-thought approach (Singh et al., 2023) across text-only, image-only, and mixed (text and image) input modalities. Our findings reveal that these models generally perform poorly across all setups and variety of tasks and modalities. Both GPT-4o and Gemini 1.5 Flash struggle with TurtleBench tasks, failing to solve more than 75% of them. Our results show that performance improves when tasks are presented as text rather than images, suggesting that integrating visual and linguistic information, especially for visual pattern recognition, requires further refinement. When tested with a custom library mimicking Python Turtle but using different command names, models showed a significant performance drop, revealing difficulties in generalizing visual reasoning to unfamiliar syntax. Even when allowed to choose their own programming language, models consistently failed to generate correct code, indicating broader challenges in translating visual instructions into functional programming outputs. These findings show that our benchmark poses a significant challenge for LMMs, offering key insights into their limitations. Our evaluation highlights gaps in integrating visual reasoning with programming and raises important questions for future research to address.

2 Overview of TurtleBench

In turtle geometry (Abelson and diSessa, 1986), a turtle is a programmable object on the screen that leaves a trace while moving. As illustrated in Figure 1 left side, we see an example of how creating a simple geometric shape, like a square, involves the turtle moving forward and executing turns four times. It is a powerful, intuitive tool that enables novice learners to start learning programming by creating aesthetically beautiful artifacts. Although Turtle programming nowadays is used more as a tool to foster computational thinking, it can be used to teach geometry and mathematical reasoning (Marji, 2014; Clements and Sarama, 2013) as it enables learners to explore and learn geometrical relationships between shapes. The intuitive nature and learnability of turtle geometry, along with its ability to generate patterns of diverse complexities, make it a compelling concept upon which to base a benchmark for LMMs. In the following, we describe our benchmark in detail.

2.1 TurtleBench Task Types

TurtleBench is a set of 260 tasks that are designed to evaluate LMMs' performance on vision and language algorithmic reasoning tasks. To ensure the novelty of the tasks and their quality in incorporating authentic geometric shapes and concepts, we craft TurtleBench manually. All the tasks in TurtleBench are accurately solvable based on the provided information for each, which means that there are no ambiguities or arbitrary parameters leading to inaccuracies in the tasks for humans as well as the models. To remove possible ambiguities in the tasks, two independent annotators worked with us to identify and resolve any unclear instructions. Each task consists of a black-and-white image illustrating a set of abstract geometric shapes as an *input*. An example of this task is presented in Figure 1. TurtleBench is made up of two different types of tasks, these types reflect the methodologies used in turtle geometry to introduce programming to children: Scratch tasks that are intended to show how well a model understands a pattern and translates its understanding to an executable code. In the general case of this type of task, an image is provided, and the requested output is code in Python Turtle that creates the shapes in the image. In all scratch tasks, the model is asked to *generate* the code in Python Turtle for the desired input shape. TurtleBench includes a total of 130 scratch tasks and 130 tweak tasks resulting in 260 tasks overall. An example of these tasks is provided in Figure 1 top rows. To distinguish between the models' visual comprehension and their textual understanding, a subset of these tasks includes a text description of the image input in addition to the visual representation. This setup facilitates the evaluation of how models respond differently to visual and textual inputs, providing a clearer understanding of their capabilities. Tweak tasks that are intended to measure how well a model uses their understanding of a visual pattern, combined with an instruction to make minimal alterations. Each tweak task presents a model with an image and an instruction; the expected output is Python Turtle code that modifies the shape in the input image according to the given instruction. These tasks are particularly insightful for determining whether a model is merely recalling memorized code for an image, or if it has developed a deeper, more human-like comprehension of the patterns depicted in the images. For instance, a model might be capable of generating code for a certain shape based on training data, but the real challenge lies in its ability to adapt that shape in response to various instructed changes. An example of these tasks is provided in Figure 1 bottom row. Here, The model is given an input image of a rectangle, with an instruction to *connect the midpoint of each side to the midpoint of adjacent sides*. As illustrated in Figure 1, we also introduce a code editing version of the tweak task. In this version, we supply the code corresponding to the input image and then instruct the models to make specific modifications to this code, aiming to achieve a change in the image as per the provided instructions. Detailed information about types of tweaks and their examples is provided in Appendix A.4.



Description: Two adjacent equilateral triangles sharing a horizontal common base both pointing upward, where the left triangle is larger than the right one

Code

```
def draw_polygon(sides, length):
  outer_turn = 360 / sides
  for _ in range(sides):
    t.forward(length)
    t.right(outer_turn)

t.setheading(60)
draw_polygon(3,
small_triangle_side)
t.left(120)
draw_polygon(3,
large_triangle_side)
```

Textual Tweak Instruction: Insert two triangles to the right of the existing shape to make it vertically symmetrical.

Desired Tweaked Shape

Figure 2: An illustration of different modes of a single task in TurtleBench along with the images generated by code from the outputs of GPT-4o and Gemini 1.5 Flash. More examples are provided in Appendix Figure 10

2.2 Automatic Evaluation of Code Output

Evaluation of the output code by an AI model is performed automatically. First, the output of the AI model is processed to extract the code piece of output. Then, this piece of code is run in a sandbox, and the shape produced by the code is stored. An illustration of this pipeline is provided in Figure

9. Finally, using the *OpenCV* module in Python, the binary versions of the correct shape and the produced shape are compared using an adjusted measure of bitwise similarity where we first use the bounding box technique with *OpenCV* to find the exact location of the shape and then calculate similarity with the formula:

$$\frac{|B_a \cap B_m|}{|B_a \cup B_m|}$$

where B_a and B_m represent black pixels in the input and LMM output, respectively. This metric measures the ratio of co-occurring black pixels to the total black pixels Here, we utilize a heuristic approach in labeling the correctness of the model's output. If the bitwise similarity between output and ground truth is higher than 95% the models' output is labeled as correct and incorrect otherwise. To make sure that our heuristic in labeling the correctness of generated shapes is reliable, we manually annotated 2000 pairs of input and output images and we found that only three instances of pairs were labeled incorrectly (two of them false negative and the other false positive.), leading to an error rate of 0.15% which shows the high level of reliability in the heuristic we used.

3 Evaluation Setup

3.1 Models

In the following section, we evaluate TurtleBench using two SOTA LMMs, GPT-4o and Gemini 1.5 Flash and also an open sourced model, namely Llava-1.5-13B (Liu et al., 2023) employing greedy decoding in our evaluations. We evaluated two other open models, namely Qwen-VL-Max (Bai et al., 2023) and CogVLM (Wang et al., 2023) on a subset of tasks in TurtleBench. However, CogVLM and Qwen are not successful in producing a syntactically correct Python Turtle piece of code even for the simplest tasks, therefore we limited our benchmark evaluation to models mentioned above.

3.2 Prompting

We use two types of prompting in our experiments, 1) basic, where we simply prompt the the model (c.f. Appendix A.2) to do our tasks. , and 2) Chain-of-Thought (CoT) prompting (Wei et al., 2022), which has shown to be an effective prompting technique in eliciting reasoning in these models. Specifically, we use a more detailed version of CoT prompting that is tailored to LMMs, namely v-CoT, recently proposed by (Singh et al., 2023). The v-CoT approach is inspired by m-CoT (Zhang et al., 2023), which shows higher performance compared to it. This prompting has been shown to improve LMMs' performance on visual tasks that involved reasoning, such as ARC (Chollet, 2019). This prompt, instructs the model to first extract all the relevant information in the image needed for answering the problem and then to reason step by step based on the information extracted. The specific prompt we used in our experiments is in Appendix A.2

4 Results

Results on the performance of the models are reported in percentage, where in each experiment we evaluate the performance of select models on all instances of the TurtleBench with test@1 method (Cobbe et al., 2021) where the model generates only one piece of code for each instance. We assign a binary value to the success/failure of each task. We then run each experiment on a model five different times and we report the average percentage of accumulative success.

4.1 Models perform poorly on TurtleBench

We initially examine the performance of the GPT-4o, Gemini 1.5 Flash and Llava-1.5-13B models on the comprehensive TurtleBench dataset. The findings, detailed in Table 1, reveal a notably poor performance across the tasks in TurtleBench, with a peak accuracy of 20% achieved by GPT-4o in the *code editing* tasks, facilitated by Chain of Thought (CoT) prompting. In the *scratch* tasks, which represent the simplest problem type within the dataset, GPT-4o's success rate was just 19%, underscoring the substantial challenges and complexities these tasks pose to the current models. A comparison between CoT and basic prompting within Table 1 illustrates that CoT prompting outperforms basic prompting on the same models, aligning with previous work that indicates CoT enhances models' reasoning abilities (Zhang et al., 2023). However, despite employing CoT, the task remains far from being solved. Examples of model output in different subsets of the task are provided in Figures 2 and 10.

As previous work suggests that in-context learning by providing examples of the task at hand can significantly improve models' performance in do-

Task Type	*Scratch* CG			*Tweak* CG	*Tweak* CE		Runnable
Modalities	T	I	I + T	I + T	I + T	I + I	
GPT-4o/basic	37.04	16.03	**37.98**	17.69	18.12	12.06	99.21
GPT-4o/CoT	38.12	19.23	**40.18**	20.00	19.61	13.84	99.85
GPT-4o/4-S	NA	21.49	NA	NA	NA	NA	99.85
Gemini/basic	**25.09**	7.71	22.22	3.85	12.00	3.00	99.13
Gemini/CoT	18.51	9.20	**20.52**	7.70	23.08	11.54	99.17
Gemini/4-S	NA	10.18	NA	NA	NA	NA	99.92
Llava/basic	**6.01**	0.82	0.03	0.31	1.04	NA	69.13
Llava/CoT	**6.22**	0.98	1.02	0.92	1.09	NA	72.34

Table 1: Performance on TurtleBench (I = image, T = text, CG = code generation, CE = code edit, 4-S = 4-shot). Performance on *Scratch* tasks that include text (T) is calculated on a subset (21%) of these tasks. Our result shows that while models' generated code is almost always runnable, they fail to generate desired shapes.

main adaptation (Brown, 2020), we evaluated the performance of GPT-4o and Gemini 1.5 Flash on *Scratch* code generation with 4-shot CoT prompting where we first provided four pairs of shape-code to the model. Yet, we did not find any major improvement in the performance of the models.

4.2 🐰 Models fail to generalize

Given that these models have been extensively trained on vast datasets sourced from the internet, there's an underlying uncertainty regarding the source of their performance—albeit poor—on the TurtleBench tasks. Specifically, it remains unclear whether this performance is the result of the models' ability to memorize aspects of our tasks, rather than genuinely understanding and solving them based on their programming and reasoning capabilities. To address this issue, our next step is to evaluate the true generalization ability of these models. By doing so, we aim to distinguish between superficial learning, potentially influenced by memorization, and genuine comprehension and problem-solving skills. To measure the generalizability of the model's performance, we define an arbitrary set of commands based on the turtle module in Python. In other words, we developed a class called Rabbit that inherits the `Turtle` class from the turtle module. Although the functions of the Rabbit class are functionally identical to those in the original turtle module, they are nominally distinct. This differentiation allows us to evaluate the models' ability to apply their knowledge to unfamiliar yet equivalent command sets. The definition of the Rabbit class in Python is provided in Appendix A.3.2. We perform a zero-shot CoT prompting to elicit the code using the new set of commands. In the context window, we provide a verbal definition of each function in the Rabbit class. The results of comparing the models' performance using the Rabbit class versus the standard Python Turtle module are presented in Table 2. We observe that, although both models were capable of generating executable pieces of code with the new class, there is a huge decline in their performance relative to their performance with the conventional Python Turtle module. This finding suggests that the visual reasoning in these models is not robust to syntax changes, and it is likely that they rely on training memorization rather than pure reasoning.

	🐢 GPT-4o	🐰 GPT-4o	🐢 Gemini	🐰 Gemini
Scratch Code Generation				
I	19.23	6.00	9.20	3.00
Tweak Code Generation				
I + T	20.00	5.04	7.70	2.37

Table 2: Performance of GPT-4o and Gemini 1.5 Flash on generalization tasks using CoT prompting, in these tasks. The performance in Rabbit drastically drops, showing poor generalization abilities in both models.

4.3 Assessing Model Proficiency Across Programming Languages

The initial suspicion might be that the models struggle with tasks in turtle geometry due to a lack of exposure to specific programming syntax during pretraining: Is the poor performance because of unfamiliarity with Python Turtle?

To answer the question, we run an ablation study

Output	Python Turtle	Any	Matplotlib
Scratch Code Generation			
I	19.23	21.6	22.15
Tweak Code Generation			
I + T	12.3	15.11	15.23

Table 3: Ablation study of CoT prompting of GPT-4o on code generation tasks where the model is given the freedom to choose the programming language. Producing code in Matplotlib does not yield qualitatively better performance.

on GPT-4o as our best-performing model in the main tasks. We allow it to generate code using any library, language, or similar tools it deems appropriate, such as Matplotlib, TikZ, etc., without restricting it to the Python Turtle library. The prompt for this subset of tasks is presented in Appendix A.2.4. We manually evaluate the GPT-4o output for this task. Despite this freedom, we observe no significant improvement in performance. The model chooses Matplotlib for 50% of the tasks and offers pseudocode for 2%, with the remainder reverting to Python Turtle, even though we do not specify Python Turtle in the prompts. Notably, it avoids using TikZ, despite its mention in the prompt and proven capabilities in prior work to produce TikZ code (Bubeck et al., 2023; Belouadi et al., 2023). We further isolate this observation by obligating the model to produce code using Matplotlib (refraining from generating pseudocode and Python Turtle) and in this experiment as well, we do not see a major improvement in the model's results. This outcome underscores a deeper issue than syntax familiarity: the models' fundamental challenge is accurately interpreting visual input and applying this understanding to generate corresponding programming code.

4.4 Limited Visual Understanding in LMMs: Insights from Textual vs. Visual Tweak Tasks

To distinctly assess the models' proficiency in interpreting textual versus visual information, we conducted an evaluation focusing on their ability to reason about the relationship between provided code and corresponding images in *Tweak* code edit tasks. In this setup, models are given a base shape along with the Python Turtle code that generated it. Subsequently, they are prompted to adjust the given code to modify the shape according to specified instructions. These instructions are delivered in two forms: 1) (I + T) as natural language descriptions, e.g., 'connect the midpoints of each side to the midpoints of its adjacent sides,' and 2) (I + I) as images explicitly showcasing the desired outcome. Examples of these subsets are in Figure 1. Our comparison of model performance across these two modalities reveals a huge decline in accuracy when instructions were provided visually rather than textually (Table 1). This outcome suggests a disparity in the models' ability to process visual versus textual instructions, revealing that their reasoning abilities may not align closely with human-like understanding. The assumption that directly viewing the desired outcome simplifies the task contrasts sharply with our findings, highlighting a reliance on textual interpretation for reasoning and a notable limitation in pure visual reasoning capabilities within these models (Roberts and Roberts, 2024).

4.5 Vision component contributes poorly

One of the questions regarding LMMs' abilities in visual abstraction and understanding tasks is the extent the incorporation of the visual component has enhanced their abilities in reasoning (Mitchell et al., 2023).

In resonance with what Mitchell et al. (2023) found, here we also found that the vision component contributes poorly to fostering the models' visual reasoning abilities, at least in the domain of TurtleBench. Specifically, we annotated 27 (21%) *Scratch* code generation tasks and provided clear descriptions for each in plain text (Textual descriptions were validated by two human annotators, one in the research team and the other recruited as a volunteer). The remaining shapes were too complex to describe without ambiguity in plain text. Then, we compared the three modes of presenting the task, image only, text only, and the blend of an image and its textual description (I, T, and I + T, respectively in Table 1). Interestingly, for both GPT-4o and Gemini 1.5 Flash, the model performed worse when the task was presented only in the image, compared to the other modes. This phenomenon is counterintuitive as for humans, perceiving the images should be easier than first reading a description, imagining it, and then writing a code for it. Additionally, as presented in Table 1 the blend of image and text only slightly improved GPT-4o's performance (from 38% to 40%). These two findings show that there is still much room for improvement especially in the visual components of LMMs.

5 Related Work

5.1 Large Multi-modal Models

Recent advancements in foundational multimodal models have marked a significant stride towards developing generalist AI systems capable of understanding and integrating information across different modalities to solve tasks without the need for task-specific fine-tuning. Among these models are closed source models such as Gemini 1.5 Flash (Team et al., 2023a), GPT-4o (OpenAI et al., 2024), and open source models as LLaVA-1.5 (Liu et al., 2023), Mini-GPT4 (Zhu et al., 2023), InstructBLIP (Dai et al., 2023) and CogVLM (Wang et al., 2024). The versatility and multimodal understanding exhibited by these foundational multimodal models have positioned them as prime candidates for applications such as AI software engineers or programming tutors for children. Our work evaluates the efficacy of these popular models on image/text-to-code tasks, measuring their potential in vision/programming context.

5.2 Probabilistic Program Induction

Recent work in Bayesian cognitive science has modeled various aspects of cognition and learning as probabilistic program induction (Lake et al., 2015; Lake and Piantadosi, 2020; Rule et al., 2020; Ellis et al., 2023; Wong et al., 2021; Grand et al., 2023). This has involved both modeling human cognition as program induction as well as designing machine learning algorithms that can generate programs for various tasks, including the kind of turtle geometry task we study here. Ellis et al. (2023) developed the DreamCoder algorithm which can learn to induce programs by using self-supervision to incrementally build up a library of programs and train a neural network to search to find the best program for a given task. They created a dataset of 160 turtle programming tasks. In contrast to our approach, where we assess the performance of out-of-the-box LMMs, DreamCoder is trained on a training set of images (i.e., half of the dataset). However, it is interesting that the algorithm is trained in an unsupervised fashion; that is, DreamCoder never receives the code used to generate the images and learns that from experience. Wong et al. (2021) extended this work by developing an algorithm (LAPS) that can induce programs given both the task and linguistic annotations for the task. They used a dataset of 311 turtle graphics with greater complexity than the original DreamCoder dataset. While their dataset includes linguistic annotations, their dataset does not include tweak tasks like in TurtleBench. Additionally, their tasks often include arbitrary aspects (for example, a gap with unspecified distance between two shapes) that makes evaluation hard; in our tasks, the positional relationships between shapes should be easy to infer exactly and hence we can evaluate models by comparing exactly with ground truth shapes. Moreover, neither of these datasets have been framed as a benchmark for visual program induction and have not been considered for evaluating LMMs. Perhaps the approach closest to our work is by Grand et al. (2023), who combined LLMs with a symbolic program induction algorithm and evaluated the performance of their model (LILO) on the turtle geometry task using the aforementioned dataset. Averaged over several runs, the performance of the best versions of these approaches on the turtle geometry task is as follows: 43% for DreamCoder, 82% for LAPS, 49% for LILO, and 32% for a LLM solver. These results seem to suggest that probabilistic programming approaches (such as LAPS) can greatly outperform LMMs on visual programming tasks. We note that the performance of the LLM solver (32%) is comparable to the performance of GPT-4o on our text-only input (37%; see Table 1). Future work could assess the performance of probabilistic program induction methods like LAPS on TurtleBench.

5.3 Multimodal Reasoning

The existing literature features a range of studies that evaluate these models using naturalistic images (Jiang et al., 2022; Johnson et al., 2017; Antol et al., 2015), yet humans naturally are able to reason over abstract shapes (Chollet, 2019; Zhang et al., 2019; Spelke and Kinzler, 2007) and also many use cases of LMMs involve understanding abstract shapes and sketches (Forbus et al., 2011; Nie et al., 2020). Moreover, unlike naturalistic images (Marjieh et al., 2022; Sucholutsky and Griffiths, 2024), the relationship between language and abstract shapes is highly intertwined as minimal alterations in language can lead to different visual perceptions in humans (Dillon, 2023; Lin and Dillon, 2023). Overall, recent surveys on deep learning for mathematical reasoning (Lu et al., 2022; Sun et al., 2023) have pointed out that most of the available datasets sand benchmarks on multimodal reasoning often rely on visual question-answering frameworks. However, these methods fall short because they are usually trained on datasets composed of natural images, rather than on datasets tailored to the integration of vision and language for mathematical tasks.

The Multimodal Algorithmic Reasoning (MAR) task tests multi-modal models on fundamental skills understandable by children, focusing on interpreting visual and linguistic information to answer questions. Perhaps the most relevant work to ours is the paper by (Cherian et al., 2023) in which they introduced a dataset with 101 multiple-choice questions inspired by the Math Kangaroo contest for 6 to 8-year-olds, involving images and texts that the model must analyze together. The task has been shown to be challenging for multimodal deep neural networks, and the following trials to solve the problem have gained less than 25% accuracy on the test set (Wu et al., 2023). TurtleBench includes abstract geometric shapes, and the task only relies on knowledge and reasoning over a set of simple functions in the Python Turtle library. Our open-ended benchmark and its flexibility over different modalities make evaluating different aspects of multimodal reasoning in LMMs more reliable.

6 Discussion on Educational Implications

While LLMs and LMMs have sparked interest among education researchers in AI tools such as tutors (KhanAcademy, 2024) for students, our work cautions against using these models without thorough evaluation. Although students can learn turtle programming on various platforms without AI help (e.g., Code.org (2024a,b)), the effectiveness of these models as tutors or copilots is uncertain due to current limitations. Our work suggests that educational researchers can engage in systematically benchmarking LMMs to ensure their reliability before integrating them into student learning processes.

7 Conclusions

This study introduces TurtleBench, the first of its kind in benchmarks that focus on converting visual inputs to code outputs. The evaluation results from TurtleBench reveal a significant disparity between how humans tackle turtle programming and how SOTA AI models perform in understanding simple geometric shapes, reasoning about these shapes, and converting such understandings into executable code (Rismanchian et al., 2024). This gap underscores the challenges that lie ahead in the quest to enhance AI's comprehension and problem-solving abilities to match human levels. We believe that TurtleBench serves as a crucial tool in the evaluation of models, offering a clear benchmark testing the limits of LMMs.

8 Limitations

One of the limitations of our work is that we did not experiment with fine-tuning techniques to better understand multimodal reasoning abilities in these models and how they could be improved. However, we argue that our experiments demonstrate that poor performance in the models is perhaps not due to their unfamiliarity with the syntax of Turtle, but rather is more related to vision components and their reasoning abilities. We plan to experiment with fine-tuning techniques in future work with smaller models such as Llava-1.5-13B with newer architectures for vision towers (Li et al., 2024b) to examine the effectiveness of these techniques.

Finally, as TurtleBench has a limited number of instances, future work can augment existing instances to produce a dataset plausible for training purposes.

Acknowledgements

We are grateful to Arghavan Rezvani for her valuable assistance and insightful contributions.

References

Harold Abelson and Andrea diSessa. 1986. *Turtle geometry: The computer as a medium for exploring mathematics*. MIT press.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. 2023. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. *arXiv preprint arXiv:2310.00367*.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Anoop Cherian, Kuan-Chuan Peng, Suhas Lohit, Kevin A Smith, and Joshua B Tenenbaum. 2023. Are deep neural networks smarter than second graders? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10834–10844.

François Chollet. 2019. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*.

Douglas H Clements and Julie Sarama. 2013. Children's mathematical reasoning with the turtle programming metaphor. In *Mathematical Reasoning*, pages 313–337. Routledge.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Code.org. 2024a. Code with anna and elsa - code.org. `https://studio.code.org/s/frozen`. (Accessed on 10/15/2024).

Code.org. 2024b. Self paced introduction to turtle programming in app lab - code.org. `https://studio.code.org/s/csp3-virtual`. (Accessed on 10/15/2024).

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Preprint*, arXiv:2305.06500.

Moira R Dillon. 2023. Divisive language. *Preprint*, 10.31234:1706.03762.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. 2023. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220050.

Kenneth Forbus, Jeffrey Usher, Andrew Lovett, Kate Lockwood, and Jon Wetzel. 2011. Cogsketch: Sketch understanding for cognitive science research and for education. *Topics in Cognitive Science*, 3(4):648–666.

Chaoyou Fu, Renrui Zhang, Haojia Lin, Zihan Wang, Timin Gao, Yongdong Luo, Yubo Huang, Zhengye Zhang, Longtian Qiu, Gaoxiang Ye, et al. 2023. A challenger to gpt-4v? early explorations of gemini in visual expertise. *arXiv preprint arXiv:2312.12436*.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913.

Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X Olausson, Muxin Liu, Joshua B Tenenbaum, and Jacob Andreas. 2023. Lilo: Learning interpretable libraries by compressing and documenting code. *arXiv preprint arXiv:2310.19791*.

Huaizu Jiang, Xiaojian Ma, Weili Nie, Zhiding Yu, Yuke Zhu, and Anima Anandkumar. 2022. Bongard-hoi: Benchmarking few-shot visual reasoning for human-object interactions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19056–19065.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910.

KhanAcademy. 2024. Khanmigo: Khan academy's ai-powered teaching assistant & tutor. `https://www.khanmigo.ai/`. (Accessed on 10/15/2024).

Brenden M Lake and Steven T Piantadosi. 2020. People infer recursive visual concepts from just a few examples. *Computational Brain & Behavior*, 3(1):54–65.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. 2024a. Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 736–783.

Wenhao Li, Yudong Xu, Scott Sanner, and Elias Boutros Khalil. 2024b. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. *arXiv preprint arXiv:2410.06405*.

Yi Lin and Moira R Dillon. 2023. We are wanderers: Abstract geometry reflects spatial navigation. *Journal of Experimental Psychology: General*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023. Improved baselines with visual instruction tuning. *Preprint*, arXiv:2310.03744.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2022. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*.

Majed Marji. 2014. *Learn to program with Scratch: A visual introduction to programming with games, art, science, and math*. No Starch Press.

Raja Marjieh, Pol Van Rijn, Ilia Sucholutsky, Theodore R Sumers, Harin Lee, Thomas L Griffiths, and Nori Jacoby. 2022. Words are all you need? language as an approximation for human similarity judgments. *arXiv preprint arXiv:2206.04105*.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland. Association for Computational Linguistics.

Melanie Mitchell, Alessandro B Palmarini, and Arseny Moskvichev. 2023. Comparing humans, gpt-4, and gpt-4v on abstraction and reasoning tasks. *arXiv preprint arXiv:2311.09247*.

Weili Nie, Zhiding Yu, Lei Mao, Ankit B Patel, Yuke Zhu, and Anima Anandkumar. 2020. Bongard-logo: A new benchmark for human-level concept learning and reasoning. *Advances in Neural Information Processing Systems*, 33:16468–16480.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong

Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Seymour Papert. 1972. On making a theorem for a child. In *Proceedings of the ACM annual conference-Volume 1*, pages 345–349.

Sina Rismanchian, Shayan Doroudi, and Yasaman Razeghi. 2024. Turtle-like geometry learning: How humans and machines differ in learning turtle geometry. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 586–587.

Denisa Roberts and Lucas Roberts. 2024. Smart vision-language reasoners. *arXiv preprint arXiv:2407.04212*.

Joshua S Rule, Joshua B Tenenbaum, and Steven T Piantadosi. 2020. The child as hacker. *Trends in cognitive sciences*, 24(11):900–915.

Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, and Gust Verbruggen. 2023. Assessing gpt4-v on structured reasoning tasks. *arXiv preprint arXiv:2312.11524*.

Elizabeth S Spelke and Katherine D Kinzler. 2007. Core knowledge. *Developmental science*, 10(1):89–96.

Ilia Sucholutsky and Tom Griffiths. 2024. Alignment with human representations supports robust few-shot learning. *Advances in Neural Information Processing Systems*, 36.

Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, et al. 2023. A survey of reasoning with foundation models. *arXiv preprint arXiv:2312.11562*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, Alexandre Frechette, Charlotte Smith, Laura Culp, Lev Proleev, Yi Luan, Xi Chen, James Lottes, Nathan Schucher, Federico Lebron, Alban Rrustemi, Natalie Clay, Phil Crone, Tomas Kocisky, Jeffrey Zhao, Bartek Perz, Dian Yu, Heidi Howard, Adam Bloniarz, Jack W. Rae, Han Lu, Laurent Sifre, Marcello Maggioni, Fred Alcober, Dan Garrette, Megan Barnes, Shantanu Thakoor, Jacob Austin, Gabriel Barth-Maron, William Wong, Rishabh Joshi, Rahma Chaabouni, Deeni Fatiha, Arun Ahuja, Ruibo Liu, Yunxuan Li, Sarah Cogan, Jeremy Chen, Chao Jia, Chenjie Gu, Qiao Zhang, Jordan Grimstad, Ale Jakse Hartman, Martin Chadwick, Gaurav Singh Tomar, Xavier Garcia, Evan Senter, Emanuel Taropa, Thanumalayan Sankaranarayana Pillai, Jacob Devlin, Michael Laskin, Diego de Las Casas, Dasha Valter, Connie Tao, Lorenzo Blanco, Adrià Puigdomènech Badia, David Reitter, Mianna Chen, Jenny Brennan, Clara Rivera, Sergey Brin, Shariq Iqbal, Gabriela Surita, Jane Labanowski, Abhi Rao, Stephanie Winkler, Emilio Parisotto, Yiming Gu, Kate Olszewska, Yujing Zhang, Ravi Addanki, Antoine Miech, Annie Louis, Laurent El Shafey, Denis Teplyashin, Geoff Brown, Elliot Catt, Nithya Attaluri, Jan Balaguer, Jackie Xiang, Pidong Wang, Zoe Ashwood, Anton Briukhov, Albert Webson, Sanjay Ganapathy, Smit Sanghavi, Ajay Kannan, Ming-Wei Chang, Axel Stjerngren, Josip Djolonga, Yuting Sun, Ankur Bapna, Matthew Aitchison, Pedram Pejman, Henryk Michalewski, Tianhe Yu, Cindy Wang, Juliette Love, Junwhan Ahn, Dawn Bloxwich, Kehang Han, Peter Humphreys, Thibault Sellam, James Bradbury, Varun Godbole, Sina Samangooei, Bogdan Damoc, Alex Kaskasoli, Sébastien M. R. Arnold, Vijay Vasudevan, Shubham Agrawal, Jason Riesa, Dmitry Lepikhin, Richard Tanburn, Srivatsan Srinivasan, Hyeontaek Lim, Sarah Hodkinson, Pranav Shyam, Johan Ferret, Steven Hand, Ankush Garg, Tom Le Paine, Jian Li, Yujia Li, Minh Giang, Alexander Neitz, Zaheer Abbas, Sarah York, Machel Reid, Elizabeth Cole, Aakanksha Chowdhery, Dipanjan Das, Dominika Rogozińska, Vitaly Nikolaev, Pablo Sprechmann, Zachary Nado, Lukas Zilka, Flavien Prost, Luheng He, Marianne Monteiro, Gaurav Mishra, Chris Welty, Josh Newlan, Dawei Jia, Miltiadis Allamanis, Clara Huiyi Hu, Raoul de Liedekerke, Justin Gilmer, Carl Saroufim, Shruti Rijhwani, Shaobo Hou, Disha Shrivastava, Anirudh Baddepudi, Alex Goldin, Adnan Ozturel, Albin Cassirer, Yunhan Xu, Daniel Sohn, Devendra Sachan, Reinald Kim Amplayo, Craig Swanson, Dessie Petrova, Shashi Narayan, Arthur Guez, Siddhartha Brahma, Jessica Landon, Miteyan Patel, Ruizhe Zhao, Kevin Villela, Luyu Wang, Wenhao Jia, Matthew Rahtz, Mai Giménez, Legg Yeung, Hanzhao Lin, James Keeling, Petko Georgiev, Diana Mincu, Boxi Wu, Salem Haykal, Rachel Saputro, Kiran Vodrahalli, James Qin, Zeynep Cankara, Abhanshu Sharma, Nick Fernando, Will Hawkins, Behnam Neyshabur, Solomon Kim, Adrian Hutter, Priyanka Agrawal, Alex Castro-Ros, George van den Driessche, Tao Wang, Fan Yang, Shuo yiin Chang, Paul Komarek, Ross McIlroy, Mario Lučić, Guodong Zhang, Wael Farhan, Michael Sharman, Paul Natsev, Paul Michel, Yong Cheng, Yamini Bansal, Siyuan Qiao, Kris Cao, Siamak Shakeri, Christina Butterfield, Justin Chung, Paul Kishan Rubenstein, Shivani Agrawal, Arthur Mensch, Kedar Soparkar, Karel Lenc, Timothy Chung, Aedan Pope, Loren Maggiore, Jackie Kay, Priya Jhakra, Shibo Wang, Joshua Maynez, Mary Phuong, Taylor Tobin,

Andrea Tacchetti, Maja Trebacz, Kevin Robinson, Yash Katariya, Sebastian Riedel, Paige Bailey, Kefan Xiao, Nimesh Ghelani, Lora Aroyo, Ambrose Slone, Neil Houlsby, Xuehan Xiong, Zhen Yang, Elena Gribovskaya, Jonas Adler, Mateo Wirth, Lisa Lee, Music Li, Thais Kagohara, Jay Pavagadhi, Sophie Bridgers, Anna Bortsova, Sanjay Ghemawat, Zafarali Ahmed, Tianqi Liu, Richard Powell, Vijay Bolina, Mariko Iinuma, Polina Zablotskaia, James Besley, Da-Woon Chung, Timothy Dozat, Ramona Comanescu, Xiance Si, Jeremy Greer, Guolong Su, Martin Polacek, Raphaël Lopez Kaufman, Simon Tokumine, Hexiang Hu, Elena Buchatskaya, Yingjie Miao, Mohamed Elhawaty, Aditya Siddhant, Nenad Tomasev, Jinwei Xing, Christina Greer, Helen Miller, Shereen Ashraf, Aurko Roy, Zizhao Zhang, Ada Ma, Angelos Filos, Milos Besta, Rory Blevins, Ted Klimenko, Chih-Kuan Yeh, Soravit Changpinyo, Jiaqi Mu, Oscar Chang, Mantas Pajarskas, Carrie Muir, Vered Cohen, Charline Le Lan, Krishna Haridasan, Amit Marathe, Steven Hansen, Sholto Douglas, Rajkumar Samuel, Mingqiu Wang, Sophia Austin, Chang Lan, Jiepu Jiang, Justin Chiu, Jaime Alonso Lorenzo, Lars Lowe Sjösund, Sébastien Cevey, Zach Gleicher, Thi Avrahami, Anudhyan Boral, Hansa Srinivasan, Vittorio Selo, Rhys May, Konstantinos Aisopos, Léonard Hussenot, Livio Baldini Soares, Kate Baumli, Michael B. Chang, Adrià Recasens, Ben Caine, Alexander Pritzel, Filip Pavetic, Fabio Pardo, Anita Gergely, Justin Frye, Vinay Ramasesh, Dan Horgan, Kartikeya Badola, Nora Kassner, Subhrajit Roy, Ethan Dyer, Víctor Campos, Alex Tomala, Yunhao Tang, Dalia El Badawy, Elspeth White, Basil Mustafa, Oran Lang, Abhishek Jindal, Sharad Vikram, Zhitao Gong, Sergi Caelles, Ross Hemsley, Gregory Thornton, Fangxiaoyu Feng, Wojciech Stokowiec, Ce Zheng, Phoebe Thacker, Çağlar Ünlü, Zhishuai Zhang, Mohammad Saleh, James Svensson, Max Bileschi, Piyush Patil, Ankesh Anand, Roman Ring, Katerina Tsihlas, Arpi Vezer, Marco Selvi, Toby Shevlane, Mikel Rodriguez, Tom Kwiatkowski, Samira Daruki, Keran Rong, Allan Dafoe, Nicholas FitzGerald, Keren Gu-Lemberg, Mina Khan, Lisa Anne Hendricks, Marie Pellat, Vladimir Feinberg, James Cobon-Kerr, Tara Sainath, Maribeth Rauh, Sayed Hadi Hashemi, Richard Ives, Yana Hasson, YaGuang Li, Eric Noland, Yuan Cao, Nathan Byrd, Le Hou, Qingze Wang, Thibault Sottiaux, Michela Paganini, Jean-Baptiste Lespiau, Alexandre Moufarek, Samer Hassan, Kaushik Shivakumar, Joost van Amersfoort, Amol Mandhane, Pratik Joshi, Anirudh Goyal, Matthew Tung, Andrew Brock, Hannah Sheahan, Vedant Misra, Cheng Li, Nemanja Rakićević, Mostafa Dehghani, Fangyu Liu, Sid Mittal, Junhyuk Oh, Seb Noury, Eren Sezener, Fantine Huot, Matthew Lamm, Nicola De Cao, Charlie Chen, Gamaleldin Elsayed, Ed Chi, Mahdis Mahdieh, Ian Tenney, Nan Hua, Ivan Petrychenko, Patrick Kane, Dylan Scandinaro, Rishub Jain, Jonathan Uesato, Romina Datta, Adam Sadovsky, Oskar Bunyan, Dominik Rabiej, Shimu Wu, John Zhang, Gautam Vasudevan, Edouard Leurent, Mahmoud Alnahlawi, Ionut Georgescu, Nan Wei, Ivy Zheng, Betty Chan, Pam G Rabinovitch, Piotr Stanczyk, Ye Zhang, David Steiner, Subhajit Naskar, Michael Azzam, Matthew Johnson, Adam Paszke, Chung-Cheng Chiu, Jaume Sanchez Elias, Afroz Mohiuddin, Faizan Muhammad, Jin Miao, Andrew Lee, Nino Vieillard, Sahitya Potluri, Jane Park, Elnaz Davoodi, Jiageng Zhang, Jeff Stanway, Drew Garmon, Abhijit Karmarkar, Zhe Dong, Jong Lee, Aviral Kumar, Luowei Zhou, Jonathan Evens, William Isaac, Zhe Chen, Johnson Jia, Anselm Levskaya, Zhenkai Zhu, Chris Gorgolewski, Peter Grabowski, Yu Mao, Alberto Magni, Kaisheng Yao, Javier Snaider, Norman Casagrande, Paul Suganthan, Evan Palmer, Geoffrey Irving, Edward Loper, Manaal Faruqui, Isha Arkatkar, Nanxin Chen, Izhak Shafran, Michael Fink, Alfonso Castaño, Irene Giannoumis, Wooyeol Kim, Mikołaj Rybiński, Ashwin Sreevatsa, Jennifer Prendki, David Soergel, Adrian Goedeckemeyer, Willi Gierke, Mohsen Jafari, Meenu Gaba, Jeremy Wiesner, Diana Gage Wright, Yawen Wei, Harsha Vashisht, Yana Kulizhskaya, Jay Hoover, Maigo Le, Lu Li, Chimezie Iwuanyanwu, Lu Liu, Kevin Ramirez, Andrey Khorlin, Albert Cui, Tian LIN, Marin Georgiev, Marcus Wu, Ricardo Aguilar, Keith Pallo, Abhishek Chakladar, Alena Repina, Xihui Wu, Tom van der Weide, Priya Ponnapalli, Caroline Kaplan, Jiri Simsa, Shuangfeng Li, Olivier Dousse, Fan Yang, Jeff Piper, Nathan Ie, Minnie Lui, Rama Pasumarthi, Nathan Lintz, Anitha Vijayakumar, Lam Nguyen Thiet, Daniel Andor, Pedro Valenzuela, Cosmin Paduraru, Daiyi Peng, Katherine Lee, Shuyuan Zhang, Somer Greene, Duc Dung Nguyen, Paula Kurylowicz, Sarmishta Velury, Sebastian Krause, Cassidy Hardin, Lucas Dixon, Lili Janzer, Kiam Choo, Ziqiang Feng, Biao Zhang, Achintya Singhal, Tejasi Latkar, Mingyang Zhang, Quoc Le, Elena Allica Abellan, Dayou Du, Dan McKinnon, Natasha Antropova, Tolga Bolukbasi, Orgad Keller, David Reid, Daniel Finchelstein, Maria Abi Raad, Remi Crocker, Peter Hawkins, Robert Dadashi, Colin Gaffney, Sid Lall, Ken Franko, Egor Filonov, Anna Bulanova, Rémi Leblond, Vikas Yadav, Shirley Chung, Harry Askham, Luis C. Cobo, Kelvin Xu, Felix Fischer, Jun Xu, Christina Sorokin, Chris Alberti, Chu-Cheng Lin, Colin Evans, Hao Zhou, Alek Dimitriev, Hannah Forbes, Dylan Banarse, Zora Tung, Jeremiah Liu, Mark Omernick, Colton Bishop, Chintu Kumar, Rachel Sterneck, Ryan Foley, Rohan Jain, Swaroop Mishra, Jiawei Xia, Taylor Bos, Geoffrey Cideron, Ehsan Amid, Francesco Piccinno, Xingyu Wang, Praseem Banzal, Petru Gurita, Hila Noga, Premal Shah, Daniel J. Mankowitz, Alex Polozov, Nate Kushman, Victoria Krakovna, Sasha Brown, MohammadHossein Bateni, Dennis Duan, Vlad Firoiu, Meghana Thotakuri, Tom Natan, Anhad Mohananey, Matthieu Geist, Sidharth Mudgal, Sertan Girgin, Hui Li, Jiayu Ye, Ofir Roval, Reiko Tojo, Michael Kwong, James Lee-Thorp, Christopher Yew, Quan Yuan, Sumit Bagri, Danila Sinopalnikov, Sabela Ramos, John Mellor, Abhishek Sharma, Aliaksei Severyn, Jonathan Lai, Kathy Wu, Heng-Tze Cheng, David Miller, Nicolas Sonnerat, Denis Vnukov, Rory Greig, Jennifer Beattie, Emily Caveness, Libin Bai, Julian Eisenschlos, Alex Korchemniy, Tomy Tsai, Mimi Jasarevic, Weize Kong, Phuong

Dao, Zeyu Zheng, Frederick Liu, Fan Yang, Rui Zhu, Mark Geller, Tian Huey Teh, Jason Sanmiya, Evgeny Gladchenko, Nejc Trdin, Andrei Sozanschi, Daniel Toyama, Evan Rosen, Sasan Tavakkol, Linting Xue, Chen Elkind, Oliver Woodman, John Carpenter, George Papamakarios, Rupert Kemp, Sushant Kafle, Tanya Grunina, Rishika Sinha, Alice Talbert, Abhimanyu Goyal, Diane Wu, Denese Owusu-Afriyie, Cosmo Du, Chloe Thornton, Jordi Pont-Tuset, Pradyumna Narayana, Jing Li, Sabaer Fatehi, John Wieting, Omar Ajmeri, Benigno Uria, Tao Zhu, Yeongil Ko, Laura Knight, Amélie Héliou, Ning Niu, Shane Gu, Chenxi Pang, Dustin Tran, Yeqing Li, Nir Levine, Ariel Stolovich, Norbert Kalb, Rebeca Santamaria-Fernandez, Sonam Goenka, Wenny Yustalim, Robin Strudel, Ali Elqursh, Balaji Lakshminarayanan, Charlie Deck, Shyam Upadhyay, Hyo Lee, Mike Dusenberry, Zonglin Li, Xuezhi Wang, Kyle Levin, Raphael Hoffmann, Dan Holtmann-Rice, Olivier Bachem, Summer Yue, Sho Arora, Eric Malmi, Daniil Mirylenka, Qijun Tan, Christy Koh, Soheil Hassas Yeganeh, Siim Põder, Steven Zheng, Francesco Pongetti, Mukarram Tariq, Yanhua Sun, Lucian Ionita, Mojtaba Seyedhosseini, Pouya Tafti, Ragha Kotikalapudi, Zhiyu Liu, Anmol Gulati, Jasmine Liu, Xinyu Ye, Bart Chrzaszcz, Lily Wang, Nikhil Sethi, Tianrun Li, Ben Brown, Shreya Singh, Wei Fan, Aaron Parisi, Joe Stanton, Chenkai Kuang, Vinod Koverkathu, Christopher A. Choquette-Choo, Yunjie Li, TJ Lu, Abe Ittycheriah, Prakash Shroff, Pei Sun, Mani Varadarajan, Sanaz Bahargam, Rob Willoughby, David Gaddy, Ishita Dasgupta, Guillaume Desjardins, Marco Cornero, Brona Robenek, Bhavishya Mittal, Ben Albrecht, Ashish Shenoy, Fedor Moiseev, Henrik Jacobsson, Alireza Ghaffarkhah, Morgane Rivière, Alanna Walton, Clément Crepy, Alicia Parrish, Yuan Liu, Zongwei Zhou, Clement Farabet, Carey Radebaugh, Praveen Srinivasan, Claudia van der Salm, Andreas Fidjeland, Salvatore Scellato, Eri Latorre-Chimoto, Hanna Klimczak-Plucińska, David Bridson, Dario de Cesare, Tom Hudson, Piermaria Mendolicchio, Lexi Walker, Alex Morris, Ivo Penchev, Matthew Mauger, Alexey Guseynov, Alison Reid, Seth Odoom, Lucia Loher, Victor Cotruta, Madhavi Yenugula, Dominik Grewe, Anastasia Petrushkina, Tom Duerig, Antonio Sanchez, Steve Yadlowsky, Amy Shen, Amir Globerson, Adam Kurzrok, Lynette Webb, Sahil Dua, Dong Li, Preethi Lahoti, Surya Bhupatiraju, Dan Hurt, Haroon Qureshi, Ananth Agarwal, Tomer Shani, Matan Eyal, Anuj Khare, Shreyas Rammohan Belle, Lei Wang, Chetan Tekur, Mihir Sanjay Kale, Jinliang Wei, Ruoxin Sang, Brennan Saeta, Tyler Liechty, Yi Sun, Yao Zhao, Stephan Lee, Pandu Nayak, Doug Fritz, Manish Reddy Vuyyuru, John Aslanides, Nidhi Vyas, Martin Wicke, Xiao Ma, Taylan Bilal, Evgenii Eltyshev, Daniel Balle, Nina Martin, Hardie Cate, James Manyika, Keyvan Amiri, Yelin Kim, Xi Xiong, Kai Kang, Florian Luisier, Nilesh Tripuraneni, David Madras, Mandy Guo, Austin Waters, Oliver Wang, Joshua Ainslie, Jason Baldridge, Han Zhang, Garima Pruthi, Jakob Bauer, Feng Yang, Riham Mansour, Jason Gelman, Yang Xu, George Polovets, Ji Liu, Honglong Cai, Warren Chen, XiangHai Sheng, Emily Xue, Sherjil Ozair, Adams Yu, Christof Angermueller, Xiaowei Li, Weiren Wang, Julia Wiesinger, Emmanouil Koukoumidis, Yuan Tian, Anand Iyer, Madhu Gurumurthy, Mark Goldenson, Parashar Shah, MK Blake, Hongkun Yu, Anthony Urbanowicz, Jennimaria Palomaki, Chrisantha Fernando, Kevin Brooks, Ken Durden, Harsh Mehta, Nikola Momchev, Elahe Rahimtoroghi, Maria Georgaki, Amit Raul, Sebastian Ruder, Morgan Redshaw, Jinhyuk Lee, Komal Jalan, Dinghua Li, Ginger Perng, Blake Hechtman, Parker Schuh, Milad Nasr, Mia Chen, Kieran Milan, Vladimir Mikulik, Trevor Strohman, Juliana Franco, Tim Green, Demis Hassabis, Koray Kavukcuoglu, Jeffrey Dean, and Oriol Vinyals. 2023a. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023b. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogvlm: Visual expert for pretrained language models. *Preprint*, arXiv:2311.03079.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, et al. 2023. Cogvlm: Visual expert for pretrained language models. *arXiv preprint arXiv:2311.03079*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Catherine Wong, Kevin M Ellis, Joshua Tenenbaum, and Jacob Andreas. 2021. Leveraging language to learn program abstractions and search heuristics. In *International conference on machine learning*, pages 11193–11204. PMLR.

Xiangyu Wu, Yang Yang, Shengdong Xu, Yifeng Wu, Qingguo Chen, and Jianfeng Lu. 2023. Solution for smart-101 challenge of iccv multi-modal algorithmic reasoning task 2023. *arXiv preprint arXiv:2310.06440*.

Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 9(1):1.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. 2019. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5317–5327.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. 2023. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *Preprint*, arXiv:2304.10592.

A Appendix

A.1 Reasons of Failure

We manually investigated GPT-4o's failures in solving Scratch tasks in a single run to find the major causes of failure. We find four major causes: 1) Shape identification error: where the model fails to completely capture existent shapes in the input image, for instance, if it confuses a semicircle with a circle or assigns non-existent shape attributes to the input image. 2) Counting error: where the model fails to count adequately, (e.g., three triangles counted as four), 3) Orientation error: where the model fails to correctly find the relationships between different components of a shape (e.g., semicircle on top of a square vs. at its bottom), and 4) Implementation error: where the model's generated code does not follow the pre-planned pseudocode.

We manually investigated GPT-4o's failure output in the scratch code generation task and the results are provided in Table 4, where the failures are not mutually exclusive as a model can perform a combination of errors in each task. Furthermore, while the first three errors are according to the vision component in these models, we see that 64% of the failures are according to these causes, and in 36% of failure cases, there are no apparent vision errors.

A.2 Prompting

A.2.1 Basic Prompt

```
In each task, the user provides an image of an
abstract geometric shape or pattern and an
instruction, you need to generate a code in
Python Turtle that follows the user's request.
```

Figure 3: basic prompt used in our experiments

A.2.2 v-CoT Prompt

A.2.3 A Complete Example

Figure 5 we provide an instance of a complete prompt we used for a tweak *code generation* task with CoT prompting.

A.2.4 Arbitrary Output

Figure 6 provides the CoT prompt we used for the model to provide a code in any arbitrary language or library that creates the desired shape.

A.3 Rabbit

A.3.1 Prompt used

The prompt we used for this experiment is provided in Figure 7.

A.3.2 Definition of the class

The rabbit class is an arbitrary class that we defined based on *Turtle* class in the Python Turtle Module. This minimal set of functions includes all functions that a programmer or a model needs to create all of the tasks in TurtleBench. We defined this new set of functions to measure how GPT-4o is able to generalize its abilities in generating code in Python Turtle to a similar but minimally different set of functions.

```python
import turtle

class Rabbit(turtle.Turtle):
    def __init__(self):
        super().__init__()
        self.setheading(90)
        self.pensize(5)
        self.hideturtle()

    def aa(self, length):
        self.forward(length)

    def bb(self, degree):
        self.right(degree)

    def cc(self, radius, degree):
        self.circle(radius, degree)

    def pp(self, vanish):
        if vanish:
            self.penup()
        else:
            self.pendown()
```

A.4 Types of Tweak Tasks

TurtleBench includes a total of 130 tweak tasks. We provide a categorization for the tweaks as follows: There are five major types of tweaks in TurtleBench;

- Deletion: Removing a specified part of a shape

- Insertion: Adding a specific shape to the pattern as directed

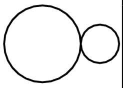
- Rotation: Rotating the entire shape

Cause	Description	Percentage
Shape identification error	The model fails to completely capture existent shapes in the input image, confusing or misattributing shapes	25%
Counting error	The model inadequately counts the elements.	35%
Orientation error	The model fails to correctly determine the spatial relationships between different components of a shape	21%
Implementation error	The model's generated code does not adhere to the pre-planned pseudocode, resulting in incorrect implementation.	45%

Table 4: Major Causes of GPT-4o's Failures in Scratch Tasks; note that the failures are not mutually exclusive, as a model can perform a combination of errors in each task

```
You are Turtle Geometrician, you are an expert in reasoning about images and generating code in
Python Turtle using images You need to follow the steps below before generating the answer:
(1) Describe the relevant information from the image needed to answer the question. List all relevant
artifacts from the image.
(2) Use the information described in (1) to reason about the problem by working step by step to
arrive at the final piece of code.
(3) Generate the final code. NEVER use "pensize" function in your code.
```

Figure 4: v-CoT prompt used in our experiments



```
System: You are Turtle Geometrician, you are an expert in reasoning about images and
generating code in Python Turtle using images. You need to follow the steps below before
generating the answer:
(1) Describe the relevant information from the image needed to answer the question. List
all relevant artifacts from the image.
(2) Use the information described in (1) to reason about the problem by working step by
step to arrive at the final piece of code.
(3) Generate the final code. NEVER use "pensize" function in your code.

Text: Provide a code in Python turtle that in the given shape inserts a circle of an equal
size to the smaller circle on the left of the bigger circle to make a vertically
symmetrical shape.

Complete the code:

import turtle
from math import *
t = turtle.Turtle()
large_circle_radius=100
small_circle_radius=50
...
```

Figure 5: An example of a complete prompt for a tweak code generation task with using v-CoT prompting.

- Reflection: Reflecting the entire shape or parts of it across specified lines
- Generalization: maintaining a pattern in the image constant while varying its parameters.

An illustration of instances of each type is provided in Figure 8. These types are not mutually exclusive as 10% of the tasks involve a combination of two types (e.g., removing one side of a square and inserting a semicircle instead). To successfully complete deletion and insertion tweaks, a model needs to demonstrate a nuanced understanding of the details in the image and program the resulting shape accordingly. In contrast, rotation tasks can be relatively easy as most of them can be solved only using a simple function in Turtle that can rotate the starting heading of the turtle which results in complete rotation in the entire shape (i.e., `turtle.right(angle)`).

```
You are an expert in reasoning about images and generating code in any language you prefer. You need
to follow the steps below before generating the code that answers the user's request:
(1) Describe the relevant information from the image needed to answer the question. List all relevant
information from the image.
(2) Use the information described in (1) to reason about the problem by working step by step to
arrive at the final piece of code.
(3) Generate the final code. Your code can be in any visual language or library, such as Matplotlib,
TikZ, etc.
```

Figure 6: The system prompt we used for the results discussed in Section 4.3

```
Suppose that I have a library named Rabbit in Python. Rabbit library has an object constructor named
Rabbit which is an object that moves on the screen and draws lines. It only has these functions:
aa(length): goes front or back (if the length is negative) and draws a line with the length of pixels.
bb(degree): The rabbit turns its head right or left (if degree is negative).
cc(radius, degree): creates an arc with the given radius for the given degree. If degree=360 it
creates a circle. The center of the circle is in the left of the rabbit.
pp(vanish): if vanish=True vanishes Rabbit object so if it moves does not draw anything, and if
vanish=False, it appears the Rabbit object so if it moves draws on the screen.
you call the functions on an object of Rabbit, such as r.aa(length) where r is an object of Rabbit.
When r is created, it faces north (up) on the screen and it does not vanish, so it is in drawing mode.

You are Rabbit Geometrician, you are an expert in reasoning about images and generating code in
Python Rabbit using images. You need to follow the steps below before generating the answer:
(1) Describe the relevant information from the image needed to answer the question. List all relevant
artifacts from the image.
(2) Use the information described in (1) to reason about the problem by working step by step to
arrive at the final piece of code.
(3) Generate the final code. Only use commands in the Rabbit class.
```

Figure 7: v-CoT prompt used for generalization experiments discussed in Section 4.2

A.5 Evaluating Image Complexity Using Contour Counts

As our result suggests that the vision component is contributing poorly to the models' performance, to gain a better understanding of the visual obstacles for the models to solve the tasks, we defined a measure as a proxy for the complexity of shapes. For each provided image, we calculated the number of contours in each shape. In OpenCV, a contour is a curve joining all the continuous points (along the boundary), having the same color or intensity. Contours are a useful tool for shape analysis and object detection and recognition. The high number of contours in an image hints that there are many shapes being involved and interleaving with each other, which makes understanding and extracting underlying patterns challenging.

We calculated the number of contours in each shape by utilizing the corresponding function in OpenCV, and defined three arbitrary levels of complexity in the images, where the images which include only one contour (e.g., the basic square in Figure 1) are at level 1 (simple), images including less than 6 contours and more than 1 are at level 2 (medium) (e.g, the base shape of insertion example in Figure 8) and the images in which there are more than 6 contours (e.g., the base shape in generalization example in Figure 8) are at level 3 of the complexity (Complex). In Turtle, the proportions of complexity levels 1, 2, and 3 are 25%, 40%, and 35%, respectively.

We investigate how models perform over tweak tasks. There are 9 different ways that a pair of input and output image can combine. As shown in Table 5, the majority of tweak tasks (74) have same levels of complexity for the input and output image.

To examine how complexity of input and output shapes impact the results, we categorize tweak tasks in the 9 different categories and count the number of tasks that are ever solved by GPT-4o under any prompting method in code generation and code edit tasks during 6 different runs. As shown in Table 5, the more complex the input shape is, the more challenging solving the task is.

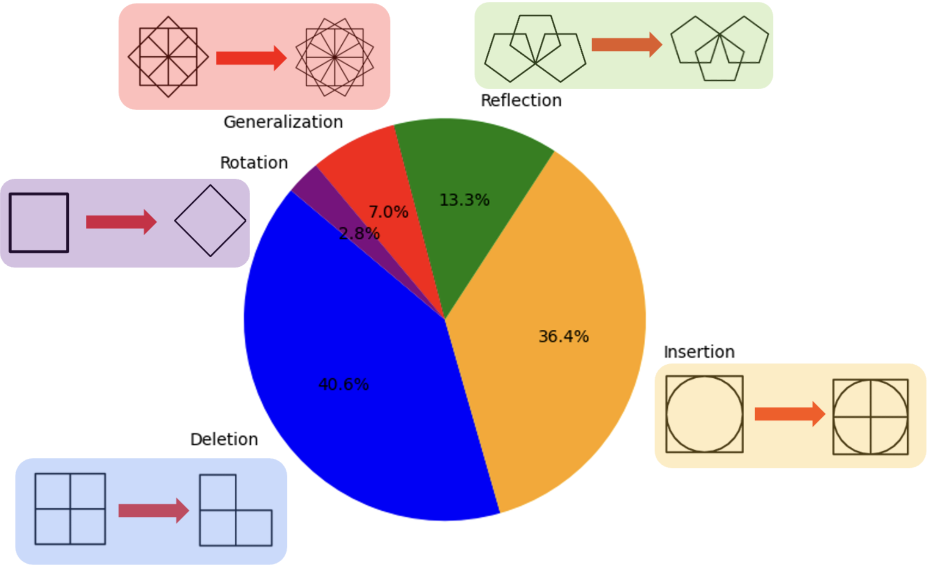
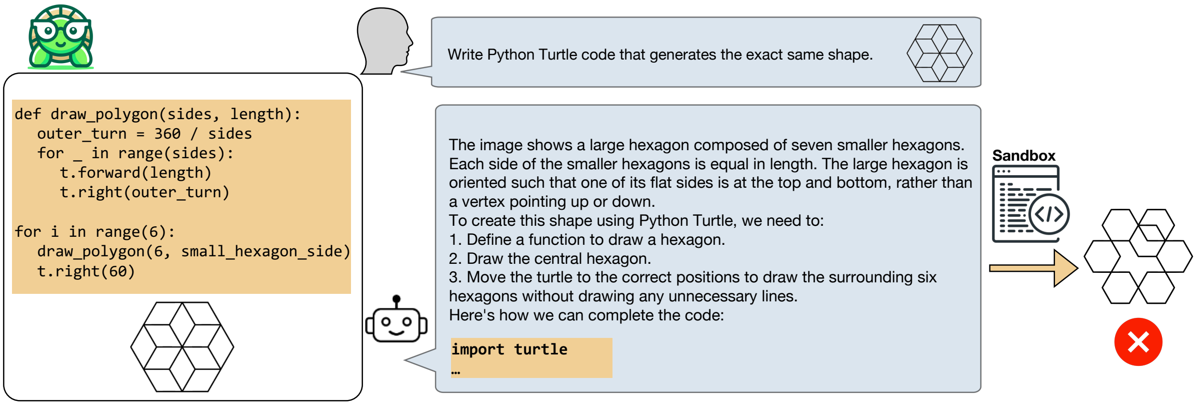


Figure 8: Types of tweaks and their share in TurtleBench



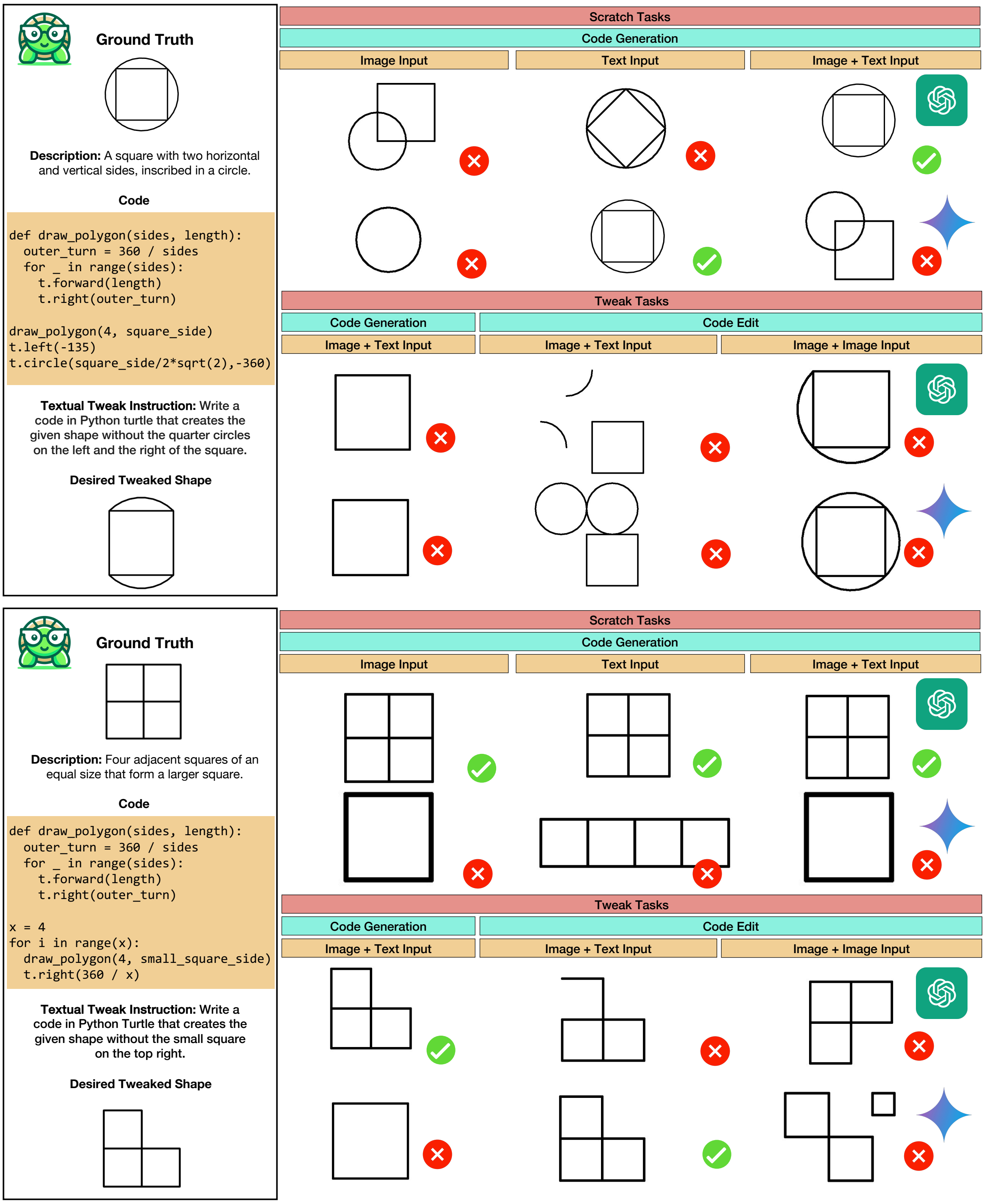
```
def draw_polygon(sides, length):
   outer_turn = 360 / sides
   for _ in range(sides):
      t.forward(length)
      t.right(outer_turn)

for i in range(6):
   draw_polygon(6, small_hexagon_side)
   t.right(60)
```

Write Python Turtle code that generates the exact same shape.

The image shows a large hexagon composed of seven smaller hexagons. Each side of the smaller hexagons is equal in length. The large hexagon is oriented such that one of its flat sides is at the top and bottom, rather than a vertex pointing up or down.
To create this shape using Python Turtle, we need to:
1. Define a function to draw a hexagon.
2. Draw the central hexagon.
3. Move the turtle to the correct positions to draw the surrounding six hexagons without drawing any unnecessary lines.
Here's how we can complete the code:

```
import turtle
…
```

Figure 9: An illustration of our evaluation pipeline



Description: A square with two horizontal and vertical sides, inscribed in a circle.

Code

```
def draw_polygon(sides, length):
  outer_turn = 360 / sides
  for _ in range(sides):
    t.forward(length)
    t.right(outer_turn)

draw_polygon(4, square_side)
t.left(-135)
t.circle(square_side/2*sqrt(2),-360)
```

Textual Tweak Instruction: Write a code in Python turtle that creates the given shape without the quarter circles on the left and the right of the square.

Desired Tweaked Shape

Description: Four adjacent squares of an equal size that form a larger square.

Code

```
def draw_polygon(sides, length):
  outer_turn = 360 / sides
  for _ in range(sides):
    t.forward(length)
    t.right(outer_turn)

x = 4
for i in range(x):
  draw_polygon(4, small_square_side)
  t.right(360 / x)
```

Textual Tweak Instruction: Write a code in Python Turtle that creates the given shape without the small square on the top right.

Desired Tweaked Shape

Figure 10: Two examples of tasks in TurtleBench across different modalities

		Output Complexity		
		Simple	Medium	Complex
Input Complexity	Simple	35% (7/20)	30% (3/10)	25% (1/4)
	Medium	40% (2/5)	18% (6/33)	7% (1/13)
	Complex	20% (1/5)	11% (2/19)	19% (4/21)

Table 5: The number of tweak tasks under each category and the percentage of those tasks ever solved by GPT-4o in different settings.