# © PIXELS LIE, CODE DOESN'T: THINKING WITH VISUAL PROGRAMMING FOR "SEEMINGLY IMPOSSIBLE" MULTIMODAL AGENTIC REASONING TASKS

# **Anonymous authors**

000

001

002

004

006

008 009 010

011 012

013

014

016

018

019

021

024

025

026

027

028

029

031

034

040

041

042

043

044

046

047

048

051

052

Paper under double-blind review

# **ABSTRACT**

To overcome the inherent limitations of Chain-of-Thought (CoT) and to further push the upper bound of multimodal reasoning capabilities, we introduce Thinking with Visual Programming (TVP), where models can iteratively interact with an external code executor to generate, run, and verify both visual and textual agentic operations as part of the reasoning loop. Motivated by the open question of how far Multimodal Large Language Models (MLLMs) still lag behind this paradigm, we introduce MMR-VIP, a MultiModal Agentic Reasoning benchmark built on Visual Impossible Problems. We design MMR-VIP with two key principles: (1) We construct a **Difficulty Ladder** grounded in computational complexity theory, structuring tasks from easy problems that can be solved with inherent perception and reasoning, through medium problems that require external computational tools, to hard problems that remain intractable even with programming assistance. (2) We decompose the paradigm of Thinking with Visual Programming into three Cognitive Skills, namely Perception, Abstraction, and Optimization, which correspond to perceiving visual inputs, abstracting them into problem formulations, and optimizing algorithms to obtain efficient solutions. Our experiments on MMR-VIP yield the following findings: (1) GPT-5, as a native TVP model, delivers the strongest overall results, yet its accuracy remains only 38.2%, underscoring substantial room for progress. (2) For commercial models, multi-turn code execution consistently surpasses direct CoT and single-turn execution, providing stable and significant improvements. (3) Across difficulty levels, performance follows a ladder-shaped trend, with negligible gains on easy tasks, the largest improvements on medium tasks, and steady advances on hard tasks. (4) From a cognitive perspective, TVP enhances optimization by offloading complex computation, search, and planning, but models still encounter bottlenecks in abstraction.

## 1 Introduction

Multimodal reasoning is a defining capability of human intelligence, enabling us to address diverse challenges such as navigating in the physical world, interpreting scientific figures, and solving geometry problems (Yue et al., 2024; Lu et al., 2024). Recent advances in Multimodal Large Language Models (MLLMs) (OpenAI, 2024; DeepMind, 2025; Bai et al., 2025) have demonstrated significant progress by leveraging Chain-of-Thought (CoT) (Wei et al., 2022; Zhang et al., 2024b), which bridges perception and reasoning through explicit sequences of textual steps. Nevertheless, existing improvements remain constrained, since they primarily extend text-based CoT, prolonging the reasoning process without enhancing the upper bound of the model's reasoning capabilities.

To address these limitations, "Thinking with Images" (TWI) (Zheng et al., 2025; Su et al., 2025a;c) has recently been proposed as an emerging paradigm, enabling models to incorporate iterative visual inspection as part of the reasoning loop. In practice, current methods rely on manipulating images with external tools (Zhang et al., 2025), such as cropping, zooming, rotating, and other image processing operations. Although such strategies enhance perceptual accuracy, they are limited to a narrow set of predefined visual tools, thereby rigidifying the reasoning process and restricting its generality. As various forms of visual manipulation can naturally be expressed and executed through programming, we propose the paradigm of "Thinking with Visual Programming" (TVP), which goes beyond fixed tools and allows models like OpenAI's o4 and GPT-5 (OpenAI, 2025a) to flexibly generate, execute, and verify both visual and textual operations within the reasoning process.

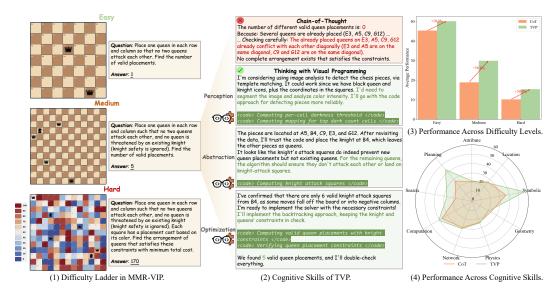


Figure 1: Thinking with Visual Programming paradigm. Figure 1(1) illustrates the three difficulty levels in MMR-VIP using the N-Queens task as an example. Figure 1(2) shows that for a medium-level problem, direct CoT reasoning fails while TVP succeeds, and in the process three key cognitive skills emerge. Figure 1(3) compares the average performance of four powerful models (GPT-4.1-mini, GPT-4.1, Gemini-2.5-Flash, and Claude-Sonnet-4) under CoT and TVP, showing minimal changes on easy tasks, the largest gains on medium tasks, and clear improvements on hard tasks. The results exhibit a ladder-shaped performance trend across difficulty levels. Figure 1(4) presents the performance differences of the four models across cognitive skills, where TVP yields notable improvements in symbolic (perception), computation, search, and planning (optimization).

Humans inherently solve complex reasoning problems in a programming-like manner by preprocessing visual inputs for better perception, applying algorithmic procedures to derive solutions, and verifying outcomes through testing. Nevertheless, it remains unclear how far current MLLMs are from this paradigm. To this end, we introduce MMR-VIP, a MultiModal Agentic Reasoning benchmark that consists of Visual Impossible Problems. Formally, we refer to Visual Impossible Problems as problems that appear intractable under CoT-based reasoning, yet become solvable when augmented with visual programming interactions. We design MMR-VIP with two key considerations:

**Difficulty Ladder.** We categorize problems into three levels of difficulty, drawing inspiration from how humans tackle tasks with and without tools, and grounded in computational complexity theory. (1) **Easy** level requires that the model can reliably solve them using its inherent perception and reasoning abilities, without any programming assistance. This level corresponds to "low-complexity problems in P", where the model can perform reasoning within its working memory; (2) **Medium** level is challenging for the model to solve independently, but can be effectively addressed when it is allowed to use a code interpreter. This level typically involves "polynomial-time solvable problems in P", where the model must rely on external computational tools to compute solutions; (3) **Hard** level remains unsolved even with programming assistance, often due to their large-scale computational complexity, highly intricate constraints, or demanding optimization requirements. This level corresponds conceptually to "NP-hard problems", which often lie beyond the capabilities of current models. As shown in Figure 1(1), the three levels form a progressive difficulty ladder, where each step reflects an increasing demand on the model's reasoning capacity and reliance on external tools.

Cognitive Skill. We decompose the Thinking with Visual Programming paradigm into three key cognitive skills, focusing on the core cognitive processes required to perceive, abstract, and optimize multimodal agentic reasoning. Taking the N-Queens problem in Figure 1(2) as an example: (1) **Perception** requires the model to transform *visual content* into *structured information*, correctly extracting relevant elements from multimodal inputs (*e.g.*, detecting and locating chess pieces on the board); (2) **Abstraction** requires the model to transform *structured information* to *problem formulation*, producing computationally useful forms and proposing feasible solutions (*e.g.*, converting piece positions into symbolic constraints that capture attack rules); (3) **Optimization** requires the model to transform *problem formulation* to *algorithmic optimization*, optimizing both algorithms

and computational procedures to obtain correct and efficient answers (e.g., applying a backtracking algorithm to search for valid queen placements under the given constraints).

MMR-VIP encompasses 28 carefully crafted task types, each designed across three difficulty levels, resulting in 1,680 instances that provide a comprehensive evaluation of multimodal agentic reasoning capabilities. These tasks span a wide spectrum, from basic skills such as *counting* and *height measurement* to advanced challenges including *graph coloring* and *circuit logic*. To avoid dataset contamination and guarantee that models solve tasks via code execution instead of memorized recall, all problems in MMR-VIP are generated using carefully designed, manually written code.

We conduct a comprehensive evaluation on MMR-VIP across a wide range of MLLMs, including commercial models such as Claude-Sonnet-4, open-source models such as Qwen2.5-VL-72B, as well as native TVP models like o4-mini and GPT-5. We further assess different reasoning paradigms, including direct CoT, single-turn code execution, and multi-turn code execution. We obtain the following conclusions: (1) Our experimental results reveal clear differences across model types and reasoning paradigms. For open-source models, introducing code execution provides little to no improvement, mainly due to their limited visual programming capabilities. For commercial models, single-turn code execution yields unstable performance, while multi-turn code execution consistently delivers substantial gains. As illustrated in Figure 1(3), multi-turn code execution improves accuracy on medium-level tasks by **58.4**% compared to direct CoT. GPT-5, as a native TVP model, achieves the best overall performance; however, its accuracy remains only 38.2%, indicating substantial room for improvement; (2) Performances across different difficulty levels align well with the design of MMR-VIP, exhibiting a ladder-shaped performance trend. Compared to direct CoT, we observe that TVP yields minimal changes on easy tasks, the largest gains on medium tasks, and consistent improvements on hard tasks; (3) From the perspective of cognitive skills, TVP shows clear progress in optimization, as it can leverage programming to offload complex computation, search, and planning operations. However, its performance still encounters bottlenecks in abstraction, where models lack the ability to translate visual inputs into high-level problem formulations. We hope that MMR-VIP will serve as a challenging benchmark to drive future research toward closing this gap.

# 2 PARADIGM DEFINITIONS

### 2.1 Multimodal Chain-of-Thought

We formalize the conventional paradigm of Multimodal Chain-of-Thought reasoning. For a model  $\theta$ , given an input image I and a textual question x, the CoT process can be defined as:

$$P_{\text{CoT}}(y \mid I, x) = P_{\theta}(r \mid I, x) \cdot P_{\theta}(y \mid I, x, r). \tag{1}$$

Here,  $r = (s_1, s_2, \dots, s_n)$  denotes the intermediate reasoning chain, which explicitly captures the sequence of textual steps bridging perception and reasoning, while y represents the final answer conditioned on both the original input (I, x) and the generated textual rationale r.

## 2.2 THINKING WITH VISUAL PROGRAMMING

We formalize the proposed paradigm of Thinking with Visual Programming. For a model  $\theta$ , given an input image I and textual question x, TVP extends conventional CoT by introducing programming actions  $a_t$ , which are executed through interaction with an external code executor  $\mathcal{E}$ . Unlike single-pass reasoning, this is a multi-turn interactive agentic process consisting of T rounds:

$$P_{\text{TVP}}(y \mid I, x) = \prod_{t=1}^{T} P_{\theta}(r_t, a_t \mid s_{t-1}) \cdot P_{\theta}(y \mid s_T).$$
 (2)

At each step t, the model generates a reasoning trace  $r_t$  and a programming action  $a_t$ , executes  $a_t$  via the external executor  $\mathcal{E}$ , and incorporates the multimodal execution result  $\mathcal{E}(a_t)$  into the state  $s_t$ :

$$s_t = s_{t-1} \cup \{r_t, a_t, \mathcal{E}(a_t)\}, \quad s_0 = \{I, x\}.$$
 (3)

Compared to CoT, TVP offers significant advantages by integrating pixel manipulations and algorithmic computation into the reasoning loop, enabling models to move beyond textual thinking. In this paper, we do not provide models with fixed external tools. Instead, we allow them to write code that can call standard libraries, such as PIL, OpenCV, and Matplotlib, among others.

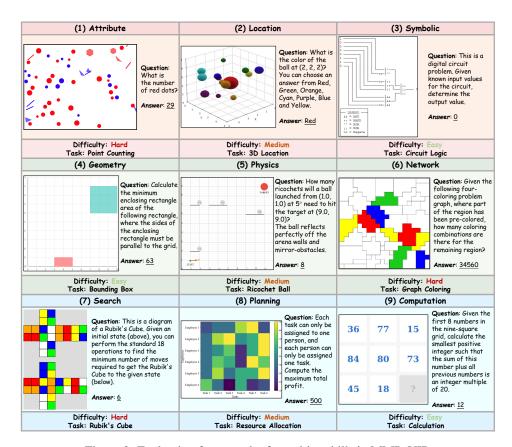


Figure 2: Evaluation framework of cognitive skills in MMR-VIP.

# 3 MMR-VIP BENCHMARK

To investigate how far current MLLMs are from the paradigm of TVP, we introduce MMR-VIP, a MultiModal Agentic Reasoning benchmark that consists of Visual Impossible Problems. These are carefully designed problems that existing MLLMs cannot reliably solve with conventional CoT reasoning alone, but instead necessitate interaction with an external code executor. We will detail the design principles behind MMR-VIP, including its difficulty ladder and cognitive skill dimensions, and describe the benchmark construction process along with dataset statistics.

#### 3.1 DIFFICULTY LADDER

We categorize problems in MMR-VIP into a three-level **Difficulty Ladder**, drawing inspiration from how humans tackle tasks of varying complexity and their reliance on external tools. At the **Easy** level, tasks can be reliably solved using the model's inherent perception and reasoning abilities, without the need for programming assistance. These correspond to "low-complexity problems in P", where solutions can be derived directly within the model's working memory. The **Medium** level encompasses tasks that models struggle to solve on their own but can successfully address when supported by external tools such as code interpreters. These tasks align with "polynomial-time solvable problems in P," where deriving solutions requires programmatic operations and computational tools beyond intuition alone. Finally, the **Hard** level captures problems that remain unsolved even with programming assistance, typically due to large-scale computational complexity, intricate constraints, or challenging optimization requirements. Conceptually, these tasks are analogous to "NP-hard problems", which often exceed the practical capabilities of current models. Such a difficulty ladder setting enables a more in-depth examination of the paradigm of TVP.

<sup>&</sup>lt;sup>1</sup>The tasks in MMR-VIP are not strictly designed or guaranteed to align with formal complexity-theoretic definitions, but rather follow the spirit of increasing computational and cognitive demands.

Table 1: Mapping between cognitive skills and task types in MMR-VIP.

Category	Tasks						
Attribute	3D Position, Bin Packing, Graph Coloring, Hanoi Tower, Point Counting, Resource Allocation,						
	Rubik's Cube, Sliding Puzzle, Snake Game, Three-Views						
Location	3D Position, Bounding Box, Height Measurement, Point Counting, Projectile Motion, Snake						
	Game, Three-Views						
Symbolic	Calculation, Chart, Circuit Logic, House Robber, Interval DP, N-Puzzle, Projectile Motion,						
•	Tableau LP						
Geometry	Area Measurement, Bounding Box, Rubik's Cube, Three-Views						
Physics	Circuit Logic, Projectile Motion, Ricochet Ball						
Network	Graph Coloring, Graph Isomorphism						
Search	Bin Packing, Bubble Sort, Calculation, Graph Coloring, Maze, N-Puzzle, N-Queens, Path						
	Counting, Rubik's Cube, Sliding Puzzle, Snake Game						
Planning	Chart, Hanoi Tower, House Robber, Interval DP, Lights Out, Resource Allocation, Tableau LP						
Computation	Calculation, Path Counting						

## 3.2 Cognitive Skill

Beyond task difficulty, we design MMR-VIP to emphasize the underlying **Cognitive Skills** required for multimodal agentic reasoning under the TVP paradigm. These skills highlight the essential processes through which models must learn to leverage external tools to approach complex problems. We define three successive skills within TVP: **Perception**, **Abstraction**, and **Optimization**, which together examine a model's visual programming ability from complementary dimensions.

**Perception**: This skill concerns the model's ability to accurately extract structured information from raw visual inputs. Unlike direct pattern recognition that relies solely on intrinsic visual perception, TVP enables models to enhance perception through programmatic operations such as counting, measuring, and localization. For example, as shown in Figure 2(1), when a task requires precise object counting, models that rely only on intrinsic perception often fail due to overlapping shapes, varying sizes, or background noise. In contrast, TVP enables the model to generate code that analyzes pixel-level cues such as color and boundary lines, allowing it to count objects more accurately. We evaluate this skill across three dimensions: **Attribute** (*i.e.*, color, shape, size), **Location** (*i.e.*, positions, distances, spatial relations), and **Symbolic** (*i.e.*, digits, letters, or graphical symbols).

**Abstraction**: This skill concerns the model's ability to transform low-level structured information into higher-level problem formulations. It requires not only recognizing surface patterns but also capturing the underlying rules and constraints, and converting them into computationally useful forms. For instance, as illustrated in Figure 2(6), the model must write code to abstract the puzzle into a network structure, representing each piece as a node and encoding adjacency relations as edges. This code-based abstraction allows the model to perform further search or optimization over the graph. In MMR-VIP, we evaluate abstraction across three dimensions: **Geometry** (*i.e.*, geometric formulations), **Physics** (*i.e.*, physical laws), and **Network** (*i.e.*, graph structures).

**Optimization:** This skill focuses on the model's ability to transform problem formulations into efficient algorithmic solutions. It requires not only identifying feasible solutions but also refining them to satisfy the given conditions. For example, as illustrated in Figure 2(7), the Rubik's Cube task requires the model to minimize the number of moves from an initial state to a target state. TVP enables the model to generate and execute code that systematically explores the space of valid cube operations, pruning redundant paths and converging to the optimal sequence of moves. We evaluate this skill across three dimensions: **Search** (*i.e.*, depth-first search, breadth-first search), **Planning** (*i.e.*, dynamic programming, linear programming), and **Computation** (*i.e.*, numerical calculations).

# 3.3 BENCHMARK CONSTRUCTION

To ensure that tasks are both solvable in the TVP paradigm and suitable for difficulty control, we adopt a Code2Task generation framework. We recruited five annotators with strong backgrounds in programming competitions and instructed them to write code that specifies task rules and automatically generates the corresponding images<sup>2</sup>, problems, and answers. As task difficulty increased, annotators were required to design new rules and introduce greater computational complexity, thereby enriching the reasoning challenges. To facilitate this process, annotators were permitted to utilize

<sup>&</sup>lt;sup>2</sup>We implemented visualization through HTML and Matplotlib.

Table 2: Experimental results on MMR-VIP. The best performance in each column is highlighted in **bold**. Red denotes cases where TVP underperforms CoT, while Green denotes cases where it outperforms CoT, with darker shades indicating larger magnitude of change.

Model		Difficulty Level			Cognitive Skill								Overall	
		Easy	Mid	Hard	Att	Loc	Sym	Geo	Phy	Net	Com	Sea	Pla	Overan
Open-source Models														
Keye-VL-1.5-8B	CoT	28.0	11.4	4.8	12.0	9.3	21.2	11.2	8.3	27.5	35.0	15.6	14.3	14.8
	T=1	9.1	3.2	2.5	4.2	3.8	5.2	3.8	4.4	12.5	6.7	4.2	3.8	4.9 (\pm 9.9)
Gemma-3-27B	CoT	16.2	5.5	5.0	7.8	6.0	11.9	2.5	10.0	16.7	17.5	10.2	6.0	8.9
	T=1	15.5	10.0	4.1	6.2	5.5	17.3	4.2	6.7	22.5	28.3	8.9	10.2	9.9 († 1.0)
Qwen2.5-VL-7B	CoT	13.2	7.0	3.9	7.5	3.6	7.3	3.8	8.9	26.7	6.7	6.8	6.4	8.0
	T=1	7.3	5.5	1.6	2.5	2.9	7.5	2.1	2.2	17.5	21.7	5.3	1.7	4.8 (↓ 3.2)
	CoT	24.3	10.9	6.4	13.7	12.6	17.3	8.3	10.0	25.0	29.2	14.7	8.6	13.9
Qwen2.5-VL-32B	T=1	13.6	4.6	4.1	7.7	8.1	6.9	4.2	9.4	13.3	6.7	5.9	4.8	7.4 (\( \psi \) 6.5)
	T=3	18.9	9.8	6.8	10.0	8.8	18.3	2.5	7.2	20.8	28.3	13.3	10.7	11.8 (\pm 2.1)
Owen2.5-VL-72B	CoT	23.9	10.4	6.1	12.3	10.7	15.8	12.5	8.9	30.8	20.8	12.1	8.6	13.4
QWCIIZ.5-VE-72B	T=1	20.5	9.6	4.5	11.5	11.7	16.2	5.8	7.8	18.3	15.8	9.1	10.9	11.6 (\( \pm 1.8 )
Commercial Models														
	CoT	42.7	20.2	9.8	23.3	23.8	32.1	19.6	21.1	30.8	34.2	18.9	26.9	24.2
GPT-4.1-mini	T=1	45.5	28.2	14.1	16.0	23.3	49.4	22.1	22.2	35.0	39.2	24.7	37.1	29.3 († 5.1)
	T=3	42.1	28.4	14.1	21.8	21.4	45.0	16.2	14.4	31.7	31.7	22.9	40.5	28.2 († 4.0)
	CoT	42.7	19.1	11.1	23.0	26.9	33.8	20.0	26.1	30.0	35.0	13.9	28.6	24.3
GPT-4.1	T=1	38.9	18.0	7.1	18.7	23.8	25.4	22.1	27.2	27.5	22.5	17.6	16.4	21.4 (\pm 2.9)
	T=3	47.1	25.5	12.1	18.7	25.0	50.4	13.8	32.8	28.3	32.5	20.3	36.9	28.3 († 4.0)
	CoT	46.4	18.0	10.9	17.8	27.6	42.5	26.2	25.6	32.5	40.0	17.7	28.1	25.1
Gemini-2.5-Flash	T=1	32.7	14.5	7.9	9.0	17.4	34.2	22.1	25.0	12.5	32.5	14.4	19.5	18.3 (\$\dagger\$ 6.8)
	T=3	59.3	34.5	16.1	21.8	30.2	64.6	27.1	33.3	29.2	40.0	30.0	49.0	36.6 († 11.5)
Gemini-2.5-Pro	CoT	58.0	20.9	10.4	21.3	25.7	44.4	29.6	29.4	27.5	37.5	26.2	32.9	29.8
Geiiiii-2.5-110	T=1	38.8	20.2	11.4	12.7	17.9	38.3	20.0	30.0	27.5	16.7	16.5	29.5	23.4 (\(\psi\) 6.4)
Claude-Sonnet-4	CoT	49.6	18.2	8.9	19.5	28.6	38.3	27.1	23.3	26.7	38.3	19.2	28.1	25.6
Claude-Sollice-4	T=1	49.5	31.6	14.3	22.0	27.6	53.5	19.6	26.1	28.3	45.8	28.6	38.3	31.8 († 6.2)
					Nat	tive TV	P Mod	els						
o4-mini		57.7	30.7	17.3	28.0	24.8	55.4	25.4	26.1	35.0	40.0	27.3	55.2	35.2
GPT-5-mini		61.8	29.8	16.8	32.7	29.8	52.9	27.9	26.1	32.5	42.5	28.3	53.3	36.1
GPT-5		65.5	33.8	15.4	27.5	31.7	60.4	30.8	31.1	39.2	39.2	28.2	56.9	38.2
Reference														
Human		69.6	55.4	35.7	48.3	54.8	75.0	37.5	27.8	58.3	66.7	50.0	69.1	53.6

AI-assisted code editors (*e.g.*, Cursor). Finally, we conducted cross-validation of all generated code to verify correctness, where each program was independently reviewed by multiple annotators.

In total, MMR-VIP encompasses 28 carefully crafted task types, each designed across three difficulty levels. For every task and difficulty, we randomly generated 20 instances, resulting in a benchmark of 1,680 instances in total. We include detailed examples of each task in the Appendix C. The mapping between task types and their corresponding cognitive skills is presented in Table 1. Since all tasks are synthesized from code, MMR-VIP is reproducible and extendable. Researchers can regenerate new instances by adjusting parameters or extend the benchmark with new task rules, making MMR-VIP a continuously evolvable framework rather than a fixed dataset.

# 4 EXPERIMENTS

In this section, we present a comprehensive evaluation of existing MLLMs on MMR-VIP. We systematically evaluate model performance across different difficulty levels and cognitive skills, and further contrast the effectiveness of CoT and TVP. We also analyze from multiple perspectives, including the effect of iteration rounds, the role of input modalities, and the distribution of error types.

## 4.1 EXPERIMENTAL SETUP

We evaluate three categories of MLLMs on MMR-VIP: commercial models (*e.g.*, GPT-4.1 (OpenAI, 2025b), Gemini-2.5-Flash, Gemini-2.5-Pro (DeepMind, 2025), Claude-Sonnet-4 (Anthropic, 2025)), open-source models (*e.g.*, Qwen2.5-VL (Bai et al., 2025), Gemma-3 (Kamath et al., 2025), Keye-VL-1.5 (Yang et al., 2025)), and native TVP models (*e.g.*, o4-mini, GPT-5). We do not include existing open-source models designed specifically for Thinking with Images, since these models primarily focus on applying fixed transformations to images rather than freely generating code to sup-

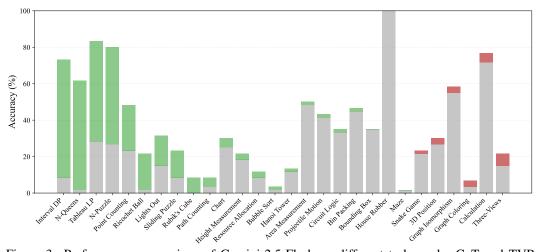


Figure 3: Performance comparison of Gemini-2.5-Flash on different tasks under CoT and TVP (T=3). Gray indicates the baseline performance of CoT, Green indicates improvements of TVP over CoT, and Red indicates degradations of TVP over CoT.

port reasoning. Moreover, to assess the effectiveness of different reasoning strategies, we compare three settings: Chain-of-Thought, single-turn TVP, where the model invokes the code executor once, and multi-turn TVP, where the model can iteratively generate, execute, and refine code for up to T=3,5,7 rounds. We provide the detailed prompts used for all settings in the Appendix D. As a reference, we randomly sample 168 instances and invite human participants to solve these tasks. Each participant is allowed to leverage search engines and interpreters during the process. We adopt accuracy as the evaluation metric. We report results along three perspectives: performance across different difficulty levels, performance across distinct cognitive skills, and the overall accuracy.

# 4.2 EXPERIMENTAL RESULTS

As shown in the Table 2, our experiments on MMR-VIP yield several key findings:

- (1) **Performance differences across model types and reasoning paradigms.** For open-source models like Qwen2.5-VL-72B, introducing TVP offers negligible gains and sometimes results in performance drops, owing to their limited visual programming capabilities. For commercial models, single-turn code execution produces unstable results, whereas multi-turn execution consistently yields significant improvements. For instance, Gemini-2.5-Flash shows an accuracy gain of 18.3% when increasing from T=1 to T=3. For native TVP models, although GPT-5 achieves the highest performance, it attains only 38.2% accuracy, reflecting the substantial limitations that remain. We can observe a clear performance gap relative to humans, underscoring that humans are more adept at leveraging external tools to solve complex visual problems.
- (2) **Clear difficulty ladder.** The results align closely with the benchmark's design, showing a distinct ladder-shaped performance trend. Compared to direct CoT, TVP shows negligible differences on easy tasks, achieves the largest improvements on medium tasks, and delivers consistent gains on hard tasks. Nevertheless, performance at the hard level remains very low, with the best accuracy reaching only 17.3%. This demonstrates that MMR-VIP effectively stratifies problems by difficulty, thereby exposing the limits of current MLLMs' reasoning capabilities.
- (3) **Imbalanced cognitive skills.** The results reveal marked disparities across cognitive skills. TVP delivers the most significant improvements in Optimization, where models effectively leverage programmatic search, planning, and computation to tackle complex problem-solving tasks. As shown in Figure 3, Gemini-2.5-Flash exhibits large gains on tasks such as *Interval Dynamic Programming*, *N-Queens*, *Tableau Linear Programming*, and *N-Puzzle*, where code execution is essential to explore solution spaces. In addition, TVP also enhances Symbolic perception, since code allows models to precisely recognize, parse, and manipulate digits, letters, or graphical symbols. However, performance in Abstraction remains the most challenging, where models still struggle to translate low-level visual cues into high-level formulations such as geometric equations, physical laws, or graph structures. This underscores the necessity of improving their ability to abstract through code.

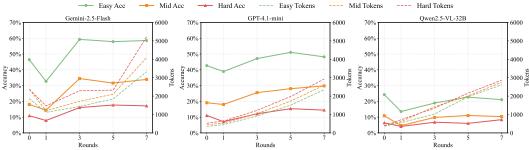


Figure 4: Performance and token consumption under different iteration rounds (T=0,1,3,5,7). T=0 corresponds to CoT. Green, orange, and red correspond to Easy, Medium, and Hard levels.

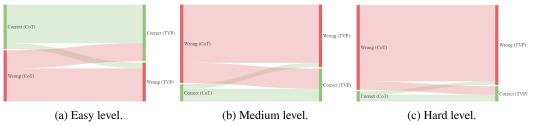


Figure 5: Correctness flow between CoT (T=0) and TVP (T=3) for Gemini-2.5-Flash.

#### 4.3 ANALYSIS

#### 4.3.1 IMPACT OF ITERATION ROUNDS

We examine the impact of iterative rounds of code execution on model performance across easy, medium, and hard tasks. As shown in Figure 4, compared to direct CoT, single-turn TVP (T=1) often leads to a drop in accuracy. To better understand this phenomenon, we compute the correlation between the performance difference of TVP (T=1) versus CoT and the success rate of program execution. The Pearson correlation coefficient is  ${\bf 0.81}~(p\approx 0.05)$ , indicating a strong positive relationship. A primary source of degradation arises when incorrect code execution propagates interpreter error messages into the reasoning process, thereby misguiding subsequent inference.

Performance generally peaks at T=3 or T=5, where iterative refinement enables more reliable program execution and reflective reasoning. As illustrated in Figure 5, we further analyze the correctness flow between CoT and TVP (T=3). The results show that the most significant changes occur at the Medium difficulty level. However, for open-source models like Qwen2.5-VL-32B, additional iterations fail to bring noticeable gains. This finding highlights that robust visual programming capabilities are indispensable for open-source models to fully realize the benefits of TVP. Meanwhile, increasing to T=7 brings little to no additional gains and instead results in significantly higher token consumption, highlighting the trade-off between accuracy and efficiency.

# 4.3.2 Influence of Input Modalities

To further investigate the role of input modalities in TVP, we select four tasks from MMR-VIP that can be represented in both textual and visual forms: *Tableau LP*, *Chart*, *Graph Coloring*, and *Maze*. This design allows us to directly compare model performance under three conditions: (1) image-only input (I), (2) text-only input (T), and (3) combined image-text input (I & T). Results in Table 3 show that text input generally outperforms image input, indicating that current models still have weaker visual

Table 3: Performance comparison under different input modalities.

Model	I	T	I & T
GPT-4.1-mini	26.3	75.0	76.3
Claude-Sonnet-4	7.5	50.0	63.8
GPT-5-mini	5.0	50.0	53.8
GPT-5	25.0	46.3	70.0

reasoning capabilities. Moreover, visual inputs sometimes introduce perception errors, which can propagate through subsequent reasoning steps. Nevertheless, combined multimodal input consistently surpasses unimodal input, particularly on tasks where the visual layout conveys structural or spatial constraints that are difficult to capture with text alone.

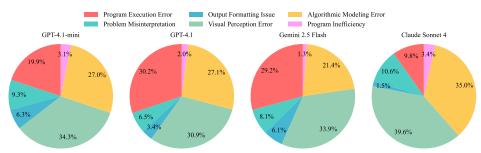


Figure 6: Error analysis of four models under TVP (T = 1).

#### 4.3.3 ERROR ANALYSIS

To better understand the limitations of TVP, we conduct a detailed error analysis by categorizing incorrect predictions into six major types: Program Execution Error, Visual Perception Error, Algorithmic Modeling Error, Program Inefficiency, Problem Misinterpretation, and Output Formatting Issue. The precise definitions and representative examples of each category are provided in the Appendix F. As illustrated in Figure 6, the most common sources of error are Visual Perception Error, Algorithmic Modeling Error, and Program Execution Error. These results align with our earlier findings: they reflect (1) the insufficiency of models in Perception and Abstraction, where they struggle to accurately extract information from visual inputs and transform it into computationally useful formulations, and (2) the limitations of current models' programming capabilities, where code errors remain prevalent. We also provide several case studies of CoT and TVP in Appendix G.

## 5 RELATED WORKS

Multimodal Reasoning. Multimodal reasoning has recently become a prominent frontier in AI research, with an expanding set of benchmarks and investigations underscoring its pivotal importance across domains such as interpreting scientific diagrams (Yue et al., 2024; Guo et al., 2025), solving geometry problems (Zhang et al., 2024a; Wang et al., 2024), and tackling visual puzzles (Chia et al., 2024; Ghosal et al., 2025; Song et al., 2025). Recent work (Huang et al., 2025; Meng et al., 2025; Chris et al., 2025; Hong et al., 2025; Deng et al., 2025; Wang et al., 2025c;b) has focused on enhancing models' reasoning ability through reinforcement learning, thereby extending reasoning depth, enabling reflection and verification, and improving performance on complex tasks. However, some studies argue that RL is constrained by an invisible leash (Wu et al., 2025a), preventing it from discovering new reasoning trajectories beyond the model's initial capabilities (Lin & Xu, 2025).

Tool-Integrated Reasoning. Rather than relying solely on parametric knowledge within the model, tool-integrated reasoning (TIR) (Jin et al., 2025; Li et al., 2025; Xue et al., 2025; Feng et al., 2025; Dong et al., 2025) enables the model to reason with external tools, such as a Python interpreter. Extending this idea to multimodal settings, the paradigm of Thinking with Images (TWI) has emerged as an effective approach (Lu et al., 2025; Su et al., 2025b;; Lai et al., 2025; Wang et al., 2025d; Wu et al., 2025b; Zhou et al., 2025). Instead of relying solely on textual reasoning, models are equipped with a predefined set of visual tools such as cropping, zooming, or rotating, which allow them to refine perception during problem solving. Recently, there has been a growing trend of enabling MLLMs to generate executable code as part of the reasoning process (Tang et al., 2025; Zhao et al., 2025; Hu et al., 2024; Zhang et al., 2025; Wang et al., 2025a), showcasing the potential of the TVP.

## 6 Conclusion

In this work, we introduced MMR-VIP, a benchmark designed to evaluate multimodal agentic reasoning under the Thinking with Visual Programming paradigm. Beyond text-based CoT and fixed visual tools, TVP allows models to flexibly generate, execute, and refine programmatic code, which serve as intermediate reasoning steps to facilitate multimodal problem solving. MMR-VIP is specifically crafted for this paradigm, featuring problems that are unsolvable under CoT-based reasoning but become tractable when integrated with visual programming interactions. Progress in multimodal agentic reasoning will depend critically on strengthening models' coding proficiency, enhancing their visual abstraction ability, and equipping them with multi-round iterative reasoning strategies.

# ETHICS STATEMENT

All experimental procedures involving human participants were conducted in accordance with the relevant ethical guidelines. Moreover, all data instances in our benchmark are puzzle-style problems that are automatically synthesized through scripts rather than collected from real-world human data. As such, the dataset contains no personal, harmful, or biased information. This ensures that MMR-VIP is entirely safe for research and avoids introducing any sensitive or ethically problematic content.

## REPRODUCIBILITY STATEMENT

Our dataset is entirely script-synthesized rather than manually annotated or generated by LLMs, ensuring full reproducibility. To facilitate this, we will release the synthesis scripts with fixed random seeds alongside the final MMR-VIP dataset. We also provide data examples in the supplementary materials. In addition, we provide detailed prompts used in all experiments in Appendix D, and we will open-source the evaluation code together with the Python interpreter environment. This guarantees that researchers can faithfully reproduce our experimental results.

# REFERENCES

- Anthropic. Introducing claude 4, May 2025. URL https://www.anthropic.com/news/claude-4.
- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923, 2025. doi: 10.48550/ARXIV.2502.13923. URL https://doi.org/10.48550/arXiv.2502.13923.
- Yew Ken Chia, Vernon Toh, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. Puzzlevqa: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, pp. 16259–16273. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.962. URL https://doi.org/10.18653/v1/2024.findings-acl.962.
- Chris, Yichen Wei, Yi Peng, Xiaokun Wang, Weijie Qiu, Wei Shen, Tianyidan Xie, Jiangbo Pei, Jianhao Zhang, Yunzhuo Hao, Xuchen Song, Yang Liu, and Yahui Zhou. Skywork R1V2: multimodal hybrid reinforcement learning for reasoning. *CoRR*, abs/2504.16656, 2025. doi: 10.48550/ARXIV.2504.16656. URL https://doi.org/10.48550/arXiv.2504.16656.
- Google DeepMind. Gemini flash, 2025. URL https://deepmind.google/technologies/gemini/flash/.
- Yihe Deng, Hritik Bansal, Fan Yin, Nanyun Peng, Wei Wang, and Kai-Wei Chang. Openvlthinker: An early exploration to complex vision-language reasoning via iterative self-improvement. *CoRR*, abs/2503.17352, 2025. doi: 10.48550/ARXIV.2503.17352. URL https://doi.org/10.48550/arXiv.2503.17352.
- Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. *CoRR*, abs/2507.19849, 2025. doi: 10. 48550/ARXIV.2507.19849. URL https://doi.org/10.48550/arXiv.2507.19849.
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *CoRR*, abs/2504.11536, 2025. doi: 10.48550/ARXIV.2504.11536. URL https://doi.org/10.48550/arXiv.2504.11536.

Deepanway Ghosal, Vernon Toh, Yew Ken Chia, and Soujanya Poria. Algopuzzlevqa: Diagnosing multimodal reasoning challenges of language models with algorithmic multimodal puzzles. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 9615–9632. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.486. URL https://doi.org/10.18653/v1/2025.naacl-long.486.

Ziyu Guo, Renrui Zhang, Hao Chen, Jialin Gao, Dongzhi Jiang, Jiaze Wang, and Pheng-Ann Heng. Sciverse: Unveiling the knowledge comprehension and visual reasoning of lmms on multi-modal scientific problems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 19683–19704. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.1010/.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Wenkai Li, Wei Jia, Xin Lyu, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuxuan Zhang, Zhanxiao Du, Zhenyu Hou, Zhao Xue, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *CoRR*, abs/2507.01006, 2025. doi: 10.48550/ARXIV.2507.01006. URL https://doi.org/10.48550/arXiv.2507.01006.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper\_files/paper/2024/hash/fb82011040977c7712409fbdb5456647-Abstract-Conference.html.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *CoRR*, abs/2503.06749, 2025. doi: 10.48550/ARXIV.2503.06749. URL https://doi.org/10.48550/arXiv.2503.06749.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL https://doi.org/10.48550/arXiv.2503.09516.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Róbert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy,

```
Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucinska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, and Ivan Nardini. Gemma 3 technical report. CoRR, abs/2503.19786, 2025. doi: 10.48550/ARXIV. 2503.19786. URL https://doi.org/10.48550/arxiv.2503.19786.
```

- Xin Lai, Junyi Li, Wei Li, Tao Liu, Tianjian Li, and Hengshuang Zhao. Mini-o3: Scaling up reasoning patterns and interaction turns for visual search, 2025. URL https://arxiv.org/abs/2509.07969.
- Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated RL. CoRR, abs/2503.23383, 2025. doi: 10.48550/ARXIV.2503.23383. URL https://doi.org/10.48550/arXiv.2503.23383.
- Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2508.19201.
- Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024.* OpenReview.net, 2024. URL https://openreview.net/forum?id=KUNzEQMWU7.
- Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *CoRR*, abs/2502.11271, 2025. doi: 10. 48550/ARXIV.2502.11271. URL https://doi.org/10.48550/arXiv.2502.11271.
- Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, Ping Luo, Yu Qiao, Qiaosheng Zhang, and Wenqi Shao. Mm-eureka: Exploring visual aha moment with rule-based large-scale reinforcement learning. *CoRR*, abs/2503.07365, 2025. doi: 10.48550/ARXIV.2503.07365. URL https://doi.org/10.48550/arXiv.2503.07365.
- OpenAI. Hello gpt-4o, 2024. URL https://openai.com/index/hello-gpt-4o/.
- OpenAI. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025a.
- OpenAI. Gpt-4.1. https://openai.com/index/gpt-4-1/, 2025b.
- Yueqi Song, Tianyue Ou, Yibo Kong, Zecheng Li, Graham Neubig, and Xiang Yue. Visualpuzzles: Decoupling multimodal reasoning evaluation from domain knowledge. *CoRR*, abs/2504.10342, 2025. doi: 10.48550/ARXIV.2504.10342. URL https://doi.org/10.48550/arXiv.2504.10342.
- Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel-space reasoning with curiosity-driven reinforcement learning. *CoRR*, abs/2505.15966, 2025a. doi: 10.48550/ARXIV.2505.15966. URL https://doi.org/10.48550/arXiv.2505.15966.
- Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, and Yu Cheng. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *CoRR*, abs/2505.08617, 2025b. doi: 10.48550/ARXIV.2505.08617. URL https://doi.org/10.48550/arxiv.2505.08617.
- Zhaochen Su, Peng Xiang, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, Linjie Li, Yu Cheng, Heng Ji, Junxian He, and Yi R. (May) Fung. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *CoRR*, abs/2506.23918, 2025c. doi: 10.48550/ARXIV.2506.23918. URL https://doi.org/10.48550/arxiv.2506.23918.

Bohao Tang, Yan Ma, Fei Zhang, Jiadi Su, Ethan Chern, Zhulin Hu, Zhixin Wang, Pengfei Liu, and Ya Zhang. Visual programmability: A guide for code-as-thought in chart understanding, 2025. URL https://arxiv.org/abs/2509.09286.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper\_files/paper/2024/hash/ad0edc7d5fala783f063646968b7315b-Abstract-Datasets\_and\_Benchmarks\_Track.html.

Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang, Houxing Ren, Mingjie Zhan, and Hongsheng Li. Mathcoder-vl: Bridging vision and code for enhanced multimodal mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics*, *ACL 2025*, *Vienna*, *Austria*, *July 27 - August 1*, 2025, pp. 2505–2534. Association for Computational Linguistics, 2025a. URL https://aclanthology.org/2025.findings-acl.128/.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, Guanzhou Chen, Zichen Ding, Changyao Tian, Zhenyu Wu, JingJing Xie, Zehao Li, Bowen Yang, Yuchen Duan, Xuehui Wang, Zhi Hou, Haoran Hao, Tianyi Zhang, Songze Li, Xiangyu Zhao, Haodong Duan, Nianchen Deng, Bin Fu, Yinan He, Yi Wang, Conghui He, Botian Shi, Junjun He, Yingtong Xiong, Han Lv, Lijun Wu, Wenqi Shao, Kaipeng Zhang, Huipeng Deng, Biqing Qi, Jiaye Ge, Qipeng Guo, Wenwei Zhang, Songyang Zhang, Maosong Cao, Junyao Lin, Kexian Tang, Jianfei Gao, Haian Huang, Yuzhe Gu, Chengqi Lyu, Huanze Tang, Rui Wang, Haijun Lv, Wanli Ouyang, Limin Wang, Min Dou, Xizhou Zhu, Tong Lu, Dahua Lin, Jifeng Dai, Weijie Su, Bowen Zhou, Kai Chen, Yu Qiao, Wenhai Wang, and Gen Luo. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *CoRR*, abs/2508.18265, 2025b. doi: 10.48550/ARXIV.2508.18265. URL https://doi.org/10.48550/arxiv.2508.18265.

Xiyao Wang, Zhengyuan Yang, Chao Feng, Hongjin Lu, Linjie Li, Chung-Ching Lin, Kevin Lin, Furong Huang, and Lijuan Wang. Sota with less: Mcts-guided sample selection for data-efficient visual reasoning self-improvement. *CoRR*, abs/2504.07934, 2025c. doi: 10.48550/ARXIV.2504. 07934. URL https://doi.org/10.48550/arXiv.2504.07934.

Ye Wang, Qianglong Chen, Zejun Li, Siyuan Wang, Shijie Guo, Zhirui Zhang, and Zhongyu Wei. Simple o3: Towards interleaved vision-language reasoning, 2025d. URL https://arxiv.org/abs/2508.12109.

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 Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022. URL http://papers.nips.cc/paper\_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Fang Wu, Weihao Xuan, Ximing Lu, Zaïd Harchaoui, and Yejin Choi. The invisible leash: Why RLVR may not escape its origin. *CoRR*, abs/2507.14843, 2025a. doi: 10.48550/ARXIV.2507. 14843. URL https://doi.org/10.48550/arXiv.2507.14843.

Junfei Wu, Jian Guan, Kaituo Feng, Qiang Liu, Shu Wu, Liang Wang, Wei Wu, and Tieniu Tan. Reinforcing spatial reasoning in vision-language models with interwoven thinking and visual drawing. *CoRR*, abs/2506.09965, 2025b. doi: 10.48550/ARXIV.2506.09965. URL https://doi.org/10.48550/arXiv.2506.09965.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2509.02479.

Biao Yang, Bin Wen, Boyang Ding, Changyi Liu, Chenglong Chu, Chengru Song, Chongling Rao, Chuan Yi, Da Li, Dunju Zang, Fan Yang, Guorui Zhou, Guowang Zhang, Han Shen, Hao Peng, Haojie Ding, Hao Wang, Haonan Fan, Hengrui Ju, Jiaming Huang, Jiangxia Cao, Jiankang Chen, Jingyun Hua, Kaibing Chen, Kaiyu Jiang, Kaiyu Tang, Kun Gai, Muhao Wei, Qiang Wang, Ruitao Wang, Sen Na, Shengnan Zhang, Siyang Mao, Sui Huang, Tianke Zhang, Tingting Gao, Wei Chen, Wei Yuan, Xiangyu Wu, Xiao Hu, Xingyu Lu, Yi-Fan Zhang, Yiping Yang, Yulong Chen, Zeyi Lu, Zhenhua Wu, Zhixin Ling, Zhuoran Yang, Ziming Li, Di Xu, Haixuan Gao, Hang Li, Jing Wang, Lejian Ren, Qigen Hu, Qianqian Wang, Shiyao Wang, Xinchen Luo, Yan Li, Yuhang Hu, and Zixing Zhang. Kwai keye-vl 1.5 technical report, 2025. URL https://arxiv.org/abs/2509.01563.

Xiang Yue, Yuansheng Ni, Tianyu Zheng, Kai Zhang, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR* 2024, Seattle, WA, USA, June 16-22, 2024, pp. 9556–9567. IEEE, 2024. doi: 10.1109/CVPR52733.2024.00913. URL https://doi.org/10.1109/CVPR52733.2024.00913.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, Peng Gao, and Hongsheng Li. MATHVERSE: does your multi-modal LLM truly see the diagrams in visual math problems? In Ales Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part VIII*, volume 15066 of *Lecture Notes in Computer Science*, pp. 169–186. Springer, 2024a. doi: 10.1007/978-3-031-73242-3\\_10. URL https://doi.org/10.1007/978-3-031-73242-3 10.

Yifan Zhang, Xingyu Lu, Shukang Yin, Chaoyou Fu, Wei Chen, Xiao Hu, Bin Wen, Kaiyu Jiang, Changyi Liu, Tianke Zhang, Haonan Fan, Kaibing Chen, Jiankang Chen, Haojie Ding, Kaiyu Tang, Zhang Zhang, Liang Wang, Fan Yang, Tingting Gao, and Guorui Zhou. Thyme: Think beyond images. *CoRR*, abs/2508.11630, 2025. doi: 10.48550/ARXIV.2508.11630. URL https://doi.org/10.48550/arXiv.2508.11630.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *Trans. Mach. Learn. Res.*, 2024, 2024b. URL https://openreview.net/forum?id=y1pPWFVfvR.

Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling, 2025. URL https://arxiv.org/abs/2507.07998.

Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deepeyes: Incentivizing "thinking with images" via reinforcement learning. *CoRR*, abs/2505.14362, 2025. doi: 10.48550/ARXIV.2505.14362. URL https://doi.org/10.48550/arXiv.2505.14362.

Zetong Zhou, Dongping Chen, Zixian Ma, Zhihan Hu, Mingyang Fu, Sinan Wang, Yao Wan, Zhou Zhao, and Ranjay Krishna. Reinforced visual perception with tools, 2025. URL https://arxiv.org/abs/2509.01656.

# A LLM USAGE STATEMENT

In this work, Large Language Models (LLMs) were used solely as general-purpose auxiliary tools. Their role was limited to polishing grammar and phrasing to enhance the clarity of the manuscript, as well as assisting in the generation of Python and LaTeX code for creating figures and tables. No parts of the research ideation, experimental design, analysis, or substantive writing relied on LLMs.

## **B** DISCUSSION

Here, we would like to discuss the relationship between Thinking with Images (TWI) and Thinking with Visual Programming (TVP).

Existing approaches under the Thinking with Images paradigm typically rely on a predefined set of visual tools, such as cropping, zooming, and rotating. These operations can indeed enhance perceptual accuracy, especially for handling high-resolution images or focusing attention on relevant regions. However, their scope is inherently narrow. While effective for improving low-level perception, such fixed transformations provide limited support for deep reasoning tasks that require abstraction, planning, or algorithmic optimization. In other words, current Thinking with Images primarily enhances *seeing more carefully*, but does not necessarily enable *thinking more deeply*.

In contrast, Thinking with Visual Programming generalizes beyond fixed toolkits by allowing models to write and execute code, thus treating visual operations themselves as programmable functions. This enables not only flexible tool selection but also the creation of new tools on demand, allowing the reasoning process to adapt dynamically to the task at hand. Under this view, cropping or rotating an image represents only one instance within a broader spectrum of programmable operations, which may also involve algorithmic simulation, complex computation, or visualization.

From this perspective, TWI can be regarded as a subset of TVP, serving as a valuable stepping stone but not the ultimate goal. As our experimental results demonstrate, current models remain far from fully realizing the TVP paradigm. While existing studies have already achieved promising outcomes under the TWI framework, a substantial gap persists between these methods and the broader vision of TVP. Bridging this gap requires equipping models with stronger visual programming capabilities and more advanced visual abstraction skills, enabling them to move beyond fixed perceptual tools toward flexible, programmable reasoning. On this foundation, agentic reinforcement learning can become truly effective. In the future, we envision equipping MLLMs with access to external resources such as web browsers. This would allow them not only to autonomously create tools through code but also to search for and integrate existing tools from the internet.

# C BENCHMARK DETAILS

To ensure the data quality of MMR-VIP, we provided annotators with a detailed guideline:

#### All tasks must:

- (1) Be code-synthesizable (problems, images, and solutions are generated by code).
- (2) Be aligned with cognitive skills (at least 1, at most 3 from the given taxonomy).
- (3) Be stratified into difficulty levels (Easy / Medium / Hard).
- (4) Be suitable for programmatic reasoning (problems solvable or aided by code execution).

#### Cognitive Skills

- (1) Attribute: identify colors, shapes, sizes.
- (2) Location: detect positions, distances, spatial relations.
- (3) Symbolic: recognize digits, letters, or visual symbols.
- (4) Geometry: formulate geometric equations or relations.
- (5) Physics: model dynamics using physical laws.
- (6) Network: construct graph structures (nodes, edges, constraints).
- (7) Search: implement DFS, BFS, or other exploration methods.
- (8) Planning: apply dynamic/linear programming to solve constrained problems.
- (9) Computation: perform numerical calculations or algorithmic procedures.

## **Difficulty Levels**

- (1) Easy: solvable using intrinsic perceptual and reasoning abilities, without code execution.
- (2) Medium: requiring programmatic operations, where external computation is essential.
- (3) Hard: remaining challenging even with programming support, typically due to high algorithmic complexity or intricate constraints.

# Workflow

Annotators should first define the problem (including its target cognitive skills and difficulty levels), then implement code that generates instances and computes the ground-truth solution. Next, the problem must be visualized using standard libraries to ensure clarity. Each program should support batch generation of images, questions, and answers across difficulty levels. Finally, the generated code must undergo validation, where outputs are independently reviewed to ensure correctness and consistency between problem, visualization, and answer.

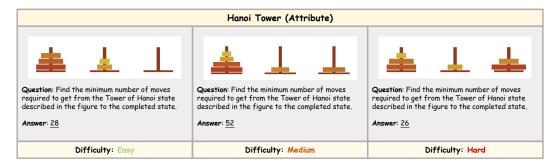


Figure 7: Data example of Hanoi Tower.



Figure 8: Data example of Sliding Puzzle.

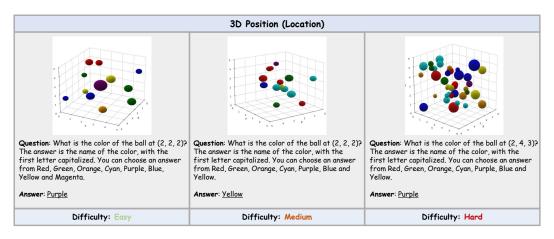


Figure 9: Data example of 3D Position.

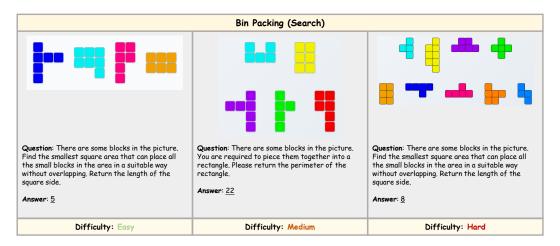


Figure 10: Data example of Bin Packing.

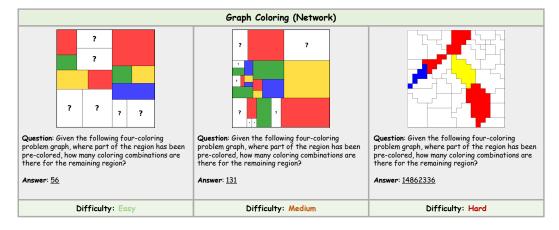


Figure 11: Data example of Graph Coloring.

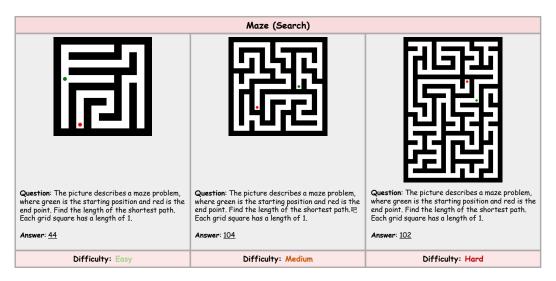


Figure 12: Data example of Maze.

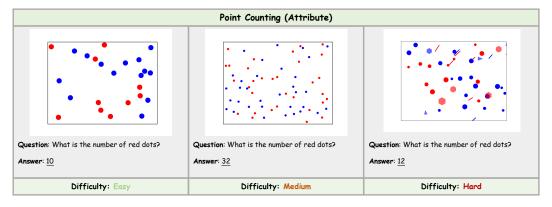


Figure 13: Data example of Point Counting.

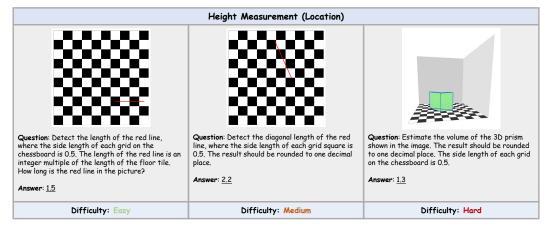


Figure 14: Data example of Height Measurement.

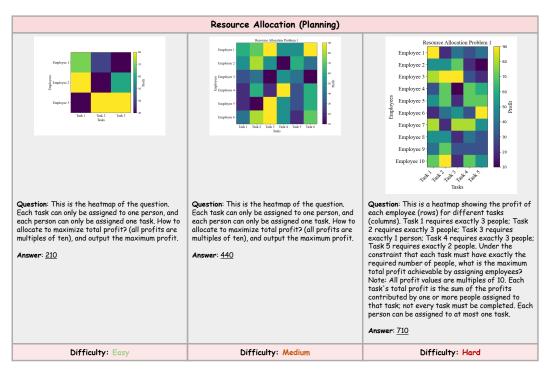


Figure 15: Data example of Resource Allocation.

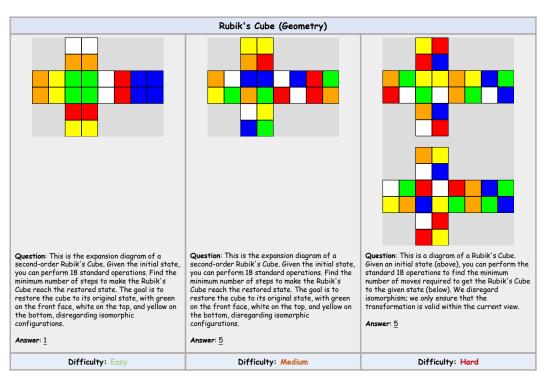


Figure 16: Data example of Rubik's Cube.

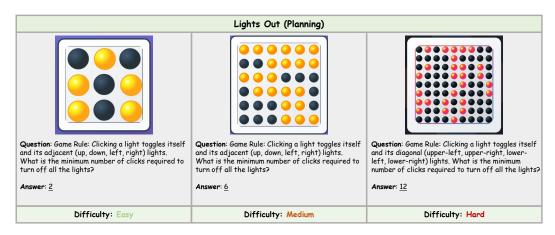


Figure 17: Data example of Lights Out.

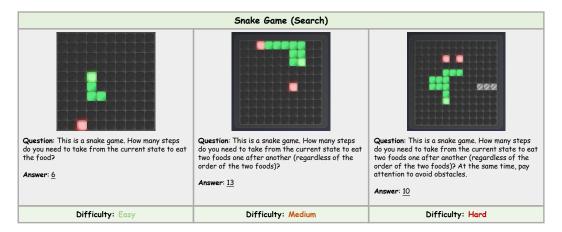


Figure 18: Data example of Snake Game.

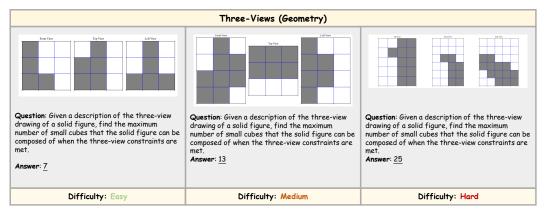


Figure 19: Data example of Three-Views.

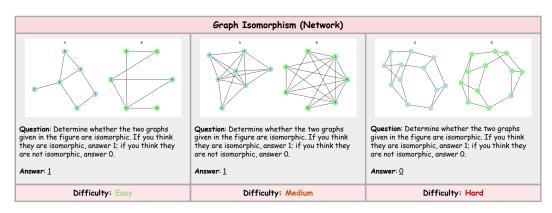


Figure 20: Data example of Graph Isomorphism.

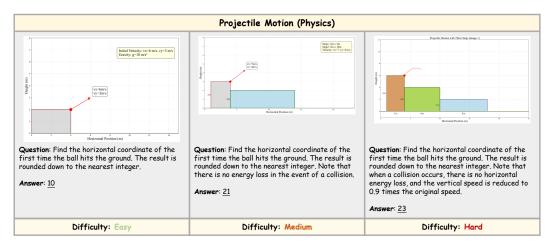


Figure 21: Data example of Projectile Motion.

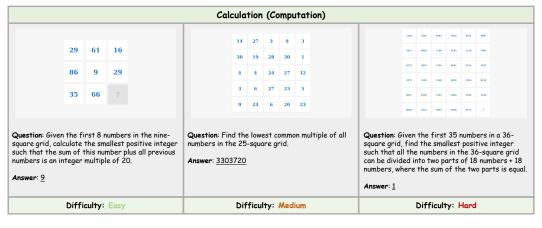


Figure 22: Data example of Calculation.

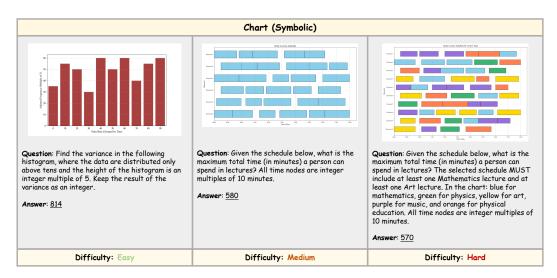


Figure 23: Data example of Chart.

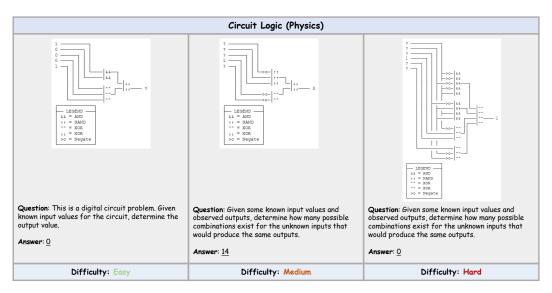


Figure 24: Data example of Circuit Logic.

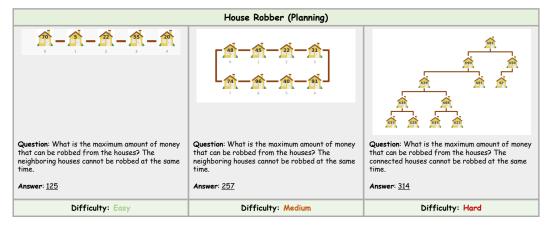


Figure 25: Data example of House Robber.

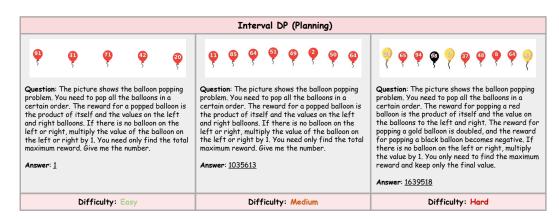


Figure 26: Data example of Interval DP.

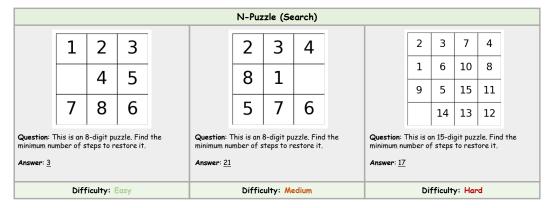


Figure 27: Data example of N-Puzzle.

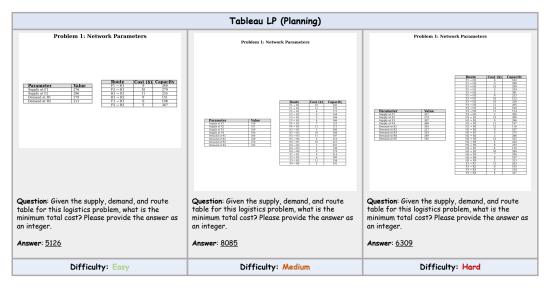


Figure 28: Data example of Tableau LP.

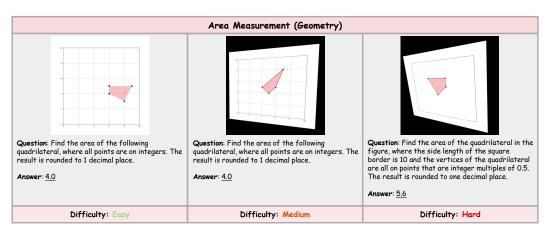


Figure 29: Data example of Area Measurement.

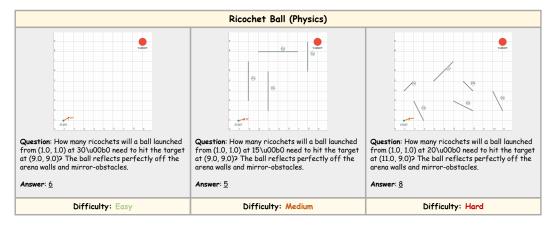


Figure 30: Data example of Ricochet Ball.

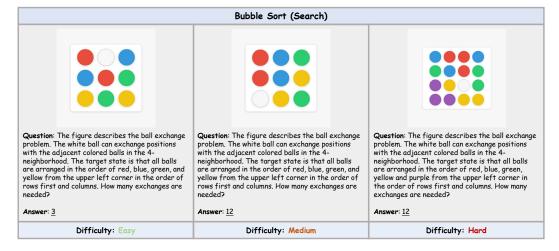


Figure 31: Data example of Bubble Sort.

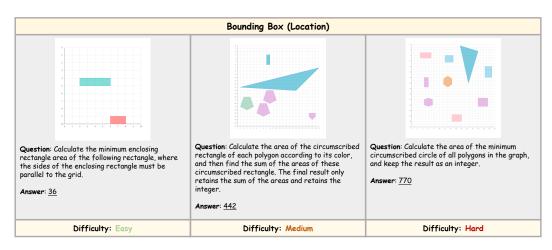


Figure 32: Data example of Bounding Box.

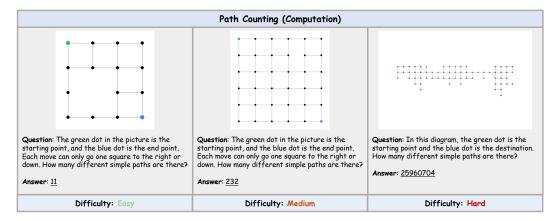


Figure 33: Data example of Path Counting.



Figure 34: Data example of N-Queens.

# D EVALUATION DETAILS

We provide the prompts for both direct CoT reasoning and multi-turn TVP reasoning, as illustrated in Figure 35 and Figure 36.

#### Prompt for Direct CoT

#### **System Prompt:**

1350

1351 1352

1353

1354 1355

1356

1357

1358

1359

1363 1364 1365

1367

1369

1370

1371

1372

1373 1374

1375

1376

1380

1381

1382

1386

1387

1388

1389

1390

1391

1392

1393

1394

1399

1400

1401 1402 1403 You FIRST think about the reasoning process as an internal monologue and then provide the final answer. The reasoning process MUST BE enclosed within <think> </think> tags. The final answer MUST BE put in \boxed{}. Please note that if the answer requires a numerical value, please keep only the number without punctuation, units, formulas or explanations. Don't run code in your own environment.

Figure 35: Prompt for direct CoT reasoning.

#### Prompt for Multi-turn TVP

# **System Prompt:**

You are a visual reasoning assistant that MUST write executable Python code to solve problems. You can iterate through multiple rounds to refine your solution (maximum {N} code executions).

#### **IMPORTANT CODE FORMATTING RULES:**

- You MUST wrap your code EXACTLY with <code> and </code> tags
- Do NOT use backticks ('), triple-backticks ('''), or any other delimiters
- Inside <code>...</code> put only valid Python code
- Do NOT HTML-escape characters (use <, >, &, not &lt;, &gt;, &amp;)

#### HELPER FUNCTIONS

```
1 import os
2 import re
3 import typing
4 def find_original_image_name(work_dir: str = '.') -> typing.Optional[str]:
         'Find the original image filename, excluding processed versions
      for f in sorted(os.listdir(work_dir)):
          if not f.lower().endswith('.png'): continue
          if f.startswith('crop_'): continue
          if re.search(r'_m(?:\d+)?\.png$', f): continue
          return f
      return None
12 def processed_image_name(original_image: str) -> str:
        ''Return processed image filename for current iteration'''
14
      base, ext = os.path.splitext(original_image)
      return f'{{base}}_m{iteration}{{ext}}}
```

#### **CODE REQUIREMENTS:**

- Use find\_original\_image\_name() to locate the input image
- Save your processed image using processed\_image\_name() (will be \*\_m{iteration}.png)
- Use only relative paths and work within the current directory
- Do not access network or write outside the current folder

### If iteration == 1:

This is your FIRST iteration. Analyze the image and question carefully, then write Python code to solve it. Focus on understanding the problem and implementing a basic solution.

#### Else:

This is iteration {iteration}/{N}. You can see your previous attempts and their results in the conversation history. Analyze what went wrong in previous iterations and improve your approach. Consider the execution results and any generated images from previous attempts.

### **Prompt for Final Answer Integration:**

```
=== FINAL INTEGRATION ===
```

Based on all your previous attempts, code executions, and any generated images, please provide your final answer to the original question. Original question: {question}

Format your final answer using \\boxed{answer} notation.

Figure 36: Prompt for multi-turn TVP reasoning.

# E ADDITIONAL EXPERIMENTAL RESULTS

As a human reference, we randomly sampled 168 instances from the benchmark and invited three participants to solve these tasks. All participants were PhD students with strong programming backgrounds. On average, each task required approximately 8 minutes to complete. During the process, participants were allowed to write code and make use of search engines to access external resources and tools when necessary.

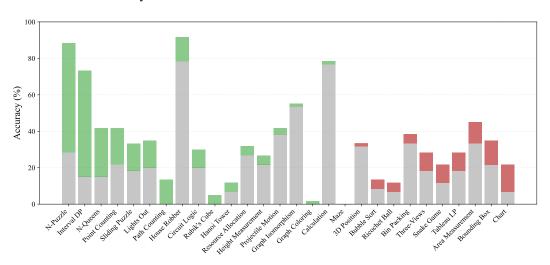


Figure 37: Performance comparison of Claude-Sonnet-4 on different tasks under CoT and TVP (T=1). Gray indicates the baseline performance of CoT, Green indicates improvements of TVP over CoT, and Red indicates degradations of TVP over CoT.

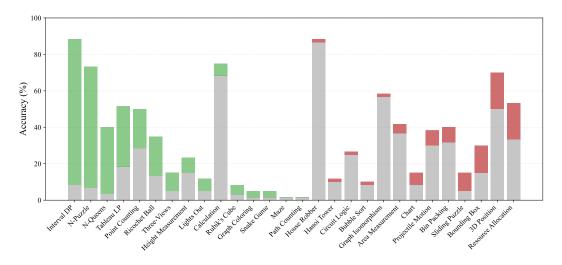


Figure 38: Performance comparison of GPT-4.1 on different tasks under CoT and TVP (T=5). Gray indicates the baseline performance of CoT, Green indicates improvements of TVP over CoT, and Red indicates degradations of TVP over CoT.

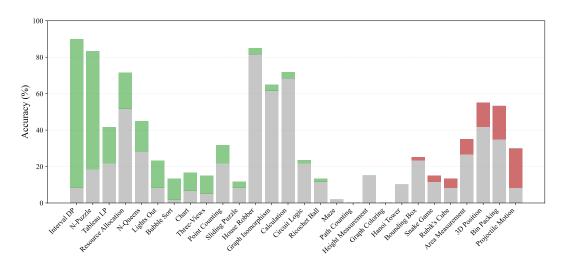


Figure 39: Performance comparison of GPT-4.1-mini on different tasks under CoT and TVP (T=5). Gray indicates the baseline performance of CoT, Green indicates improvements of TVP over CoT, and Red indicates degradations of TVP over CoT.

# F ERROR ANALYSIS

 To better understand the limitations of TVP, we classify incorrect predictions into six categories:

- Problem Misinterpretation. The model misunderstands the problem requirements or constraints, leading it to pursue an incorrect solution approach.
- **Program Execution Error.** The generated code contains syntax, logical, or runtime errors that prevent successful execution.
- Visual Perception Error. The model fails to accurately extract relevant information from visual inputs, such as misrecognizing digits, shapes, or object positions.
- Algorithmic Modeling Error. The model adopts an inappropriate algorithm, data structure, or fails to encode the relationships and constraints necessary for solving the problem.
- **Program Inefficiency.** Although the solution logic is correct, the implementation is computationally inefficient and cannot handle the input within reasonable time limits.
- Output Formatting Issue. The model produces outputs in an incorrect format (e.g., explanatory text instead of a numeric answer).

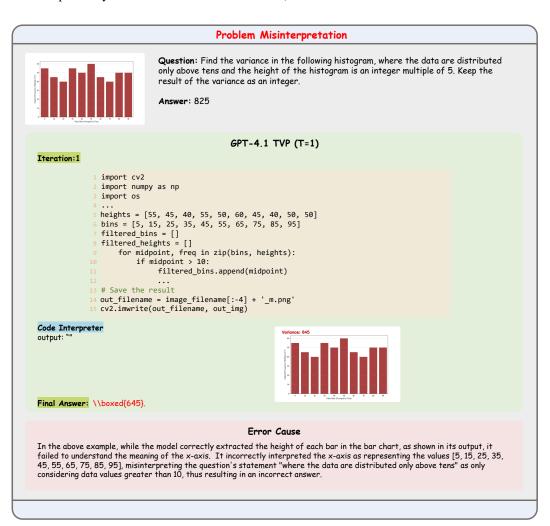


Figure 40: Problem Misinterpretation.

```
Program Execution Error
                         Question: This is an 8-digit puzzle. Find the minimum number of steps to restore it.
              3
1
       2
                         Answer: 5
4
              8
       6
              5
7
                                                     GPT-4.1-mini TVP (T=1)
Iteration:1
                 import cv2
import numpy as np
                  from collections import deque
                 def neighbors(state):
                      idx = state.index(0)
r, c = divmod(idx, 3)
res = []
                       for dr, dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                           nr, nc = r + dr, c + dc
                 # Save the result image
                 cv2.putText(output_img, text, (10, h-10), cv2.FONT_HERSHEY_SIMPLEX, 1,
                 (0,0,255), 2)
                 cv2.imwrite(out_filename, output_img)
Code Interpreter
error: Traceback (most recent call last):\n... line 154, in <module>\n grid[row_i][col_i] = digit\n ~~~^^^^^^\nIndexError: list index out of range\n.
Final Answer: The puzzle initial state from the image is:\n\n1 2 3 \n4 6 8 \n7 0 5 \n\n...Using BFS on the 8-puzzle starting
from the given state to the goal state, the minimum number of steps to solve it is: \n\
                                                             Error Cause
In the initial answer to this question, an array index out-of-bounds error caused the program to crash. In the subsequent summary answer, the model claimed that it had obtained the solution using the BFS algorithm, resulting in 6 steps (presumably
based on its own reasoning), whereas the correct answer is 5 steps.
```

Figure 41: Program Execution Error.

```
1627
1628
1629
                                                                    Visual Perception Error
1630
                                          Question: Game Rule: Clicking a light toggles itself and its diagonal (upper-left, upper-right, lower-left, lower-right) lights. What is the minimum number of clicks required to turn off all the
1631
                                          lights?
1632
1633
                                          Answer: 10
1634
1635
1636
                                                                 Gemini-2.5-Flash TVP (T=1)
1637
                   Iteration:1
1638
                   The problem describes a \"lights out\" type puzzle where clicking a light toggles its state and the state of its four diagonal
                   neighbors..
1639
                                     import cv2
import numpy as np
1640
                                     import os
1641
1642
                                     for r_click in range(num_rows):
                                          for c_click in range(num_cols):
1643
                                              k_click = rc_to_idx(r_click, c_click)
A[rc_to_idx(r_click, c_click), k_click] = 1
if r_click - 1 >= 0 and c_click - 1 >= 0:
1644
                                                   A[rc_to_idx(r_click - 1, c_click - 1), k_click] = 1
1645
                                     # Save the result image
1646
                                     output_filename = os.path.splitext(image_filename)[0] + '_m.png'
1647
                                     cv2.imwrite(output_filename, result_img)
print(f"Solution image saved as {output_filename}")
1648
1649
                   1650
                   of clicks required: 31.
1651
1652
1653
1654
1655
                   Final Answer: The minimum number of clicks required to turn off all the lights is 31.
1656
1657
                                                                           Error Cause
1658
                   Based on the description, it seems the model understood the game rules, but differed from standard approaches to solving
1659
                   similar puzzle problems. During the code execution, it generated an incorrect matrix representation of the current state, leading
                   to an incorrect solution. In the final step of summarizing and organizing the answer, the model failed to detect any anomalies in
1660
                   the output image, ultimately resulting in the error.
1661
1662
1663
```

Figure 42: Visual Perception Error.

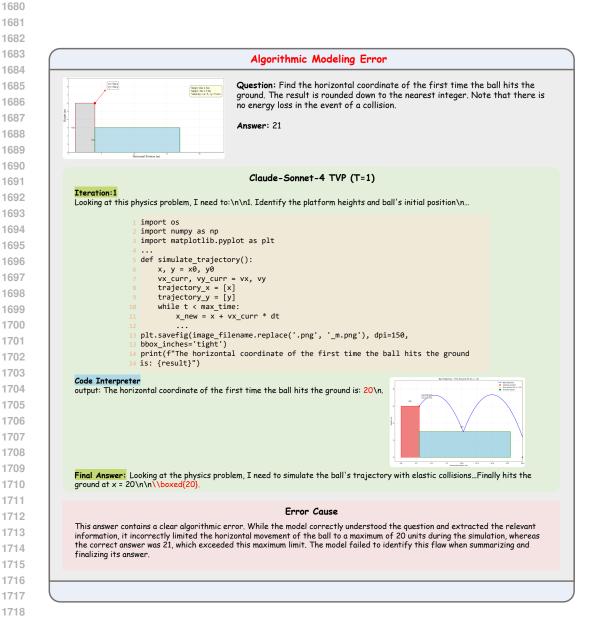


Figure 43: Algorithmic Modeling Error.

```
Program Inefficiency
                                                                 Question: There are some blocks in the picture. Find the smallest
                                                                 square area that can place all the small blocks in the area in a suitable
                                                                 way without overlapping. Return the length of the square side.
                                                                  Answer: 4
                                                                GPT-4.1-mini TVP (T=1)
Iteration:1
                         from PIL import Image
                        import numpy as np
import os
                         def can_place(board, block, r, c):
                               h, w = board.shape
bh, bw = block.shape
                               if r + bh > h or c + bw > w:
                        def backtrack(board, blocks, idx):
                               if idx == len(blocks):
                        blocks = extract_blocks(image)
answer = solve(blocks)
print(answer)
error: Code execution timed out after 60 seconds.
Final Answer: \\boxed{5}.
                                                                          Error Cause
In the example above, the algorithm first tries to count the total number of small squares, and then starts by trying the smallest possible square under those conditions, using a BFS method to find a feasible arrangement. However, this algorithm is too inefficient and fails to find a solution within the 60-second time limit. Therefore, the model incorrectly concluded that the
minimum side length should be at least 5.
```

Figure 44: Program Inefficiency.

# G CASE STUDY

 We provide several case studies to analyze the performance of CoT, single-turn TVP, and multi-turn TVP. As illustrated in Figure 45, the model uses code to abstract three-dimensional views for solving the task. In contrast, CoT can only describe this process textually, lacking the ability to perform direct computation or manipulation of the visual input. As illustrated in Figure 46, TVP utilizes the model's advanced coding capabilities to accurately model the Rubik's Cube rotation rules. By integrating this with a search algorithm, TVP provides a precise solution, demonstrating the power of visual programming in solving complex tasks. However, for simpler problems shown in Figure 47, writing code may actually introduce errors, as the overhead of coding can outweigh the benefits of using a more direct reasoning approach. When comparing single-turn TVP with multi-turn TVP, we observe that the latter enables models to iteratively refine their outputs, effectively correcting coding mistakes through agentic interaction, as shown in Figure 48. Furthermore, Figure 49 shows cases where the model employs visualizations as an intermediate verification mechanism, ensuring that no red points are overlooked and demonstrating reflective reasoning through explicit inspection of its own outputs.

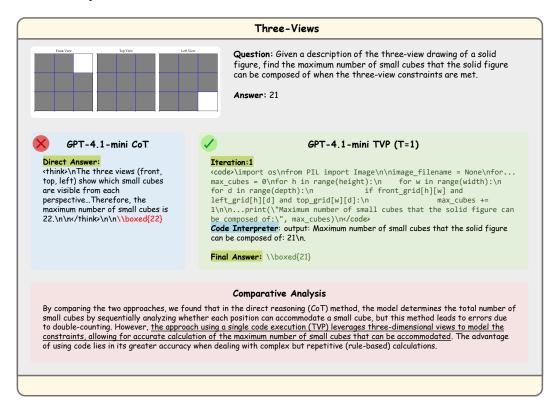


Figure 45: Case study of Tree-Views.

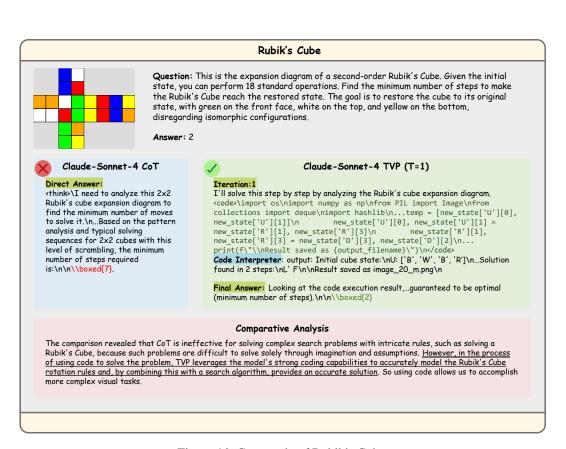


Figure 46: Case study of Rubik's Cube.

#### Area Measurement Question: Find the area of the following quadrilateral, where all points are on integers. The result is rounded to 1 decimal place. Answer: 6.0 Qwen2.5-VL-32B CoT Qwen2.5-VL-32B TVP (T=1) Direct Answer: Iteration:1 <think>\To find the area of the I'll solve this step by step by analyzing the Rubik's cube expansion diagram. <code>\import os\nimport cv2\nimport numpy as np\nimport re\n...contours, \_ = cv2.findContours(image=mask, mode=cv2. RETR\_LIST, given triangle, we can use the Shoelace Theorem...calculate the re\n...contours, \_ = cv2.findContours(image=mask, mode=cv2. RETR\_LIST, method=cv2.CHAIN\_APPROX\_SIMPLE)\ncentroids = []\nfor cnt in contouns:\n M = cv2.moments(contour)\n if M['m00'] != 0:\n cx = int(M['m10'] / M['m00'])\n cy = int(M['m01'] / M['m00'])\n centroids.append((cy, cx))...# Save the marked image\noutput\_filename = re.sub(r'(\\.png)', 'm\\\1', image\_filename\ncv2.imwrite(output\_filename, cv2.cvtColor(image, cv2.cv area:\n\n\\[\n\\text{Area} = \\frac{1|2 \\cdot 12 = 6\n\]\n\nTherefore, the area of the triangle is:\n\n\\boxed{6.0}. cv2.COLOR\_BGR2RGB))\n\n</code> Code Interpreter: error: ...line 36\n contours, \_ = cv2.findContours\tmask, cv2. RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)\n ^\nSyntaxError: unmatched ')'\n Final Answer: None Comparative Analysis Comparing CoT and TVP (T=1), we can see that CoT accurately obtained the coordinates of the points directly from the imag information and then used the quadrilateral area formula to calculate the area accurately. However, the code provided by TVP attempted to identify the vertices of the quadrilateral to calculate the area, but it failed to correctly locate the vertices, resulting in an error. Therefore, for models with limited coding capabilities, using code to assist in problem-solving may actually lead to lower accuracy than simply providing a direct answer.

Figure 47: Case study of Area Measurement.

1989

1991 1992 1993

1995 1996 1997

1948 1949 1950 1951 Resource Allocation 1952 1953 Question: This is the heatmap of the question. Each task can only be assigned to one 1954 person, and each person can only be assigned one task. How to allocate to maximize total profit? (all profits are multiples of ten), and output the maximum profit. Above is the image 1955 of this auestion. Answer: 180 1957 1958 GPT-4.1 TVP (T=1) GPT-4.1 TVP (T=5) 1959 1960 Iteration: 1 Iteration:1 <code>\import os\nimport <code>\import numpy as np\nfrom scipy.optimize import 1961 linear\_sum\_assignment\n...print(max\_profit)\n</code>
Code Interpreter: error: ...NameError: name 'find\_original\_image\_name' is not cv2\nimport numpy as np\nimport 1962 pytesseract\n...print('Maximum 1963 Total Profit:',
max\_profit)\n</code> 1964 Code Interpreter: error: Traceback <code>\import numpy as np\nfrom scipy.optimize import (most recent call last):\n...raise TesseractNotFoundError()\npytes linear\_sum\_assignment\n...print(max\_profit)\n</code> 1965 Code Interpreter: output: 180\n. 1966 seract.pytesseract.TesseractNotF oundError: tesseract is not installed or it's not in your PATH. 1967 <code>\import numpy as np\nfrom scipy.optimize import linear\_sum\_assignment\n...print(\"Maximum Profit:\", See README file for more 1968 max\_profit)\n</code>
Code Interpreter: output: Allocation (Employee, Task): ... Profit: 180\n. information.\n 1969 Final Answer: 1970 \boxed{230} <code>\import numpy as np\nfrom scipy.optimize import linear\_sum\_assignment\n...print(\"Maximum Profit:\", 1971 1972 max\_profit)\n</code> Code Interpreter: output: Employee 1 assigned to Task 2, ... Total Profit: 180\n. 1973 1974 <code>\import numpy as np\nfrom scipy.optimize import
linear\_sum\_assignment\n...print(f\"Maximum Total Profit: 1975 {max\_profit}\")</code> 1976 Code Interpreter: output: Optimal Assignment:\nEmployee 1 -> Task 2...Total 1977 Final Answer: \boxed(180)\n\nOptimal allocation to maximize total profit:\n-Employee 1  $\rightarrow$  Task 2 (Profit: 90)\n- Employee 2  $\rightarrow$  Task 1 (Profit: 10)\n- Employee 3  $\rightarrow$  Task 3 (Profit: 80) 1978 1979 1980 1981 1982 Comparative Analysis Comparing the overall output and final results of single-round and five-round code execution, we can see that in the single-round execution, TVP (T=1) encountered an error because it used a library function that was not available in the environment (a non-1983 1984 standard library). <u>In the five-round execution TVP (T=5), a path loading error occurred during the first code execution, but</u> subsequent executions corrected this issue, ultimately producing the correct answer. This demonstrates the advantage of 1985 multiple rounds of execution: the ability to identify and correct errors. 1986 1987 1988

Figure 48: Case study of Area Measurement.

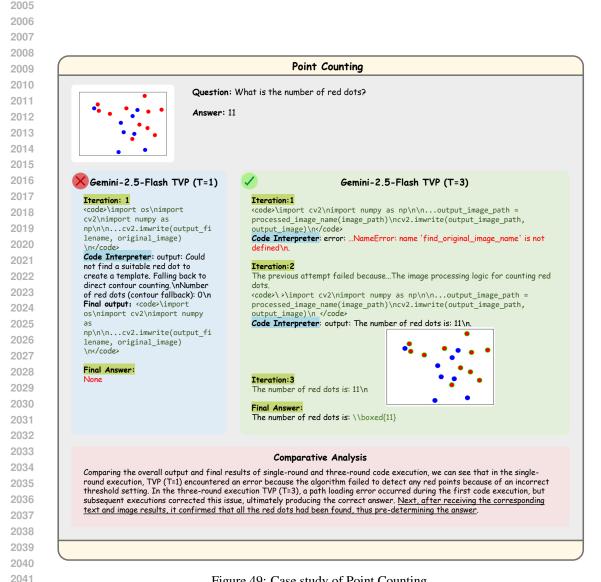


Figure 49: Case study of Point Counting.

2098

```
2057
2058
                                                                             Ricochet Ball
2060
                                          Question: How many ricochets will a ball launched from (1.0, 1.0) at 80^\circ need to hit the target at
2061
                                          (9.0, 9.0)? The ball reflects perfectly off the arena walls and mirror-obstacles.
2062
                                          Answer: 7
2063
2064
2065
2066
                      Gemini-2.5-Flash TVP (T=1)
                                                                                            Gemini-2.5-Flash TVP (T=3)
2067
                                                                Iteration: 1
2068
                                                                 <code>\import math\nimport os\nimport re\nimport typing\nimport
                    The problem requires simulating
                                                                the path of a ball launched from a
2069
                    starting point with a given angle...
2070
                    <code>\import os\nimport
math\nfrom PIL import Image,
                                                                 dy = current_pos.y - target_pos.y\n\n... # Print the final
answer\nprint(ricochets)/code>
2071
                    ImageDraw\n\n# --- Geometry
                                                                 Code Interpreter: output: "7\n"
                   Classes ---\nclass Point:\n
def __init__(self, x, y):\n...
# --- Simulation Loop ---\nfor
2072
2073
                                                                 The previous iteration's code correctly calculated 7 ricochets and produced a visual
                    step count in
                    range(MAX_SIM_STEPS):\n
                                                                path that supports this answer...
<code>\import math\nimport os\nimport re\nimport typing\nimport
                    Calculate t if the ray hits the target directly on its
2075
                                                                 matplotlib.pyplot as plt\nimport matplotlib.patches as
                    current path segment\n
                                                                 patches\n\n...def get_time_to_hit_wall(current_pos,
2076
                    t_to_target = float('inf')\n
\n vec_to_target_x =
                                                                                            dt_x = float('inf')\n
                                                                 current_velocity):\n
                                                                 2077
                    target pos.x - current pos.x\n
                                                                 Code Interpreter: error: .
2078
                    vec_to_target_y = target_pos.y
                                                                  find_original_image_name' is not defined\n
                    - current_pos.y\n\n
...print('Maximum Total
2079
                                                                Iteration:3
                        it:', max_profit)\n</code>
                                                                 <code>\import math\nimport os\nimport re\nimport typing\nimport
2080
                    Code Interpreter: output: -1\n
                                                                 matplotlib.pyplot as plt\nimport matplotlib.patches as patches\n\n...#
2081
                                                                Snap position to the wall\n if abs(current_pos_next.y - Y

EPS: current_pos_next.y = Y_MIN\n if abs(current_pos_next

Y_MAX) < EPS: current_pos_next.y = Y_MAX\n\n...# Print the final
                                                                                                        if abs(current_pos_next.y - Y_MIN) <
    if abs(current_pos_next.y -</pre>
                    Final Answer:
2082
                                                                   swer\nprint(ricochets)</code>
2083
                                                                 Code Interpreter: output: "7\n"
                                                                                                                        • 1547
14661
2084
2085
2086
                                                                 Final Answer: The final answer is $\\boxed{7}$
2089
2090
2091
                                                                      Comparative Analysis
2092
                    Comparing TVP (T=1) and TVP (T=3), we can see that when only one iteration of the code is used, the model fails to find a solution
                    within a certain number of steps due to errors in its modeling of the overall layout, resulting in a return value of -1. However,
2093
                    with multiple iterations, <u>although the second iteration encountered a path-related error, the first and third iterations executed</u>
                   correctly, and the model successfully used the code to generate a simulated path diagram, which can assist the model in making
2094
2095
2096
2097
```

Figure 50: Case study of Ricochet Ball.