VISGYM: DIVERSE, CUSTOMIZABLE, SCALABLE ENVIRONMENTS FOR MULTIMODAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern Vision–Language Models (VLMs) remain poorly characterized in multi-step visual interactions, particularly in how they integrate perception, memory, and action over long horizons. We introduce VISGYM, a gymnasium of 17 environments for evaluating and training VLMs. The suite spans symbolic puzzles, real-image understanding, navigation, and manipulation, and provides flexible controls over difficulty, input representation, planning horizon, and feedback. We also provide multi-step solvers that generate structured demonstrations, enabling supervised finetuning. Our evaluations show that all frontier models struggle in interactive settings, achieving low success rates in both the easy (26.8%) and hard (12.6%) configurations. Our experiments reveal notable limitations: models struggle to effectively leverage long context, performing worse with an unbounded history than with truncated windows. Furthermore, we find that several text-based symbolic tasks become substantially harder once rendered visually. However, explicit goal observations, textual feedback, and exploratory demonstrations in partially observable or unknown-dynamics settings for supervised finetuning yield consistent gains, and solver-generated multi-step demonstrations generalize across tasks and domains. Together, these results position VISGYM as a principled and scalable foundation for diagnosing and training visually interactive agents.

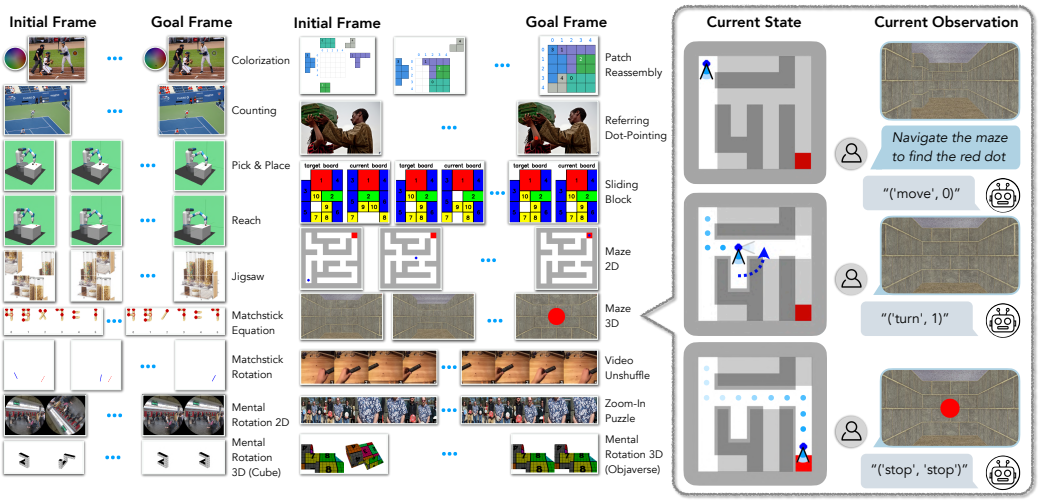


Figure 1: An overview of VISGYM. (*Left*) VISGYM consists of 17 diverse, long-horizon environments designed to systematically evaluate, diagnose, and train VLMs on visually interactive tasks with different domains, levels of state observability, and types of observations. (*Right*) An example trajectory for the Maze 3D navigation task illustrates a partially observable environment consisting of non-structured synthetic renderings. Here, a VLM is prompted with (1) the task description (*simplified in the figure*) and (2) a set of available actions to use (*not shown in the figure for simplicity*). The agent must select each action conditioned on both its past actions and observation history for its decision-making.

1 INTRODUCTION

Humans navigate complex tasks in visually rich and interactive settings: manipulating objects, using devices, or exploring unfamiliar environments. Success in these settings hinges on the tight coupling

Table 1: Comparison among frameworks for visually interactive decision-making. Struct. Obs. and **Non-struct. Obs.** indicate whether visual inputs can be parsed into structured text. **POMDP** denotes partial observability with hidden states. **Multi-Domain** covers diversity across domains (*e.g.*, robotics, computer use, games, puzzles). **Scalable Episodes** marks automatic, large-scale generation. **SFT** and **Online RL** show support for finetuning and reinforcement learning.

Framework	# Tasks	Environments					Training	
		Struct. Obs.	Non-struct. Obs.	POMDP	Multi Domain	Scalable Episodes	SFT	Online RL
Evaluation-only								
OSWorld (Xie et al., 2024)	369	✓		✓				
LIBERO (Liu et al., 2023)	130		✓	✓		✓		
VideoGameBench (Zhang et al., 2025)	23	✓	✓	✓				
LMGame-Bench (Hu et al., 2025)	6	✓	✓	✓				
Evaluation and Training								
VLABench (Zhang et al., 2025)	100		✓	✓		✓	✓	✓
VLM-Gym (Chen et al., 2025)	4	✓				✓	✓	✓
KORGym (Shi et al., 2025)	6	✓		✓		✓		✓
VisualAgentBench (Liu et al., 2024)	5	✓	✓	✓	✓		✓	✓
VAGEN (Wang et al., 2025)	5	✓	✓	✓	✓	✓		✓
VISGYM (Ours)	17	✓	✓	✓	✓	✓	✓	✓

of perception, memory, and action over multiple steps (Gibson, 1979; Henderson, 2001). Foundation Vision–Language Models (VLMs) have made remarkable progress on static vision–language benchmarks (Yue et al., 2024; Lu et al., 2023; Wang et al., 2024) and on text-based multi-step tasks such as web browsing and coding (Sirdeshmukh et al., 2025; Wei et al., 2025; Jimenez et al., 2023). Yet when visual observations must be integrated into multi-step decision-making, their behavior remains far less understood. Recent evaluations across robotic manipulation, computer-use agents, and gaming agents highlight a range of challenges for visually interactive decision-making, including low task success rates, brittle visual grounding, and weak generalization (Zhang et al., 2025; Xie et al., 2024; Zhang et al., 2025; Liu et al., 2023; Hu et al., 2025; Chen et al., 2025; Shi et al., 2025; Liu et al., 2024). Although these insights are valuable, they tend to be domain-specific and observational, offering limited *systematic, controlled* diagnosis of how domain-agnostic factors such as context length, representation modality, feedback, or goal visibility affect model performance across tasks.

We introduce VISGYM, a highly diverse, scalable, and customizable gymnasium with 17 long-horizon environments designed to isolate what limits interactive decision-making across domains and to expose where current VLMs break down. The suite spans symbolic puzzles, real-image understanding, navigation, and manipulation tasks, each with distinct observability and dynamics and equipped with oracle multi-step solvers for supervised finetuning (framework comparison in Tab. 1). Crucially, VISGYM provides fine-grained controls over input representation, difficulty, history length, planning horizon, and feedback, enabling domain-agnostic, systematic analysis of model behavior. Building on prior domain-specific studies, we conduct cross-domain controlled experiments that examine how these factors, together with module finetuning and data curation, affect performance in multi-step visual decision-making.

Across 11 state-of-the-art models, even the strongest achieve only 26.79% and 12.57% success in the easy and hard settings, respectively. Our analyses reveal several concrete, cross-domain failure modes: ***(1)*** models struggle to effectively leverage long-term context, showing a reversed-U relationship where performance degrades as the context grows unbounded; ***(2)*** VLMs struggle with low-level perceptual grounding, a limitation highlighted by symbolic variants of tasks being substantially easier than their visually rendered counterparts; ***(3)*** models struggle to infer task states and outcomes from purely visual transitions, consistently relying on explicit textual feedback to boost performance; ***(4)*** the benefit of providing explicit goal observations is brittle and can backfire: while explicit goals can yield large gains, limited visual perception can cause models to misidentify them and, paradoxically, perform worse than with no goal at all; ***(5)*** models fail to learn from standard demonstrations under partial observability or unknown dynamics, requiring information-revealing demonstrations that expose hidden states or clarify dynamics to significantly improve supervised finetuning outcomes.

Beyond diagnosis, we show that supervised finetuning on solver-generated multi-step demonstrations yields improvements that generalize across unseen tasks and transfer to downstream domains requiring visual interaction. Together, these findings establish VISGYM as a unified and extensible framework for diagnosing, understanding, and improving VLMs in visually interactive decision-making.

Table 2: VISGYM environments. For each environment, we specify (1) **Domain**: whether observations come from **Real** or **Synthetic** images, (2) **Observability (Obs.)**: **Full** or potentially **Partial**, (3) **Dynamics (Dyn.)**: **Known** vs. **Unknown** dynamics, (4) **Parameters (P.)**: number of difficulty parameters, and (5) **Available Actions**.

Environment	Domain	Obs.	Dyn.	P.	Available Actions
Colorization (104)	Real	Full	Known	1	`rotate`(θ); `saturate`(δ); `stop`()
Counting (31)	Real	Full	Known	2	`mark`(x, y); `undo`(); `guess`(N); `stop`()
Jigsaw (28)	Real	Full	Known	2	`swap`((r_1, c_1), (r_2, c_2)); `reorder`([...]); `stop`()
Matchstick Equation (43)	Synthetic	Full	Known	1	`move`([i, s, j, t]); `undo`(); `stop`()
Matchstick Rotation (45)	Synthetic	Full	Unknown	3	`move`([$dx, dy, d\theta$]); `stop`()
Maze 2D (44)	Synthetic	Full	Known	2	`move`(d); `stop`()
Maze 3D (44)	Synthetic	Partial	Known	2	`move`(0); `turn`(d); `stop`()
Mental Rotation 2D (19)	Real	Full	Known	1	`rotate`(θ); `stop`()
Mental Rotation 3D (CUBE) (67; 71)	Synthetic	Partial	Known	3	`rotate`([dy, dp, dr]); `stop`()
Mental Rotation 3D (OBJAVERSE) (71; 21)	Synthetic	Partial	Known	1	`rotate`([dr, dp, dy]); `stop`()
MuJoCo Fetch (PICK-AND-PLACE) (86)	Synthetic	Partial	Unknown	0	`move`([x, y, z]); `gripper`(g); `stop`()
MuJoCo Fetch (REACH) (86)	Synthetic	Partial	Unknown	0	`move`([x, y, z]); `stop`()
Patch Reassembly (29)	Synthetic	Full	Known	2	`place`(p, r, c); `remove`(p); `stop`()
Referring Dot-Pointing (40)	Real	Full	Known	0	`mark`(x, y); `stop`()
Sliding Block (76)	Synthetic	Full	Known	1	`move`(b, d); `stop`()
Video Unshuffle (30; 61)	Real	Full	Known	3	`swap`(i, j); `reorder`([...]); `stop`()
Zoom-In Puzzle (6)	Real	Full	Known	5	`swap`(i, j); `reorder`([...]); `stop`()

2 VISGYM

VISGYM contains 17 visually interactive environments. Each environment exposes initialization parameters that control task configuration and difficulty. We provide a high-level overview of the environments in Tab. 2 and detailed descriptions with visualizations in Sec. §C. VISGYM is built on top of the Gymnasium framework (Brockman et al., 2016; Towers et al., 2024), the same library underlying MuJoCo (Todorov et al., 2012) and Atari (Bellemare et al., 2013). Since vision–language agents can interpret images, read instructions, and produce free-form text, we extend Gymnasium with the following enhancements:

Function-Conditioned Action Space. Instead of the discrete or continuous action vectors used in standard Gymnasium environments, we represent actions as function calls with parameters (*e.g.*, `('swap', (1, 2)), ('rotate', (30.5, 20.4, 15.1))`). This abstraction allows models to leverage their function-calling capabilities and compose strategies across domains.

Function Instructions. Each task defines a set of functions and their parameter spaces. To enable zero-shot rollouts, we provide a natural-language description of these functions and their argument constraints as part of the initial prompt before the model takes its first action. Instructions for each task are shown in Sec. §C.

Environment Feedback. In addition to visual feedback as the model interacts with the environment, it optionally provides textual feedback describing the effect of each action (*e.g.*, "invalid format," "out of bounds," "executed"). This helps models with weaker visual perception better ground their actions.

Solver. We implement heuristic multi-step solvers that complete each task using the available actions. The solver supports (1) multiple solving strategies and (2) optional stochasticity, enabling the generation of diverse demonstration trajectories for supervised fine-tuning. See Sec. §B for the solver design of each task.

Together, these design choices yield a highly customizable interface. Each task can define its own action functions, instruction set, and solver, while the unified `step` function handles parsing, validation, execution, and feedback (Algorithm 1 in Sec. §D). This modular structure makes it easy to add new tasks, vary action spaces, and generate visual and textual supervision for VLM agents.

3 EVALUATING FRONTIER MODELS WITH VISGYM

In this section, we evaluate vision-language models on VISGYM. We describe our evaluation setup in Sec. §3.1 and present results and observational analysis in Sec. §3.2.

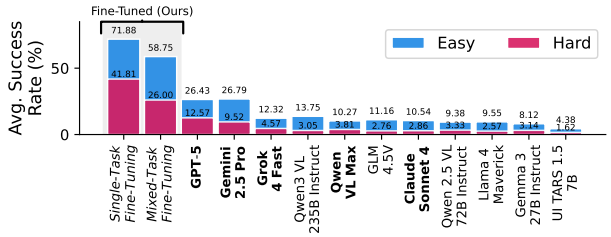


Figure 2: Average task success rate for frontier models and our finetuned models. Proprietary models are in **bold** and our finetuned models are *italicized*.

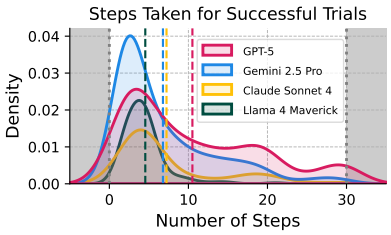


Figure 3: Density curve of steps taken for successful trajectories. Colored dashed line marks each model's mean number of steps.

3.1 Evaluation Setup

We evaluate 11 vision-language models spanning three categories: **proprietary** (Gemini 2.5 Pro (DeepMind, 2025), GPT-5 (OpenAI, 2025), Claude Sonnet 4 (Team, 2025), Grok 4 Fast (xAI, 2025), Qwen-VL-Max (Bai et al., 2025)); **open-weight** models (Qwen3-VL-235B-Instruct (Yang et al., 2025), GLM-4.5V (Hong et al., 2025), Llama-4-Maverick (Touvron et al., 2023), Qwen-2.5-VL-72B-Instruct (Bai et al., 2025), Gemma 3-27B-Instruct (Team et al., 2025)); and **specialized** models targeted at GUI/game environments (UI-Tars-1.5-7B (Qin et al., 2025)). We access all proprietary and hosted models through OpenRouter and thus ensure a consistent prompting interface and inference pipeline. We additionally evaluate models that we finetune on solver demonstrations. We provide details of the supervised finetuning setup in Sec. §5.1.

All models are evaluated in a multi-turn manner. At each step t, the model receives the full history

$$H_t = \big(I,\ \{(o_\tau, a_\tau, f_\tau)\}_{\tau<t}\big), \tag{1}$$

where $I \in \mathbb{R}^{L_I}$ is the task instruction, $o_\tau \in \mathbb{R}^{H\times W\times C}$ the observation, $a_\tau \in \mathbb{R}^{L_a}$ the action, and $f_\tau \in \mathbb{R}^{L_f}$ the environment feedback. The model then outputs an action a_t. If it outputs the stop action, the environment terminates and returns a binary reward indicating task success. In addition, we limit the number of interaction steps to 20 for the easy setting and 30 for the hard setting. All tasks are designed to be solvable within these limits, and the environment explicitly provides the number of remaining steps as part of its feedback. We also ensure that the length of interaction history is within models' context window. We evaluate each model on 70 episodes per task and setting (*i.e.*, easy, hard).

3.2 Result and Analysis

Frontier VLMs Fail on VisGym. We show the per-task success rate and the average task success rate of the frontier models in Figure 4 and Figure 2, respectively.

Even the best-performing frontier model, GPT-5, achieves only 26.43% on VisGym (Easy) and 12.57% on VisGym (Hard), indicating that VisGym poses a significant challenge for existing models.

Model Specialization. We compare the 3 strongest models: Gemini 2.5 Pro, GPT-5, Qwen3-VL-235B Instruct. GPT-5 shows the best ability to handle long-context visual interactions. This is reflected in its stronger performance on matchstick rotation where the scale is unknown, its higher scores overall on the hard setting (Fig. 2), and its visibly longer tail in the number of steps taken to successfully solve tasks compared to the other models (Fig. 3). Gemini 2.5 Pro is good at low-level visual perception. This is reflected in its strongest performance on Jigsaw, Maze 2D, Zoom-In Puzzle, and Sliding Block, all of which demand tight spatial alignment, accurate correspondence of local patterns, and sensitivity to subtle visual cues. Qwen-3-VL is in particular capable of object localization (*e.g.*, strongest in Referring Dot-Pointing).

Examining the step count distribution (smoothed density curve) for successful trajectories across models (Fig. 3), we found that most models (*i.e.*, Gemini 2.5 Pro, Claude Sonnet 4, and Llama-4-Maverick) only peaked around 3-5 steps, followed by a sharp drop in successful trajectories when they spend more steps. This indicates limited capability in effectively handling long-context multi-step visual interactions.

Common Failure Patterns. We identify recurring failures using automated failure discovery methods (Dunlap et al., 2025; Lisa Dunlap et al., 2025), which employ a VLM annotator (GPT-4.1)

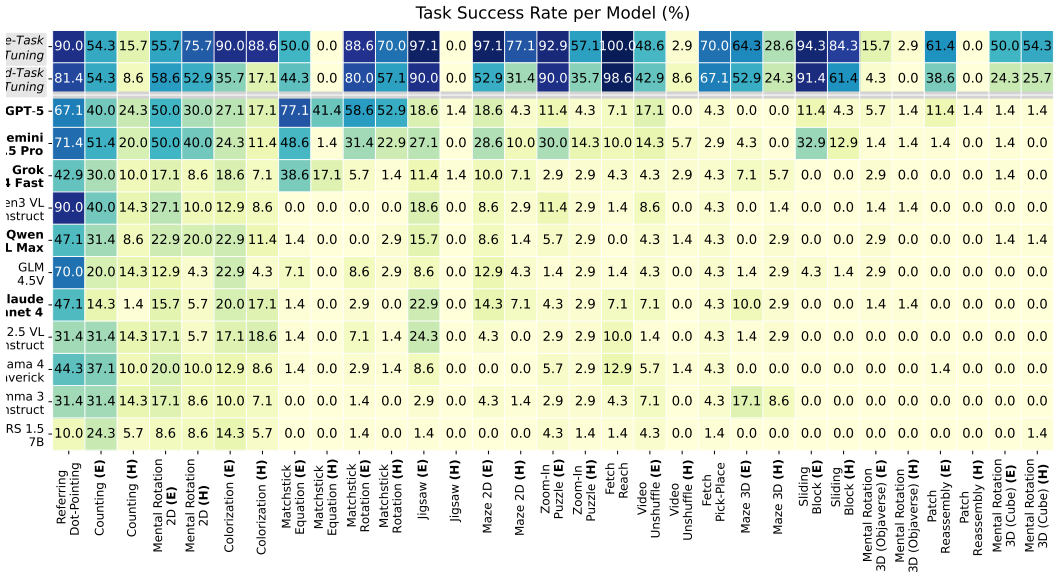


Figure 4: Task success rate of frontier and finetuned models. Proprietary models are shown in **bold**, and our finetuned models in *italics*. *(E)* and *(H)* denote easy and hard task settings. Darker cells indicate higher success rates. Models are ordered by average task performance (top = better), and tasks by average model performance, excluding our finetuned ones (right = harder).

to extract negative behaviors from each trajectory and cluster them into categories observed across datasets. This analysis reveals four failure types that appear consistently across multiple tasks (see Sec. §G for details):

(1) Restricted action space and action looping: models often rely on a single repeated operation or fixed-magnitude action, such as continually moving in the same direction in Fetch Pick & Place, using "swap" in Jigsaw instead of "reorder", or rotating by the same angle in Mental Rotation 3D and Match Rotation rather than converging to an optimal magnitude.

(2) State mismanagement: models fail to maintain or update internal state across steps. They ignore textual or environmental feedback, revisit previously explored areas, or repeat illegal actions despite prior errors—for example, continuing to move into a wall after being told they have collided, or repeating invalid moves in the Match Equation, Sliding Block, and Toy Maze 2D tasks.

(3) Early termination: the model terminates before the maximum steps despite not reaching the goal.

(4) Failure to use visual or spatial information: models ignore the visual information provided, such as the target leaving the frame or the item being successfully aligned (*e.g.*, Mental Rotation).

4 Diagnosing Frontier Models with VisGym

In this section, we show the flexibility of VisGym by presenting controlled diagnoses of how different designs of multi-step interactions can drastically change frontier models' performance and provide our conjectures on why these designs make a difference.

We perform diagnoses with the two best performing proprietary models, *i.e.*, GPT-5 (OpenAI, 2025) and Gemini 2.5 Pro (DeepMind, 2025), and the two best performing open-weight models, *i.e.*, Qwen3-VL-235B Instruct (Yang et al., 2025) and GLM-4.5V (Hong et al., 2025).

4.1 Turns to Keep in Conversation History

Vision–language models are known to degrade with long visual context (Wang et al., 2025; Wu et al., 2024; Sharma et al., 2024). This creates a dilemma: while long histories provide more information about the environment (*e.g.*, 3D layouts, unknown dynamics), they also introduce redundant observations that may harm performance. We study this trade-off in Maze2D, Sliding

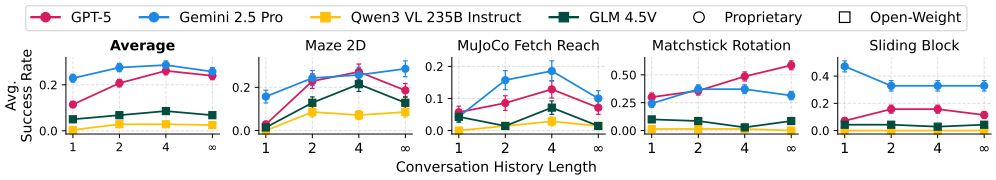


Figure 5: Effect of truncating conversational context on model performance. The settings 1, 2, 4, and ∞ correspond to retaining only the current turn, the current + previous turn, the current + previous 3 turns, and the full history, respectively. Error bars show the standard error of the mean.

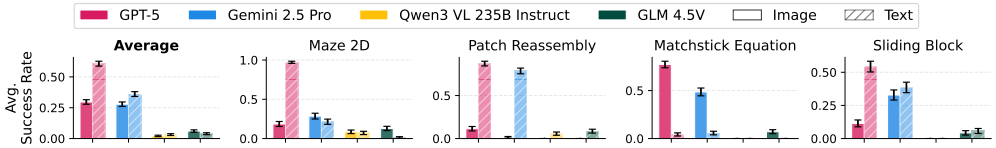


Figure 6: Effect of visualizing observations with ASCII (*text*). "Image" and "Text" denote the observation modalities. Error bars show the standard error of the mean.

Block, MuJoCo Fetch Reach, and Matchstick Rotation, where history provides useful signals such as textual feedback (*e.g.*, invalid actions) or correspondence between action magnitude and perceptual effect, but also introduces stale information.

As shown in Fig. 5, models benefit from including a limited number of previous turns up to roughly four, following a drop when given the full unbounded history. This indicates that expanding visual context helps multi-step visual decision-making only to a point, after which irrelevant or stale observations become detrimental. We also observe task-specific idiosyncrasies: Gemini 2.5 Pro scales well in Maze2D, GPT-5 scales well on Matchstick Rotation, while Sliding Block exhibits clear *reverse scaling* for Gemini 2.5 Pro. These highlight that the value of interaction history is both task-dependent.

4.2 Representing Observation in Text

Inspired by prior work examining how different task representations affect agent performance (Hu et al., 2025; Shi et al., 2025; Ruoss et al., 2024), we select four symbolic tasks–Matchstick Equation, Maze 2D, Patch Reassembly, and Sliding Block–and implement alternate versions rendered entirely in ASCII (sample ASCII visualizations are provided in Sec. §E). This allows tasks to be solved without any visual encoding module.

The results in Fig. 6 show that GPT-5 substantially improves in most tasks, often achieving $3-4\times$ higher success rates than in the visual setting, suggesting that its main bottleneck lies in visual grounding rather than long-horizon reasoning. Gemini 2.5 Pro shows mixed behavior: two tasks do not exhibit significant performance change, one task improves, and one task degrades, indicating possible limitations in both perception and planning. Open-weight models struggle across all tasks in both modalities, indicating general weaknesses in long-horizon decision-making regardless of representation. Interestingly, Matchstick Equation exhibits a *reverse* trend: all models perform substantially better with the visual representation than with ASCII, likely because the figlet-style ASCII has irregular shapes and spacing that create distorted glyphs which models are known to struggle with (Stojanovski et al., 2025).

4.3 Removal of Text-based Feedback

Humans can infer action consequences directly from visual changes (Michotte, 1963), but it remains unclear whether VLMs can do the same. To study this, we select four tasks—Maze 3D, Maze 2D, Sliding Block, and Matchstick Equation—in which the environment feedback f (see Equation (1)) provides not only formatting errors but also constraint violations (*e.g.*, hitting a wall in Maze, sliding a block into an occupied cell). We remove this textual feedback and evaluate model using only visual state transitions; results are shown in Fig. 7.

All models show consistent drops in average performance. This indicates that models struggle to infer action validity directly from visual transitions. These findings show that current VLMs depend

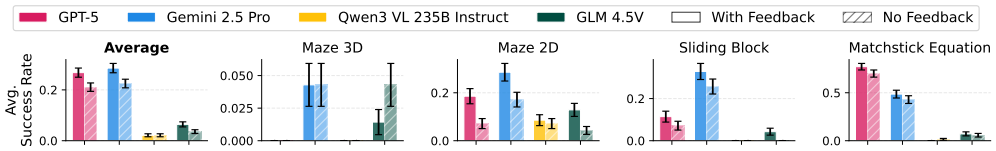


Figure 7: Effect of removing text-based environment feedback. "With Feedback" includes environment feedback describing action execution at each turn; "No Feedback" removes this channel. Error bars show the standard error of the mean.

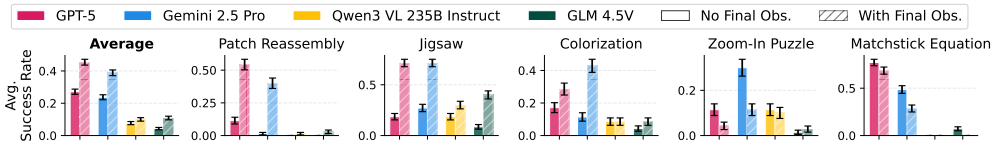


Figure 8: Effect of providing the final goal observation at the beginning of the episode. "No Final Obs." and "With Final Obs." denote settings without and with access to the goal observation at the start. Error bars show the standard error of the mean.

heavily on text-based feedback during visually interactive decision-making and are less sensitive to pure visual feedback.

4.4 Providing Final Goal at Beginning

Providing the solution image upfront simplifies the tasks to visually aligning current observations with a known target, shifting the difficulty from reasoning to visual perception and tool-calling. We test this on five tasks, Patch Reassembly, Jigsaw, Colorization, Zoom-In Puzzle, and Matchstick Equation, where constructing the goal observation involves significant effort. For these tasks, we augment the instruction with the ground-truth final observation o_{gt}, and show results in Fig. 8.

Across tasks, models improve substantially, indicating that a major bottleneck lies in *constructing or imagining the target state*. However, performance remains far from perfect, indicating additional limitations beyond reasoning, such as fine-grained visual perception and action calling. Surprisingly, GPT-5 and Gemini 2.5 Pro *underperform* on the Zoom-In Puzzle and Matchstick Equation when the final goal observation is provided, often terminating early despite visible misalignment. A follow-up test confirms this stems from visual misjudgment due to limited visual perception: we queried Gemini 2.5 Pro on 100 pairs of initial and final-goal observations with the prompt "*Do the two images look exactly the same?*" and it incorrectly judged images as identical 80% and 57% of the time for these tasks, versus only 18%, 2%, and 0% for Colorization, Jigsaw, and Patch Reassembly. This confirms that perception errors can *invert* the expected benefit of an explicit goal observation.

5 Training with VisGym

We describe our supervised fine-tuning experiments with VisGym, present results, and provide insights on generalization, module specificity, and data curation.

5.1 Supervised Fine-Tuning Experiments

Setup. We generate demonstration trajectories for supervised fine-tuning using the multi-step solver described in Sec. §2. We apply two preprocessing filters: (1) discarding trajectories that fail to complete the task, and (2) removing trajectories with initial states overlapping the test split to prevent data leakage.

We evaluate two fine-tuning configurations: *single-task* and *mixed-task*. In the *single-task* setting, we fine-tune a separate model for each task, whereas in the *mixed-task* setting, a single model is trained jointly on all tasks. Notably, demonstrations are sourced exclusively from the easy difficulty level; thus, performance on the hard setting serves as a metric for difficulty generalization. All experiments employ Qwen2.5-VL-7B-Instruct (Bai et al., 2025) with full-parameter fine-tuning, a global batch size of 64, a learning rate of 1×10^{-5}, and `bf16` precision. Models are trained for $1,500$ steps in

the single-task setting and $5,000$ steps in the mixed-task setting. We utilize LlamaFactory (Zheng et al., 2024) for all data preprocessing and training orchestration.

Results. As shown in Figs. 2 and 4, finetuned models achieve state-of-the-art performance on most tasks, validating both the learnability of our environments and the effectiveness of our multi-step solvers. These gains confirm that current VLMs can substantially benefit from structured, solver-generated demonstrations in visually grounded multi-step settings.

5.2 Stronger Base Model Generalizes Better

Existing work has discussed the limitations of supervised finetuning (Ross et al., 2011) and found that it exhibits limited generalization to task variants (Caccia et al., 2024; Deng et al., 2023; Jang et al., 2022). This motivates re-examining generalization in the context of modern VLMs, whose capabilities may shift the boundary of what supervised finetuning can or cannot retain.

To this end, we select a set of environments where the easy-to-hard difficulty gap introduces substantial state changes (*e.g.*, more views in the Zoom-In Puzzle, more patches in Patch Reassembly, larger maze sizes; details in Sec. §F). We finetune Qwen2.5-VL-7B-Instruct and Qwen3-VL-8B-Instruct on the same mixed-task training data using identical optimization hyperparameters (see Sec. §5.1), and report performance in Fig. 9.

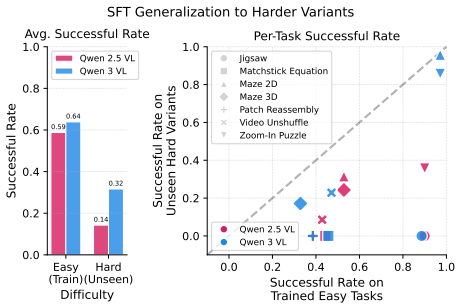


Figure 9: Generalization to unseen difficulty from mixed-task supervised finetuning. *(Left)* Avg. success in easy (seen) vs. hard (unseen). *(Right)* Per-task easy vs. hard success; x = easy, y = hard.

As shown, both models achieve comparable performance on the easy variants they were trained on (*e.g.*, 0.59 vs. 0.64), but the more recent Qwen3-VL generalizes substantially better to the harder variants, nearly doubling the success rate on average relative to Qwen2.5-VL. This trend highlights that newer VLMs provide stronger out-of-distribution generalization in multi-step visual decision-making despite being finetuned on an identical setup.

5.3 Vision and LLM Both Matter

Classic perception–action theories emphasize that fine-grained visual encoding and temporal integration are jointly necessary for interactive behavior (Gibson, 1979). We examine whether this holds for VLMs by fine-tuning variants that modify either the vision encoder or the LLM backbone to isolate each module's contribution, where the vision encoder provides fine-grained perceptual features and the LLM performs temporal integration across steps.

As shown in Fig. 10, most tasks benefit from finetuning both components, with the LLM contributing the larger performance gain—particularly in tasks with partial observability or unknown environment dynamics. This highlights that temporal reasoning and history integration remain the primary bottlenecks for current VLMs, while strong fine-grained visual encoding is necessary (*e.g.,* Zoom-In Puzzle primarily benefits from vision finetuning) but often not sufficient for multi-step decision-making.

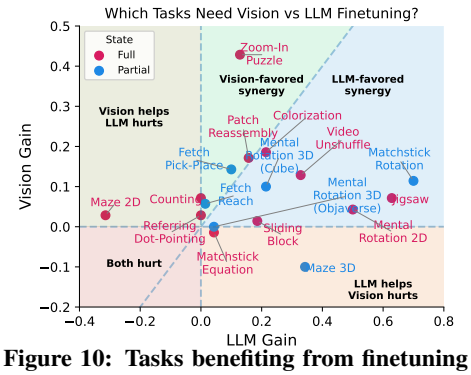


Figure 10: Tasks benefiting from finetuning different modules. "Vision Gain" and "LLM Gain" denote improvements from jointly finetuning both components, compared to finetuning only the *LLM* or the *vision* part. The dashed line ($y = x$) divides vision-favored (above) and LLM-favored (below) synergy. "Full" and "Partial" denote whether observability and dynamics are fully known.

5.4 Importance of Information-Revealing Behaviors for SFT Curation

Not all experiences contribute equally to decision-making: trajectories that reveal hidden states or disambiguate perceptual aliasing are often far more valuable (McCallum, 1994; Fujii et al., 1998).

(a) Environment dynamics are unknown. **(b) Environment is partially observable.**

Figure 11: Effect of data curation strategies on task performance. (a) Numbers represent average task success (higher is better); "3 Moves" and "2 Unit Moves + 1 Move" are two curation strategies. **(b)** "Solve-Only" and "Rotate-Then-Solve" are two curation strategies, and "Continued Training on Solve-Only" denotes further finetuning on Solve-Only after training on Rotate-Then-Solve. *(Left)* Final angular error on the test set (lower is better). *(Right)* Average task success rate (higher is better).

We ask whether inducing such information-revealing behaviors during supervised finetuning helps VLMs form more accurate state representations. We evaluate this on two tasks, Matchstick Rotation (unknown dynamics) and Mental Rotation 3D Objaverse (partial observability), with results in Figs. 11a and 11b.

In Matchstick Rotation, the baseline demonstrations perform three stochastic moves toward the target. In contrast, the information-revealing demonstrations first perform two unit-scale steps to expose the correspondence between action magnitude and perceptual effect before executing the final aligning move. This structured exploration raises success from 32.9% to 70.0%.

In Mental Rotation, the baseline trajectories rotate along each principal axis once to reach the goal, while the information-revealing ones deliberately fully rotate along each axis to expose the full 3D geometry before settling on the target orientation. This strategy improves performance in both metrics. To verify that gains are not simply due to longer trajectories, we further continue training on baseline demonstrations starting from the model already finetuned with information-revealing data. Performance deteriorates in this setting, confirming that the observed improvements stem from the *informative structure* of the demonstrations rather than quantity or length. These results highlight that SFT effectiveness depends on whether demonstrations induce state-disambiguating behaviors, not merely on the number of examples.

5.5 Generalization Across Tasks and Other Domains

While Sec. §5.2 shows that stronger models generalize across difficulty levels within the same task, we now evaluate whether the visual interaction priors learned from VisGym-generated episodes transfer to unseen tasks and to new domains that rely on such capabilities, including vision-language-action (VLA) models (Kim et al., 2024) for robotics.

We first study cross-task transfer via mixed-task fine-tuning (*e.g.*,, Sec. §5.1) on 16 environments, leaving one environment held out for zero-shot evaluation (e.g., Referring Dot-Pointing or Video Unshuffle). Results are reported in Tab. 3. We observe substantial performance gains on both held-out tasks compared to the base model (e.g., Qwen2.5-VL-7B-Instruct), indicating that visual interaction capabilities learned in VisGym feasibly generalize across tasks.

Table 3: Cross-task zero-shot transfer.

	RefDot	Unshuffle
Base.	20.0	4.3
Ours	**28.6**	**15.7**

Next, we examine whether these learned capabilities benefit domain-specific adaptation. We first perform mixed-task fine-tuning on all 17 VisGym environments for 500 steps. Starting from this VisGym-trained VLM, we then follow recent VLM-to-VLA finetuning protocols (Zhang et al., 2026) and train a VLA model on the CALVIN (ABC-D) dataset (Mees et al., 2022) for 16k steps. We compare against an identical VLA training procedure initialized from the base VLM (e.g., Qwen2.5-VL-7B-Instruct). As shown in Tab. 4, VLA models initialized from VisGym-trained VLM consistently outperform the baseline across all metrics, demonstrating that visual interaction priors learned via VisGym transfer effectively to downstream robotic control tasks.

Table 4: CALVIN performance (Mees et al., 2022) by number of instructions chained.

	1	2	3	4	5	Total
Base.	0.817	0.639	0.501	0.394	0.312	2.663
Ours	**0.854**	**0.679**	**0.542**	**0.411**	**0.331**	**2.817**

Together, these results suggest that VisGym provides a principled and scalable framework for instilling transferable visual interaction capabilities, enabling models to generalize beyond individual tasks and domains, and serving as a strong foundation for training interactive multimodal agents.

REFERENCES

[1] Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*, 2023.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Yoshua Bengio, Bernhard Schölkopf, Manuel Wüthrich, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35: 23716–23736, 2022.

[5] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[6] John C. Baird. *Psychophysical Analysis of Visual Space*. International Series of Monographs in Experimental Psychology. Pergamon Press / Elsevier, 1970. ISBN 978-0-08-013876-3. URL https://www.sciencedirect.com/book/9780080138763/psychophysical-analysis-of-visual-space.

[7] Mohamad Ballout, Okajevo Wilfred, Seyedalireza Yaghoubi, Nohayr Muhammad Abdelmoneim, Julius Mayer, and Elia Bruni. Can you splice it together? a human curated benchmark for probing visual reasoning in vlms. *arXiv preprint arXiv:2509.24640*, 2025.

[8] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, jun 2013.

[10] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[11] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*, 2024.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[13] Massimo Caccia, Megh Thakkar, Léo Boisvert, Thibault Le Sellier de Chezelles, Alexandre Piché, Nicolas Chapados, Alexandre Drouin, Maxime Gasse, and Alexandre Lacoste. Fine-tuning web agents: It works, but it's trickier than you think. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024. URL https://openreview.net/forum?id=SkwtxEkst2.

[14] Guanqun Cao, Ryan Mckenna, Erich Graf, and John Oyekan. Learn from the past: Language-conditioned object rearrangement with large language models. *arXiv preprint arXiv:2501.18516*, 2025.

[15] Liang Chen, Hongcheng Gao, Tianyu Liu, Zhiqi Huang, Flood Sung, Xinyu Zhou, Yuxin Wu, and Baobao Chang. G1: Bootstrapping perception and reasoning abilities of vision-language model via reinforcement learning. *arXiv preprint arXiv:2505.13426*, 2025.

[16] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024.

[17] Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. Lota-bench: Benchmarking language-oriented task planners for embodied agents. *arXiv preprint arXiv:2402.08178*, 2024.

[18] João Coelho, Jingjie Ning, Jingyuan He, Kangrui Mao, Abhijay Paladugu, Pranav Setlur, Jiahe Jin, Jamie Callan, João Magalhães, Bruno Martins, et al. Deepresearchgym: A free, transparent, and reproducible evaluation sandbox for deep research. *arXiv preprint arXiv:2505.19253*, 2025.

[19] Lynn A. Cooper and Roger N. Shepard. Chronometric studies of the rotation of mental images. In WILLIAM G. CHASE, editor, *Visual Information Processing*, pages 75–176. Academic Press, 1973. ISBN 978-0-12-170150-5. doi: https://doi.org/10.1016/B978-0-12-170150-5.50009-3. URL https://www.sciencedirect.com/science/article/pii/B9780121701505500093.

[20] Google DeepMind. Gemini 2.5 Pro Technical Report. Technical report, Google DeepMind, 2025. URL https://modelcards.withgoogle.com/assets/documents/gemini-2.5-pro.pdf. Accessed: 2025-11-06.

[21] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023.

[22] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[23] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

[24] Lisa Dunlap, Krishna Mandal, Trevor Darrell, Jacob Steinhardt, and Joseph E Gonzalez. Vibecheck: Discover and quantify qualitative differences in large language models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://lisadunlap.github.io/VibeCheck/.

[25] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4497–4506, 2021.

[26] Atsushi Fujii, Kentaro Inui, Takenobu Tokunaga, and Hozumi Tanaka. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4):573–597, 1998. URL https://aclanthology.org/J98-4002/.

[27] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, 1979. ISBN 0-395-27049-9.

[28] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[29] Solomon W. Golomb. *Polyominoes: Puzzles, Patterns, Problems, and Packings*. Princeton University Press, Princeton, NJ, 2nd edition, 1994. ISBN 978-0-691-02444-8.

[30] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *Proceedings of the IEEE international conference on computer vision*, pages 5842–5850, 2017.

[31] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5356–5364, 2019.

[32] John M. Henderson. A sensorimotor account of vision and visual consciousness. *The Behavioral and brain sciences*, 24 5:939–73; discussion 973–1031, 2001. URL https://api.semanticscholar.org/CorpusID:22606536.

[33] Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv e-prints*, pages arXiv–2507, 2025.

[34] Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are llms at playing games? *arXiv preprint arXiv:2505.15146*, 2025.

[35] Huang Huang, Fangchen Liu, Letian Fu, Tingfan Wu, Mustafa Mukadam, Jitendra Malik, Ken Goldberg, and Pieter Abbeel. Otter: A vision-language-action model with text-aware feature extraciton. *arXiv preprint arXiv:2503.03734*, 2025.

[36] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2): 3019–3026, 2020.

[37] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[38] Lawrence Jang, Yinheng Li, Dan Zhao, Charles Ding, Justin Lin, Paul Pu Liang, Rogerio Bonatti, and Kazuhito Koishida. Videowebarena: Evaluating long context multimodal agents with video understanding web tasks. *arXiv preprint arXiv:2410.19100*, 2024.

[39] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[40] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. ReferItGame: Referring to objects in photographs of natural scenes. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 787–798, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1086. URL https://aclanthology.org/D14-1086/.

[41] Mukul Khanna, Ram Ramrakhya, Gunjan Chhablani, Sriram Yenamandra, Theophile Gervet, Matthew Chang, Zsolt Kira, Devendra Singh Chaplot, Dhruv Batra, and Roozbeh Mottaghi. Goat-bench: A benchmark for multi-modal lifelong navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16373–16383, 2024.

[42] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[43] Günther Knoblich, Stellan Ohlsson, Hilde Haider, and Detlef Rhenius. Constraint relaxation and chunk decomposition in insight problem solving. *Journal of Experimental Psychology: Learning Memory and Cognition*, 25(6):1534–1555, November 1999. ISSN 0278-7393. doi: 10.1037/0278-7393.25.6.1534.

[44] Anurag Koul. maze-world: Random maze environments with different size and complexity for reinforcement learning and planning research. https://koulanurag.dev/maze-world/index.html, 2024. Accessed: 2025-11-07.

[45] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott, and Anthony Tang. Fluid integration of rotation and translation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, page 601–610, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1581139985. doi: 10.1145/1054972.1055055. URL https://doi.org/10.1145/1054972.1055055.

[46] Bohao Li, Yuying Ge, Yixiao Ge, Guangzhi Wang, Rui Wang, Ruimao Zhang, and Ying Shan. Seed-bench: Benchmarking multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13299–13308, 2024.

[47] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024.

[48] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR, 2022.

[49] Lisa Dunlap, Trevor Darrell, Jacob Steinhardt, and Joseph Gonzalez. Stringsight: Automating analysis of model outputs. https://www.stringsight.com/, 2025.

[50] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

[51] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[52] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[53] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*, 2023.

[54] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024.

[55] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer, 2024.

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

[57] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[58] R. Andrew McCallum. Instance-based state identification for reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994. URL https://proceedings.neurips.cc/paper_files/paper/1994/file/d2ed45a52bc0edfa11c2064e9edee8bf-Paper.pdf.

[59] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[60] Albert Michotte. *The Perception of Causality*. Routledge, 1 edition, 1963. doi: 10.4324/9781315519050. URL https://doi.org/10.4324/9781315519050.

[61] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European conference on computer vision*, pages 527–544. Springer, 2016.

[62] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[63] Dantong Niu, Yuvan Sharma, Giscard Biamby, Jerome Quenum, Yutong Bai, Baifeng Shi, Trevor Darrell, and Roei Herzig. Llarva: Vision-action instruction tuning enhances robot learning. *arXiv preprint arXiv:2406.11815*, 2024.

[64] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[65] OpenAI. GPT-5 System Card. System card, OpenAI, August 2025. URL https://cdn.openai.com/gpt-5-system-card.pdf. Accessed: 2025-11-06.

[66] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[67] Santhosh Kumar Ramakrishnan, Erik Wijmans, Philipp Kraehenbuehl, and Vladlen Koltun. Does spatial cognition emerge in frontier models? *arXiv preprint arXiv:2410.06468*, 2024.

[68] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[69] Anian Ruoss, Fabio Pardo, Harris Chan, Bonnie Li, Volodymyr Mnih, and Tim Genewein. Lmact: A benchmark for in-context imitation learning with long multimodal demonstrations. *arXiv preprint arXiv:2412.01441*, 2024.

[70] Aditya Sharma, Michael Saxon, and William Yang Wang. Losing visual needles in image haystacks: Vision language models are easily distracted in short and long contexts. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5429–5451, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.312. URL https://aclanthology.org/2024.findings-emnlp.312/.

[71] R. N. Shepard and J. Metzler. Mental rotation of three-dimensional objects. *Science*, 171 (3972):701–703, 1971. doi: 10.1126/science.171.3972.701.

[72] Jiajun Shi, Jian Yang, Jiaheng Liu, Xingyuan Bu, Jiangjie Chen, Junting Zhou, Kaijing Ma, Zhoufutu Wen, Bingli Wang, Yancheng He, et al. Korgym: A dynamic game platform for llm reasoning evaluation. *arXiv preprint arXiv:2505.14552*, 2025.

[73] Modi Shi, Li Chen, Jin Chen, Yuxiang Lu, Chiming Liu, Guanghui Ren, Ping Luo, Di Huang, Maoqing Yao, and Hongyang Li. Is diversity all you need for scalable robotic manipulation? *arXiv preprint arXiv:2507.06219*, 2025.

[74] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[75] Ved Sirdeshmukh, Kaustubh Deshpande, Johannes Mols, Lifeng Jin, Ed-Yeremai Cardona, Dean Lee, Jeremy Kritz, Willow Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. *arXiv preprint arXiv:2501.17399*, 2025.

[76] Ruben Spaans. Solving sliding-block puzzles. Specialisation project report, Norwegian University of Science and Technology (NTNU), 2009. URL https://www.pvv.org/~spaans/spec-cs.pdf.

[77] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on robot learning*, pages 477–490. PMLR, 2022.

[78] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

[79] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266, 2021.

[80] Claude Team. Claude 4 sonnet, 2025. URL https://www.anthropic.com/claude/sonnet.

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

[82] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[83] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[84] Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*, 2025.

[85] TRI LBM Team, Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, Naveen Kuppuswamy, Kuan-Hui Lee, Katherine Liu, Dale McConachie, Ian McMahon, Haruki Nishimura, Calder Phillips-Grafflin, Charles Richter, Paarth Shah, Krishnan Srinivasan, Blake Wulfe, Chen Xu, Mengchao Zhang, Alex Alspach, Maya Angeles, Kushal Arora, Vitor Campagnolo Guizilini, Alejandro Castro, Dian Chen, Ting-Sheng Chu, Sam Creasey, Sean Curtis, Richard Denitto, Emma Dixon, Eric Dusel, Matthew Ferreira, Aimee Goncalves, Grant Gould, Damrong Guoy, Swati Gupta, Xuchen Han, Kyle Hatch, Brendan Hathaway, Allison Henry, Hillel Hochsztein, Phoebe Horgan, Shun Iwase, Donovon Jackson, Siddharth Karamcheti, Sedrick Keh, Joseph Masterjohn, Jean Mercat, Patrick Miller, Paul Mitiguy, Tony Nguyen, Jeremy Nimmer, Yuki Noguchi, Reko Ong, Aykut Onol, Owen Pfannenstiehl, Richard Poyner, Leticia Priebe Mendes Rocha, Gordon Richardson, Christopher Rodriguez, Derick Seale, Michael Sherman, Mariah Smith-Jones, David Tago, Pavel Tokmakov, Matthew Tran, Basile Van Hoorick, Igor Vasiljevic, Sergey Zakharov, Mark Zolotas, Rares Ambrus,

Kerri Fetzer-Borelli, Benjamin Burchfiel, Hadas Kress-Gazit, Siyuan Feng, Stacie Ford, and Russ Tedrake. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025. URL https://arxiv.org/abs/2507.05331.

[86] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[87] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[88] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[89] Hengyi Wang, Haizhou Shi, Shiwei Tan, Weiyi Qin, Wenyuan Wang, Tunyu Zhang, Akshay Nambi, Tanuja Ganu, and Hao Wang. Multimodal needle in a haystack: Benchmarking long-context capability of multimodal large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3221–3241, 2025.

[90] Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, Qineng Wang, Hanyang Chen, Chi Wan, Yiping Lu, Zhengyuan Yang, et al. Vagen: Reinforcing world model reasoning for multi-turn vlm agents. *arXiv preprint arXiv:2510.16907*, 2025.

[91] Zifu Wang, Junyi Zhu, Bo Tang, Zhiyu Li, Feiyu Xiong, Jiaqian Yu, and Matthew B Blaschko. Jigsaw-r1: A study of rule-based visual reinforcement learning with jigsaw puzzles. *arXiv preprint arXiv:2505.23590*, 2025.

[92] Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Advances in Neural Information Processing Systems*, 37: 113569–113697, 2024.

[93] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[94] Tsung-Han Wu, Giscard Biamby, Jerome Quenum, Ritwik Gupta, Joseph E Gonzalez, Trevor Darrell, and David M Chan. Visual haystacks: A vision-centric needle-in-a-haystack benchmark. *arXiv preprint arXiv:2407.13766*, 2024.

[95] xAI. Grok 4 Model Card. Model card, xAI, August 2025. URL https://data.x.ai/2025-08-20-grok-4-model-card.pdf. Accessed: 2025-11-06.

[96] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[97] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[98] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

[99] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[100] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567, 2024.

[101] Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15134–15186, 2025.

[102] Alex L Zhang, Thomas L Griffiths, Karthik R Narasimhan, and Ofir Press. Videogamebench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025.

[103] Jianke Zhang, Xiaoyu Chen, Qiuyue Wang, Mingsheng Li, Yanjiang Guo, Yucheng Hu, Jiajun Zhang, Shuai Bai, Junyang Lin, and Jianyu Chen. Vlm4vla: Revisiting vision-language-models in vision-language-action models. *arXiv preprint arXiv:2601.03309*, 2026.

[104] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[105] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, et al. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11142–11152, 2025.

[106] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

[107] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

[108] Zhongyi Zhou, Yichen Zhu, Junjie Wen, Chaomin Shen, and Yi Xu. Vision-language-action model with open-world embodied reasoning from pretrained knowledge. *arXiv preprint arXiv:2505.21906*, 2025.

[109] Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

TABLE OF CONTENTS

1 Introduction **1**

2 VISGYM **3**

3 Evaluating Frontier Models with VISGYM **3**
- 3.1 Evaluation Setup 4
- 3.2 Result and Analysis 4

4 Diagnosing Frontier Models with VISGYM **5**
- 4.1 Turns to Keep in Conversation History 5
- 4.2 Representing Observation in Text 6
- 4.3 Removal of Text-based Feedback 6
- 4.4 Providing Final Goal at Beginning 7

5 Training with VISGYM **7**
- 5.1 Supervised Fine-Tuning Experiments 7
- 5.2 Stronger Base Model Generalizes Better 8
- 5.3 Vision and LLM Both Matter 8
- 5.4 Importance of Information-Revealing Behaviors for SFT Curation 8
- 5.5 Generalization Across Tasks and Other Domains 9

A Related Work **2**

B Solver Design **3**

C Environment Episode Progression **5**

D VISGYM Interface **5**

E ASCII-based Observation Visualization **6**

F Configuration of Environments **7**

G Analyzing Model Failures **8**

H Additional Performance Analysis **11**

A Related Work

The development of foundation models (Achiam et al., 2023; Team et al., 2023; Team, 2025; Dubey et al., 2024; Yang et al., 2025), particularly vision-language models (VLMs) (Hong et al., 2025; Bai et al., 2025; Team et al., 2025; Zhu et al., 2025; Team et al., 2025; Liu et al., 2023; Alayrac et al., 2022; Li et al., 2022) and vision-language-action models (VLAs) (Black et al., 2024; Bjorck et al., 2025; Team et al., 2025; Kim et al., 2024; Octo Model Team et al., 2024; Huang et al., 2025; Team et al., 2025; Zhou et al., 2025; Niu et al., 2024; Shi et al., 2025), has reshaped how AI agents perceive, make decisions, and act across physical and simulated environments. To properly assess the capabilities and limitations of the models, a plethora of benchmarks have been developed.

Early benchmarks such as Atari, OpenAI Gym, and DeepMind Lab (Mnih et al., 2013; Brockman et al., 2016; Towers et al., 2024; Beattie et al., 2016) were developed to evaluate vision-based control and decision-making in fully observable environments. These platforms laid the groundwork for reinforcement learning but focused primarily on low-level motor control. Subsequent efforts extended these ideas to robotic manipulation and navigation, introducing partially observable, multi-task, and long-horizon settings that better reflect real-world complexity (Yu et al., 2020; Ahmed et al., 2020; Shridhar et al., 2020; Li et al., 2024; Cao et al., 2025; Liu et al., 2023; Khanna et al., 2024; Choi et al., 2024; Yang et al., 2025; Ehsani et al., 2021; Szot et al., 2021; Srivastava et al., 2022; Mees et al., 2022; Mandlekar et al., 2021; James et al., 2020). These modern suites enable training and evaluation of multi-task imitation learning and meta-learning policies across diverse embodiment and task horizons.

Concurrently, many VLM benchmarks have been developed to probe models' cognitive and perceptual limits. Early efforts focused on visual question answering—first as multiple-choice tasks and later as open-ended reasoning (Yue et al., 2024; 2025; Liu et al., 2024; Li et al., 2024; Chen et al., 2024; Ballout et al., 2025). As visual grounding and reasoning improved, newer benchmarks began representing actions through text instead of fixed, predefined action spaces, enabling studies of the interplay between perception, reasoning, and control (Wang et al., 2025; Jang et al., 2024; Liu et al., 2023; Shi et al., 2025; Stojanovski et al., 2025; Abdulhai et al., 2023; Coelho et al., 2025). For example, G1 (Chen et al., 2025) introduces VLM-Gym, a suite of visual game environments with unified interfaces and adjustable difficulty. Broader evaluation suites such as VisualAgentBench (Liu et al., 2024), EmbodiedBench (Yang et al., 2025), and WebArena (Zhou et al., 2023) aggregate tasks across embodied control, graphical interfaces, and visual reasoning, challenging agents with multi-step planning and tool use.

VisGym unifies reasoning and control under an RL-style "gym" paradigm, combining 17 multimodal tasks spanning visual puzzles, spatial reasoning, manipulation, and grounding. Each environment includes an oracle solution to ensure solvability and allow synthetic trajectory generation for post-training. Moreover, VisGym introduces controllable difficulty along with targeted diagnostics—such as history utilization, representation variants, feedback specificity, and perception–action causality—allowing researchers to examine not only whether models fail but also what causes failures. We hope these designs enable more systematic analysis of VLMs and VLAs across domains and levels of interactivity (Tab. 1).

B SOLVER DESIGN

This section provides detailed descriptions of the multi-step solvers introduced in Sec. §2 and used for supervised finetuning across all environments.

Colorization. The solver computes how far the current hue and saturation are from the target, breaks those differences into small incremental steps, and outputs a sequence of rotate and saturate actions that move steadily toward the correct color. If a target number of steps is requested or if the color is already close enough, it fills the sequence with reversible rotate/saturate pairs that cancel out and don't change the final state.

Counting. *mark_all strategy:* The solver places a dot at the center of each target instance, then submits the correct total count and stops. *guess_only strategy:* The solver directly submits the correct total count and stops, without placing any dots.

Jigsaw. *reorder strategy:* The solver computes a single permutation payload that, when applied via the reorder' action, instantly rearranges the current pieces into their correct target positions. *swap strategy:* The solver generates a minimal sequence of swap' actions by repeatedly finding a misplaced piece and swapping it with the piece at its correct target location. If a target number of steps is requested, it pads this sequence with reversible pairs of swaps (*e.g.*, swapping two pieces and then immediately swapping them back) until the desired length is reached.

Matchstick Equation. *bfs strategy:* The solver finds the shortest possible sequence of move' actions to correct the equation using a Breadth-First Search (BFS) and then stops. *dfs strategy:* The solver finds a solution using a Depth-First Search (DFS), producing a sequence of move' actions and undo' actions that represent its full exploratory and backtracking process before stopping. *sos strategy:* The solver first finds the shortest solution path (via BFS), then pads this path by inserting random, reversible detours. Before an optimal step, it takes one or more random move' actions and immediately undo'es them, returning to the optimal path before proceeding.

Matchstick Rotation. The solver first performs one or more translation-only move' actions, which are typically unit-length moves in the general direction of the target. It then executes a final move' action that applies the entire required rotation and corrects any remaining translation error, before stopping.

Maze 2D. The solver uses a graph search algorithm to find the optimal coordinate path from the agent to the target, which is converted into the shortest sequence of move' actions. If a target number of steps is requested, the solver pads this optimal sequence by inserting random, reversible move' pairs (*e.g.*, move up' followed by move down') at valid locations along the path until the desired length is met, before stopping.

Maze 3D. The solver uses a graph search algorithm to find the optimal coordinate path from the agent's location to the target. It then converts this path into the shortest sequence of turn' (left, right, or around) and move' actions required to follow that path, accounting for the agent's current orientation. If a target number of steps is requested, the solver pads this optimal sequence by inserting random, reversible turn' pairs (*e.g.*, turn left' followed by turn right') at locations along the path until the desired length is met, before stopping.

Mental Rotation 2D. The solver first calculates the shortest total rotation angle required to align the current image with the target. If the requested number of steps is 1, it outputs a single rotate' action for that total angle. If a larger number of steps is requested, it stochastically divides the total rotation into that many smaller rotate' actions, which are executed sequentially and sum to the correct total angle, before stopping.

Mental Rotation 3D (CUBE). The solver decomposes the total required rotation into its yaw, pitch, and roll components. It then corrects each component sequentially. Before applying the corrective 'rotate' action for a specific axis (*e.g.*, yaw), it first executes a padding sequence of four 90-degree rotations around that same axis. After this 360-degree padding, it applies the single action to correct the yaw. It repeats this pad-then-correct process for the pitch and roll axes, then stops.

Mental Rotation 3D (OBJAVERSE). The same as Mental Rotation 3D (CUBE).

MuJoCo Fetch (PICK-AND-PLACE). The solver is a state-machine-based oracle. It follows a sequence: (0) move the gripper to a safe height above the object, (1) open the gripper, (2) descend to the object, (3) close the gripper to grasp. (4) Once grasped, it moves the object directly toward the 3D goal position using a greedy, per-axis strategy (correcting the axis with the largest error at each step). (7) Finally, it holds the object at the target location and stops.

MuJoCo Fetch (REACH). The solver is a greedy, per-axis oracle. At each step, it identifies the single axis (x, y, or z) with the largest error between the gripper and the goal. It then outputs a 'move' action along that single axis to reduce the error, repeating this process until the goal is reached, at which point it stops.

Patch Reassembly. The solver uses a backtracking search to find the optimal sequence of 'place' actions that perfectly tile the grid. If a target number of steps is requested, it pads this sequence by repeatedly inserting "mistake-and-correct" actions: it finds a correct 'place' action in the solution, finds a valid wrong location for that piece, and inserts this "mistake" 'place' action immediately before the "correct" 'place' action. If no valid mistakes can be found, it falls back to inserting a 'remove' and a duplicate 'place' action. This repeats until the desired number of 'place' actions is met.

Referring Dot-Pointing. The solver first samples a random pixel from within the target object's segmentation mask and also calculates the mask's center of mass. It then generates a sequence of 'mark' actions by linearly interpolating from the random starting point to the center of mass over the requested number of steps. The final action in this sequence places a mark at the exact center of mass, which is then followed by a 'stop' action.

Sliding Block. The solver uses a Breadth-First Search (BFS) to find the shortest sequence of 'move' actions from the current board state to the target configuration. If a target number of steps is requested, it pads this optimal path by first reconstructing all intermediate board states. At each state, it identifies all valid "back-and-forth" moves (*e.g.*, move block 1 right, then move block 1 left). It then randomly samples from these opportunities and inserts the required number of 'move' and 'reverse-move' pairs into the solution path until the desired length is met, before stopping.

Video Unshuffle. *reorder strategy:* The solver computes a single permutation payload that, when applied via the 'reorder' action, instantly rearranges the shuffled frames into their correct chronological order, then stops. *swap strategy:* The solver generates a minimal sequence of 'swap' actions to sort the frames. It iterates through the positions, and if a frame is in the wrong place, it finds the correct frame and swaps it into its target position, repeating until all frames are sorted, then stops.

Zoom-In Puzzle. The same as Video Unshuffle.

C ENVIRONMENT EPISODE PROGRESSION

Referenced in Tab. 2 and Sec. §2, this section presents detailed episode progressions for each environment. A summary index with page numbers is provided below:

Colorization B.2

Counting B.4

Jigsaw B.6

Matchstick Equation B.8

Matchstick Rotation B.10

Maze 2D B.12

Maze 3D B.14

Mental Rotation 2D B.16

Mental Rotation 3D (CUBE) B.18

Mental Rotation 3D (OBJAVERSE) B.20

MuJoCo Fetch PICK-AND-PLACE B.22

MuJoCo Fetch REACH B.24

Patch Reassembly B.26

Referring Dot-Pointing B.28

Sliding Block B.30

Video Unshuffle B.32

Zoom-In Puzzle B.34

D VISGYM INTERFACE

In this section, we present the pseudocode for the `step` function (Algorithm 1) used in VISGYM (*i.e.,* Sec. §2). The function initializes the reward and both termination flags, then parses the model's output string into an action name and payload. If parsing fails, it immediately returns an observation with "invalid format" as feedback.

If the parsed action name is supported and its payload is valid for the corresponding action space, the function calls `Apply`, which executes the action and returns the environment feedback. Otherwise, it ends early with "invalid action" as feedback.

Termination and truncation are determined inside `Apply`. If the action triggers termination (*e.g.*, `stop`), the function computes the final reward based on the environment state. Thus, the returned reward is always zero for non-terminal transitions and the final score upon termination.

Finally, the function returns the new observation, reward, termination, and truncation flags, and the feedback describing the action outcome.

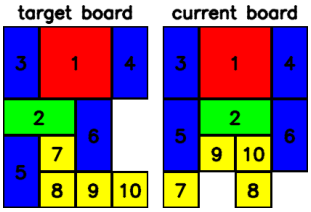
Algorithm 1 Generic Step Function (Sec. §D). Symbols: ρ = reward, τ = terminated, υ = truncated, φ = feedback, α = action name, π = payload, ι = info.

```
function STEP(a)
    ρ ← 0
    (τ, υ) ← (false, false)
    Parse a → (α, π)
    if invalid format then
        return (obs(), 0, τ, υ, ι("invalid format"))
    end if
    if α ∈ A and π ∈ A[α] then
        (φ, τ, υ) ← Apply(α, π)
    else
        return (obs(), 0, τ, υ, ι("invalid action"))
    end if
    if τ = true then
        ρ ← ComputeReward()
    end if
    return (obs(), ρ, τ, υ, ι(φ))
end function
```

E ASCII-BASED OBSERVATION VISUALIZATION

In this section, we present example episode variants rendered in text, as discussed in Sec. §4.2, for the Sliding Block, Maze 2D, Patch Reassembly, and Matchstick Equation environments in Fig. 12.

Note that the instructions are slightly adapted to fit the text-based format (*e.g.*, in the visual version of Patch Reassembly, we describe the *anchor* as "the cell that shows the patch's ID number," while in the text version we note that "the anchor cell for each parked patch is marked with a '*' instead of its ID number").

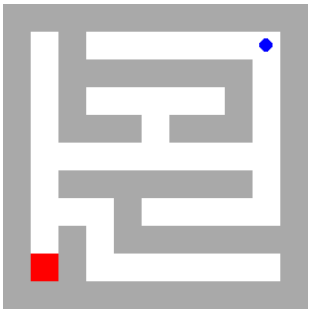


Visual Rendering (default)

```
Target Current
-----------
3114 | 3114
3114 | 3114
226. | 5226
576. | 5906
5890 | 7.8.
```

Text Representation (variant)

Sliding Block

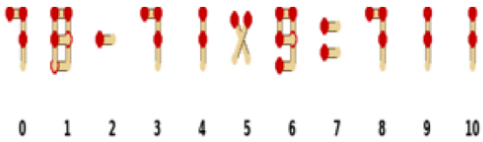


Visual Rendering (default)

```
#############
#############
## #      A##
## ####### ##
## #     # ##
## ### ### ##
##         ##
## ####### ##
##   #     ##
## # ########
##T#        ##
#############
#############
```

Text Representation (variant)

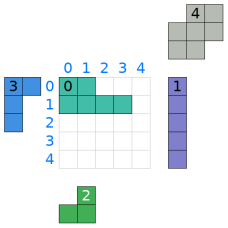
Maze 2D



Visual Rendering (default)

Text Representation (variant)

Matchstick Equation



Visual Rendering (default)

```
   0  1  2  3  4
  ---------------
0 | 0  0  .  .  .
1 | 0  0  0  0  .
2 | .  .  .  .  .
3 | .  .  .  .  .
4 | .  .  .  .  .

--- Parked Patches ---

Patch 1:
  *
  1
  1
  1
  1

Patch 2:
   *
  22

Patch 3:
  *3
  3
  3

Patch 4:
   *4
  444
  44
```

Text Representation (variant)

Patch Reassembly

Figure 12: Visual and text representations across four environments

F CONFIGURATION OF ENVIRONMENTS

In this section, we provide the tunable parameters for each task that determine its difficulty. Note that in some environments, higher difficulty primarily increases the complexity of the input observations (e.g., Sec. §5.2), while in others it tightens the reward criteria and requires more precise control. Details are in Tab. 5.

Table 5: Configuration summary for tasks: tunable parameters, difficulty, and source datasets.

Task	Tunable Difficulty Parameters	Easy	Hard	Src. Dataset
Colorization	Accuracy radius `ar` (precision required for hue and saturation match).	`ar` = 11	`ar` = 16	LLaVA Liu et al. (2023)
Counting	Minimum and maximum count range `c_min`, `c_max`.	`c_min` = 2, `c_max` = 20	`c_min` = 5, `c_max` = 30	LVIS Gupta et al. (2019)
Jigsaw	number of rows and columns `nr`, `nc`.	`nr` = 2, `nc` = 2	`nr` = 3, `nc` = 3	LLaVA Liu et al. (2023)
Matchstick Equation	Number of break moves `bm` (corruptions to fix).	`bm` = 1	`bm` = 2	—
Matchstick Rotation	Hidden scale range `sr`, position tolerance `pt`, angular tolerance `at`.	`pt` = 10, `at` = 15	`pt` = 5, `at` = 10	—
Maze 2D	Maze width and height `mw`, `mh`.	`mw` = 9, `mh` = 9	`mw` = 11, `mh` = 11	—
Maze 3D	Maze width and height `mw`, `mh`.	`mw` = 7, `mh` = 7	`mw` = 9, `mh` = 9	—
Mental Rotation 2D	Angular tolerance `at`.	`at` = 10.0	`at` = 5.0	LLaVA Liu et al. (2023)
Mental Rotation 3D (CUBE)	Number of segments `ns`, length range `lr`, angular tolerance `at`.	`ns` = 4	`ns` = 6	—
Mental Rotation 3D (OBJAVERSE)	Angular tolerance `at`.	`at` = 15.0	`at` = 5.0	Objaverse Deitke et al. (2023)
MuJoCo Fetch PICK-AND-PLACE	No user-tuned difficulty parameters (standardized task).	Standard	Standard	—
MuJoCo Fetch REACH	No user-tuned difficulty parameters (standardized task).	Standard	Standard	—
Patch Reassembly	Grid size `gs`, number of patches `np`.	`gs` = (6, 6), `np` = 5	`gs` = (8, 8), `np` = 6	—
Referring DOT-POINTING	No user-tuned difficulty parameters (standardized task).	Standard	Standard	RefCOCO Kazemzadeh et al. (2014)
Sliding Block	Number of shuffle moves `sm`.	`sm` = 30	`sm` = 90	—
Video Unshuffle	Number of frames `nf`, sampling strategy `ss`, minimum frame-diff threshold `mfd`.	`nf` = 4	`nf` = 5	SS2 Goyal et al. (2017)
Zoom-In Puzzle	Zoom gap `zg`, zoom variability `zs`, minimum zoom `mz`, num. of views `zv`, nested crop `nest`.	`zv` = 4	`zv` = 5	LLaVA Liu et al. (2023)

Table 6: Example clusters discovered by StringSight.

Cluster Description
MuJoCo Fetch (Pick-and-Place)
• The model issues repetitive movement commands without adapting based on task progress or environment feedback, resulting in oscillatory behaviors like moving up and down, left and right, or in a single direction with no meaningful progress toward grasping or placing the cube. For example, it may move forward repeatedly even after overshooting the target or oscillate without ever attempting a grasp.
• Ignores visual feedback from images and does not adjust its actions in response to clear cues about the state or position of the cube, gripper, or target marker. This results in the model issuing irrelevant or counterproductive actions, such as continuing to move when the cube has not been grasped.
• Issues the "stop" command prematurely before the end-effector is even close to the target marker, ending the task early without justification or clear success criteria.
Mental Rotation 3D (Cube)
• The model repeatedly issues the same rotation action, often on a single axis, for many steps without adapting based on visual feedback, resulting in rigid loops or unbroken patterns.
• The model oscillates between a small set of orientations, alternating or repeating similar rotations (e.g., $+45^\circ$, -45°), causing the object to cycle endlessly without making progress toward the target alignment.
Zoom-In Puzzle
• The model finalizes the arrangement immediately without performing any swaps, reordering, or verification, accepting the initial sequence as correct regardless of its accuracy. For example, it issues a "stop" command on the first step even if the order is incorrect.
Matchstick Rotation
• The model issues fixed or monotonically decreasing movement and rotation magnitudes without attempting to estimate the unknown scale or using exploratory actions to resolve scale ambiguity, proceeding as if the appropriate step size is already known.
• Action sequences do not adapt based on feedback or observed outcomes; the model follows a predetermined or repetitive strategy without checking if moves are effective or responding to evidence from the environment.
Maze 2D
• The model repeatedly issues the same invalid movement commands, such as trying to move into walls, even after receiving explicit feedback that these actions are not possible. For example, it continues to try moving left into a wall after being told each time that the move is blocked.
• Does not build, update, or use any internal map or memory of previous moves or environmental feedback, resulting in repeated visits to the same locations, blocked paths, and inefficient looping navigation.
Maze 3D
• Movement decisions are based exclusively on immediate sensory feedback, without building or referencing internal memory or a map of previously explored locations, leading to repeated wall collisions, revisiting dead ends, and inefficient navigation.
• Frequently issues long sequences of consecutive turning actions—such as alternating left and right—without forward movement or meaningful progress toward the goal, resulting in wasted steps and inefficient pathfinding.

G Analyzing Model Failures

We run StringSight Dunlap et al. (2025); Lisa Dunlap et al. (2025), a pipeline for automatically uncovering failure cases and comparing models. It uses a VLM annotator (GPT-4.1) to extract behaviors from each trace (e.g., "uses `move(1,1)` for all 20 steps") and clusters these behaviors into higher-level patterns (e.g., "repeats the same action"). Examples of discovered cluster descriptions are shown in Table 6. We then manually examine the top failure cases for each task and identify four common failure modes across all tasks.

(1) Restricted action space and action looping: models often rely on a single repeated operation or fixed-magnitude action, such as continually moving in the same direction in Fetch Pick & Place, using "swap" in Jigsaw instead of "reorder", or rotating by the same angle in Mental Rotation 3D and Match Rotation rather than converging to an optimal magnitude.

(2) State mismanagement: models fail to maintain or update internal state across steps. They ignore textual or environmental feedback, revisit previously explored areas, or repeat illegal actions despite prior errors—for example, continuing to move into a wall after being told they have collided, or repeating invalid moves in the Match Equation, Sliding Block, and Toy Maze 2D tasks.

System Prompt: Labeling failure modes in traces

You are an expert model behavior analyst. Your task is to meticulously analyze the trace of a large language model to identify whether it contains any of the following behaviors:

- **Restricted action space and action looping**: The model keeps repeating the same or nearly identical action without making progress. Look for consecutive turns with the same command or sequence of commands, movements by the same amount, or the same tool being used even when it is ineffective.
- **State mismanagement**: The model forgets or ignores what it already learned in earlier steps. It may revisit old states, contradict past reasoning, or repeat mistakes it was corrected for (e.g. being told it hit a wall and then continuing to move forward in the same direction). Do not include if this is simply action looping, where the model is repeating the same action without making progress; this is specifically when the model is ignoring feedback or not adjusting its behavior based on its previous actions, but is still issuing different commands.
- **Early termination**: The model stops too early. Early means terminating before the maximum number of steps is reached.
- **Failure to use visual or spatial information**: The model ignores visible or spatial cues. This applies to traces where either an image or ASCII art is provided, and the model does not react to changes in the scene. For example, if the object leaves the frame, but the model continues to move towards it. Look for actions that contradict what's visually or spatially clear. Do not include if the model is simply action looping, where the model is repeating the same action without making progress; this is specifically when the model is not utilizing the visual information when it is available. If the trace does not provide visual information (either images or ASCII), do not include this label.

If the trace contains any of the behaviors, return a list of objects with the following structure. If a trace has more than one behavior, return a list of objects with the structure below for each behavior. If the trace contains none of the behaviors, return an empty list.

JSON Output Structure

```
[
  {
    "property_description": "which behavior is present in the
    trace",
    "reason": "an explanation of the exact behaviors in the
    trace that fall under the property_description
    (1-2 sentences)",
    "evidence": "What exactly in the trace exhibits this
    property? Include quotes/tool calls/actions when possible."
  }
]
```

(3) Early termination: the model terminates the episode before the maximum steps, despite not reaching the goal.

(4) Failure to use visual or spatial information: models ignore the visual information provided, such as the target leaving the frame or the item being successfully aligned (*e.g.*, Mental Rotation).

Finally, we quantify the prevalence of each failure mode by having a VLM annotator (GPT-4.1) label each trace for these behaviors (a trace may exhibit multiple behaviors).

Frequency of failures. Figure 13 shows the proportion of traces that contain each failure. We see that action looping is very common, occurring in more than 60% of traces, followed in frequency by early termination, state mismanagement, and failure to use visual or spatial information. Looking at how the frequency of the behaviors changes compared across tasks, we see in Figure 14 (b) that certain tasks, like Matchstick Equation and Sliding Block, result in a particularly large amount of action repetition and state mismanagement failures, likely due to the difficulty of the task and the frequency of invalid moves. We additionally see that tasks like the Maze task, which provide clear

visual signals of task progress, have a very high (up to 70%) rate of ignoring this important visual information and high action repetition. Based on this information, we see that often when a model is uncertain, it defaults to repeating its previous moves, regardless of the visual or language feedback it is given from previous turns. This is further supported in Figure 14 (a), which shows that weaker models like UI TARS 1.5 7B have very high rates of action looping (87%) and state mismanagement (35%).

We additionally find interesting cases of early termination, such as giving up on the task entirely, where the model says things like "I give up." and "I'm stopping. This is unsolvable". These specific instances of giving up happen much more often for hard tasks like Matchstick Equation, indicating that the models' limited task comprehension leads them to question whether a solution exists in the current instance. We also see this phenomenon occur more often in Gemini and Gemma models, which we suspect is because these models are chattier and more anthropomorphic and thus may express their internal reasoning more often than others.

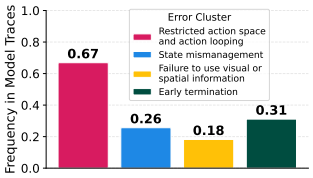
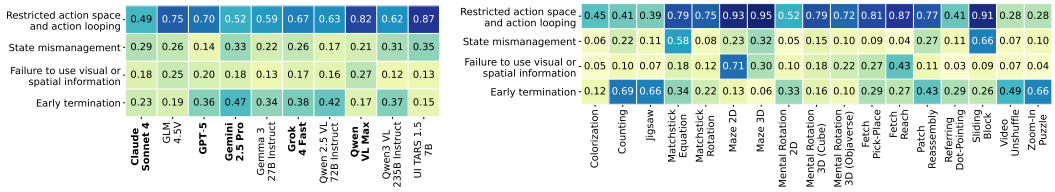


Figure 13: Frequency of failure patterns.



(a) Frequency of failure patterns per model.

(b) Frequency of failure patterns per task on easy variants.

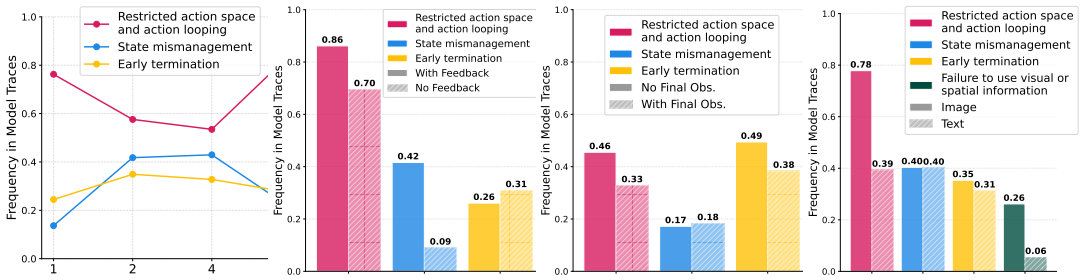
Figure 14: Detailed Analysis of Failure Patterns by Model and Task.

G.1 Failure changes per ablation

To examine the effects of our ablations on model behavior, we run the failure labeling pipeline above on the ablations described in Section 4 and show the comparison in Figure 15. We find the following:

Different amounts of chat history (Figure 15a): As more history is given, the model is less likely to repeat immediate actions, but still suffers from state mismanagement. We suspect the decreased action looping occurs because the model has a default action (e.g., moving left), so with no history, it continues to repeat this move. With history, it is less likely to immediately repeat prior actions, but after a certain amount of context, the model struggles to manage earlier state and reverts to its default behavior. This is reflected by action looping decreasing when full history is given, consistent with decreased performance under full history.

Feedback vs. no feedback (Figure 15b): When no feedback is provided, the model is less likely to terminate. Inspecting these traces shows that this is largely due to a reduction in "giving up," since the model often gives up when told its moves are invalid. We also observe decreases in action looping



(a) Different amounts of chat history.

(b) Feedback vs. no feedback.

(c) Ground-truth obs. at the start.

(d) Text vs. image representation.

Figure 15: Failure-pattern frequency under different information settings. Due to cost, images were only analyzed in the original split, thus the "failure to use visual or spatial information" case is removed in all but the text vs image representation ablation (d).

and state mismanagement, which is surprising given that overall performance decreases without feedback. This suggests the presence of additional failure modes not captured by our taxonomy, which we leave for future work.

Ground truth state given at the beginning (Figure 15c): When given the ground truth state at the start of the task, the model is less likely to "guess" or give up early, reflected in lower rates of action looping and early termination.

Image vs. text representation (Figure 15d): For tasks aside from Matchstick Equation, models process visual information more effectively when it is presented as text rather than an image. The large reduction in action looping suggests that this text-based representation provides clearer guidance for selecting actions.

G.2 Failure Trajectories Visualization

Using StringSight, we visualize the trajectory for each failure type. In each trajectory, we show the prompt, the image, the models' raw output, and the action parsed from the raw output. We also show the output from StringSight for each trajectory, tagged "Reason" and "Evidence" at the top, where "Reason" stands for StringSight's reason for classifying this trajectory into a specific failure category, and "Evidence" stands for the evidence in the trajectory that leads to the conclusion.

(1) Restricted action space and action looping: As in Sec. §F.0.2, we show a case of action looping of GPT-5 on the Jigsaw task. The model repeatedly takes the same action "("swap", (0, 0), (0, 1))", resulting in looping behaviors without making any progress.

(2) State mismanagement: As in Sec. §F.0.4, we show a case of Claude Sonnet 4 on Maze 2D. In Observation 7, the model takes action "("move", 2)", which leads to the environment feedback "Cannot move into a wall." However, at Observation 16, the model is in the exact same state, disregards the previous feedback, and takes the same action "("move", 2)" again.

(3) Early termination: We show in Sec. §F.0.6 a case of Gemma 3 27B Instruct on Matchstick Equation, where the model decides to give up and terminate at step 13, while the model is allowed to take 30 steps in total.

(4) Failure to use visual or spatial information: As in Sec. §F.0.8, we show a case of Gemini 2.5 Pro on Mental Rotation 3D (Cube). In the last three steps, after rotating in the wrong direction, the model does not take the visual information into account and continues to rotate in the same direction, which moves the object even farther away from the target position.

H Additional Performance Analysis

Difficulty of Each Task. In Fig. 16, we compute the average accuracy across models for each task and sort tasks from easiest to hardest based on these averages. In general, we found that Referring Dot-Pointing and Counting are the easiest, with models achieving over 20% accuracy on average, whereas Mental Rotation 3D (Cube), Patch Reassembly, and Mental Rotation 3D (Objaverse) are the hardest with an accuracy around 1%. Sliding Block, Maze 3D, Fetch Pick-Place, and Video Unshuffle also pose significant challenges for the models, with less than 5% on average. This suggests that tasks requiring memory and long-horizon planning, or strong 3D spatial understanding, remain the most difficult for current models.

Number of Steps. In Fig. 17, we calculate the number of steps taken on all trajectories for each model and calculate the number of correct trajectories (green) and the number of incorrect trajectories (red). There is a clear cutoff on steps 20 (maximum steps allowed for Easy setting) and 30 (maximum steps allowed for Hard setting), indicating that all models tend to reach the maximum number of steps. We also observed a "U-shaped" trend over the steps for all models, where they tend to either terminate early or continue until the final step.

Easy to Hard Performance Drop. In Fig. 18, we calculate the average accuracy in Easy and Hard, respectively, on all models, and then visualize the performance gap between easy and hard on each task. The biggest Easy to Hard performance drops occur on

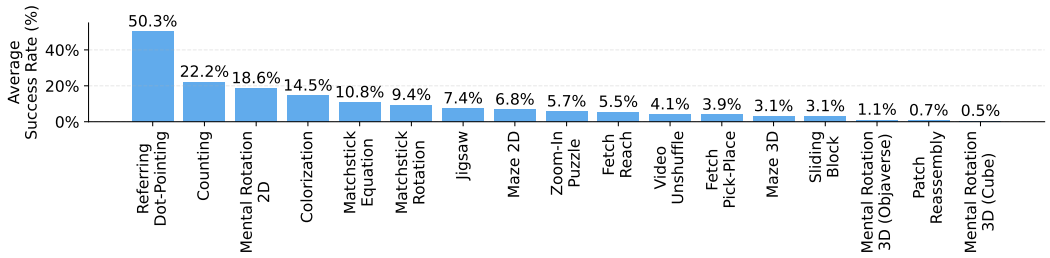


Figure 16: Average success rate across frontier models on each task. The easiest tasks are Referring Dot-Pointing, and Counting, with over 20% accuracy on average across all models, while the hardest tasks are Mental Rotation 3D (Cube), Patch Reassembly, and Mental Rotation, with the average accuracy less than 2% on average.

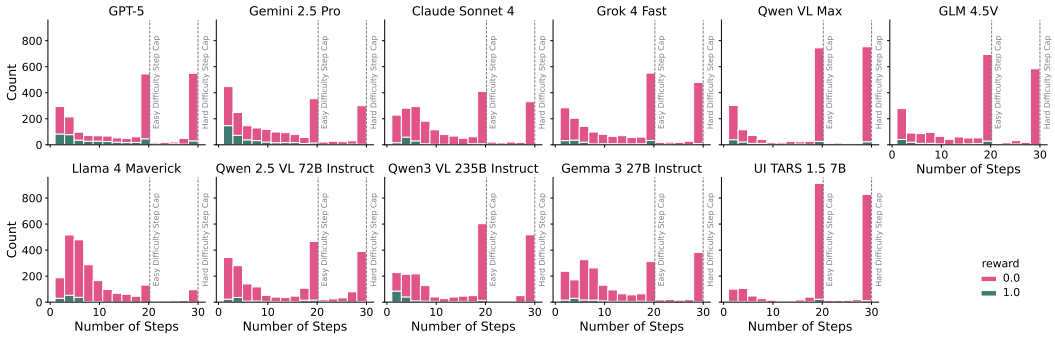


Figure 17: The Number of Steps each Model Takes Over all Tasks. We calculate the number of steps over all trajectories for each model and visualize the correct trajectories (green) and the incorrect trajectories (red).

Counting and Jigsaw. For Counting, accuracy drops sharply as the number of objects increases. For Jigsaw, performance drops to near zero as the puzzle changes from 2x2 to 3x3, suggesting that this task can be further scaled to even more difficult n×n configurations. For some tasks (*e.g.*, Patch Reassembly, Sliding Block, Video Unshuffle), the absolute gap is smaller, likely because Easy performance is already very low (≈0). These tasks are also naturally scalable in terms of difficulty. For example, increasing the number of patches for Patch Reassembly, the number of blocks for Sliding Block, or the number of frames for Video Unshuffle. As VLMs improve and begin to reliably solve the Easy settings, we expect to see larger easy-to-hard gaps on these tasks, and that our gym can be correspondingly scaled to provide harder task variants.

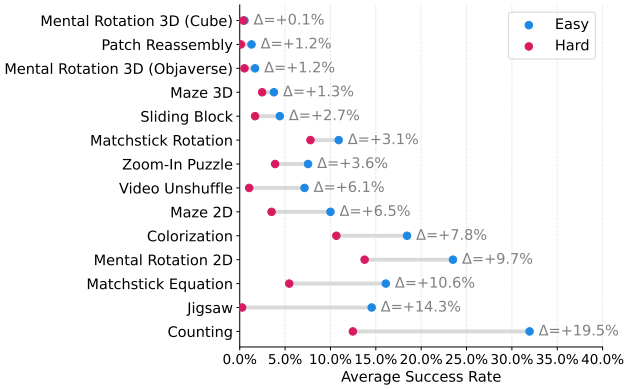


Figure 18: Easy → Hard Performance Drop. For each task, we calculate the average accuracy on Easy and Hard, respectively, over all models, and then visualize the performance drop between Easy and Hard.

Model Rankings. In Fig. 19, we plot each model's ranking on every task. When multiple models tie on a task, we assign them the worse (higher) numerical rank. For example, if two models are tied for the best score, we label both as rank 2, so that ties do not overstate how clearly a model is separated from others. While these mostly align with the global leaderboard, they also reveal clear task-specific strengths and weaknesses. For example, on specialization, Qwen2.5 VL 72B Instruct performs well on Counting (rank 2), Gemma 3 27B Instruct performs well on Maze 3D (rank 3), and Qwen3 VL 235B Instruct performs well on Referring Dot-Pointing. On weakness, despite strong average performance, GPT-5 performs poorly on Video Unshuffle and Maze 3D, while Gemini 2.5 Pro performs poorly on Jigsaw (Hard) and Patch Reassembly.

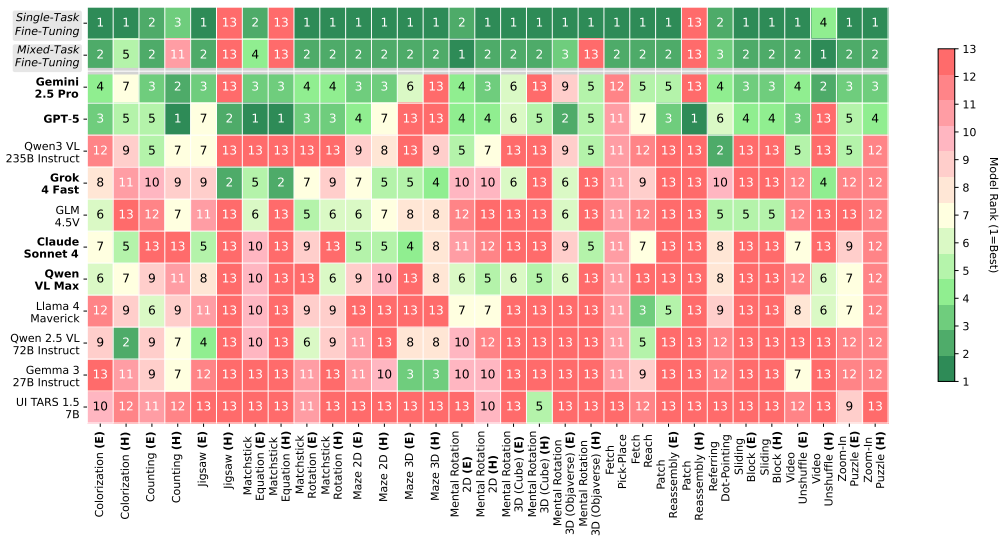


Figure 19: Model Rankings Per Task. We rank all the models on each task and show the ranking in the table.

F.0.2 SAMPLE TRAJECTORY FOR "ACTION LOOPING" (BY GPT-5) (Back to Table of Contents)

Reason

```
The model repeatedly issues the same swap command between the same two pieces without
making any progress towards solving the puzzle, as evidenced by identical actions over
multiple consecutive steps.
```

Evidence

```
"('swap', ((0, 0), (0, 1)))" is issued in steps 1, 2, 3, 4, 5, and 6, with no change in
strategy or progress.
```

Prompt p_0

```
You are solving a 2x2 jigsaw puzzle. The puzzle pieces are currently scrambled. Your goal
is to rearrange the pieces to recover the image.

Available actions:
1. 'swap': Swap two pieces by specifying their coordinates. Format: `('swap', ((row1, col1
), (row2, col2)))` where coordinates start from (0,0) at the top-left corner.
2. 'reorder': Reorder all pieces at once. Format: `('reorder', [0, 1, 2, ..., 3])` where
the list represents the desired order of pieces from top-left to bottom-right.
3. 'stop': End the puzzle solving session. Format: `('stop', 'stop')`

Please respond with exactly one action and its arguments in the specified format. For
example:
- To swap two pieces: `('swap', ((0, 0), (1, 1)))`
- To reorder all pieces: `('reorder', [0, 1, 2, ..., 3])`
- To stop: `('stop', 'stop')`

Index-to-cell mapping (0-based rows/cols):
- Index = row * 2 + col.
- Row 0 covers indices 0..1, row 1 covers 2..3, etc.
Example for 2x2: (0,0)->0, (0,1)->1, (1,0)->2, (1,1)->3.

This is step 1. You are allowed to take 19 more steps.
```

Observation o_0

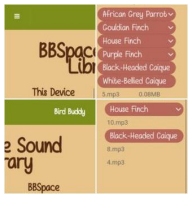
Action a_0

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_1



Prompt p_1

```
This is step 2. You are allowed to take 18
more steps.
```

Action a_1

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_2

Prompt p_2

```
This is step 3. You are allowed to take 17
more steps.
```

Action a_2

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_3

Prompt p_3

```
This is step 4. You are allowed to take 16
more steps.
```

Action a_3

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_4

Prompt p_4

```
This is step 5. You are allowed to take 15
more steps.
```

Action a_4

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_5

Prompt p_5

```
This is step 6. You are allowed to take 14
more steps.
```

Action a_5

```
('swap', ((0, 0), (0, 1)))
```

VLM Raw Output:

```
('swap', ((0, 0), (0, 1)))
```

Observation o_6

Prompt p_6

```
This is step 7. You are allowed to take 13
more steps.
```

Action a_6

```
('stop', 'stop')
```

VLM Raw Output:

```
('stop', 'stop')
```

F.0.4 SAMPLE TRAJECTORY FOR "STATE MISMANAGEMENT" (BY CLAUDE SONNET 4) (Back to Table of Contents)

Reason

```
The model revisits previously blocked directions and fails to remember which moves have
already been proven impossible, leading to repeated failed actions instead of seeking new
routes.
```

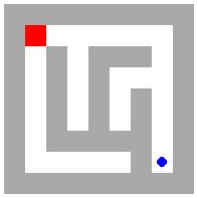
Evidence

```
After being told 'Cannot move into a wall.' for left and up, the model continues to try
those directions in subsequent steps, e.g., after failing to move left at step 9, it tries
 left again at steps 11, 13, and 17; after failing to move up at step 10, it tries up
again at steps 15, 16, and 18.
```

Prompt p_0

```
You are navigating a 9x9 maze. The maze consists of walls (gray) and paths (white). You
are represented by a blue circle, and your goal is to reach the red target square.

Available actions:
1. 'move': Move in one of four directions. Format: `('move', direction)` where direction
is an integer:
   - 0=right, 1=up, 2=left, 3=down
2. 'stop': End the navigation session. Format: `('stop', 'stop')`

Success: You succeed if you reach the red target square.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move right: `('move', 0)`
- To move up: `('move', 1)`
- To move left: `('move', 2)`
- To move down: `('move', 3)`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 19 more steps.
```

Observation o_0



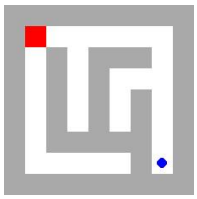
Action a_0

```
('move', 2)
```

VLM Raw Output:

```
I need to analyze the maze to find a path from my current position (blue circle) to the
red target square.

Looking at the maze, I can see:
- I'm currently positioned in the bottom right area (blue circle)
- The red target is in the top left corner
- There are gray walls and white paths
- I need to navigate through the available white paths

From my current position, I can see there's a white path that I can move through. Let me
start by moving left to begin navigating toward the target.

`('move', 2)`
```

Observation o_1



Prompt p_1

```
Environment feedback: Cannot move into a
wall.

This is step 2. You are allowed to take 18
more steps.
```

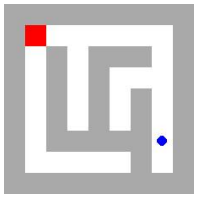
Action a_1

```
('move', 1)
```

VLM Raw Output:

```
I see that I cannot move left from my
current position because there's a wall
there. Let me reassess my surroundings.

Looking at my current position (blue circle
), I need to find which directions have
white paths available. From what I can see,
 I should try moving up to find an
available path.

`('move', 1)`
```

Observation o_2



Prompt p_2

```
Environment feedback: Action executed
successfully.

This is step 3. You are allowed to take 17
more steps.
```

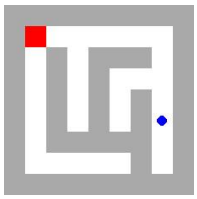
Action a_2

```
('move', 1)
```

VLM Raw Output:

```
Good! I successfully moved up. Now I can
see I'm in a vertical white path. Looking
at the maze structure, I need to continue
moving up to find a way to navigate toward
the red target in the top left corner.

`('move', 1)`
```

Observation o_3



Prompt p_3

```
Environment feedback: Action executed
successfully.

This is step 4. You are allowed to take 16
more steps.
```

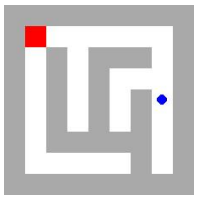
Action a_3

```
('move', 1)
```

VLM Raw Output:

```
Great! I've moved up again and I'm
continuing along the white path. I can see
that I'm in a vertical corridor on the
right side of the maze. I need to keep
moving up to find a horizontal path that
will allow me to move toward the left side
where the red target is located.

`('move', 1)`
```

Observation o_4



Prompt p_4

```
Environment feedback: Action executed
successfully.

This is step 5. You are allowed to take 15
more steps.
```

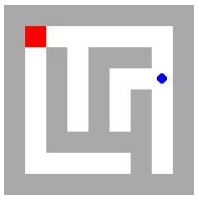
Action a_4

```
('move', 1)
```

VLM Raw Output:

```
Excellent! I've moved up again and I'm
getting closer to the top of the maze. I
can see that I'm still in the vertical
white path on the right side. I need to
continue moving up to reach a point where I
 can turn left and start moving toward the
red target in the top left corner.

`('move', 1)`
```

Observation o_5



Prompt p_5

```
Environment feedback: Action executed
successfully.

This is step 6. You are allowed to take 14
more steps.
```

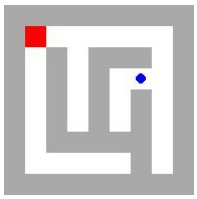
Action a_5

```
('move', 2)
```

VLM Raw Output:

```
Perfect! I've moved up again and I can see
that I'm now at a position where there
appears to be a horizontal white path
available to my left. This should allow me
to start moving toward the red target. Let
me try moving left now.

`('move', 2)`
```

Observation o_6



Prompt p_6

```
Environment feedback: Action executed
successfully.

This is step 7. You are allowed to take 13
more steps.
```

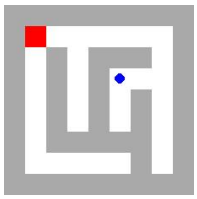
Action a_6

```
('move', 2)
```

VLM Raw Output:

```
Great! I successfully moved left and I'm
now in a horizontal white path. I can see I
'm making progress toward the left side of
the maze where the red target is located. I
 should continue moving left to get closer
to the target.

`('move', 2)`
```

Observation o_7



Prompt p_7

```
Environment feedback: Action executed
successfully.

This is step 8. You are allowed to take 12
more steps.
```

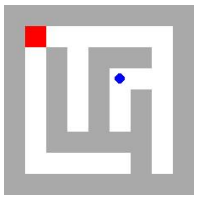
Action a_7

```
('move', 2)
```

VLM Raw Output:

```
Excellent! I've moved left again and I'm
getting closer to the left side of the maze.
 I can see that I'm still in a white path.
Looking at the maze structure, I need to
continue navigating to reach the red target
 in the top left corner. Let me try moving
left again to continue in this direction.

`('move', 2)`
```

Observation o_8



Prompt p_8

```
Environment feedback: Cannot move into a
wall.

This is step 9. You are allowed to take 11
more steps.
```

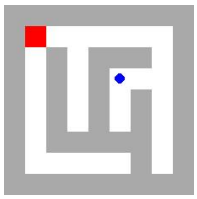
Action a_8

```
('move', 1)
```

VLM Raw Output:

```
I see that I can't move left anymore
because there's a wall blocking that
direction. Looking at my current position,
I need to find another route. Since I can't
 go left, let me try moving up to see if
there's a path that will eventually lead me
 to the red target.

`('move', 1)`
```

Observation o_9



Prompt p_9

```
Environment feedback: Cannot move into a
wall.

This is step 10. You are allowed to take 10
 more steps.
```

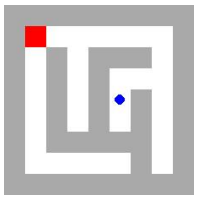
Action a_9

```
('move', 3)
```

VLM Raw Output:

```
I can't move up either because there's a
wall there. Let me try moving down to see
if there's an alternative path that will
allow me to navigate around this obstacle
and find another route to the red target.

`('move', 3)`
```

Observation o_{10}



Prompt p_{10}

```
Environment feedback: Action executed
successfully.

This is step 11. You are allowed to take 9
more steps.
```

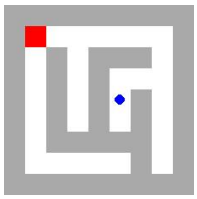
Action a_{10}

```
('move', 2)
```

VLM Raw Output:

```
Good! I successfully moved down. Now I'm in
 a different position and I need to find a
way to continue toward the red target.
Looking at the maze structure, I should try
 moving left again to see if there's now a
path available in that direction.

`('move', 2)`
```

Observation o_{11}



Prompt p_{11}

```
Environment feedback: Cannot move into a
wall.

This is step 12. You are allowed to take 8
more steps.
```

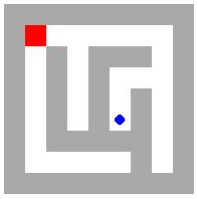
Action a_{11}

```
('move', 3)
```

VLM Raw Output:

```
I can't move left from this position either.
 Let me try moving down again to continue
exploring the available paths and find a
route that will eventually lead me to the
red target.

`('move', 3)`
```

Observation o_{12}



Prompt p_{12}

```
Environment feedback: Action executed
successfully.

This is step 13. You are allowed to take 7
more steps.
```

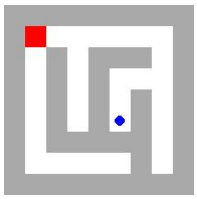
Action a_{12}

```
('move', 2)
```

VLM Raw Output:

```
Good! I moved down successfully. Now I'm in
 a lower position in the maze. Let me try
moving left from this new position to see
if I can find a path that leads toward the
red target.

`('move', 2)`
```

Observation o_{13}



Prompt p_{13}

```
Environment feedback: Cannot move into a
wall.

This is step 14. You are allowed to take 6
more steps.
```

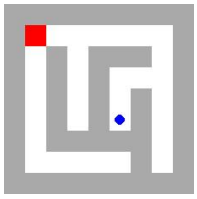
Action a_{13}

```
('move', 3)
```

VLM Raw Output:

```
I can't move left from here either. Let me
continue moving down to explore more of the
 maze and find an alternative path to the
red target.

`('move', 3)`
```

Observation o_{14}



Prompt p_{14}

```
Environment feedback: Cannot move into a
wall.

This is step 15. You are allowed to take 5
more steps.
```

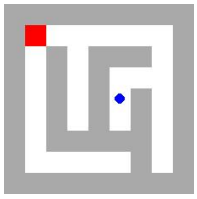
Action a_{14}

```
('move', 1)
```

VLM Raw Output:

```
I can't move down anymore because there's a
 wall. Let me try moving up to go back and
explore a different route.

`('move', 1)`
```

Observation o_{15}



Prompt p_{15}

```
Environment feedback: Action executed
successfully.

This is step 16. You are allowed to take 4
more steps.
```

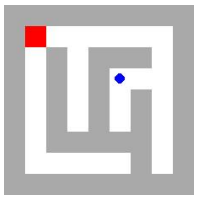
Action a_{15}

```
('move', 1)
```

VLM Raw Output:

```
Good! I moved up successfully. Now I'm back
 to a previous position. Let me try moving
up again to continue exploring and find a
path that will lead me toward the red
target.

`('move', 1)`
```

Observation o_{16}



Prompt p_{16}

```
Environment feedback: Action executed
successfully.

This is step 17. You are allowed to take 3
more steps.
```

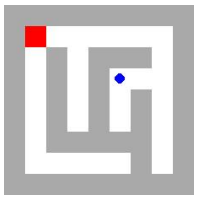
Action a_{16}

```
('move', 2)
```

VLM Raw Output:

```
Good! I'm back to this position. I'm
running low on steps (only 3 left), so I
need to think carefully. From what I can
see of the maze structure, let me try
moving left again to see if I can find a
path that leads to the red target.

`('move', 2)`
```

Observation o_{17}



Prompt p_{17}

```
Environment feedback: Cannot move into a
wall.

This is step 18. You are allowed to take 2
more steps.
```

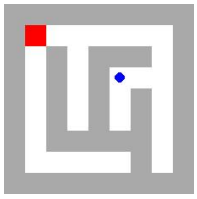
Action a_{17}

```
('move', 1)
```

VLM Raw Output:

```
I'm running out of steps and haven't found
a clear path to the red target yet. Looking
 at the maze structure, I need to find a
way to get to the left side. Let me try
moving up to see if there's a path at a
higher level that I might have missed.

`('move', 1)`
```

Observation o_{18}



Prompt p_{18}

```
Environment feedback: Cannot move into a
wall.

This is step 19. You are allowed to take 1
more steps.
```

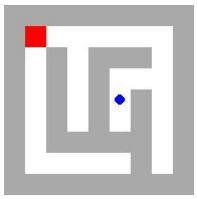
Action a_{18}

```
('move', 3)
```

VLM Raw Output:

```
I'm down to my last step and haven't found
a successful path to the red target.
Looking at my options, I can only move down
 from this position since all other
directions are blocked by walls. Let me use
 my final move.

`('move', 3)`
```

Observation o_{19}



Prompt p_{19}

```
Environment feedback: Action executed
successfully.

This is step 20. You are allowed to take 0
more steps.
```

Action a_{19}

```
('stop', 'stop')
```

VLM Raw Output:

```
I've used all my available steps and was
not able to reach the red target square.
The maze appears to be more complex than I
initially assessed, and I may have gotten
trapped in a section without a clear path
to the target. I should stop now since I
have no more moves remaining.

`('stop', 'stop')`
```

F.0.6 SAMPLE TRAJECTORY FOR "EARLY TERMINATION" (BY GEMMA3 27B) (Back to Table of Contents)

Reason

```
After a long sequence of failed moves, the model explicitly gives up and submits the
equation as final, stating 'I give up.'
```

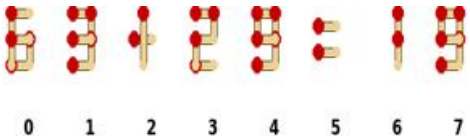
Evidence

```
"Twelve illegal moves. I give up. I suspect the segment definitions or move validation is
broken. I will submit a nonsense move. \n\n('stop', 'stop')"
```

Prompt p_0

```
You see a broken matchstick equation.
Your goal is to fix the equation by moving ONE match per action.
You see an image of the equation.
Symbols are indexed 0..N-1 from left to right (N = number of symbols).

Available actions:
1. 'move': Remove one match from segment 'src_seg' of symbol at 'src_idx', then add it to
segment 'dst_seg' of symbol at 'dst_idx'. Format: `('move', [src_idx, src_seg, dst_idx,
dst_seg])` where:
   - src_idx, dst_idx ∈ [0, N-1], with src_idx ≠ dst_idx
   - src_seg, dst_seg ∈ matchstick_puzzles.TOTAL (e.g., {0..12})
   - The move must result in valid symbols at BOTH positions.
2. 'undo': Revert the last move (if any). Format: `('undo', 'undo')`
3. 'stop': Submit your current equation as final. Format: `('stop', 'stop')`

Success: The submitted equation must be mathematically correct (evaluated as lhs == rhs).
Segment legend (indices depend on symbol):
  0..6 : 7-seg digits (a,b,c,d,e,f,g), 6 is also the horizontal stroke used by '+'
  7    : plus vertical stroke (used by '+')
  8    : the multiply sign that goes from top left to bottom right
  9    : the multiply sign that goes from top right to bottom left
  11,12: equals upper/lower bars (used by '=')
A segment is valid for a symbol only if the resulting set of segments maps to a known
glyph.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move a match: `('move', [0, 6, 2, 0])`
- To undo: `('undo', 'undo')`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 29 more steps.
```

Observation o_0



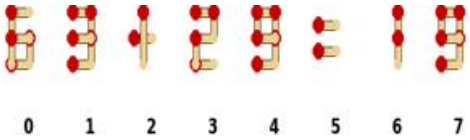
Action a_0

```
('move', [0, 0, 2, 0])
```

VLM Raw Output:

```
The given equation is visually represented as:

```
0 1 + 2 3 = 4 5 6 7
```

The equation seems to be aiming for a simple addition. Looking at the digits, it might
currently be something like "01 + 23 = 4567", which is clearly incorrect. A feasible
starting point is to try to make this equation "1 + 1 = 2".

Let's move a match to change '0' to '1', and '2' to '1'.

```
('move', [0, 0, 2, 0])
```
```

Observation o_1



Prompt p_1

```
Environment feedback: Illegal move.

This is step 2. You are allowed to take 28
more steps.
```

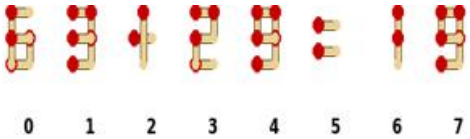
Action a_1

```
('move', [4, 0, 0, 0])
```

VLM Raw Output:

```
Okay, the previous move was illegal. Let's
rethink the approach. The current equation
appears to be something like 01 + 23 = 4567.
 A simple correction could be to transform
it into 1 + 1 = 2, or 1 + 0 = 1. Let's try
to get 1 + 1 = 2 first.

We need to change the zero in '0' to '1'.
We can accomplish this by moving one match
from '4' (which currently looks like a 4)
to the '0' (making it '1').

```
('move', [4, 0, 0, 0])
```
```

Observation o_2



Prompt p_2

```
Environment feedback: Illegal move.

This is step 3. You are allowed to take 27
more steps.
```

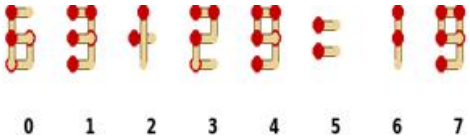
Action a_2

```
('move', [1, 0, 2, 0])
```

VLM Raw Output:

```
Okay, two illegal moves in a row. Let's
analyze the situation carefully. We want to
 transform the initial state into a valid
equation. The current symbols are messy.
Let's try to get something like 1 + 0 = 1.

I will try moving a match from '1' (
currently looking like a '1') to '2' making
 it a '0'. Then, I'll move a match from '3'
 to '0' to create a '1'.

```
('move', [1, 0, 2, 0])
```
```

Observation o_3



Prompt p_3

```
Environment feedback: Illegal move.

This is step 4. You are allowed to take 26
more steps.
```

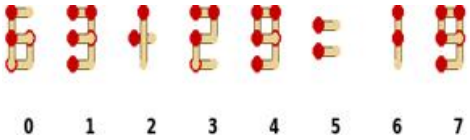
Action a_3

```
('move', [1, 0, 0, 0])
```

VLM Raw Output:

```
Okay, three illegal moves. This is getting
frustrating. Let's take a step back.
Instead of focusing on 1+0=1, let's try to
make something simpler, like 0 + 0 = 0.

I will move a match from '1' to '0' to make
 it '0'.

```
('move', [1, 0, 0, 0])
```
```

Observation o_4



Prompt p_4

```
Environment feedback: Illegal move.

This is step 5. You are allowed to take 25
more steps.
```

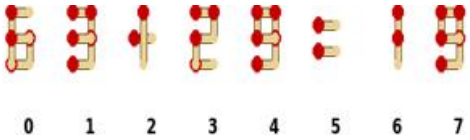
Action a_4

```
('move', [0, 0, 1, 0])
```

VLM Raw Output:

```
Alright, four illegal moves. It appears I'm
 consistently misjudging valid moves based
on the visual representation. Let's try a
different strategy. Let's attempt to make
the equation 1 + 1 = 2.

I will move a match from '0' (currently
looking like a zero) to '1' (currently
looking like a one) to make it a '1'.

```
('move', [0, 0, 1, 0])
```
```

Observation o_5



Prompt p_5

```
Environment feedback: Illegal move.

This is step 6. You are allowed to take 24
more steps.
```

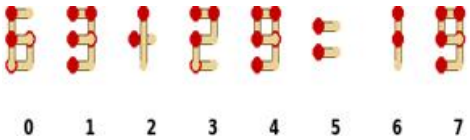
Action a_5

```
('move', [6, 0, 7, 0])
```

VLM Raw Output:

```
Five illegal moves! Okay, I'm clearly
having trouble visualizing the valid moves.
 I need to be more conservative. Let's try
to change the equals sign to a plus sign.

I will move a match from '6' (the
horizontal bar of the equals sign) to '7' (
the vertical bar of the equals sign).

```
('move', [6, 0, 7, 0])
```
```

Observation o_6



Prompt p_6

```
Environment feedback: Illegal move.

This is step 7. You are allowed to take 23
more steps.
```

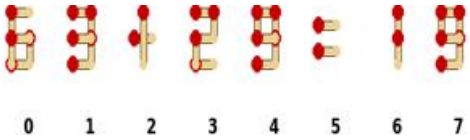
Action a_6

```
('undo', 'undo')
```

VLM Raw Output:

```
Six illegal moves... This is incredibly
frustrating. I'm clearly not understanding
how the move validation works. I will
attempt to undo the last move.

```
('undo', 'undo')
```
```

Observation o_7



Prompt p_7

```
Environment feedback: Nothing to undo.

This is step 8. You are allowed to take 22
more steps.
```

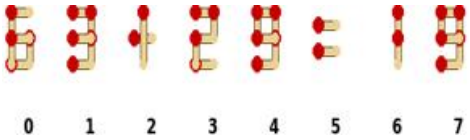
Action a_7

```
('move', [0, 0, 4, 0])
```

VLM Raw Output:

```
Okay, no moves to undo. Let's try to make
'4' into '5'. I will attempt to move a
match from '0' to '4'.

```
('move', [0, 0, 4, 0])
```
```

Observation o_8



Prompt p_8

```
Environment feedback: Illegal move.

This is step 9. You are allowed to take 21
more steps.
```

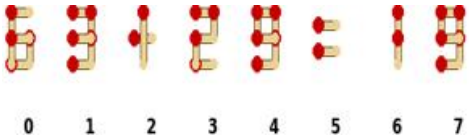
Action a_8

```
('move', [1, 0, 0, 0])
```

VLM Raw Output:

```
Eight illegal moves... This is getting
ridiculous. I suspect there's a hidden
constraint or a misunderstanding of the
segment labeling. Let's try a very simple
move to see if *anything* works. I will
move a match from '1' to '0'.

```
('move', [1, 0, 0, 0])
```
```

Observation o_9



Prompt p_9

```
Environment feedback: Illegal move.

This is step 10. You are allowed to take 20
 more steps.
```

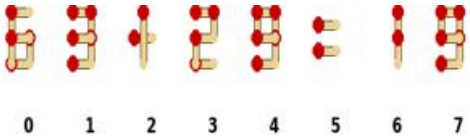
Action a_9

```
('move', [2, 0, 5, 0])
```

VLM Raw Output:

```
Nine illegal moves. I am at a complete loss.
 Since every attempt is failing, I'm going
to try a completely random move to see if
the system provides any clue about what's
going wrong.

```
('move', [2, 0, 5, 0])
```
```

Observation o_{10}



Prompt p_{10}

```
Environment feedback: Illegal move.

This is step 11. You are allowed to take 19
 more steps.
```

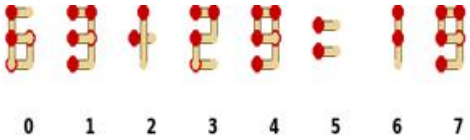
Action a_{10}

```
('undo', 'undo')
```

VLM Raw Output:

```
Ten illegal moves. This is clearly not
working. I suspect the problem lies in the
segment definitions and my inability to map
 them correctly to the visual
representation. I will try undoing the last
 move, although there is no history.

```
('undo', 'undo')
```
```

Observation o_{11}



Prompt p_{11}

```
Environment feedback: Nothing to undo.

This is step 12. You are allowed to take 18
 more steps.
```

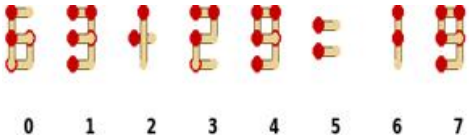
Action a_{11}

```
('move', [3, 0, 6, 0])
```

VLM Raw Output:

```
Eleven illegal moves and no undo history.
This is extremely frustrating. I'm going to
 just try moving a match from a seemingly
arbitrary location to another.

```
('move', [3, 0, 6, 0])
```
```

Observation o_{12}



Prompt p_{12}

```
Environment feedback: Illegal move.

This is step 13. You are allowed to take 17
 more steps.
```

Action a_{12}

```
('stop', 'stop')
```

VLM Raw Output:

```
Twelve illegal moves. I give up. I suspect
the segment definitions or move validation
is broken. I will submit a nonsense move.

```
('stop', 'stop')
```
```

F.0.8 SAMPLE TRAJECTORY FOR "VISUAL/SPATIAL PERCEPTION" (BY GEMINI 2.5 PRO)

(Back to Table of Contents)

Reason

```
The model frequently misjudges the effect of its rotations, such as applying large yaw or
roll corrections that result in the object being further from the target orientation,
indicating it is not fully utilizing the visual feedback provided after each step.
```

Evidence

```
After '('rotate', [-45.0, 0.0, 0.0])', the model observes the object is now vertical and
then applies '('rotate', [90.0, 0.0, 0.0])', which overcorrects. Later, it issues '('
rotate', [-90.0, 0.0, 0.0])' and then again '('rotate', [-90.0, -30.0, 0.0])', showing it
is not accurately interpreting the visual state changes. The model also states, 'This is a
 mess,' and attempts to 'reset and reconsider,' indicating confusion about the spatial
state.
```

Prompt p_0

```
You see two images side-by-side:
- Left: the original 3D object (target orientation)
- Right: the same object under an unknown rotation

Your goal is to rotate the right object back to the original target orientation.
Available actions:
1. 'rotate': Rotate the 3D object by Euler angles. Format: `('rotate', [dyaw, dpitch,
droll])` where each of dyaw, dpitch, droll is in degrees between -180 and 180.
2. 'stop': Submit your final rotation. Format: `('stop', 'stop')`

Success: You succeed if you end up within 10.0 degrees of target orientation.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To rotate in yaw: `('rotate', [15.0, 0.0, 0.0])`
- To rotate in roll: `('rotate', [0.0, 0.0, -10.0])`
- To submit: `('stop', 'stop')`
This is step 1. You are allowed to take 19 more steps.
```

Observation o_0

Action a_0

```
('rotate', [0.0, 60.0, 0.0])
```

VLM Raw Output:

```
The user wants me to rotate the object on the right to match the orientation of the object
 on the left.

**Analysis of Target Object (Left):**
*   The object is a chain of connected blocks forming a Z-like shape.
...
(trimmed by author for space)

**Analysis of Current Object (Right):**
*   It's the same object, but its orientation is different.
...
(trimmed by author for space)

**Plan:**
1.  My first step will be to correct the most obvious distortion, which is the pitch. The
object is tilted away from the camera, making it look foreshortened.
...
(trimmed by author for space)

**Action:**
I will apply a positive pitch of 60 degrees.
`('rotate', [0.0, 60.0, 0.0])`
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully.

This is step 2. You are allowed to take 18
more steps.
```

Action a_1

```
('rotate', [-45.0, 0.0, 0.0])
```

VLM Raw Output:

```
My previous action `('rotate', [0.0, 60.0,
0.0])` has brought the object into a mostly
 flat orientation, viewed from above.

**Analysis of Current State:**
*   **Target (Left):** The object is in a '
Z' shape, with its main body oriented
horizontally, but slightly tilted. The
bottom right leg extends to the right, and
the top left leg extends to the left.
*   **Current (Right):** The object is now
flat, but it's oriented diagonally from the
 top-left to the bottom-right. It looks
like a backward 'S'.

**Plan:**
1.  The primary misalignment is now in the
yaw axis. I need to rotate the object
clockwise to make its central axis more
horizontal.
...
(trimmed by author for space)

**Action:**
`('rotate', [-45.0, 0.0, 0.0])`
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully.

This is step 3. You are allowed to take 17
more steps.
```

Action a_2

```
('rotate', [90.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** The object is in a
horizontal 'Z' configuration, viewed
slightly from above. The top faces are
black, and the side faces have a triangular
 pattern.
*   **Current (Right):** After my last
rotation, the object is in a vertical 'S'
configuration. I wanted to make it
horizontal, but my yaw rotation made it
more vertical. This means I rotated in the
wrong direction. The shape went from a top-
left to bottom-right diagonal to a more
vertical orientation. I need to reverse
this and go further in the opposite
direction. I applied -45 yaw, so to undo it
 and apply the correct rotation, I need to
apply a positive yaw. A rotation of +90
degrees seems appropriate to turn the
vertical 'S' into a horizontal 'Z'.

**Plan:**
1.  The most significant remaining error is
 the yaw rotation. The object is oriented
vertically instead of horizontally.
...
(trimmed by author for space)

**Action:**
`('rotate', [90.0, 0.0, 0.0])`
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully.

This is step 4. You are allowed to take 16
more steps.
```

Action a_3

```
('rotate', [0.0, 0.0, 180.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A Z-like shape,
viewed from above and slightly to the side.
 The top surfaces are black.
*   **Current (Right):** The object is now
a sideways T-shape. The overall structure
is rotated incorrectly. It seems my large
yaw rotation was an overcorrection. More
importantly, the object seems flipped.
Notice how in the target, the zig-zag part
forms a "Z", whereas the object I have been
 manipulating has been forming an "S". This
 usually indicates a 180-degree roll error.
 The object is upside down.

**Plan:**
1.  The most critical error seems to be
that the object is upside down, which makes
 it look like a mirror image ('S' instead
of 'Z') when viewed from above.
2.  I will apply a 180-degree roll to flip
the object over. This should correct the
fundamental shape from an 'S' to a 'Z'.
3.  After this correction, I will likely
need to adjust the yaw and pitch again, but
 the basic shape should then match the
target.

**Action:**
`('rotate', [0.0, 0.0, 180.0])`
```

Observation o_4

Prompt p_4

```
Environment feedback: Action executed
successfully.

This is step 5. You are allowed to take 15
more steps.
```

Action a_4

```
('rotate', [-45.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A Z-like shape,
viewed from above, with its main axis
roughly horizontal. The bottom-right leg
extends to the right, and the top-left leg
extends to the left.
*   **Current (Right):** The 180-degree
roll was successful. The object now has the
 correct 'Z' configuration. However, it's
oriented diagonally, from the top-left to
the bottom-right. The top surfaces are
black, which is correct.

**Plan:**
1.  The main remaining error is in the yaw
axis. I need to rotate the object clockwise
 to make it horizontal.
2.  The current angle appears to be about
45 degrees off the horizontal.
3.  I will apply a clockwise (negative) yaw
 of approximately 45 degrees. This should
align the primary axis of the object with
the target.

**Action:**
`('rotate', [-45.0, 0.0, 0.0])`
```

Observation o_5

Prompt p_5

```
Environment feedback: Action executed
successfully.

This is step 6. You are allowed to take 14
more steps.
```

Action a_5

```
('rotate', [35.0, -20.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A horizontal 'Z'
shape, viewed from above, showing the top
black faces and the side patterned faces.
There's a slight pitch, making us look down
 on it.
*   **Current (Right):** My previous yaw
rotation seems to have overshot. The object
 is now tilted the other way, from top-
right to bottom-left. It looks like a 'Z'
that needs to be rotated counter-clockwise.
 Also, the side faces are not visible, only
 the black top faces. This indicates a
pitch error. The object is perfectly flat
to the camera, while the target has some
perspective.

**Plan:**
1.  **Yaw Correction:** I rotated -45
degrees, which was an overcorrection from
the previous state. I need to rotate it
back counter-clockwise (positive yaw). The
previous state was at ~45 degrees, the
current state is at ~-45 degrees. To get to
 the target's slightly-off-horizontal
position (let's say -10 degrees), I need to
 rotate by about 35 degrees counter-
clockwise.
2.  **Pitch Correction:** The target is
viewed from slightly above. The current
object is viewed from directly above (
orthographic-like view, only top faces
visible). This means I need to pitch the
object down (negative pitch) slightly to
reveal the side faces as seen in the target.
 I'll estimate this at -20 degrees.

I will combine these two corrections into
one action.

**Action:**
`('rotate', [35.0, -20.0, 0.0])`
```

Observation o_6

Prompt p_6

```
Environment feedback: Action executed
successfully.

This is step 7. You are allowed to take 13
more steps.
```

Action a_6

```
('rotate', [40.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A horizontal 'Z'
shape, viewed from above.
*   **Current (Right):** The object is now
a 'Z' shape, but it's still oriented
diagonally from top-right to bottom-left.
My previous yaw correction of +35 was not
enough. The pitch correction seems to have
worked well, as the side faces are now
visible, similar to the target.

**Plan:**
1.  The primary remaining error is the yaw.
 It's still rotated too much clockwise. I
need to apply a counter-clockwise (positive
) yaw rotation.
2.  The current angle is about -45 to -50
degrees from horizontal. The target is
maybe -10 degrees from horizontal. So I
need to rotate it by about 35-40 degrees
counter-clockwise. Let's try another +40
degrees of yaw.

**Action:**
`('rotate', [40.0, 0.0, 0.0])`
```

Observation o_7

Prompt p_7

```
Environment feedback: Action executed
successfully.

This is step 8. You are allowed to take 12
more steps.
```

Action a_7

```
('rotate', [-90.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** The object forms a '
Z' that is mostly horizontal, tilted
slightly down on the right. We see the top
black surfaces and the side patterned
surfaces.
*   **Current (Right):** My previous yaw
correction overshot the target
significantly. The object is now oriented
vertically. The overall 'Z' shape is
correct, as is the perspective (showing top
 and side faces), but the rotation around
the vertical (yaw) axis is wrong.

**Plan:**
1.  The current object is vertical, while
the target is horizontal. This represents
approximately a 90-degree difference in yaw.

2.  To correct this, I need to rotate the
object clockwise by about 90 degrees.
3.  I will apply a negative yaw of 90
degrees.

**Action:**
`('rotate', [-90.0, 0.0, 0.0])`
```

Observation o_8

Prompt p_8

```
Environment feedback: Action executed
successfully.

This is step 9. You are allowed to take 11
more steps.
```

Action a_8

```
('rotate', [-90.0, -30.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A horizontal 'Z'
shape viewed from a slightly elevated
position, showing both top (black) and side
 (patterned) faces.
*   **Current (Right):** The object is
shaped like a chair or an 'h'. The main
axis is vertical instead of horizontal.
This is a 90-degree yaw error. Additionally
, the perspective looks off. We're seeing
it almost perfectly top-down, without the
slight angle that reveals the side faces,
suggesting a pitch error.

**Plan:**
1.  **Yaw Correction:** The object is
rotated 90 degrees counter-clockwise from
the target orientation. I need to rotate it
 90 degrees clockwise to make its main axis
 horizontal. This corresponds to a negative
 yaw of 90 degrees.
2.  **Pitch Correction:** The target object
 is pitched slightly away from the camera (
negative pitch), revealing the side faces.
The current object appears flat. I will re-
introduce a negative pitch to match the
target's perspective. I'll estimate this at
 -30 degrees.

I will combine these two adjustments into a
 single action.

**Action:**
`('rotate', [-90.0, -30.0, 0.0])`
```

Observation o_9

Prompt p_9

```
Environment feedback: Action executed
successfully.

This is step 10. You are allowed to take 10
 more steps.
```

Action a_9

```
('rotate', [45.0, 10.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
...
(trimmed by author for space)

**Let's reset and reconsider the total
rotation needed from the start.**
...
(trimmed by author for space)

**Plan:**
1.  Apply a positive yaw of 45 degrees to
make the main axis horizontal.
2.  I will also add a small positive pitch
of about 10 degrees to make it slightly
flatter.

**Action:**
`('rotate', [45.0, 10.0, 0.0])`
```

Observation o_{10}

Prompt p_{10}

```
Environment feedback: Action executed
successfully.

This is step 11. You are allowed to take 9
more steps.
```

Action a_{10}

```
('rotate', [0.0, 0.0, -90.0])
```

VLM Raw Output:

```
This is getting complicated, and my
previous moves have clearly been leading me
 astray. The current object's shape is
heavily distorted, looking more like '<'
than the target 'Z'. This indicates a
significant error in an axis other than yaw
, likely roll or pitch. The fundamental
structure is incorrect on the screen.

Let's try to restore the Z-like structure.
The current '<' shape can be thought of as
a 'Z' that has been rolled onto its side.
To bring it upright, I need to apply a roll.
 Visualizing the transformation, if a
horizontal 'Z' is rolled 90 degrees counter
-clockwise (+90 roll), its limbs would
point up and down, similar to the current
'<' shape. Therefore, to reverse this, I
should apply a clockwise roll (-90 degrees).


This is a significant correction, but it's
aimed at restoring the basic recognizable
shape, after which I can fine-tune the yaw
and pitch.

**Plan:**
1.  The primary problem is that the object
no longer projects as a 'Z'. It looks like
it's been rolled on its side.
2.  I will apply a large clockwise roll to
try and bring the 'Z' structure back into
the main viewing plane.
3.  I'll use -90 degrees for this
correction.

**Action:**
`('rotate', [0.0, 0.0, -90.0])`
```

Observation o_{11}

Prompt p_{11}

```
Environment feedback: Action executed
successfully.

This is step 12. You are allowed to take 8
more steps.
```

Action a_{11}

```
('rotate', [-20.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** A horizontal 'Z'
shape. More specifically, the line
connecting the two ends is almost
horizontal, tilted slightly clockwise.
*   **Current (Right):** The -90 degree
roll was very effective. The object is now
a clearly recognizable 'Z' shape. The pitch
 and roll seem very close to the target.
The main difference is the yaw. The line
connecting the two ends of the current
object is tilted counter-clockwise by about
 15-20 degrees. I need to rotate it
clockwise.

**Plan:**
1.  The alignment is very close. The only
significant remaining error is a small yaw
rotation.
2.  I need to rotate the object clockwise (
negative yaw) to make the overall shape
more horizontal and match the target's
slight clockwise tilt.
3.  I estimate the required rotation to be
about -20 degrees.

**Action:**
`('rotate', [-20.0, 0.0, 0.0])`
```

Observation o_{12}

Prompt p_{12}

```
Environment feedback: Action executed
successfully.

This is step 13. You are allowed to take 7
more steps.
```

Action a_{12}

```
('rotate', [-5.0, 0.0, 0.0])
```

VLM Raw Output:

```
**Analysis of Current State:**
*   **Target (Left):** The target object is
 a Z-shape.
*   **Current (Right):** The current object
 is also a Z-shape. The orientation is
extremely close now.

**Comparison:**
*   **Yaw:** The current object's main axis
 is slightly more horizontal than the
target's. The target is tilted a tiny bit
more clockwise. I might need a very small
negative yaw (-5 degrees).
*   **Pitch:** The perspective seems almost
 identical. The amount of side-face visible
 is very similar.
*   **Roll:** The roll seems perfect.

The alignment is very good, likely within
the 10-degree success threshold. I'll make
one final, small adjustment to the yaw and
then stop.

**Plan:**
1.  Apply a tiny clockwise (negative) yaw
to match the slight downward tilt of the
target's right side.
2.  I estimate this to be around -5 degrees.

3.  Then I will submit.

**Action:**
`('rotate', [-5.0, 0.0, 0.0])`
```

Observation o_{13}

Prompt p_{13}

```
Environment feedback: Action executed
successfully.

This is step 14. You are allowed to take 6
more steps.
```

Action a_{13}

```
('stop', 'stop')
```

VLM Raw Output:

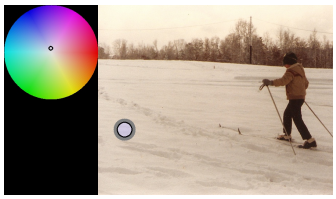
```
**Analysis of Current State:**
*   **Target (Left):** A Z-shaped object
viewed from a specific angle.
*   **Current (Right):** The object is now
in an orientation that is visually almost
identical to the target. The yaw, pitch,
and roll all appear to match.

The correction in the previous step has
successfully aligned the object. There are
no obvious remaining rotational errors. I
believe the current orientation is well
within the 10-degree success threshold.

**Plan:**
Submit the current orientation as the final
 answer.

**Action:**
`('stop', 'stop')`
```

B.2 COLORIZATION (Back to Table of Contents)

Prompt p_0

```
You are performing a color-matching task. You see two images side by side:
- LEFT: A color wheel showing your current hue and saturation selection
- RIGHT: An image with a circular region colored with your current selection (gray outside
 the circle)

Your goal is to adjust the hue and saturation to match the original color that appears at
the center of the circular region in the right image. The circle's border shows the exact
target location.

Success criteria: You succeed when your color selection closely matches the target color
in both hue and saturation.

Available actions:
1. 'rotate': Adjust the hue by rotating around the color wheel (circular motion). Format:
`('rotate', angle)` where angle is an integer between -360 and 360 degrees.
2. 'saturate': Adjust the saturation by moving toward or away from the center of the wheel
. Format: `('saturate', delta)` where delta is an integer between -255 and 255.
3. 'stop': Submit your final color choice when you're satisfied with the match. Format:
`('stop', 'stop')`.

Please respond with exactly one action and its arguments in the specified format. For
example:
- `('rotate', 30)` to rotate the hue +30 degrees clockwise
- `('rotate', -45)` to rotate the hue 45 degrees counter-clockwise
- `('saturate', 20)` to move away from center (increase saturation, more vivid)
- `('saturate', -30)` to move toward the center (decrease saturation, more muted)
- `('stop', 'stop')` to submit your answer when the colors match
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0



Action a_0

```
('rotate', 25)
```

Observation o_1

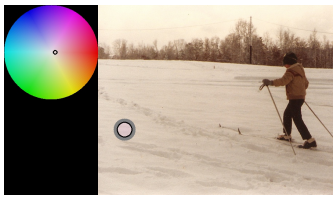
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('rotate', 73)
```

Observation o_2



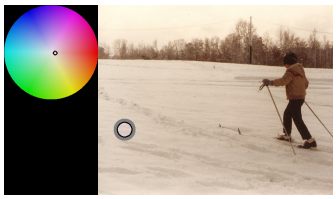
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('rotate', 15)
```

Observation o_3



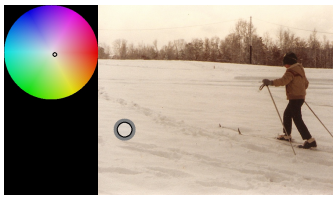
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('rotate', 17)
```

Observation o_4



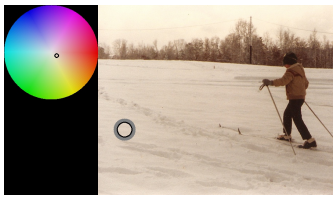
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('saturate', 12)
```

Observation o_5



Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('stop', 'stop')
```

B.4 COUNTING (Back to Table of Contents)

Prompt p_0

```
You are solving a horse-counting task. Count the number of horse in the image. You can
place dots to mark instances and then record your final count.

Available actions:
1. 'mark': Place a dot at normalized coordinates. Format: `('mark', (x, y))` where x and y
 are normalized coordinates between 0.0 and 1.0.
2. 'undo': Remove your last placed dot. Format: `('undo', 'undo')`
3. 'guess': Record your count guess. Format: `('guess', N)` where N is an integer between
5 and 30.
4. 'stop': End the counting session. Format: `('stop', 'stop')`

Success: You succeed if your final count guess matches the true number of objects.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To mark a point: `('mark', (0.5, 0.3))`
- To undo: `('undo', 'undo')`
- To guess count: `('guess', 5)`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('mark', (0.2122, 0.5766))
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('mark', (0.4058, 0.5663))
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('mark', (0.8144, 0.6096))
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('mark', (0.6180, 0.6005))
```

Observation o_4

Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('mark', (0.4069, 0.7038))
```

Observation o_5

Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('guess', 5)
```

Observation o_6

Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('stop', 'stop')
```

B.6 JIGSAW (Back to Table of Contents)

Prompt p_0

```
You are solving a 3x3 jigsaw puzzle. The puzzle pieces are currently scrambled. Your goal
is to rearrange the pieces to recover the image.

Available actions:
1. 'swap': Swap two pieces by specifying their coordinates. Format: `('swap', ((row1, col1
), (row2, col2)))` where coordinates start from (0,0) at the top-left corner.
2. 'reorder': Reorder all pieces at once. Format: `('reorder', [0, 1, 2, ..., 8])` where
the list represents the desired order of pieces from top-left to bottom-right.
3. 'stop': End the puzzle solving session. Format: `('stop', 'stop')`

Please respond with exactly one action and its arguments in the specified format. For
example:
- To swap two pieces: `('swap', ((0, 0), (1, 1)))`
- To reorder all pieces: `('reorder', [0, 1, 2, ..., 8])`
- To stop: `('stop', 'stop')`

Index-to-cell mapping (0-based rows/cols):
- Index = row * 3 + col.
- Row 0 covers indices 0..2, row 1 covers 3..5, etc.
Example for 2x2: (0,0)->0, (0,1)->1, (1,0)->2, (1,1)->3.

This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('swap', ((2, 0), (0, 2)))
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('swap', ((2, 1), (1, 0)))
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('swap', ((0, 2), (1, 0)))
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('swap', ((1, 2), (1, 1)))
```

Observation o_4

Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('swap', ((2, 2), (0, 1)))
```

Observation o_5

Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('swap', ((1, 1), (0, 1)))
```

Observation o_6

Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('swap', ((0, 0), (0, 1)))
```

Observation o_7

Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

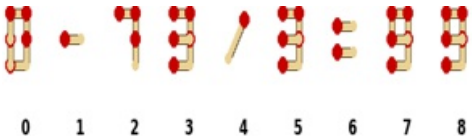
```
('stop', 'stop')
```

B.8 MATCHSTICK EQUATION (Back to Table of Contents)

Prompt p_0

```
You see an image of a broken matchstick equation.
Your goal is to fix the equation by moving ONE match per action.

Symbols (left->right) with indices:
[0:'0', 1:'-', 2:'7', 3:'3', 4:'/', 5:'3', 6:'=', 7:'9', 8:'9']

Available actions:
1. 'move': Remove one match from segment 'src_seg' of symbol at 'src_idx', then add it to
segment 'dst_seg' of symbol at 'dst_idx'. Format: `('move', [src_idx, src_seg, dst_idx,
dst_seg])` where:
   - src_idx, dst_idx ∈ [0, N-1], with src_idx ≠ dst_idx
   - src_seg, dst_seg ∈ matchstick_puzzles.TOTAL (e.g., {0..12})
   - The move must result in valid symbols at BOTH positions.
2. 'undo': Revert the last move (if any). Format: `('undo', 'undo')`
3. 'stop': Submit your current equation as final. Format: `('stop', 'stop')`

Success: The submitted equation must be mathematically correct (evaluated as lhs == rhs).

Segment legend (indices depend on symbol):
  0..6 : 7-seg digits (a,b,c,d,e,f,g), 6 is also the horizontal stroke used by '+'
  7    : plus vertical stroke (used by '+')
  8    : the multiply sign that goes from top left to bottom right
  9    : the multiply sign that goes from top right to bottom left
  11,12: equals upper/lower bars (used by '=')
A segment is valid for a symbol only if the resulting set of segments maps to a known
glyph.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move a match: `('move', [0, 6, 2, 0])`
- To undo: `('undo', 'undo')`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

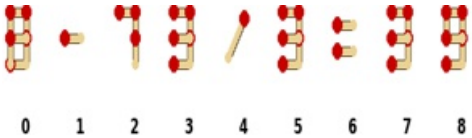
Observation o_0



Action a_0

```
('move', [7, 5, 0, 6])
```

Observation o_1



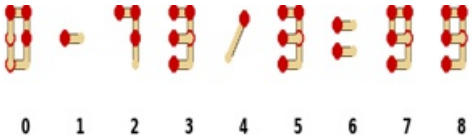
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('undo', 'undo')
```

Observation o_2



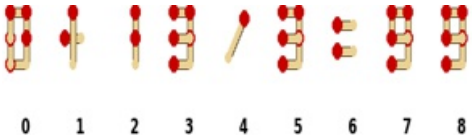
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', [2, 0, 1, 7])
```

Observation o_3



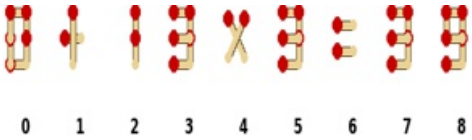
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', [7, 5, 4, 8])
```

Observation o_4



Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

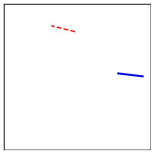
```
('stop', 'stop')
```

B.10 MATCHSTICK ROTATION (Back to Table of Contents)

Prompt p_0

```
You see a blue stick and a red target stick on a canvas. Your goal is to move and rotate
the blue stick to match the red target stick. Unit moves are in an *unknown* scale (could
be tiny or huge).

Available actions:
1. 'move': Translate by dx,dy units (any real number) and rotate by dθ° (any real number).
 Format: `('move', [dx, dy, dθ])`
2. 'stop': Submit your current stick as final. You win (+1) if your stick is within 5px
and 10° of the target. Format: `('stop', 'stop')`

Please respond with exactly one action and its arguments in the specified format. For
example:
- Move right by 2 units and rotate 45°: `('move', [2, 0, 45])`
- Move diagonally and rotate: `('move', [1.5, -0.8, 90])`
- Finalize the stick position: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

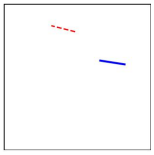
Observation o_0



Action a_0

```
('move', [-66.8, 46.2, -2.0])
```

Observation o_1



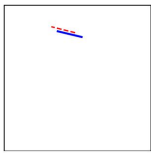
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', [-158.3, 109.4, -4.7])
```

Observation o_2



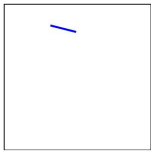
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', [-23.8, 16.5, -0.7])
```

Observation o_3



Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

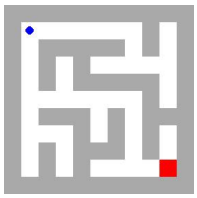
```
('stop', 'stop')
```

B.12 MAZE 2D (Back to Table of Contents)

Prompt p_0

```
You are navigating a 11x11 maze. The maze consists of walls (gray) and paths (white). You
are represented by a blue circle, and your goal is to reach the red target square.

Available actions:
1. 'move': Move in one of four directions. Format: `('move', direction)` where direction
is an integer:
   - 0=right, 1=up, 2=left, 3=down
2. 'stop': End the navigation session. Format: `('stop', 'stop')`

Success: You succeed if you reach the red target square.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move right: `('move', 0)`
- To move up: `('move', 1)`
- To move left: `('move', 2)`
- To move down: `('move', 3)`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0



Action a_0

```
('move', 3)
```

Observation o_1

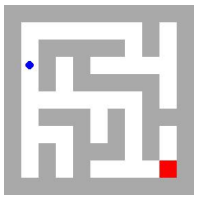
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', 3)
```

Observation o_2



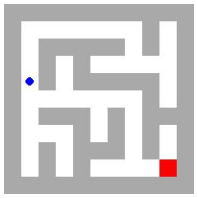
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', 3)
```

Observation o_3



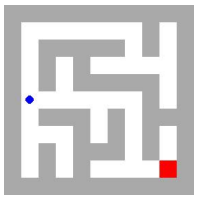
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', 3)
```

Observation o_4



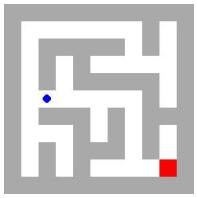
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('move', 0)
```

Observation o_5



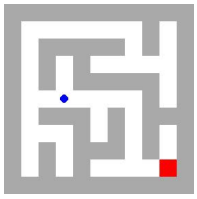
Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('move', 0)
```

Observation o_6



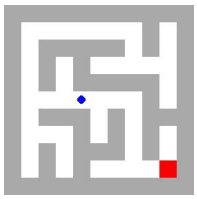
Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('move', 0)
```

Observation o_7



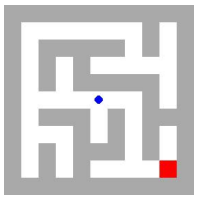
Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

```
('move', 0)
```

Observation o_8



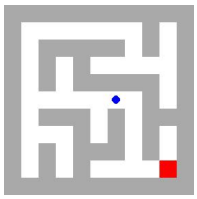
Prompt p_8

```
Environment feedback: Action executed
successfully. This is step 9. You are
allowed to take 91 more steps.
```

Action a_8

```
('move', 0)
```

Observation o_9



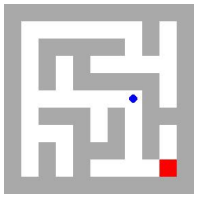
Prompt p_9

```
Environment feedback: Action executed
successfully. This is step 10. You are
allowed to take 90 more steps.
```

Action a_9

```
('move', 0)
```

Observation o_{10}



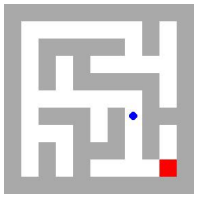
Prompt p_{10}

```
Environment feedback: Action executed
successfully. This is step 11. You are
allowed to take 89 more steps.
```

Action a_{10}

```
('move', 3)
```

Observation o_{11}



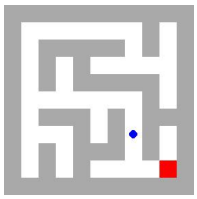
Prompt p_{11}

```
Environment feedback: Action executed
successfully. This is step 12. You are
allowed to take 88 more steps.
```

Action a_{11}

```
('move', 3)
```

Observation o_{12}



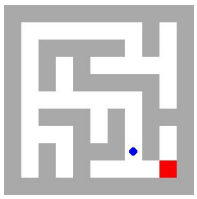
Prompt p_{12}

```
Environment feedback: Action executed
successfully. This is step 13. You are
allowed to take 87 more steps.
```

Action a_{12}

```
('move', 3)
```

Observation o_{13}



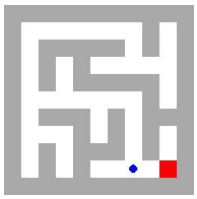
Prompt p_{13}

```
Environment feedback: Action executed
successfully. This is step 14. You are
allowed to take 86 more steps.
```

Action a_{13}

```
('move', 3)
```

Observation o_{14}



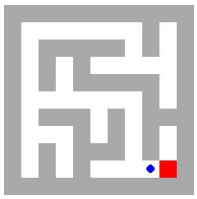
Prompt p_{14}

```
Environment feedback: Action executed
successfully. This is step 15. You are
allowed to take 85 more steps.
```

Action a_{14}

```
('move', 0)
```

Observation o_{15}



Prompt p_{15}

```
Environment feedback: Action executed
successfully. This is step 16. You are
allowed to take 84 more steps.
```

Action a_{15}

```
('move', 0)
```

Observation o_{16}

Prompt p_{16}

```
Environment feedback: Action executed
successfully. This is step 17. You are
allowed to take 83 more steps.
```

Action a_{16}

```
('stop', 'stop')
```

B.14 MAZE 3D (Back to Table of Contents)

Prompt p_0

```
You are navigating a 9x9 3D maze environment. The maze consists of walls and open paths.
You are given the first-person view from your current position and orientation.

Your goal is to reach the target location which ismarked by a red sphere.

Available actions:
1. 'move': Move one step forward in your current facing direction. Format: `('move', 0)`
2. 'turn': Rotate your view in the specified direction. Format: `('turn', direction)`
where direction is 1 (left), 2 (right), or 3 (around).
3. 'stop': Stop the episode. Format: `('stop', 'stop')`

Success: You succeed when you reach the target location (red sphere).

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move forward: `('move', 0)`
- To turn left: `('turn', 1)`
- To turn right: `('turn', 2)`
- To turn around: `('turn', 3)`
- To stop: `('stop', 'stop')`

Note: If you try to move forward into a wall, you will remain in your current position.
Turning actions do not change your position, only your facing direction.
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('move', 0)
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', 0)
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', 0)
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', 0)
```

Observation o_4

Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('move', 0)
```

Observation o_5

Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('move', 0)
```

Observation o_6

Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('turn', 2)
```

Observation o_7

Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

```
('move', 0)
```

Observation o_8

Prompt p_8

```
Environment feedback: Action executed
successfully. This is step 9. You are
allowed to take 91 more steps.
```

Action a_8

```
('move', 0)
```

Observation o_9

Prompt p_9

```
Environment feedback: Action executed
successfully. This is step 10. You are
allowed to take 90 more steps.
```

Action a_9

```
('turn', 2)
```

Observation o_{10}

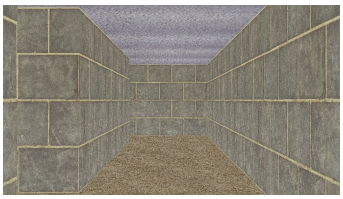
Prompt p_{10}

```
Environment feedback: Action executed
successfully. This is step 11. You are
allowed to take 89 more steps.
```

Action a_{10}

```
('move', 0)
```

Observation o_{11}



Prompt p_{11}

```
Environment feedback: Action executed
successfully. This is step 12. You are
allowed to take 88 more steps.
```

Action a_{11}

```
('move', 0)
```

Observation o_{12}

Prompt p_{12}

```
Environment feedback: Action executed
successfully. This is step 13. You are
allowed to take 87 more steps.
```

Action a_{12}

```
('turn', 1)
```

Observation o_{13}

Prompt p_{13}

```
Environment feedback: Action executed
successfully. This is step 14. You are
allowed to take 86 more steps.
```

Action a_{13}

```
('move', 0)
```

Observation o_{14}

Prompt p_{14}

```
Environment feedback: Action executed
successfully. This is step 15. You are
allowed to take 85 more steps.
```

Action a_{14}

```
('move', 0)
```

Observation o_{15}

Prompt p_{15}

```
Environment feedback: Action executed
successfully. This is step 16. You are
allowed to take 84 more steps.
```

Action a_{15}

```
('move', 0)
```

Observation o_{16}

Prompt p_{16}

```
Environment feedback: Action executed
successfully. This is step 17. You are
allowed to take 83 more steps.
```

Action a_{16}

```
('move', 0)
```

Observation o_{17}

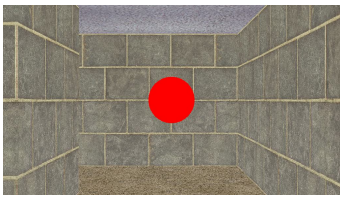
Prompt p_{17}

```
Environment feedback: Action executed
successfully. This is step 18. You are
allowed to take 82 more steps.
```

Action a_{17}

```
('turn', 1)
```

Observation o_{18}



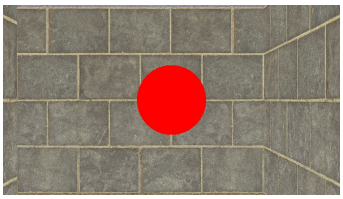
Prompt p_{18}

```
Environment feedback: Action executed
successfully. This is step 19. You are
allowed to take 81 more steps.
```

Action a_{18}

```
('move', 0)
```

Observation o_{19}



Prompt p_{19}

```
Environment feedback: Action executed
successfully. This is step 20. You are
allowed to take 80 more steps.
```

Action a_{19}

```
('stop', 'stop')
```

B.16 MENTAL ROTATION 2D (Back to Table of Contents)

Prompt p_0

```
You are solving a mental rotation task. Two panels appear side by side:
- Left: the original circular image.
- Right: the image has been rotated by a secret angle.
Your job is to undo that rotation and align the right image back to match the left.

Available actions:
1. 'rotate': Rotate the right image by an integer angle. Format: `('rotate', angle)` where
 angle is an integer between -180 and 180 degrees (positive is clockwise, negative is
counterclockwise).
2. 'stop': Submit your final adjustment. Format: `('stop', 'stop')`

Success: You succeed if your final adjustment undoes the secret rotation within ±5.0°.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To rotate clockwise: `('rotate', 45)`
- To rotate counterclockwise: `('rotate', -30)`
- To submit: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('rotate', 104)
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('stop', 'stop')
```

B.18 MENTAL ROTATION 3D (CUBE) (Back to Table of Contents)

Prompt p_0

```
You see two images side-by-side:
- Left: the original 3D object (target orientation)
- Right: the same object under an unknown rotation

Your goal is to rotate the right object back to the original target orientation.

Available actions:
1. 'rotate': Rotate the 3D object by Euler angles. Format: `('rotate', [dyaw, dpitch,
droll])` where each of dyaw, dpitch, droll is in degrees between -180 and 180.
2. 'stop': Submit your final rotation. Format: `('stop', 'stop')`

Note: Rotations are applied in the world (extrinsic) axes

Success: You succeed if you end up within 10.0 degrees of target orientation.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To rotate in yaw: `('rotate', [15.0, 0.0, 0.0])`
- To rotate in roll: `('rotate', [0.0, 0.0, -10.0])`
- To submit: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('rotate', [89.6, 48.5, 74.6])
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('stop', 'stop')
```

B.20 MENTAL ROTATION 3D (OBJAVERSE) (Back to Table of Contents)

Prompt p_0

```
You are solving a 3D mental rotation task. Two panels are shown:
- Left: the target view of the object (identity orientation).
- Right: the current view that you can rotate.
Your job is to rotate the object on the right so it matches the left.

Available actions:
1. 'rotate': Apply an incremental Euler rotation (yaw, pitch, roll). Format: `('rotate', [
d_roll, d_yaw, d_pitch])` with angles in degrees.
2. 'stop': Submit your final orientation. Format: `('stop', 'stop')`

Success: You succeed if the final rotation error is less than or equal to 5.0°.

Note: Rotations are applied in the world (extrinsic) axes

For example:
- Roll by 15°: `('rotate', [15, 0, 0])`
- To submit: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('rotate', [-3.7, -91.2, 95.9])
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

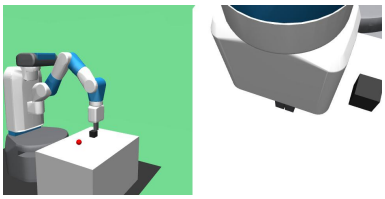
```
('stop', 'stop')
```

B.22 MuJoCo Fetch (Pick-and-Place) (Back to Table of Contents)

Prompt p_0

```
You are controlling a robotic arm to pick up a grey cube and place it at the target 3D
position. The robot end-effector (gripper) and the cube are visible in the image. The left
 image is a front view of the robot, and the right image moving camera attached to the
gripper. Your goal is to grasp the cube and move it to the red target marker. Each action
moves the end-effector by a fixed step size in the specified direction.

Available actions:
1. 'move': Move the end-effector in 3D space. Format: `('move', [x, y, z])` where each
element is -1, 0, or 1:
   - x: -1=left, 0=no change, 1=right
   - y: -1=backward, 0=no change, 1=forward
   - z: -1=down, 0=no change, 1=up
2. 'gripper': Control the gripper (sticky behavior). Format: `('gripper', value)` where
value is:
   - 0: open the gripper
   - 1: close the gripper
Note: Gripper state is sticky - once set, it continues until changed.
3. 'stop': End the pick and place session. Format: `('stop', 'stop')`

Success: You succeed when you pick up the cube and place it at the target position (red
marker).

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move right and up: `('move', [1, 0, 1])`
- To move forward: `('move', [0, 1, 0])`
- To move down and left: `('move', [-1, 0, -1])`
- To stay in place: `('move', [0, 0, 0])`
- To open gripper: `('gripper', 0)`
- To close gripper: `('gripper', 1)`
- To stop: `('stop', 'stop')`

Gripper Behavior: The gripper has sticky behavior - once you open or close it,

Feedback: Action executed successfully.
This is step 1. You are allowed to take 99 more steps.
```

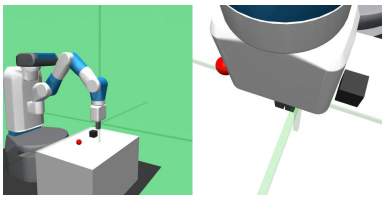
Observation o_0



Action a_0

```
('move', [0, 1, 0])
```

Observation o_1



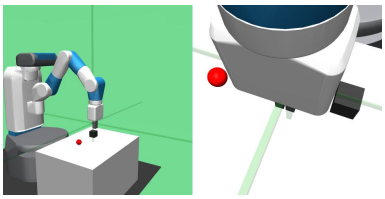
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', [-1, 0, 0])
```

Observation o_2



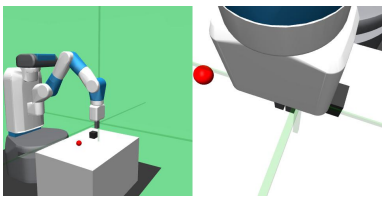
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', [0, 1, 0])
```

Observation o_3



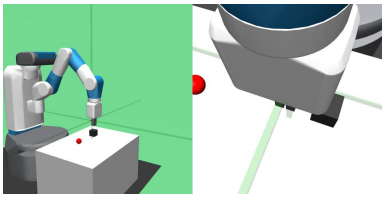
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', [-1, 0, 0])
```

Observation o_4



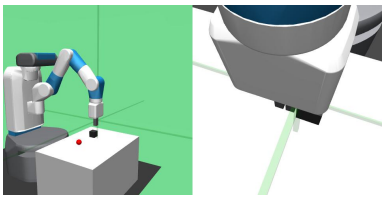
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('move', [0, 1, 0])
```

Observation o_5



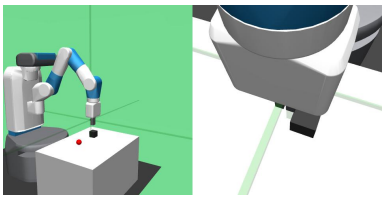
Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('move', [-1, 0, 0])
```

Observation o_6



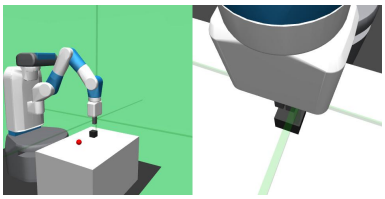
Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('move', [0, 1, 0])
```

Observation o_7



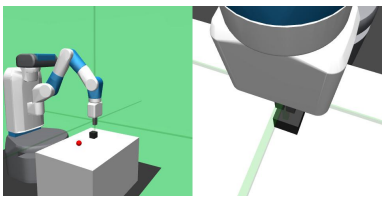
Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

```
('move', [0, 0, 0])
```

Observation o_8



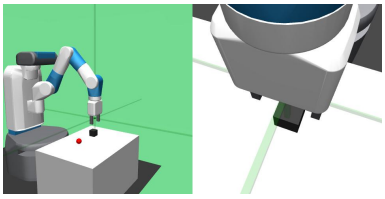
Prompt p_8

```
Environment feedback: Action executed
successfully. This is step 9. You are
allowed to take 91 more steps.
```

Action a_8

```
('gripper', 0)
```

Observation o_9



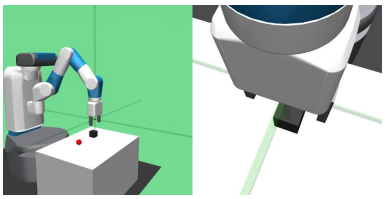
Prompt p_9

```
Environment feedback: Action executed
successfully. This is step 10. You are
allowed to take 90 more steps.
```

Action a_9

```
('move', [0, 0, 0])
```

Observation o_{10}



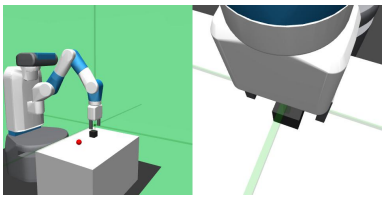
Prompt p_{10}

```
Environment feedback: Action executed
successfully. This is step 11. You are
allowed to take 89 more steps.
```

Action a_{10}

```
('move', [0, 0, -1])
```

Observation o_{11}



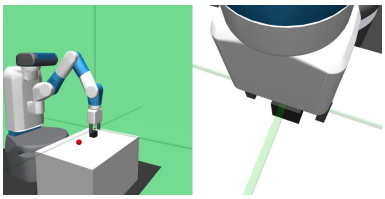
Prompt p_{11}

```
Environment feedback: Action executed
successfully. This is step 12. You are
allowed to take 88 more steps.
```

Action a_{11}

```
('move', [0, 0, -1])
```

Observation o_{12}



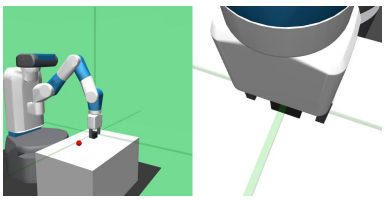
Prompt p_{12}

```
Environment feedback: Action executed
successfully. This is step 13. You are
allowed to take 87 more steps.
```

Action a_{12}

```
('move', [0, 0, -1])
```

Observation o_{13}



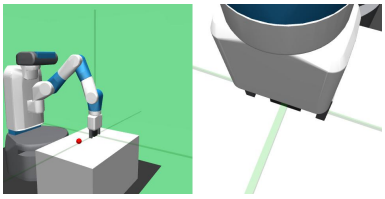
Prompt p_{13}

```
Environment feedback: Action executed
successfully. This is step 14. You are
allowed to take 86 more steps.
```

Action a_{13}

```
('move', [0, 0, -1])
```

Observation o_{14}



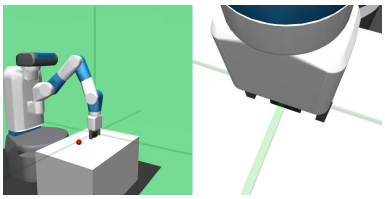
Prompt p_{14}

```
Environment feedback: Action executed
successfully. This is step 15. You are
allowed to take 85 more steps.
```

Action a_{14}

```
('move', [0, 0, 0])
```

Observation o_{15}



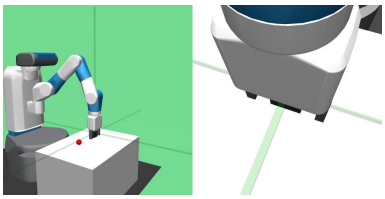
Prompt p_{15}

```
Environment feedback: Action executed
successfully. This is step 16. You are
allowed to take 84 more steps.
```

Action a_{15}

```
('gripper', 1)
```

Observation o_{16}



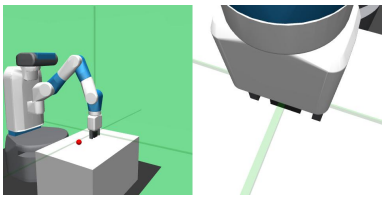
Prompt p_{16}

```
Environment feedback: Action executed
successfully. This is step 17. You are
allowed to take 83 more steps.
```

Action a_{16}

```
('gripper', 1)
```

Observation o_{17}



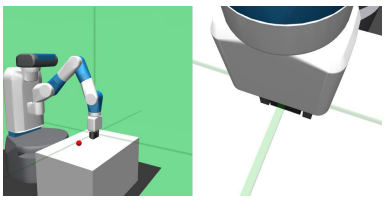
Prompt p_{17}

```
Environment feedback: Action executed
successfully. This is step 18. You are
allowed to take 82 more steps.
```

Action a_{17}

```
('gripper', 1)
```

Observation o_{18}



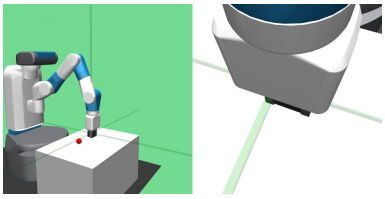
Prompt p_{18}

```
Environment feedback: Action executed
successfully. This is step 19. You are
allowed to take 81 more steps.
```

Action a_{18}

```
('move', [0, -1, 0])
```

Observation o_{19}



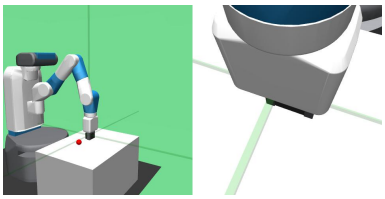
Prompt p_{19}

```
Environment feedback: Action executed
successfully. This is step 20. You are
allowed to take 80 more steps.
```

Action a_{19}

```
('move', [0, -1, 0])
```

Observation o_{20}



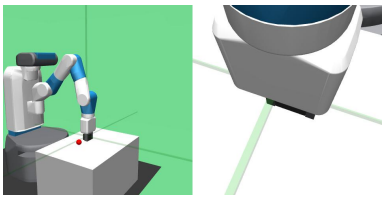
Prompt p_{20}

```
Environment feedback: Action executed
successfully. This is step 21. You are
allowed to take 79 more steps.
```

Action a_{20}

```
('move', [0, -1, 0])
```

Observation o_{21}



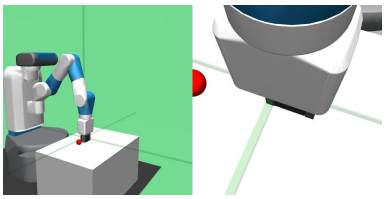
Prompt p_{21}

```
Environment feedback: Action executed
successfully. This is step 22. You are
allowed to take 78 more steps.
```

Action a_{21}

```
('move', [0, -1, 0])
```

Observation o_{22}



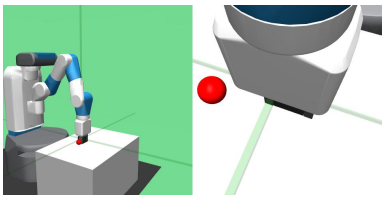
Prompt p_{22}

```
Environment feedback: Action executed
successfully. This is step 23. You are
allowed to take 77 more steps.
```

Action a_{22}

```
('move', [0, -1, 0])
```

Observation o_{23}



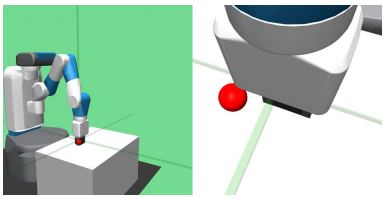
Prompt p_{23}

```
Environment feedback: Action executed
successfully. This is step 24. You are
allowed to take 76 more steps.
```

Action a_{23}

```
('move', [0, -1, 0])
```

Observation o_{24}



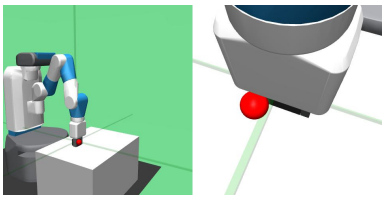
Prompt p_{24}

```
Environment feedback: Action executed
successfully. This is step 25. You are
allowed to take 75 more steps.
```

Action a_{24}

```
('move', [0, -1, 0])
```

Observation o_{25}



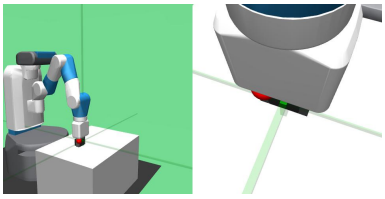
Prompt p_{25}

```
Environment feedback: Action executed
successfully. This is step 26. You are
allowed to take 74 more steps.
```

Action a_{25}

```
('move', [1, 0, 0])
```

Observation o_{26}



Prompt p_{26}

```
Environment feedback: Action executed
successfully. This is step 27. You are
allowed to take 73 more steps.
```

Action a_{26}

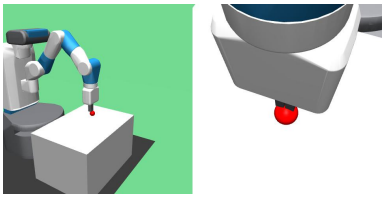
```
('stop', 'stop')
```

B.24 MuJoCo Fetch (Reach) (Back to Table of Contents)

Prompt p_0

```
You are controlling a robotic arm to reach a target 3D position. The left image is a front
 view of the robot, and the right image moving camera attached to the gripper. Your goal
is to move it to the red target marker. Each action moves the end-effector by a fixed step
 size in the specified direction.

Available actions:
1. 'move': Move the end-effector in 3D space. Format: `('move', [x, y, z])` where each
element is -1, 0, or 1:
   - x: -1=left, 0=no change, 1=right
   - y: -1=backward, 0=no change, 1=forward
   - z: -1=down, 0=no change, 1=up
2. 'stop': End the reaching session. Format: `('stop', 'stop')`

Success: You succeed when the end-effector reaches the target position (red marker).

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move right and up: `('move', [1, 0, 1])`
- To move forward: `('move', [0, 1, 0])`
- To move down and left: `('move', [-1, 0, -1])`
- To stay in place: `('move', [0, 0, 0])`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

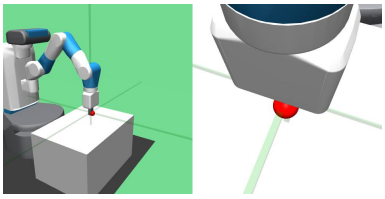
Observation o_0



Action a_0

```
('move', [0, 0, -1])
```

Observation o_1



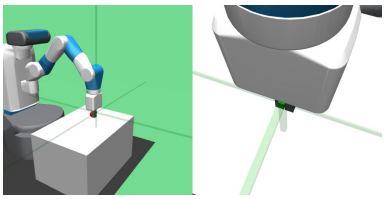
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', [1, 0, 0])
```

Observation o_2



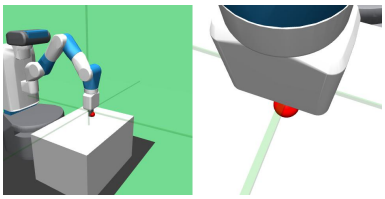
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', [-1, 0, 0])
```

Observation o_3



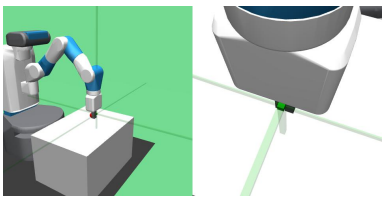
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', [1, 0, 0])
```

Observation o_4



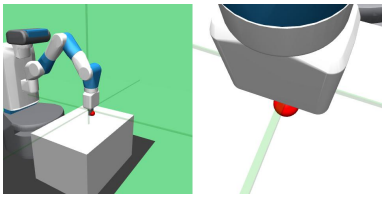
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('move', [-1, 0, 0])
```

Observation o_5



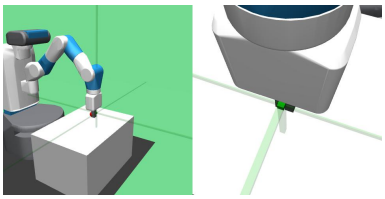
Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('move', [1, 0, 0])
```

Observation o_6



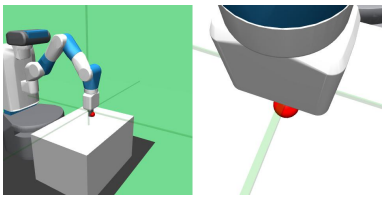
Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('move', [-1, 0, 0])
```

Observation o_7



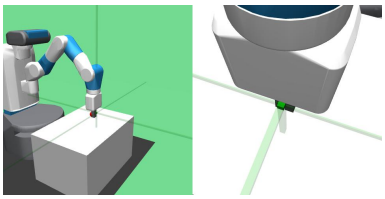
Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

```
('move', [1, 0, 0])
```

Observation o_8



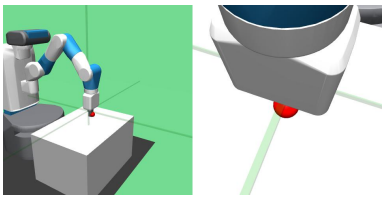
Prompt p_8

```
Environment feedback: Action executed
successfully. This is step 9. You are
allowed to take 91 more steps.
```

Action a_8

```
('move', [-1, 0, 0])
```

Observation o_9



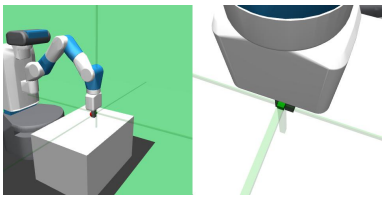
Prompt p_9

```
Environment feedback: Action executed
successfully. This is step 10. You are
allowed to take 90 more steps.
```

Action a_9

```
('move', [1, 0, 0])
```

Observation o_{10}



Prompt p_{10}

```
Environment feedback: Action executed
successfully. This is step 11. You are
allowed to take 89 more steps.
```

Action a_{10}

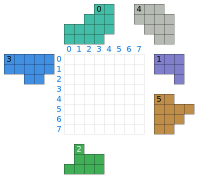
```
('stop', 'stop')
```

B.26 PATCH REASSEMBLY (Back to Table of Contents)

Prompt p_0

```
You are solving a 8×8 patch-reassembly puzzle. There are 6 irregular pieces parked around
 the board. Your goal is to place them so the grid is completely filled--no gaps, no
overlaps.

Available actions:
1. 'place': Snap a patch onto the grid by aligning its anchor (the cell that shows the
patch's ID number) with a chosen board coordinate. Format: `('place', (patch_id, row, col)
)` where patch_id ∈ [0, 5], row ∈ [0, 7], col ∈ [0, 7].
2. 'remove': Return a patch to its parking slot. Format: `('remove', patch_id)` where
patch_id ∈ [0, 5].
3. 'stop': End the episode and submit your solution. Format: `('stop', 'stop')`

Success: You succeed if the grid is completely filled with no gaps or overlaps.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To place patch 0 at (2, 3): `('place', (0, 2, 3))`
- To remove patch 0: `('remove', 0)`
- To finish: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

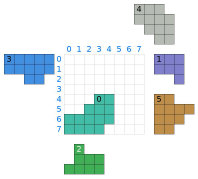
Observation o_0



Action a_0

```
('place', (0, 4, 3))
```

Observation o_1



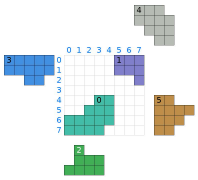
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('place', (1, 0, 5))
```

Observation o_2



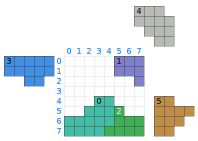
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('place', (2, 5, 5))
```

Observation o_3



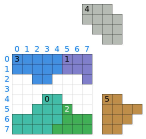
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('place', (3, 0, 0))
```

Observation o_4



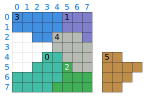
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('place', (4, 2, 4))
```

Observation o_5



Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('place', (5, 2, 0))
```

Observation o_6

Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('stop', 'stop')
```

B.28 REFERRING DOT-POINTING (Back to Table of Contents)

Prompt p_0

```
Your goal is to point to 'plant with brown pot' in the image.

Available actions:
1. 'mark': Mark a point at the specified coordinates. Format: `('mark', (x_norm, y_norm))`
 where x_norm, y_norm are normalized coordinates between 0.0 and 1.0, which map to pixel x
 = round(x_norm*(500-1)), y = round(y_norm*(375-1)).
2. 'stop': Submit your current mark as final answer. Format: `('stop', 'stop')`

Please respond with exactly one action and its arguments in the specified format. For
example:
- To mark center: `('mark', (0.5, 0.5))`
- To submit: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('mark', (0.3578, 0.5295))
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('mark', (0.3530, 0.4949))
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('mark', (0.3481, 0.4602))
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

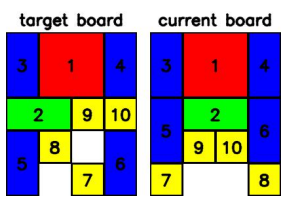
```
('stop', 'stop')
```

B.30 SLIDING BLOCK (Back to Table of Contents)

Prompt p_0

```
You are solving a 5x4 sliding block puzzle (Klotski). The puzzle consists of rectangular
blocks that can be moved in four directions. Your goal is to rearrange the blocks to match
 the target configuration.You see two boards side-by-side:
- Left: the target configuration
- Right: the current configuration that you need to rearrange


Available actions:
1. 'move': Move a block in one of four directions. Format: `('move', (block_id, direction)
)` where:
   - block_id is the number of the block to move (1-10)
   - direction is 0=up, 1=right, 2=down, 3=left
2. 'stop': End the puzzle solving session. Format: `('stop', 'stop')`

Success: You succeed if all blocks are in their target positions when you stop.

Please respond with exactly one action and its arguments in the specified format. For
example:
- To move block 1 up: `('move', (1, 0))`
- To move block 2 right: `('move', (2, 1))`
- To move block 3 down: `('move', (3, 2))`
- To move block 4 left: `('move', (4, 3))`
- To stop: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

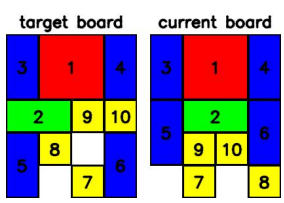
Observation o_0



Action a_0

```
('move', (7, 1))
```

Observation o_1



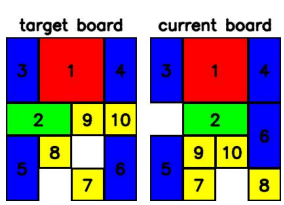
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('move', (5, 2))
```

Observation o_2



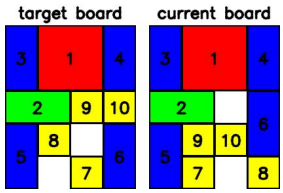
Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('move', (2, 3))
```

Observation o_3



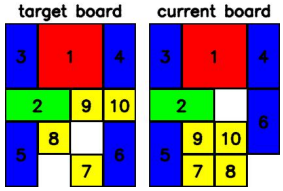
Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('move', (8, 3))
```

Observation o_4



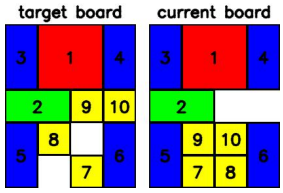
Prompt p_4

```
Environment feedback: Action executed
successfully. This is step 5. You are
allowed to take 95 more steps.
```

Action a_4

```
('move', (6, 2))
```

Observation o_5



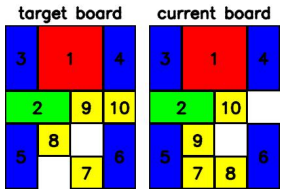
Prompt p_5

```
Environment feedback: Action executed
successfully. This is step 6. You are
allowed to take 94 more steps.
```

Action a_5

```
('move', (10, 0))
```

Observation o_6



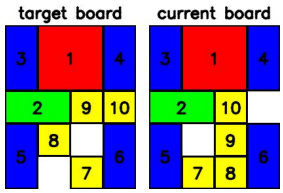
Prompt p_6

```
Environment feedback: Action executed
successfully. This is step 7. You are
allowed to take 93 more steps.
```

Action a_6

```
('move', (9, 1))
```

Observation o_7



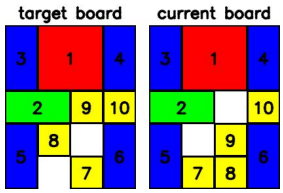
Prompt p_7

```
Environment feedback: Action executed
successfully. This is step 8. You are
allowed to take 92 more steps.
```

Action a_7

```
('move', (10, 1))
```

Observation o_8



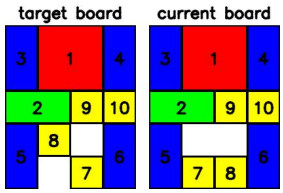
Prompt p_8

```
Environment feedback: Action executed
successfully. This is step 9. You are
allowed to take 91 more steps.
```

Action a_8

```
('move', (9, 0))
```

Observation o_9



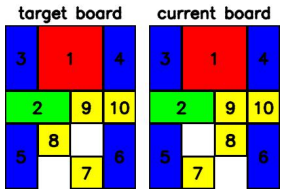
Prompt p_9

```
Environment feedback: Action executed
successfully. This is step 10. You are
allowed to take 90 more steps.
```

Action a_9

```
('move', (8, 0))
```

Observation o_{10}



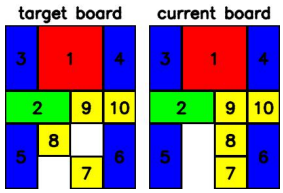
Prompt p_{10}

```
Environment feedback: Action executed
successfully. This is step 11. You are
allowed to take 89 more steps.
```

Action a_{10}

```
('move', (7, 1))
```

Observation o_{11}



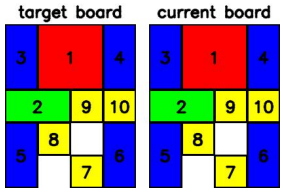
Prompt p_{11}

```
Environment feedback: Action executed
successfully. This is step 12. You are
allowed to take 88 more steps.
```

Action a_{11}

```
('move', (8, 3))
```

Observation o_{12}



Prompt p_{12}

```
Environment feedback: Action executed
successfully. This is step 13. You are
allowed to take 87 more steps.
```

Action a_{12}

```
('stop', 'stop')
```

B.32 VIDEO UNSHUFFLE (Back to Table of Contents)

Prompt p_0

```
You are given 5 video frames extracted from a short clip, laid out left->right in a
shuffled order and labeled 1 through 5. The action being performed in the video is: '
pushing a chair with a hand'.

Your goal is to rearrange the frames so they appear in their original chronological order
from left to right. Pay attention to the temporal progression of the action described
above.

Available actions:
1. 'swap': Swap two frames by specifying their 1-based positions. Format: `('swap', (i, j)
)`
2. 'reorder': Provide the complete ordering in one shot. Format: `('reorder', [1, 2, ...,
5])` where the list gives the desired left-to-right arrangement.
3. 'stop': Finish and lock in the current ordering. Format: `('stop', 'stop')`

Please respond with exactly one action and its arguments in the specified format. For
example:
- Swap the first and third frames: `('swap', (1, 3))`
- Reorder all frames: `('reorder', [2, 1, 4, 3, ..., 5])`
- Finalize the ordering: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('swap', (1, 4))
```

Observation o_1

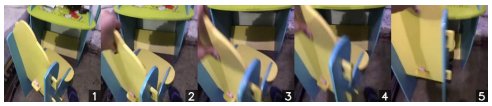
Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('swap', (3, 5))
```

Observation o_2



Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('stop', 'stop')
```

B.34 ZOOM-IN PUZZLE (Back to Table of Contents)

Prompt p_0

```
You are given an original image and 5 zoomed-in views laid out left→right. Your goal is
to rearrange them so they are ordered from *least* to *most* zoomed.

Available actions:
1. 'swap': Swap two zoom-tiles by specifying their 1-based positions. Format: `('swap', (i
, j))` where i and j are 1-based positions.
2. 'reorder': Provide the complete ordering in one shot. Format: `('reorder', [1, 2, ...,
5])` where the list gives the desired left-to-right arrangement.
3. 'stop': Finish and lock in the current ordering. Format: `('stop', 'stop')`

Success: Arrange the views from least to most zoomed (ascending zoom level order).

Please respond with exactly one action and its arguments in the specified format. For
example:
- To swap views: `('swap', (1, 3))`
- To reorder all: `('reorder', [1, 3, 2, 4])`
- To finalize: `('stop', 'stop')`
This is step 1. You are allowed to take 99 more steps.
```

Observation o_0

Action a_0

```
('swap', (1, 4))
```

Observation o_1

Prompt p_1

```
Environment feedback: Action executed
successfully. This is step 2. You are
allowed to take 98 more steps.
```

Action a_1

```
('swap', (2, 3))
```

Observation o_2

Prompt p_2

```
Environment feedback: Action executed
successfully. This is step 3. You are
allowed to take 97 more steps.
```

Action a_2

```
('swap', (3, 4))
```

Observation o_3

Prompt p_3

```
Environment feedback: Action executed
successfully. This is step 4. You are
allowed to take 96 more steps.
```

Action a_3

```
('stop','stop')
```