Ascending the Infinite Ladder: Benchmarking Spatial Deformation Reasoning in Vision-Language Models

Anonymous ACL submission

Abstract

Humans naturally possess the spatial reasoning ability to form and manipulate images and structures of objects in space. There is an increasing effort to endow Vision-Language Models (VLMs) with similar spatial reasoning capabilities. However, it remains unclear whether these models truly understand and manipulate spatial objects or not. To address this question, we propose a new evaluation framework aimed at assessing the performance of VLMs in spatial deformation reasoning tasks. Specifically, we construct a benchmark for spatial deformation reasoning from 2D to 3D. We explore whether the model can effectively perform spatial deformation reasoning from two directions: forward reasoning (given the operations, find the final state) and reverse reasoning (given the final state, determine the operations). We adopt a ladder competition format, using the number of deformation steps as the level classification criterion, with the goal of exploring the boundaries of the model's deformation reasoning capabilities. Interestingly, the benchmarking results reveal that almost no model demonstrates plausible spatial deformation reasoning abilities. Furthermore, even after applying targeted training and mainstream reasoning enhancement methods, the models are still unable to perform well on 3D spatial deformation reasoning. Please visit our anonymous project page for more details: Anonymous Page.

1 Introduction

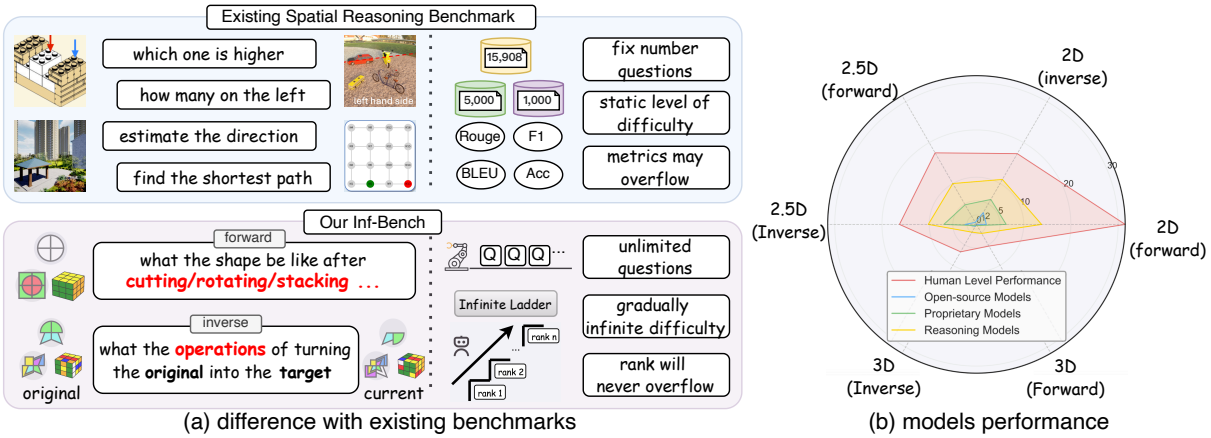
Imagine molding clay into a target object via sequential actions while tracking shape and topology. Each action requires predicting deformation and planning for a smooth, consistent result. This involves spatial deformation (Gain and Bechmann, 2008), a common task in daily life, such as modeling with clay or crafting sculptures. While current Vision-Language Models (VLMs) excel in language understanding and image recognition (Hou et al., 2024; Bordes et al., 2024; Ghosh et al., 2024), the question remains: *can they effectively handle complex spatial deformation tasks?*

Existing benchmarks explore aspects of three-dimensional spatial reasoning and visual-language reasoning in dynamic environments, they predominantly focus on static or dynamic scenes and spatial relationships between objects (Mayer et al., 2025; Tang et al., 2025a; Wang et al., 2025; Xu et al., 2025; Yang et al., 2024; Zhan et al., 2025; Tang et al., 2025b), as shown in Figure 1 . However, no benchmark executes in-depth evaluations of spatial deformation, specifically how models handle dynamic transformation of objects' shapes in space.

To fill this gap and probe model deformation limits, we introduce Inf-Bench, using Shapez (Springer, 2020) and Rubiks Cube to test VLM deformation reasoning. These games feature simple, prior-free rules focused on object manipulation, avoiding pattern or knowledge reliance. We test forward reasoning (initialtarget stepwise transforms) and inverse reasoning (inferring steps from targetinitial). We further build an automated data engine that generates unbounded, diverse, leakage-free evaluation tasks with no data leakage.

We propose an "Infinite Ladder Competition," where the number of required deformation steps defines task difficulty. All models start at one-step tasks and advance to more complex ones as they succeed. Unlike traditional benchmarks (shown in Figure 1) with fixed tasks and metrics, which become obsolete once models achieve perfect scores (Mayer et al., 2025; Xu et al., 2025; Oliveira et al., 2025), our system allows unlimited scalability. Model performance is reflected by the highest level completed, enabling continuous, dynamic evaluation of reasoning capabilities within a flexible and extensible framework.

We evaluate several mainstream VLMs and find that human performance outperforms all models across tasks, particularly in 3D reasoning, where most models, even the powerful model OpenAI



(a) difference with existing benchmarks (b) models performance

Figure 1: Existing spatial reasoning benchmarks (Tang et al., 2025a; Zhan et al., 2025; Wang et al., 2025; Tang et al., 2025b) for Vision-Language Models (VLMs) focus on tasks like 'which is higher' or 'find the shortest path,' with a static level of difficulty, which may easily get outdated with the fast evolving of VLMs. Our Inf-Bench introduces tasks requiring forward and reverse **spatial deformation reasoning** (e.g., shape changes after cutting/rotating/stacking) across 2D-3D. Our **ladder competition** offers evolving tasks, ensuring continuous capability exploration without performance saturation. The results show that even the best reasoning models still lag **significantly behind humans** in spatial deformation reasoning.

o3 (OpenAI, 2024b), fail to complete even basic reasoning tasks. Further analysis reveal that the models primarily rely on two strategies: "direct pairing" and "step-by-step reasoning execution". The former involves making intuitive choices based on patterns and regularities, while the latter first encodes the figure and then analyzes deformations in the encoding before mapping it back to the figure. However, the results indicate that both strategies have significant performance bottlenecks, highlighting the models' difficulties in managing high-dimensional coupling relationships.

Additionally, we fine-tune models on datasets of different scales. We find that for low-dimensional tasks, models can learn forward iterative transforms and develop generalizable reasoning. In contrast, 3D forward tasks still show limited reasoning depth, indicating a core bottleneck. We also test common VLM enhancement methods such as Chain-of-Thought (COT) (Wei et al., 2022) and ReAct (Yao et al., 2023b), but observe little gains, and occasional hurt. Overall, current VLMs still lack stable, generalizable spatial deformation reasoning.

The main contributions of this paper can be summarized as follows: (i) We introduce the first effective benchmark for comprehensively exploring the spatial deformation reasoning ability of VLMs. (ii) We propose an infinitely scalable benchmarking paradigm based on the ladder competition system. (iii) We provide a thoroughly evaluation and identifying the VLMs' shortcomings in spatial deformation reasoning, providing a clear direction for future model development.

2 Related Work

VLMs Spatial Reasoning Benchmark. Recent studies highlight reasoning as key for spatial intelligence. Although multimodal learning has advanced, VLMs still perform poorly on complex spatial tasks. iVISPAR (Mayer et al., 2025) shows notable gaps in planning and spatial awareness. SP-Gym (Oliveira et al., 2025) reveals weak generalization across visuals. LEGO-Puzzles (Tang et al., 2025a) finds VLMs solve only 50% of spatial questions, far below 90% human accuracy. Interactive evaluation is supported by ThreeDWorld (Gan et al., 2021) and ThreeDWorld (Gan et al., 2021). VisuLogic (Xu et al., 2025) and LLM-SRBench (Shojaee et al., 2025) confirm limitations beyond memorization, while Sparkle (Tang et al., 2025b) and VSI-Bench (Yang et al., 2024) show gains from targeted supervision and cognitive map generation. Our work fills the gap in 2D-to-3D spatial deformation reasoning with structured mechanics and a ladder competition that identifies limitations.

VLMs Reasoning Improvement. The reasoning capabilities in LLMs stem from extensive training datasets and improved methodologies (OpenAI, 2024a). VLMs have built on this by demonstrating visual reasoning through learning the relationships between visual inputs and text (Cheng et al., 2024b; Li et al., 2024). Motivated by LLM advancements, several studies have aimed to enhance VLM reasoning. Approaches include CoT (Wei et al., 2022; Zhang et al., 2024), ToT (Yao et al., 2023a), and Monte Carlo methods (Trinh et al.,

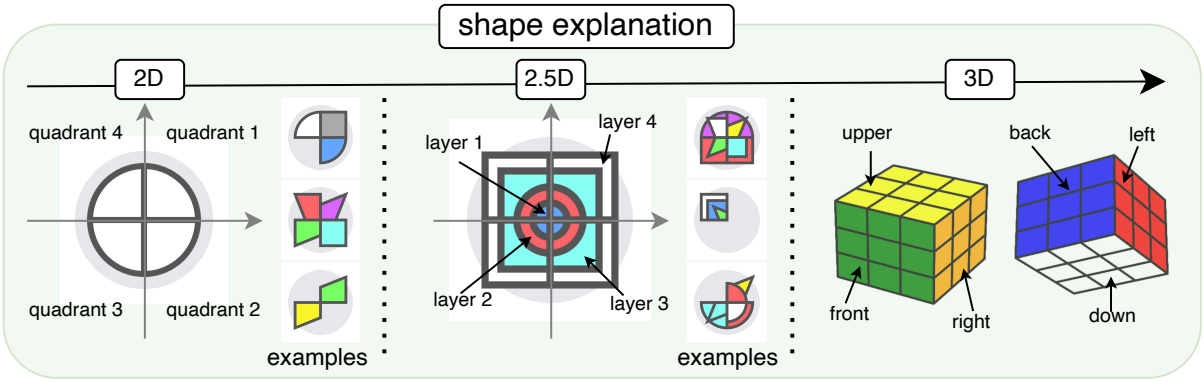


Figure 2: **Shape Explanation.** 2D shapes are presented on a flat plane, while 2.5D introduces additional (up to four) layers of dimension. 3D shapes, based on a Rubik's Cube, introduce depth and spatial orientation.

2024; Wan et al., 2024; Wu et al., 2025), along with SFT datasets to improve reasoning performance (Ye et al., 2025; Muennighoff et al., 2025). Despite progress, spatial understanding, such as geometry and relationships, remains a challenge for VLMs (Liu et al., 2024; Ramakrishnan et al., 2025). This has sparked research to improve spatial abilities through pre-training (Xia et al., 2025), 3D data construction (Cai et al., 2024; Chen et al., 2024), reasoning frameworks (Cheng et al., 2024a), multimodal alignment (Ray et al., 2024), and Chain-of-Thought methods (Liu et al., 2025; Wu et al., 2024). Despite these efforts, spatial imagination and deformation remain underexplored.

3 Spatial Deformation Reasoning

In this study, **spatial deformation reasoning** refers to the models ability to understand, predict, and execute complex shape deformations without prior knowledge, learning via observation or manipulation. Unlike visual spatial intelligence in VSI-Bench (Yang et al., 2024), which focuses on localization and spatial perception, our spatial deformation emphasizes dynamic, multi-step shape transformations that alter object states. We define its core capabilities as:

Spatial Recognition. The ability to comprehend the initial shape of an object and accurately identify the areas that require deformation.

Abstraction of Operational Law Principles. The ability to grasp the principles behind deformation operations and abstract them into deformation laws, crucial for accurate reasoning.

Stable Reasoning Execution. The ability to stepwise execute inferred reasoning rules with stability and produce the final state.

4 Inf-Bench

4.1 Benchmark Overview

We introduce Inf-Bench, a dataset for evaluating VLMs on spatial deformation reasoning. It spans 2D to 3D tasks across three spatial levels:

2D Tasks: This task, inspired by single-plane deformation in Shapez, evaluates the model's ability to recognize objects and perform shape transformations in a two-dimensional plane. Shapes are single-layer figures divided into four quadrants, each containing one of four predefined patterns or an "empty" state (see Figure 2). The task includes six deformation operations, involving cutting, rotating, or coloring to transform the overall shape.

2.5D Tasks: Building on the 2D task, we add vertical stacking (4 layers) for multi-layer deformation (see Figure 2). Models must handle intra-layer transforms and cross-layer alignment/composition reasoning. We define 7 deformations to test multi-layer and spatial reasoning.

3D Tasks: Based on the Rubiks Cube, this task tests 3D spatial manipulation and reasoning (see Figure 2). The cube contains 27 units / 54 visible faces, whose positions and orientations change via rotation. We include all 54 basic rotations to assess 3D spatial reasoning, action planning, and hierarchical reasoning. Full shape and operation details are in Appendix E.

Each spatial level contains Forward Reasoning and Inverse Reasoning tasks. Forward Reasoning deforms an object from the initial state to the target state, testing sequence understanding and spatial operation execution. Inverse Reasoning infers the deformation path starting from the target state, evaluating reverse-engineering ability. Task examples are shown in the Figure 3 and all the prompts are presented in the Appendix B.2.

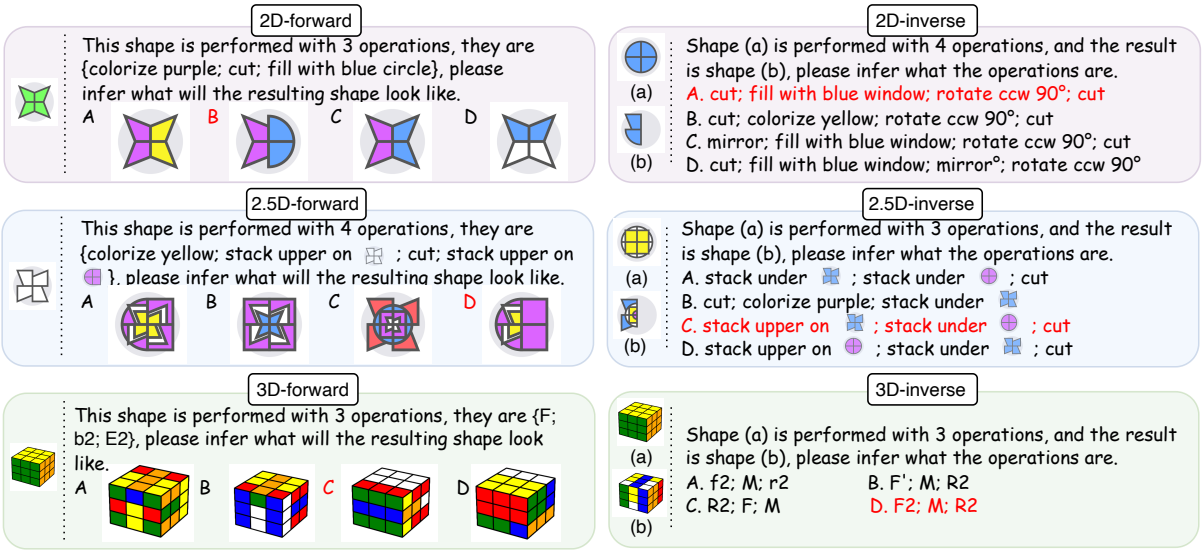


Figure 3: **Inf-Bench Task Examples.** Tasks are presented from 2D to 3D, with forward tasks (left) and reverse tasks (right). Note: All problems have been simplified to enhance clarity and conciseness.

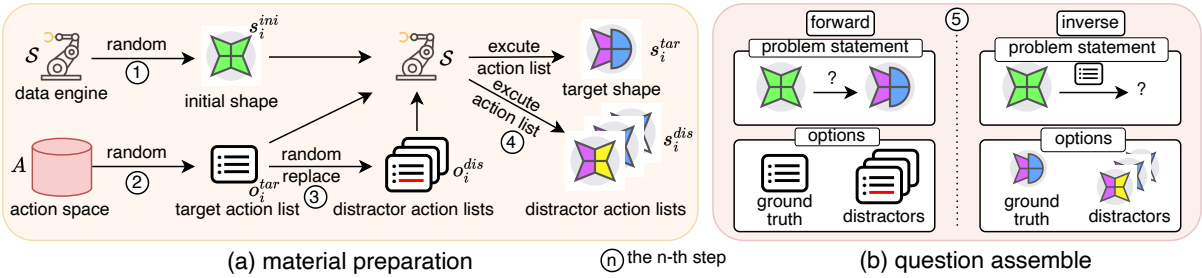


Figure 4: **Data Generation Pipeline.** The process consists of five steps: First, a shape is generated, and then a target deformation operation list is created. Next, an interference list is generated by modifying the target list. Target and interference shapes are then produced, and finally, materials are assembled into both forward and reverse tasks.

4.2 Dataset Construction

To ensure data quality and accuracy, we designed automated **shape generation engine**: $\mathcal{S}$, which performs two main functions: 1) Randomly generating an initial shape, and 2) Applying a sequence of deformations (action list) to generate the corresponding target shape. The shape generation engine is rule-based, with generation and operation execution implemented as fixed Python functions, detailed in the Appendix C.3. The deformation execution is deterministic, ensuring that the same initial shape and action list always produce a unique and consistent final shape. This guarantees reproducibility, eliminates manual annotation errors, and enhances the dataset's reliability for evaluation.

The data generation pipeline consists of two main processes: **material preparation** and **question assembly**, which are divided into five steps shown in Figure 4:

Step 1: The shape generation engine $\mathcal{S}$ randomly selects an initial shape s_i^{ini}, given by $s_i^{\text{ini}} = \mathcal{S}(\text{Init})$.

Step 2: n actions are randomly chosen from the action space A to create the action list $o_i^{\text{tar}} = \text{RandList}(n, A)$.

Step 3: Distractor action lists are generated by replacing selected operations in o_i^{tar} with new ones from A, forming k distractor lists $o_{i,j}^{\text{dis}} = \text{RandReplace}(o_i^{\text{tar}}, r, A)$, i.e., $o_i^{\text{dis}} = \{o_{i,j}^{\text{dis}}\}_{j=1}^{k}$.

Step 4: The action lists o_i^{tar} and $o_{i,j}^{\text{dis}}$ are applied to the initial shape s_i^{ini} to generate the target shape $s_i^{\text{tar}} = \mathcal{S}(s_i^{\text{ini}}, o_i^{\text{tar}})$ and distractor shapes $s_{i,j}^{\text{dis}} = \mathcal{S}(s_i^{\text{ini}}, o_{i,j}^{\text{dis}})$, i.e., $s_i^{\text{dis}} = \{s_{i,j}^{\text{dis}}\}_{j=1}^{k}$.

Step 5: Multiple-choice questions are generated as follows. The forward question q_i^{for} consists of an option set $op_i^{\text{for}} = \{s_i^{\text{tar}}, s_i^{\text{dis}}\}$, and its composition is given by $q_i^{\text{for}} = \left(\{s_i^{\text{ini}}, o_i^{\text{tar}}, op_i^{\text{for}}\}, \text{gt}_i = s_i^{\text{tar}}\right)$. The inverse question q_i^{inv} has the option set $op_i^{\text{inv}} = \{o_i^{\text{tar}}, o_i^{\text{dis}}\}$, with the composition $q_i^{\text{inv}} = \left(\{s_i^{\text{ini}}, s_i^{\text{tar}}, op_i^{\text{inv}}\}, \text{gt}_i = o_i^{\text{tar}}\right)$, where gt_i represents the ground truth for each question.

5 Evaluation on Inf-Bench

5.1 Ladder Competition Framework

To explore the boundaries of spatial deformation reasoning abilities and systematically evaluate models' performance, we employ the ladder competition framework. The goal is progressively increasing difficulty, providing a comprehensive test of the model's adaptability in deformation tasks. In the ladder competition, models begin at the lowest difficulty level, denoted as $R = 1$ with only one step deformation. Each level consists of five questions. If a model successfully answers at least three questions, it advances to a higher difficulty level; otherwise, it is downgraded to the same level. If it fails at the same level twice, the competition ends:

$$R = \begin{cases} R-1 \text{ and } f_R = f_R + 1, & \text{if } c < 3 \\ R+1, & \text{if } c \geq 3 \\ R, & \text{if } f_R = 2 \text{ or } R = 0 \end{cases}$$

where f_R represents failure times of a particular R, and c denotes the number of correctly answered questions at this level. This mechanism thoroughly tests the model's reasoning abilities under increasing cognitive load. The rules for advancing, downgrading, and the number of allowed failures at each level ensure the rigor of the evaluation and the stability of the model's reasoning capabilities. **The final value of R, representing reasoning depth, is used as the Inf-Bench metric.**

5.2 Evaluation Setting

We thoroughly evaluate 17 image-supported VLMs, representing a diverse range of model series, parameter scales, training strategies, and advanced reasoning capabilities. We also recruit 100 volunteers to conduct the testing. For open-source models, we evaluate those from the Qwen3-VL series(Bai et al., 2025), Llama 4 series (Meta, 2025), and Gemma 3 series(Team et al., 2025). The proprietary models under consideration include GPT-5 series (OpenAI, 2025a) and Gemini 3 Flash (Google DeepMind, 2025a). Additionally, the reasoning models incorporated comprise O3 (OpenAI, 2024b), GPT-5.2 pro(OpenAI, 2025b), Gemini 3 Pro(Google DeepMind, 2025b), Grok 4.1(xAI, 2025), and Claude Opus 4.5(Anthropic, 2025).

All evaluations are conducted in a zero-shot setting. To ensure robustness and impartiality, each model is evaluated on the same task set across all ranks, with 10 independent runs per model for statistical validity. Greedy decoding was used for all models to enhance reproducibility. The results are shown in Table 1.

Table 1: **Evaluation on Inf-Bench.** The data shows the average reasoning depth R achieved by the models after 10 ladder competitions. Light pink indicates best performance by open-source models, light green by proprietary models, and light blue by reasoning models.

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
Human	31.5	17.3	17.5	16.3	6.7	5.3
Open-source Models						
Qwen3-VL-2B	1.7	1.2	0.6	1.0	0.2	0.0
Qwen3-VL-8B	4.2	6.4	1.3	5.6	0.7	0.0
Qwen3-VL-32B	6.5	9.6	3.4	8.6	1.1	0.0
Llama 4-scout	0.4	0.6	0.6	0.6	0.8	0.0
Llama 4-maverick	3.6	5.2	3.0	2.9	1.0	0.0
Gemma 3-4B	1.6	1.0	0.5	0.9	0.2	0.0
Gemma 3-12B	3.7	7.3	1.1	5.3	0.6	0.0
Gemma 3-27B	6.1	8.5	3.2	8.3	1.1	0.0
Proprietary Models						
GPT-5 nano	2.0	1.4	0.7	1.2	0.2	0.0
GPT-5 mini	6.6	8.4	4.4	6.3	0.7	0.0
GPT-5	9.9	10.5	7.6	9.3	1.2	0.0
Gemini 3 Flash	14.5	21.1	14.2	19.4	1.4	1.7
Reasoning Models						
O3	27.0	15.3	13.6	10.4	3.2	3.9
GPT-5.2 pro	28.8	16.6	16.4	13.1	3.9	4.2
Gemini 3 Pro	17.0	17.1	16.1	15.8	2.8	4.3
Grok 4.1	13.8	13.9	10.4	12.5	2.2	3.0
Claude Opus 4.5	16.4	14.1	11.0	12.2	2.2	3.9

5.3 Main Results

Human-Level Performance. As expected, the final scores of human evaluators all exceed those of the best-performing models in 2D, 2.5D, or 3D tasks. They generally perform better in forward reasoning tasks than reverse reasoning tasks.

Model Performance Overview. It is evident that models generally show a competitive performance in 2D and 2.5D tasks. It is also found that the performance of the reasoning model and proprietary models is significantly better than that of the open-source models. However, in tasks that require actual 3D spatial transformation reasoning, both open-source and proprietary models struggle to perform even a single step of reasoning. This highlights a significant limitation in their spatial deformation reasoning capabilities. Only powerful reasoning models, such as GPT-5.2 pro, demonstrate some initial 3D deformation abilities. However, most of their performance still lags significantly behind human capabilities, indicating substantial room for improvement.

Figure 5: **Case Study.** (a) illustrates the direct pairing, where valuable information is extracted from the prompt and used to pair options directly. (b) shows the example of step-by-step reasoning execution, where the model first encodes the statement, abstracts actions into operations, and then executes sequentially. (c) compares the proportions of the two methods, highlighting that more complex tasks favor the step-by-step reasoning approach, which also tends to have higher execution accuracy. (d) displays the distribution of main error types made by models.

6 Discussion

6.1 How Do VLMs Perform on Spatial Deformation Reasoning?

To better understand model performance, we analyze the internal reasoning processes to reveal the mechanisms. We select representative models from three categories: Qwen3-VL-32B, GPT-5, and Gemini 3 Pro, and collect their reasoning processes. Each collect 100 responses per task, totaling 900 responses. Through analysis, we find that the model primarily utilizes two approaches to response. One is **direct pairing** (see Figure 5 (a)), where the model extracts valuable information directly from the operations and pairs them with the options. The second is **step-by-step reasoning** (see Figure 5 (b)), where the model encodes the input shapes and abstracts various operations into a set of standardized encoding rules. The model then executes the operations based on these rules to derive the final answer. From Figure 5 (c), it is evident that stronger models derive more conclusions through step-by-step reasoning. As difficulty increases, the proportion of direct pairings also rises, reflecting the model's tendency to infer from valuable information in complex tasks.

We analyze 476 incorrect responses and categorize the errors into four types, as shown in Figure 5 (d). Full error analysis and examples are in Appendix B.1.

Pairing Errors occur when models incorrectly pair based on wrong information from the problem statement, accounting for 31.72% of errors.

Encoding Errors arise when models confuse shape encoding, especially multi-layered shapes, leading to incorrect answers, representing 15.76%.

Abstracting Errors happen when models misinterpret operations or their effects, accounting for 41.81%, particularly in cube tasks where coupled shape states complicate abstraction. Lastly,

Execution Errors occur during multi-step reasoning, with one step incorrectly executed, making up 10.71% of errors. These are more common in non-reasoning models.

6.2 How Do VLMs Perform on Spatial Deformation Reasoning with Only Encoded Input?

As noted earlier, models primarily rely on shape encoding for spatial deformation reasoning. To minimize perceptual encoding errors, we pre-encode all images uniformly, allowing models to process pure

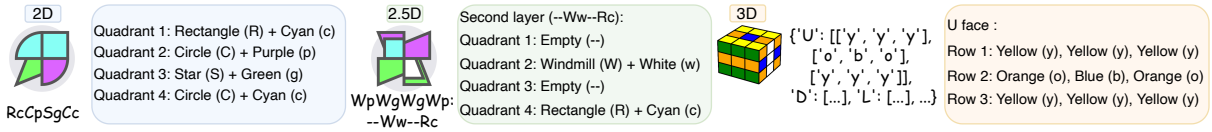


Figure 6: **Encoded Shape Example.**

text input for reasoning. The encoding methods are detailed in the Appendix C.1, examples in Figure 6 and results in Table 2.

Table 2: **Evaluation on Inf-Bench (only Encoded Input).** The data shows the average reasoning depth R achieved by the models after 10 ladder competitions. Light pink indicates the best performance by open-source models, light green by proprietary models, and light blue by reasoning models.

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
Human	7.2	5.7	6.4	4.1	1.9	1.3
Open-source Models						
Qwen3-VL-2B	2.7	0.4	2.8	3.2	0.0	0.0
Qwen3-VL-8B	24.1	4.1	8.7	4.9	0.0	0.0
Qwen3-VL-32B	32.5	8.3	15.2	11.1	0.2	0.0
Llama 4-scout	0.1	0.0	1.0	0.9	0.0	0.0
Llama 4-maverick	4.1	3.0	3.2	2.2	1.4	0.2
Gemma 3-4B	2.6	0.3	2.2	2.7	0.0	0.0
Gemma 3-12B	21.4	3.6	7.5	4.6	0.0	0.0
Gemma 3-27B	30.6	7.3	14.5	10.7	0.1	0.0
Proprietary Models						
GPT-5 nano	7.4	2.0	2.9	2.2	0.4	0.0
GPT-5 mini	41.2	13.3	18.4	13.7	0.6	0.0
GPT-5	75.9	13.2	20.9	14.3	1.9	0.0
Gemini 3 Flash	177.4	87.6	22.1	33.6	3.1	3.2
Reasoning Models						
O3	671.4	61.6	61.1	17.6	4.4	4.2
GPT-5.2 pro	711.7	69.0	65.4	19.4	5.0	4.4
Gemini 3 Pro	590.7	134.6	52.1	18.1	5.6	6.8
Grok 4.1	452.2	55.2	29.0	13.6	4.4	3.1
Claude Opus 4.5	483.8	78.4	30.4	15.0	5.0	3.4

The key findings are that models exhibit significant improvement in 2D and 2.5D tasks, however, they still face challenges in 3D tasks, specifically:

Performance in 2D and 2.5D tasks improves significantly, suggesting that models effectively abstract operational patterns for tasks involving operations like cutting and rotating, which typically lack complex coupled states. Reasoning models show even greater improvements, with GPT-5.2 pro achieving forward reasoning depth even over 711.7, and Gemini 3 Pro reaching an inverse reasoning depth of 134.6, demonstrating stable performance.

However, in 3D tasks, most models still struggle with operations on 3D shapes, where each operation involves interactions between coupled shape states. For instance, Rubik's Cube operations require considering interactions across multiple faces. These multidimensional dependencies increase reasoning complexity. Only high-performance models like Gemini 3 Pro offer partial solutions, with reasoning depth R of 6.8, indicating room for improvement.

Interestingly, human performance on text-based tasks is lower than visual input, often lagging behind the models, highlighting the difference in thinking processes between humans and models.

6.3 Can Supervised Fine-tuning Effectively Enhance Spatial Deformation Reasoning Ability?

Table 3: **Evaluation on Inf-Bench After SFT.** The data shows the average R of 10 ladder challenges, after SFT. $S_{\max}$ represents the highest difficulty level in the fine-tuning data. "For" refers to forward reasoning training, while "Inv" refers to inverse reasoning.

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
Qwen3-VL-8B						
Vanilia	4.2	6.4	1.3	5.6	0.7	0.0
For (max = 1)	6.9	10.5	6.4	16.2	1.5	0.0
Inv (max = 1)	7.2	10.9	4.6	15.4	1.3	1.1
For (max = 5)	15.4	11.7	14.2	18.1	6.6	3.9
Inv (max = 5)	9.9	84.4	8.3	47.0	1.2	71.9
For (max = 10)	45.0	17.2	15.7	26.1	6.6	5.7
Inv (max = 10)	11.8	169.5	10.3	159.1	2.2	139.3
Qwen3-VL-32B						
Vanilia	6.5	9.6	3.4	8.6	1.1	0.0
For (max = 1)	7.6	10.2	11.1	16.5	1.9	0.0
Inv (max = 1)	8.4	11.9	8.4	17.9	1.7	1.5
For (max = 5)	37.3	17.6	15.8	28.7	6.3	6.0
Inv (max = 5)	15.7	367.0	11.1	265.9	2.7	85.8
For (max = 10)	51.1	18.5	19.1	30.6	7.6	6.9
Inv (max = 10)	15.4	436.5	13.8	357.4	3.0	273.2

The above discussion shows existing models still lag in spatial deformation reasoning, raising a key question: Can classical training paradigms solve this? To explore, we run SFT on Qwen3-VL-8B/32B-Instruct, two open-source models differ in scale that perform poorly in initial tests. We use the $S_{\max}$ to classify the datasets, where $S_{\max}$ represents the most difficult data covered in each dataset. Each group contains 20,000 samples, with equal

data across steps from 1 to $S_{\max}$. Training details are in the Appendix C.2.

Key findings show that SFT improves reasoning depth and generalization, but 3D forward task performance remains constrained, exposing the limitations of traditional training for high-dimensional deformation reasoning, specifically:

Inverse task achieves deeper inference, forward task limited. In the forward task, reasoning depth improves with increasing $S_{\max}$ but remains limited in 3D tasks. In contrast, the inverse task is more robust, reducing error accumulation by gradually verifying the target state, which enhances solution depth, especially as $S_{\max}$ increases.

SFT enhances 2D/2.5D depth, 3D forward task limited. SFT significantly boosts reasoning depth in 2D and 2.5D tasks, enabling multi-step reasoning beyond training examples, suggesting the learning of a reusable deformation operator. However, in 3D tasks, even with high-difficulty data (e.g., $S_{\max} = 10$), the forward task struggles to exceed seven steps, highlighting the difficulty of capturing 3D deformation sequences and error accumulation with only supervised examples.

Forward task shows stronger generalization than inverse task. Forward-task-trained models generalize better to the backward task than inverse-task-trained ones, suggesting that stepwise forward reasoning helps solve the inverse task, whereas the absence of forward reasoning limits generalization in inverse-task of models.

Table 4: **Evaluation on Inf-Bench After Reasoning Enhancement.** The data shows the mean highest rank difference after 10 ladder challenges with the reasoning enhancement method, compared to the baseline performance (i.e., vanilla results).

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
GPT-5						
Vanilia	9.9	10.5	7.6	9.3	1.2	0.0
+COT	-0.2	-0.4	-0.6	+0.1	-0.2	0.0
+Few-shot	+0.6	+0.3	+0.5	+0.1	+0.0	0.0
+Self-reflection	-5.3	-1.9	-1.0	+0.3	-0.8	0.0
+ReAct	-3.5	-2.8	-6.0	-0.5	-0.9	0.0
+Tools	-0.4	-0.3	-1.6	+1.8	-1.1	+0.1
Llama 4-scout						
Vanilia	0.4	0.6	0.6	0.6	0.8	0.0
+COT	+0.3	-0.0	-0.1	-0.1	-0.3	0.0
+Few-shot	+0.5	+0.1	-0.0	-0.0	-0.1	0.0
+Self-reflection	+0.1	-0.1	+0.0	+0.1	+0.2	0.0
+ReAct	+0.0	-0.1	-0.2	-0.0	-0.1	0.0
+Tools	+0.1	+0.1	-0.0	-0.0	-0.1	0.0

6.4 Can Reasoning Enhancement Methods Improve Spatial Deformation Reasoning?

We explore some mainstream reasoning enhancement methods to assess whether they could effectively improve spatial deformation reasoning, including Chain-of-Thought (COT) (Wei et al., 2022), Few-shot learning, Self-reflection, Tool Invocation, and Reasoning and Action (ReAct) (Yao et al., 2023b). Implementation details and complete results can be found in the Appendix C.4. The results (see Table 4) indicate that these methods led to slight improvements in specific tasks and dimensions, but overall, **they do not significantly enhance the ability of spatial deformation reasoning.** This is especially evident in 3D tasks and backward tasks, where the effects are more limited.

What's more, we conducted ablations on different prompt formulations to study reasoning impact. Results show that prompt changes cannot remove inherent spatial reasoning limits, exposing a core lack of deep spatial inference. Due to space constraints, the full ablation analysis, along with a detailed description of the prompt types used in the experiments, will be provided in Appendix D.

6.5 Scalability to Other Domains and Real-World Deployment?

Our design principle is to decompose complex spatial reasoning into controllable, measurable core capabilities: compositional transformation, inverse planning, and long-range causal dependency, treated as atomic operations for higher-order systems like embodied agents and autonomous driving. For instance, an embodied agent organizing a room must reason about how objects can be rotated and translated to fit into a drawer. Such high-level tasks ultimately reduce to the same atomic operations tested in our benchmark.

7 Conclusion

We introduce Inf-Bench to evaluate VLMs on spatial deformation reasoning. Results reveal major weaknesses in complex 3D and multi-step reasoning, persisting even with reasoning enhancement. Humans outperform all models, showing a gap from human spatial imagination. Our findings suggest that improvements are needed in error reduction, multi-step abstract reasoning, and 3D reasoning capabilities. This study contributes to the development of robust spatial reasoning frameworks and identifies key directions for improving VLM.

8 Limitations

Although we explore various spatial transformation tasks from 2D to 3D based on Shapez and Rubik's Cube, real-world spatial deformations often involve more complex scenarios including continuous transformations, non-rigid objects, and topological changes. Future research should extend to a broader range of deformation types. Our supervised fine-tuning approach, while yielding improvements on certain tasks, fails to address fundamental limitations in spatial reasoningparticularly the lack of mechanisms for maintaining global consistency and correcting accumulated errors in long sequence dependencies. This suggests the need for innovative architectural designs incorporating physics-inspired attention mechanisms or symbolic reasoning tools.

Additionally, our evaluation primarily focused on model performance at fixed step numbers rather than exploring transfer capabilities between tasks of varying difficulty. Future work should systematically investigate knowledge transfer and generalization abilities from low-dimensional to high-dimensional tasks.

While our experiments demonstrate fundamental limitations in current models' spatial reasoning capabilities, we have not exhaustively explored all possible enhancement techniques. Combining multimodal pretraining with neural-symbolic methods, embodied learning, or external memory mechanisms might provide breakthroughs in this domain.

References

Anthropic. 2025. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Florian Bordes, Richard Yuanzhe Pang, Anurag Ajay, Alexander C Li, Adrien Bardes, Suzanne Petryk, Oscar Mañas, Zhiqiu Lin, Anas Mahmoud, Bargav Jayaraman, and 1 others. 2024. An introduction to vision-language modeling. *arXiv preprint arXiv:2405.17247*.

Wenxiao Cai, Iaroslav Ponomarenko, Jianhao Yuan, Xiaoqi Li, Wankou Yang, Hao Dong, and Bo Zhao. 2024. Spatialbot: Precise spatial understanding with vision language models. *arXiv preprint arXiv:2406.13642*.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. 2024. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465.

An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. 2024a. Spatialrgpt: Grounded spatial reasoning in vision language models. *arXiv preprint arXiv:2406.01584*.

Zesen Cheng, Sicong Leng, Hang Zhang, Yifei Xin, Xin Li, Guanzheng Chen, Yongxin Zhu, Wenqi Zhang, Ziyang Luo, Deli Zhao, and 1 others. 2024b. Video-llama 2: Advancing spatial-temporal modeling and audio understanding in video-llms. *arXiv preprint arXiv:2406.07476*.

James Gain and Dominique Bechmann. 2008. A survey of spatial deformation from a user-centered perspective. *ACM Transactions on Graphics (TOG)*, 27(4):1–21.

Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Feigelis, Daniel M. Bear, Dan Gutfreund, David Cox, and 5 others. 2021. ThreeDWorld: a platform for interactive multi-modal physical simulation. *arXiv preprint*. ArXiv:2007.04954 [cs].

Akash Ghosh, Arkadeep Acharya, Sriparna Saha, Vinija Jain, and Aman Chadha. 2024. Exploring the frontier of vision-language models: A survey of current methodologies and future directions. *arXiv preprint arXiv:2404.07214*.

Google DeepMind. 2025a. Gemini 3 flash. https://deepmind.google/models/gemini/flash/.

Google DeepMind. 2025b. Gemini 3 pro. https://deepmind.google/models/gemini/pro/.

Yifan Hou, Buse Giledereli, Yilei Tu, and Mrinmaya Sachan. 2024. Do vision-language models really understand visual language? *arXiv preprint arXiv:2410.00193*.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and 1 others. 2024. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Yike Yuan, Wangbo Zhao, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. 2024. MMBench: Is your multi-modal model an all-around player?

Yuecheng Liu, Dafeng Chi, Shiguang Wu, Zhanguang Zhang, Yaochen Hu, Lingfeng Zhang, Yingxue Zhang, Shuang Wu, Tongtong Cao, Guowei Huang, and 1 others. 2025. Spatialcot: Advancing spatial reasoning through coordinate alignment and chain-of-thought for embodied task planning. *arXiv preprint arXiv:2501.10074*.

Julius Mayer, Mohamad Ballout, Serwan Jassim, Farbod Nosrat Nezami, and Elia Bruni. 2025. ivispar–an interactive visual-spatial reasoning benchmark for vlms. *arXiv preprint arXiv:2502.03214*.

AI Meta. 2025. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, april 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Bryan L. M. de Oliveira, Murilo L. da Luz, Bruno Brandão, Luana G. B. Martins, Telma W. de L. Soares, and Luckeciano C. Melo. 2025. Sliding puzzles gym: a scalable benchmark for state representation in visual reinforcement learning. *arXiv preprint*. ArXiv:2410.14038 [cs].

OpenAI. 2024a. Openai o1 system card. OpenAI Website. Https://cdn.openai.com/o1-system-card-20241205.pdf, Accessed: 2025-05-14.

OpenAI. 2024b. Openai o3 and o4-mini system card. OpenAI Website. Https://openai.com/index/o3-o4-mini-system-card/, Accessed: 2025-05-14.

OpenAI. 2025a. Introducing gpt-5. https://openai.com/zh-Hans-CN/index/introducing-gpt-5/.

OpenAI. 2025b. Introducing gpt-5.2. https://openai.com/index/introducing-gpt-5-2/.

Santhosh Kumar Ramakrishnan, Erik Wijmans, Philipp Kraehenbuehl, and Vladlen Koltun. 2025. Does spatial cognition emerge in frontier models? In *The Thirteenth International Conference on Learning Representations*.

Arijit Ray, Jiafei Duan, Reuben Tan, Dina Bashkirova, Rose Hendrix, Kiana Ehsani, Aniruddha Kembhavi, Bryan A Plummer, Ranjay Krishna, Kuo-Hao Zeng, and 1 others. 2024. Sat: Spatial aptitude training for multimodal language models. *arXiv preprint arXiv:2412.07755*.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D. Doan, and Chandan K. Reddy. 2025. LLM-SRBench: a new benchmark for scientific equation discovery with large language models. *arXiv preprint*. ArXiv:2504.10415 [cs].

Tobias Springer. 2020. Shapez. https://store.steampowered.com/app/1318690/shapez/. PC, Released June 7, 2020.

Kexian Tang, Junyao Gao, Yanhong Zeng, Haodong Duan, Yanan Sun, Zhening Xing, Wenran Liu, Kaifeng Lyu, and Kai Chen. 2025a. Lego-puzzles: How good are mllms at multi-step spatial reasoning? *arXiv preprint arXiv:2503.19990*.

Yihong Tang, Ao Qu, Zhaokai Wang, Dingyi Zhuang, Zhaofeng Wu, Wei Ma, Shenhao Wang, Yunhan Zheng, Zhan Zhao, and Jinhua Zhao. 2025b. Sparkle: mastering basic spatial capabilities in vision language models elicits generalization to spatial reasoning. *arXiv preprint*. ArXiv:2410.16162 [cs].

Gemma Team, 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, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. In *Forty-first International Conference on Machine Learning*.

Xingrui Wang, Wufei Ma, Tiezheng Zhang, Celso M de Melo, Jieneng Chen, and Alan Yuille. 2025. Pulsecheck457: A diagnostic benchmark for 6d spatial reasoning of large multimodal models. *arXiv e-prints*, pages arXiv–2502.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Ruihan Jin, Feihu Che, Zengqi Wen, and Jianhua Tao. 2025. Boosting multimodal reasoning with mcts-automated structured thinking. *arXiv preprint arXiv:2502.02339*.

Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. 2024. Mind's eye of llms: Visualization-of-thought elicits spatial reasoning in large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

xAI. 2025. Grok 4.1. https://x.ai/news/grok-4-1.

Renqiu Xia, Mingsheng Li, Hancheng Ye, Wenjie Wu, Hongbin Zhou, Jiakang Yuan, Tianshuo Peng, Xinyu Cai, Xiangchao Yan, Bin Wang, Conghui He, Botian Shi, Tao Chen, Junchi Yan, and Bo Zhang. 2025.

Geox: Geometric problem solving through unified formalized vision-language pre-training. In *The Thirteenth International Conference on Learning Representations*.

Weiye Xu, Jiahao Wang, Weiyun Wang, Zhe Chen, Wengang Zhou, Aijun Yang, Lewei Lu, Houqiang Li, Xiaohua Wang, Xizhou Zhu, and 1 others. 2025. Visulogic: A benchmark for evaluating visual reasoning in multi-modal large language models. *arXiv preprint arXiv:2504.15279*.

Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. 2024. Thinking in space: How multimodal large language models see, remember, and recall spaces. *arXiv preprint arXiv:2412.14171*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Weichen Zhan, Zile Zhou, Zhiheng Zheng, Chen Gao, Jinqiang Cui, Yong Li, Xinlei Chen, and Xiao-Ping Zhang. 2025. Open3dvqa: A benchmark for comprehensive spatial reasoning with multimodal large language model in open space. *arXiv preprint arXiv:2503.11094*.

Ruohong Zhang, Bowen Zhang, Yanghao Li, Haotian Zhang, Zhiqing Sun, Zhe Gan, Yinfei Yang, Ruoming Pang, and Yiming Yang. 2024. Improve vision language model chain-of-thought reasoning. *arXiv preprint arXiv:2410.16198*.

A Appendix Outline

In the appendix, we provide the following:

- The detailed error analysis and evaluation setup for the Inf-Bench experiments (Appendix B);
- Technical details on the construction of Inf-Bench, including our method for encoding graphics, training details, and the implementation specifics of the reasoning enhancement methods we employ (Appendix C);
- Prompts ablation study (Appendix D).
- A comprehensive introduction to the graphics, colors, and operations (Appendix E).

B Evaluation Details

B.1 Error Analysis Detail and Example

For each erroneous case, we classify its primary error into one of four main categories: Pairing Errors, Encoding Errors, Abstracting Errors, and Execution Errors. Suppose an incorrect prediction is attributed to multiple causes. In that case, it is proportionally assigned to each category based on the number of applicable error categories, with n representing the number of error categories. Examples of these four types of errors are presented in Figure 7.

Pairing Error: Models incorrectly pair based on the incorrect information extracted from the problem statement, leading to mistakes.

Encoding Error: Models confuse in encoding shapes, such as inconsistencies between the front and back, leading to incorrect answers. These errors are primarily concentrated in shapes with multiple layers, as the complexity of encoding increases.

Abstracting Error: Models misinterpret operations or incorrectly predict their effects. These errors are mainly observed in cube tasks, where each operation must consider the coupled states of the shapes, making it more difficult to abstract the resulting state changes.

Execution Error: Model make errors during multi-step reasoning, where one step of reasoning is incorrectly executed. This type of error is more common in non-reasoning models, as reasoning models have more mature thought chains and higher reasoning stability.

Question: The initial shape is  . Please predict what the shape will look like after performing {cutting, rotating 90° clockwise, stacking on  , and colorize the top layer purple}.

Incorrect Answer: The initial shape shows four red windmills, and the final operation mentions changing the top layer to purple. Therefore, options with purple windmills are most likely to be the correct answer.

Explanation: The four red windmills in the initial shape have actually already been moved to the bottom layer during the stacking process, and will not be dyed purple.

(a) Pairing Error

Question: The initial shape is  . Please predict what the shape will look like after performing {cutting, rotating 90° clockwise, stacking on  , and colorize the top layer purple}.

Incorrect Answer: The initial shape is encoded as (Top layer: RW, RW, RW).

Explanation: There are four quadrants in total, but only three quadrants are represented, which causes issues in the subsequent operations. Therefore, the initial shape should be encoded as (Top layer: RW, RW, RW, Empty).

(b) Encoding Error

Question: The initial shape is  . Please predict what the shape will look like after performing {dyeing the top layer purple, cutting, mirroring, and rotating 90° clockwise}.

Incorrect Answer: The initial shape is encoded as (Top layer: RW, RW, RW, Empty), where R represents red and W represents windmill. After dyeing, the shape is encoded as (Top layer: RP, RP, RP, RP).

Explanation: In the initial shape, R represents the color. Therefore, during the dyeing operation, R should be changed to P. However, the shape code W was incorrectly changed to P in this case.

(c) Abstracting Error

Question: The initial shape is  . Please predict what the shape will look like after performing {dyeing the top layer purple, cutting, mirroring, and rotating 90° clockwise}.

Incorrect Answer: The initial shape is encoded as (Top layer: RW, RW, RW, Empty). After dyeing, the shape is encoded as (Top layer: PW, PW, PW, PW). After cutting, the shape is encoded as (Top layer: Empty, Empty, PW, PW). After rotating 90° clockwise, the shape is encoded as (Top layer: Empty, PW, PW, Empty).

Explanation: There are four quadrants in total, but only three quadrants are represented, which causes issues in the subsequent operations. Therefore, the initial shape should be encoded as (Top layer: RW, RW, RW, Empty).

(d) Execution Error

Figure 7: Examples of Errors

To gain a deeper understanding of the model's performance in Spatial Deformation Reasoning tasks, this study analyzes the models internal reasoning processes, aiming to reveal the mechanisms it employs to handle these tasks. As shown in the figure 5, we can find that models primarily processe the input through encoding the shapes, and by abstracting various operations into a set of standardized encoding operation rules, it then performs transformation reasoning based on this encoding.

It is apparent that, due to the relatively low complexity of the graphical shapes used, which closely resemble simple versions of real-world objects, the model faces fewer challenges in correctly encoding the shapes. Therefore, whether the model can reason accurately depends on its ability to effectively map spatial operations to the application of the encoding operation rules. Specifically, a correct mapping leads to accurate reasoning results, while an incorrect mapping results in erroneous reasoning outcomes.

Furthermore, we analyzed the models reasoning process across tasks with different dimensionalities and found that the models ability to abstract rules is smoother for 2D and 2.5D tasks. However, when faced with 3D tasks, its ability to abstract rules significantly decreases, making it difficult to establish effective correspondences in the mapping process, which ultimately affects the accuracy of reasoning. From the perspective of task nature, operations in 2D and 2.5D tasks, such as cutting, rotating, and stacking, typically do not involve complex coupled states, making the abstraction of rules relatively easier. However, when performing operations on three-dimensional shapes, each operation must account for the coupled states between the shapes. For example, each operation on a Rubiks Cube involves the interaction of colors across multiple faces. This means that, when mapping operations, the model must thoroughly consider the complex dependencies between the faces. This multidimensional interaction effect undoubtedly significantly increases the complexity and difficulty of the reasoning process.

B.2 General Evaluation Setup

To ensure reproducibility, unless otherwise specified, we apply a greedy decoding strategy (with a temperature setting of 0, and both top-p and top-k set to 1) for all models. In forward tasks, we present all options within a single image, as shown in Figure 8. All the prompt used in Inf-Bench is shown in Figures 9, 10, and 11.

B.3 Human Evaluation Setup

When evaluating human-level performance on Inf-Bench, human evaluators are allowed unlimited time to answer the questions but can only submit one answer. They receive both the question and the corresponding image simultaneously, and we do

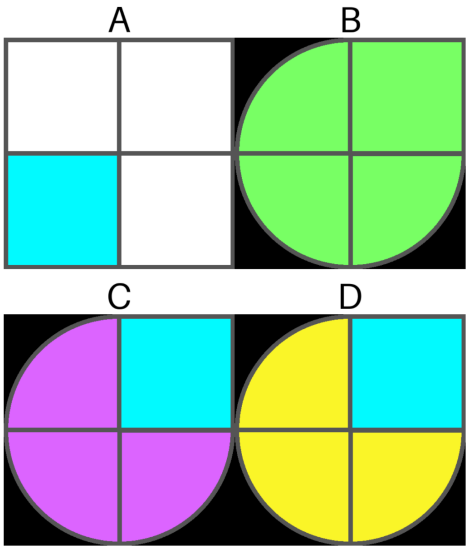


Figure 8: Example of the Option of Forward Task

not impose any restrictions on their ability to draft responses.

For 2D tasks, drawing intermediate steps is not allowed, while gestures can be used as an auxiliary form of expression. Based on my experience and responses from those I interviewed, gestures often help in problem-solving. Additionally, for tasks with many operations, they tend to match pattern options directly rather than reason step-by-step. Since 2D reasoning is relatively simple (involving only four quadrants), these tasks are not challenging for humans when accompanied by images.

C Technical Details

C.1 Detailed Overview of Encoding Methods

This section provides a detailed explanation of the encoding methods mentioned in section 6.2 of the main text.

C.1.1 2D Encoding Method

In the 2D task, shapes consist of two components: color and shape type. The color of each shape is represented by a lowercase letter, including red (r), green (g), blue (b), yellow (y), purple (p), cyan (c), colorless (u), and white (w). The shape type is represented by an uppercase letter, including circle (C), rectangle (R), windmill (W), sector (F), and star (S). Each shape comprises four quadrants, arranged in the order of quadrant 1, quadrant 2, quadrant 3, and quadrant 4, starting from the top right quadrant and proceeding clockwise. Each quadrant is represented by a pair of letters: the first letter represents the shape type, and the second letter represents the color. If a quadrant is empty, it is denoted by "–". A shape can consist of up to four layers, each comprising four quadrants. The shape encoding starts from the top layer and is arranged in ascending order from bottom to top, with layers separated by a colon. For example, "Su–Ry–" represents that in the first layer, quadrant 1 is a colorless star, quadrants 2 and 4 are empty, and quadrant 3 is a yellow rectangle. In this task, the player needs to transform the original shape into the target shape through a series of operations. Shape operations include cutting (removing quadrants 1 and 2, which is equivalent to cutting off the right half of the shape), clockwise rotation by 90°(rotating the quadrants clockwise), counterclockwise rotation by 90°(rotating the quadrants counterclockwise), filling (filling empty quadrants with specified shapes), mirroring (horizontally mirroring the entire shape), and coloring (changing the color of all shapes in the selected layer). These operations gradually transform the shape to match the target.

C.1.2 2.5D Encoding Method

In the 2.5D task, shapes not only have multiple layers, each consisting of four quadrants, but the maximum number of layers is four. The shape of each layer is represented by the same rules and arranged in ascending order from bottom to top. Each layer consists of four quadrants, with the order of quadrants being quadrant 1, quadrant 2, quadrant 3, and quadrant 4. Unlike the 2D task, the shapes in 2.5D tasks stack multiple layers on top of each other, with each layer separated by a colon (:) and arranged from the bottom layer to the top layer. The encoding rules for the shape are similar to those in the 2D task, where each layer is composed of four quadrants, and each quadrant is represented by a character pair for shape and color. If a quadrant is empty, it is represented by "–". Layers are separated by colons. For example, {"Layer 1": "Su–Ry–", "Layer 2": "——Wp–"} indicates that in the first layer, quadrant 1 is a colorless star, quadrant 3 is a yellow rectangle, and in the second layer, quadrant 3 is a windmill. In 2.5D tasks, in addition to basic operations (such as rotation, cutting, filling, coloring), players must also follow physical constraints: for instance, if a quadrant in one layer contains a shape, the corresponding quadrant in the lower layer must also contain a shape to satisfy the physical constraints. Operations include cutting (removing quadrants 1 and 2 from each layer), clockwise rotation by 90°(rotating each layer by 90°clockwise), coun-

2D

System Persona

You are a player of the game Shapez, and your goal is to transform a set of raw shapes into a target shape through a series of operations. First, I'll introduce the game's shape notation:
Each shape is represented by two characters, where the first character denotes the color, and the second character denotes the shape type. The color and type correspond as follows:
C: Circle; R: Rectangle; W: Windmill; Fan; S: Star; r: Red; g: Green; b: Blue ; y: Yellow; p: Purple; c: Cyan; u: Uncolored; w: White; --: Empty (No shape or color)
A single layer's shape code consists of the shape codes for four quadrants, in the following order: 'Quadrant 1, Quadrant 2, Quadrant 3, Quadrant 4'
For example, 'Su--Ry--' indicates that Quadrant 1 'Su' has an uncolored star, Quadrant 2 '--' is empty, Quadrant 3 'Ry' has a yellow rectangle, and Quadrant 4 '--' is empty.A shape can consist of one layers, with each layer represented by a shape code of up to four quadrants.
Next, I'll explain the available operations and the rules for each:

- Cutting (removes shapes in Quadrants 1 and 2 of the input shape, which is equal to cut the right side of the shape);
- Rotate clockwise by 90° (rotates all shapes clockwise by 90°, so Quadrant 1 becomes Quadrant 2, Quadrant 2 becomes Quadrant 3, Quadrant 3 becomes Quadrant 4, and Quadrant 4 becomes Quadrant 1);
- Rotate counterclockwise by 90° (rotates all shapes counterclockwise by 90°, so Quadrant 1 becomes Quadrant 4, Quadrant 4 becomes Quadrant 3, Quadrant 3 becomes Quadrant 2, and Quadrant 2 becomes Quadrant 1);
- Filling (fill all blank quadrants within the input shape using another shape);
- Mirror (perform a horizontal mirror operation on the entire shape, so Quadrant 1 becomes Quadrant 4, Quadrant 4 becomes Quadrant 1, Quadrant 2 becomes Quadrant 3, and Quadrant 3 becomes Quadrant 2);
- Coloring (input a shape and a color; change the color of every shape in each quadrant of the top layer to the given color);

Great, now you have become a Shapez master. From now on, you will answer my questions, and you only need to output the letter corresponding to your choice.

User Input (Forward)

This is the original shape, with its image: {initial shape image}.
If you perform {steps_number} operations on this shape, what will the resulting configuration look like? The operations are: {target_action_list}. Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {option shape image}.

User Input (Inverse)

This is the original shape, with its image: {initial shape image}.
If this shape is performed with {steps_number} operations, and the result is {target shape image}, can you infer what the operations are? Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {options action lists}

Figure 9: Prompt of Inf-Bench (2D)

terclockwise rotation by 90°(rotating each layer by 90°counterclockwise), stacking (stacking one shape layer on top of another), and coloring (changing the color of all shapes in a layer). These operations allow players to complete shape transformations in a multi-layered structure while adhering to specific physical constraints and operational rules.

C.1.3 3D Encoding Method

In the 3D task, shapes are represented as a Rubik's Cube. The Rubik's Cube consists of six faces, each represented by a 3x3 matrix of colors. Each face is composed of 3 rows and 3 columns of colored squares. The faces of the cube are represented by letters: Up (U), Down (D), Left (L), Right (R), Front (F), and Back (B). Each face's color is represented by color characters: yellow (y), white (w), red (r), orange (o), green (g), and blue (b). For example,

$$U : \begin{bmatrix} y & y & y \\ y & y & y \\ y & y & y \end{bmatrix}$$

Indicates that all squares on the upper face are yellow. The operations on the Rubik's Cube include a clockwise rotation of 90°(rotating a face 90°clockwise, changing the colors of that face and adjacent faces), and a counterclockwise rotation of 90°(rotating a face 90°counterclockwise, changing the colors of that face and adjacent faces). The shape encoding method is straightforward, where each face is represented by a 3x3 matrix, and each matrix element represents the color of the square at that position. Players need to transform the Rubik's Cube from its initial state to the target state by performing a series of rotations. The position of each face and the relative position of each square are

2.5D

System Persona

You are a player of the game Shapez, and your goal is to transform a set of raw shapes into a target shape through a series of operations. First, I'll introduce the game's shape notation:
Each shape is represented by two characters, where the first character denotes the color, and the second character denotes the shape type. The color and type correspond as follows:
C: Circle; R: Rectangle; W: Windmill; Fan; S: Star; r: Red; g: Green; b: Blue ; y: Yellow; p: Purple; c: Cyan; u: Uncolored; w: White; --: Empty (No shape or color) A single layer's shape code consists of the shape codes for four quadrants, in the following order: 'Quadrant 1, Quadrant 2, Quadrant 3, Quadrant 4' For example, 'Su--Ry--' indicates that Quadrant 1 'Su' has an uncolored star, Quadrant 2 '--' is empty, Quadrant 3 'Ry' has a yellow rectangle, and Quadrant 4 '--' is empty.
A shape can consist of multiple layers, with each layer represented by a shape code of up to four quadrants. The layers are arranged from bottom to top, as follows: {'Layer 1': 'Shape code for Layer 1', 'Layer 2': 'Shape code for Layer 2', 'Layer 3': 'Shape code for Layer 3', 'Layer 4': 'Shape code for Layer 4'}, the layer with the larger number is on the top. Note that not every shape has a fixed number of layers; some shapes have only one layer, while others may have two, three, or four layers. Note, each shape has a maximum of four layers.
The whole shapes follows specific physical rules:
1.If a layer contains only one quadrant with a shape, that quadrant must have a corresponding shape in the layer below. For example, in {'Layer 1': 'Su--Ry--', 'Layer 2': '------Wp--'}, quadrant 3's 'Wp' is the only shape in Layer 2, so quadrant 3 of the layer below (Layer 1) must also contain a shape, which in this case is 'Ry'. Conversely, in {'Layer 1': 'SuRy----', 'Layer 2': '------Wp--'}, quadrant 3's 'Wp' in Layer 2 is not supported by a shape in the corresponding quadrant of Layer 1, which is blank ('--'). This does not comply with physical laws.
2.If a layer contains two non-adjacent quadrants with shapes, those two quadrants must also have corresponding shapes in the layer below. For example, in {'Layer 1': 'Su--Ry--', 'Layer 2': 'Sb--Wp--'}, quadrants 1 ('Sb') and 3 ('Wp') are non-adjacent shapes in Layer 2. Therefore, quadrants 1 and 3 in Layer 1 must also contain shapes, as seen in 'Su' and 'Ry'. Conversely, in {'Layer 1': '----Ry--', 'Layer 2': 'Sb--Wp--'}, quadrant 1 in Layer 1 is blank ('--'), which does not conform to physical laws.
3.If a layer contains two adjacent quadrants with shapes, then at least one of those quadrants must contain a corresponding shape in the layer below. For example, in {'Layer 1': 'Su------', 'Layer 2': 'SbWp----'}, quadrants 1 ('Sb') and 2 ('Wp') are adjacent shapes in Layer 2. Therefore, either quadrant 1 or quadrant 2 in Layer 1 must contain a shape, as shown by 'Su' in quadrant 1. Conversely, in {'Layer 1': '----Ry--', 'Layer 2': 'SbWp----'}, both quadrants 1 and 2 in Layer 1 are blank ('--'), which does not comply with physical laws. 4.If a layer contains three quadrants with shapes, then at least one of those quadrants must have a corresponding shape in the layer below. For example, in {'Layer 1': '--Su----', 'Layer 2': 'SbWpCb--'}, quadrants 1 ('Sb'), 2 ('Wp'), and 3 ('Cb') are the three shapes in Layer 2. Therefore, at least one of quadrants 1, 2, or 3 in Layer 1 must contain a shape, as seen in quadrant 2 with 'Su'. Conversely, in {'Layer 1': '--------Ry', 'Layer 2': 'SbWpCb--'}, all quadrants in Layer 1 are blank ('--'), which does not comply with physical laws.
Next, I'll explain the available operations and the rules for each, please remember that all shapes obtained after operations must follow the laws of physics:

- Cutting (removes shapes in Quadrants 1 and 2 from each layer of the input shape, as well as cut the right side of the shape);
- Rotate clockwise by 90° (rotates all shapes clockwise by 90°, so Quadrant 1 becomes Quadrant 2, Quadrant 2 becomes Quadrant 3, Quadrant 3 becomes Quadrant 4, and Quadrant 4 becomes Q uadrant 1));
- Rotate counterclockwise by 90° (rotates all shapes counterclockwise by 90°, so Quadrant 1 becomes Quadrant 4, Quadrant 4 becomes Quadrant 3, Quadrant 3 becomes Quadrant 2, and Quadrant 2 becomes Quadrant 1);
- Mirror (perform a horizontal mirror operation on the entire shape, so Quadrant 1 becomes Quadrant 4, Quadrant 4 becomes Quadrant 1, Quadrant 2 becomes Quadrant 3, and Quadrant 3 becomes Quadrant 2);
- Stacking (stacks one shape on top of another);
- Coloring (input a shape and a color; change the color of every shape in each quadrant of the top layer to the given color); Great, now you have become a Shapez master.

From now on, you will answer my questions, and you should output the letter corresponding to your choice, and why you choose it as detailed as possible.

User Input (Forward)

This is the original shape, with its image: {initial shape image}.
If you perform {steps_number} operations on this shape, what will the resulting configuration look like? The operations are: {target_action_list}. Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {option shape image}.

User Input (Inverse)

This is the original shape, with its image: {initial shape image}.
If this shape is performed with {steps_number} operations, and the result is {target shape image}, can you infer what the operations are? Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {options action lists}

Figure 10: Prompt of Inf-Bench (2.5D)

crucial, requiring precise control of the rotations to ensure the final state matches the target. The rotation operations involve rotating each face by 90°, with interdependencies between faces, making

3D

System Persona

System Persona You are a Rubik's Cube master, you will answer my questions, and you should output the letter corresponding to your choice, and why you choose it as detailed as possible.

User Input (Forward)

This is the original shape, with its image: {initial shape image}.
If you perform {steps_number} operations on this shape, what will the resulting configuration look like? The operations are: {target_action_list}. Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {option shape image}.

User Input (Inverse)

This is the original shape, with its image: {initial shape image}.
If this shape is performed with {steps_number} operations, and the result is {target shape image}, can you infer what the operations are? Please select the correct answer from the options below. You only need to output the letter corresponding to your choice. The options are: {options action lists}

Figure 11: Prompt of Inf-Bench (3D)

the task more complex.

C.2 Training Details

In our experiment, we perform supervised fine-tuning using the LLaMA-Factory framework to enhance its multimodal processing capabilities further. We chose the LoRA (Low-Rank Adaptation) method as the core optimization strategy. This method efficiently adjusts model parameters through low-rank matrix decomposition, avoiding the high computational cost of full fine-tuning and significantly optimizing the performance of model on multimodal tasks (e.g., visual and linguistic integration). Specifically, we set the rank of LoRA to 8, an empirical compromise balancing model expressiveness with reduced parameter updates and lower memory requirements.

To ensure the efficiency and stability of distributed training, we integrate the DeepSpeed ZeRO-3 optimization strategy. This strategy optimizes model parameters, gradients, and optimizer states through partitioning, enabling efficient memory management and computational resource allocation. Thus, it significantly improves training speed and parallelism in multi-GPU or multi-node environments.

For the data, we use two datasets for supervised fine-tuning: shapez and cube. These datasets are designed to evaluate and enhance the model's multimodal understanding capabilities. The shapez dataset is further subdivided into shapez and shapez_2d subsets. The shapez subset focuses on the stitching, segmentation, and combining 2D images, involving complex reasoning tasks such as spatial relationship analysis and geometric transformations. In contrast, shapez_2d emphasizes specific variants of 2D images, such as simplified projections or graphic manipulations. The cube dataset, on the other hand, addresses 3D image understanding tasks. By fine-tuning on these diverse datasets, we aim to strengthen the model's generalization ability for visual-linguistic tasks across different dimensions and complexities.

For the training configuration, we set the per-device training batch size to 4, which helps avoid out-of-memory (OOM) errors while maintaining reasonable sample diversity given the limited GPU memory. We further set the gradient accumulation steps to 4, allowing us to simulate larger effective batch sizes despite smaller actual batches, thereby improving gradient estimation accuracy without increasing per-step computation load. To optimize the convergence process, we set the learning rate to 1e-4 (i.e., 0.0001) to maintain stable training. The number of training epochs is set to 3 to ensure efficient iteration under resource constraints.

C.3 Data Engine

C.3.1 Shapez Data Engine

The Shapez Data Engine is primarily used to generate and manage shape structures, supporting the generation of various shapes under specified conditions and transforming these shapes through a series of operations. Its core attributes include two types of fundamental data: shape types (where 'C', 'R', 'W', 'S' represent circle, rectangle, windmill, and star, respectively) and colors ('r', 'g', 'b', 'y', 'p', 'c', 'u', 'w' represent red, green, blue, yellow, purple, cyan, colorless, and white, respectively). The key methods of the en-

gine include `generate_shape_structures` and `execute_actions`. The `generate_shape_structures` method is used to generate a set of shape structures, where the number of layers, shape count, color, and consistency of shapes across layers can be specified. During the generation process, shapes and colors are randomly selected, and shapes for each layer are created based on the specified number of layers. Each layer's shape is ensured to contain at least one shape via the `generate_non_empty_layer` method, and the matching between layers is validated using the `check_layer_validity` method. Finally, the `generate_shape_structure` method combines these layers into a complete shape structure and returns it. The `execute_actions` method is used to apply a series of operations (such as rotation, cutting, stacking, coloring, etc.) to the generated shapes. These operations are applied individually through the `execute` method, and the transformed final shape is returned. The specific algorithm is found in Algorithm 1.

Algorithm 1 Shapez Data Engine

Input: $num\_structures$, num_layers, $color$, num_shapes, num_colors, all_the_same, $seed$ (optional)
Output: Generated shape structures
***Phase 0**: Initialize Parameters*
if $seed$ is provided **then**
    Set random seed
end if
Randomly select num_shapes from $shapes$
Randomly select num_colors from $colors$
***Phase 1**: Generate Shapes and Layers*
for each layer i from 1 to num_layers **do**
    Generate shape using $color$ and $shape$
    Generate a non-empty layer with at least one shape
    Check the validity of the layer with respect to the previous layer
end for
***Phase 2**: Create Shape Structure*
Combine valid layers to form a complete shape structure
Repeat the above steps for $num\_structures$ shapes
return Generated shape structures

C.3.2 Cube Data Engine

The Cube Data Engine focuses on generating and manipulating 3D cubes. Its design is intended to simulate the polyhedral structure and rotational operations of a Rubik's Cube. Each face of the cube is represented by a 3x3 matrix, with each position's color represented by specific color characters (e.g., yellow 'y', white 'w', red 'r', green 'g', etc.). This engine supports generating different cube configurations based on an initial cube state and executing various rotational operations on the cube. The `generate_cube_structure` method generates an initial cube state by specifying different colors and layouts. The position and color of each face must strictly match to ensure the validity of the cube's structure. The `execute_cube_actions` method performs a series of rotation actions, changing the color layout of the cube's faces. Each rotation operation is implemented through the `rotate_face` method, which can rotate the face clockwise or counterclockwise. Each operation affects multiple faces of the cube and their adjacent faces, resulting in a new state. Players can adjust the cube's state according to the task requirements through these operations, ultimately reaching the target configuration. The specific algorithm is found in Algorithm 2.

Algorithm 2 Cube Data Engine

Input: $initial\_cube\_state$, $actions$
Output: Modified cube state
***Phase 0**: Initialize Cube*
Generate initial cube state from $initial\_cube\_state$
***Phase 1**: Execute Cube Actions*
for each action in $actions$ **do**
    if action is a rotation **then**
        Rotate the specified face of the cube
    else if action is another type **then**
        Execute other cube operations (e.g., color change, swap)
    end if
end for
return the Final modified cube state after applying all actions

C.4 More Enhancement Details and Results

As mentioned in section 6.4 of the main text, we explore several mainstream reasoning enhancement methods, including Chain-of-Thought (COT), Few-

shot Learning, Self-reflection, Tool Invocation, and Reasoning and Acting (ReAct). This section provides a detailed explanation of the specific implementation of these methods.

C.4.1 COT.

The CoT presented in our paper is a zero-shot CoT, where we added the phrase "Lets think step by step" to the prompt. Futhermore, we have supplemented the manuscript with three different CoT methods:

One-shot CoT: The One-shot CoT method includes examples related to the target task within the prompt, allowing the model to improve its reasoning ability by learning from the reasoning process in these examples. Specifically, we add one example to the prompt and ask the model to Lets think step by step following the example.

CoT-Self-consistency: This method improves answer accuracy by performing a majority vote on the final answers from multiple CoT reasoning paths. Specifically, five reasoning paths are generated by the VLM, and the final result is selected through major voting based on the outcomes of these five paths.

ToT (Tree of Thoughts) Framework: In this framework, the model selects multiple potential paths at each reasoning stage and makes the final decision based on the reasoning results from each path. This approach helps the model make more reasonable decisions in multi-level reasoning by structuring the reasoning process.

As shown in the The Table 5, whether in the One-shot CoT, CoT-Self-consistency (CSc), or ToT (Tree of Thoughts) paradigms, we did not observe a significant performance improvement in most tasks. This suggests that, within the current model architecture and training process, changes in the prompt formulation did not help improve the model's ability in spatial transformation reasoning. We speculate that these results primarily reflect the fundamental challenges that current models face when processing complex spatial transformations, namely, the lack of deep spatial reasoning capabilities.

C.4.2 Few-shot Learning.

In the few-shot setting, we made attempts with varying numbers of examples, ranging from 1 to 5 (due to space limitations, only the result from 1 example was presented in the main text). Each example is aligned with the difficulty level (step) of the task, and each difficulty level has a fixed example (which will be fully displayed in the latest version). Additionally, we generate the complete reasoning chain through a ruled-based operation and incorporate it into the prompt. The Table 6 are the results from the experiments with different numbers of examples.

We found that few-shot prompting with 1-3 examples produced no significant performance gains. However, performance declined when using 4-5 examples. This suggests that prompting with examples alone could not overcome the model's inherent limitations in spatial transformation reasoning.

Table 5: **Evaluation on Inf-Bench with Different CoT methods.** The data represents the difference in the mean highest rank achieved by the models after 10 ladder challenges using the reasoning enhancement method, compared to the baseline performance (i.e., vanilla results).

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
	GPT-5					
Vanilla	9.91	10.49	7.56	9.27	1.25	0.00
Zero-shot CoT	-0.16	-0.40	-0.62	+0.13	-0.19	0.00
One-shot CoT	+0.09	-0.12	-0.37	-0.22	-0.14	0.00
CSc	+0.16	+0.07	-0.04	+0.37	+0.06	0.00
ToT	-0.46	-0.34	-0.06	-0.20	-0.10	0.00
	Llama-4-maverick					
Vanilla	3.64	5.17	2.97	2.89	0.97	0
Zero-shot CoT	0.21	0.04	0.03	0.11	0.01	+0.10
One-shot CoT	0.07	0.13	+0.05	0.06	+0.06	0.00
CSc	+0.02	+0.19	+0.30	+0.15	+0.15	+0.15
ToT	0.52	0.22	0.09	0.14	+0.11	0.00

Table 6: **Evaluation on Inf-Bench with Different Number of Examples in Few-shot learning.** The data represents the difference in the mean highest rank achieved by the models after 10 ladder challenges using the reasoning enhancement method, compared to the baseline performance (i.e., vanilla results).

	2D		2.5D		3D	
Number	For	Inv	For	Inv	For	Inv
	GPT-5					
Vanilla	9.91	10.49	7.56	9.27	1.25	0.00
1	+0.58	+0.32	+0.45	+0.10	+0.01	0.00
2	+0.37	+0.26	+0.17	+0.17	0.00	0.00
3	+0.34	+0.36	-0.05	-0.15	+0.05	0.00
4	-1.02	-0.94	-1.02	-0.80	-0.04	0.00
5	-1.60	-1.30	-1.49	-1.05	-0.25	0.00
	Llama-4-maverick					
Vanilla	3.64	5.17	2.97	2.89	0.97	0.00
1	0.07	0.13	+0.16	+0.09	+0.19	0.00
2	0.27	0.16	0.12	0.08	+0.01	+0.02
3	+0.07	+0.21	0.01	0.13	+0.01	0.00
4	1.05	1.01	0.92	0.82	0.51	0.00
5	1.29	1.41	1.10	0.99	0.54	0.00

C.4.3 Self-reflection.

A cycle mechanism of initialization, validation, and correction is introduced to simulate humans' self-reflection process when solving problems. This framework first generates an initial answer, then interacts with external tools to validate the answer's quality, generates self-criticism, and finally refines the answer based on this criticism. Given the model M and input x, the initial answer is generated by the prompt $\mathcal{P}$:

$$\hat{y}_0 \sim P_M(\cdot \mid \mathcal{P} \oplus x).$$

Subsequently, the model interacts with external tools to evaluate $\hat{y}_i$ and generate criticism $c_i \sim P_M(\cdot \mid \mathcal{P} \oplus x \oplus \hat{y}_i, T)$. These task-specific criticisms can be used to assess various attributes of the output, such as truthfulness, feasibility, or safety. Finally, the model generates an improved answer based on input x, previous output $\hat{y}_i$, and criticism c_i:

$$\hat{y}_{i+1} \sim P_M(\cdot \mid \mathcal{P} \oplus x \oplus \hat{y}_i \oplus c_i).$$

Criticism plays a key role in correcting errors by identifying errors, providing feasible suggestions, or offering reliable justifications through interactions with external tools, guiding the new generation to avoid similar mistakes. This "validate-correct-validate" cycle can repeat multiple times until a specific stopping condition is met, such as the validation process satisfying a requirement, reaching the maximum number of iterations, or receiving environmental feedback. The specific algorithm is found in Algorithm 3.

Algorithm 3 SELF-REFLECTION

Require: Input x, prompt $\wp$, model M, external tools $T = \{T_1, T_2, \ldots, T_k\}$, number of iterations n
Ensure: Corrected output $\hat{y}$ from M

Generate initial output $\hat{y}_0 \sim P_M(\cdot|\wp \oplus x)$ ▷ Initialization
for $i \leftarrow 0$ to $n-1$ **do**
  Verify $\hat{y}_i$ through interaction with T to obtain critiques $c_i \sim P_M(\cdot|\wp \oplus x \oplus \hat{y}_i, T)$ ▷ Verification
  if c_i indicates that $\hat{y}_i$ is correct **then** ▷ Stopping Criteria
    return $\hat{y}_i$
  end if
  $\hat{y}_{i+1} \sim P_M(\cdot|\wp \oplus x \oplus \hat{y}_i \oplus c_i)$ ▷ Correction
end for
return $\hat{y}_n$

C.4.4 Tool Invocation.

This is the most straightforward tool usage paradigm, taking the form of an alternating dialogue between the language model and the tools. In this framework, the model generates outputs that include tool invocation requests. The system extracts and executes these calls and then provides the results of these executions back to the model, creating a dialogue loop. The specific algorithm is found in Algorithm 4.

Here are the tools we used: **Unity Integration (Advanced):** Provides advanced and in-depth control over the Unity editor, such as direct code execution and file access.

Unity3D Game Engine: Provides standard interaction capabilities with the Unity editor, such as accessing logs, running tests, and viewing hierarchical structures.

Blender (by ahujasid): Allows LLMs to create, model, and operate in 3D scenes within Blender through instructions.

3D-MCP / Rodin: A general-purpose 3D model context protocol designed to provide standardized interfaces for LLMs and various 3D software. Rodin is one of its implementations and can generate 3D models.

E2B (Code Sandbox): Executes AI-generated code (e.g., Python, JavaScript) in a secure cloud-based sandbox.

Algorithm 4 TOOL

Require: Input x, prompt $\wp$, model M, external tools $T = \{T_1, T_2, \ldots, T_k\}$, max steps n
Ensure: Final output y from M

Initialize conversation history $h \leftarrow \wp \oplus x$
for $i \leftarrow 1$ to n **do**
  Generate model response $r_i \sim P_M(\cdot|h)$
  if no tool call detected in r_i **then**
    return r_i
  end if
  Extract tool call $(tool\_name, tool\_input)$ from r_i
  Execute tool: $tool\_output \leftarrow T[tool\_name](tool\_input)$
  Update history: $h \leftarrow h \oplus r_i \oplus tool\_output$
end for
Generate final response $y \sim P_M(\cdot|h)$
return y

C.4.5 ReAct (Reasoning and Acting).

Here, the model implements a cycle of thinking, acting, and observing, significantly enhancing the problem-solving ability of the language model. It introduces two additional spaces: an action space A, which contains a variety of executable operations, and a thinking space L, for internal reasoning within the model. Unlike direct tool invocation, ReAct emphasizes explicit reasoning steps, enabling the model to "think" about the following action. In the ReAct framework, each iteration includes three key steps:

1. First, the model generates a reasoning process in the thinking space, outlining the problem-solving approach;

2. Second, based on this reasoning, the model decides on the appropriate action to take;

3. Third, it observes the result of the action and integrates this information into the context.

This structured reasoning-action-observation cycle allows the model to handle complex problems more systematically, especially tasks that require multi-step reasoning and tool collaboration. The specific algorithm is found in Algorithm 5.

Algorithm 5 REACT

Require: Input x, prompt $\wp$, model M, external tools $T = \{T_1, T_2, \ldots, T_k\}$, max steps n, action space A, language space L

Ensure: Final output y from M

Initialize context $c_1 \leftarrow \wp \oplus x$
for $i \leftarrow 1$ to n **do**
  Generate thought $t_i \sim P_M(\cdot|c_i)$ where $t_i \in L$ ▷ Thinking
  Update context: $c_i \leftarrow c_i \oplus t_i$
  Generate action $a_i \sim P_M(\cdot|c_i)$ where $a_i \in A$ ▷ Acting
  if a_i is a final answer **then**
    return a_i
  end if
  Execute action to get observation: $o_i \leftarrow Execute(a_i, T)$ ▷ Observing
  Update context: $c_{i+1} \leftarrow c_i \oplus a_i \oplus o_i$
end for
Generate final answer $y \sim P_M(\cdot|c_{n+1})$
return y

D Ablation of Prompts Impact

This part will provide a complete ablation analysis in this section and introduce the types of prompts involved in the experiments:

Prompt 1: Rule Introduction with Text (currently used prompt form, presented in the paper). This form includes detailed textual descriptions, introducing task rules, operational steps, and possible transformations, with the model reasoning based on these descriptions.

Prompt 2: Rule Introduction with Images. Instead of text descriptions, this prompt uses visual illustrations to display the task rules, allowing the model to directly understand task operations from the images. We aim to observe the impact of image-based prompts on reasoning performance.

Prompt 3: Rule Introduction with Text and Images. This multimodal prompt combines both text and images, providing both textual descriptions and images of the operations. We assess whether this combination of modalities helps improve the models reasoning ability.

Prompt 4: Simplified Prompt. In this form, we provide only the most essential task instructions, removing all detailed explanations about the game background, shape encoding rules, and physical laws. We focus on how the model handles simplified information and evaluate its impact on the reasoning process.

Prompt 5: Detailed Prompt. In this form, we provide a comprehensive textual description of each operation, offering the model full guidance. We aim to investigate whether the model can leverage detailed task descriptions to improve reasoning accuracy.

Prompt 6: Chain of Thought (CoT). We add the phrase Lets think step by step to the prompt to encourage the model to reason incrementally. We assess whether this step-by-step reasoning guidance helps with reasoning performance on complex tasks.

Prompt 7: Few-shot Prompt. This prompt includes examples with difficulty levels similar to the current task, evaluating whether few-shot prompting can help the model better understand the task and improve reasoning ability.

The experimental results in Figure 7 indicate that prompt adjustments did not significantly improve the model's reasoning ability. In fact, certain simplified versions of the prompts actually had a detrimental effect on performance. Specifically, when

Table 7: **Evaluation on Inf-Bench with Different Prompts.** The data represents the difference in the mean highest rank achieved by the models after 10 ladder challenges using the reasoning enhancement method, compared to the baseline performance (i.e., vanilla results).

	2D		2.5D		3D	
	For	Inv	For	Inv	For	Inv
			GPT-5			
Vanilla	9.91	10.49	7.56	9.27	1.25	0.00
P2	-2.02	-2.51	-1.75	-1.32	-0.81	0.00
P3	-0.15	-0.54	+0.03	-0.08	-0.13	0.00
P4	-4.90	-6.08	-3.69	-5.20	-1.02	0.00
P5	-0.20	+0.51	+0.28	+0.19	+0.03	+0.11
P6	-0.16	-0.42	-0.63	+0.13	-0.22	0.00
P7	+0.56	+0.32	+0.45	+0.09	+0.01	0.00
			Llama-4-maverick			
Vanilla	3.64	5.17	2.97	2.89	0.97	0
P2	-1.72	-2.60	-1.55	-2.12	-0.77	0.00
P3	-0.55	-0.06	-0.23	-1.73	-0.11	+0.10
P4	-2.60	-3.82	-2.34	-2.02	-0.97	0.00
P5	+0.08	-0.08	-0.08	+0.24	+0.05	0.00
P6	-0.21	-0.04	-0.03	-0.11	-0.01	+0.10
P7	-0.07	-0.13	+0.16	+0.09	+0.19	0.00

using image-based rule introduction (Prompt 2) and the simplified prompt (Prompt 4), the model's reasoning ability declined, with lower accuracy and reasoning depth. This suggests that these forms were ineffective for complex spatial transformation tasks. Despite attempts with detailed text prompts (Prompt 5), multimodal prompts (Prompt 3), and enhanced reasoning methods (Prompt 6 and 7), the model's performance on most tasks did not show significant improvement. This indicates that changes in the prompt did not overcome the inherent limitations of the model in spatial reasoning, reflecting the model's lack of deep spatial reasoning capabilities.

These findings highlight the challenges faced by VLMs in spatial reasoning, not only as a matter of prompt design but also as a fundamental bottleneck in the model's ability to handle complex spatial transformations, particularly in 3D tasks.

E More Details about Inf-Bench.

E.1 Shapes and Colors

In the Shapez game, shapes can be combined using different types and colors. The shape types include circles, rectangles, windmills, sectors, and stars. Each shape can be represented in various colors, such as red, green, blue, yellow, purple, cyan, colorless, and white. All of these are illustrated in Figure 12.

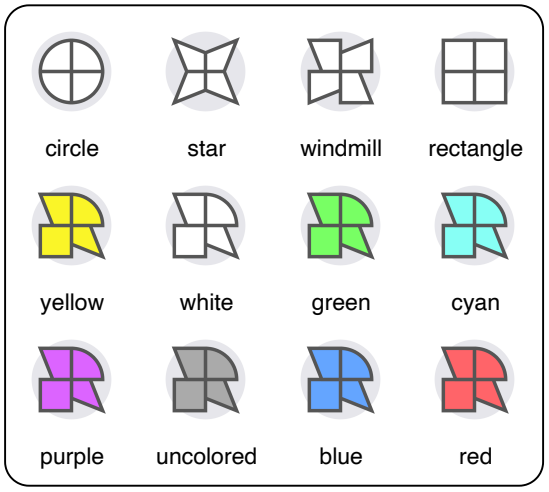


Figure 12: All Shapes and Colors

E.2 Operations

The action space of all the operations are shown in Figure 13.

E.2.1 All Operations in 2D and 2.5D

- **Cutting**: Removing a portion of the shape, typically by cutting off the right-side quadrants (quadrant 1 and quadrant 2). For example, cutting a shape with four quadrants into two parts from the right side.
- **Clockwise Rotation 90°**: Rotating all the quadrants of the shape 90°clockwise, with quadrant 1 becoming quadrant 2, quadrant 2 becoming quadrant 3, quadrant 3 becoming quadrant 4, and quadrant 4 becoming quadrant 1.
- **Counterclockwise Rotation 90°**: Rotating all quadrants of the shape 90°counterclockwise, with quadrant 1 becoming quadrant 4, quadrant 4 becoming quadrant 3, quadrant 3 becoming quadrant 2, and quadrant 2 becoming quadrant 1.
- **Filling**: Filling empty quadrants with a specified shape, such as filling the blank quadrant with a rectangle or circle.
- **Mirror**: Performing a horizontal mirror operation on the entire shape, swapping quadrants 1 and 4, and quadrants 2 and 3. The mirror operation reverses the left and right sides of the shape.
- **Coloring**: Changing the color of all quadrants in the shape allows a new color to be applied

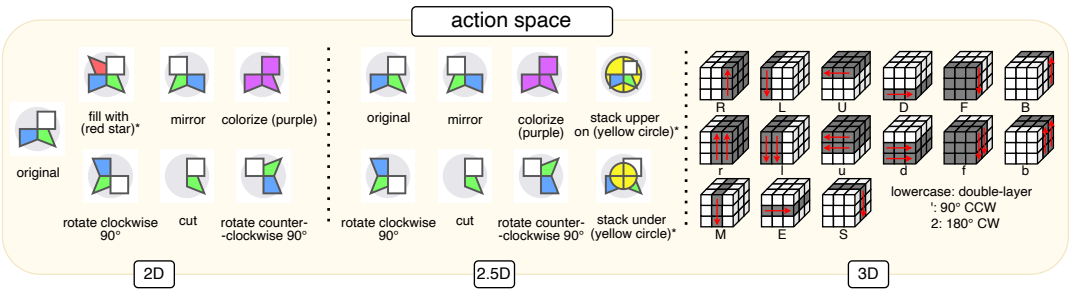


Figure 13: **Action Space.** Actions like rotation and coloring are common in 2D and 2.5D, with 2D including "filling" and 2.5D adding "stacking." The 3D section focuses on Rubik's Cube rotations and movements in three dimensions.

to the shapes in a specified layer or across all layers.

- **Stacking**: This operation is primarily used in the 2.5D task, where one shape layer is stacked on top of another, forming a multi-layered structure.

E.2.2 All Rotation Operations in 3D

Face Rotations

These operations rotate one of the six faces of the Rubik's Cube. Each face can be rotated clockwise, counterclockwise, or by 180°.

- **R**: Rotate the right face 90°clockwise
- **U**: Rotate the upper face 90°clockwise
- **F**: Rotate the front face 90°clockwise
- **D**: Rotate the down face 90°clockwise
- **L**: Rotate the left face 90°clockwise
- **B**: Rotate the back face 90°clockwise
- **R'**: Rotate the right face 90°counterclockwise
- **U'**: Rotate the upper face 90°counterclockwise
- **F'**: Rotate the front face 90°counterclockwise
- **D'**: Rotate the down face 90°counterclockwise
- **L'**: Rotate the left face 90°counterclockwise
- **B'**: Rotate the back face 90°counterclockwise
- **R2**: Rotate the right face 180°
- **U2**: Rotate the upper face 180°
- **F2**: Rotate the front face 180°
- **D2**: Rotate the down face 180°
- **L2**: Rotate the left face 180°
- **B2**: Rotate the back face 180°

Wide Layer Rotations

These operations rotate two layers of the Rubik's Cube at once, affecting adjacent layers.

- **r**: Rotate the right two layers 90°clockwise
- **u**: Rotate the upper two layers 90°clockwise
- **f**: Rotate the front two layers 90°clockwise
- **d**: Rotate the down two layers 90°clockwise
- **l**: Rotate the left two layers 90°clockwise
- **b**: Rotate the back two layers 90°clockwise
- **r'**: Rotate the right two layers 90°counterclockwise
- **u'**: Rotate the upper two layers 90°counterclockwise
- **f'**: Rotate the front two layers 90°counterclockwise
- **d'**: Rotate the down two layers 90°counterclockwise
- **l'**: Rotate the left two layers 90°counterclockwise
- **b'**: Rotate the back two layers 90°counterclockwise
- **r2**: Rotate the right two layers 180°
- **u2**: Rotate the upper two layers 180°
- **f2**: Rotate the front two layers 180°

- **d2**: Rotate the down two layers 180°
- **l2**: Rotate the left two layers 180°
- **b2**: Rotate the back two layers 180°

Middle Layer Rotations

These operations rotate the middle layers of the Rubik's Cube, affecting rotations between faces.

- **M**: Rotate the middle layer between the left and right faces 90°clockwise
- **S**: Rotate the middle layer between the front and back faces 90°clockwise
- **E**: Rotate the middle layer between the top and bottom faces 90°clockwise
- **M'**: Rotate the middle layer between the left and right faces 90°counterclockwise
- **S'**: Rotate the middle layer between the front and back faces 90°counterclockwise
- **E'**: Rotate the middle layer between the top and bottom faces 90°counterclockwise
- **M2**: Rotate the middle layer between the left and right faces 180°
- **S2**: Rotate the middle layer between the front and back faces 180°
- **E2**: Rotate the middle layer between the top and bottom faces 180°