MolmoAct: Action Reasoning Models that can Reason in Space

Jason Lee[1,2,*,†], Jiafei Duan[1,2,*,†], Haoquan Fang[1,2,*,†], Yuquan Deng[1,†], Boyang Li[2,†], Shou Liu[1,2,†], Bohan Fang[2,†], Jieyu Zhang[1,2,†], Yi Ru Wang[1,2,†], Sangho Lee[1], Winson Han[1], Wilbert Pumacay[1], Angelica Wu[2], Rose Hendrix[1,†], Karen Farley[1], Eli Vanderbilt[1], Ali Farhadi[1,2,†], Dieter Fox[1,2,†], Ranjay Krishna[1,2,†]

[1]Allen Institute for AI [2]University of Washington

[*]Equal contribution [†]Core contributors

Abstract: Reasoning is central to purposeful action, yet most robotic foundation models map perception and instructions directly to control, which limits adaptability, generalization, and semantic grounding. We introduce Action Reasoning Models (ARMs), a class of robotic foundation models that integrates perception, planning, and control through a structured three-stage pipeline. Our model, MOLMOACT, encodes observations and instructions into depth-aware perception tokens, generates mid-level spatial plans as editable trajectory traces, and predicts precise low-level actions, enabling explainable and steerable behavior. MOLMOACT-7B-D achieves strong performance across simulation and real-world settings: 70.5% zero-shot accuracy on SimplerEnv Visual Matching tasks, surpassing closed-source π_0 and GR00T N1; 86.6% average success on LIBERO, including a $+6.3\%$ gain over ThinkAct on long-horizon tasks; and in real-world fine-tuning, $+10\%$ (single-arm) and $+22.7\%$ (bimanual) task progression over π_0-FAST. It also outperforms baselines by $+23.3\%$ on out-of-distribution generalization and achieves top human-preference scores for open-ended instruction following and trajectory steering. Furthermore, we release, for the first time, the MOLMOACT DATASET —a mid-training robot dataset comprising over 10,000 high-quality robot trajectories across diverse scenarios and tasks. Training with this dataset yields an average 5.5% improvement in general performance over the base model. We release all model weights, training code, MOLMOACT DATASET and our action reasoning dataset, establishing MOLMOACT as both a state-of-the-art robotics foundation model and an open blueprint for building ARMs that transform perception into purposeful action through grounded reasoning.

Keywords: Action Reasoning Model, Reasoning in Space, Vision-Language-Action Model, Robots, Learning, Manipulation

1 Introduction

Reasoning allows us to act with intention. Before reaching for a cup or moving through a room, we subconsciously weigh context, goals, and constraints—transforming perception into purpose. This process, grounded in our physical experience of the world, makes our actions coherent, adaptable, and explainable. For robots to operate with the same fluency, they must do more than map images and instructions to robot control. They must learn to reason.

In contrast to the rapid generalization gains seen in large language and vision models, progress in robotics has lagged behind [1, 2, 3]. Vision-Language-Action (VLA) models [4, 5, 6, 7, 8] aim to bring similar capabilities to physical agents, but have yet to reach the same level of flexibility or robustness. Despite massive efforts in dataset collection and model scaling, today's VLAs remain

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

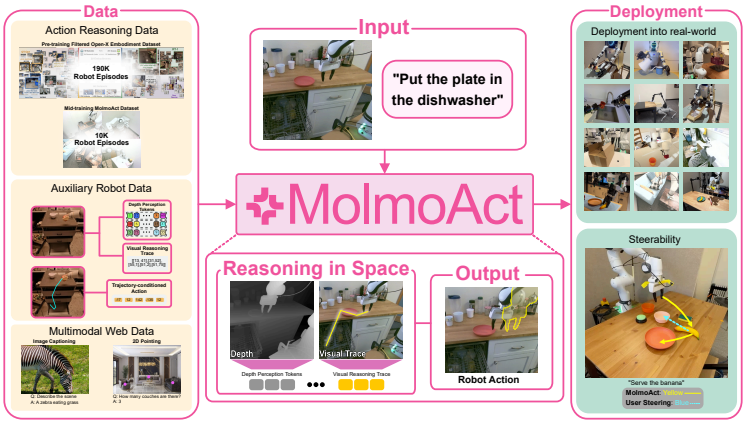


Figure 1: **Overview**. MOLMOACT is an open action reasoning model that, given a user's language instruction, reasons in space and autoregressively predicts three structured reasoning chains: DEPTH PERCEPTION TOKENS for sensing and reconstructing the 3D environment, VISUAL REASONING TRACE TOKENS for representing its planned trajectory in the scene, and ACTION TOKENS for generating the corresponding robot control commands. Each explainable reasoning chain can be independently decoded—yielding a depth map of the scene, a 2D trajectory overlay on the image plane, and executed actions in the physical world—providing explicit, spatially grounded reasoning at every stage.

brittle and opaque—struggling to transfer across tasks, scenes, or embodiments, and offering little insight into why a robot chose one action over another [9, 10].

This gap stems not just from limited data, but from a lack of structure. While language and vision tasks benefit from abundant, loosely labeled web-scale data, robotics demands fine-grained, embodied interaction—data that is costly, ambiguous, and difficult to scale. Yet in parallel, language models have begun to shift away from brute-force scaling toward *structured learning*: building intermediate representations that support reasoning, abstraction, and control [11, 12, 13]. We believe robotics can—and must—do the same.

We introduce MOLMOACT (**M**ultimodal **O**pen **L**anguage **Mo**del for **Act**ion), a family of completely open **A**ction **R**easoning **M**odels (ARM) that integrate perception, planning, and control through a structured reasoning pipeline. MOLMOACT learns to interpret language instructions, sense its environment, generate spatial plans, and execute them as smooth, goal-directed trajectories. The model first encodes observations and instructions into structured 2.5D representations via autoregressive prediction of depth-aware perception tokens. These tokens condition the generation of mid-level planning representations, which, when visualized as visual traces in image space, guide the prediction of precise, low-level robot actions. This three-stage reasoning architecture enables MOLMOACT to produce explainable and steerable behavior as shown in Figure 1.

MOLMOACT's structured design delivers both strong performance and high explainability. On standard benchmarks such as LIBERO and SimplerEnv (Google Robot), MOLMOACT consistently outperforms competitive baselines including GR00T N1 [7], π_0 and π_0-FAST [4], RT-1 [14], and TraceVLA [15]. In arena-style human evaluations for open-ended language instruction following, MOLMOACT is preferred over baselines, achieving significantly higher Elo ratings. The model adapts to novel tasks more effectively through lightweight fine-tuning, surpassing other strong baselines in efficiency. Moreover, it generalizes well to diverse environments and task perturbations in both simulation and real-world settings. Its visual reasoning traces offer an explainable view into the model's decision-making, while also enabling direct action steering by editing trajectory lines—an approach we find more reliable than language commands, which can suffer from ambiguity.

MOLMOACT is fully open in every aspect: we release the model weights, training code, and all components of our action reasoning dataset. We aim for MOLMOACT to be more than a high-

performing robotics foundation model that serves as a blueprint for building agents that reason, transforming perception into purposeful action through reasoning. We provided related work in Appendix A.

2 MolmoAct

In the following sections, we describe the VLM preliminaries (subsection 2.1), our method to adapt VLMs for action prediction via action tokenization(subsection 2.2), how we transform Molmo into an action reasoning model (ARM, subsection 2.3), and our approach to steer action by visual reasoning traces (Appendix B).

2.1 Vision Language Model

Our work builds on Molmo, the Multimodal Open Language Model, which follows this standard design. It employs a Vision Transformer (ViT) visual encoder, a two-layer MLP connector for projecting vision features into the language embedding space, and a decoder-only LLM backbone. In our implementation, we use vision encoders such as OpenAI ViT-L/14 336px CLIP [16] and ViT-SO400M/14 384px SigLIP2 [17], paired with open LLMs including OLMo2-7B [18] and Qwen2.5-7B [19]. We trained MOLMOACT-7B-O with a VLM backbone based on OpenCLIP and OLMo2-7B, and MOLMOACT-7B-D with a backbone based on SigLIP2 and Qwen2.5-7B. For full details of our model architecture and implementation, please refer to Appendix D.

2.2 Vision-Language-Action Model

A standalone VLM—even when expertly prompted—cannot directly control a robot: it lacks a representation of the robot's action space and dynamics, and thus can only provide high-level planning over the current observation. To produce accurate, executable commands, we follow prior work [14, 5, 20] in formulating action prediction as a vision–language sequence modeling task. For each action dimension, we normalize using dataset quantiles and discretize into 256 uniform-width bins between the first and ninety-ninth percentiles, which reduces the influence of outliers while preserving effective granularity. We first identify the final 256 tokens in the Qwen2 tokenizer and, for each, use its underlying byte-level BPE symbol. We then assign them monotonically to the 256 bins so that adjacent bins map to adjacent symbols.

2.3 Action Reasoning Model

Chain-of-Thought (CoT) [11] has been shown to significantly improve Language Models' performance on complex tasks. Likewise, Multimodal Language Models (MLLM) also benefit from multimodal Chain of Thought (MCoT) [21] in processing long multimodal contexts. However, this "think-before-you-act" paradigm is rarely present in robotic control policies. While some work attempts to incorporate reasoning to VLAs, they focus on high-level language reasoning [22, 23, 24] such as decomposing a high-level semantic task into subtasks. While useful, they ignore two crucial aspects for precise control: depth perception and precise motion planning. First, most VLMs are trained solely on RGB images and hence lack the ability of depth estimation and 3D understanding, which is critical for robotic manipulation. Moreover, attempting to distill complex 3D trajectories into linguistic descriptions often results in significant loss of spatial and temporal information.

Contrast to previous approaches, MOLMOACT does not incorporate intermediate reasoning through language; rather, we teach our models to reason in space. Conditioned on images and instructions, the model autogregressively generates a sequence of *depth perception tokens*, followed by *visual reasoning trace* of the intended end-effector motion, before predicting the action tokens.

Action Reasoning Procedure With depth perception tokens and visual reasoning trace, MOLMOACT can finally perform action reasoning in space with the following procedure: given an RGB

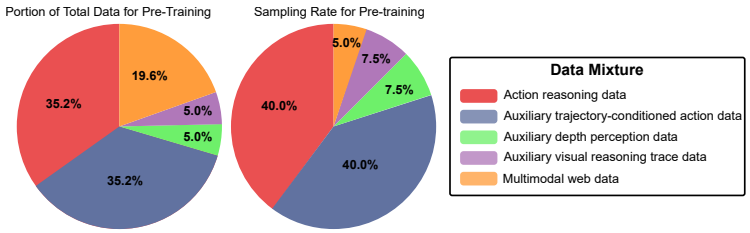


Figure 2: Distribution of data mixture in the overall pre-training mixture (left) and in the sampled subset used for MOLMOACT pre-training (right). The mixture contains primarily action reasoning data (38.7%), trajectory-conditioned data (38.7%), and multimodal web data (21.5%), with small fractions of auxiliary depth and trace data (0.5% each). The sampled subset increases the proportion of auxiliary data (7.5% each for depth and line) while reducing multimodal web data to 5%.

image observation I and a language instruction T (which includes an action CoT prompt), the model autoregressively generates three token sequences in order: (i) depth perception tokens $\mathbf{d}$; (ii) visual reasoning trace $\boldsymbol{\tau}$; (iii) action tokens $\mathbf{a}$, where $\mathbf{a} = (a_1, \ldots, a_D) \in V_{\text{action}}^D$ with D degree of freedoms. These are produced according to the factorization

$$p(\mathbf{d}, \boldsymbol{\tau}, \mathbf{a} \mid I, T) = \prod_{i=1}^{M+2} p(d_i \mid I, T, \mathbf{d}_{<i}) \times \prod_{j=1}^{L} p(\tau_j \mid I, T, \mathbf{d}, \boldsymbol{\tau}_{<j}) \times \prod_{k=1}^{D} p(a_k \mid I, T, \mathbf{d}, \boldsymbol{\tau}, \mathbf{a}_{<k}). \tag{1}$$

3 Data Curation and Generation

MOLMOACT is trained on a diverse set of datasets. During Pre-training, MOLMOACT is trained on Multimodal Web data, Auxiliary Robot Data as well as Action Reasoning Data. Furthermore, we collected and trained with the MOLMOACT DATASET for the Mid-training stage. Below, we describe each dataset and its collection process; further details and examples are provided in the Appendix H.

3.1 Action Reasoning Data

A robot episode typically consists of a sequence of timesteps, where each timestep is a tuple $(I, T, \mathbf{a})_t$, containing an RGB observation image I, a language instruction T, and a ground-truth action $\mathbf{a}$, specified either in end-effector space or joint space. To convert any robot data into the Action Reasoning data format, we generate ground-truth *Depth Perception Tokens* and *Visual Reasoning Traces* for each timestep in the episode. We explain the details for generating ground-truth labels for Depth Perception Tokens and Visual Reasoning Trace below.

Depth Perception Tokens To generate *Depth Perception Tokens* for each frame of a demonstration, we first train a VQVAE on 10 million depth maps of tabletop manipulation images collected from the RT-1, BridgeData V2, and BC-Z datasets. We use DepthAnything-v2 to obtain a depth map for each observation RGB image. The VQVAE is trained with a standard reconstruction objective to minimize reconstruction loss between input RGB images and their corresponding depth maps for 20 epochs. Once the VQVAE has been trained, we encode each observation image with the VQVAE to get their latent embeddings. We then represent the latent embedding with a learned codebook with 128 dimension based on a one-to-one index to depth token mapping. Note that all images are resized to 320×320 px during training and inference to enforce the representation of 100 tokens per image. This allows us to express the depth map of each observation image as a tokenized string of 100 tokens, which we use for ground truth labeling for our depth perception token.

Visual Reasoning Trace To generate a *Visual Reasoning Traces* for each frame of a demonstration, we employ Molmo, a vision-language model trained on diverse 2D pointing datasets, for data generation akin to synthetic data generation in NLP. For each timestep t, we extract the pixel

Table 1: **SimplerEnv evaluation across different policies on Google Robot tasks.** The zero-shot and fine-tuning results denote performance of OXE dataset [26] pre-trained models and RT-1 dataset [14] fine-tuned models, respectively.

Model	Visual Matching			Avg	Variant Aggregation			Avg
	Pick Coke Can	Move Near	Open/Close Drawer		Pick Coke Can	Move Near	Open/Close Drawer	
HPT [27]	56.0%	60.0%	24.0%	46.0%	—			—
TraceVLA [15]	28.0%	53.7%	57.0%	42.0%	60.0%	56.4%	31.0%	45.0%
RT-1-X [14]	56.7%	31.7%	59.7%	53.4%	49.0%	32.3%	29.4%	39.6%
RT-2-X [20]	78.7%	77.9%	25.0%	60.7%	82.3%	79.2%	35.3%	64.3%
Octo-Base [28]	17.0%	4.2%	22.7%	16.8%	0.6%	3.1%	1.1%	1.1%
OpenVLA [5]	16.3%	46.2%	35.6%	27.7%	54.5%	47.7%	17.7%	39.8%
RoboVLM (zero-shot) [9]	72.7%	66.3%	26.8%	56.3%	68.3%	56.0%	8.5%	46.3%
RoboVLM (fine-tuned)	77.3%	61.7%	43.5%	63.4%	75.6%	60.0%	10.6%	51.3%
Emma-X [22]	2.3%	3.3%	18.3%	8.0%	5.3%	7.3%	20.5%	11.0%
Magma [8]	56.0%	65.4%	83.7%	68.4%	53.4%	65.7%	68.8%	62.6%
π_0 (fine-tuned) [4]	72.7%	65.3%	38.3%	58.7%	75.2%	63.7%	25.6%	54.8%
π_0-FAST (fine-tuned)	75.3%	67.5%	42.9%	61.9%	77.6%	68.2%	31.3%	59.0%
GR00T N1 (fine-tuned) [7]	0.7%	1.9%	2.9%	1.8%		—		—
SpatialVLA [29]	81.0%	69.6%	59.3%	70.0%	89.5%	71.7%	36.2%	65.8%
MOLMOACT (zero-shot)	71.3%	73.8%	66.5%	70.5%	57.8%	43.8%	76.7%	59.3%
MOLMOACT (fine-tuned)	77.7%	77.1%	60.0%	**71.6%**	76.1%	61.3%	78.8%	**72.1%**

coordinates (u_t, v_t) of the robot's gripper, and aggregate these across the episode to obtain a visual reasoning trace. For each observation frame, we prompt Molmo with the instruction `"point to the robot gripper"` for single-arm robots or `"point to the robot gripper on the left/right"` for bimanual embodiments. Molmo returns a 2D coordinate $(x_t, y_t) \in \mathbb{R}^2$ and $(x_t, y_t) \in [0, 100]$, corresponding to the predicted gripper location in image space. We rescale the coordinate values so that $(u_t, v_t) \in \mathbb{Z}^2$ and $(u_t, v_t) \in [0, 255]$. We then apply this query at every timestep in the episode, resulting in one gripper location per frame. Linking these predictions sequentially yields the full trajectory $\boldsymbol{\tau}$.

Auxiliary Robot Data To strengthen MOLMOACT's ability to reason in space, we extend the same data generation pipeline used for depth perception token, visual reasoning trace to curate three auxiliary supervision dataset: (i) Auxiliary Depth Data—given an RGB observation and language instruction, the model only predicts the target Depth Perception Token sequence; (ii) Auxiliary Trace Data—given an RGB observation and language instruction, the model only predicts the corresponding Visual Reasoning Trace; and (iii) Trajectory-conditioned Action Data—given $o_t = (I, T, \boldsymbol{\tau})_t$, where I is the current image, T the instruction, and $\boldsymbol{\tau} = (p_1, \ldots, p_L)$ the ground truth Visual Reasoning Trace, the model predicts the next action by taking the language T and the trace-overlaid image $I^+ = I \oplus \boldsymbol{\tau}$. Note that we curate the Trajectory-conditioned Action Data mainly for enabling the steerability feature of MOLMOACT. Once we generate the ground truth label for each frame, we construct the action reasoning dataset by sequentailly aligning the ground-truth Depth Perception Tokens, Visual Reasoning Trace, and Action for instruction tuning. We also use the same data generation approach to obtain auxiliary robot data.

3.2 MolmoAct Dataset

We curated the MOLMOACT DATASET to improve the model's general manipulation performance and spatial reasoning in real household environments. The dataset contains 10,689 high-quality trajectories of a single-arm Franka robot performing 93 unique manipulation tasks in both home and tabletop environments as shown in Figure 12. The average length of each trajectory spans 112 timesteps. Data collection spanned two months and involved five full-time operators following strict protocols. For further details, see Appendix H. The MOLMOACT DATASET includes manipulation data from two primary settings: home environments and tabletops. To collect diverse **Home Environment Data**, we mounted a single-arm Franka robot on a lightweight, mobile platform similar to DROID [25], enabling us to transport the robot and capture scenes across living rooms, kitchens, bathrooms, and bedrooms. Each task was designed to reflect a specific household chore. In total, we collected 7,730 trajectories spanning 73 distinct tasks and 20 verbs across a wide variety of scenes. For **Tabletop data** We also collected 2,959 tabletop trajectories covering 20 atomic tasks, each performed with a diverse set of objects to promote robustness and generalization. Each task was decomposed into atomic motions and reinforced in a simplified tabletop environment.

3.3 Multimodal Web Data

Prior works have shown that co-training VLAs with the data mixture from the VLM training leads to more generalizable policies. These policies are more robust to perturbations such as lighting and background changes, and can generalize better to unseen environments and objects. We include a mixture of multimodal web data from Molmo's Supervised fine-tuning stage involving academic datasets (VQA v2.0 [30], Text VQA [31], OK-VQA [32], ChartQA [33], DocVQA [34], Infographic VQA [35], AI2D [36], A-OKVQA [37], AndroidControl [38], ScienceQA [39], TabMWP [40], ST-VQA [41], TallyQA [42], DVQA [43], FigureQA [44], and PlotQA [45]) for general visual skills and PixMo [46] for fine-grained understanding and pointing. Furthermore, we include LVIS [47] where the model is asked to predict the bounding box center of instances of a certain category to ground language to image regions.

4 Training Recipe

MOLMOACT is first pre-trained on action reasoning data curated from a subset of the OXE dataset, along with the auxiliary robot data and multimodal web data. To further enhance its capabilities, we mid-train the model on the MOLMOACT DATASET before post-training it for specific downstream tasks and embodiments. In this section, we describe the different data mixtures and training configurations used at each stage of MOLMOACT 's training (as shown in Figure 8.

4.1 Pre-training

In the first training stage, MOLMOACT is pre-trained on a mixture of action reasoning data, auxiliary robot data, and multimodal web data. For all robot data, we use a subset of OXE comprising RT-1, BridgeData V2, and BC-Z, totaling 10.5M samples, which we convert into action reasoning data using our reasoning in space formulation. We also include auxiliary robot data—auxiliary depth data (1.5M), auxiliary trace data (1.5M), and trajectory-conditioned action data (10.5M), and co-train with 2M samples of multimodal web data. During pre-training, data is sampled at the following rates: RT-1 (20%), BridgeData V2 (12.5%), BC-Z (7.5%) for both action reasoning and trajectory-conditioned data, 7.5% from the auxiliary depth and trace data, and 5% from multimodal web data as shown in Figure 2. The whole data mixture used for pre-training MOLMOACT totals up to 26.3M samples.

To pre-train MOLMOACT with the data mixture mentioned above, we use 256 H100s to train the model with 100k gradient steps using a batch size of 512, which takes around 9,728 GPU hours. At each training step, a batch of data pairs is drawn randomly from the entire data mixture by their assigned sampling rate defined above. Hyperparameter details are listed in Appendix E.

4.2 Mid-training

After pre-training, we conduct a mid-training stage on MOLMOACT DATASET, consisting of 1M action reasoning and 1M trajectory-conditioned samples tailored to household manipulation. Each sample includes two side and one wrist camera view, reformulated into paired-view training examples. Reasoning traces and depth tokens are derived from side views, while the wrist view provides complementary context. We train for 50k steps with batch size 128 on 128 H100 GPUs, around 2,304 GPU hours; further hyperparameters are detailed in Appendix E.

4.3 Post-training

In the post-training stage, we adapt MOLMOACT to new tasks and embodiments using 30–50 tele-operated demonstrations per task, converted into action reasoning and trajectory-conditioned data with action chunking ($N = 8$). Post-training is performed via LoRA fine-tuning (rank=32, alpha=16) to preserve pre-trained capabilities, using batch size 128 for simulation (LIBERO) and 64 for real-world tasks. Training steps vary by task.

Table 2: **LIBERO benchmark success rates** across four task categories (Spatial, Object, Goal, and Long-horizon) along with the average performance. MOLMOACT achieves the highest overall average success rate of 86.6%, outperforming all baselines, with strong performance across all categories, particularly in long-horizon tasks.

Baseline	Spatial	Object	Goal	Long	Avg
TraceVLA [15]	84.6%	85.2%	75.1%	54.1%	74.8%
Octo-Base [28]	78.9%	85.7%	84.6%	51.1%	75.1%
OpenVLA [5]	84.7%	88.4%	79.2%	53.7%	76.5%
SpatialVLA [29]	88.2%	89.9%	78.6%	55.5%	78.1%
CoT-VLA [48]	87.5%	91.6%	87.6%	69.0%	83.9%
NORA-AC [49]	85.6%	89.4%	80.0%	63.0%	79.5%
WorldVLA [50]	87.6%	96.2%	83.4%	60.0%	79.1%
π_0-FAST [4]	96.4%	96.8%	88.6%	60.2%	85.5%
ThinkAct [51]	88.3%	91.4%	87.1%	70.9%	84.4%
MOLMOACT-7B-D	87.0%	95.4%	87.6%	77.2%	**86.6%**

5 Experimental Evaluation

Our experimental evaluation comprises a broad suite of studies that rigorously benchmark MOLMOACT against strong baselines. We assess MOLMOACT with MOLMOACT-7B-D version in (i) its pre-training, "out-of-the-box" capabilities, (ii) its post-training adaptability across varied tasks, domains, and robotic embodiments, and (iii) its additional feature of being an interactive and steerable action reasoning model. By testing the model on a comprehensive range of scenarios both in simulation and real-world, we aim to answer the following research questions, where the last four questions are answered in Appendix C:

1. **How well does MOLMOACT perform, after pre-training, on tasks drawn from the same distribution as its training data?** We address this question by benchmarking MOLMOACT against strong VLA models on the SimplerEnv simulation benchmark [52].
2. **How effectively does MOLMOACT adapt to novel tasks, domains, and embodiments through lightweight post-training fine-tuning?** We fine-tune MOLMOACT with LoRA [53] and benchmark it against strong baselines on the LIBERO simulation suite [54]. We then validate its real-world performance on two hardware setups—a single and a bimanual Franka arm—to demonstrate adaptability across embodiments.
3. **How effectively can MOLMOACT generalize beyond its training distribution?** We investigate this through real-world out-of-distribution (OOD) tests and variant-aggregation experiments in SimplerEnv.
4. **How does mid-training on the MOLMOACT DATASET improve MOLMOACT's generalist performance?** We address this through ablation experiments in the real-world evaluations to compare MOLMOACT 's performance with and without mid-training on the MOLMOACT DATASET.
5. **How effectively does MOLMOACT follow language commands?** We benchmark MOLMOACT against strong baselines in an open-ended simulation setup where human evaluators provide free-form prompts and assess each model's resulting actions.
6. **How steerable is MOLMOACT, and how can this steerability enhance user interaction?** We perform extensive real-world ablations, guiding MOLMOACT by sketching trajectory cues on the interface and analyzing its responses, then examine the resulting human–robot interaction dynamics.

5.1 MOLMOACT After Pre-training

Evaluation Setup and Baselines. We evaluated MOLMOACT 's zero-shot capability immediately after pre-training, comparing it to generalist policies such as TraceVLA, RT-1X, OpenVLA, RoboVLM, Emma-x, π_0, π_0-FAST, Octo, Magma, HPT, SpatialVLA, and GR00T N1. Unlike models trained on massive private datasets (e.g., π_0 with 903M samples), MOLMOACT was trained on 26.3M

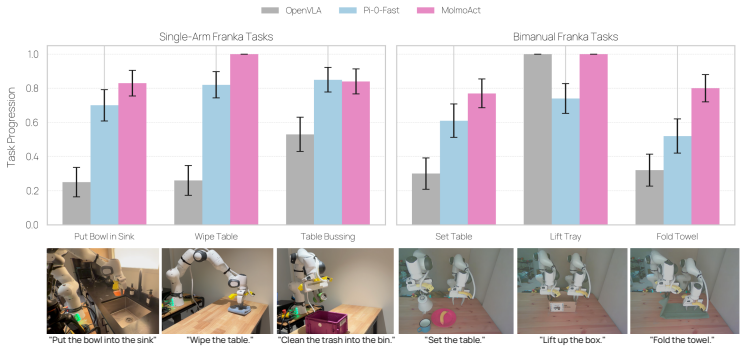


Figure 3: Real-world evaluation of OpenVLA, π_0-FAST, and MOLMOACT on single-arm (left) and bimanual (right) Franka tasks. Bar plots report average task progression with standard error across 25 trials per task. MOLMOACT consistently outperforms baselines, particularly on single-arm tasks such as *Wipe Table* and *Table Bussing*, and maintains strong performance on bimanual tasks including *Fold Towel* and *Set Table*. Bottom row shows example task setups with corresponding natural language instructions.

curated samples from subsets of OXE (BC-Z, BridgeData V2, RT-1), plus multimodal web and auxiliary robot data (subsection 4.1). Evaluation used the SimplerEnv benchmark, focusing on Google Robot visual-matching tasks to isolate in-distribution performance. We also fine-tuned MOLMOACT-7B-D-PRETRAIN on the RT-1 subset of OXE for comparison.

Evaluation Results. MOLMOACT-7B-D-PRETRAIN achieved 70.5% zero-shot success on SimplerEnv, outperforming GR00T N1, π_0, π_0-FAST, and Magma. Fine-tuning further improved performance to 71.6%, surpassing Magma by 3.2% (Table 1), showing MOLMOACT 's strength both as a zero-shot generalist and as a fine-tuning initialization.

5.2 Fast Adaptation of MOLMOACT in Post-training

We assess MOLMOACT 's post-training adaptation in both simulation and real-world settings. In simulation, we use the LIBERO benchmark [54]. We follow prior works ([5]) and evaluate on four task suites – LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long – each with 500 demonstrations across 10 tasks and filtered no-op and failed demonstrations. We set chunk size $K = 8$, and fine-tune MOLMOACT-7B-D with LoRA, comparing against state-of-the-art generalist policies (TraceVLA, OpenVLA, SpatialVLA, π_0-FAST, CoT-VLA, WorldVLA, ThinkAct, NORA-AC). In real world experiments, we evaluate on six Franka tasks—three single-arm `put_bowl_in_sink`, `wipe_table`, and `table_bussing` and three bimanual `set_table`, `lift_tray`, and `fold_towel` with 50 demonstrations each and 25 evaluation trials.

Evaluation Results. MOLMOACT-7B-D achieves the best average success rate (86.6%) on LIBERO, outperforming all baselines and surpassing ThinkAct on the challenging Long suite by 6.3%. In real-world tasks, it improves task progression over π_0-FAST by 10% (single-arm) and 22.7% (bimanual) (Figure 3).

6 Conclusion

We introduced MOLMOACT, a fully open family of action reasoning models that unify perception, planning, and control through spatial reasoning. Combining depth tokens, visual reasoning traces, and action prediction, MOLMOACT produces explainable, steerable behaviors and consistently outperforms VLA baselines, adapting efficiently to new tasks and generalizing robustly. We release model weights, code, and the MOLMOACT DATASET dataset to foster reproducibility and advance research on foundation models that transform perception into purposeful action.

7 Limitations and Potential Solutions

While MOLMOACT is all quite capable as a general-purpose action reasoning model, it is not without limitations. In the following sections, we discuss some of these limitations and potential solutions.

Camera Occlusion of End-effector. During post-training, MOLMOACT can process multiple camera views (e.g., front and wrist cameras), but its spatial reasoning primarily relies on the front camera, which typically provides a full view of the end-effector. This visibility is crucial for accurate visual reasoning trace prediction. However, if the end-effector is occluded in the front camera's view, visual trace prediction—and thus overall performance—can degrade. A potential solution is to use a wide field-of-view camera (e.g., fisheye lens) and generate visual traces via SLAM, enabling temporal rather than purely spatial reasoning.

Robustness of Steerability via Visual Traces. Robust action steerability relies on two factors: (i) precise yet diverse 2D visual traces during pre- and mid-training, and (ii) abundant, high-quality post-training data

- *Trace Quality and Diversity.* For trajectory-conditioned action data, bounding-box–based detectors (e.g., Detectron) are problematic: predicted points collapse toward box centers, reducing spatial variation. They also require task-specific fine-tuning to localize robot grippers and transfer poorly across embodiments. In contrast, VLM-based point annotations (e.g., Molmo [46], RoboPoint [55]) yield accurate, non-degenerate traces and markedly improve steerability.
- *Coverage of Action Compositions.* To achieve steerability in real-world settings, post-training data should span as many action compositions as possible. Practically, this means inducing the robot to explore motion variants while still completing tasks, so the model learns rich correspondences between image-space traces and resulting actions.

MOLMOACT only learns to directly predict action simply based on the trace-overlaid image. So when we steer actions with a 2D visual trace, we are not leveraging the capability of MOLMOACT to perform action reasoning in space. Thus, we observe that this form of action steering still cannot enable the model to follow more complicated tasks. In particular, because the cue is purely 2D, the model lacks an explicit notion of depth: it often follows the intended path within the image plane (in-plane motion) but exhibits unintended or imprecise translation along the camera's depth axis (out-of-plane). We hypothesize this could be mitigated by conditioning on—or reusing—the model's predicted depth-perception tokens to lift the trace into 3D, which we leave for future exploration. Despite these limitations, our scheme demonstrates the feasibility of action steerability based on pure visual cues, and offers a simple, practical insight that the robotics community can build upon.

Speed of Action Reasoning Model prediction. Similar to many existing VLAs, our model exhibits a mismatch between its control inference frequency and the control frequency used during data collection. This gap may stem from server-to-robot communication latency and the additional time required to predict a larger number of reasoning tokens. Future work could explore techniques to reduce inference time, as seen in VLM optimization, or develop smaller parameter models optimized for efficient execution on edge or local devices.

Precision in Depth Perception Token. For depth perception token prediction, we follow [56] and use a fixed set of 100 tokens to represent depth. However, fine-grained manipulation tasks require higher-resolution depth estimation. Increasing the number of depth perception tokens could enhance spatial reasoning and improve performance on such tasks.

Acknowledgments

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

[1] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2): 230–244, 2022.

[2] Z. Xu, K. Wu, J. Wen, J. Li, N. Liu, Z. Che, and J. Tang. A survey on robotics with foundation models: toward embodied ai. *arXiv preprint arXiv:2402.02385*, 2024.

[3] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, et al. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research*, 44(5):701–739, 2025.

[4] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*.

[5] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[6] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[7] NVIDIA, :, J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. J. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, J. Jang, Z. Jiang, J. Kautz, K. Kundalia, L. Lao, Z. Li, Z. Lin, K. Lin, G. Liu, E. Llontop, L. Magne, A. Mandlekar, A. Narayan, S. Nasiriany, S. Reed, Y. L. Tan, G. Wang, Z. Wang, J. Wang, Q. Wang, J. Xiang, Y. Xie, Y. Xu, Z. Xu, S. Ye, Z. Yu, A. Zhang, H. Zhang, Y. Zhao, R. Zheng, and Y. Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

[8] J. Yang, R. Tan, Q. Wu, R. Zheng, B. Peng, Y. Liang, Y. Gu, M. Cai, S. Ye, J. Jang, et al. Magma: A foundation model for multimodal ai agents. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 14203–14214, 2025.

[9] H. Liu, X. Li, P. Li, M. Liu, D. Wang, J. Liu, B. Kang, X. Ma, T. Kong, and H. Zhang. Towards generalist robot policies: What matters in building vision-language-action models. 2025.

[10] W. Pumacay, I. Singh, J. Duan, R. Krishna, J. Thomason, and D. Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[12] E. Zelikman, Y. Wu, J. Mu, and N. Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[13] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

[14] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[15] R. Zheng, Y. Liang, S. Huang, J. Gao, H. Daumé III, A. Kolobov, F. Huang, and J. Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*, 2024.

[16] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[17] M. Tschannen, A. Gritsenko, X. Wang, M. F. Naeem, I. Alabdulmohsin, N. Parthasarathy, T. Evans, L. Beyer, Y. Xia, B. Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025.

[18] T. OLMo, P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[19] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[20] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[21] H. Lai and M. Nissim. mcot: Multilingual instruction tuning for reasoning consistency in language models. *arXiv preprint arXiv:2406.02301*, 2024.

[22] Q. Sun, P. Hong, T. D. Pala, V. Toh, U. Tan, D. Ghosal, S. Poria, et al. Emma-x: An embodied multimodal action model with grounded chain of thought and look-ahead spatial reasoning. *arXiv preprint arXiv:2412.11974*, 2024.

[23] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024.

[24] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

[25] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[26] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[27] L. Wang, X. Chen, J. Zhao, and K. He. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. *Advances in neural information processing systems*, 37: 124420–124450, 2024.

[28] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[29] D. Qu, H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, B. Zhao, D. Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

[30] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *CVPR*, 2017.

[31] A. Singh, V. Natarjan, M. Shah, Y. Jiang, X. Chen, D. Parikh, and M. Rohrbach. Towards VQA models that can read. In *CVPR*, 2019.

[32] K. Marino, M. Rastegari, A. Farhadi, and R. Mottaghi. OK-VQA: A visual question answering benchmark requiring external knowledge. In *CVPR*, 2019.

[33] A. Masry, D. Long, J. Q. Tan, S. Joty, and E. Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *ACL*, 2022.

[34] M. Mathew, D. Karatzas, and C. Jawahar. DocVQA: A dataset for VQA on document images. In *WACV*, 2021.

[35] M. Mathew, V. Bagal, R. Tito, D. Karatzas, E. Valveny, and C. Jawahar. InfographicVQA. In *WACV*, 2022.

[36] A. Kembhavi, M. Salvato, E. Kolve, M. Seo, H. Hajishirzi, and A. Farhadi. A diagram is worth a dozen images. In *ECCV*, 2016.

[37] D. Schwenk, A. Khandelwal, C. Clark, K. Marino, and R. Mottaghi. A-OKVQA: A benchmark for visual question answering using world knowledge. In *ECCV*, 2022.

[38] W. Li, W. Bishop, A. Li, C. Rawles, F. Campbell-Ajala, D. Tyamagundlu, and O. Riva. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*, 2024.

[39] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *NeurIPS*, 2022.

[40] P. Lu, L. Qiu, K.-W. Chang, Y. N. Wu, S.-C. Zhu, T. Rajpurohit, P. Clark, and A. Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *ICLR*, 2023.

[41] A. F. Biten, R. Tito, A. Mafla, L. Gomez, M. Rusinol, E. Valveny, C. Jawahar, and D. Karatzas. Scene text visual question answering. In *ICCV*, 2019.

[42] M. Acharya, K. Kafle, and C. Kanan. TallyQA: Answering complex counting questions. In *AAAI*, 2019.

[43] K. Kafle, B. Price, S. Cohen, and C. Kanan. DVQA: Understanding data visualizations via question answering. In *CVPR*, 2018.

[44] S. E. Kahou, V. Michalski, A. Atkinson, Á. Kádár, A. Trischler, and Y. Bengio. FigureQA: An annotated figure dataset for visual reasoning. *arXiv preprint arXiv:1710.07300*, 2017.

[45] N. Methani, P. Ganguly, M. M. Khapra, and P. Kumar. PlotQA: Reasoning over scientific plots. In *WACV*, 2020.

[46] M. Deitke, C. Clark, S. Lee, R. Tripathi, Y. Yang, J. S. Park, M. Salehi, N. Muennighoff, K. Lo, L. Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv e-prints*, pages arXiv–2409, 2024.

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

[48] Q. Zhao, Y. Lu, M. J. Kim, Z. Fu, Z. Zhang, Y. Wu, Z. Li, Q. Ma, S. Han, C. Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1702–1713, 2025.

[49] C.-Y. Hung, Q. Sun, P. Hong, A. Zadeh, C. Li, U. Tan, N. Majumder, S. Poria, et al. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025.

[50] J. Cen, C. Yu, H. Yuan, Y. Jiang, S. Huang, J. Guo, X. Li, Y. Song, H. Luo, F. Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

[51] C.-P. Huang, Y.-H. Wu, M.-H. Chen, Y.-C. F. Wang, and F.-E. Yang. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *arXiv preprint arXiv:2507.16815*, 2025.

[52] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[53] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[54] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36: 44776–44791, 2023.

[55] W. Yuan, J. Duan, V. Blukis, W. Pumacay, R. Krishna, A. Murali, A. Mousavian, and D. Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. *arXiv preprint arXiv:2406.10721*, 2024.

[56] M. Bigverdi, Z. Luo, C.-Y. Hsieh, E. Shen, D. Chen, L. G. Shapiro, and R. Krishna. Perception tokens enhance visual reasoning in multimodal language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3836–3845, 2025.

[57] L. Berscheid, P. Meißner, and T. Kröger. Robot learning of shifting objects for grasping in cluttered environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 612–618. IEEE, 2019.

[58] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*, 2019.

[59] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

[60] H.-S. Fang, H. Fang, Z. Tang, J. Liu, C. Wang, J. Wang, H. Zhu, and C. Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.

[61] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[62] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

[63] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[64] N. M. M. Shafiullah, A. Rai, H. Etukuru, Y. Liu, I. Misra, S. Chintala, and L. Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[65] A. Xie, L. Lee, T. Xiao, and C. Finn. Decomposing the generalization gap in imitation learning for visual robotic manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3153–3160. IEEE, 2024.

[66] F. Lin, Y. Hu, P. Sheng, C. Wen, J. You, and Y. Gao. Data scaling laws in imitation learning for robotic manipulation. *arXiv preprint arXiv:2410.18647*, 2024.

[67] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[68] D. Groeneveld, I. Beltagy, P. Walsh, A. Bhagia, R. Kinney, O. Tafjord, A. H. Jha, H. Ivison, I. Magnusson, Y. Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[69] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[70] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[71] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[72] Q. Li, Y. Liang, Z. Wang, L. Luo, X. Chen, M. Liao, F. Wei, Y. Deng, S. Xu, Y. Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[73] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[74] Y. Li, Y. Deng, J. Zhang, J. Jang, M. Memmel, R. Yu, C. R. Garrett, F. Ramos, D. Fox, A. Li, et al. Hamster: Hierarchical action models for open-world robot manipulation. *arXiv preprint arXiv:2502.05485*, 2025.

[75] Y. Shentu, P. Wu, A. Rajeswaran, and P. Abbeel. From llms to actions: Latent codes as bridges in hierarchical robot control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8539–8546. IEEE, 2024.

[76] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[77] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[78] H. Bharadhwaj, J. Vakil, M. Sharma, A. Gupta, S. Tulsiani, and V. Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795. IEEE, 2024.

[79] H. Fang, M. Grotz, W. Pumacay, Y. R. Wang, D. Fox, R. Krishna, and J. Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

[80] P. Liu, Y. Orru, J. Vakil, C. Paxton, N. M. M. Shafiullah, and L. Pinto. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.

[81] L. X. Shi, Z. Hu, T. Z. Zhao, A. Sharma, K. Pertsch, J. Luo, S. Levine, and C. Finn. Yell at your robot: Improving on-the-fly from language corrections. *arXiv preprint arXiv:2403.12910*, 2024.

[82] Y. Wang, L. Wang, Y. Du, B. Sundaralingam, X. Yang, Y.-W. Chao, C. Perez-D'Arpino, D. Fox, and J. Shah. Inference-time policy steering through human interactions. *arXiv preprint arXiv:2411.16627*, 2024.

[83] J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, P. Sundaresan, P. Xu, H. Su, K. Hausman, C. Finn, Q. Vuong, and T. Xiao. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches, 2023.

[84] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, and A. Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

[85] J. Clark, S. Mirchandani, D. Sadigh, and S. Belkhale. Action-free reasoning for policy generalization. *arXiv preprint arXiv:2502.03729*, 2025.

[86] J. Yang, C. K. Fu, D. Shah, D. Sadigh, F. Xia, and T. Zhang. Bridging perception and action: Spatially-grounded mid-level representations for robot generalization. *arXiv preprint arXiv:2506.06196*, 2025.

[87] H. Xu, S. Xie, X. Tan, P.-Y. Huang, R. Howes, V. Sharma, S.-W. Li, G. Ghosh, L. Zettlemoyer, and C. Feichtenhofer. Demystifying CLIP data. In *ICLR*, 2024.

[88] M. Guerquin. Introducing AI2's beaker. *AI2 Blog*, 2022. URL https://web.archive.org/web/20241231204439/https://medium.com/ai2-blog/beaker-ed617d5f4593. Accessed: 2024-12-31. Original: https://medium.com/ai2-blog/beaker-ed617d5f4593.

[89] W. Wang, M. Ghobadi, K. Shakeri, Y. Zhang, and N. Hasani. Rail-only: A low-cost high-performance network for training llms with trillion parameters. *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1–10, 2023. URL https://api.semanticscholar.org/CorpusID:260125277.

Appendix

The appendix includes the following sections:

- §A - Related Work
- §B - Action Steerability via Visual Reasoning Trace
- §C - Experiment Evaluation
- §D - Model Details
- §E - Training Details
- §F - Action Vocabulary
- §G - Evaluation Details
- §H - Data Details
- §I - Dataset Examples

A Related Work

A.1 Generalist robot manipulation policies

Recent advances in robotic manipulation have shifted from training narrow, single-task specialists to learning from large, diverse datasets spanning many scenes, tasks, and embodiments[57, 14, 58, 59, 60, 61, 25, 62, 63, 64]. This shift has enabled policies that not only excel within their training distribution but also generalize to out-of-distribution scenes, environments, language instructions, and novel objects [10, 65, 66]. Much of this progress has been fueled by Large Language Models (LLMs) [26, 67, 68, 69] and Vision-Language Models (VLMs) [70, 71, 46], giving rise to the paradigm of Vision-Language-Action models (VLAs) [4, 14, 15, 48, 6, 72, 29, 5]. While some adopt hierarchical designs, where a robotics-focused VLM [73, 74, 75] outputs intermediate representations for pre-trained, task-specific policies to improve generalization. However, a major bottleneck for these models is their heavy reliance on large amounts of robotics-specific data, often collected via tele-operation. In contrast, MOLMOACT aims to leverage reasoning in space to train an action reasoning model that achieves competitive or superior performance with only a fraction of the data required by existing methods.

A.2 Robot reasoning and planning with language

In recent years, numerous works have demonstrated that augmenting end-to-end robotic policies with high-level reasoning—either by integrating LLMs or VLMs directly into robotic systems, or by incorporating their reasoning outputs into policies—can substantially improve performance on long-horizon tasks and enhance generalization [76, 77, 78, 79, 80, 81, 82, 83].

MOLMOACT supports steering through both natural language and interactive visual trace sketches, enhancing explainability and diagnosis. Prior methods—RT-Trajectory [83], HAMSTER [74], and inference-time policy steering [82]—are more limited: RT-Trajectory and inference-time approaches are tied to specific architectures, while HAMSTER produces only 2D trajectories with low-level execution constrained to fixed tasks. In contrast, MOLMOACT generalizes steering to novel spatial layouts, unseen objects, and ambiguous language, providing a more versatile and semantically grounded control interface.

A.3 Embodied reasoning for robotic manipulation

Chain-of-thought (CoT) prompting [11] has boosted multi-step reasoning in LLMs and has since been extended to multimodal settings [56, 84]. Inspired by this, robotics research has begun integrating reasoning into vision-language-action (VLA) models. ECoT [23] generates subgoals via prompting; CoT-VLA [48] uses visual subgoal frames; RAD [85] curates reasoning from human video; and

ThinkAct [51] links reinforcement learning with latent planning. Emma-X [22] fine-tunes OpenVLA with subtasks and predicted gripper states, while [86] explores trajectory traces with depth but only on small diffusion-policy datasets. Unlike ECoT, CoT-VLA, RAD, and ThinkAct—which rely on latent embeddings, sub-goals, or text that are hard to ground—MOLMOACT grounds each reasoning step directly in the scene. Unlike Emma-X, which focuses on gripper positions without full 3D context, MOLMOACT reasons in space, with steps visualizable in both 2D and 3D. This explicit spatial grounding improves explainability and action prediction within a CoT framework.

B Action Steerability via Visual Reasoning Trace

We define *steerability* as the ability to guide a policy at test time to perform different behaviors using user-provided instructions. Most prior VLA systems rely exclusively on language for steering. However, language-only steering faces three practical challenges: (i) it requires large and diverse corpora of high-quality language–action pairs to learn a reliable grounding between words and control, (ii) natural language is inherently ambiguous about magnitudes, scales, and endpoints, and (iii) post-trained models often exhibit narrow prompting habits, making them brittle to out-of-distribution phrasing. For manipulation, these issues translate into imprecise or inconsistent control. We therefore seek a steering modality that is both precise and scalable. Rather than relying on ambiguous language prompts, we allow the user to draw a visual reasoning trace $\boldsymbol{\tau}$ directly on the camera image to indicate the desired end-effector path. A trace $\boldsymbol{\tau} = (p_1, \ldots, p_L)$, $1 \leq L \leq 5$ (i.e., 0 to 4 line segments), is overlaid onto the RGB image I to form an augmented observation $I^+ = I \oplus \boldsymbol{\tau}$. visual reasoning traces are unambiguous, easily edited, and generalize across tasks without large text–action corpora or brittle language patterns.

At test time, given I, instruction T, and a user-drawn trace $\boldsymbol{\tau}$, we construct $I^+ = I \oplus \boldsymbol{\tau}$ and generate the next-step action tokens $\mathbf{a} = (a_1, \ldots, a_D)$ autoregressively:

$$p(\mathbf{a} \mid I^+, T) = \prod_{k=1}^{D} p\big(a_k \mid I^+, T, \mathbf{a}_{<k}\big). \tag{2}$$

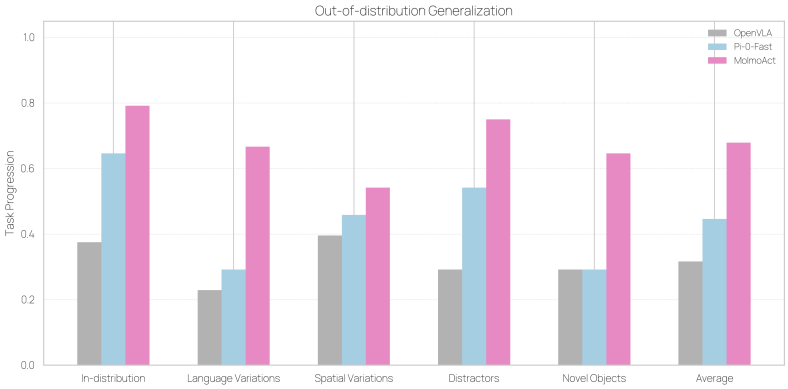
By conditioning directly on the overlaid trace, the model executes closed-loop control that follows the user's sketch. Repeating this at each timestep yields precise, interactive steering that is both scalable and robust to out-of-distribution phrasing.

C Experimental Evaluation

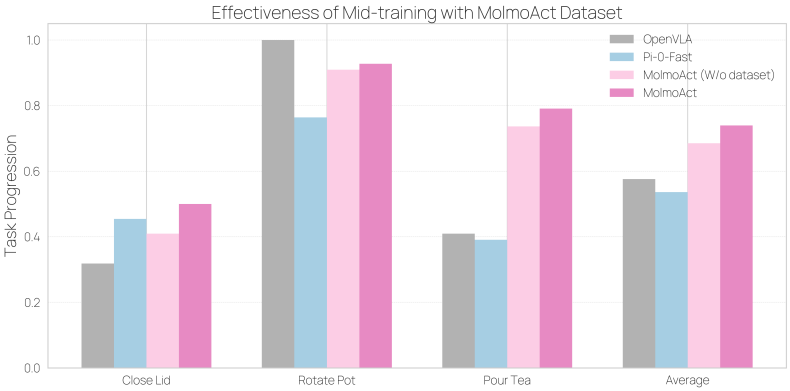
C.1 Effectiveness of MOLMOACT in Out-of-Distribution Generalization

We evaluate MOLMOACT in both simulation and real-world settings to assess its ability to generalize beyond the training data distribution, both in zero-shot and fine-tuned regimes. In simulation, we follow the SimplerEnv variant-aggregation protocol, which introduces distribution shifts through changes in lighting, textures, and camera viewpoints. We compare MOLMOACT-7B-D-PRETRAIN and its RT-1 fine-tuned variant against several state-of-the-art generalist policies—TraceVLA, RT-1X, OpenVLA, RoboVLM, Emma-X, π_0-FAST, and SpatialVLA. For real-world evaluation, we test MOLMOACT-7B-D using a single Franka arm on a multi-task setup involving three objects and two different-colored plates arranged on a tabletop. We collect over 300 tele-operated demonstrations spanning three task types, then post-train MOLMOACT-7B-D and baselines in a multi-task setting. During evaluation, we test generalization in four aspects: (1) **Language Variation** — rephrased instructions, (2) **Spatial Variation** — changes in target object position, (3) **Distractors** — addition of irrelevant objects, and (4) **Novel Objects** — substitution of target objects with unseen ones. We benchmark MOLMOACT-7B-D against π_0-FAST and OpenVLA, testing three variants per task and four trials per variant. Full task details are presented in subsection G.4.

Evaluation Results. In simulation, fine-tuned MOLMOACT-7B-D-PRETRAIN achieves 72.1% on the variant aggregation tasks as shown in Table 1, outperforming all baselines and exceeding the second-best model, RT-2-X, by 7.8%. The performance difference between variant aggregation and visual



(a) MOLMOACT generalizes beyond training distributions.



(b) MOLMOACT DATASET improves task performance.

Figure 4: **MOLMOACT outperforms baselines across generalization and mid-training settings.** (a) Out-of-distribution generalization: Task progression scores for OpenVLA, π_0-FAST, and MOLMOACT across in-distribution, language variation, spatial variation, distractors, and novel object conditions, showing consistent gains for MOLMOACT. (b) Effectiveness of mid-training with the MOLMOACT DATASET: Comparison of task progression on real-world tasks (Close Lid, Rotate Pot, Pour Tea) for MOLMOACT with and without the dataset, π_0-FAST, and OpenVLA, demonstrating that mid-training with the dataset improves performance across tasks.

matching is less than 1%, highlighting MOLMOACT's robustness to visual and distributional shifts. In the real world, MOLMOACT-7B-D consistently surpasses all baselines across all generalization axes, achieving a 23.3% average improvement in task progression over π_0-FAST as shown in Figure 4a.

C.2 Effect of the MOLMOACT DATASET on MOLMOACT Performance

Evaluation Setups and Baselines. To assess the effectiveness of mid-training with the MOLMOACT DATASET, we conducted real-world experiments on three curated tasks that go beyond simple pick-and-place: `close_lid`, `rotate_pot`, and `pour_tea`. For each task, we collected 50 demonstrations and trained four models: MOLMOACT-7B-D, MOLMOACT-7B-D without mid-training, π_0-FAST, and OpenVLA. Each model was then evaluated over 10 trials per task. Task details are shown in subsection G.5.

Evaluation Results. Based on the real-world ablation studies shown in Figure 4b, MOLMOACT-7B-D outperforms its counterpart without mid-training by an average margin of 5.5% across the three tasks, demonstrating that mid-training on the MOLMOACT DATASET yields a consistent performance boost of around 5%. Even without mid-training, MOLMOACT-7B-D-PRETRAIN surpasses π_0-FAST and OpenVLA by 14.8% and 10.9%, respectively.

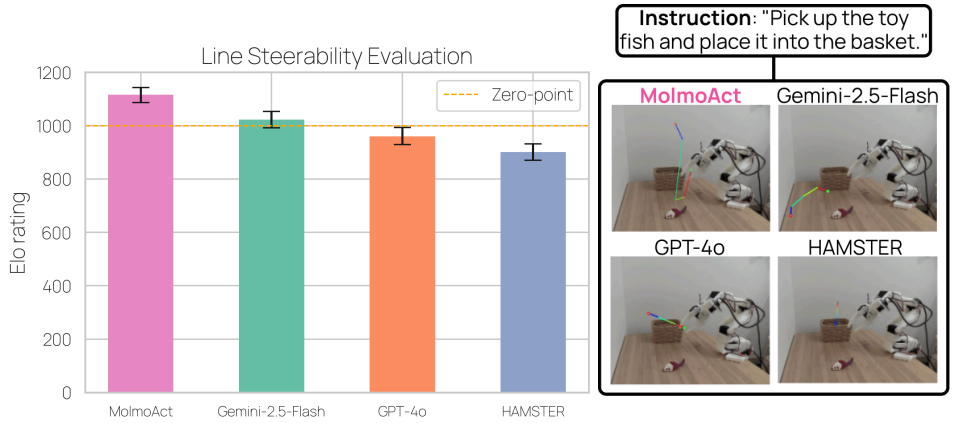


Figure 5: **Line steerability evaluation across models. Left:** Elo ratings show that MOLMOACT achieves the highest performance, surpassing Gemini-2.5-Flash, GPT-4o, and HAMSTER, with error bars indicating 95% confidence interval (CI). **Right:** Example qualitative results showing predicted visual traces overlaid on robot camera views.

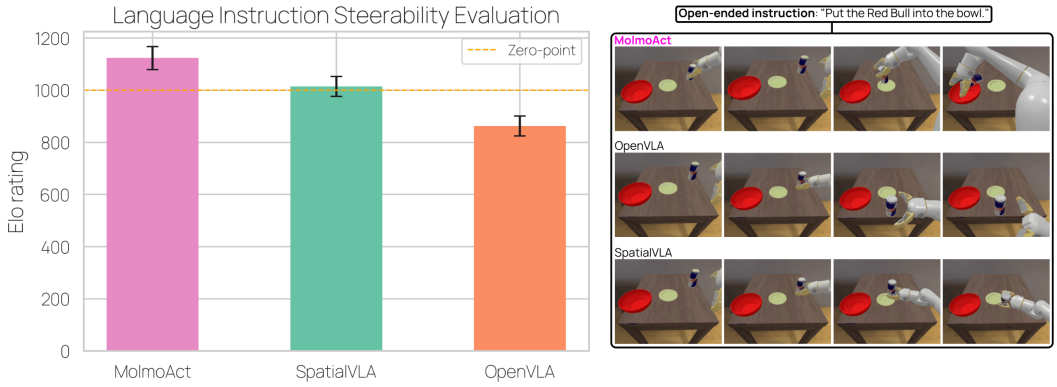


Figure 6: **Language Instruction Evaluation. Left:** Elo ratings for three models based on human votes in a head-to-head instruction-following evaluation. **Right:** Qualitative comparison of execution traces for the open-ended instruction "Put the redbull into the bowl." MOLMOACT aligns more closely with the intended instruction than other models.

C.3 Instruction Following of MOLMOACT

We evaluated MOLMOACT's ability to follow natural language instructions in two settings: (i) executing tasks with open-ended commands in simulation, and (ii) generating visual traces conditioned on language prompts. For the first one, we curated five manipulation scenarios in the SimplerEnv environment using a Google Robot, each involving novel out-of-distribution objects. Ten participants provided 29 open-ended instructions (e.g., *"Put the redbull into the bowl."*). We compared MOLMOACT-7B-D-PRETRAIN to **SpatialVLA** and **OpenVLA**, both pre-trained on the OXE dataset. For each instruction, the models generated rollouts, which were evaluated in a head-to-head arena-style web interface. Human annotators (n=100) selected which rollout best matched the instruction. We collected over 1,500 votes, which were converted into Elo ratings (see Figure 6). For visual trace generation, 10 participants wrote 87 language prompts for 30 internet-sourced images depicting tabletop and mobile manipulation scenarios. MOLMOACT-7B-D-PRETRAIN was evaluated against **Gemini-2.5-Flash**, **GPT-4o**, and **HAMSTER**—a VLM fine-tuned for trace generation. Participants voted in a similar blind arena interface, resulting in over 1,000 votes. Details with our curated manipulation scenes and instructions provided by participants are shown in subsection G.6.

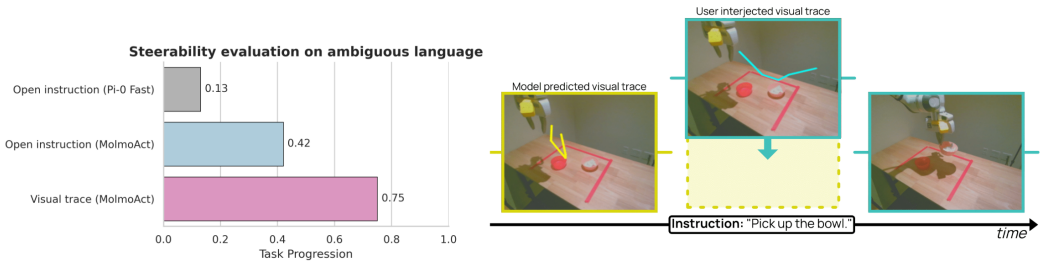


Figure 7: **Steerability evaluation with open instructions and visual traces. Left:** Success rates for different steering modes, showing that MOLMOACT with visual trace steering achieves the highest success rate (0.75), outperforming its open-instruction variant and π_0-FAST. **Right:** Example of the *"Pick up the bowl"* task: the model-predicted trajectory (yellow) is adjusted via a user-provided steering trajectory (cyan), resulting in the corrected task completion.

Evaluation Results. MOLMOACT-7B-D-PRETRAIN achieved the highest Elo rating in the simulation instruction-following task, outperforming SpatialVLA by 109 points and OpenVLA by an even larger margin. Pairwise win rates also show that MOLMOACT-7B-D-PRETRAIN winning over SpatialVLA in 58% of comparisons and over OpenVLA in 81%. A sample rollout comparison for the instruction *"Put the redbull into the bowl."* is shown on the right in Figure 6. In the visual trace task, MOLMOACT-7B-D-PRETRAIN again outperformed all baselines, achieving significantly higher Elo scores with non-overlapping 95% confidence intervals, demonstrating strong language-grounded generalization in both action execution and trace generation as shown in Figure 5.

C.4 Steerability of MOLMOACT

Evaluation Setups and Baselines. We aim to evaluate MOLMOACT's ability to steer robot actions, particularly when initial language instructions are ambiguous. Specifically, we investigate the effectiveness of different interaction mediums in guiding MOLMOACT toward user-intended targets during task execution. For this purpose, we set up a `pick_up_bowl` task, post-training MOLMOACT-7B-D and the baseline model (π_0-FAST) with 100 collected demonstrations, each annotated with two distinct language instructions: one specifying the clean bowl and the other the dirty bowl, as depicted in Figure 7. During evaluation, we first provide ambiguous instructions such as "pick up (the) bowl," prompting MOLMOACT-7B-D to predict an initial trajectory towards one of the bowls. Subsequently, we test two steering methods: visual trace sketches to visually instruct the model toward the alternative bowl, and open-ended natural language instructions provided interactively by participants (N=10) that are different from the ground-truth instruction. For comparison, we also attempt to steer the actions of π_0-FAST by changing language instructions at test-time. Each model is evaluated in 15 trials, and the performance is evaluated according to the progression of the task. For more details about the setting, please refer to subsection G.7.

Evaluation Results. Based on our experiments, we observed that MOLMOACT-7B-D is notably more steerable via visual trace inputs, achieving a success rate of 75%. Additionally, steering using visual traces significantly outperforms steering via open-ended natural language instructions by a margin of 33%. Lastly, we demonstrate that MOLMOACT-7B-D exhibits superior instruction-following capabilities compared to the baseline model, π_0-FAST. Specifically, when steering robot actions using open-ended language instructions, MOLMOACT surpasses π_0-FAST by a substantial margin of 29%, highlighting its enhanced instruction-following capabilities to user commands.

D Model Details

This section summarizes the MOLMOACT model architecture, which inherits Molmo with slight modification. The design combines a pre-processor for multi-scale cropping and optionally image padding, a ViT image encoder, a vision–language connector, and a LLM.

D.1 Backbone Overview

MOLMOACT has the following parts:

1. **Pre-processor:** converts each input image into one low-resolution crop and several high-resolution crops.
2. **ViT Image Encoder:** encodes each crop independently into per-patch features.
3. **Vision–language Connector:** pools and projects patch features into the LLM embedding space.
4. **LLM:** autoregressively processes vision and text tokens.

From this template MOLMOACT instantiates a family of models by selecting a vision encoder and an LLM while keeping the training recipe mainly consistent. Vision encoders include OpenAI ViT-L/14 336px CLIP and ViT-SO400M/14 384px SigLIP2. LLM backbones include fully open OLMo-2-1124-7B and open-weight Qwen2.5-7B. With the combination of ViT-SO400M/14 384px SigLIP2 with Qwen2.5-7B, we have MOLMOACT-7B-D, our best and demo model. With the combination of OpenAI ViT-L/14 336px CLIP with OLMo-2-1124-7B, we have MOLMOACT-7B-O, our most open model. Note that although OpenAI ViT-L/14 336px CLIP uses closed data, it can be reproduced from scratch, as shown by MetaCLIP [87].

D.2 Image Encoding and Cropping

Most ViTs accept square images at a fixed resolution, which is insufficient for fine-grained details. This mainly applies to the multimodal web data, as much of the robot data is not high-resolution, and there is also work [5] showing that image resolution doesn't affect much for robot control. To make MOLMOACT more general, it still inherits the way Molmo addresses the high-resolution problem by tiling each image with multiple square high-resolution crops plus a resized low-resolution full image. Cropping proceeds as follows.

Grid Selection and Overlap. A rectangular grid (e.g., 2×2, 3×1) is chosen so each grid cell matches the ViT input size. The grid squares are then moved closer together to introduce a fixed overlap margin (default 4 patches, approximately 56 pixels), which supplies border patches with neighbor context. Features from overlapping pixels are *not* forwarded to the connector or LLM, so the resulting tokens exactly tile the high-resolution image. Although overlap slightly reduces the effective tiled resolution, this can be offset by using more crops, and empirically improves performance.

Resizing and Padding. For OpenAI CLIP vison encoder, we follow the way Molmo [46] does to resize and pad the image to keep its original aspect ratio before processing. The scheme is the following. The image is upscaled to fit the grid while preserving aspect ratio, choosing the scale that minimizes upscaling; ties are broken by minimizing the overall size. A maximum number of high-resolution crops is enforced. If covering the image would exceed this limit, the image is downscaled to fit. In all cases, the image is padded with black borders so each crop is square and aligned to the grid. The low-resolution crop is produced by resizing and padding the full image to the ViT's native resolution. Each crop is encoded independently by the ViT and connector to produce patch features. A learned embedding indicating the crop's padding status (no padding, some padding, or all padding) is added to the patch features so the model can distinguish natural black regions from artificial padding.

Note that for SigLIP2 vision encoder, we use the standard way to resize all image inputs to a square image without padding, which follows the original transform in SigLIP2 training.

D.3 Vision–language Connector

After ViT encoding, Molmo aggregates features in two steps:

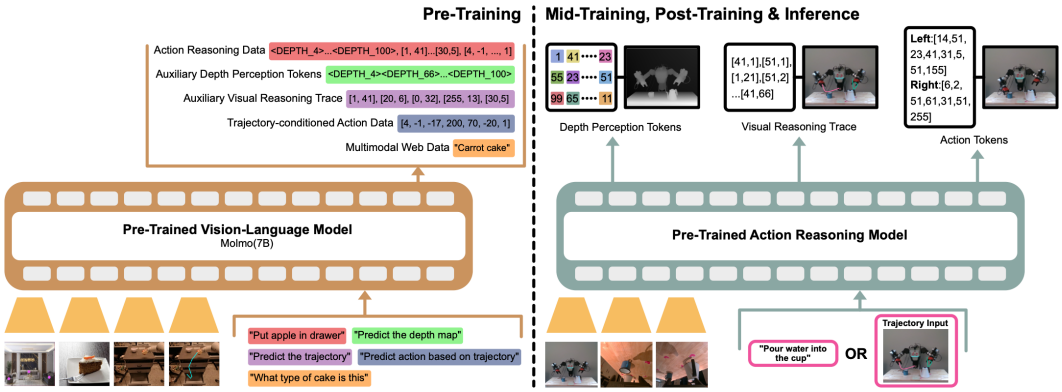


Figure 8: **Training process of MOLMOACT.** The model training process consists of two stages: **Pre-training** (left) and **Post-training, Mid-training & Inference** (right). During pre-training, the vision–language backbone (Molmo) is trained on multimodal and robot reasoning data for diverse objectives, including discretized robot control, 2D pointing, trajectory drawing, open-vocabulary question answering, and perception token prediction. In post-training, the action reasoning model consumes multi-view camera images and either natural language instructions or visual trajectory inputs, generating perception tokens, visual reasoning trace tokens, and action tokens for execution.

1. **Layer selection and concatenation:** features from the third-to-last (OpenAI CLIP) or fourth-to-last (SigLIP2) and the tenth-from-last ViT layers are concatenated for each patch; this slightly outperforms using a single layer as shown by Molmo [46].

2. **Attention pooling in 2×2 windows:** within each 2×2 patch window, a multi-headed attention layer pools the four patches to a single vector, using the mean of the patches as the query. This pooling reduces sequence length while preserving local spatial structure and outperforms naive concatenation as shown by Molmo [46].

Pooled features are then mapped to the LLM embedding space with a small MLP.

D.4 Arranging Vision Tokens

Pooled patch features (vision tokens) are serialized left-to-right and top-to-bottom. Tokens from the low-resolution full image appear first, followed by high-resolution crop tokens arranged in row-major order. Special tokens mark the start and end of both the low- and high-resolution sequences. Row-end tokens are inserted between rows to indicate row transitions.

D.5 Multi-image Inputs

Molmo [46] itself doesn't provide the capability to take in multi-image inputs. We implement this in a straightforward way: we process all the images to vision tokens in the same way as mentioned above, then we append index tokens at the beginning of the vision tokens of each image, and we finally concatenate all images together as the input. The index tokens are just text tokens of `"Image i"`, where `i` stands for the ith image.

D.6 Full Hyperparameters

The full hyperparameters of MOLMOACT architecture are shown in Table 3. Note that for LoRA implementation, adapters are applied to all linear layers in the model.

E Training Details

E.1 Implementation

Our training implementation mainly follows Molmo. We train in PyTorch using Fully Sharded Data Parallel (FSDP), and use PyTorch's Scaled Dot-Product Attention (SDPA) attention implementation. For numeric precision, we enable Automatic Mixed Precision (AMP) with bfloat16 for most operations. As an exception, we compute layer normalization and Rotary Position Embeddings (RoPE) in fp32.

With FSDP, each GPU forms a local mini-batch, computes gradients, and then we average the gradients across devices. When normalizing the loss on each device, we divide the device's total loss by the global average number of loss-tokens per example across all devices, rather than by the device-local count. This avoids a subtle bias that can arise when examples with up-weighting also contain fewer loss-tokens (e.g., shorter responses) and happen to co-occur on devices with smaller token counts. Using the global average corrects this mismatch and is especially important when the global batch is much larger than any single device batch.

For parameter-efficient fine-tuning, we do not shard LoRA adapter parameters under FSDP. Instead, each GPU keeps a full copy of the LoRA parameters, and we register a gradient hook on those tensors to synchronize their gradients across ranks before the optimizer step. Because LoRA adds only a small fraction of the total parameters, this replication has negligible memory and communication overhead while simplifying the training setup and avoiding sharding edge cases for the adapters.

Batches mix examples from multiple tasks. We cap the sequence length at 2304 tokens for both pre-training and fine-tuning, truncating only when necessary (e.g., heavily annotated synthetic data or rare outliers). Training is stable under this recipe—without loss spikes or NaNs—which we attribute in part to initializing from pre-trained models.

To enable the model to learn to understand and output depth perception tokens, we follow the training scheme of LLaVA-AURORA [56] by unfreezing the tokenizer embedding and lm head. For MOLMOACT-7B-D, which uses Qwen2.5-7B, we simply replace the first 130 padding tokens with the depth perception tokens $\{\langle \text{DEPTH_START} \rangle, \langle \text{DEPTH_END} \rangle\} \cup \{\langle \text{DEPTH_}k \rangle\}_{k=1}^{128}$. However, for MOLMOACT-7B-O, which uses Olmo2-7B, since it has less than 130 padding tokens, we first pad the tokenizer and lm head to its next multiple of 512, then replace the first 130 tokens with our depth perception tokens in the same way as MOLMOACT-7B-D.

All of our collected data used for the mid- and post-training stages is recorded at 640×480 px, which triggers the high-resolution cropping procedure described in Appendix D. By contrast, the OXE robot data used for pre-training has lower resolution, so no high-res crop is applied. To match OXE during pre-training, we downscale our collected images from 640×480 to 320×240 px while preserving the original aspect ratio. This alignment also reduces the number of vision tokens and accelerates training.

Full training hyperparameters and information are shown in Table 4. Note that for post-training, we train the model until it fully converges, which is determined by its training loss and evaluation performance. Therefore, training steps largely vary across different tasks and scenarios. We will show the training details for post-training in different tasks in later sections.

E.2 GPU Cluster

MOLMOACT was trained on Jupiter, an Ai2 GPU cluster in Austin, Texas. MOLMOACT workloads were scheduled using Beaker [88], a custom workload management system. Jupiter comprises 128 GPU nodes and is operated by Cirrascale Cloud Services[1].

[1] cirrascale.com

Compute Jupiter provides 1,024 NVIDIA H100 GPUs (80GB HBM3, 700W) across 128 servers. Each server has 2,×,Intel Xeon Platinum8468 CPUs, 2TB DDR5 system memory, and 18 TB local NVMe storage.

Storage The servers are connected over an 800Gbps local network to a WEKA high-performance storage cluster[2]. The storage system provides 1PB of NVMe SSD across 11 storage servers and 5PB of HDD across 12 hosts. Each Jupiter server has two bonded 25Gbps Mellanox Ethernet NICs (50Gbps per host). In benchmarks, we achieved 761Gbps aggregate read/write throughput using 64 client machines.

Interconnect Cross-node GPU communication uses RDMA over InfiniBand on a two-tier Rail-Optimized, balanced, full-bisection network [89]. Each server is equipped with eight 400Gbps InfiniBand adapters (3.2Tbps peak per host), supporting concurrent distributed jobs without topological scheduling.

Cooling The servers are racked in *Dynamic Density Cabinets*[3]. Each cabinet houses five servers with dedicated cooling and power. Air circulates in a closed loop through an overhead plenum where it is cooled via heat transfer to water, enabling a datacenter PUE of 1.2. Under heavy utilization, H100 temperatures peak around 75°C, with typical averages between 60°C and 65°C.

	Image Encoder		**V/L Connector**		**LLM**	
Parameter	7B-D	7B-O	7B-D	7B-O	7B-D	7B-O
Params	383M	278M	121M	75M	7.6B	7.3B
Dim	1152	1024	—	—	3584	4096
MLP Dim	4304	4096	37888	22016	37888	22016
Activation	GELU	GELU	SwiGLU	SwiGLU	SwiGLU	SwiGLU
Heads	16	16	16	16	28	32
KV Heads	16	16	—	—	4	32
Layers	27	23	—	—	28	32
Image Size	384×384	336×336	—	—	—	—
Patch Size	14	14	—	—	—	—
Pool Size	—	—	2×2	2×2	—	—
Pool Dim	—	—	1152	1024	—	—
Pool Heads	—	—	16	16	—	—
Theta	—	—	—	—	1M	0.5M
Dropout	0.0	0.0	0.0	0.0	0.1	0.1

Table 3: **MOLMOACT's Architecture Hyperparameters.** We specify all hyperparameter information for the different model architectures for MOLMOACT-7B-D and MOLMOACT-7B-O.

F Action Tokenization

We provide our full action vocabulary in Table 5 and Table 6, which show the mapping from discrete bin index to its corresponding action token. Note that the string `\u00e2\u00bd\u0139` is a sequence of Unicode escape codes. Each `\uXXXX` gives one code point in hexadecimal. When decoded, those code points become the actual characters, concatenated in order.

[2] weka.io

[3] cirrascale.com/products-and-services/cabinet-technologies

	Pre-train		Mid-train		Post-train
Parameter	7B-D	7B-O	7B-D	7B-O	7B-D
Warm-up ViT	200	200	200	200	200
Warm-up Conn.	200	200	200	200	200
Warm-up LLM	200	200	200	200	200
LR ViT	1×10^{-5}	1×10^{-5}	5×10^{-6}	5×10^{-6}	5×10^{-4}
LR Conn.	1×10^{-5}	1×10^{-5}	5×10^{-6}	5×10^{-6}	5×10^{-4}
LR LLM	2×10^{-5}	2×10^{-5}	1×10^{-5}	1×10^{-5}	5×10^{-4}
Cosine Decay	10%	10%	10%	10%	10%
Eps.	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
Betas	0.9/0.95	0.9/0.95	0.9/0.95	0.9/0.95	0.9/0.95
LoRA Rank	—	—	—	—	32
LoRA Alpha	—	—	—	—	16
LoRA Dropout	—	—	—	—	0
LoRA Bias	—	—	—	—	None
Multi-image Input	No	No	Yes	Yes	Yes
Steps	100k	100k	50k	50k	Varies
Global Batch Size	512	512	256	256	64 (real) or 128 (sim)
GPUs (H100s)	256	256	128	128	32 (real) or 64 (sim)
Time (Hours)	38	32	18	15	Varies
GPU Hours	9728	8192	2304	1920	Varies

Table 4: **MOLMOACT's Training Hyperparameters.** We specify all hyperparameter information for different training schemes for MOLMOACT-7B-D and MOLMOACT-7B-O. Note that for MOLMOACT-7B-D-PRETRAIN, we train the model with 150K steps, but it reaches better performance at 100K steps.

G Evaluation Details

G.1 Evaluation on SimplerEnv (Google Robot)

We evaluate on SimplerEnv [52] simulation to test MOLMOACT's out-of-the-box performance on the Google Robot setup. The simulation evaluation consists of a Google Robot arm, front-view camera image (640 x 480 px, resized to 320 x 240 px for our case), task language instructions, and delta end-effector pose actions. SimplerEnv evaluation consists of two components – Visual Matching and Variant Aggregation.

G.2 Evaluation on LIBERO

We evaluate on the LIBERO simulation benchmark [54], which consists of a Franka Emika Panda arm in simulation with demonstrations containing front and wrist view camera images (256 x 256px), tasks language instructions, and delta end-effector pose actions. We follow prior works [5] and evaluate on the four task suites – LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long – each with 500 expert demonstration across 10 tasks. Following [5], we trained on a modified dataset which filtered out no-ops actions and unsuccessful demonstrations. Moreover, we set action chunk size to $K = 8$ for evaluation on each task suites and execute full chunks before replanning. We report details of our post-training hyperparameters for LIBERO in Table 7.

Table 5: **Action token vocabulary**: Mapping from discrete bin index (0 to 127) to the actual token string.

Bin | Action Token	Bin | Action Token	Bin | Action Token	Bin | Action Token
0 \u00e2\u00bd\u0139	1 \u00e2\u00ba\u0141	2 \u00e2\u012f\u00a8	3 \u00e1\u0137\u00b7
4 \u00ef\u00a8\u012c	5 \u00e3\u0129\u00bd	6 \u00e3\u0129\u00ba	7 \u00e2\u00bd\u00ba
8 \u00e2\u0134\u0142	9 \u00e3\u012c\u00a5	10 \u00e2\u00bc\u0143	11 \u00e2\u00b0\u00a1
12 \u00e2\u00b0\u0142	13 \u00e2\u00b0\u0141	14 \u00e2\u00b0\u0133	15 \u00e2\u00b0\u0132
16 \u00e2\u00b0\u0130	17 \u00e2\u00b0\u012f	18 \u00e2\u00b0\u0124	19 \u00e2\u0134\u00a1
20 \u00e2\u0134\u0141	21 \u00e2\u0122\u00b4	22 \u00e2\u0136\u00b2	23 \u00f0\u0135\u0131\u00a7
24 \u00ef\u00a8\u00b7	25 \u00e3\u012a\u00bc	26 \u00e2\u0140\u00b6	27 \u00e2\u0138\u00a4
28 \u00e2\u0129\u0140	29 \u00e2\u0128\u00b7	30 \u00e2\u0128\u00a4	31 \u00e1\u00a5\u00a4
32 \u00e1\u00a5\u0136	33 \u00e1\u0127\u00a3	34 \u00e0\u00ba\u0124	35 \u00ef\u00b1\u012c
36 \u00ea\u00a6\u0136	37 \u00e3\u012b\u00ab	38 \u00e3\u0127\u0138	39 \u00e3\u0126\u00a7
40 \u00e3\u0126\u0135	41 \u00e3\u0126\u012f	42 \u00e2\u0141\u00b0	43 \u00e2\u013f\u00ab
44 \u00e2\u013f\u00aa	45 \u00e2\u013d\u0131	46 \u00e2\u013d\u0129	47 \u00e2\u0137\u012c
48 \u00e2\u0136\u00bd	49 \u00e1\u00b8\u012c	50 \u00e1\u00a4\u012c	51 \u00e1\u013d\u0132
52 \u00e1\u013d\u0127	53 \u00e1\u013c\u012e	54 \u00e1\u013b\u00b3	55 \u00e0\u0142\u012e
56 \u00c6\u012a	57 \u00f0\u0141\u0127\u0135	58 \u00f0\u0141\u0127\u0127	59 \u00f0\u013f\u013c\u0131
60 \u00f0\u013f\u013c\u0126	61 \u00f0\u013f\u013b\u00bf	62 \u00f0\u013f\u013b\u00bd	63 \u00f0\u013f\u013b\u00bc
64 \u00f0\u013f\u013b\u00ba	65 \u00f0\u013f\u013b\u00b8	66 \u00f0\u013f\u013b\u00b0	67 \u00f0\u013f\u013b\u00ae
68 \u00f0\u013f\u013a\u013c	69 \u00f0\u013f\u013a\u0132	70 \u00f0\u013f\u013a\u0131	71 \u00f0\u013f\u0138\u0138
72 \u00f0\u013f\u0137\u00b1	73 \u00f0\u013f\u0137\u00a1	74 \u00f0\u013f\u0137\u012f	75 \u00f0\u013f\u0136\u0135
76 \u00f0\u013f\u0135\u00be	77 \u00f0\u013f\u0135\u00b9	78 \u00f0\u013f\u0135\u00ac	79 \u00f0\u013f\u0135\u0137
80 \u00f0\u013f\u0133\u00b3	81 \u00f0\u0138\u00a5\u00a8	82 \u00f0\u0138\u00a5	83 \u00f0\u0132\u00b1\u0127
84 \u00f0\u0132\u0143\u012c	85 \u00ef\u0143\u00b2	86 \u00ef\u00a5\u00b1	87 \u00ef\u00a5\u0142
88 \u00ef\u00a4\u00a6	89 \u00ed\u0135\u00bb	90 \u00ed\u0135\u00b6	91 \u00ed\u0135\u00ae
92 \u00ed\u0135\u00ac	93 \u00ed\u012d\u012f	94 \u00ec\u00bc\u0129	95 \u00ec\u0128\u012c
96 \u00eb\u00a1\u00bc	97 \u00ea\u00b3\u0124	98 \u00ea\u00b2\u00b4	99 \u00ea\u00b2\u013b
100 \u00e4\u00b6\u00b5	101 \u00e3\u012a\u00aa	102 \u00e2\u00b2\u00a2	103 \u00e2\u013c\u00a3
104 \u00e2\u013a\u00b5	105 \u00e2\u0136\u0140	106 \u00e1\u00b8\u00bb	107 \u00e1\u00b8\u0125
108 \u00e1\u00a8\u0123	109 \u00e1\u0142\u0126	110 \u00e1\u0136\u012c	111 \u00e1\u0136\u0127
112 \u00e1\u0134\u012e	113 \u00e1\u0132\u00a7	114 \u00e1\u012e\u0136	115 \u00e1\u012e\u0126
116 \u00e1\u012d\u00a9	117 \u00e1\u012c\u0134	118 \u00e1\u012b\u00a8	119 \u00e1\u0123\u00bc
120 \u00e1\u0122\u0131	121 \u00e0\u00b2\u0141	122 \u00e0\u00b0\u00b5	123 \u00e0\u00b0\u00b3
124 \u00e0\u00ac\u012b	125 \u00e0\u00a5\u00b1	126 \u00e0\u00a4\u0133	127 \u00dd\u00a5

G.3 Evaluation on Real-world Post-training

To evaluate MOLMOACT 's efficiency in fine-tuning, we curated six tasks: three for a single-arm Franka setup—`put bowl in sink`, `wipe table`, and `table bussing`—and three for a bimanual Franka setup—`set table`, `lift tray`, and `fold towel`. We benchmarked against OpenVLA and π_0-FAST by training each model until convergence. In the single-arm setup, the Franka was mounted on a movable platform to allow relocation across different positions, whereas the bimanual setup was fixed to a tabletop configuration. For sensing, we employed an Intel RealSense D405 for the wrist-mounted camera and an Intel RealSense D435 for the front-facing view. In each efficiency fine-tuning task evaluation, we pre-marked the locations of all task objects in the scene to ensure that the evaluation conditions matched the distribution of the demonstrations used for fine-tuning. We defined the task, its description, the corresponding language instruction, and the task progression metric ratings. Refer to the complete results in Table 15 to 20.

1. **Task Name:** `put_bowl_in_sink`
   Task Description: The robot picks up the orange bowl next to the sink and place it all the way into the sink.
   Language Description: *Put the bowl into the sink.*
   Task Progression Score Metrics: grasp bowl (0.25), move into the sink (0.4), open gripper (0.7), drop bowl at target location (1).

Table 6: **Action token vocabulary**: Mapping from discrete bin index (128 to 255) to the actual token string.

Bin | Action Token	Bin | Action Token	Bin | Action Token	Bin | Action Token
128 \u00dd\u0135	129 \u00d4\u0133	130 \u00d4\u012a	131 \u00ca\u00b6
132 \u00c8\u00b2	133 \u00f0\u0141\u0131\u0129	134 \u00f0\u0141\u0127\u00a2	135 \u00f0\u013f\u013c\u0123
136 \u00f0\u013f\u013b\u013e	137 \u00f0\u013f\u0135\u00b0	138 \u00f0\u013f\u0135\u0140	139 \u00f0\u0132\u00b0\u00bc
140 \u00f0\u0132\u0143\u0135	141 \u00f0\u0132\u00a4\u0136	142 \u00ef\u00a8\u0124	143 \u00ef\u00a7\u00a9
144 \u00ef\u00a6\u0125	145 \u00ef\u00a4\u0128	146 \u00ef\u00a4\u0127	147 \u00ed\u013d\u013e
148 \u00ed\u0137\u00b1	149 \u00ed\u0135\u0143	150 \u00ed\u0135\u0138	151 \u00ed\u0125\u013b
152 \u00ed\u0123\u00bb	153 \u00ec\u00bb\u0123	154 \u00ec\u00b3\u0127	155 \u00ec\u013e\u00be
156 \u00ec\u013d\u00a2	157 \u00eb\u00b1\u0132	158 \u00eb\u00b1\u012d	159 \u00eb\u00a7\u0142
160 \u00eb\u00a4\u0124	161 \u00eb\u0138\u00b0	162 \u00e2\u00a4\u00a6	163 \u00e2\u00a1\u00a2
164 \u00e2\u013c\u0139	165 \u00e2\u013c\u0124	166 \u00e2\u013b\u013b	167 \u00e1\u00bf\u013c
168 \u00e1\u00bf\u0132	169 \u00e1\u00be\u0136	170 \u00e1\u00b6\u0131	171 \u00e1\u00a9\u012d
172 \u00e1\u00a8\u00b8	173 \u00e1\u0142\u00ac	174 \u00e1\u0142\u0124	175 \u00e1\u0136\u0143
176 \u00e1\u012e\u00bd	177 \u00e1\u012e\u0125	178 \u00e1\u012b\u0132	179 \u00e1\u012a\u00be
180 \u00e1\u012a\u00a8	181 \u00e1\u012a\u012c	182 \u00e1\u0128\u00ba	183 \u00e0\u00bd\u0127
184 \u00e0\u00b4\u00b4	185 \u00d5\u0125	186 \u00ca\u0135	187 \u00c9\u013a
188 \u00f0\u0141\u0137\u012d	189 \u00f0\u0141\u0128\u0134	190 \u00f0\u0141\u0127\u00b1	191 \u00ef\u00ae\u0131
192 \u00ed\u0137\u00ae	193 \u00ed\u012c\u0143	194 \u00ec\u00a5\u012b	195 \u00ec\u0142\u00b0
196 \u00ec\u0141\u013b	197 \u00ec\u013f\u00bf	198 \u00ec\u013f\u00a9	199 \u00ec\u0139\u00a4
200 \u00ec\u0131\u00b1	201 \u00ec\u012d\u00b2	202 \u00ec\u012b\u00a1	203 \u00ec\u0126\u0132
204 \u00eb\u00bc\u013f	205 \u00eb\u00bb\u0127	206 \u00eb\u00af\u0133	207 \u00eb\u00a1\u0133
208 \u00eb\u0139\u012f	209 \u00eb\u0136\u012b	210 \u00ea\u00b8\u0133	211 \u00ea\u013b\u012d
212 \u00e3\u00b3\u00ac	213 \u00e2\u013d\u00a4	214 \u00e2\u013c\u00a7	215 \u00e2\u0126\u00ac
216 \u00e1\u00bd\u013f	217 \u00e1\u00bc\u00ae	218 \u00e1\u00ba\u0122	219 \u00e1\u00b8\u00b0
220 \u00e1\u00a1\u012e	221 \u00da\u0130	222 \u00d1\u00a8	223 \u00f0\u0141\u0139\u0123
224 \u00f0\u0141\u0138\u00b6	225 \u00f0\u0141\u0138\u0133	226 \u00f0\u0141\u0138\u0129	227 \u00f0\u0141\u0137\u00b3
228 \u00f0\u0141\u0137\u00a2	229 \u00f0\u0141\u0137\u0142	230 \u00f0\u0141\u0137\u0140	231 \u00f0\u0141\u0137\u013f
232 \u00f0\u0141\u0137\u013e	233 \u00f0\u0141\u0137\u013c	234 \u00f0\u0141\u0137\u0138	235 \u00f0\u0141\u0136\u00a9
236 \u00f0\u0141\u0136\u00a4	237 \u00f0\u0141\u0136\u00a2	238 \u00f0\u0141\u0136\u0135	239 \u00f0\u0141\u0136\u0129
240 \u00f0\u0141\u0136\u0125	241 \u00f0\u0141\u0136\u0124	242 \u00f0\u0141\u0136\u0122	243 \u00f0\u0141\u0135\u00bc
244 \u00f0\u0141\u0135\u00aa	245 \u00f0\u0141\u0135\u0141	246 \u00f0\u0141\u0134\u00ba	247 \u00f0\u0141\u0134\u00b9
248 \u00f0\u0141\u0133\u013f	249 \u00f0\u0141\u0132\u0122	250 \u00f0\u0141\u0131\u00af	251 \u00f0\u0141\u0131\u00a9
252 \u00f0\u0141\u0131\u0134	253 \u00f0\u0141\u0131\u0131	254 \u00f0\u0141\u0130\u00bf	255 \u00f0\u0141\u0130\u0133

2. **Task Name:** `wipe_table`
   Task Description: The robot grasp onto the table cloth, and move across the surface in one direction.
   Language Description: *Wipe the table.*
   Task Progression Score Metrics: Grasp the towel (0.25), Move in the right direction (0.5), Complete the wipe (1).

3. **Task Name:** `table_bussing`
   Task Description: The robot grasp onto the green tea can and place it into the purple bin.
   Language Description: *Clean the trash into the bin*
   Task Progression Score Metrics: Grasp onto the can (0.25), Lift up the can (0.5), Move to above the bin (0.75), Drop the can into the bin (1).

4. **Task Name:** `set_table`
   Task Description: The right arm grasp onto the banana and place it onto the plate, and the left arm grasp onto the teapot to pour.
   Language Description: *Set the table*
   Task Progression Score Metrics: Put banana on plate (0.25), Grasp onto the teapot (0.75), Pour the tea (1).

5. **Task Name:** `lift_tray`
   Task Description: The left and right arm approaches the box and grasp onto it, and lift up the

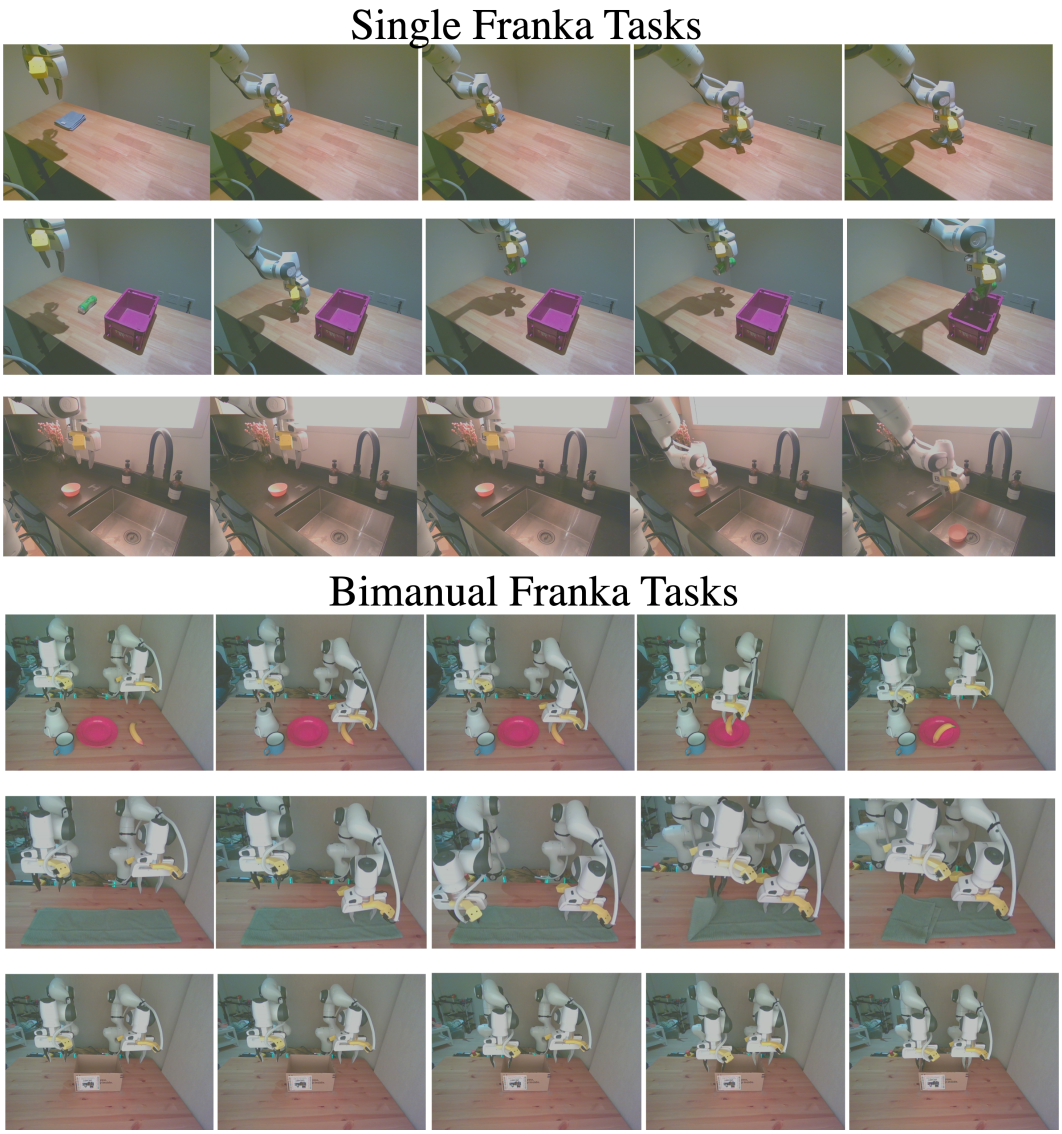


Figure 9: **Examples of Single-arm and Bimanual Tasks.** We list the observation breakdown to show how the robot performs each task.

box together.
Language Description: *Lift up the box*
Task Progression Score Metrics: Left arm grasp onto the tray (0.3), Right arm grasp onto the tray (0.6), Both arms lift up the tray (1).

6. **Task Name:** `fold_towel`
Task Description: The right arm press down on the centre of the towel, while the left arm grasp onto the towel to fold.
Language Description: *Fold the towel*
Task Progression Score Metrics: Grasp onto the towel (0.25), Put the towel over the right location for folding (0.75), Drop the towel so that it is folded (1).

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table 10 (single-arm) and Table 11 (bimanual). For all other baseline models, we follow their official model

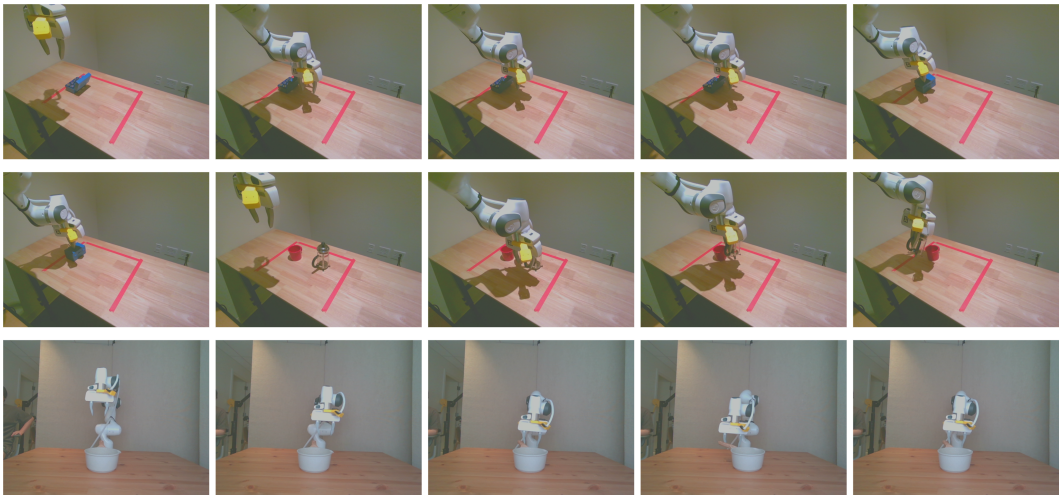


Figure 10: **Examples of MOLMOACT DATASET ablation experiments.**

Figure 11: **Language instruction following.** These are the customized scenes curated for open-ended prompting by users.

and training implementation and use their default configurations. We also make sure that they are all fully converged. Image examples of each task are shown in Figure 9.

G.4 Evaluation on Generalization in Real-world

We collected a multi-task set that contains the full permutation of the scene: `put_green_can_in_yellow_plate` (*put the green can into the yellow plate*), `put_green_can_in_blue_plate` (*put the green can into the blue plate*), `put_red_cup_in_yellow_plate` (*put the red cup into the yellow plate*), and `put_red_cup_in_blue_plate` (*put the red cup into the blue plate*), and `put_banana_in_yellow_plate` (*put the banana into the yellow plate*). `put_banana_in_blue_plate` (*put the banana into the blue plate*). And all models are trained on all tasks under a multi-task setting.

We evaluated generalization across four perturbations and one in-distribution setting on three tasks drawn from the previous multi-task set: `put_green_can_in_yellow_plate` (*put the green can into the yellow plate*), `put_red_cup_in_yellow_plate` (*put the red cup into the yellow plate*), and `put_banana_in_blue_plate` (*put the banana into the blue plate*). The perturbations were: (1) **Language variation** – modified instructions to *put the green tea into the yellow plate*, *put the fruit into the blue plate*, and *put the red cylinder into the yellow plate*; (2) **Spatial variation** – altered the positions of objects in each task; (3) **Distractors** – added unrelated distractor objects to the scene; and (4) **Novel objects** – replaced the green can with a sponge, the red cup with a coke can, and the banana with a bowl.

- **Task Name:** `put_<object>_in_(yellow/blue)_plate`
  Task Description: The robot first pickup the <object>, then put it into the yellow/blue plate.
  Language Description: *Put the <object> into the yellow/blue plate*

	LIBERO Task Suite			
Parameter	Spatial	Object	Goal	Long
Steps	50K	50K	40K	80K
Global Batch Size	128			
GPUs (H100s)	64			
Time (Hours)	23	23	18	36
GPU Hours	1472	1472	1152	2304
Input Images	1 Third-person + 1 Wrist-mounted			
Image Size	256×256 px			
DoF	7 (3 Translations + 3 Rotations + 1 Gripper State)			
Observation History	No (Single-step Inputs)			
Use Proprioception	No			
Action Chunk Size	8 Steps (Predict 8; Execute All 8 Open-loop)			
# Trainable Params	97 M LoRA adapter			

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width), scale=(0.9, 0.9),`
	`ratio=(width/height, width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`			

Table 7: **MOLMOACT's Post-training Hyperparameters for LIBERO.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias. Note that for LIBERO-Goal, we train the model with 50K steps, but it reaches better performance at 40K steps.

Task Progression Score Metrics: Move towards the correct <object> (0.25). Pick up the correct <object> (0.5). Move towards the correct plate (0.75). Put the correct <object> into the correct plate (1).

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table 12. For all other baseline models, we follow their official model and training implementation and use their default configurations. We also make sure that they are all fully converged. The full details of this evaluation are listed in Table 21.

G.5 Evaluation on the Effect of MOLMOACT DATASET for MOLMOACT Mid-training

To evaluate the effectiveness of mid-training with the MOLMOACT DATASET, we curated three real-world tasks: `close_lid`, `rotate_pot`, and `pour_tea`. For each task, we collected 50 demonstrations and pre-marked object locations to ensure repeatability in evaluating MOLMOACT, MOLMOACT without the MOLMOACT DATASET, OpenVLA, and π_0-FAST. We conducted 10 evaluation trials per task for each model. Refer to the complete results in Table 22

1. **Task Name:** `close_lid`
   Task Description: The robot goes to the back of the lid, closes its gripper and push the lid to close.
   Language Description: *Close the lid*

Task Progression Score Metrics: Move the lid towards the closing direction (0.5). Close the lid (1).

2. **Task Name:** `rotate_pot`
   Task Description: The robot goes to a target position to the handle, and rotate it by 90 degree.
   Language Description: *Rotate the pot*
   Task Progression Score Metrics: Go target position of pot handle (0.3). Rotate the pot by 45 degree (0.6). Close the 90 degree rotation (1).

3. **Task Name:** `pour_tea`
   Task Description: The robot grasp onto the teapot handle, and lift it up to above the cup to pour.
   Language Description: *Pour tea into cup*
   Task Progression Score Metrics: Grasp onto the teapot (0.5). Move the teapot on top of cup (0.8). Pour tea into cup (1).

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table 13. For all other baseline models, we follow their official model and training implementation and use their default configurations. We also make sure that they are all fully converged. Image examples of each task are shown in Figure 10.

G.6 Evaluation of MOLMOACT on Instruction-following

For the evaluation of language-instruction following, we curated five customized scenes using SimplerEnv [52] and asked participants to provide open-ended prompts for each scene. After filtering, we obtained 29 prompts in total, which were executed by MOLMOACT, OpenVLA, and SpatialVLA for 200 steps to generate robot rollouts. These rollouts were then rated by 100 participants in an arena-style interface. Images of different scenes are shown in Figure 11, and language prompts are shown in Table 8.

G.7 Evaluation of MOLMOACT on Action Steerability

We curated the task `pick_up_bowl`, featuring one dirty and one clean bowl. As shown in Figure 7, we built a web interface that enables users to modify the language instruction or sketch five points on the image for visual trace steering at test time, which are then passed to the model to generate actions. We evaluate task progression for this task based on: correct direction of the target bowel (0.5), grasp onto the correct bowl (0.85), lift up the bowl (1).

Unlike the usual straightforward way of collecting tele-operated real-world demonstrations, where we control the robot to directly complete the task, we collected half of the number of demonstrations in the regular way and the other half exploring alternative paths towards the same target conditioned on the language. Thus, in total, we have 50 demonstrations picking up the dirty bowl, 50 demonstrations picking up the clean bowl, 50 demonstrations picking up the dirty bowl while exploring other paths, and 50 demonstrations picking up the clean bowl while exploring other paths. We believe that this helps the model to learn more about how visual traces correlate with physical actions.

During test time, we collected open-ended instructions from 10 participants to steer the robot through language. We restrict the variation of the open-ended instructions only to verbs, nouns, or adjectives. The collected and used instructions are shown in Table 9.

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table 14. For all other baseline models, we follow their official model and training implementation and use their default configurations. We also make sure that they are all fully converged. All results are reported Table 23.

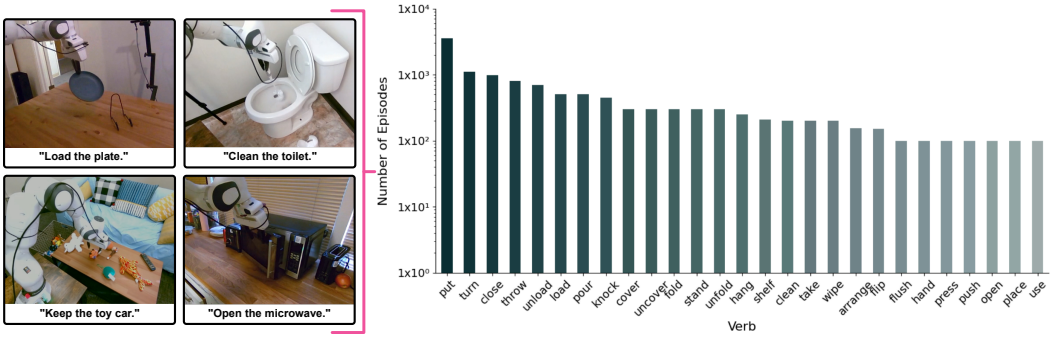


Figure 12: **Examples and verb distribution in the MOLMOACT DATASET.** Left: Sample robot manipulation tasks paired with natural language instructions, spanning diverse household activities such as closing a laptop, loading a plate, cleaning a toilet, and opening a microwave. Right: Log-scale distribution of the top verbs in the dataset, showing a long-tail pattern with "put," "turn," and "close" as the most frequent actions.

H Data Details

H.1 MOLMOACT DATASET

MOLMOACT DATASET has two external camera views and a single wrist camera view. In the home environment data, the camera view configuration may vary between tasks, whereas for the tabletop data it remains the same for all tasks. For each task, we first rank the two external camera views based on scene clarity (i.e., how well the robot and objects are visible) and whether the view is occluded by the robot during task execution. Based on this ranking, we label them as the *primary* and *secondary* camera views. All home environment data is recorded at 15 Hz, while all tabletop data is recorded at 20 Hz. The tabletop data additionally includes extrinsic and intrinsic camera calibration matrices for both external cameras available on . We list details of the tasks we collected for MOLMOACT Dataset in Table 24 and 25.

I Data Examples

This section include **randomly selected** examples from MOLMOACT's Action Reasoning Data and Multimodal web data used in pre-training, as well as MOLMOACT DATASET used in mid-training, and demonstrations collected for post-training. Prompts are shown in bold and Visual Reasoning Trace are annotated with a yellow line.

- **Action Reasoning Data - Figure 13**
- **Auxiliary Visual Reasoning Trace - Figure 14**
- **Auxiliary Depth Perception Tokens - Figure 15**
- **Trajectory-conditioned Action Data - Figure 16**
- **Multimodal Web Data - Figure 17**
- **MOLMOACT DATASET (Home Environment) - Figure 18**
- **MOLMOACT DATASET (Tabletop) - Figure 19**
- **Post-Training Single Arm Franka - Figure 20**
- **Post-Training Bimanual Franka - Figure 21**
- **Post-Training Rainbow - Figure 22**

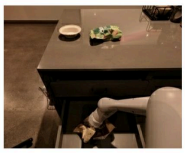
The task is knock redbull can over. What is the action that the robot should take. To figure out the action that the robot should take to knock redbull can over, let's think through it step by step. First, what is the depth map for this image? Second, what is the trajectory of the end effector? Based on the depth map of the image and the trajectory of the end effector, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_116><DEPTH_116><DEPTH_55><DEPTH_9><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_8><DEPTH_8><DEPTH_55><DEPTH_28><DEPTH_55><DEPTH_32><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_9><DEPTH_69><DEPTH_9><DEPTH_55><DEPTH_28><DEPTH_116><DEPTH_8><DEPTH_116><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_9><DEPTH_9><DEPTH_116><DEPTH_74><DEPTH_56><DEPTH_9><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_75><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_30><DEPTH_116><DEPTH_9><DEPTH_56><DEPTH_25><DEPTH_116><DEPTH_69><DEPTH_9><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_69><DEPTH_28><DEPTH_8><DEPTH_32><DEPTH_116><DEPTH_69><DEPTH_9><DEPTH_116><DEPTH_101><DEPTH_75><DEPTH_9><DEPTH_116><DEPTH_28><DEPTH_9><DEPTH_55><DEPTH_101><DEPTH_56><DEPTH_32><DEPTH_75><DEPTH_56><DEPTH_9><DEPTH_25><DEPTH_55><DEPTH_69><DEPTH_69><DEPTH_116><DEPTH_28><DEPTH_28><DEPTH_26><DEPTH_26><DEPTH_8><DEPTH_30><DEPTH_8><DEPTH_9><DEPTH_8><DEPTH_30><DEPTH_30><DEPTH_8><DEPTH_28><DEPTH_9><DEPTH_END>. The trajectory of the end effector is [[132,207],[148,192],[195,107],[145,37],[183,33]]. Based on these information, the action that the robot should take is [ï³h, áÌHf, ïœÎ, áÌHf, à¬ï, âÈÌ, Ÿ¥].

The task is Place the spoon on the lower left table edge. What is the action that the robot should take. To figure out the action that the robot should take to Place the spoon on the lower left table edge, let's think through it step by step. First, what is the depth map for this image? Second, what is the trajectory of the end effector? Based on the depth map of the image and the trajectory of the end effector, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_75><DEPTH_101><DEPTH_69><DEPTH_8><DEPTH_69><DEPTH_56><DEPTH_56><DEPTH_30><DEPTH_32><DEPTH_101><DEPTH_26><DEPTH_56><DEPTH_32><DEPTH_116><DEPTH_56><DEPTH_101><DEPTH_32><DEPTH_8><DEPTH_69><DEPTH_8><DEPTH_75><DEPTH_32><DEPTH_116><DEPTH_9><DEPTH_9><DEPTH_116><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_69><DEPTH_101><DEPTH_75><DEPTH_116><DEPTH_69><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_69><DEPTH_9><DEPTH_101><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_32><DEPTH_56><DEPTH_56><DEPTH_8><DEPTH_8><DEPTH_69><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_9><DEPTH_56><DEPTH_56><DEPTH_56><DEPTH_56><DEPTH_8><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_56><DEPTH_69><DEPTH_101><DEPTH_56><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_8><DEPTH_56><DEPTH_101><DEPTH_69><DEPTH_69><DEPTH_75><DEPTH_101><DEPTH_69><DEPTH_25><DEPTH_74><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_56><DEPTH_101><DEPTH_26><DEPTH_26><DEPTH_25><DEPTH_76><DEPTH_76><DEPTH_76><DEPTH_103><DEPTH_69><DEPTH_25><DEPTH_74><DEPTH_116><DEPTH_116><DEPTH_END>. The trajectory of the end effector is [[30,39],[56,81],[58,70]]. Based on these information, the action that the robot should take is [áÌJ, âÈÌ, ðIJœÈ, ï³h, ë±IJ, á¿IJ, ðŁIij].

The task is pick brown chip bag from middle drawer and place on counter. What is the action that the robot should take. To figure out the action that the robot should take to pick brown chip bag from middle drawer and place on counter, let's think through it step by step. First, what is the depth map for this image? Second, what is the trajectory of the end effector? Based on the depth map of the image and the trajectory of the end effector, what is the action that the robot should take?



The depth map of the image is
<DEPTH_START><DEPTH_116><DEPTH_116><DEPTH_55><DEPTH_9><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_8><DEPTH_8><DEPTH_55><DEPTH_28><DEPTH_55><DEPTH_32><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_9><DEPTH_69><DEPTH_9><DEPTH_55><DEPTH_28><DEPTH_116><DEPTH_8><DEPTH_116><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_9><DEPTH_9><DEPTH_116><DEPTH_74><DEPTH_56><DEPTH_9><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_75><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_30><DEPTH_116><DEPTH_9><DEPTH_56><DEPTH_25><DEPTH_116><DEPTH_69><DEPTH_9><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_69><DEPTH_28><DEPTH_8><DEPTH_32><DEPTH_116><DEPTH_69><DEPTH_9><DEPTH_116><DEPTH_101><DEPTH_75><DEPTH_9><DEPTH_116><DEPTH_28><DEPTH_9><DEPTH_55><DEPTH_101><DEPTH_56><DEPTH_32><DEPTH_75><DEPTH_56><DEPTH_9><DEPTH_25><DEPTH_55><DEPTH_69><DEPTH_69><DEPTH_116><DEPTH_28><DEPTH_28><DEPTH_26><DEPTH_26><DEPTH_8><DEPTH_30><DEPTH_8><DEPTH_9><DEPTH_8><DEPTH_30><DEPTH_30><DEPTH_8><DEPTH_28><DEPTH_9><DEPTH_END>. The trajectory of the end effector is [[132,207],[148,192],[195,107],[145,37],[183,33]]. Based on these information, the action that the robot should take is [ï³h, áÌHf, ïœÎ, áÌHf, à¬ï, âÈÌ, Ÿ¥].

The task is wipe red bowl with towel. What is the action that the robot should take. To figure out the action that the robot should take to wipe red bowl with towel, let's think through it step by step. First, what is the depth map for this image? Second, what is the trajectory of the end effector? Based on the depth map of the image and the trajectory of the end effector, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_25><DEPTH_9><DEPTH_74><DEPTH_8><DEPTH_56><DEPTH_55><DEPTH_56><DEPTH_74><DEPTH_25><DEPTH_28><DEPTH_28><DEPTH_55><DEPTH_69><DEPTH_116><DEPTH_8><DEPTH_69><DEPTH_116><DEPTH_28><DEPTH_69><DEPTH_8><DEPTH_101><DEPTH_32><DEPTH_69><DEPTH_8><DEPTH_8><DEPTH_76><DEPTH_9><DEPTH_8><DEPTH_8><DEPTH_69><DEPTH_69><DEPTH_103><DEPTH_69><DEPTH_103><DEPTH_28><DEPTH_69><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_28><DEPTH_28><DEPTH_25><DEPTH_9><DEPTH_103><DEPTH_9><DEPTH_55><DEPTH_25><DEPTH_28><DEPTH_56><DEPTH_116><DEPTH_32><DEPTH_26><DEPTH_75><DEPTH_30><DEPTH_55><DEPTH_55><DEPTH_28><DEPTH_75><DEPTH_26><DEPTH_101><DEPTH_56><DEPTH_101><DEPTH_26><DEPTH_69><DEPTH_116><DEPTH_25><DEPTH_25><DEPTH_25><DEPTH_56><DEPTH_116><DEPTH_55><DEPTH_32><DEPTH_26><DEPTH_75><DEPTH_8><DEPTH_30><DEPTH_28><DEPTH_25><DEPTH_8><DEPTH_32><DEPTH_69><DEPTH_116><DEPTH_26><DEPTH_26><DEPTH_9><DEPTH_55><DEPTH_28><DEPTH_30><DEPTH_55><DEPTH_9><DEPTH_25><DEPTH_84><DEPTH_55><DEPTH_55><DEPTH_32><DEPTH_30><DEPTH_30><DEPTH_END>. The trajectory of the end effector is [[53,192],[53,196],[54,199],[53,205],[50,192]]. Based on these information, the action that the robot should take is [ðIJNj, ðŁIg, à¬ï, âœIij, ðŁJŁ, ïœh, Ÿ¥].

Figure 13: Randomly selected examples from **Action Reasoning Data** used in the pre-training stage.

Scene	Prompts
Scene 1	• Pick up the green cube • Pick up the red cube • Pick up the blue cube • Put the green cube on the blue cube • Put the green cube on the red cube • Put the blue cube onto the red cube • Put the blue cube onto the green cube • Put the red cube onto the green cube • Put the red cube onto the blue cube • Put the green cube onto the blue cube and then the red cube onto the green cube • Move the blue cube next to the green cube
Scene 2	• Pick up the apple • Pick up the spoon • Put the apple onto the plate • Put the spoon onto the plate • Put the spoon next to the plate • Move the spoon to the right of the apple • Put the apple onto the plate and move the spoon nearer to the plate • Put the blue cube onto the green cube • Put the red cube onto the green cube • Put the red cube onto the blue cube • Put the green cube onto the blue cube and then the red cube onto the green cube
Scene 3	• Put the coke can onto the former president's image • Put the coke can on the image of Obama • Move the coke can to the image of Taylor Swift
Scene 4	• Pick up the cup • Pick up the marker • Put the marker into the mug • Pick up the mug by the handle
Scene 5	• Pick up the bowl • Pick up the Red Bull • Put the Red Bull onto the plate • Put the Red Bull in the red bowl

Table 8: Open-ended prompts provided by users grouped by scene for language instruction following.

#	Instruction
1	pick up the orange bowl
2	lift up the dirty bowl
3	pick up the bowl on the left
4	pick up the empty bowl
5	pick up the dirty container
6	pick up the bowl with object inside
7	pick up the left bowl
8	pick up the bowl that is pink
9	pick up the bowl that is pink
10	pick up the bowl further
11	pick up the bowl nearer to the camera
12	pick up the right bowl
13	pick up the bowl without tissue
14	pick up the bowl with tissue
15	pick up the bowl that is dirty

Table 9: **Open-ended Language Instructions.** These are the collected open-ended instructions from 10 participants, where they were only allowed to make changes to verbs, nouns, or adjectives from the ground-truth instructions (i.e, "<verb> the <adj.> <noun.>").

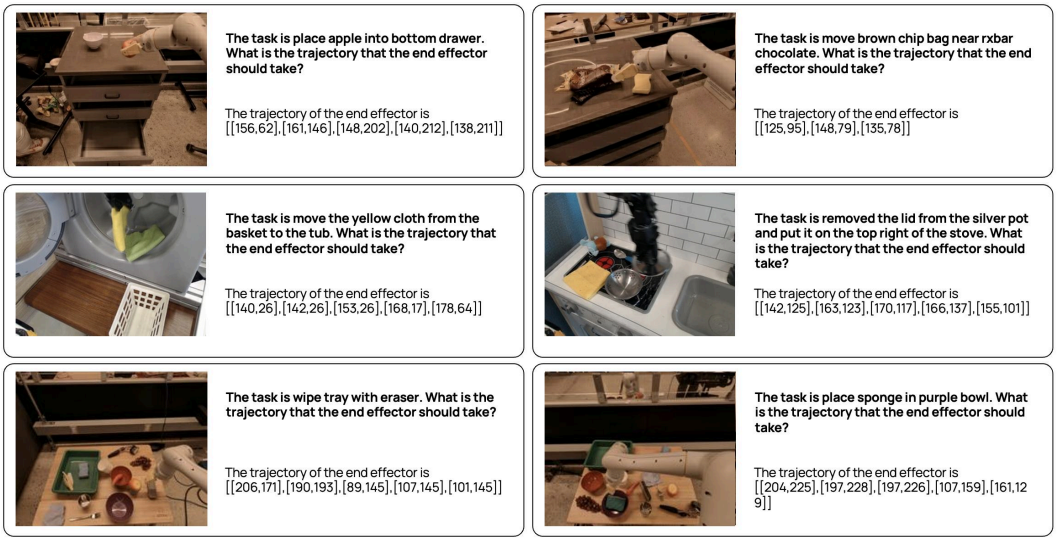


Figure 14: Randomly selected examples from **Auxiliary Visual Reasoning Trace** data used in the pre-training stage.

	Task Name		
Parameter	`put_bowl_in_sink`	`wipe_table`	`table_bussing`
Steps	9K	7K	5K
Global Batch Size		64	
GPUs (H100s)		32	
Time (Hours)	5	4	3
GPU Hours	160	128	96
Multi-task Training		No	
Input Images		1 Third-person + 1 Wrist-mounted	
Image Size		640×320 px (Resized to 320×240 px)	
DoF		7 (3 Translations + 3 Rotations + 1 Gripper State)	
Observation History		No (Single-step Inputs)	
Use Proprioception		No	
Action Chunk Size		8 Steps (Naive Action Chunking with Close-loop Prediction)	
# Trainable Params		97M LoRA adapter	

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width),`
	`scale=(0.9, 0.9), ratio=(width/height,`
	`width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`		

Table 10: **MOLMOACT's Post-training Hyperparameters for In-distribution Single-arm Tasks.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias.

	Task Name		
Parameter	`set_table`	`lift_tray`	`fold_towel`
Steps	9K	6K	7K
Global Batch Size		64	
GPUs (H100s)		32	
Time (Hours)	5	3	4
GPU Hours	160	96	128
Multi-task Training		No	
Input Images		1 Third-person + 2 Wrist-mounted	
Image Size		640×320 px (Resized to 320×240 px)	
DoF		14 (6 Translations + 6 Rotations + 2 Gripper States)	
Observation History		No (Single-step Inputs)	
Use Proprioception		No	
Action Chunk Size		8 Steps (Naive Action Chunking with Close-loop Prediction)	
# Trainable Params		97M LoRA adapter	

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width),`
	`scale=(0.9, 0.9), ratio=(width/height,`
	`width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`		

Table 11: **MOLMOACT's Post-training Hyperparameters for In-distribution Bimanual Tasks.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias.

	Task Name
Parameter	`put_(green_can/red_cup/banana)_in_(yellow/blue)_plate`
Steps	44K
Global Batch Size	64
GPUs (H100s)	32
Time (Hours)	23
GPU Hours	736
Multi-task Training	Yes
Input Images	1 Third-person + 1 Wrist-mounted
Image Size	640×320 px (Resized to 320×240 px)
DoF	7 (3 Translations + 3 Rotations + 1 Gripper State)
Observation History	No (Single-step Inputs)
Use Proprioception	No
Action Chunk Size	8 Steps (Naive Action Chunking with Close-loop Prediction)
# Trainable Params	97M LoRA adapter

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width),`
	`scale=(0.9, 0.9), ratio=(width/height,`
	`width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`

Table 12: **MOLMOACT's Post-training Hyperparameters for Out-of-distribution Single-arm Tasks.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias.

	Task Name		
Parameter	`close_lip`	`rotate_pot`	`pour_tea`
Steps	8K	6K	15K
Global Batch Size		64	
GPUs (H100s)		32	
Time (Hours)	4	3	8
GPU Hours	128	96	256
Multi-task Training		No	
Input Images		1 Third-person + 1 Wrist-mounted	
Image Size		640×320 px (Resized to 320×240 px)	
DoF		7 (3 Translations + 3 Rotations + 1 Gripper State)	
Observation History		No (Single-step Inputs)	
Use Proprioception		No	
Action Chunk Size		8 Steps (Naive Action Chunking with Close-loop Prediction)	
# Trainable Params		97M LoRA adapter	

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width),`
	`scale=(0.9, 0.9), ratio=(width/height,`
	`width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`		

Table 13: **MOLMOACT's Post-training Hyperparameters for Evaluation on MOLMOACT DATASET.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias.

	Task Name
Parameter	`pick_up_bowl`
Steps	11K
Global Batch Size	64
GPUs (H100s)	32
Time (Hours)	6
GPU Hours	192
Multi-task Training	No
Input Images	1 Third-person + 1 Wrist-mounted
Image Size	640×320 px (Resized to 320×240 px)
DoF	7 (3 Translations + 3 Rotations + 1 Gripper State)
Observation History	No (Single-step Inputs)
Use Proprioception	No
Action Chunk Size	8 Steps (Naive Action Chunking with Close-loop Prediction)
# Trainable Params	97M LoRA adapter

Image Augmentations	`import torchvision.transforms as T`
	`transform = T.Compose([`
	`  T.RandomResizedCrop(size=(height, width),`
	`scale=(0.9, 0.9), ratio=(width/height,`
	`width/height)),`
	`  T.Resize((height, width)),`
	`  T.ColorJitter(`
	`  brightness=0.2,`
	`  contrast=(0.8, 1.2),`
	`  saturation=(0.8, 1.2),`
	`  hue=0.05`
	`  ),`
	`])`

Table 14: **MOLMOACT's Post-training Hyperparameters for Steerability Evaluation.** We specify the hyperparameters for MOLMOACT post-training. Note that we conduct all our post-training experiments on MOLMOACT-7B-D, with a fixed learning rate of 5e-4, LoRA rank of 32, LoRA alpha of 16, LoRA dropout of 0, and no LoRA bias.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Fold Towel	0	0.25	0.25	0.25
	1	1.00	0.50	0.25
	2	1.00	0.25	0.25
	3	1.00	1.00	0.50
	4	1.00	0.25	0.25
	5	1.00	0.25	0.25
	6	1.00	0.25	1.00
	7	1.00	0.25	0.25
	8	0.25	0.25	0.25
	9	1.00	1.00	0.25
	10	1.00	1.00	0.25
	11	1.00	1.00	0.25
	12	1.00	1.00	0.25
	13	0.75	0.25	0.25
	14	0.25	0.25	0.25
	15	0.25	0.00	0.25
	16	1.00	0.25	0.25
	17	1.00	1.00	0.25
	18	0.25	0.50	0.25
	19	0.75	1.00	0.25
	20	0.25	0.25	0.25
	21	1.00	0.25	0.25
	22	1.00	0.25	0.25
	23	1.00	0.75	0.25
	24	1.00	1.00	1.00
Average		**0.80**	**0.52**	**0.32**

Table 15: Detailed per-trial performance for *Fold Towel* for Bimanual tasks. Each row shows the task progress score for a specific trial.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Lift Tray	0	1.00	1.00	1.00
	1	1.00	0.00	1.00
	2	1.00	0.60	1.00
	3	1.00	1.00	1.00
	4	1.00	0.60	1.00
	5	1.00	0.00	1.00
	6	1.00	1.00	1.00
	7	1.00	1.00	1.00
	8	1.00	1.00	1.00
	9	1.00	1.00	1.00
	10	1.00	1.00	1.00
	11	1.00	1.00	1.00
	12	1.00	1.00	1.00
	13	1.00	1.00	1.00
	14	1.00	1.00	1.00
	15	1.00	0.00	1.00
	16	1.00	1.00	1.00
	17	1.00	1.00	1.00
	18	1.00	0.00	1.00
	19	1.00	1.00	1.00
	20	1.00	1.00	1.00
	21	1.00	0.00	1.00
	22	1.00	0.30	1.00
	23	1.00	1.00	1.00
	24	1.00	1.00	1.00
Average		**1.00**	**0.74**	**1.00**

Table 16: Detailed per-trial performance for *Lift Tray* for Bimanual tasks. Each row shows the task progress score for a specific trial.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Set up Table	0	1.00	0.50	0.25
	1	1.00	0.00	0.25
	2	0.25	0.25	0.00
	3	1.00	0.50	0.00
	4	1.00	0.25	0.25
	5	1.00	0.25	0.75
	6	1.00	0.00	0.25
	7	1.00	0.25	0.25
	8	0.75	0.00	0.25
	9	0.25	0.00	1.00
	10	1.00	0.25	0.25
	11	0.75	0.25	0.25
	12	1.00	0.75	0.25
	13	0.75	0.25	0.25
	14	1.00	0.25	0.25
	15	0.25	0.50	1.00
	16	0.00	0.25	0.25
	17	1.00	0.25	0.25
	18	1.00	0.00	0.25
	19	0.25	0.50	0.25
	20	1.00	0.00	0.25
	21	1.00	0.50	0.00
	22	0.50	0.00	0.25
	23	0.50	0.00	0.25
	24	1.00	0.25	0.25
	Average	**0.77**	**0.24**	**0.30**

Table 17: Detailed per-trial performance for *Set up Table* for Bimanual tasks. Each row shows the task progress score for a specific trial.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Put bowl in the sink	0	1.00	1.00	0.25
	1	1.00	1.00	0.25
	2	1.00	0.00	0.25
	3	0.25	1.00	0.25
	4	1.00	1.00	0.25
	5	1.00	0.40	0.25
	6	1.00	1.00	0.25
	7	0.25	0.25	0.25
	8	1.00	0.25	0.25
	9	0.40	0.40	0.25
	10	1.00	1.00	0.25
	11	0.25	1.00	0.25
	12	1.00	1.00	0.25
	13	1.00	0.25	0.25
	14	1.00	0.25	0.25
	15	1.00	0.75	0.25
	16	1.00	1.00	0.25
	17	1.00	1.00	0.25
	18	0.25	0.40	0.25
	19	1.00	0.75	0.25
	20	0.25	0.25	0.25
	21	1.00	1.00	0.25
	22	1.00	1.00	0.25
	23	1.00	1.00	0.25
	24	1.00	0.75	0.25
	Average	**0.826**	**0.708**	**0.25**

Table 18: Detailed per-trial performance for *Put bowl in the sink* for Single arm tasks. Each row shows the task progress score for a specific trial.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Wipe Table	0	1.00	1.00	0.25
	1	1.00	0.50	0.25
	2	1.00	1.00	0.25
	3	1.00	1.00	0.25
	4	1.00	1.00	0.25
	5	1.00	1.00	0.25
	6	1.00	0.25	0.25
	7	1.00	1.00	0.25
	8	1.00	1.00	0.25
	9	1.00	1.00	0.25
	10	1.00	1.00	0.25
	11	1.00	1.00	0.25
	12	1.00	1.00	0.25
	13	1.00	0.50	0.25
	14	1.00	1.00	0.25
	15	1.00	1.00	0.25
	16	1.00	1.00	0.25
	17	1.00	1.00	0.25
	18	1.00	1.00	0.25
	19	1.00	1.00	1.00
	20	1.00	1.00	0.25
	21	1.00	1.00	0.25
	22	1.00	1.00	0.25
	23	1.00	1.00	0.25
	24	1.00	1.00	0.25
	Average	**1.000**	**0.817**	**0.265**

Table 19: Detailed per-trial performance for *Wipe Table* for Single arm tasks. Each row shows the task progress score for a specific trial.

Task	Trial	MolmoAct	π_0-FAST	OpenVLA
Clean the table	0	1.00	1.00	0.50
	1	1.00	1.00	1.00
	2	0.50	1.00	0.50
	3	1.00	0.25	0.50
	4	1.00	0.75	0.00
	5	0.50	1.00	0.75
	6	1.00	0.75	0.50
	7	1.00	0.50	0.50
	8	1.00	1.00	0.50
	9	1.00	1.00	0.75
	10	1.00	1.00	0.50
	11	1.00	1.00	0.75
	12	0.50	0.25	0.25
	13	1.00	1.00	0.50
	14	0.50	1.00	1.00
	15	0.50	1.00	0.75
	16	1.00	0.25	0.75
	17	1.00	1.00	0.25
	18	1.00	1.00	0.25
	19	0.50	0.50	0.25
	20	0.50	1.00	0.25
	21	1.00	1.00	1.00
	22	1.00	1.00	0.25
	23	1.00	1.00	0.25
	24	0.50	1.00	0.75
	Average	**0.84**	**0.85**	**0.53**

Table 20: Detailed per-trial performance for *Clean the table* for Single arm tasks. Each row shows the task progress score for a specific trial.

Category	Task	OpenVLA	π_0-FAST	MolmoAct
In Distribution	put the green can into the yellow plate	0.375	0.8125	1.0
In Distribution	put the red cup into the yellow plate	0.5	0.5	0.625
In Distribution	put the banana into the blue plate	0.25	0.625	0.75
Language Variation	put the green tea into the yellow plate	0.375	0.8125	0.625
Language Variatiion	put the fruit into the blue plate	0.0	0.0625	0.625
Language Variatiion	put the red cylinder into the yellow plate	0.3125	0.0	0.75
Spatial Variation	put the green can into the yellow plate	0.4375	0.5625	0.625
Spatial Variation	put the red cup into the yellow plate	0.5	0.375	0.4375
Spatial Variation	put the banana into the blue plate	0.25	0.4375	0.5625
Distractor (Coke Can, Sponge)	put the green can into the yellow plate	0.125	0.875	0.9375
Distractor (Coke Can, Sponge)	put the red cup into the yellow plate	0.5	0.3125	0.6875
Distractor (Coke Can, Sponge)	put the banana into the blue plate	0.25	0.4375	0.625
Novel Object	put the sponge into the yellow plate	0.25	0.0	0.875
Novel Object	put the coke can into the yellow plate	0.375	0.5625	0.625
Novel Object	put the bowl into the yellow plate	0.25	0.3125	0.4375

Table 21: Detailed results of real-world evaluation. The first column indicates the variation category while the second column presents the language instruction. For each task, the detailed task progress score used to evaluate each model are detailed at section G.4

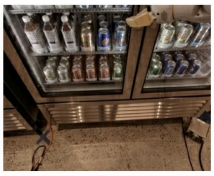
Task	Trial	MolmoAct	MolmoAct (W/o MolmoAct Data)	π_0-FAST	OpenVLA
Pour Tea	0	0.8	0.5	0.5	1.0
	1	0.8	0.8	0.0	0.5
	2	0.5	0.5	0.0	0.0
	3	1.0	1.0	0.0	0.5
	4	1.0	1.0	0.8	0.5
	5	0.8	0.5	1.0	0.5
	6	1.0	1.0	1.0	0.5
	7	0.5	0.5	1.0	0.0
	8	0.5	1.0	0.0	0.0
	9	1.0	0.5	0.0	0.5
	10	0.8	0.8	0.0	0.5
Close Lid	0	0.5	0.0	0.5	0.0
	1	0.5	0.5	0.0	0.5
	2	0.5	0.0	0.5	1.0
	3	0.5	0.5	0.5	0.5
	4	0.5	0.0	1.0	0.0
	5	1.0	0.5	0.5	0.0
	6	0.5	0.0	0.0	0.5
	7	0.5	1.0	0.0	0.0
	8	0.5	1.0	1.0	0.0
	9	0.0	1.0	0.5	0.5
	10	0.5	0.0	0.5	0.5
Rotate Pot	0	1.0	1.0	0.6	1.0
	1	0.6	1.0	1.0	1.0
	2	1.0	1.0	0.0	1.0
	3	1.0	1.0	1.0	1.0
	4	1.0	1.0	0.6	1.0
	5	1.0	1.0	1.0	1.0
	6	1.0	1.0	0.6	1.0
	7	0.6	1.0	1.0	1.0
	8	1.0	1.0	1.0	1.0
	9	1.0	1.0	1.0	1.0
	10	1.0	0.0	0.6	1.0

Table 22: Detailed per-trial performance for three tasks (*Pour Tea*, *Close Lid*, and *Rotate Pot*). Each row shows the task progress score for a specific trial.

Task	Task Detail	Episode	Open instruction (MolmoAct)	Open instruction (π_0-FAST)	Visual Trace (MolmoAct)
pick up the orange bowl	steer from dirty to clean	0	0.00	0.50	1.00
lift up the dirty bowl	steer from clean to dirty	1	1.00	0.00	1.00
pick up the bowl on the left	steer from clean to dirty	2	0.00	0.00	1.00
pick up the empty bowl	steer from dirty to clean	3	0.85	0.50	0.85
pick up the dirty container	steer from clean to dirty	4	0.50	0.00	1.00
pick up the bowl with object inside	steer from clean to dirty	5	0.00	0.00	0.50
pick up the left bowl	steer from clean to dirty	6	0.00	0.50	0.50
pick up the bowl that is pink	steer from clean to dirty	7	0.00	0.00	0.00
pick up the bowl that is pink	steer from clean to dirty	8	0.50	0.00	1.00
pick up the bowl further	steer from dirty to clean	9	0.85	0.50	0.85
pick up the bowl nearer to the camera	steer from dirty to clean	10	0.50	0.00	1.00
pick up the right bowl	steer from dirty to clean	11	0.50	0.00	0.50
pick up the bowl without tissue	steer from clean to dirty	12	0.50	0.00	0.50
pick up the bowl with tissue	steer from clean to dirty	13	0.00	0.00	0.50
pick up the bowl that is dirty	steer from clean to dirty	14	1.00	0.00	1.00

Table 23: Per-episode evaluation results for bowl-picking tasks with different steering conditions. Scores indicate task progression for each model configuration.

The task is open left fridge door. What is the depth map for the image?



The depth map of the image is
<DEPTH_START><DEPTH_75><DEPTH_101><DEPTH_101><DEPTH_75><DEPTH_101><DEPTH_32><DEPTH_101><DEPTH_26><DEPTH_26><DEPTH_26><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_69><DEPTH_9><DEPTH_69><DEPTH_75><DEPTH_75><DEPTH_56><DEPTH_101><DEPTH_101><DEPTH_56><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_101><DEPTH_8><DEPTH_125><DEPTH_9><DEPTH_101><DEPTH_8><DEPTH_69><DEPTH_9><DEPTH_69><DEPTH_9><DEPTH_56><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_101><DEPTH_69><DEPTH_116><DEPTH_69><DEPTH_28><DEPTH_69><DEPTH_56><DEPTH_25><DEPTH_9><DEPTH_9><DEPTH_56><DEPTH_8><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_69><DEPTH_69><DEPTH_9><DEPTH_69><DEPTH_69><DEPTH_101><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_9><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_25><DEPTH_55><DEPTH_69><DEPTH_25><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_55><DEPTH_28><DEPTH_74><DEPTH_55><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_55><DEPTH_25><DEPTH_8><DEPTH_8><DEPTH_30><DEPTH_30><DEPTH_30><DEPTH_30><DEPTH_30><DEPTH_30><DEPTH_30><DEPTH_84><DEPTH_END>"

The task is move spoon to below the bowl on right. What is the depth map for the image?

The depth map of the image is
<DEPTH_START><DEPTH_101><DEPTH_32><DEPTH_8><DEPTH_69><DEPTH_8><DEPTH_56><DEPTH_32><DEPTH_30><DEPTH_32><DEPTH_101><DEPTH_101><DEPTH_32><DEPTH_8><DEPTH_116><DEPTH_32><DEPTH_101><DEPTH_32><DEPTH_32><DEPTH_69><DEPTH_8><DEPTH_101><DEPTH_69><DEPTH_25><DEPTH_69><DEPTH_9><DEPTH_116><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_69><DEPTH_32><DEPTH_9><DEPTH_69><DEPTH_9><DEPTH_9><DEPTH_69><DEPTH_8><DEPTH_69><DEPTH_116><DEPTH_101><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_69><DEPTH_69><DEPTH_8><DEPTH_8><DEPTH_9><DEPTH_32><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_69><DEPTH_8><DEPTH_8><DEPTH_9><DEPTH_56><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_8><DEPTH_8><DEPTH_69><DEPTH_101><DEPTH_101><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_8><DEPTH_56><DEPTH_32><DEPTH_8><DEPTH_8><DEPTH_75><DEPTH_101><DEPTH_69><DEPTH_25><DEPTH_74><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_56><DEPTH_101><DEPTH_26><DEPTH_26><DEPTH_25><DEPTH_76><DEPTH_76><DEPTH_76><DEPTH_103><DEPTH_69><DEPTH_25><DEPTH_74><DEPTH_9><DEPTH_116><DEPTH_END>

The task is pick banana from white bowl. What is the depth map for the image?

The depth map of the image is
<DEPTH_START><DEPTH_69><DEPTH_116><DEPTH_28><DEPTH_101><DEPTH_56><DEPTH_8><DEPTH_55><DEPTH_103><DEPTH_9><DEPTH_74><DEPTH_8><DEPTH_116><DEPTH_28><DEPTH_56><DEPTH_56><DEPTH_26><DEPTH_32><DEPTH_74><DEPTH_76><DEPTH_74><DEPTH_74><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_8><DEPTH_9><DEPTH_75><DEPTH_101><DEPTH_101><DEPTH_116><DEPTH_55><DEPTH_69><DEPTH_8><DEPTH_116><DEPTH_69><DEPTH_55><DEPTH_28><DEPTH_101><DEPTH_101><DEPTH_56><DEPTH_74><DEPTH_69><DEPTH_69><DEPTH_32><DEPTH_25><DEPTH_8><DEPTH_75><DEPTH_101><DEPTH_75><DEPTH_75><DEPTH_74><DEPTH_8><DEPTH_9><DEPTH_9><DEPTH_116><DEPTH_26><DEPTH_75><DEPTH_101><DEPTH_26><DEPTH_8><DEPTH_55><DEPTH_32><DEPTH_8><DEPTH_25><DEPTH_8><DEPTH_26><DEPTH_75><DEPTH_26><DEPTH_56><DEPTH_28><DEPTH_55><DEPTH_32><DEPTH_69><DEPTH_69><DEPTH_116><DEPTH_26><DEPTH_26><DEPTH_75><DEPTH_30><DEPTH_28><DEPTH_55><DEPTH_101><DEPTH_8><DEPTH_8><DEPTH_55><DEPTH_75><DEPTH_26><DEPTH_26><DEPTH_8><DEPTH_28><DEPTH_28><DEPTH_103><DEPTH_103><DEPTH_76><DEPTH_76><DEPTH_69><DEPTH_55><DEPTH_55><DEPTH_84><DEPTH_103><DEPTH_76><DEPTH_END>"

Figure 15: Randomly selected examples from **Auxiliary Depth Perception Tokens** data used in the pre-training stage.

Scene	Task	Language Instruction	Object(s)
Kitchen	put_fork_sink	put the fork in the sink	Fork (2 types)
Kitchen	put_spoon_sink	put the spoon in the sink	Spoon (2 types)
Kitchen	put_bowl_sink	put the bowl in the sink	Bowl (2 types)
Kitchen	clean_spill	Clean the spill	Sponge
Kitchen	wipe_counter	Wipe the counter	Towels
Kitchen	put_plate_in_dishwasher	Put the plate in the dishwasher	Plate
Kitchen	put_fork_in_dishwasher	put the fork in the dishwasher	Fork
Kitchen	put_spoon_in_dishwasher	put the spoon in the dishwasher	Spoon
Kitchen	uncover_food_container	Uncover the lid of the food container	Large Container
Kitchen	uncover_container_lid	Uncover the lid of the food container	Small Container
Kitchen	put tongs in the holder	Put the tongs back in the holder	Tongs
Kitchen	press_toaster	Turn on the toaster	Toaster
Kitchen	close_the_microwave	Close the microwave	Microwave
Kitchen	put_spoon_into_plate	Put the spoon on the plate	Spoon
Kitchen	put_fork_into_plate	Put the fork on the plate	Fork
Kitchen	put_apple_into_container	Put apple in the food container	Apple(red and green)
Kitchen	put_cereal_into_container	Put the cereal in the food container	Cereal(2 types)
Kitchen	put_protein_bar_into_container	Put the protein bar in the food container	Protein Bar(2-3 types)
Kitchen	put_chips_into_container	Put the chip bag in the food container	chip bag(2-3 types)
Kitchen	turn_off_light_kitchen	Turn off the light	Light switch
Kitchen	close_drawer	close the drawer	Drawer
Kitchen	turn_on_faucet	Turn on the faucet	Faucet
Kitchen	close_oven	Close the oven	Oven
Kitchen	open_the_oven	Open the oven	Oven
Kitchen	turn_on_stove	Turn on the Stove	Stove
Kitchen	turn_off_stove	Turn off the Stove	Stove
Kitchen	unload_the_dishwasher_mug	Unload the mug from the dish wisher	Mugs
Kitchen	put_snacks_in_container	Put the Snacks in the Containers	Snacks
Bedroom	hang_the_cap	hang cap	Cap
Bathroom	wipe_sink_bathroom	Wipe the sink	towels(gray and brown towels)
Bathroom	press_hand_sanitizer	Press sanitizer	sanitizer(high and low)
Bathroom	clean_toilet	Clean the toilet	Toliet brush
Bathroom	turn_on_hot_water	Turn on hot water	Faucet
Bathroom	turn_on_cold_water	Turn on the cold water	Faucet
Bathroom	turn_off_hot_water	Turn off the hot water	Faucet
Bathroom	turn_off_cold_water	Turn off the cold water	Faucet
Bathroom	throw_tissue_bathroom_left	Throw the tissue	Tissue
Bathroom	throw_tissue_bathroom_right	Throw the tissue	Tissue
Bathroom	flush_toilet	Flush the toliet	Toliet brush
Bedroom	put_markers_hack_holder	Put the markers back in the holder	Pen holder 1(shape)
Bedroom	put_markers_hack_holder	Put the markers back in the holder	Pen holder 2(shape)
Bedroom	hang_headphone	hand the headphone	Headphone
Bedroom	throw_bottle_bedroom	Throw the water bottle in the trash bin	Bottle
Bedroom	throw_can_bedroom	Throw the can in the trash bin	Can(2 types)
Bedroom	close_laptop_lid_bedroom	Close the laptop lid	Laptop
Livingroom	throw_can_livingroom	Throw the can in the trash bin	Can(2 types)
Livingroom	throw_plastic_bottle_livingroom	Throw the plastic bottle in the trash bin	Bottle
Livingroom	throw_chip_bag_livingroom	Throw the chip bag in the trash bin	Chip bag(2-3 types)
Livingroom	throw_tissue_livingroom	Throw the tissue in the trash bin	Tissue
Livingroom	put_apple_tray_livingroom	Put the apple in the tray	Apple
Livingroom	put_tangerine_livingroom	Put the tangerine in the tray	Tangerine
Livingroom	put_banana_tray_livingroom	Put the banana in the tray	Banana
Livingroom	arrange_pillow	Arrange pillows	Pillow
Livingroom	shelf_book	Shelf books	Books

Table 24: Tasks details of MOLMOACT DATASET Home Environemnt including scene, task name, language instruction and all objects used for data collection.

Scene	Task	Language Instruction	Object(s)
Tabletop	`stand_water_bottle`	stand the water bottle	Water bottle(2 types)
Tabletop	`flip_mug`	flip mug	Mug (2 colors)
Tabletop	`close_top_drawer`	close the top drawer	Drawer
Tabletop	`close_box`	close the box	Box
Tabletop	`close_laptop`	close the laptop	Laptop
Tabletop	`knock_water_bottle`	Knock water bottle	Bottle
Tabletop	`stand_sanitizer`	Stand sanitizer	Sanitizer
Tabletop	`knock_sanitizer`	Knock Sanitizer	Sanitizer
Tabletop	`knock_dish_soap`	Knock dish soap	Dish Soap
Tabletop	`fold_towel`	Fold Towel	Towel
Tabletop	`unfold_towel`	Unfold Towel	Towel
Tabletop	`fold_shorts`	fold shorts	Shorts
Tabletop	`unfold_shorts`	Unfold shorts	Shorts

Table 25: Tasks details of MOLMOACT DATASET Tabletop Environemnt including scene, task name, language instruction and all objects used for data collection.

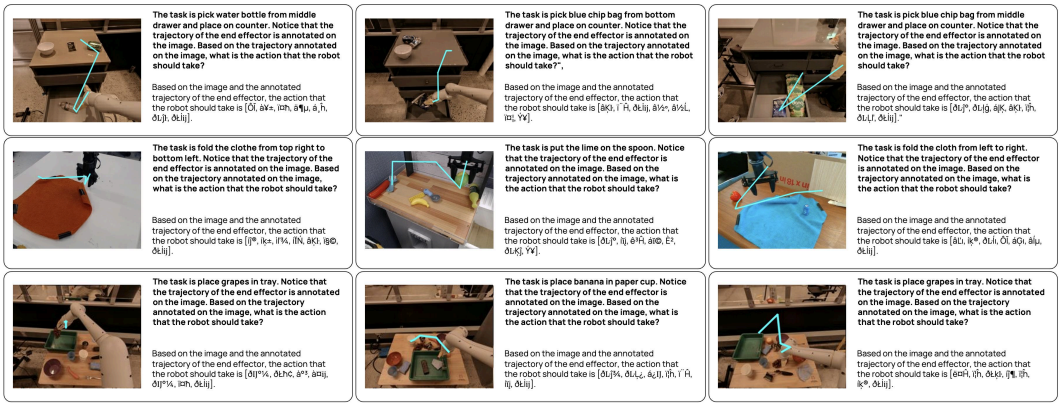


Figure 16: Randomly selected examples from **Trajectory-conditioned Action Data** used in the pre-training stage.

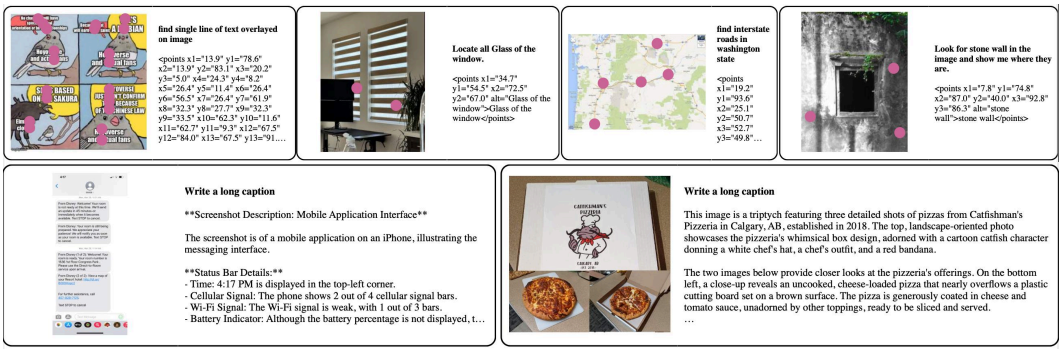


Figure 17: Randomly selected examples from **Multimodal Web Data** used in the pre-training stage.

The task is put the plate in the dishwasher. What is the action that the robot should take. To figure out the action that the robot should take to put the plate in the dishwasher, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the first image is
<DEPTH_START><DEPTH_26><DEPTH_56><DEPTH_75><DEPTH_101><DEPTH_69><DEPTH_55><DEPTH_25><DEPTH_69><DEPTH_8><DEPTH_32><DEPTH_101><DEPTH_125><DEPTH_26><DEPTH_75><DEPTH_25><DEPTH_55><DEPTH_25><DEPTH_9><DEPTH_69><DEPTH_32><DEPTH_26><DEPTH_74><DEPTH_116><DEPTH_84><DEPTH_75><DEPTH_74><DEPTH_116><DEPTH_116><DEPTH_9><DEPTH_69><DEPTH_26><DEPTH_116><DEPTH_74><DEPTH_9><DEPTH_125><DEPTH_74><DEPTH_25><DEPTH_116><DEPTH_9><DEPTH_69><DEPTH_26><DEPTH_84><DEPTH_28><DEPTH_25><DEPTH_8><DEPTH_25><DEPTH_116><DEPTH_25><DEPTH_9><DEPTH_8><DEPTH_101><DEPTH_55><DEPTH_103><DEPTH_25><DEPTH_74><DEPTH_56><DEPTH_9><DEPTH_116><DEPTH_9><DEPTH_32><DEPTH_75><DEPTH_116><DEPTH_9><DEPTH_74><DEPTH_28><DEPTH_32><DEPTH_8><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_75><DEPTH_26><DEPTH_32><DEPTH_55><DEPTH_74><DEPTH_8><DEPTH_56><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_75><DEPTH_26><DEPTH_8><DEPTH_74><DEPTH_74><DEPTH_9><DEPTH_56><DEPTH_101><DEPTH_56><DEPTH_56><DEPTH_69><DEPTH_8><DEPTH_84><DEPTH_76><DEPTH_103><DEPTH_30><DEPTH_69><DEPTH_55><DEPTH_25><DEPTH_116><DEPTH_END>. The trajectory of the end effector in the first image is [[108,59],[81,89],[76,111],[76,125],[78,71]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [â½Ł, âĺĨ, â½Ł, ðŁķ±, âĸ¤, ðŁĶĠ, Ý¥].

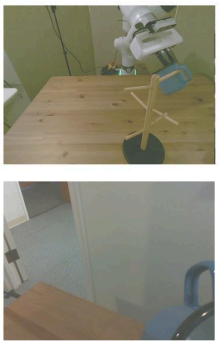
The task is clean the toilet. What is the action that the robot should take. To figure out the action that the robot should take to clean the toilet, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the first image is
<DEPTH_START><DEPTH_56><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_8><DEPTH_69><DEPTH_32><DEPTH_56><DEPTH_101><DEPTH_26><DEPTH_56><DEPTH_32><DEPTH_8><DEPTH_69><DEPTH_116><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_101><DEPTH_26><DEPTH_56><DEPTH_8><DEPTH_30><DEPTH_76><DEPTH_55><DEPTH_9><DEPTH_8><DEPTH_32><DEPTH_101><DEPTH_75><DEPTH_9><DEPTH_25><DEPTH_28><DEPTH_74><DEPTH_84><DEPTH_125><DEPTH_69><DEPTH_32><DEPTH_56><DEPTH_75><DEPTH_84><DEPTH_75><DEPTH_32><DEPTH_101><DEPTH_116><DEPTH_30><DEPTH_25><DEPTH_8><DEPTH_32><DEPTH_101><DEPTH_26><DEPTH_26><DEPTH_26><DEPTH_56><DEPTH_116><DEPTH_30><DEPTH_25><DEPTH_8><DEPTH_32><DEPTH_101><DEPTH_26><DEPTH_125><DEPTH_75><DEPTH_55><DEPTH_32><DEPTH_69><DEPTH_74><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_75><DEPTH_74><DEPTH_74><DEPTH_9><DEPTH_25><DEPTH_55><DEPTH_55><DEPTH_9><DEPTH_8><DEPTH_56><DEPTH_84><DEPTH_84><DEPTH_74><DEPTH_74><DEPTH_55><DEPTH_74><DEPTH_74><DEPTH_116><DEPTH_69><DEPTH_56><DEPTH_74><DEPTH_69><DEPTH_9><DEPTH_76><DEPTH_30><DEPTH_84><DEPTH_103><DEPTH_84><DEPTH_30><DEPTH_30><DEPTH_END>. The trajectory of the end effector in the first image is [[136,170],[140,137],[128,204],[128,204],[128,204]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [á©ï, ï¥ł, ðĽíĻ, ïîj, íĥĻ, áǵ¼, Ý¥].

The task is lift the pillow. What is the action that the robot should take. To figure out the action that the robot should take to lift the pillow, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

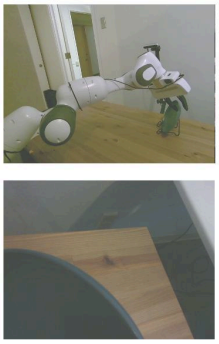
The depth map of the first image is
<DEPTH_START><DEPTH_101><DEPTH_32><DEPTH_32><DEPTH_8><DEPTH_76><DEPTH_76><DEPTH_125><DEPTH_55><DEPTH_69><DEPTH_69><DEPTH_101><DEPTH_32><DEPTH_103><DEPTH_116><DEPTH_25><DEPTH_74><DEPTH_84><DEPTH_74><DEPTH_9><DEPTH_69><DEPTH_56><DEPTH_8><DEPTH_9><DEPTH_28><DEPTH_75><DEPTH_103><DEPTH_28><DEPTH_76><DEPTH_74><DEPTH_28><DEPTH_56><DEPTH_116><DEPTH_56><DEPTH_26><DEPTH_75><DEPTH_26><DEPTH_26><DEPTH_125><DEPTH_74><DEPTH_28><DEPTH_32><DEPTH_28><DEPTH_26><DEPTH_30><DEPTH_116><DEPTH_9><DEPTH_101><DEPTH_84><DEPTH_28><DEPTH_69><DEPTH_32><DEPTH_32><DEPTH_101><DEPTH_28><DEPTH_9><DEPTH_55><DEPTH_8><DEPTH_26><DEPTH_55><DEPTH_9><DEPTH_101><DEPTH_32><DEPTH_9><DEPTH_8><DEPTH_28><DEPTH_116><DEPTH_32><DEPTH_101><DEPTH_28><DEPTH_25><DEPTH_56><DEPTH_69><DEPTH_69><DEPTH_116><DEPTH_69><DEPTH_8><DEPTH_101><DEPTH_8><DEPTH_26><DEPTH_75><DEPTH_26><DEPTH_84><DEPTH_69><DEPTH_8><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_26><DEPTH_26><DEPTH_74><DEPTH_25><DEPTH_25><DEPTH_8><DEPTH_76><DEPTH_32><DEPTH_28><DEPTH_8><DEPTH_9><DEPTH_116><DEPTH_END>. The trajectory of the end effector in the first image is [[68,113],[58,113],[59,105],[71,95],[97,106]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [Êj, à²Ł, áĶĨ, ðŁıĩ, ðIJ°¼, áĢı, ðŁlij].

Figure 18: Randomly selected examples from **MOLMOACT DATASET (Home Environment)** used in the mid-training stage.



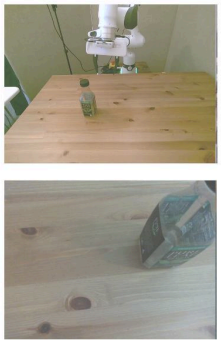
The task is unload the mug. What is the action that the robot should take. To figure out the action that the robot should take to unload the mug, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?",

The depth map of the first image is
<DEPTH_START><DEPTH_69><DEPTH_9><DEPTH_8><DEPTH_28><DEPTH_8><DEPTH_8><DEPTH_101><DEPTH_32><DEPTH_84><DEPTH_8><DEPTH_69><DEPTH_116><DEPTH_116><DEPTH_25><DEPTH_28><DEPTH_32><DEPTH_101><DEPTH_56><DEPTH_69><DEPTH_32><DEPTH_25><DEPTH_116><DEPTH_116><DEPTH_25><DEPTH_116><DEPTH_30><DEPTH_101><DEPTH_75><DEPTH_103><DEPTH_69><DEPTH_116><DEPTH_28><DEPTH_55><DEPTH_74><DEPTH_25><DEPTH_32><DEPTH_75><DEPTH_32><DEPTH_28><DEPTH_69><DEPTH_28><DEPTH_74><DEPTH_74><DEPTH_55><DEPTH_55><DEPTH_9><DEPTH_74><DEPTH_75><DEPTH_56><DEPTH_69><DEPTH_25><DEPTH_28><DEPTH_56><DEPTH_69><DEPTH_69><DEPTH_8><DEPTH_101><DEPTH_26><DEPTH_56><DEPTH_9><DEPTH_55><DEPTH_101><DEPTH_9><DEPTH_8><DEPTH_25><DEPTH_8><DEPTH_56><DEPTH_116><DEPTH_9><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_69><DEPTH_8><DEPTH_56><DEPTH_74><DEPTH_101><DEPTH_101><DEPTH_8><DEPTH_32><DEPTH_26><DEPTH_8><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_56><DEPTH_56><DEPTH_8><DEPTH_32><DEPTH_32><DEPTH_25><DEPTH_116><DEPTH_28><DEPTH_28><DEPTH_25><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_116><DEPTH_9><DEPTH_END>. The trajectory of the end effector in the first image is [[179,84],[207,100],[143,120],[140,143],[180,111]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [áJ], Ýj, áJ], Õij, ðIJº¼, ðŁj̃ł, Ý¥]."



The task is unload the plate. What is the action that the robot should take. To figure out the action that the robot should take to unload the plate, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the first image is
<DEPTH_START><DEPTH_101><DEPTH_103><DEPTH_56><DEPTH_74><DEPTH_30><DEPTH_84><DEPTH_32><DEPTH_32><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_25><DEPTH_9><DEPTH_125><DEPTH_69><DEPTH_9><DEPTH_8><DEPTH_32><DEPTH_32><DEPTH_56><DEPTH_84><DEPTH_30><DEPTH_125><DEPTH_74><DEPTH_9><DEPTH_75><DEPTH_30><DEPTH_69><DEPTH_8><DEPTH_32><DEPTH_26><DEPTH_26><DEPTH_26><DEPTH_101><DEPTH_26><DEPTH_101><DEPTH_125><DEPTH_69><DEPTH_69><DEPTH_32><DEPTH_26><DEPTH_26><DEPTH_101><DEPTH_74><DEPTH_25><DEPTH_101><DEPTH_125><DEPTH_8><DEPTH_69><DEPTH_32><DEPTH_26><DEPTH_28><DEPTH_25><DEPTH_55><DEPTH_25><DEPTH_75><DEPTH_30><DEPTH_69><DEPTH_8><DEPTH_32><DEPTH_26><DEPTH_9><DEPTH_74><DEPTH_74><DEPTH_55><DEPTH_69><DEPTH_28><DEPTH_69><DEPTH_116><DEPTH_8><DEPTH_9><DEPTH_8><DEPTH_25><DEPTH_32><DEPTH_69><DEPTH_75><DEPTH_8><DEPTH_69><DEPTH_9><DEPTH_9><DEPTH_101><DEPTH_56><DEPTH_101><DEPTH_56><DEPTH_32><DEPTH_56><DEPTH_101><DEPTH_32><DEPTH_101><DEPTH_56><DEPTH_32><DEPTH_28><DEPTH_28><DEPTH_28><DEPTH_28><DEPTH_25><DEPTH_25><DEPTH_28><DEPTH_28><DEPTH_116><DEPTH_END>. The trajectory of the end effector in the first image is [[148,150],[153,162],[153,165],[124,153],[124,172]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [áJ], âĶĺ, â²¢, ïŸ±, ì»ğ, áĵĶ, ðŁlij]."



The task is knock down the bottle. What is the action that the robot should take. To figure out the action that the robot should take to knock down the bottle, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the first image is
<DEPTH_START><DEPTH_8><DEPTH_69><DEPTH_69><DEPTH_69><DEPTH_32><DEPTH_101><DEPTH_101><DEPTH_25><DEPTH_69><DEPTH_56><DEPTH_69><DEPTH_9><DEPTH_116><DEPTH_74><DEPTH_75><DEPTH_56><DEPTH_69><DEPTH_28><DEPTH_69><DEPTH_32><DEPTH_116><DEPTH_25><DEPTH_116><DEPTH_9><DEPTH_75><DEPTH_125><DEPTH_32><DEPTH_28><DEPTH_9><DEPTH_32><DEPTH_125><DEPTH_74><DEPTH_74><DEPTH_69><DEPTH_101><DEPTH_56><DEPTH_28><DEPTH_74><DEPTH_9><DEPTH_8><DEPTH_25><DEPTH_74><DEPTH_55><DEPTH_74><DEPTH_55><DEPTH_9><DEPTH_74><DEPTH_55><DEPTH_116><DEPTH_8><DEPTH_69><DEPTH_55><DEPTH_56><DEPTH_32><DEPTH_125><DEPTH_69><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_74><DEPTH_75><DEPTH_9><DEPTH_101><DEPTH_32><DEPTH_69><DEPTH_8><DEPTH_8><DEPTH_8><DEPTH_32><DEPTH_56><DEPTH_75><DEPTH_32><DEPTH_56><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_56><DEPTH_32><DEPTH_32><DEPTH_26><DEPTH_101><DEPTH_26><DEPTH_75><DEPTH_75><DEPTH_75><DEPTH_75><DEPTH_75><DEPTH_75><DEPTH_75><DEPTH_74><DEPTH_55><DEPTH_55><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_74><DEPTH_28><DEPTH_END>. The trajectory of the end effector in the first image is [[125,68],[120,79],[126,84],[100,127],[82,150]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [ðŁhĊ, áJ], áĶĬ, Ýj, ðŁĻĺ', àĊij, ðŁlij].

Figure 19: Randomly selected examples from **MOLMOACT DATASET (Tabletop Environment)** used in the mid-training stage.

The task is clean the table. What is the action that the robot should take. To figure out the action that the robot should take to clean the table, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_74><DEPTH_61><DEPTH_104><DEPTH_86><DEPTH_70><DEPTH_41><DEPTH_102><DEPTH_102><DEPTH_90><DEPTH_98><DEPTH_96><DEPTH_32><DEPTH_13><DEPTH_50><DEPTH_9><DEPTH_93><DEPTH_71><DEPTH_91><DEPTH_102><DEPTH_60><DEPTH_8><DEPTH_85><DEPTH_17><DEPTH_13><DEPTH_80><DEPTH_63><DEPTH_70><DEPTH_3><DEPTH_23><DEPTH_34><DEPTH_20><DEPTH_54><DEPTH_49><DEPTH_21><DEPTH_3><DEPTH_27><DEPTH_71><DEPTH_18><DEPTH_23><DEPTH_44><DEPTH_8><DEPTH_95><DEPTH_62><DEPTH_49><DEPTH_13><DEPTH_86><DEPTH_71><DEPTH_43><DEPTH_23><DEPTH_44><DEPTH_32><DEPTH_61><DEPTH_56><DEPTH_77><DEPTH_124><DEPTH_51><DEPTH_68><DEPTH_44><DEPTH_76><DEPTH_31><DEPTH_104><DEPTH_110><DEPTH_7><DEPTH_122><DEPTH_98><DEPTH_33><DEPTH_16><DEPTH_123><DEPTH_70><DEPTH_102><DEPTH_39><DEPTH_15><DEPTH_80><DEPTH_122><DEPTH_57><DEPTH_10><DEPTH_15><DEPTH_69><DEPTH_90><DEPTH_31><DEPTH_32><DEPTH_9><DEPTH_90><DEPTH_10><DEPTH_127><DEPTH_31><DEPTH_99><DEPTH_14><DEPTH_75><DEPTH_60><DEPTH_2><DEPTH_61><DEPTH_122><DEPTH_78><DEPTH_30><DEPTH_93><DEPTH_117><DEPTH_123><DEPTH_30><DEPTH_50><DEPTH_END>. The trajectory of the end effector is [[153,200],[153,204],[153,188],[140,135],[108,77]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[âłĺĤ, ðŁĵº, ãĺª, Ýł¥, Ê², ájłĦ, Ýł¥], [âłĹ, ðŁĻľ, ãĺª, àł¥±, ĺĵĸ, ájłĦ, Ýł¥], [á¶ı, ðŁĥĉ, â²ĉ, ðŁĥĉ, ðŁĥĉ, ðŁĥĉ, áǵ¼, Ýł¥], [áłĦĤ, ðŁıǵ, ãĺª, ðŁĵĺ, Ê², Ýł¥, ðŁĺij], [áłĥ, ðŁĻľ, âłĢ, ðIJ¤Ķ, ðŁĥĉ, àł¥±, ðŁĺij], [áłĨ, ðŁĵº, âłĢ, ïłĥ, iŁľ, à°³, ðŁĺij], [ðŁķï, ðŁĵĺ, âĶł, iŁĉ, ï¨Ĥ, à°³, ðŁĺij], [Ôĥ, ðŁĵĺ, á¸», iŁĉ, ïłĥ, à²Ł, ðŁĺij]].

The task is wipe the table. What is the action that the robot should take. To figure out the action that the robot should take to wipe the table, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

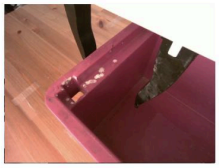
The depth map of the image is
<DEPTH_START><DEPTH_85><DEPTH_45><DEPTH_116><DEPTH_52><DEPTH_7><DEPTH_82><DEPTH_121><DEPTH_78><DEPTH_90><DEPTH_40><DEPTH_17><DEPTH_112><DEPTH_27><DEPTH_19><DEPTH_46><DEPTH_41><DEPTH_68><DEPTH_91><DEPTH_126><DEPTH_35><DEPTH_2><DEPTH_4><DEPTH_19><DEPTH_92><DEPTH_68><DEPTH_93><DEPTH_107><DEPTH_11><DEPTH_59><DEPTH_98><DEPTH_18><DEPTH_95><DEPTH_1><DEPTH_118><DEPTH_53><DEPTH_15><DEPTH_69><DEPTH_42><DEPTH_59><DEPTH_58><DEPTH_27><DEPTH_30><DEPTH_40><DEPTH_14><DEPTH_113><DEPTH_69><DEPTH_91><DEPTH_42><DEPTH_76><DEPTH_58><DEPTH_45><DEPTH_4><DEPTH_79><DEPTH_124><DEPTH_113><DEPTH_70><DEPTH_83><DEPTH_58><DEPTH_59><DEPTH_37><DEPTH_32><DEPTH_51><DEPTH_82><DEPTH_121><DEPTH_93><DEPTH_41><DEPTH_102><DEPTH_59><DEPTH_110><DEPTH_60><DEPTH_32><DEPTH_27><DEPTH_35><DEPTH_70><DEPTH_122><DEPTH_115><DEPTH_41><DEPTH_126><DEPTH_60><DEPTH_37><DEPTH_3><DEPTH_74><DEPTH_36><DEPTH_109><DEPTH_19><DEPTH_53><DEPTH_7><DEPTH_94><DEPTH_103><DEPTH_98><DEPTH_30><DEPTH_94><DEPTH_15><DEPTH_87><DEPTH_85><DEPTH_102><DEPTH_31><DEPTH_81><DEPTH_17><DEPTH_28><DEPTH_END>. The trajectory of the end effector is [[94,131],[101,128],[88,132],[95,156],[79,91 Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[àł¥±, ãĺī, ákĶĺ, âłĢ, i³ħ, ðŁĵª, Ýł¥], [à¬ī, âĺī, áłĦ, âłĢ, i³ħ, ðŁıj³, Ýł¥], [à¬ī, âĺī, áłĦ, â²ĉ, i³ħ, ðŁıj³, Ýł¥], [à¬ī, âĺī, áłĦ, â²ĉ, i³ħ, ðŁĵķ, Ýł¥], [à¬ī, âĺī, áłĦ, âłµ, i³ħ, ðκ¥¨, Ýł¥], [à¬ī, ãĺī, áłĦ, â²ĉ, i³ħ, ĺĵ», Ýł¥], [à¬ī, ãĺī, áłĦ, ãĺª, i³ħ, ê³Ĥ, Ýł¥], [àł¥±, âĺī, áłĦ, âłĢ, ĺǵ», ê³Ĥ, Ýł¥]].

The task is put the bowl in the sink. What is the action that the robot should take. To figure out the action that the robot should take to put the bowl in the sink, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_74><DEPTH_61><DEPTH_104><DEPTH_86><DEPTH_70><DEPTH_41><DEPTH_102><DEPTH_102><DEPTH_90><DEPTH_98><DEPTH_96><DEPTH_32><DEPTH_13><DEPTH_50><DEPTH_9><DEPTH_93><DEPTH_71><DEPTH_91><DEPTH_102><DEPTH_60><DEPTH_8><DEPTH_85><DEPTH_17><DEPTH_13><DEPTH_80><DEPTH_63><DEPTH_70><DEPTH_3><DEPTH_23><DEPTH_34><DEPTH_20><DEPTH_54><DEPTH_49><DEPTH_21><DEPTH_3><DEPTH_27><DEPTH_71><DEPTH_18><DEPTH_23><DEPTH_44><DEPTH_8><DEPTH_95><DEPTH_62><DEPTH_49><DEPTH_13><DEPTH_86><DEPTH_71><DEPTH_43><DEPTH_23><DEPTH_44><DEPTH_32><DEPTH_61><DEPTH_56><DEPTH_77><DEPTH_124><DEPTH_51><DEPTH_68><DEPTH_44><DEPTH_76><DEPTH_31><DEPTH_104><DEPTH_110><DEPTH_7><DEPTH_122><DEPTH_98><DEPTH_33><DEPTH_16><DEPTH_123><DEPTH_70><DEPTH_102><DEPTH_39><DEPTH_15><DEPTH_80><DEPTH_122><DEPTH_57><DEPTH_10><DEPTH_15><DEPTH_69><DEPTH_90><DEPTH_31><DEPTH_32><DEPTH_9><DEPTH_90><DEPTH_10><DEPTH_127><DEPTH_31><DEPTH_99><DEPTH_14><DEPTH_75><DEPTH_60><DEPTH_2><DEPTH_61><DEPTH_122><DEPTH_78><DEPTH_30><DEPTH_93><DEPTH_117><DEPTH_123><DEPTH_30><DEPTH_50><DEPTH_END>. The trajectory of the end effector is [[153,200],[153,204],[153,188],[140,135],[108,77]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[âłĺĤ, ðŁĵº, ãĺª, Ýł¥, Ê², ájłĦ, Ýł¥], [âłĹ, ðŁĻľ, ãĺª, àł¥±, ĺĵĸ, ájłĦ, Ýł¥], [á¶ı, ðŁĥĉ, â²ĉ, ðŁĥĉ, ðŁĥĉ, ðŁĥĉ, áǵ¼, Ýł¥], [áłĦĤ, ðŁıǵ, ãĺª, ðŁĵĺ, Ê², Ýł¥, ðŁĺij], [áłĥ, ðŁĻľ, âłĢ, ðIJ¤Ķ, ðŁĥĉ, àł¥±, ðŁĺij], [áłĨ, ðŁĵº, âłĢ, ïłĥ, iŁľ, à°³, ðŁĺij], [ðŁķï, ðŁĵĺ, âĶł, iŁĉ, ï¨Ĥ, à°³, ðŁĺij], [Ôĥ, ðŁĵĺ, á¸», iŁĉ, ïłĥ, à²Ł, ðŁĺij]].

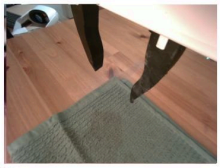
Figure 20: Randomly selected examples from **Single Arm Franka** demonstrations used in the post-training stage.

The task is lift the tray. What is the action that the robot should take. To figure out the action that the robot should take to lift the tray, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?



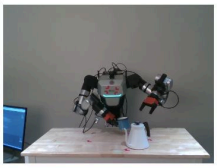
The depth map of the image is
<DEPTH_START><DEPTH_24><DEPTH_88><DEPTH_25><DEPTH_90><DEPTH_91><DEPTH_110><DEPTH_53><DEPTH_80><DEPTH_126><DEPTH_120><DEPTH_118><DEPTH_79><DEPTH_2><DEPTH_115><DEPTH_83><DEPTH_28><DEPTH_41><DEPTH_31><DEPTH_34><DEPTH_40><DEPTH_86><DEPTH_104><DEPTH_32><DEPTH_99><DEPTH_46><DEPTH_96><DEPTH_8><DEPTH_115><DEPTH_36><DEPTH_40><DEPTH_14><DEPTH_65><DEPTH_20><DEPTH_29><DEPTH_24><DEPTH_10><DEPTH_99><DEPTH_72><DEPTH_34><DEPTH_40><DEPTH_84><DEPTH_108><DEPTH_13><DEPTH_26><DEPTH_115><DEPTH_85><DEPTH_17><DEPTH_56><DEPTH_36><DEPTH_40><DEPTH_100><DEPTH_37><DEPTH_2><DEPTH_42><DEPTH_115><DEPTH_94><DEPTH_92><DEPTH_57><DEPTH_124><DEPTH_40><DEPTH_58><DEPTH_100><DEPTH_64><DEPTH_32><DEPTH_100><DEPTH_25><DEPTH_99><DEPTH_107><DEPTH_97><DEPTH_40><DEPTH_58><DEPTH_93><DEPTH_43><DEPTH_89><DEPTH_12><DEPTH_75><DEPTH_31><DEPTH_82><DEPTH_34><DEPTH_120><DEPTH_10><DEPTH_75><DEPTH_44><DEPTH_48><DEPTH_48><DEPTH_36><DEPTH_78><DEPTH_44><DEPTH_121><DEPTH_120><DEPTH_29><DEPTH_120><DEPTH_120><DEPTH_40><DEPTH_10><DEPTH_120><DEPTH_120><DEPTH_120><DEPTH_35><DEPTH_109><DEPTH_END>. The trajectory of the left end effector is [[66,175],[81,153],[78,155],[68,148],[76,127]]. The trajectory of the right end effector is [[153,126],[153,148],[153,167],[148,165],[153,131]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[íÍN, á¨ġ, áłĦ, ï¥†, ðŁĮĶ, ãĮ£, Ý¥, ðŁ·ķı, ðŁ·ĵĶ, ãĮ£, ðŁ·κκ, áĶÑ, á¿Ţ, Ý¥], [áĮ½, áġ¼, ê³Ĥ, ãĪ¼, ãĦ§, ï¨Ĥ, Ý¥, ðŁ·ĵ¬, ðŁ·ķı, ãĮ£, ðŁ·ķj, áłĦ, ãα¦, Ý¥], [ëκº, ðŁïĪ, ïα¦, ãºj, ðŁ·ÍIĮ, ë±IĮ, Ý¥, ðκ¥¨, ðŁ·ĶĵĶĮ, ðκ¥, á¾Ķ, ãα¦, Ý¥], [á¾Ķ, Ê¶, ê³Ĥ, á¥α, ãĶ½, ïĪ¾, Ý¥, ï¥±, ðIĮ±h, áġ¼, á¸h, íĵÑ, ëκº, Ý¥], [ë±ï, ðIĮαĶ, ê²Ţ, ðIĮÑÍ, áġ¼, Ý¥, Ý¥, ðIĮÑÍ, ðŁĵ¾, á¥±, áĶ, ðIĮαĶ, ë±ï, Ý¥], [íķ±, ïĪ¾, áĶh, áÍĴ, áº³, ëş†, Ý¥, ðŁ·ĶĵĶ, ðŁ·Íl, áαij, á¨ġ, ï¨Ĥ, ëş†, Ý¥], [íαĪ, ðŁķĪ, ðIl°¼, íĥĻ, ðŁ·ĵ¬, ãĮş, Ý¥, ðŁ·Íı, ðŁ·Ţ½, áÍĴ, íĵ¶, ì³h, ìĽç, Ý¥], [ï¼ï, ìĹ¤, ãĪ», ãĮ¼, ãĪ¾, Ý¥, ðŁ·ÍIĮ, ðŁ·ÍIĮ, áÇı, ï¼ï, íġ», ãĮĹ, Ý¥]].

The task is fold the towel. What is the action that the robot should take. To figure out the action that the robot should take to fold the towel, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?



The depth map of the image is
<DEPTH_START><DEPTH_27><DEPTH_127><DEPTH_66><DEPTH_70><DEPTH_107><DEPTH_122><DEPTH_93><DEPTH_80><DEPTH_101><DEPTH_17><DEPTH_108><DEPTH_117><DEPTH_13><DEPTH_117><DEPTH_117><DEPTH_12><DEPTH_84><DEPTH_97><DEPTH_105><DEPTH_73><DEPTH_27><DEPTH_26><DEPTH_18><DEPTH_92><DEPTH_55><DEPTH_2><DEPTH_26><DEPTH_59><DEPTH_105><DEPTH_29><DEPTH_61><DEPTH_8><DEPTH_17><DEPTH_46><DEPTH_123><DEPTH_4><DEPTH_32><DEPTH_55><DEPTH_48><DEPTH_73><DEPTH_24><DEPTH_96><DEPTH_89><DEPTH_63><DEPTH_115><DEPTH_41><DEPTH_33><DEPTH_124><DEPTH_101><DEPTH_29><DEPTH_69><DEPTH_13><DEPTH_13><DEPTH_19><DEPTH_23><DEPTH_70><DEPTH_13><DEPTH_57><DEPTH_101><DEPTH_73><DEPTH_75><DEPTH_117><DEPTH_70><DEPTH_89><DEPTH_56><DEPTH_103><DEPTH_3><DEPTH_59><DEPTH_75><DEPTH_73><DEPTH_103><DEPTH_111><DEPTH_58><DEPTH_80><DEPTH_58><DEPTH_37><DEPTH_59><DEPTH_26><DEPTH_80><DEPTH_73><DEPTH_109><DEPTH_35><DEPTH_98><DEPTH_98><DEPTH_58><DEPTH_58><DEPTH_105><DEPTH_34><DEPTH_36><DEPTH_73><DEPTH_20><DEPTH_73><DEPTH_29><DEPTH_73><DEPTH_73><DEPTH_73><DEPTH_30><DEPTH_73><DEPTH_40><DEPTH_73><DEPTH_END>. The trajectory of the left end effector is [[61,143],[51,170],[61,170],[69,135],[105,147]]. The trajectory of the right end effector is [[153,173],[161,174],[155,175],[156,175],[155,172]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[âĴł, ëκº, ðŁĴ¹, ðŁķĮ, ãĪ¼, Õij, Ý¥, ï¼ï, ðŁ·ĵ¬, ðŁĶĵĶ, à¥±, ì»ġ, ðŁ·ĶĵĶ, Ý¥], [âĴł, ëş†, ðŁĴ¹, ðŁķĮ, ãĪ¼, áαij, Ý¥, ê³Ĥ, ðκ¥¨, ðκ¥, á¨ġ, á¿Ţ, ðŁ·ĵĶ, Ý¥], [âĴł, ëκº, ðŁĴ¹, ðŁķĮ, ãĪ¼, Ýĵ, Ý¥, ãĪª, ï¥†, ïÍĪ, Õl, íĵÑ, íĵ¬, Ý¥], [ãĪ¥, ëκº, ðŁĴ¹, ðŁķκ, ï¨·, Ýĵ, Ý¥, áĵĶ, ê²Ţ, Ê², ðŁķ³, ðŁ·Ţ®, áĶÍ, Ý¥], [âĴł, á¿Ţ, ðŁĴ¹, ðŁķκ, ãĪ¼, ðŁ·ĻĪ, Ý¥, ãĪª, ä¶µ, áº³, ë±IĮ, áº³, áĶÍ, Ý¥], [âĴł, âłĦ, ðŁĴ¹, ðŁķκ, ï¨·, ðŁĥĉ, Ý¥, âłµ, ãĪª, áï¨, íķ±, Õij, áłĦ, Ý¥], [âĴł, â¡ç, ðŁĴ¹, ðŁķĮ, ãĪ¼, Ê¶, Ý¥, á¨ġ, ê²Ţ, áĶĪ, Ýĵ, ïαÍ, á¸h, Ý¥], [âĴł, âα¦, ðŁĴ¹, ðŁķκ, ãĪ¼, Õl, Ý¥, áIĮş, ë¡¼, ïÍĪ, áĴĮ, ëş†, â¡£, Ý¥]].

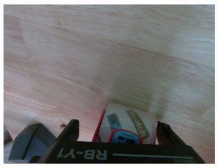
Figure 21: Randomly selected examples from **Bimanual Franka** demonstrations used in the post-training stage.



The task is pour the water into the cup. What is the action that the robot should take. To figure out the action that the robot should take to pour the water into the cup, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?

The depth map of the image is
<DEPTH_START><DEPTH_5><DEPTH_53><DEPTH_57><DEPTH_57><DEPTH_57><DEPTH_15><DEPTH_15><DEPTH_41><DEPTH_41><DEPTH_15><DEPTH_14><DEPTH_46><DEPTH_5><DEPTH_46><DEPTH_46><DEPTH_53><DEPTH_53><DEPTH_57><DEPTH_15><DEPTH_57><DEPTH_14><DEPTH_5><DEPTH_19><DEPTH_53><DEPTH_46><DEPTH_53><DEPTH_57><DEPTH_57><DEPTH_53><DEPTH_53><DEPTH_47><DEPTH_5><DEPTH_46><DEPTH_45><DEPTH_47><DEPTH_92><DEPTH_14><DEPTH_77><DEPTH_15><DEPTH_53><DEPTH_14><DEPTH_19><DEPTH_106><DEPTH_115><DEPTH_37><DEPTH_60><DEPTH_125><DEPTH_44><DEPTH_6><DEPTH_53><DEPTH_9><DEPTH_46><DEPTH_118><DEPTH_7><DEPTH_78><DEPTH_25><DEPTH_102><DEPTH_103><DEPTH_47><DEPTH_53><DEPTH_23><DEPTH_14><DEPTH_24><DEPTH_105><DEPTH_87><DEPTH_4><DEPTH_119><DEPTH_99><DEPTH_113><DEPTH_57><DEPTH_93><DEPTH_74><DEPTH_47><DEPTH_88><DEPTH_6><DEPTH_36><DEPTH_31><DEPTH_74><DEPTH_113><DEPTH_106><DEPTH_27><DEPTH_121><DEPTH_66><DEPTH_110><DEPTH_122><DEPTH_37><DEPTH_32><DEPTH_71><DEPTH_35><DEPTH_86><DEPTH_89><DEPTH_8><DEPTH_2><DEPTH_2><DEPTH_29><DEPTH_29><DEPTH_29><DEPTH_29><DEPTH_17><DEPTH_21><DEPTH_END>. The trajectory of the left end effector is [[126,183],[126,178],[124,182],[124,178],[94,169]]. The trajectory of the right end effector is [[178,157],[186,166],[188,178],[205,131],[188,125]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[Ý¥, áÇh, Ý¥, ðÉhÇ, á¾Ç, Õh, âÏ¨, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, ðLıl, âI£, âI§, âI£, âI§, âÏ¨, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, á¸Ī, ïj®, âI§, ðIJNÎ, ìĹcı, áÏ¨, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, âL«, ïj®, ðÉĶĤ, ê²L, ï¨Ĥ, áÏ¨, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, áï«, ïj®, ðÉĶĤ, ì¼ï, ï¨Ĥ, ìL¿, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, áłĵ, ðIJNÎ, ðÉĶĤ, ê²L, ï¨Ĥ, ë»h, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij], [Ý¥, áï«, ïj®, ðÉĶĤ, ïj», ï¨Ĥ, âĹ¤, Ý¥, Ý¥, à²Ł, àcij, ì³h, ðLj°, ðLj¹, ðLı̧ġ, ðÉlij]]."

The task is clean the table. What is the action that the robot should take. To figure out the action that the robot should take to clean the table, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? Based on the depth map of the first image and the trajectory of the end effector in the first image, along with other images from different camera views as additional information, what is the action that the robot should take?



The depth map of the image is
<DEPTH_START><DEPTH_7><DEPTH_5><DEPTH_5><DEPTH_84><DEPTH_5><DEPTH_84><DEPTH_19><DEPTH_5><DEPTH_47><DEPTH_5><DEPTH_41><DEPTH_19><DEPTH_0><DEPTH_47><DEPTH_47><DEPTH_24><DEPTH_14><DEPTH_0><DEPTH_47><DEPTH_5><DEPTH_69><DEPTH_106><DEPTH_106><DEPTH_65><DEPTH_118><DEPTH_27><DEPTH_62><DEPTH_55><DEPTH_9><DEPTH_115><DEPTH_69><DEPTH_52><DEPTH_62><DEPTH_6><DEPTH_57><DEPTH_48><DEPTH_115><DEPTH_28><DEPTH_114><DEPTH_27><DEPTH_93><DEPTH_88><DEPTH_0><DEPTH_42><DEPTH_10><DEPTH_105><DEPTH_40><DEPTH_103><DEPTH_26><DEPTH_9><DEPTH_69><DEPTH_27><DEPTH_120><DEPTH_21><DEPTH_49><DEPTH_112><DEPTH_60><DEPTH_67><DEPTH_28><DEPTH_94><DEPTH_82><DEPTH_88><DEPTH_124><DEPTH_38><DEPTH_27><DEPTH_50><DEPTH_49><DEPTH_4><DEPTH_16><DEPTH_47><DEPTH_84><DEPTH_62><DEPTH_96><DEPTH_4><DEPTH_11><DEPTH_87><DEPTH_13><DEPTH_67><DEPTH_108><DEPTH_57><DEPTH_79><DEPTH_110><DEPTH_36><DEPTH_39><DEPTH_43><DEPTH_97><DEPTH_75><DEPTH_114><DEPTH_26><DEPTH_21><DEPTH_2><DEPTH_73><DEPTH_28><DEPTH_2><DEPTH_28><DEPTH_29><DEPTH_29><DEPTH_2><DEPTH_2><DEPTH_33><DEPTH_END>. The trajectory of the left end effector is [[78,187],[74,204],[95,125],[123,135],[109,129]]. The trajectory of the right end effector is [[181,129],[78,204],[181,129],[180,125],[181,127]]. Based on these information, along with other images from different camera views as additional information, the action that the robot should take is [[Ý¥, âïı½, áIJ§, ë±IJ, Õĵ, ìķ®, ÆÏ, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áIJ§, ì»ġ, Õĵ, Êĵ, âL«, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áIJ§, ì»ġ, Õĵ, iĹcı, âï«, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áIJ§, ì»ġ, Õĵ, ìķ®, á¥¤, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áłĴ, ì»ġ, Õĵ, ìķ®, âL«, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áłĦ, ì»ġ, Õĵ, ìķ®, âL«, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [Ý¥, âïı½, áIJ§, ìk±, Õĵ, Êĵ, ðLĻ½, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij], [â½Ł, âïı½, âÏ³, ðLı̧ġ, ëcĤ, ïš©, ðLĶĵ, Ý¥, Ý¥, á«Ç, ðÉhÇ, ïcth, ëj¼, íÍN, ðÉh±, ðÉlij]]."

Figure 22: Randomly selected examples from **Rainbow** demonstrations used in the post-training stage.