SpaceTools: Tool-Augmented Spatial Reasoning via Double Interactive RL

Siyi Chen[1,3] * Mikaela Angelina Uy[3] Chan Hee Song[2] Faisal Ladhak[3] Adithyavairavan Murali[3]
Qing Qu[1] Stan Birchfield[3] Valts Blukis[‡,3] Jonathan Tremblay[‡,3]

[1]University of Michigan [2]The Ohio State University [3]NVIDIA
[‡] Project Leads

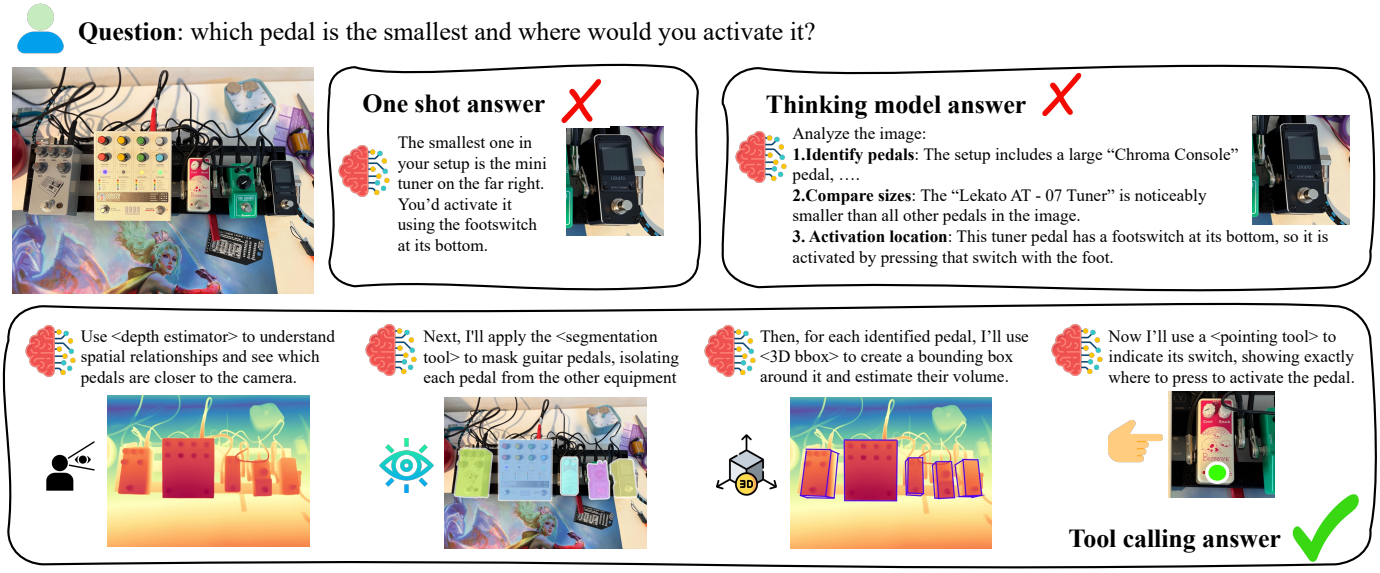


Figure 1. SpaceTools uses multiple computer vision tools to solve complex problems. Shown here is a motivating example.

Abstract

Vision Language Models (VLMs) demonstrate strong qualitative visual understanding, but struggle with metrically precise spatial reasoning required for embodied applications. The agentic paradigm promises that VLMs can use a wide variety of tools that could augment these capabilities, such as depth estimators, segmentation models, and pose estimators. Yet it remains an open challenge how to realize this vision without solely relying on handcrafted prompting strategies or enforcing fixed, predefined tool pipelines that limit VLMs' ability to discover optimal tool-use patterns. Reinforcement Learning could overcome this gap, but has so far been limited to reasoning with a single visual tool due to the large search space in multi-tool reasoning. We introduce Double Interactive Reinforcement Learning (DIRL), a two-phase training framework where VLMs learn to coordinate multiple tools through interactive exploration and feedback. In the teaching phase, we combine demonstrations from a single tool specialist trained via interactive RL with traces from a frontier model using all tools. In the exploration phase, the model further refines multi-tool coordination through continued RL. Our model, SpaceTools, with tool-augmented spatial reasoning ability, achieves state-of-the-art performance on spatial understanding benchmarks (RoboSpatial-Home, BLINK, BOP-ASK) and demonstrates reliable real-world manipulation using a 7-DOF robot as a tool. DIRL provides substantial improvements over the vanilla SFT (+12% on RoboSpatial) and RL (+16% on RoboSpatial) baselines. Project page: https://spacetools.github.io/.

1. Introduction

Spatial reasoning—the ability to understand geometric relationships between objects and their environment—is an important capability for vision-language models (VLMs). It

*Work Done during an internship at NVIDIA.

enables models to answer geometric questions, such as relative positions, spatial configurations, and physical affordances, which is vital to support the integration of VLMs into embodied systems, such as robots. While recent VLMs [3, 27, 28, 46] have achieved strong performance on open-ended visual questions, their ability to do spatial understanding remains an active field of research [22, 40, 71, 77], particularly in settings that require diverse multi-step reasoning intertwined with precise geometric perception and 3D awareness (see Figure 1). These challenges are amplified in robotics, where perception must seamlessly translate into decision-making and physical action [24].

The conventional approach to teach VLMs new capabilities involves fine-tuning on task-specific datasets [8, 12, 20, 32, 49, 83], an approach limited by the need for large-scale annotations and extensive data engineering. We present a scalable alternative: we empower VLMs to use tools, that is, to call computer vision and robotics modules when needed, and use their outputs to aid in solving the spatial reasoning task. Such tool use provides access to precise measurements and intermediate geometric representations, can leverage computer vision models from VLM-incompatible settings (*e.g.*, dense prediction), and allows combining the strengths of heterogeneous models to augment base-model capability. Recently, ViGoRL [50] demonstrated that reinforcement learning can enable a VLM to learn grounded reasoning with a single visual tool, namely a cropping operation, showing the promise of interactive RL for tool use. However, naïve application of RL to many tools creates a prohibitively large search space where exploration fails to discover effective policies.

To address this gap, we introduce Double Interactive Reinforcement Learning (DIRL), a two-phase framework where interactive RL is applied twice. The key insight is that RL with a pointing tool is tractable and teaches grounding, while multi-tool RL can refine diverse reasoning, but requires good initialization for stable learning. DIRL uses a two-phase training scheme with a teaching phase followed by an exploration phase. In the teaching phase, the model is trained with Supervised Fine-Tuning (SFT) on the basics of tool usage—method signatures, outputs, and information flow using a mix of single-tool Interactive Reinforcement Learning (IRL) traces and multi-tool demonstrations. In the exploration phase, we apply Interactive Reinforcement Learning (IRL) with the full toolset, enabling the model to refine tool coordination for spatial reasoning tasks.

Unlike prior work, DIRL allows the model to call tools interactively during training, instead of relying on fixed pipelines or precomputed contexts (Table 1), enabling this behavior at scale requires addressing a key systems challenge: how to efficiently serve diverse, compute-intensive tools during interactive training. To address this, we develop *Toolshed*, a platform which hosts computationally intensive computer vision tools such as SAM2 [48], Depth Pro [6], RoboRefer [83], and GraspGen [42] as rapid on-demand services during training, decoupling tool resource management from RL or inference workloads, and achieving high tool throughput and utilization. By incorporating real and stochastic tool outputs into the learning loop, DIRL exposes models to actual tool behavior, encouraging reasoning about tool reliability and discovering improved ways to query the tools.

Table 1. Comparison of related work for training supervision and tool-call interactivity during training. '-' indicates that only a single tool is used.

Method	SFT	RL	Use tools	Non-fixed tool pipeline	Interactive tool call
SpatialVLM [10]	✓	×	×	×	×
RoboRefer [83]	✓	✓	×	×	×
SpatialPIN [35]	×	×	✓	×	×
APC [25]	×	×	✓	×	×
ViGoRL [50]	×	✓	-	✓	×
SpatialReasoner [37]	×	✓	×	×	×
TIGeR [18]	✓	✓	✓	✓	×
SpaceTools (ours)	✓	✓	✓	✓	✓

We conduct extensive experiments on a diverse set of spatial reasoning problems, such as determining object-location fit, estimating distances between items, reasoning about occlusions and orientations, pose estimation, and predicting grasping affordances. Our trained model, *SpaceTools*, achieves state-of-the-art performance across multiple spatial reasoning benchmarks, including RoboSpatial-Home [55], BLINK [16], RefSpatial [83], CVBench [83], and BOP-ASK [4]. By integrating a real robot as a tool, SpaceTools completes pick-and-place tasks with an 86% success rate, demonstrating effective transfer from spatial reasoning to embodied control and outperforming frontier models equipped with the same tools. In summary, our contributions are:

1. **DIRL:** a novel training paradigm that enables interactive training with a large set of tools.
2. **Toolshed:** an interactive platform for hosting diverse computer vision tools, to be open-sourced.
3. **SpaceTools:** A VLM trained for spatial reasoning via interactive multi-tool use, which achieves state-of-the-art results across spatial reasoning benchmarks and performs robot control via alternating perception and action tool calls.

2. Related Work

Spatial Reasoning with VLMs. Spatial reasoning with VLMs [3, 27–29, 46] refers to understanding geometric relationships among objects and their environment [2, 16, 26, 47, 54, 58, 69]. Recent progress shows that VLMs can increasingly support robots in perceiving and inter-

acting with the physical world [7, 55, 83]. However, VLM spatial reasoning remains insufficient for real-world robotic demands, where multi-step reasoning, precise geometric understanding, and strong 3D awareness are required [44, 71, 77]. Conventional approaches teach VLMs spatial understanding by fine-tuning on task-specific question-answering datasets [9, 12, 31, 49, 50, 55, 67, 83]. Yet these methods require large-scale data collection and architecture modifications even to introduce a single low-level perceptual capability such as depth [8], pointing [14, 55, 83], and 3D-awareness [36, 38]. Instead of baking all perceptual skills into the model, we propose to enable VLMs to invoke external computer vision and robotics tools as needed, allowing them to solve spatial reasoning tasks and perform real-world manipulation.

Tool-augmented Reasoning. Tool-augmented reasoning aims to enrich model capabilities by supplying additional information from external modules [11, 21, 30, 76]. Typical applications include integrating search engines [11, 15, 21], calculators [43, 79], or code executors [57, 65] into LLMs, and vision tools for VLMs [19, 39, 73]. In the context of spatial reasoning, the community has explored equipping VLMs with vision tools during intermediate reasoning steps. However, most approaches rely on handcrafted prompting strategies [17, 19, 41, 70] or enforce a fixed, predefined tool pipeline [25, 35] in a training-free way, which limits their ability to handle diverse, precise, and 3D-aware reasoning required for robotics. TIGeR [18] is a concurrent work we learned of during the preparation of this manuscript. They focus more on problem-solving via code generation and not interactive learning, deriving their supervision from a predefined synthetic tool pipeline with large-model-based rewriting. In contrast, we enable the model to learn to coordinate a diverse set of vision and robotic tools through both teacher demonstrations involving real tool interactions and self-exploration enabled by interactive RL.

Reinforcement Learning for Reasoning. Reinforcement learning (RL) has been widely applied to enhance the reasoning capabilities of LLMs or VLMs on verifiable tasks such as math [51], coding [13, 62], and general visual question answering (VQA) [72, 75, 78, 80]. Recent work further explores RL for spatial reasoning, enabling models to produce interpretable or grounded reasoning [23, 33, 52, 67, 68]. Some works adopt RL to stengthen chain-of-thought style reasoning before predicting answers [44, 64], while others focus on teaching grounded spatial understanding [50, 67, 82]. Although prior works demonstrate that RL can teach spatial reasoning with use of a single light-weight tool (*e.g.*, cropping), scaling to multiple heterogeneous tools poses a fundamental challenge: with 10+ tools, the action space grows combinatorially, causing naive RL exploration to fail. Our training paradigm decomposes the problem into progressive and tractable phases, enabling the model to learn effective coordination strategies with diverse tools.

Algorithm 1 Spatial Reasoning with Tools

Require: VLM π_θ, User Query $\mathcal{I}$, Max Turns $T_{\max}$
Ensure: Answer A_{final}

$t \leftarrow 1, h_1 \leftarrow \mathcal{I}$ ▷ Initialize dialogue history
while $t \leq T_{\max}$ **do** ▷ t is a counter
  $a_t \leftarrow \pi_\theta(h_t)$ ▷ Generate VLM response
  $h_{t+1} \leftarrow h_t \oplus a_t$
  if <answer> detected in a_t **then**
    $A_{\text{final}} \leftarrow \text{Parse}(a_t, \texttt{<answer>}, \texttt{</answer>})$
    break ▷ Final turn: task is complete
  else if <tool_call> detected in a_t **then**
    $\mathcal{Q}_{\text{tools}} \leftarrow \text{Parse}(a_t, \texttt{<tool\_call>}, \texttt{</tool\_call>})$
    for each $q \in \mathcal{Q}_{\text{tools}}$ **do**
      $h_{t+1} \leftarrow h_{t+1} \oplus \text{CallTool}(q)$ ▷ Execute tool
    end for
  end if
  $t \leftarrow t + 1$
end while
return A_{final}

3. Problem Formulation

We formulate spatial reasoning as a sequential decision-making problem where a VLM policy π_θ interacts with external tools $\mathcal{Q}_{tools}$ to respond to a user query $\mathcal{I}$, which may consist of an image-text pair or a robotic manipulation task. The model can reason and interact with tools in multiple turns until it produces a final answer A_{final} or reaches a maximum of $T_{\max}$ interaction steps.

At each step t, the VLM receives the historical context h_t, which contains the full dialogue between the user, the VLM, and the tools (initialized as $h_1 = \mathcal{I}$). The model then generates a response a_t according to its policy: If a_t includes tool calls, tools are executed sequentially. Their outputs, together with a_t, are appended to the historical context h_t to form h_{t+1}. The updated context is then used to generate the next-step response.

The complete workflow is outlined in Algorithm 1. The model is required to follow a structured conversational format: reasoning is enclosed within <think> tags, tool calls within <tool_call> tags, and the final answer within <answer> tags.

The goal of this work is to **learn a policy** π_θ that addresses user queries through multi-turn interaction with vision and robotic tools. To achieve this, we propose a new training paradigm accompanied by a novel tool platform.

4. Double Interactive Reinforcement Learning

Training a VLM to reason and act through external tools benefits from both teacher-guided supervision and interactive exploration. We introduce **Double Interactive Rein-**

forcement Learning (DIRL), a two-stage framework that unifies these two forms of learning. Enabling DIRL requires seamless communication between the VLM and a diverse set of vision and robotic tools during both data collection and training. We solve this challenging problem by designing **Toolshed**, a distributed infrastructure that manages large-scale tool interaction.

4.1. DIRL

We introduce a new training paradigm that enables VLMs to effectively use multiple tools. Our approach is motivated by two observations: (i) naïvely applying IRL (interactive RL) to all tools at once creates an extremely large search space, resulting in weak optimization signals, and (ii) pure SFT on tool-interaction traces yields models that struggle to coordinate with tools effectively or to go beyond the training traces. Our method, DIRL, addresses these limitations and improves the model's ability to integrate and sequence multiple tools effectively. DIRL is composed of two phases, a teaching phase and an exploration phase.

Teaching phase. This phase establishes basic tool use capabilities without the exploration challenges of full multi-tool RL. We build the teaching dataset from two complementary sources. First, we apply IRL to train the base model to use a single pointing tool for spatial reasoning tasks (*e.g.*, spatial relationship, spatial compatibility, and relative depth are trained together). This constrained search space, allows IRL to reliably converge and produce competent behavior. The resulting *IRL-trained teacher* is then used to generate supervised demonstrations of grounded reasoning for the first portion of our teaching dataset. Second, we prompt a *universal teacher*, which is a frontier model, to solve spatial reasoning and robot manipulation tasks with the full set of tools (*e.g.*, pointing, segmentation, 3D bbox, *etc.*), retaining only trajectories that lead to correct solutions. Finally, we combine both datasets—one part generated by the IRL-trained teacher and three parts from the universal teacher—to form the complete teaching dataset. We then perform supervised fine-tuning (SFT) on the base model, yielding a policy with initial tool-usage behaviors.

Exploration phase. This phase refines multi-tool coordination through interactive exploration. We resume IRL training on all tasks from the SFT-initialized policy with access to all available tools, allowing the model to enhance tool chaining strategies. The strong initialization prevents exploration collapse in the large multi-tool action space, while interactive feedback offers additional refinement of tool coordinations. These two rounds of IRL give our method its name, as DIRL involves two IRL phases—one for teaching and one for exploration.

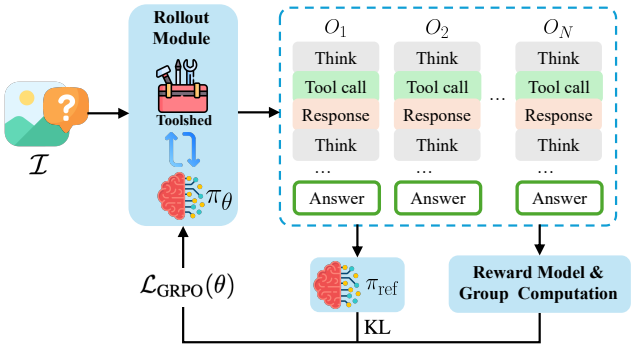


Figure 2. Interactive reinforcement learning (IRL) with Toolshed. The rollout module executes multi-turn trajectories under policy π_θ, alternating between reasoning and tool use before answering. Task rewards are aggregated and used to update π_θ via GRPO with KL regularization against π_{ref}.

Policy Update. We employ Group Relative Policy Optimization (GRPO) [51] as our RL training algorithm, as visualized in Figure 2. For each input $\mathcal{I}$, in total N rollout procedures are launched asynchronously under the current policy π_θ. Each rollout proceeds as Algorithm 1, generating in total N multi-turn rollouts O_1, O_2, ..., O_N. Their rewards are calculated as r_1, r_2, ..., r_N, and the policy is updated by optimizing the GRPO objective $\mathcal{L}_{\text{GRPO}}(r_1, \ldots, r_N)$, described in full in the Appendix.

4.2. Toolshed

Our method, DIRL, assumes access to an efficient system for invoking tools during training. In prior work, tool usage is either tightly coupled with the training loop, limiting to simple tools (*e.g.*, cropping [50]), or in case of text-only tools (*e.g.*, search [21]), highly decoupled via web APIs that lack the throughput needed for VLM interactive learning with images. Others side-step the issue altogether by using pre-computed tool outputs [18], preventing models from learning interactive, state-dependent tool use.

We introduce Toolshed, a scalable framework for deploying multiple compute-heavy tools alongside training or inference workloads that mitigates these bottlenecks through: (1) resource and environment isolation for each tool instance; (2) decoupled scaling and execution from the policy's main inference loop; and (3) asynchronous parallel workers per tool, allowing scaling tool resources independently from training resources. Toolshed hosts modular vision tools (*e.g.*, segmentation, pointing, monocular depth, 3D box fitting, grasp prediction, and various image operations) and robotic tools (*e.g.*, image capture, grasp execution, object placement). Implementation details and complete tool APIs are provided in the Appendix.

Table 2. Performance comparison across spatial reasoning benchmarks. All values are normalized accuracy (%). **Bold** indicates the best performance within each column, and underline denotes the second-best result. Values of 0 indicate the model either fails to produce valid responses, outputs answers in wrong formats, or produces entirely incorrect predictions, reflecting an inability to handle that task type.

Model	RoboSpatial			BLINK	RefSpatial	CVBench		BOP-ASK		
	VQA	Vacant	Overall	Depth		2D Rel.	3D Depth	Pose	Grasp-MACE	Grasp-SR
Proprietary Models										
Claude Sonnet 4.5	75.44	23.77	57.43	78.23	7.49	89.85	78.50	1.67	40.12	48.33
GPT-4o	61.61	25.10	48.88	63.71	8.48	88.77	75.50	0.00	5.50	1.67
GPT-5	76.50	22.17	58.39	66.13	23.10	95.54	91.33	9.03	39.59	41.67
Gemini-ER 1.5	79.30	31.10	62.50	69.23	41.72	95.54	90.50	0.00	30.06	23.33
General Open-Source Models										
LLaVA-NeXT-8B	69.31	0.00	45.15	53.23	0.78	72.15	73.67	0.00	5.04	1.67
Qwen2.5-VL-32B	61.84	3.28	41.43	70.16	7.28	90.46	86.67	0.00	29.86	23.33
Qwen2.5-VL-3B	53.07	0.00	35.71	70.98	0.00	70.62	65.33	0.00	6.06	0.00
Spatial VLMs										
SpaceLLaVA-13B	61.00	2.50	40.61	51.61	3.25	61.08	62.83	0.00	0.00	0.00
RoboPoint-13B	70.18	19.70	52.58	54.84	15.59	74.00	76.50	0.00	0.00	0.00
Molmo-7B	39.92	0.82	26.29	54.03	0.00	72.15	73.33	0.00	36.74	18.33
RoboBrain2.0-7B	59.64	44.35	54.31	84.68	32.50	87.23	90.00	0.00	0.00	0.00
RoboRefer-8B-SFT	58.33	**61.48**	59.43	88.71	48.37	**96.31**	**96.50**	0.00	0.00	0.00
Tool-free Fine-tuning										
Qwen2.5-VL-3B-Tool-free SFT	66.66	41.80	58.00	80.65	20.22	91.54	83.33	2.44	39.47	35.00
Qwen2.5-VL-3B-Tool-free RL	67.54	28.69	54.00	80.65	23.10	87.38	70.83	12.00	38.79	36.67
SpaceTools-3B (Ours)	**79.38**	52.46	**70.00**	**90.32**	**53.07**	94.92	96.00	**34.37**	**43.06**	**50.00**

4.3. Rewards

Reinforcement learning covers spatial reasoning tasks such as multiple choice question answering, 2D bounding box localization, pointing, pose, and grasp estimation. We design normalized, task-specific rewards based on the correctness of the final answer A_{final}. Each reward measures the accuracy or geometric consistency of A_{final} against the ground-truth label or annotation. We additionally experimented with a structural *format score* to encourage output correctness, but found it provided no measurable improvement and excluded it from final training. See details in the Appendix.

- **Multiple choice questions.** The reward is binary: $R_{\text{B}} = 1$ if A_{Final} is correct, else 0.
- **2D bounding boxes.** We compute Mean IoU (MIoU) between predicted and ground-truth boxes: $R_{\text{MIoU}} = \frac{1}{N}\sum_{i=1}^{N} \max_j \text{IoU}(\hat{B}_i, B_j)$, where $\hat{B}_i$ and B_j denote predicted and ground-truth boxes.
- **Pointing.** For single-point spatial prediction, we use the Normalized Negative Distance to Centroid (NNDC): $R_{\text{NNDC}} = \frac{\exp(-5d)-\exp(-5\sqrt{2})}{1-\exp(-5\sqrt{2})}$, where d is the distance to the target-region centroid. To emphasize precision, we clip with the binary accuracy term: $R = \max(R_{\text{NNDC}}, R_{\text{B}})$.
- **Pose estimation.** Predicted and ground-truth poses are converted to eight 2D projected corners. The reward is the IoU between convex hulls of predicted ($\hat{C}$) and ground truth (C) corner sets. $R_{\text{IoU}} = \text{IoU}(\hat{C}, C)$ when both sets are valid ($|\hat{C}| = |C| = 8$), and 0 otherwise.
- **Grasp estimation.** We adopt the Normalized Negative Coordinate Error (NNCE): $R_{\text{NNCE}} = 1 - \frac{1}{\delta_{\max}} \min\left(\delta_{\max}, \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{p}_i - p_i\|_2}{d}\right)$, where $\hat{p}_i$ and p_i are predicted and ground-truth contact points, d is the gripper width, N is the number of reference points, and $\delta_{\max}$ caps extreme errors. This rewards accurate geometric grasp alignment. In this work, $\delta_{\max} = 10$.

5. Experiments

Dataset. During the first phase of DIRL, we generate a teaching SFT dataset composed of 8k high-quality tool-use trajectories: 6k from the universal teacher and 2k from the IRL-trained teacher. The IRL teacher is trained to use a Pointing tool (RoboRefer [83]), a common first step before querying other vision and robotic tools in spatial reasoning. For the universal teacher, we use Claude Sonnet 4.5 [1], integrated with Toolshed, consisting of all tools. Image–question pairs are sampled from RoboSpatial [55], RefSpatial [83], and BOP-ASK [4]. To extend our setup to robot manipulation, we augment the HOPE dataset [63] with grasping and pick-and-place control tasks. Each trajectory in the SFT dataset is a multi-turn dialogue between the user, assistant, and tools. The same spatial reasoning image-question pairs in the teaching SFT dataset are used in the stage-2 IRL.

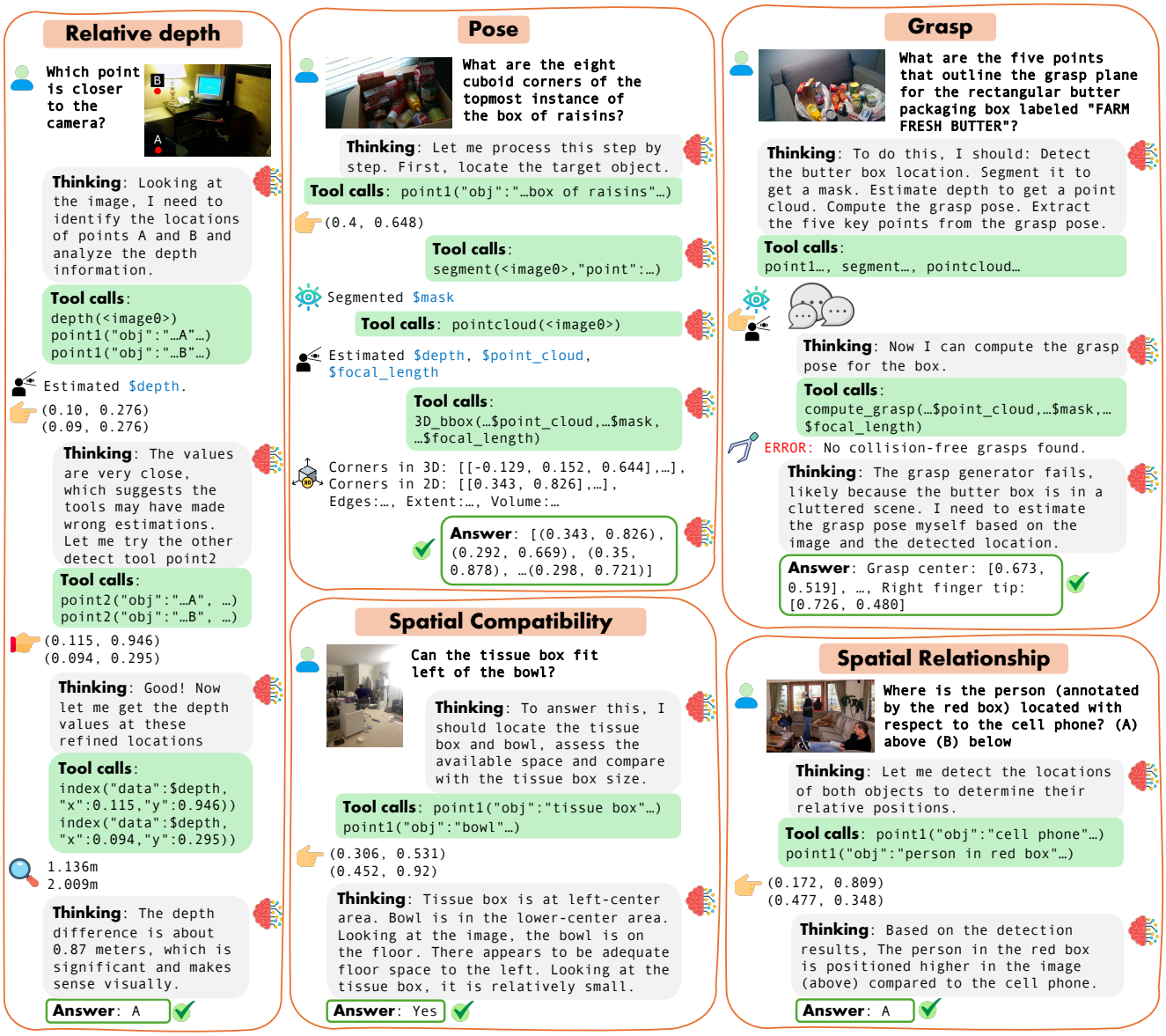


Figure 3. Spatial reasoning examples of SpaceTools. It performs diverse spatial reasoning tasks including relative depth, pose, grasp, spatial compatibility, and spatial relationship by interleaving reasoning (gray) and vision tool calls (green) before producing the final answer. Images are taken from BLINK [16], RoboSpatial-Home [55], and BOP-ASK [4].

Training. We use Qwen2.5-VL-3B-Instruct [3] as our base model. During training, we use a system prompt that defines the available tools, their schemas, and usage formats (detailed prompt is in the Appendix). High-level instructions within the prompt further specify the model's reasoning structure and output format. We use the following perception tools: segmentation [48], pointing [14, 83], depth estimation [6], cuboid fitting, grasp generation [42], cropping, array indexing, perspective projection, and robot tools: image capture, depth capture, grasp, place. The first-stage SFT starts with the base model and uses cross-entropy next-token prediction loss over all assistant turns in a multi-turn dialogue, training the policy π_θ to imitate the teachers' reasoning processes and tool-use behaviors. In the second-stage IRL, the SFT-initialized model interacts with Toolshed to refine its tool-augmented spatial reasoning through interactive feedback from task rewards. For SFT, we use the LLaMA-Factory platform [81], and for RL we integrate Toolshed into the VERL framework [53]. More implementation details are in the Appendix.

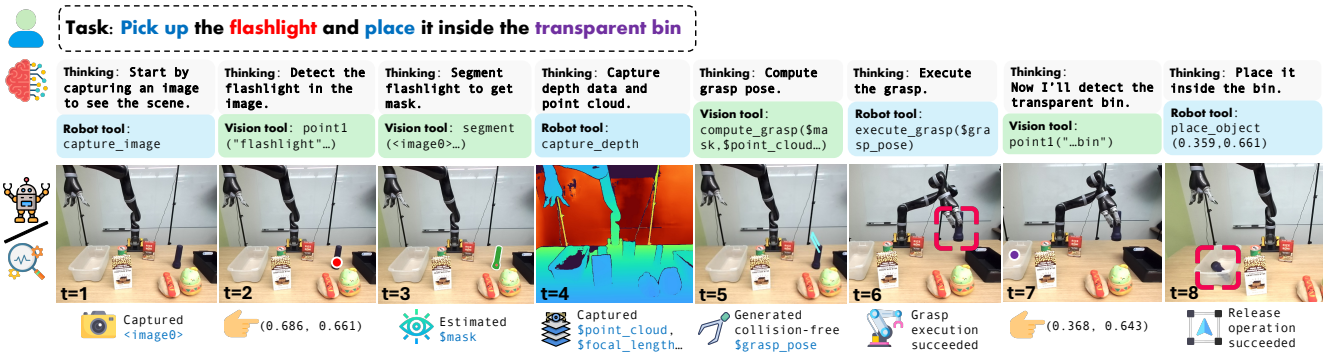


Figure 4. Real-world robot manipulation fully controlled by SpaceTools. The model completes a multi-step task, "picking up the flashlight and placing it in the transparent bin", via alternating reasoning (gray), vision tools (green) for perception, and robot tools (blue) for action.

5.1. Spatial Reasoning Evaluation

Benchmarks and Metrics. We evaluate our model on a suite of spatial reasoning benchmarks, including RoboSpatial-Home [55] (spatial VQA and vacant space pointing), CVBench [60] (2D relations and 3D relative depth), RefSpatial [83] (placement, location, and unseen), BLINK [16] (relative depth), and BOP-ASK [4]. They cover positional relationship understanding, depth estimation, pointing, 3D pose estimation, and robotic grasp prediction. We adopt the following metrics: (1) Answer accuracy for multiple-choice and pointing questions. (2) For object pose estimation, we use the normalized Intersection-over-Union (IoU) in range $[0, 100]$ (%). (3) For grasp estimation, which outputs five 2D coordinates of grasp center, and two finger bases and tips, we use the *Mean Angular Coordinate Error (MACE)* to jointly score grasp location and finger-orientation, defined in the Appendix. We report MACE as a normalized score in range $[0, 100]$ (%), and the *Success Rate (SR)* as the percentage of grasps achieving MACE > 40.

Baselines. We compare our model (SpaceTools) against four categories of baselines. (1) **Proprietary models** include Claude Sonnet 4.5 [1], GPT-4o [45], GPT-5 [46], and Gemini-ER 1.5 [61], which represent state-of-the-art commercial vision-language systems. (2) **General open-source models** include LLaVA-NeXT-8B [28] and Qwen2.5-VL-32B [3], which serve as publicly available multimodal foundations without spatial specialization. (3) **Spatial VLMs** include SpaceLLaVA-13B [10], RoboPoint-13B [74], Molmo-7B [14], RoboBrain2.0-7B [59], and RoboRefer-8B-SFT, which are trained with additional spatial reasoning or robotic data. (4) **Tool-free fine-tuning** contains variants of the same base model (Qwen2.5-VL-3B) trained without tool use, only on the 8k source question and answer samples[1] from DIRL's stage-1: (4a) *Tool-free SFT* is a supervised fine-tuning baseline. (4b) *Tool-free RL* applies reasoning RL à la Deepseek R1 [13] without tool use.

[1] no tool calling

5.2. Spatial Reasoning Results

As shown in Table 2, SpaceTools achieves state-of-the-art results on nearly all benchmarks, surpassing proprietary, open-source, and spatial VLM baselines. SpaceTools outperforms Gemini-ER 1.5 by +7.5% on RoboSpatial, exceeds Claude Sonnet 4.5 by +24.4% on pose estimation, and surpasses GPT-5 by +8.3% on grasp prediction. Moreover, tool-augmented training yields substantially stronger results on spatial reasoning than tool-free fine-tuning of the same base model on the the same 8k VQA pairs regardless of learning technique. SpaceTools-3B achieves higher accuracy on all tasks, notably +12% and +16% on RoboSpatial, than tool-free SFT and RL respectively.

Figure 3 shows qualitative examples. We find that SpaceTools dynamically adapts its reasoning and tool-use strategies to each task. For example, it primarily relies on pointing for tasks such as spatial compatibility and relationship; it invokes depth estimation for relative-depth queries; and it composes multiple tools for more advanced reasoning like pose or grasp prediction. Moreover, SpaceTools has learned corrective behaviors, such as falling back to self-estimation when a tool fails, or switching to alternative pointing tools to refine uncertain detections. Therefore, the model has *learned internal procedures* for tool selection, ordering, and error recovery, rather than relying on hand-crafted pipelines in prior works [25, 34].

5.3. Experiments on Real Robot Manipulation

In order to validate SpaceTools we conduct an experiment where robotics controls are presented as tools, see Figure 4. The robot arm serves as an action tool, complementing vision-based perception tools. By alternating between perception (pointing, segmentation, depth, grasp estimation) and action (capture, grasp) tools, the VLM orchestrates a closed perception–action loop fully guided by language reasoning, in contrast to prior work where robot action is an ex-

Table 3. Real-world robotic manipulation performance of SpaceTools and zero-shot VLM baselines equipped with Toolshed. Values are success rates (%) for *Pick* and *Relation Pick* tasks, partial success rates (%) for *Pick & Place*, and seconds for Time-to-First-Movement (TTFM).

	Real Robot Manipulation Tasks			
Model	**Pick**	**Rel. Pick**	**Pick & Place**	**TTFM**
$\pi_{0.5}$	0 (0/7)	0 (0/6)	0 (0/14)	1s
GPT-5 + Toolshed	71 (5/7)	33 (2/6)	65 (9/14)	36s
Claude Sonnet 4.5 + Toolshed	**86** (6/7)	50 (3/6)	79 (11/14)	30s
Qwen2.5-VL-3B + Toolshed	0 (0/7)	0 (0/6)	0 (0/14)	-
SpaceTools (Ours)	**86** (6/7)	**83** (5/6)	**86** (12/14)	10s

Table 4. Ablation on training recipes. IRL-T denotes the IRL-trained teacher; Univ-T denotes the universal (frontier-model) teacher; S2-IRL denotes the Stage-2 interactive RL phase. Checkmarks indicate which components are included.

Variant	**IRL-T**	**Univ-T**	**S2-IRL**	**RoboSpatial**	**RefSpatial**	**Pose**	**Mean**
with Interactive RL							
SpaceTools (Ours)	✓	✓	✓	70.00	53.07	34.37	52.48
w/o IRL Teacher	×	✓	✓	61.14	29.60	34.29	41.68
w/o Univ. Teacher	✓	×	✓	65.14	54.51	8.92	42.86
w/o Stage 2 IRL	✓	✓	×	67.71	51.98	33.28	50.99
without Interactive RL							
Tool SFT	×	✓	×	59.71	24.91	32.94	39.19
Tool NIRL	×	✓	×	55.14	28.16	30.89	38.06

ternal process to model reasoning [55]. We evaluate SpaceTools, Claude Sonnet 4.5, and GPT-5 in this tool-augmented system as well as comparing with a strong vision-language-action model, $\pi_{0.5}$ [5]. We focus on three type of tasks; pick, relational pick, and pick & place, results from this experiments are presented in Table 3. During the experiments we observed that SpaceTools is better grounded in spatial reasoning as well as being capable of orchestrating multiple tools, whereas other methods, like GPT-5, fail to chain tools coherently, sometimes inventing grasp poses or camera intrinsics instead of reusing computed values. Please consult the Appendix for further details.

5.4. Ablation Study

To analyze the contribution of each component in the DIRL framework, we perform systematic ablations on spatial reasoning benchmarks by removing (1) the *IRL-trained teacher* (IRL-T), (2) the *universal teacher* (Univ-T), and (3) the *Stage 2 IRL* phase (S2-IRL).

In addition, to evaluate the importance of interactive reinforcement learning, we compare DIRL with two classic non-interactive training schemes: (a) *Tool SFT* with the universal teacher, where the model is trained on multi-turn tool-use traces through direct supervision, and (b) *Tool Non-Interactive Reinforcement Learning* (*Tool NIRL*), which follows the conventional tool-learning setup in large language models (LLM) [79]. In Tool NIRL, ground-truth tool call traces are required, and the reward is based on the correctness of tool names, tool arguments, and answers. Detailed configurations are provided in the Appendix.

Table 5. Comparison of proprietary models with and without the Toolshed enhancement across robotic spatial reasoning benchmarks. Values are normalized accuracy (%).

Model	**RoboSpatial**	**BLINK**	**RefSpatial**	**BOP-ASK**	
				Pose	**Grasp (MACE)**
GPT-5	58.39	66.13	23.10	9.03	39.59
+ Toolshed	55.14	90.32 ↑	36.10 ↑	15.00 ↑	41.49 ↑
Claude	57.43	78.23	7.49	1.67	40.12
+ Toolshed	52.86	75.00	27.80 ↑	25.00 ↑	44.19 ↑

Quantitative results are in Table 4, with our main findings summarized: (1) Removing the IRL-trained teacher leads to a sharp performance drop, particularly on tasks requiring fine spatial grounding such as RefSpatial and RoboSpatial. (2) Removing the universal teacher also degrades performance, especially on pose tasks that require multi-tool composition (*e.g.*, segmentation + depth + 3D bbox). (3) Stage 2 IRL provides the final boost of tool-augmented reasoning. Eliminating the Stage 2 IRL phase affects performance across RoboSpatial, RefSpatial, and pose tasks. (4) Both Tool SFT and Tool NIRL baselines underperform DIRL by a large margin (+13.4 and +14.4 mean improvement, respectively). This suggests that interactive RL is key to teaching VLMs consistent reasoning over complex tool sequences.

6. Discussion & Conclusion

Agentic VLMs hold the promise of reasoning through arbitrary external tools. Motivated by this, we examine whether large VLMs can improve their spatial reasoning by leveraging vision tools in a fully zero-shot setting. As shown in Table 5, tool integration yields clear gains on tasks requiring precise spatial grounding or explicit geometric reasoning. For example, GPT-5 with Toolshed improves on RefSpatial (from 23.1 to 36.1) and pose estimation (from 9.0 to 15.0), suggesting that tool feedback mitigates limitations in spatial grounding and 3D understanding. In contrast, high-level tasks such as RoboSpatial and BLINK show mixed trends, as models tend to overuse tools and struggle to correctly interpret nuanced tool outputs. We also find that IRL improves out-of-domain generalization. When a model is trained to use a single powerful tool such as pointing [83], it not only performs better on its in-domain benchmark but also transfers unexpectedly well. For instance, a model trained only on RoboSpatial [55] reaches 72.3% accuracy on that benchmark and still achieves 34.3% on RefSpatial—where other fine-tuning approaches score zero. These results highlight the promise of agentic VLMs and their ability to acquire new skills through tool use.

In conclusion, we introduce DIRL, a new method for

training tool-augmented VLMs through progressive and interactive learning. To support this, we built Toolshed, a system for deploying diverse tools at scale for online interaction during training. Our experiments show that our trained model, SpaceTools, achieves state-of-the-art performance on multiple spatial reasoning benchmarks and exhibits strong out-of-distribution generalization, including the ability to use a robot as a tool. This work demonstrates that VLMs can acquire complex spatial reasoning capabilities through learned tool coordination rather than architectural modification or large-scale data-driven fine-tuning.

7. Acknowledgments

The authors would like to thank Vineet Bha, Alex Zook, Stephen Tyree, and Huijie Zhang for their inputs and comments on this manuscript.

References

[1] Anthropic. Claude sonnet 4.5 system card. Technical report, Anthropic, 2025. Accessed: 2025-11-09. 5, 7

[2] Daichi Azuma, Taiki Miyanishi, Shuhei Kurita, and Motoaki Kawanabe. Scanqa: 3d question answering for spatial scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[3] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. 2, 6, 7

[4] Vineet Bhat, Sungsu Kim, Valts Blukis, Greg Heinrich, Prashanth Krishnamurthy, Ramesh Karri, Stan Birchfield, Farshad Khorrami, and Jonathan Tremblay. Bop-ask: Object-interaction reasoning for vision-language models, 2025. 2, 5, 6, 7

[5] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, pages 17–40. PMLR, 2025. 8

[6] Aleksei Bochkovskii, Amaël Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R. Richter, and Vladlen Koltun. Depth pro: Sharp monocular metric depth in less than a second. In *International Conference on Learning Representations*, 2025. 2, 6

[7] Wenxiao Cai, Iaroslav Ponomarenko, Jianhao Yuan, Xiaoqi Li, Wankou Yang, Hao Dong, and Bo Zhao. Spatialbot: Precise spatial understanding with vision language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9490–9498, 2025. 3

[8] Zhipeng Cai, Ching-Feng Yeh, Hu Xu, Zhuang Liu, Gregory Meyer, Xinjie Lei, Changsheng Zhao, Shang-Wen Li, Vikas Chandra, and Yangyang Shi. Depthlm: Metric depth from vision language models, 2025. 2, 3

[9] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14455–14465, 2024. 3

[10] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465, 2024. 2, 7

[11] Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. Research: Learning to reason with search for llms via reinforcement learning, 2025. 3

[12] An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. Spatialrgpt: Grounded spatial reasoning in vision-language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 2, 3

[13] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. 3, 7, 11

[14] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, Yen-Sung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 3, 6, 7, 2

[15] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. 3

[16] Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models

can see but not perceive. *arXiv preprint arXiv:2404.12390*, 2024. 2, 6, 7

[17] Tanmay Gupta and Aniruddha Kembhavi. Visual Programming: Compositional visual reasoning without training . In *CVPR*, pages 14953–14962, 2023. 3

[18] Yi Han, Cheng Chi, Enshen Zhou, Shanyu Rong, Jingkun An, Pengwei Wang, Zhongyuan Wang, Lu Sheng, and Shanghang Zhang. Tiger: Tool-integrated geometric reasoning in vision-language models for robotics. *arXiv preprint arXiv:2510.07181*, 2025. 2, 3, 4

[19] Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024. 3

[20] Yuheng Ji, Huajie Tan, Jiayu Shi, Xiaoshuai Hao, Yuan Zhang, Hengyuan Zhang, Pengwei Wang, Mengdi Zhao, Yao Mu, Pengju An, et al. Robobrain: A unified brain model for robotic manipulation from abstract to concrete. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1724–1734, 2025. 2

[21] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. 3, 4

[22] Amita Kamath, Jack Hessel, and Kai-Wei Chang. What's "up" with vision-language models? investigating their struggle with spatial reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. 2

[23] Dongyoung Kim, Huiwon Jang, Sumin Park, Jaehyung Kim, Younggyo Seo, and Jinwoo Shin. Robot-r1: Reinforcement learning for enhanced embodied reasoning in robotics. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 3

[24] 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. 2

[25] Phillip Y Lee, Jihyeon Je, Chanho Park, Mikaela Angelina Uy, Leonidas Guibas, and Minhyuk Sung. Perspective-aware reasoning in vision-language models via mental imagery simulation. *arXiv preprint arXiv:2504.17207*, 2025. 2, 3, 7

[26] Chengzu Li, Wenshan Wu, Huanyu Zhang, Yan Xia, Shaoguang Mao, Li Dong, Ivan Vulić, and Furu Wei. Imagine while reasoning in space: Multimodal visualization-of-thought, 2025. 2

[27] Ji Lin, Hongxu Yin, Wei Ping, Yao Lu, Pavlo Molchanov, Andrew Tao, Huizi Mao, Jan Kautz, Mohammad Shoeybi, and Song Han. Vila: On pre-training for visual language models. *arXiv preprint arXiv:2312.07533*, 2023. 2

[28] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26296–26306, 2024. 2, 7

[29] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge. Online blog / model documentation, 2024. Improved Vision-Language Model over LLaVA. 2

[30] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*, 2024. 3

[31] Yuecheng Liu, Dafeng Chi, Shiguang Wu, Zhanguang Zhang, Yaochen Hu, Lingfeng Zhang, Yingxue Zhang, Shuang Wu, Tongtong Cao, Guowei Huang, et al. Spatialcot: Advancing spatial reasoning through coordinate alignment and chain-of-thought for embodied task planning. *arXiv preprint arXiv:2501.10074*, 2025. 3

[32] Yang Liu, Ming Ma, Xiaomin Yu, Pengxiang Ding, Han Zhao, Mingyang Sun, Siteng Huang, and Donglin Wang. Ssr: Enhancing depth perception in vision-language models via rationale-guided spatial reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 2

[33] Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Visual-rft: Visual reinforcement fine-tuning. *arXiv preprint arXiv:2503.01785*, 2025. 3

[34] Chenyang Ma, Kai Lu, Ta-Ying Cheng, Niki Trigoni, and Andrew Markham. Spatialpin: Enhancing spatial reasoning capabilities of vision-language models through prompting and interacting 3d priors. *arXiv preprint arXiv:2403.13438*, 2024. 7

[35] Chenyang Ma, Kai Lu, Ta-Ying Cheng, Niki Trigoni, and Andrew Markham. Spatialpin: Enhancing spatial reasoning capabilities of vision-language models through prompting and interacting 3d priors. *Advances in neural information processing systems*, 37:68803–68832, 2024. 2, 3

[36] Wufei Ma, Yu-Cheng Chou, Qihao Liu, Xingrui Wang, Celso de Melo, Jianwen Xie, and Alan Yuille. Spatialreasoner: Towards explicit and generalizable 3d spatial reasoning. *arXiv preprint arXiv:2504.20024*, 2025. 3

[37] Wufei Ma, Yu-Cheng Chou, Qihao Liu, Xingrui Wang, Celso de Melo, Jianwen Xie, and Alan Yuille. Spatialreasoner: Towards explicit and generalizable 3d spatial reasoning. *arXiv preprint arXiv:2504.20024*, 2025. 2

[38] Wufei Ma, Luoxin Ye, Celso M de Melo, Jieneng Chen, and Alan Yuille. Spatialllm: A compound 3d-informed design towards spatially-intelligent large multimodal models. *arXiv preprint arXiv:2505.00788*, 2025. 3

[39] Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks, 2024. 3

[40] Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, Karmesh Yadav, Qiyang Li, Ben Newman, Mohit Sharma, Vincent Berges, Shiqi Zhang, Pulkit Agrawal, Yonatan Bisk,

Dhruv Batra, Mrinal Kalakrishnan, Franziska Meier, Chris Paxton, Sasha Sax, and Aravind Rajeswaran. Openeqa: Embodied question answering in the era of foundation models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[41] Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic ai for spatial reasoning with a dynamic api. *arXiv preprint arXiv:2502.06787*, 2025. 3

[42] Adithyavairavan Murali, Balakumar Sundaralingam, Yu-Wei Chao, Wentao Yuan, Jun Yamada, Mark Carlson, Fabio Ramos, Stan Birchfield, Dieter Fox, and Clemens Eppner. Graspgen: A diffusion-based framework for 6-dof grasping with on-generator training, 2025. 2, 6

[43] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022. 3

[44] NVIDIA. Cosmos-reason1: From physical common sense to embodied reasoning, 2025. 3

[45] OpenAI. Gpt-4 technical report, 2024. 7

[46] OpenAI. Gpt-5 system card. Technical report, OpenAI, 2025. Accessed: 2025-11-09. 2, 7

[47] Santhosh Kumar Ramakrishnan, Erik Wijmans, Philipp Kraehenbuehl, and Vladlen Koltun. Does spatial cognition emerge in frontier models? In *International Conference on Learning Representations*, 2025. 2

[48] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 2, 6

[49] Arijit Ray, Jiafei Duan, Ellis L Brown II, Reuben Tan, Dina Bashkirova, Rose Hendrix, Kiana Ehsani, Aniruddha Kembhavi, Bryan A. Plummer, Ranjay Krishna, Kuo-Hao Zeng, and Kate Saenko. SAT: Dynamic spatial aptitude training for multimodal language models. In *Second Conference on Language Modeling*, 2025. 2, 3

[50] Gabriel Sarch, Snigdha Saha, Naitik Khandelwal, Ayush Jain, Michael J. Tarr, Aviral Kumar, and Katerina Fragkiadaki. Vigorl: Visually grounded reinforcement learning. *arXiv preprint arXiv:2505.23678*, 2025. 2, 3, 4

[51] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. 3, 4, 8

[52] Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, Ruochen Xu, and Tiancheng Zhao. Vlm-r1: A stable and generalizable r1-style large vision-language model, 2025. 3

[53] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024. 6

[54] Fatemeh Shiri, Xiao-Yu Guo, Mona Golestan Far, Xin Yu, Reza Haf, and Yuan-Fang Li. An empirical analysis on spatial reasoning capabilities of large multimodal models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21440–21455, Miami, Florida, USA, 2024. Association for Computational Linguistics. 2

[55] Chan Hee Song, Valts Blukis, Jonathan Tremblay, Stephen Tyree, Yu Su, and Stan Birchfield. RoboSpatial: Teaching spatial understanding to 2D and 3D vision-language models for robotics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. Oral Presentation. 2, 3, 5, 6, 7, 8

[56] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023. 10

[57] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898, 2023. 3

[58] Emilia Szymanska, Mihai Dusmanu, Jan-Willem Buurlage, Mahdi Rad, and Marc Pollefeys. Space3D-Bench: Spatial 3D Question Answering Benchmark. In *European Conference on Computer Vision (ECCV) Workshops*, 2024. 2

[59] BAAI RoboBrain Team. Robobrain 2.0 technical report, 2025. 7

[60] CVBench Team. Cvbench: A benchmark for cross-video multimodal reasoning, 2025. 7

[61] Gemini Robotics Team. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer, 2025. 7

[62] Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025. 3

[63] Stephen Tyree, Jonathan Tremblay, Thang To, Jia Cheng, Terry Mosier, Jeffrey Smith, and Stan Birchfield. 6-dof pose estimation of household objects for robotic manipulation: An accessible dataset and benchmark. In *International Conference on Intelligent Robots and Systems (IROS)*, 2022. 5

[64] Peiyao Wang and Haibin Ling. Svqa-r1: Reinforcing spatial reasoning in mllms via view-consistent reward optimization. *arXiv preprint arXiv:2506.01371*, 2025. 3

[65] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org, 2024. 3

[66] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2022. Curran Associates Inc. 11

[67] Junfei Wu, Jian Guan, Kaituo Feng, Qiang Liu, Shu Wu, Liang Wang, Wei Wu, and Tieniu Tan. Reinforcing spatial reasoning in vision-language models with interwoven thinking and visual drawing, 2025. 3

[68] Mingyuan Wu, Jingcheng Yang, Jize Jiang, Meitang Li, Kaizhuo Yan, Hanchao Yu, Minjia Zhang, Chengxiang Zhai, and Klara Nahrstedt. Vtool-r1: Vlms learn to think with images via reinforcement learning on multimodal tool use, 2025. 3

[69] Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. Mind's eye of llms: visualization-of-thought elicits spatial reasoning in large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc. 2

[70] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023. 3

[71] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *Forty-second International Conference on Machine Learning*, 2025. 2, 3

[72] Yi Yang, Xiaoxuan He, Hongkun Pan, Xiyan Jiang, Yan Deng, Xingtao Yang, Haoyu Lu, Dacheng Yin, Fengyun Rao, Minfeng Zhu, Bo Zhang, and Wei Chen. R1-onevision: Advancing generalized multimodal reasoning through cross-modal formalization. *arXiv preprint arXiv:2503.10615*, 2025. 3

[73] Shaofeng Yin, Ting Lei, and Yang Liu. Toolvqa: A dataset for multi-step reasoning vqa with external tools. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4424–4433, 2025. 3

[74] Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. In *Conference on Robot Learning (CoRL)*, 2024. Also available as arXiv preprint arXiv:2406.10721. 7

[75] Yufei Zhan, Yousong Zhu, Shurong Zheng, Hongyin Zhao, Fan Yang, Ming Tang, and Jinqiao Wang. Vision-r1: Evolving human-free alignment in large vision-language models via vision-guided reinforcement learning, 2025. 3

[76] Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024. 3

[77] Jiahui Zhang, Yurui Chen, Yanpeng Zhou, Yueming Xu, Ze Huang, Jilin Mei, Junhui Chen, Yujie Yuan, Xinyue Cai, Guowei Huang, Xingyue Quan, Hang Xu, and Li Zhang. From flatland to space: Teaching vision-language models to perceive and reason in 3d. *arXiv preprint arXiv:2503.22976*, 2025. 2, 3

[78] Jingyi Zhang, Jiaxing Huang, Huanjin Yao, Shunyu Liu, Xikun Zhang, Shijian Lu, and Dacheng Tao. R1-vl: Learning to reason with multimodal large language models via step-wise group relative policy optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 3

[79] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning, 2025. 3, 8, 10

[80] Xintong Zhang, Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaowen Zhang, Yang Liu, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Chain-of-focus: Adaptive visual search and zooming for multimodal reasoning via rl. *ArXiv*, abs/2505.15436, 2025. 3

[81] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 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)*, pages 400–410, Bangkok, Thailand, 2024. Association for Computational Linguistics. 6

[82] Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deepeyes: Incentivizing "thinking with images" via reinforcement learning, 2025. 3

[83] Enshen Zhou, Jingkun An, Cheng Chi, Yi Han, Shanyu Rong, Chi Zhang, Pengwei Wang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, et al. Roborefer: Towards spatial referring with reasoning in vision-language models for robotics. *arXiv preprint arXiv:2506.04308*, 2025. 2, 3, 5, 6, 7, 8

SpaceTools: Tool-Augmented Spatial Reasoning via Double Interactive RL

Supplementary Material

We provide additional details and extended results in the supplementary materials,

- Appendix A: Limitations and future directions.
- Appendix B: Further details on the Toolshed system.
- Appendix C: Expanded method descriptions.
- Appendix D: Additional implementation details.
- Appendix E: Extended experimental results.

A. Limitations and Future Directions

Our work shows that tool-augmented spatial reasoning, enabled through DIRL and the Toolshed infrastructure, provides an effective and scalable foundation for training VLMs with strong spatial reasoning, robust tool coordination, and broad generalization across diverse tasks and embodiments. At the same time, this framework opens several promising directions that fall beyond our present scope but merit deeper exploration. We discuss these limitations and future opportunities below.

Application scope. A natural next step is to broaden the range of tasks and environments in which tool-augmented spatial reasoning is applied. Our current scope focuses on short- or medium-horizon tasks, such as spatial question answering or grasp-and-place manipulations. Extending to more complex, longer-horizon, or multi-stage tasks may further reveal the potential of tool-augmented reasoning, allowing the model to concentrate on reasoning and decision-making rather than learning numerous precise perceptual subtasks. Moreover, integrating richer environments, including large-scale robotic simulation, interactive game environments, or physics-rich virtual worlds, could support more diverse experiences and ultimately more general embodied spatial intelligence.

Methodology. From a methodological perspective, several directions could strengthen the flexibility and robustness of tool-augmented spatial reasoning. Although Toolshed supports image-level tool outputs, this work primarily explores tools that return structured text or variables (*e.g.*, point cloud). Extending the model to reason over visual outputs from tools may unlock more expressive or fine-grained reasoning behaviors. Another important direction is systematically improving how VLMs perceive, verify, and recover from tool errors or inaccuracies. Moreover, under a modular perspective, future work could investigate enhancing particular system capabilities by upgrading a single tool without modifying other components, while ensuring that overall tool coordination remains robust. Finally,

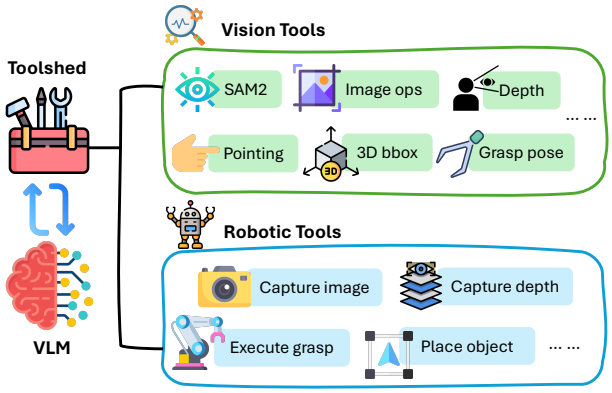


Figure 5. The **Toolshed** infrastructure linking a VLM with modular vision and robotic tools under a unified toolbox for perception and control.

while our RL exploration focuses on prompt design, loss functions, and task-specific reward formulations, alternative RL approaches, such as stepwise reward formulations, may improve learning effectiveness in large multi-tool action spaces.

Infrastructure. On the system side, Toolshed provides a scalable backbone for interactive tool use and learning, but there remains room to further improve its efficiency and resource utilization. Serving many heterogeneous tools in parallel can introduce latency and memory bottlenecks, particularly for high-resolution vision tools or robot-in-the-loop executions. In this work, we mitigate the latter by using mock robot tools during training. Future advances in scheduling, caching, batching, and asynchronous execution could potentially enhance performance and even support real robot execution effectively during interactive learning. Additionally, developing lighter-weight tools, model-side approximators, or memory-optimized deployment strategies may reduce overhead and enable larger-scale training or more complex task environments. These improvements would allow the framework to scale more gracefully as tool diversity and task complexity increase.

B. Details of Toolshed

B.1. System Design

This work aims to enable learning and inference with multiple interactive vision tools for spatial reasoning. Effective tool-augmented spatial reasoning requires multi-turn, state-dependent communication between the VLM and its vision

tools. However, many essential tools such as object detectors, depth estimators, and 3D reconstruction modules, are computationally heavy and often dominate both inference and training time in VLM–RL pipelines. Moreover, modern training relies on batched generation, where multiple conversations are executed in parallel. In naïve implementations, a single blocking tool call can stall the entire batch, effectively reducing tool interaction to a serial process. This makes it crucial to keep tools continuously available, preloaded on device, and capable of serving multiple concurrent conversations. To address these challenges, we introduce **Toolshed** (visualized in Figure 5), a distributed toolkit that enables scalable, asynchronous, and parallel vision tool interaction.

- **Decoupled execution.** Tool invocations run independently from the policy's main inference loop, avoiding blocking calls that would otherwise stall unrelated computations.
- **Asynchronous Processing.** Multiple parallel instances can serve the same tool, each receiving inputs and producing outputs asynchronously, enabling high throughput even under large-scale rollouts.
- **Resources isolation.** Tool instances are assigned isolated resources based on the computational profile of each tool.
- **Environment isolation.** Each tool type is hosted in a dedicated python environment, solving the dependency compatibility issue that comes with hosting multiple computer vision tools in a single system.
- **Elastic scaling.** The system design supports automatic spawning of additional tool workers in response to bursts of tool usage, allowing throughput to remain stable even for large batch rollouts. (This capability is part of the infrastructure design but was not enabled in our training experiments.)
- **Multimodal data passing.** Seamless exchange of text, images, and structured variables (*e.g.*, 3D point clouds) is supported between the VLM and tools, even when they run on different devices or GPU nodes. This enables tool workflows that require different types of inputs and outputs.

In practice, Toolshed is implemented on top of the Ray[2] distributed execution framework, which provides lightweight task scheduling, actor management, and high-throughput message passing. For interactive reinforcement learning, Toolshed integrates seamlessly with VERL[3]: VERL's asynchronous multi-turn rollouts align naturally with Toolshed's asynchronous tool actors, enabling us to parallelize expensive perception, generation, and simulation steps without slowing down rollouts. This results in significantly higher steps-per-second compared to monolithic or synchronous training setups. For inference, Toolshed can be attached both to our trained model and to proprietary models (e.g., GPT-5, Claude) via simple API calls, enhancing their robotic spatial reasoning capabilities.

[2] ray.io/

[3] github.com/volcengine/verl

B.2. Provided Tools

Vision tools. We provide the following vision tools. `image_ops` offers basic image manipulations such as point- and mask-based indexing. `sam2` performs instance segmentation from one or more clicks, powered by Segment Anything 2 [48]. `point1` and `point2` are two object-pointing detectors backed by RoboRefer [83] and Molmo [14], respectively. `depth_estimator` predicts monocular depth and reconstructs 3D point clouds using DepthPro [6]. `compute_bbox` estimates 3D bounding boxes and object poses from reconstructed geometry, while `compute_grasp` predicts collision-free grasp poses for robotic manipulation. Finally, `code_executor` allows the VLM to execute small Python snippets for orchestrating multi-tool workflows, returning results with captured `stdout`/`stderr` and optionally caching intermediate outputs for reuse.

Robotic tools. We integrate a set of robotic tools that enable embodied perception and manipulation. `capture_image` captures RGB observations from the robot's onboard camera and stores them for subsequent visual processing. `capture_depth` acquire depth information from the scene, returning a depth map with focal length and a full 3D point cloud reconstruction. `execute_grasp` executes a grasp given a 4×4 transformation matrix representing the target grasp pose and reports execution success and timing feedback. `place_object` places an object at a specified 2D image coordinate, confirming successful placement in the returned text message. Apart from the above real robotic tools that control real-world robot arms, we also provide a set of `mock_robot` tools without relying on real robots for the ease of data generation and training. Together, these tools provide a physical interface that allows the VLM to perceive, reason, and act within the real world, enabling unified spatial reasoning and robotic control.

B.3. Example Process of Launching Tool IRL

We show a condensed workflow of launching tools and interactive reinforcement learning via a single bash script in Listing 1.

Listing 1. Example workflow for launching Toolshed and running VERL GRPO training with tool calling enabled.

```
# ------------------------------------
# Graceful cleanup (kills the Toolshed actor
    process so detached actors vanish)
# ------------------------------------
TOOLSHED_PID=""
```

```
cleanup() {
  echo "Cleaning up"
  if [[ -n "$TOOLSHED_PID" ]]; then
    echo "Killing (PID=$TOOLSHED_PID)"
    kill $TOOLSHED_PID 2>/dev/null
    wait $TOOLSHED_PID 2>/dev/null
    echo "Stopped."
  fi
  exit 0
}
trap cleanup SIGINT SIGTERM EXIT

# 1. Launch Toolshed with GPU-backed vision
    tools
python - <<'PY'
import ray
from toolshed import start_toolkit

ray.init(address='auto')

tool_configs = {
   "point1": {"num_actors": 2, "resources": {"
       num_gpus": 0.5}},
   "point2": {"num_actors": 2, "resources": {"
       num_gpus": 0.5}},
   "sam2": {"num_actors": 4, "resources": {"
       num_gpus": 0.2}},
   "depth_estimator": {"num_actors": 4, "
       resources": {"num_gpus": 0.2}}
   ...
}

pg = ray.util.placement_group([{"CPU": 8, "GPU
    ": 8}], strategy="STRICT_PACK")
ray.get(pg.ready())
router = start_toolkit(tool_configs, dashboard
    =True, placement_group=pg)
print("Started:", list(tool_configs))
PY

# 2. Generate a YAML tool config and make sure
     tools are ready
python generate_toolshed_config.py --output
    toolshed_config.yaml

# 3. Launch VERL GRPO training with Toolshed
    Integration
python -m verl.trainer.main_ppo \
   actor_rollout_ref.model.path=Qwen/Qwen2.5-
       VL-3B-Instruct \
   ...
   actor_rollout_ref.rollout.multi_turn.
       tool_config_path=toolshed_config.yaml

echo "Finished."
```

B.4. Example Toolshed Integration with Proprietary Models

Figure 6 illustrates our integration of Toolshed with a proprietary model (Claude Sonnet 4.5), enabling seamless interactive communication among the user, external tools, and the VLM.

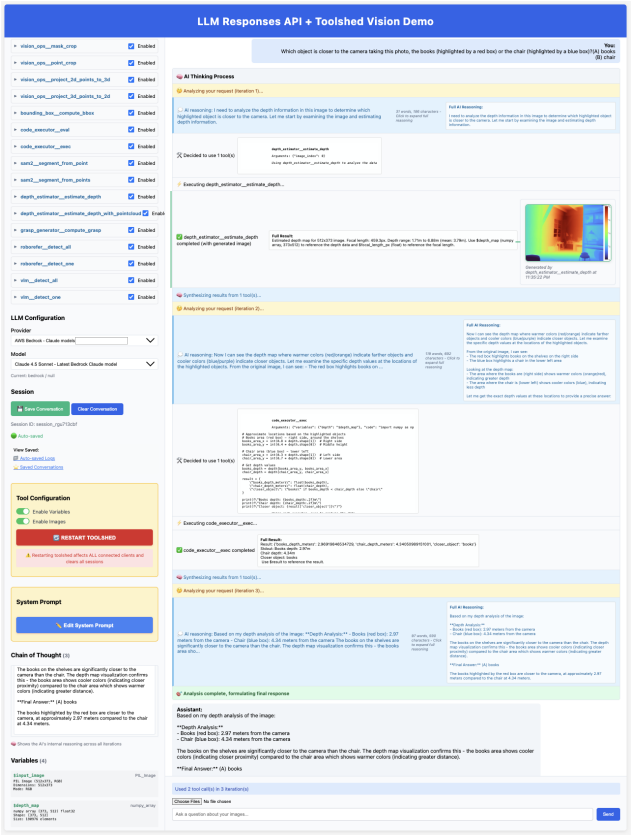


Figure 6. Interactive web demo illustrating Claude's tool-augmented reasoning when integrated with Toolshed.

B.5. Detailed Tool APIs

Complete API of the computer vision and robotic tools supported by Toolshed include:

`image_ops.point_crop(data, x, y)`

Purpose: Get the data value in the numpy ndarray 'data' at the given coordinate.

- **Inputs**:
  - `data`: Numpy ndarray of shape (H, W) or (H, W, C), or PIL Image
  - `x`: Normalized x-coordinate, float in $[0, 1]$
  - `y`: Normalized y-coordinate, float in $[0, 1]$
- **Outputs**:
  - **Raw value**: `data` value indexed at the input coordinate.
  - **Text**: Reports the information about the pixel value at the given coordinates.

`image_ops.point_crop(image, points)`

Crop image to minimally encompass all given points.

- **Inputs**:
  - `image`: PIL.Image
  - `points`: list of (x, y) normalized floats in $[0, 1]$
- **Outputs**:
  - **Raw value**: `PIL.Image` cropped to the bounding box of the points
  - **Text**: Reports the crop box, size, and number of points
  - **Image**: The cropped region only (no overlay)
  - **Variables**: `cropped_image` (`PIL.Image`)

`image_ops.mask_crop(image, mask)`

Crop to the tight bounding box of a boolean mask; outside-mask pixels are set to white.

- **Inputs**:
  - `image`: PIL.Image
  - `mask`: boolean `numpy.ndarray` of shape $H \times W$ matching the image
- **Outputs**:
  - **Raw value**: `PIL.Image` of the masked region on white background, cropped to mask bounds
  - **Text**: Reports crop box, size, and mask coverage percentage
  - **Image**: Masked crop on white background (no overlay)
  - **Variables**: `masked_crop` (`PIL.Image`)

`sam2.segment_from_point(image, x, y)`

Segment the object at a single pixel coordinate.

- **Inputs**:
  - `image`: PIL.Image
  - `x`: Normalized x-coordinate, float in $[0, 1]$
  - `y`: Normalized y-coordinate, float in $[0, 1]$
- **Outputs**:
  - **Raw value**: `dict` with `mask` (boolean $H \times W$ `numpy.ndarray`) and `iou_score` (float)
  - **Text**: Reports the click location and IoU score
  - **Image**: Original image with semi-transparent green mask, a white mask outline, and a red circular point marker (white outline)
  - **Variables**: `segmentation_mask` (boolean $H \times W$ `numpy.ndarray`)

`sam2.segment_from_points(image, points)`

Segment an object using multiple foreground points.

- **Inputs**:
  - `image`: PIL.Image
  - `points`: list of (x, y) normalized floats in $[0, 1]$
- **Outputs**:
  - **Raw value**: `dict` with `mask` (boolean $H \times W$ `numpy.ndarray`) and `iou_scores` (1-D `numpy.ndarray`)
  - **Text**: Reports the number of points and the best IoU score
  - **Image**: Original image with semi-transparent green mask, a white mask outline, and red circular markers at all provided points.
  - **Variables**: `segmentation_mask` (boolean $H \times W$ `numpy.ndarray`)

`point1.detect_one(image, obj_name)`

Identify one instance of the named object by pointing to them with Roborefer.

- **Inputs**:
  - `image`: PIL.Image
  - `obj_name`: string
- **Outputs**:
  - **Raw value**: String serialization of a normalized point coordinate $(x, y) \in [0, 1]^2$
  - **Text**: Reports the object name, count, and the normalized point
  - **Image**: Original image with red circular point markers at detected locations (white outlines)
  - **Variables**: *<obj_name>*`_detection` ((x, y) floats in $[0, 1]$; spaces in *obj_name* replaced with underscores)

`point1.detect_all(image, obj_name)`

Identify instances of the named object by pointing to them with Roborefer.

- **Inputs**:
  - `image`: PIL.Image
  - `obj_name`: string
- **Outputs**:
  - **Raw value**: String serialization of a list of normalized point coordinates $(x, y) \in [0, 1]^2$
  - **Text**: Reports the object name, count, and the list of normalized points
  - **Image**: Original image with red circular point markers at detected locations (white outlines)

- **Variables**: `<obj_name>_detections` (list of (x, y) floats in $[0, 1]$; spaces in *obj_name* replaced with underscores)

`point2.detect_one(image, obj_name)`
Identify one instance of the named object by pointing to them with Molmo.

- **Inputs**:
  - `image`: PIL.Image
  - `obj_name`: string
- **Outputs**:
  - **Raw value**: String serialization of a normalized point coordinate $(x, y) \in [0, 1]^2$
  - **Text**: Reports the object name, count, and the normalized point
  - **Image**: Original image with red circular point markers at detected locations (white outlines)
  - **Variables**: `<obj_name>_detection` ((x, y) floats in $[0, 1]$; spaces in *obj_name* replaced with underscores)

`point2.detect_all(image, obj_name)`
Identify instances of the named object by pointing to them with Molmo.

- **Inputs**:
  - `image`: PIL.Image
  - `obj_name`: string
- **Outputs**:
  - **Raw value**: String serialization of a list of normalized point coordinates $(x, y) \in [0, 1]^2$
  - **Text**: Reports the object name, count, and the list of normalized points
  - **Image**: Original image with red circular point markers at detected locations (white outlines)
  - **Variables**: `<obj_name>_detections` (list of (x, y) floats in $[0, 1]$; spaces in *obj_name* replaced with underscores)

`depth_estimator.estimate_depth(image)`
Monocular depth estimation with DepthPro.

- **Inputs**:
  - `image`: PIL.Image
- **Outputs**:
  - **Raw value**: `dict` with `depth_map` ($H \times W$ float array, meters), `focal_length_px` (float), `width` (int), `height` (int)
  - **Text**: Reports image size, focal length, and depth range statistics
  - **Image**: Colorized depth map with a vertical scale bar on the right labeled "Depth (m)"
  - **Variables**: `depth_map` ($H \times W$ float array), `focal_length_px` (float)

`depth_estimator.estimate_depth_with_pointcloud(image)`
Monocular depth estimation and 3D point cloud generation with DepthPro.

- **Inputs**:
  - `image`: PIL.Image
- **Outputs**:
  - **Raw value**: `dict` with `depth_map` ($H \times W$ float array, meters), `point_cloud` ($N \times 3$ float array of 3D points in camera coordinates), `focal_length_px` (float), `width` (int), `height` (int)
  - **Text**: Reports image size, focal length, and depth range statistics
  - **Image**: Colorized depth map with a vertical scale bar on the right labeled "Depth (m)"
  - **Variables**: `depth_map` ($H \times W$ float array), `point_cloud` ($N \times 3$ float array of 3D points in camera coordinates), `focal_length_px` (float)

`grasp_generator.compute_grasp(point_cloud, mask, image, focal_length_px)`
Generate a single grasp pose for a masked subset of a point cloud with GraspGen.

- **Inputs**:
  - `point_cloud`: $N \times 3$ numpy float array
  - `mask`: Boolean segmentation mask
  - `image`: PIL.Image
  - `focal_length_px`: float
- **Outputs**:
  - **Raw value**: Collision-free grasp pose, and collision-free confidence
  - **Text**: Reports the collision-free grasp confidence, the total number of generated grasps and the percentage of collision-free grasps, and the projected 2D gripper points of the best grasp pose in normalized coordinates
  - **Image**: Original image overlayed with projected X-(red), Y-(green), Z-(blue) gripper axes
  - **Variables**: `grasp_pose` (4×4 ndarray, OpenCV camera frame)

`3d_bbox.compute_bbox(point_cloud, mask, focal_length_px)`
Compute an oriented bounding box for a masked subset of a point cloud.

- **Inputs**:
  - `point_cloud`: $N \times 3$ numpy float array
  - `mask`: Boolean segmentation mask
  - `focal_length_px`: float
- **Outputs**:
  - **Raw value**: Box corners in 3D, box corners in 2D, edges, and extent

  - **Text**: Summary containing number of input points, the point coordinates in 3d and 2d, mask shape, box extents, and edges
  - **Image**: No image output
  - **Variables**: `obb_corners_3d` (8×3 list of lists, meters in opencv camera frame), `obb_corners_2d` (8×2 list of lists, normalized image coordinates), `extent` (3-element ndarray, extent of the bounding box in meters), `edges` (list of pairs of integers, edges of the bounding box defined by the indices of the corners)

`code_executor.exec(code)`

Execute a multi-line Python block (imports limited to `math` and `numpy`).

- **Inputs**:
  - `code`: string
- **Outputs**:
  - **Raw value**: tuple $(result, stdout, stderr)$
  - **Text**: Summarizes the result, captured stdout, and stderr; notes a stored variable if applicable
  - **Image**: No image output
  - **Variables**: `last_exec_result` (present iff a non-`None` result and variables are enabled)

`code_executor.eval(expression)`

Evaluate a single Python expression.

- **Inputs**:
  - `expression`: string
- **Outputs**:
  - **Raw value**: tuple $(result, stdout, stderr)$
  - **Text**: Summarizes the result, captured stdout, and stderr; notes a stored variable if applicable
  - **Image**: No image output
  - **Variables**: `last_eval_result` (present iff a non-`None` result and variables are enabled)

`mock_robot.capture_image(mock_data)`

Return the mock image from the dataset without a real robot.

- **Inputs**:
  - `mock_data`: `dict` with `mock_image`
- **Outputs**:
  - **Raw value**: Image from mock camera
  - **Text**: Image dimensions and capture status
  - **Image**: Captured image from mock camera
  - **Variables**: `captured_image` (PIL Image)

`mock_robot.get_depth(mock_data)`

Return the mock depth from the dataset without a real robot.

- **Inputs**:
  - `mock_data`: `dict` with `mock_image` (PIL Image), `mock_depth_map` (numpy array), `mock_focal_length_px` (float), `image_width` (int), `image_height` (int)
- **Outputs**:
  - **Raw value**: `image` (PIL Image), `depth_map` (numpy array), `focal_length_px` (float), `width` (int), `height` (int)
  - **Text**: Summary of depth data including image dimensions, focal length, and depth statistics
  - **Image**: A colorized depth map visualization where closer objects appear cooler (blue/purple) and distant objects appear warmer (red/yellow)
  - **Variables**: `depth_map` (2D numpy array of depth values in meters), `foca_length_px` (float, estimated focal length in pixels)

`mock_robot.get_depth_with_pointcloud(mock_data)`

Return the mock depth and and point cloud generation from the dataset without a real robot.

- **Inputs**:
  - `mock_data`: `dict` with `mock_image` (PIL Image), `mock_depth_map` (numpy array), `mock_point_cloud` (numpy array), `mock_focal_length_px` (float), `image_width` (int), `image_height` (int).
- **Outputs**:
  - **Raw value**: `image` (PIL Image), `depth_map` (numpy array), `mock_point_cloud` (numpy array), `focal_length_px` (float), `width` (int), `height` (int).
  - **Text**: Summary of depth data and and point cloud generation including image dimensions, focal length, depth statistics, and point cloud size
  - **Image**: A colorized depth map visualization where closer objects appear cooler (blue/purple) and distant objects appear warmer (red/yellow)
  - **Variables**: `depth_map` (2D numpy array of depth values in meters), `point_cloud` ($N \times 3$ array of 3D points), `foca_length_px` (float, estimated focal length in pixels)

`mock_robot.execute_grasp(grasp_pose)`

Simulate executing a grasp (always succeeds).

- **Inputs**:
  - `grasp_pose`: 4×4 transformation matrix representing the grasp pose in the robot's camera frame, OpenCV convention
- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float)
  - **Text**: Confirmation that grasp was successful

- **Image**: No image output
- **Variables**: No variable output

`mock_robot.place_object_at_2d_location(point_2d)`

Simulate placing object at 2D location (always succeeds).

- **Inputs**:
  - `point_2d`: 2D normalized image coordinate where the object should be placed
- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float)
  - **Text**: Confirmation that placement was successful
  - **Image**: No image output
  - **Variables**: No variable output

`mock_robot.place_object_at_3d_location(point_3d)`

Simulate placing object at 3D location (always succeeds).

- **Inputs**:
  - `point_3d`: 3D point in the robot's camera frame (list or numpy array) where the object should be placed
- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float)
  - **Text**: Confirmation that placement was successful
  - **Image**: No image output
  - **Variables**: No variable output

`robot.capture_image()`

Return the mock image from the dataset without a real robot.

- **Inputs**: No input required
- **Outputs**:
  - **Raw value**: `image_shape` (numpy array or list), `image` (PIL Image)
  - **Text**: Image dimensions and capture status
  - **Image**: Captured image from robot camera
  - **Variables**: `captured_image` (PIL Image)

`robot.get_depth()`

Retrieve depth map from the robot's depth sensor.

- **Inputs**: No input required
- **Outputs**:
  - **Raw value**: `depth_map` (numpy array), `depth_map_visualization` (PIL Image), `focal_length_px` (float), `width` (int), `height` (int)
  - **Text**: Summary of depth data including image dimensions, focal length, and depth statistics
  - **Image**: A colorized depth map visualization where closer objects appear cooler (blue/purple) and distant objects appear warmer (red/yellow)
  - **Variables**: `depth_map` (2D numpy array of depth values in meters), `foca_length_px` (float, estimated focal length in pixels)

`robot.get_depth_with_pointcloud()`

Retrieve depth map from robot's depth sensor and generate 3D point cloud.

- **Inputs**: No input required
- **Outputs**:
  - **Raw value**: `image` (PIL Image), `depth_map` (numpy array), `point_cloud` (numpy array), `focal_length_px` (float), `width` (int), `height` (int).
  - **Text**: Summary of depth data and and point cloud generation including image dimensions, focal length, depth statistics, and point cloud size
  - **Image**: A colorized depth map visualization where closer objects appear cooler (blue/purple) and distant objects appear warmer (red/yellow)
  - **Variables**: `depth_map` (2D numpy array of depth values in meters), `point_cloud` ($N \times 3$ array of 3D points), `foca_length_px` (float, estimated focal length in pixels)

`robot.execute_grasp(grasp_pose)`

Execute a grasp by moving the robot to the specified pose via a pre-grasp point, and closing the gripper.

- **Inputs**:
  - `grasp_pose`: 4×4 transformation matrix representing the grasp pose in the robot's camera frame, OpenCV convention
- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float), `image` (PIL Image)
  - **Text**: Status of the grasp execution
  - **Image**: View from robot camera after the grasp is executed
  - **Variables**: `captured_image` after the grasp is executed

`robot.place_object_at_2d_location(point_2d)`

Move the robot to a place it's currently held object based on a 2D normalized image coordinate. The tool will convert to a 3D placement location automatically by shooting a ray.

- **Inputs**:
  - `point_2d`: 2D normalized image coordinate where the object should be placed

- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float), `image` (PIL Image)
  - **Text**: Status of the release operation
  - **Image**: View from robot camera after the placement is executed
  - **Variables**: `captured_image` (PIL Image) after the placement is executed

`robot.place_object_at_3d_location(point_3d)`
Move the robot to a 3D placement point and open the gripper to place the object.

- **Inputs**:
  - `point_3d`: 3D point in the robot's camera frame (list or numpy array) where the object should be placed
- **Outputs**:
  - **Raw value**: `success` (boolean), `execution_time_s` (float), `image` (PIL Image)
  - **Text**: Status of the placement operation
  - **Image**: View from robot camera after the placement is executed
  - **Variables**: `captured_image` (PIL Image) after the placement is executed

C. Additional Method Details

C.1. Group Relative Policy Optimization

We employ Group Relative Policy Optimization (GRPO) [51] as our RL training algorithm. We present the details of GRPO below.

For each input $\mathcal{I}$, in total N rollout procedures are launched asynchronously under the current policy π_θ. Each rollout generates in total N multi-turn rollouts $O_1, O_2, \ldots, O_N$. Their rewards are calculated as $r_1, r_2, \ldots, r_N$. Each r_i is standardized into a relative advantage A_i via group computation:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_N\})}{\text{std}(\{r_1, r_2, \ldots, r_N\})}. \tag{1}$$

The policy is then optimized by minimizing the GRPO loss:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_i\Big[-\min\big(\rho_i A_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i\big) + \beta\, \text{KL}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)\Big], \tag{2}$$

where $\rho_i = \frac{\pi_\theta(i)}{\pi_{\text{ref}}(i)}$, and π_{ref} denotes the reference policy model, i.e., the VLM trained after stage 1. Here, ϵ and β are tunable hyperparameters controlling the clipping range and KL regularization strength. This formulation encourages the policy to increase the probability of high-reward responses while maintaining stability through KL regularization.

System Prompt

You are an expert in 3D spatial reasoning for robotics.
Given a spatial reasoning task, **follow this process**:
1. First, think about the reasoning process as an internal monologue the first time you receive the question, and every time you receive new information. Your reasoning process MUST be enclosed within <think> </think> tags.
2. After thinking, if you need additional information to answer the question, or conduct external control, call the appropriate tool.
3. When you receive a tool response, use that information to continue your analysis.
4. Once no further visual analysis or tool calls are needed, you **MUST** provide your final answer inside <answer> </answer> tags without detailed illustrations.
Example answer format: <answer> <Your final answer here> </answer>
Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools> </tools> tags:
<tools> {tools} </tools>
For each function call, return a **JSON** object with function name and arguments within <tool_call> </tool_call> tags:
<tool_call> {"name": <func-name>, "arguments": <args-json-object>} </tool_call>

Figure 7. **System prompt.** Instructional prompt guiding the model's reasoning, tool-call, and answer process.

C.2. Alternative Reward Design

Other reward for the pointing questions. Considering that pointing is the first step of solving many spatial reasoning tasks or using other tools, we have experimented with several different rewards for pointing before finalizing the NNDC reward. We show results in Appendix E.2, emphasizing the importance the reward design for tasks requiring explicit numerical estimation.

- **Binary:**

$$R_{\text{B}} = \begin{cases} 1.0, & \text{if the predicted point lies within} \\ & \text{the ground truth convex hull,} \\ 0, & \text{otherwise.} \end{cases}$$

- **Normalized Signed Distance to Hull (NSDH):**

$$R_{\text{NSDH}} = \begin{cases} 0.5 + 0.5\exp(s), & \text{if } s \le 0, \\ 0.5\exp(-s), & \text{if } s > 0, \end{cases}$$

where s is the signed distance from the predicted point to the convex hull boundary (negative inside, positive outside).

- **Normalized Area Change (NAC):** Let ΔA be the change in area after adding the predicted point to the convex hull, and A_0 be the original area. Then:

$$R_{\text{NAC}} = \exp\left(-\frac{\Delta A}{A_0}\right).$$

Similar to NNDC, by default, we also apply clipping with the binary accuracy term to emphasize precision for alternative non-binary rewards. (*e.g.*, $R = \max(R_{\text{NSDH}}, R_{\text{B}})$)

Format score details. In addition to task-specific rewards, we explored defining a *format score* to enforce the structural correctness of model outputs (as defined in the system prompt shown in Figure 7), but did not use it in the final training. The format score verifies that every <tool_call> tag is immediately preceded by a <think> tag, and that the single final <answer> is also directly preceded by a <think>. The output must contain exactly one <answer> tag, positioned at the end of the response. Additional optional constraints can be applied, such as requiring at least one <tool_call> in the output. Predictions failing to meet these structural requirements receive a format score of zero, while perfectly formatted predictions receive a score of one.

The final reward is computed as a weighted sum of the accuracy-based reward and the format score:

$$R_{\text{final}} = R_{\text{acc}} + \lambda\, R_{\text{format}},$$

where R_{acc} denotes the task-specific accuracy reward, $R_{\text{format}} \in \{0, 1\}$ is the format score (equal to 1.0 if all structural criteria described above are satisfied, and 0 otherwise), and $\lambda \in [0, 1]$ is a scalar weight controlling the influence of the format score. Following prior work, λ is often set to 0.3.

D. Additional Implementation Details

D.1. More Training Details

For Phase-1 SFT of the base model and Phase-2 IRL, we use the system prompt shown in Figure 7. To improve the effectiveness of Phase-1 IRL training for the teacher model, we augment the prompt with two tool-use examples: one demonstrating how to solve a spatial relationship problem using the pointing tool, and another illustrating 2D bounding box estimation with the same tool. Moreover, when generating teaching data with the Claude Sonnet 4.5 model, we include additional instructional prompts that encourage careful interpretation of tool outputs and help understanding of image coordinate systems.

Due to the substantial latency of robot-in-the-loop training and data collection, we construct grasp and place data using the HOPE dataset and provide it to Claude Sonnet 4.5 alongside mock robot tools to generate the robotic portion of the teaching dataset. The interactive learning stages themselves do not incorporate this synthetic robot-tool component; instead, they focus solely on spatial reasoning with vision tools.

Another practical consideration is ensuring balanced answer distributions for multiple-choice questions. For example, the original RoboSpatial VQA dataset contains more than 75% "no" answers, which biases the VLM toward predicting "no" during both SFT and IRL. We found rebalancing the data mitigates this issue and improves answer calibration across tasks.

Finally, the major hyperparameters used during training are summarized in Table 6. Through our experiments, we find that the KL coefficient is a critical hyperparameter: a relatively small KL value is necessary to encourage sufficient exploration during RL. However, this introduces a trade-off in training stability—specifically, we observe an initial drop in rewards during Phase-1 IRL when using a smaller KL coefficient. We experimented with format rewards, format penalties, alternative KL loss formulations, and related variants, but were unable to eliminate this effect, suggesting that further investigation is needed.

Table 6. Training configurations for Phase-1 IRL, Phase-1 SFT, and Phase-2 IRL. A dash (–) indicates that the setting is not applicable to that phase.

	Phase-1 IRL	**Phase-1 SFT**	**Phase-2 IRL**
Data			
Dataset	Direct VQA	Teaching tool-use	Direct VQA
#Samples	4k	8k	≈8k
Model			
Trainable Part	Language model (LLM) only; vision encoder + projector frozen		
#Tunable Parameters	2.55B		
Training			
Batch Size	64	8	64
Learning Rate	1e-6	1e-5	1e-6
Epoch	5	2	2
Warmup Ratio	0.0	0.1	0.0
LR Schedule	NA	cosine	NA
KL Coefficient	1e-4	–	1e-4
Entropy Coefficient	0.0	–	0.0
Temperature	1.0	–	1.0
Max Prompt Length	8192	8192	8192
Max Response Length	8192	8192	8192
Rollout Number	5	–	5
#GPU (VLM)	8	8	8
#GPU (Tools)	8	–	8

D.2. MACE Metric for Grasp Affordances

The grasp estimation task requires the model to predict five key points in normalized image coordinates: the grasp center, left finger base, right finger base, left finger tip, and right finger tip. From these points, we define four finger direction vectors: grasp center → left finger base, grasp center → right finger base, left finger base → left finger tip, and right finger base → right finger tip.

The Mean Angular Coordinate Error (MACE) metric is defined as follows. Given the predicted and ground-truth grasp centers $\hat{c}$ and c, and the set of four corresponding finger direction vectors $\{\hat{r}_k\}_{k=1}^4$ and $\{r_k\}_{k=1}^4$, we define:

$$\text{MACE} = 1 - \frac{1}{2}\left(\frac{\|\hat{c} - c\|_2}{w} + \frac{1}{4}\sum_{k=1}^{4}\frac{1 - \cos(\hat{r}_k, r_k)}{2}\right), \tag{3}$$

where w denotes the gripper width used for spatial normalization, and $\cos(\hat{r}_k, r_k) = \frac{\hat{r}_k \cdot r_k}{\|\hat{r}_k\|\|r_k\|}$ represents the cosine similarity between the predicted and ground-truth orientations of the k-th finger-related vector.

Task	Ours	Claude	GPT
Pick			
Pick up the dark blue object	✓	✓	✓
Pick up the soft toy	×	×	×
Pick up the solid toy	✓	✓	×
Pick up the tall cylindrical tennis ball container	✓	✓	✓
Pick up the coconut water	✓	✓	✓
Pick up the plastic bottle	✓	✓	✓
Pick up the red box	✓	✓	✓
Relational Pick			
Pick up the far coconut water	✓	×	×
Pick up the coconut water that is closer to the camera	✓	✓	×
Pick up the left pineapple juice can	✓	×	×
Pick up the right pineapple juice can	✓	✓	×
Pick up the further purple drink	×	✓	✓
Pick up the near purple bottle	✓	×	✓
Pick & Place			
Pick up the hot dog and place it in the black bin	0	2	2
Pick up the tall cylindrical container and place it in the transparent bin	2	2	2
Pick up the leftmost condiment and place it in the transparent bin	2	2	2
Pick up the cinnamon and place it in front of the rice box	2	2	0
Pick up the rice box and place it next to the hot sauce	2	1	0
Pick up the plushie and place it left of the coconut water	2	1	1
Pick up the pony and place it left of the two plushies	2	1	2

Table 7. Per-task breakdown of the real-world manipulation results, comparing **Ours** (SpaceTools), Claude Sonnet 4.5 and GPT-5.

D.3. Robot Manipulation Setup

Robot System We conduct robot experiments on a Kinova Jaco arm equipped with the CuRobo [56] motion planner and a ZED2 RGB-D camera. We expose the robot as a tool and make it available to the VLM. The tool has functions: `capture_image` retrieves the current RGB image from the camera, `get_depth` and `get_depth_with_pointcloud` retrieve the current depth image, optionally with a pointcloud in the robot frame, `execute_grasp` moves the end-effector to a specified grasp pose via a pre-grasp point and closes the gripper, `place_object_at_2d_location` and `place_object_at_3d_location` offer two ways to parameterize a place operation that moves the robot hand holding an object over a location in the scene and opens the gripper. All motions are executed with the motion planner.

Robot Experiments Tasks and Results We design a suite of tasks across three categories. *Pick*, *Relational Pick*, and *Pick & Place*. We score both *Pick* tasks based on the success rate, and *Pick & Place* based on partial success rate, awarding 1 point each for a correct pick and place operation.

The full results at individual task level are in Table 7, omitting methods that fail to achieve any points. In *Pick up the soft toy* task, all models failed due to a common failure in pointing tool not being able to differentiate the soft toy from a rigid toy. In *Relational Pick* and *Pick & Place* tasks, SpaceTools shows superior ability than Claude and GPT-5 in correctly using the pointing tool to resolve spatial relations, reflecting an understanding of its strengths and limitations likely attributable to the interactive training with the tool.

Additional Details on Robot SFT Data Collection In order to collect SFT data of from the Universal Teacher (Claude Sonnet 4.5) on using the robot tool, we design a "mock robot" that has the same API as the robot tool, but it always simulates successful actions provided the API was called with valid arguments. This allows collecting a small number ($\sim$500) examples of valid robot API calls without requiring the physical robot in the loop and ensuring that our robot is unseen during training.

D.4. Details of Non-interactive RL Baseline

We present the detailed description of the Tool NIRL baseline referenced in the ablation section of the main paper, as space limitations prevented us from including all details there.

We follow the conventional tool-learning setup used in LLMs [79] to perform reinforcement learning of tool usage *without* executing tools during training. The core idea is to compute the reward solely from the correctness of the predicted tool name and its arguments, which requires access to ground-truth tool call traces for supervision. After obtaining tool-augmented reasoning traces from Claude, each multi-turn trace with T turns is decomposed into T single-turn training instances: the i^{th} instance contains the conversation history up to turn i as input, and the corresponding ground-truth output for turn i as the target.

During training, for tool-call turns, we adopt the binary reward used in [79]. A reward of 1.0 is given only when both the tool call format and the tool call content match the ground truth:

$$r = \begin{cases} 1, & \text{if } \texttt{FormatCorrect} \wedge \texttt{ToolCallMatch} \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where `FormatCorrect` verifies that the model output is wrapped in the required tags, and `ToolCallMatch` checks that both the tool name and its arguments exactly match the ground-truth tool call. For final-answer turns (i.e., non-tool turns), we reuse the same task-specific normalized rewards introduced in this paper.

E. Additional Experimental Results

E.1. A Closer Look at Tool IRL Alone.

While direct IRL over diverse tools poses challenges due to the vast action space, we demonstrate its effectiveness

Table 8. Comparison of Qwen2.5-VL-3B, its inference variants, and fine-tuned models on RoboSpatial and RefSpatial. *Direct Inference* refers to answering the question without intermediate reasoning or tool use. *CoT* denotes chain-of-thought inference. *+Toolshed* indicates tool-augmented inference without any additional training. Among all variants, Tool IRL achieves the highest overall accuracy on RoboSpatial and the strongest generalization to RefSpatial.

Model	RoboSpatial			RefSpatial
	VQA	Vacant	Overall	
Inference Baseline (no fine-tuning)				
Direct Inference	53.07	0.00	35.71	0.00
CoT	66.67	0.00	43.71	0.00
+Toolshed	47.37	9.02	34.00	17.69
Fine-tuned on RoboSpatial				
Tool-free SFT	75.88	13.11	54.00	0.00
Tool-free RL	72.37	20.49	54.28	0.00
Tool IRL	77.64	62.30	72.30	34.30

within a constrained setup using the RoboSpatial dataset and pointing tools. As shown in Table 8, this approach substantially improves spatial reasoning compared with both direct tool-free SFT and vanilla tool-free GRPO baselines (e.g., classic reasoning models like R1 [13]), as well as other inference approaches [66]. On RoboSpatial, the IRL with Tools model achieves 72.3% overall accuracy, outperforming SFT and vanilla GRPO. Notably, IRL with Tools is the only method that generalizes to unseen tasks: achieving 34.3% on RefSpatial, whereas other fine-tuning strategies yield zero accuracy. These results show that interactive tool use during RL enables the model to internalize transferable geometric reasoning skills beyond the training domain. Even without fine-tuning, connecting Toolshed to the pretrained model yields measurable improvements on RefSpatial, highlighting the intrinsic generalization benefit of tool-augmented spatial reasoning.

E.2. Other Ablations

Direct IRL on all tasks with all tools. As mentioned in the main paper, directly reinforcement learning with all tools on all tasks result in a large search space and is hard to learn effectively. We provide qualitative performance in Table 9, supporting this argument.

Table 9. Direct IRL on all tasks (*Direct IRL All.*) with all tools compared with our method.

Variant	IRL-T	Univ-T	S2-IRL	RoboSpatial	RefSpatial	Pose	Mean
SpaceTools (Ours)	✓	✓	✓	70.00	53.07	34.37	52.48
Direct IRL All.	×	×	✓	52.86	3.25	3.26	19.79

Reward and prompt for IRL. Due to computational constraints that prevent running full-scale IRL ablations, we evaluate different pointing rewards (*e.g.*, NSDH, NAC, Binary) and prompt design choices introduced in Appendix C.2 on a subset of 1k vacant-space localization questions from RoboSpatial using the Molmo pointing tool. The results in Table 10 show that NNDC without an additional format reward yields the most stable and reliable learning behavior. Accordingly, we adopt NNDC (without a format reward) for all subsequent training stages. These experiments also highlight the importance of normalizing rewards to the $[0, 1]$ range, a practice we apply consistently across all tasks. More broadly, this study underscores the richness of the reward-design space for spatial reasoning tasks, especially those requiring explicit numerical estimation.

Table 10. Ablation on reward and prompt design for the pointing task as introduced in Appendix C.2. *Norm.* indicates whether normalization to range $[0, 1]$ is applied to the reward function. *Clip.* indicates whether binary clipping is applied. *Format* indicates whether the format reward is applied. *Example in Prompt* indicates whether two tool-use examples are added in the prompt. Checkmarks indicate which components are included for each variant.

Reward Variant	Norm.	Clip.	Format	Example in Prompt	Acc.
NNDC (Ours)	✓	✓	×	✓	**35.25**
w/o Clip.	✓	×	×	✓	14.8
w/o Norm.	×	✓	×	✓	0.00
w Format.	✓	✓	✓	✓	33.61
w/o Example.	✓	✓	×	×	17.21
NSDH	✓	✓	×	✓	21.31
w/o Clip.	✓	×	×	✓	22.31
w/o Norm.	×	✓	×	✓	0.00
NAC	✓	✓	×	✓	22.95
w/o Clip.	✓	×	×	✓	22.95
w/o Norm.	×	✓	×	✓	0.00
Binary	✓	✓	×	✓	15.57

Dataset size and type for IRL. We conduct preliminary experiments on how dataset size and data-type composition affect IRL performance using the RoboSpatial dataset and the Roborefer pointing tool. RoboSpatial contains four data types: configuration, compatibility, grounding, and vacant-space localization. Configuration and compatibility are binary yes/no questions, grounding requires predicting 2D bounding boxes, and vacant-space localization involves predicting a free-space location. We vary the mixture of these four types and evaluate performance on RoboSpatial-Home, with results summarized in Table 11. Notably, for configuration and compatibility, we ensure a balanced distribution of yes/no answers, as discussed in Appendix D. Interestingly, although grounding data are not present in RoboSpatial-Home, including grounding during training improves performance on the other tasks. In contrast, in-

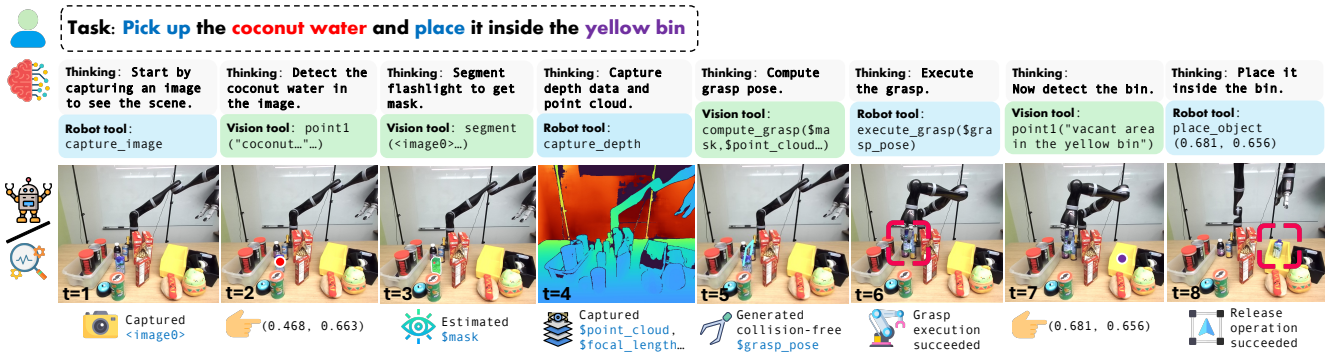


Figure 8. A hard real-world robot manipulation example with SpaceTools. The model successfully identifies the target object and completes the manipulation task in a cluttered and visually complex scene

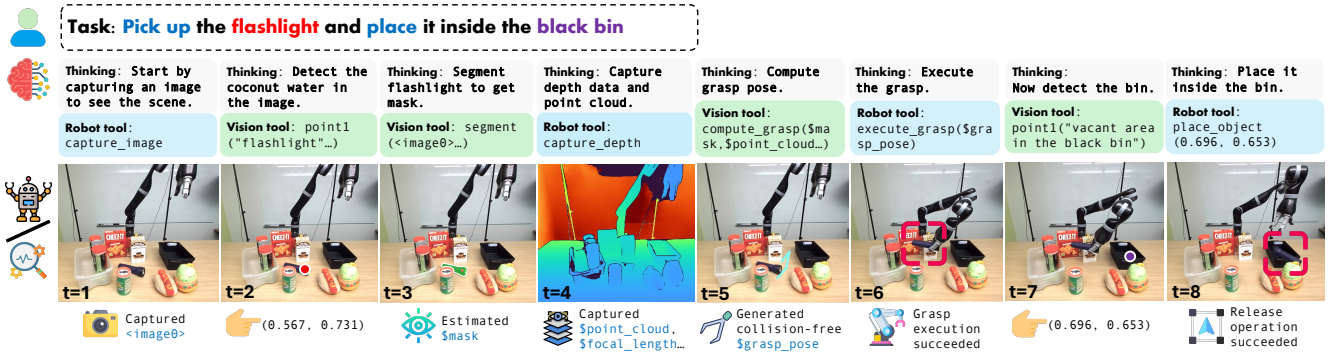


Figure 9. A failure case in real-world robot manipulation with SpaceTools. The model localizes a valid vacant area but selects a point too close to the boundary, resulting in a failed placement of the flashlight on the box boundary.

creasing the overall dataset size beyond a moderate scale yields limited gains, suggesting that data diversity and label balance contribute more to IRL effectiveness than raw quantity alone.

Table 11. Evaluation on RoboSpatial-Home using models trained with Tool IRL under different data compositions drawn from the four RoboSpatial data types. *Config.* refers to configuration data, *Compat.* to compatibility data, *Ground.* to grounding (2D bounding box) data, and *Vacant* to vacant-space localization data. Each entry in the middle columns indicates the number of samples included for that data category. *Overall Acc.* reports the final accuracy on RoboSpatial-Home.

Variant	Config.	Compat.	Ground.	Vacant	Overall Acc.
All-v1	0.5k	0.5k	0.5k	0.5k	69.70
All-v2	1.0k	1.0k	1.0k	1.0k	69.70
All-v3	2.0k	2.0k	1.0k	1.0k	69.10
w/o Ground.	2.0k	2.0k	0.0k	2.0k	56.90

E.3. Additional Demonstrations

Visualizations of success cases. Apart from the cases where the grasp tool fails to find a collision-free pose or the pointing tool fails to localize points accurately for relative depth questions, we also present detailed examples of successful tool executions containing all special format-related tokens in Figure 10. In this example, the grasp tool successfully finds an accurate pose in a cluttered scene. Figure 11 provides examples for relative depth questions. One example shows the tool accurately predicts the required point locations, enabling the VLM to produce the correct answer. The other example shows multiple chairs are present in the image, and the VLM must identify the specific one highlighted by the provided bounding box. Moreover, as illustrated in Figure 8, SpaceTools can reliably identify target objects and execute the required manipulation steps even in cluttered, visually complex real-world environments.

Visualizations of failure cases. Although the model performs well across many scenarios, it is not universally reliable. We therefore examine representative failure cases to highlight remaining challenges and motivate future exploration. Grasp estimation currently yields the lowest accuracy, largely due to difficulties in detecting the target object in cluttered scenes and in predicting accurate yet collision-free grasp poses, as illustrated in Figure 12. Object detec-

Grasp

What are the five normalized (x, y) points between 0 and 1 in the image that outline the grasp plane for the box of oatmeal-raisin cookies? Your answer should be as Grasp center: [], Left finger base: [], Right finger base: [], Left finger tip: [], Right finger tip: [].

Thinking: <think> I need to find the grasp plane points for the box of oatmeal-raisin cookies. To do this, I should:
1. First detect the oatmeal-raisin cookies box in the image
2. Segment it to get a mask
3. Estimate depth to get a point cloud
4. Compute a grasp pose
5. Extract the five key points that define the grasp plane
Let me start by detecting the oatmeal-raisin cookies box. </think>

Tool calls:
<tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "oatmeal-raisin cookies", "image_index": 0}}</tool_call>

<tool_response>Detected 1 instance(s) of 'oatmeal-raisin cookies': [(0.479, 0.417)]</tool_response>

Thinking: <think> Good, I found the oatmeal-raisin cookies at coordinates (0.479, 0.417). Now I need to segment it to get a mask of the object. </think>

Tool calls: <tool_call>{"name": "sam2.segment_from_point", "arguments": {"x": 0.479, "y": 0.417, "image_index": 0}}</tool_call>

<tool_response>Segmented object at normalized coords (0.479, 0.417) with IoU score: 0.940. Use $segmentation_mask (boolean array, 289x511) to reference it.</tool_response>

Thinking: <think> Great! I have the segmentation mask. Now I need to estimate depth and generate a point cloud so I can compute the grasp pose. </think>

Tool calls: <tool_call>{"name": "depth_estimator.estimate_depth_with_pointcloud", "arguments": {"image_index": 0}}</tool_call>

<tool_response>Generated 511x289 depth map and 147,679 3D points. Depth range in point cloud: 1.01m to 1.44m (mean: 1.18m). Use $depth_map (numpy array, 289x511), $point_cloud (numpy array, 147679x3) and $focal_length_px (float) to reference the results.</tool_response>

Thinking: <think> Perfect! Now I have:- $segmentation_mask: the mask of the oatmeal-raisin cookies box- $point_cloud: the 3D point cloud- $focal_length_px: the focal length.
Now I can compute the grasp pose for the object. </think>

Tool calls: <tool_call> {"name": "grasp_generator.compute_grasp", "arguments": {"point_cloud": "$point_cloud", "mask": "$segmentation_mask", "focal_length_px": "$focal_length_px", "image_index": 0}} </tool_call>

<tool_response>Generated collision-free grasp with confidence 0.978 from 1980/147679 masked points. Filtered 100 → 12 collision-free grasps. Projected 2D gripper points: [(0.501, 0.213), (0.460, 0.168), (0.546, 0.261), (0.461, 0.262), (0.543, 0.359)].</tool_response>

Thinking: <think> Excellent! The grasp generator has computed a grasp pose and provided the five projected 2D gripper points in normalized coordinates. These points represent: 1. Grasp center: (0.501, 0.213) 2. Left finger base: (0.460, 0.168) 3. Right finger base: (0.546, 0.261) 4. Left finger tip: (0.461, 0.262) 5. Right finger tip: (0.543, 0.359)
These five points outline the grasp plane for the oatmeal-raisin cookies box. </think>

Answer: <answer>
Grasp center: [0.501, 0.213],
Left finger base: [0.460, 0.168],
Right finger base: [0.546, 0.261], Left finger tip: [0.461, 0.262], Right finger tip: [0.543, 0.359]</answer>

Figure 10. A detailed example of tool-augmented reasoning of a grasp estimation question.

tion in complex, visually crowded environments remains a major bottleneck for both grasp and pose estimation, leaving substantial room for improvement. These issues point not only to opportunities for improving individual tools but

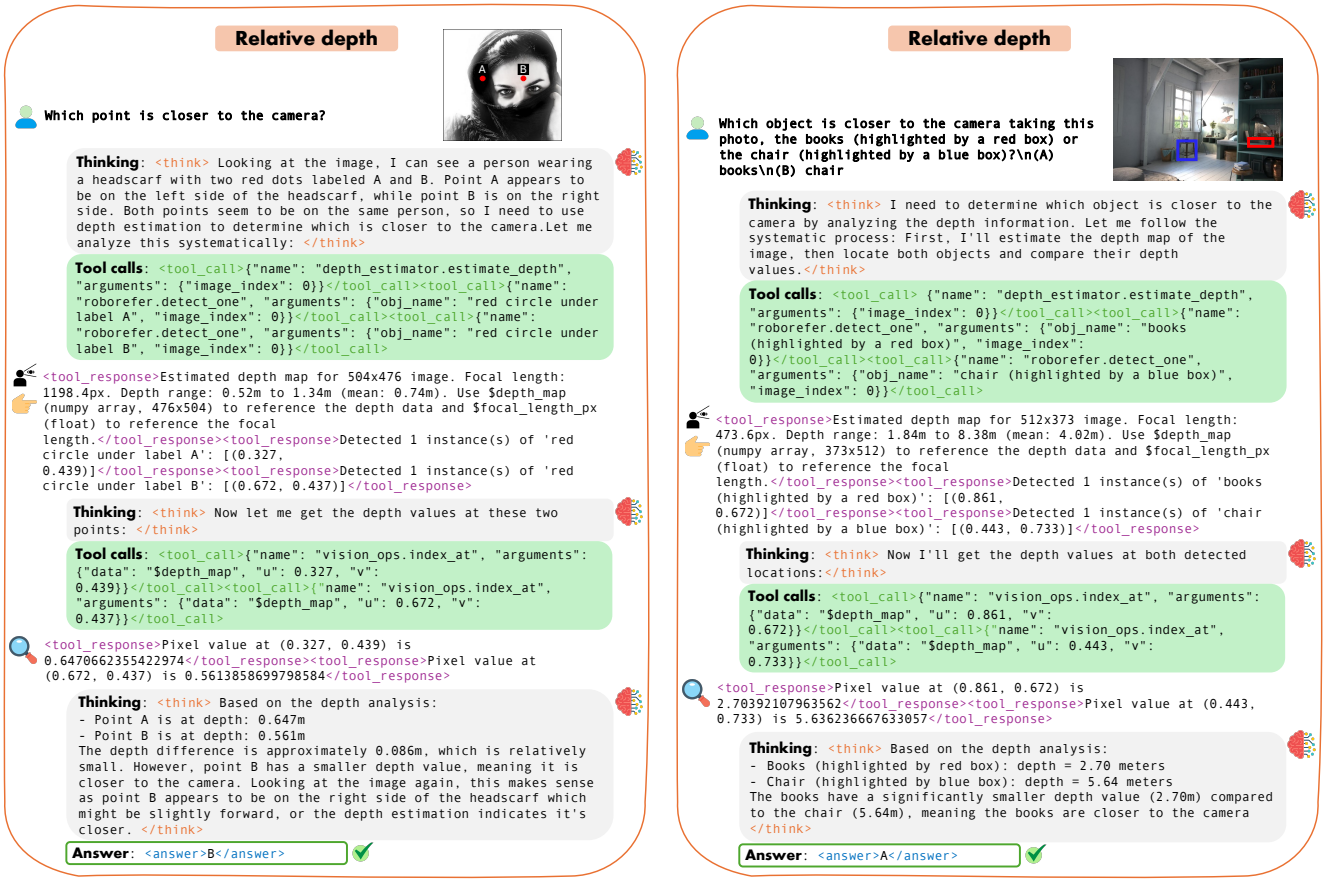
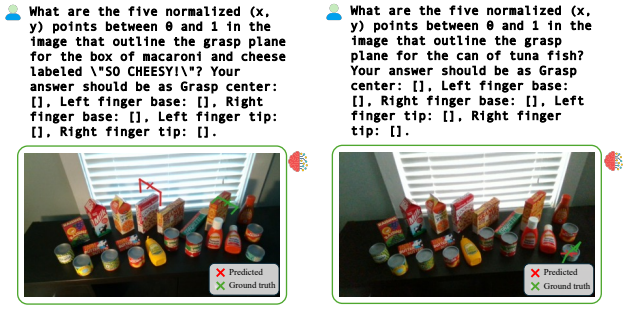


Figure 11. Detailed examples of tool-augmented reasoning of relative depth questions.



(a) Wrong object localization (b) Inaccurate pose estimation

Figure 12. Failure cases for the grasp estimation task. Intermediate tool-augmented reasoning steps are omitted for clarity.

also to the potential of modular enhancements to spatial reasoning by strengthening specific tool components instead of retraining or finetuning the VLM.

Another failure example of real-world robot manipulation is shown in Figure 9. This failure case highlights a subtle but important challenge in real-world manipulation. Although the model correctly identifies a vacant region, it selects a point near the boundary, and thus, the robot finally performs the placement on the boundary. Such cases underscore the need for highly precise geometric reasoning and tighter coupling between 2D point predictions and physical feasibility on the robot. They also reveal a current limitation of the model's tool coordination and point-selection strategy, suggesting promising directions for future improvements, including integrating real or simulated robot feedback into the training process.