Robotics Foundation Models with Reasoning in the Loop

Jiafei Duan

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2026

Reading Committee:
Dieter Fox, Co-Chair
Ranjay Krishna, Co-Chair
Wolfram Burgard

Program Authorized to Offer Degree:
Computer Science and Engineering

© Copyright 2026

Jiafei Duan

University of Washington

Abstract

Robotics Foundation Models with Reasoning in the Loop

Jiafei Duan

Co-chairs of the Supervisory Committee:

Professor Dieter Fox
Computer Science and Engineering

Assistant Professor Ranjay Krishna
Computer Science and Engineering

Recent advances in generative AI have demonstrated the power of scaling: large language and vision models trained on internet-scale data now exhibit remarkable capabilities in perception, generation, and reasoning, often generalizing to tasks and domains far beyond those seen during training. These successes have inspired growing interest in bringing foundation-model paradigms to robotics, with the goal of moving beyond narrow, task-specific autonomy in constrained environments toward general-purpose robots that perceive, plan, and act robustly across diverse open-world settings. However, robotics fundamentally differs from language and vision in ways that resist a direct transfer of the scaling recipe. Robot learning cannot rely on the passive accumulation of internet data, since embodied interaction data must be physically collected through hardware, human teleoperation, or simulation, each of which is expensive, slow, and difficult to scale to the diversity of real-world tasks. Moreover, embodied data is inherently long-tailed: rare events, edge cases, and failure modes are precisely the situations in which robustness matters most, yet the hardest to obtain. As a result, simply scaling data and model parameters is insufficient. Building general-purpose, robust robotics foundation models therefore demands a different question: how can robots learn more from less data, generalize beyond their training distribution, and continue to improve over time through their own experience?

This thesis argues that reasoning in the loop offers a promising path forward. Rather than treating reasoning as a downstream capability applied after learning, I show how reasoning can be integrated directly into the learning process itself, shaping how data is represented, how models are trained, and how policies are refined through deployment. By reasoning over structured feedback, temporal context, and failure, robots can extract substantially more signal from each interaction, compensating for data scarcity and improving generalization to unseen objects, scenes, and tasks. I develop this agenda along three complementary axes. First, I introduce approaches for spatial reasoning that enable robots to ground language in 3D space and reason explicitly about geometric and relational structure among objects, supporting precise manipulation in cluttered and previously unseen environments. Second, I develop temporal reasoning through memory-centric models that retain, query, and reason over past observations and actions, allowing robots to maintain coherent behavior across extended time horizons and to perform high-precision, long-horizon tasks that exceed the scope of purely reactive policies. Third, I show how reasoning over failures allows robots to diagnose why actions fail, attribute errors to specific causes, and use that understanding to self-improve, increasing robustness without additional human supervision or demonstrations. Taken together, these contributions reframe robotics foundation models as systems that learn through reasoning, closing the loop between perception, action, and structured inference. This perspective points toward a new generation of self-improving autonomous systems capable of operating reliably in the open world, and lays a foundation for scalable, data-efficient embodied intelligence.

Acknowledgements

The four years of my PhD have gone by quickly, yet they have been deeply fulfilling, intellectually rich, and truly once-in-a-lifetime. This journey has not always been smooth sailing, but it has been made possible by the guidance, support, encouragement, and companionship of my advisors, mentors, collaborators, friends, and family. Their generosity and belief in me have shaped not only this thesis, but also the person I have become throughout this journey.

I am deeply and sincerely grateful to both of my advisors, Dieter Fox and Ranjay Krishna. I would especially like to thank Dieter for taking a chance on me at the very beginning and welcoming me as a PhD student even though I had no prior experience in robotics and had never even touched a robot before starting my journey. That leap of faith changed the course of my life, transforming me from someone who might have pursued a PhD in cognitive science into someone who found a home in robotics. Throughout these years, Dieter has offered unwavering support for every idea I chose to pursue, trusting me fully and empowering me to turn those ideas into something meaningful. Beyond his support, Dieter has shaped the way I think as a scientist. He taught me the importance of rigor, of grounding ambitious ideas in reality, and of constantly striving for clarity and depth in research. Whenever my thoughts drifted too far into abstraction, he patiently brought me back, helping me see things more clearly. His insight, openness to new directions, and steady guidance created an environment where I could grow, explore, and take risks. More than anything, it is Dieter's trust and belief in me that made this PhD journey not only possible, but truly enjoyable and transformative.

I would also like to express my deepest gratitude to my co-advisor, Ranjay Krishna. From the very beginning, you welcomed me as your student and chose to co-advise me, and I am profoundly thankful for that decision. Your support has been unwavering. You have always believed in my ideas, encouraged me to

pursue them boldly, and, more importantly, shown me what it means to be the kind of researcher and mentor I aspire to become. When people ask me why I want to become faculty, my answer is simple: because I hope to one day be like you. I still vividly remember the first paper I worked on during my PhD, AR2D2. In the two weeks leading up to the submission, you worked alongside me day and night, refining every detail, from the writing to the figures. I can still hear you saying that just because others do not make good figures does not mean we should not. That standard of care and pride in our work has stayed with me. The paper was accepted to CoRL with outstanding reviews, and more than anything, it gave me a deep sense of confidence at the very start of my PhD journey.

Beyond that moment, I am incredibly grateful for all the opportunities you have given me, both at UW and at the Allen Institute for Artificial Intelligence. Your guidance during my faculty applications was invaluable. You consistently provided thoughtful feedback on my job talks, pushing me through countless refinements. I believe we went through nine iterations together. I am also deeply touched by the times you presented a body of work that was entirely mine. That gesture meant more to me than I can fully express. We began this journey at the same time, with you starting your faculty career and me beginning my PhD, and it has been truly special to grow alongside you. Looking back, I would choose you as my advisor again without hesitation.

I would also like to express my sincere gratitude to Ali Farhadi for his continued support during my time at the Allen Institute for Artificial Intelligence. As both a leader and mentor, Ali has been instrumental in supporting the development of MolmoAct and MolmoAct2. I am especially thankful for his thoughtful feedback, career guidance, and the time he dedicated to reviewing my talks and letters, which opened many doors and gave me invaluable opportunities to succeed. Many of the most ambitious aspects of my work would not have been possible without his encouragement and belief in me. His support has had a lasting impact on both my research and my growth as a scientist.

Above all, I would also like to sincerely thank my thesis committee members: Dieter Fox, Ranjay Krishna, Wolfram Burgard, Leslie Pack Kaelbling, and Karen Leung. I am deeply grateful for their time, guidance, and insightful feedback throughout this final stage of my PhD journey, from the final examination to the completion of my thesis defense. Their thoughtful questions, constructive suggestions, and encouragement helped me sharpen my ideas, strengthen the presentation of my work, and reflect more deeply on the broader

impact and future directions of my research. I feel very fortunate to have had such an exceptional committee supporting and challenging me as I brought this thesis to completion.

In my research journey, I am deeply grateful to the many collaborators and mentors who have supported and shaped my work across the University of Washington, the Allen Institute for Artificial Intelligence, the NVIDIA Seattle Robotics Lab, and beyond. Working alongside all of you has been one of the most meaningful parts of my PhD. You have continually challenged me to think more deeply, pushed me to improve the quality of my research, and inspired me to aim higher. I am especially thankful to Helen Wang, Arijit Ray, Yejin Kim, Rose Hendrix, Noah A. Smith, Yuquan Deng, Winson Han, Suvir Mirchandani, Mia Tang, Dorsa Sadigh, Michael Cho, Markus Grotz, Siddhartha Srinivasa, Kuo-Hao Zeng, Ainaz Eftekhar, Kiana Ehsani, Som Sager, Yuke Zhu, Ajay Mandlekar, Yijie Guo, Wentao Yuan, Harold Soh, Ishika Singh, Ani Kembhavi, Mohit Shridhar, Nishanth Kumar, and Cheston Tan. Your insights, generosity, and collaboration have been invaluable, and many of the projects we worked on together would not have been possible without you.

Beyond my collaborators, I would also like to express my sincere appreciation to the students I have had the privilege to mentor across PhD, master's, and undergraduate levels. Working with all of you has been one of the most meaningful and rewarding parts of my journey. Thank you for trusting me with your projects and for allowing me to be part of your growth as emerging researchers. Watching each of you develop into thoughtful, independent thinkers has been truly inspiring. I am especially grateful to Haoquan Fang, Shuo Liu, Donovan Clay, Yalcin Tur, Wei-Chuan Tsai, Shirui Chen, Wilbert Pumacay, Ishneet Sukhvinder Singh, and Shulin Tian. Your curiosity, dedication, and drive have not only shaped your own paths, but have also deeply enriched my experience as a mentor.

Beyond research, I would like to thank the many people I have had the joy of crossing paths with over the past four years, especially during my time in Seattle. My experiences at the RSE Lab, the Allen Institute for Artificial Intelligence, and the Krishna Lab have been made truly memorable by the people in them. It is the everyday conversations, shared moments, and collaborations that brought warmth and energy to this journey. I am especially grateful to Helen Wang, Ainaz Eftekhar, Amita Kamath, Arjun Guru, Benlin Liu, Chenhao Zheng, George Stoica, Jieyu Zhang, Lindsey Li, Scott Geng, Xiang Fan, Zixian Ma, Mahtab Bigverdi, Eric Cai, Ge Yan, Jesse Zhang, Marius Memmel, Zoey Chen, Aaron Walsman, Wentao Yuan, Yi Li, Joel Jang, Markus Grotz, Xiangyun, Mohit Shridhar, Kristin Cha, Rose Hendrix, Yejin Kim, Winson Han, Stone Tao,

and Jason Ren. Thank you all for the laughter, the support, and the sense of community you brought into my life. These moments, beyond the research itself, are what I will carry with me long after this chapter ends.

My PhD was generously supported by the A*STAR National Science PhD Fellowship. I would also like to extend my gratitude to UW HPC and the AI2 Beaker team for providing access to the HYAK and Beaker compute clusters, respectively. I also thank ChatGPT and Claude for helping fix LaTeX tables and for providing feedback on flow, structure, and grammar.

Last, but most importantly, I would like to thank the two women who mean the world to me. First, I would like to thank my mother, Zhang Heng. Thank you for raising me, for bringing me from China to Singapore, and for giving me the opportunity and life to become the person I am today. Life was never easy for you. As a single parent, you worked multiple shifts and carried countless burdens quietly, all so that I could have a better life. For you, the world has always been simple: as long as your son is healthy, safe, and happy, everything is enough. I know that behind every opportunity I have had, every step I have taken, and every dream I have been able to pursue, there has been your sacrifice, your strength, and your unconditional love. Now, after all these years, I am finally done studying. It is my turn to take care of you for the rest of your life. You have given me more than I could ever repay, and words will never be enough to express my lifelong gratitude. Everything I have achieved, and everything I will become, carries a part of you.

Lastly, I would like to thank my partner, Helen, the love of my life. We became friends at the very beginning of my PhD, and from then on, you have been by my side through both the rainy and sunny days of Seattle. You comforted me when I was filled with self-doubt, supported me through every difficult decision, and believed in me even when I struggled to believe in myself. Even when I made decisions that were not always the best, and even when I annoyed you more than I should have, you still chose to stay beside me with patience, love, and kindness. Your positivity, cheerfulness, and ability to see the good in people and in life have been a tower of light for me. You have guided me through some of the most stressful and uncertain moments of this journey, and you always seem to know how to calm me down when the world feels too heavy. You make me a better person every day. Without you, and without the warmth of your never-fading smile, my PhD journey would have been missing so much of its color. You made the difficult days bearable, the ordinary days joyful, and the happiest days even brighter. You are not only my collaborator in research, but also my one and only collaborator in this much longer journey of life. I am endlessly grateful for your love,

your patience, and your belief in me. I look forward to growing old with you, and to continuing this journey together, wherever life takes us, for as long as we are given in this lifetime and the next.

Contents

1 Introduction **29**

1.0.1 Bridging the Spatial–Semantic–Action Gap in Robotics Foundation Models 31

1.0.2 Representing and Storing Memory in a Computationally Tractable Way for Manipulation 32

1.0.3 Scaling Failure Reasoning for Open-World Manipulation 34

2 Spatial Reasoning: Bridging Spatial Semantics and Action in Robotics Foundation Models **37**

2.1 Introduction 37

2.2 Related Work 39

2.2.1 Generalist Robot Manipulation Policies 39

2.2.2 Robot Reasoning and Planning with Language 40

2.2.3 Embodied Reasoning for Robotic Manipulation 41

2.3 MolmoAct 42

2.3.1 Vision Language Model 42

2.3.2 Vision-Language-Action Model 43

2.3.3 Action Reasoning Model 44

2.3.4 Action Steerability via Visual Reasoning Trace 48

2.4 Data Curation and Generation 49

2.4.1 Action Reasoning Data 49

2.4.2 MolmoAct Dataset 51

2.4.3 Multimodal Web Data 52

2.5 Training Recipe . . . 52
2.5.1 Pre-training . . . 53
2.5.2 Mid-training . . . 53
2.5.3 Post-training . . . 54
2.6 Experimental Evaluation . . . 55
2.6.1 MOLMOACT After Pre-training . . . 56
2.6.2 Fast Adaptation of MOLMOACT in Post-training . . . 57
2.6.3 Effectiveness of MOLMOACT in Out-of-Distribution Generalization . . . 58
2.6.4 Effect of the MOLMOACT DATASET on MOLMOACT Performance . . . 60
2.6.5 Instruction Following of MOLMOACT . . . 60
2.6.6 Steerability of MOLMOACT . . . 61
2.7 Conclusion . . . 62

3 Temporal Episodic Reasoning: Storing and Representing Memory for Robotics Foundation Models 63

3.1 Introduction . . . 63
3.2 Related Work . . . 65
3.2.1 3D-Based Robotic Transformers for Manipulation . . . 65
3.2.2 Visual Representations for Robot Learning . . . 66
3.2.3 Memory in Robotics . . . 66
3.3 `MemoryBench`: A Memory Benchmark for Robotic Manipulation . . . 67
3.3.1 Task Design . . . 68
3.3.2 Spatial Memory-Based Tasks . . . 68
3.4 Method . . . 69
3.4.1 SAM2Act: Multi-Resolution Upsampling for Enhanced Visual Feature Representation 70
3.4.2 SAM2Act+: Action Memory Architecture for Improved Spatial Awareness in Past Observations . . . 72
3.5 Experiments . . . 74
3.5.1 Experimental Setup . . . 74

3.5.2 Performance Across 18 RLBench Tasks . . . 75
3.5.3 Semantic Generalization Across Tasks . . . 76
3.5.4 Performance on `MemoryBench` . . . 77
3.5.5 Real-Robot Evaluations . . . 77
3.6 Conclusion and Limitations . . . 77

4 Failure Reasoning: Moving from Binary Success to Non-Scalar Feedback 79
4.1 Introduction . . . 79
4.2 Related Work . . . 83
4.2.1 Failure Detection in Robotic Manipulation . . . 83
4.2.2 Data Generation in Robotics . . . 83
4.2.3 Foundation Models for Robotic Manipulation . . . 84
4.3 The AHA Dataset . . . 84
4.3.1 Failure Modes in Robotic Manipulation . . . 84
4.3.2 Implementation of the AHA Dataset . . . 86
4.4 Method . . . 87
4.4.1 Failure Reasoning Formulation . . . 87
4.4.2 Synthetic Data for Instruction-Tuning . . . 88
4.4.3 Instruction Fine-Tuning . . . 88
4.4.4 Impact on Downstream Tasks . . . 89
4.5 Experimental Results . . . 89
4.5.1 Experimental Setup . . . 90
4.5.2 Quantitative Experimental Results . . . 91
4.5.3 Downstream Robotic Tasks . . . 93
4.6 Conclusion . . . 94

5 Conclusion 97

A Appendix: Bridging Spatial Semantics and Action in Robotics Foundation Models 125
A.1 Appendix . . . 125

A.1.1 Overview . . . 125
A.1.2 Model Details . . . 126
A.1.3 Training Details . . . 128
A.1.4 Evaluation Details . . . 132
A.1.5 Data Details . . . 137
A.1.6 Data Examples . . . 138
A.1.7 Limitations and Potential Solutions . . . 138

B Appendix: Storing and Representing Memory for Robotics Foundation Models 169

B.1 Appendix . . . 169
B.1.1 Overview . . . 169
B.1.2 Model Architecture . . . 170
B.1.3 Training Implementation . . . 171
B.1.4 Full Comparisons for RLBench 18 Tasks . . . 173
B.1.5 Full Results for `The Colosseum` . . . 173
B.1.6 Ablation on SAM2Act . . . 173
B.1.7 RLBench Tasks . . . 176
B.1.8 `MemoryBench` Tasks . . . 177
B.1.9 `MemoryBench` Update . . . 177
B.1.10 Real-World Experiments . . . 178

C Appendix: Moving from Binary Success to Non-Scalar Feedback 181

C.1 Appendix . . . 181
C.1.1 Overview . . . 181
C.1.2 Failure Taxonomy . . . 182
C.1.3 FailGen Data-Generation Pipeline . . . 182
C.1.4 AHA Dataset . . . 183
C.1.5 Additional Experimental Results . . . 184
C.1.6 Downstream Robotic Application: VLM Reward Generation . . . 185

C.1.7 Downstream Robotic Application: VLM Task-Plan Generation 187
C.1.8 Downstream Robotic Application: VLM Sub-Task Verification 188
C.1.9 Limitations and Opportunities 189

List of Figures

2.1 **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. 38

2.2 **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. 44

2.3 Distribution of the 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 trace) while reducing multimodal web data to 5%. 45

2.4 **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. 47

2.5 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*. The bottom row shows example task setups with corresponding natural language instructions. 56

2.6 **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. 58

2.7 **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. 59

2.8 **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. . . 61

3.1 SAM2Act is a multi-view, language-conditioned behavior cloning policy trained with fewer demonstrations. Given a language instruction, it can execute high-precision tasks, such as turning the tiny knob on the lamp. It also generalizes to various environmental variations, such as changes in lighting conditions. Through further training with our proposed memory architecture, it now evolves into SAM2Act+, which is capable of solving tasks that require implicit spatial memory—such as remembering where the robot previously stored the pliers, as depicted in the figure. 64

3.2 **Simulation and real tasks.** We demonstrate the effectiveness of SAM2Act+ in solving memory-based tasks by evaluating it against baselines on the three benchmark memory tasks (top). Additionally, we validate our approach using a Franka Panda robot on four real-world tasks (bottom), including tests under out-of-distribution perturbations. 67

3.3 **Overview of the SAM2Act (top) and SAM2Act+ (bottom) architectures.** The SAM2Act architecture leverages the SAM2 image encoder to generate prompt-conditioned, multi-resolution embeddings, fine-tuned with LoRA for efficient adaptation to manipulation tasks. A multi-view transformer aligns spatial coordinates with language instructions, while a cascaded multi-resolution upsampling mechanism refines feature maps and generates accurate translation heatmaps. SAM2Act+ extends this architecture by incorporating memory-based components, including the Memory Encoder, Memory Attention, and Memory Bank, into the coarse branch. These components enable memory-driven reasoning by processing historical heatmaps and integrating prior observations, allowing the agent to predict actions based on stored contextual information. Observations are reconstructed into point clouds, rendered into three virtual images, and lifted into 3D translation points, enabling precise spatial reasoning across both architectures. 70

3.4 **SAM2Act module and multi-resolution upsampling mechanism.** A cascade of three convex upsamplers processes feature maps at increasing resolutions, integrating multi-resolution embeddings from the SAM2 image encoder through elementwise addition and layer normalization. The upsamplers progressively refine features, doubling spatial dimensions at each stage, to generate accurate translation heatmaps while capturing fine-grained spatial details critical for manipulation tasks. 71

4.1 AHA is a Vision-Language Model designed to detect and reason about failures in robotic manipulation. As an instruction-tuned VLM, it can enhance task performance in robotic applications that utilize VLMs for reward generation, task planning, or sub-task verification. By incorporating AHA into the reasoning pipeline, these applications can achieve accelerated and improved performance. 80

4.2 **Overview of the AHA pipeline.** (Top) The data generation for AHA is accomplished by taking a normal task trajectory in simulation and procedurally perturbing all keyframes using our taxonomy of failure modes. Through `FailGen`, we systematically alter keyframes to synthesize failure demonstrations conditioned on the original tasks. Simultaneously, we generate corresponding query and answer prompts for each task and failure mode, which are used for instruction-tuning. (Bottom) The instruction-tuning pipeline follows the same fine-tuning procedure as LLaVA-v1.5 [Liu et al., 2023b], where we fine-tune only the LLM base model—in this case, LLaMA-2-13B—and the projection linear layers, while freezing the image encoder and tokenizer. 82

4.3 (Left) **Scaling law with the AHA dataset.** Effect of model performance with varying domain-specific fine-tuning data. (Right) **Downstream robotic application performance.** AHA outperforms GPT-4o in reasoning about failures within these robotic applications, leading to improved performance of the downstream tasks. 91

4.4 **Downstream robotic application.** We demonstrate that AHA can be integrated into existing LLM/VLM-assisted robotic applications to provide failure reasoning and feedback, helping to accelerate and improve task success rates in these systems. 92

A.1 **Examples of single-arm and bimanual tasks.** We list the observation breakdown to show how the robot performs each task. . . . 141
A.2 **Examples of MOLMOACT DATASET ablation experiments.** . . . 142
A.3 **Language-instruction following.** These are the customized scenes curated for open-ended prompting by users. . . . 142
A.4 Randomly selected examples from **action-reasoning data** used in the pre-training stage. . . . 160
A.5 Randomly selected examples from **auxiliary visual reasoning trace** data used in the pre-training stage. . . . 161
A.6 Randomly selected examples from **auxiliary depth-perception tokens** data used in the pre-training stage. . . . 162
A.7 Randomly selected examples from **trajectory-conditioned action data** used in the pre-training stage. . . . 162
A.8 Randomly selected examples from **multimodal web data** used in the pre-training stage. . . . 163
A.9 Randomly selected examples from the **MOLMOACT DATASET (home environment)** used in the mid-training stage. . . . 164
A.10 Randomly selected examples from the **MOLMOACT DATASET (tabletop environment)** used in the mid-training stage. . . . 165
A.11 Randomly selected examples from **single-arm Franka** demonstrations used in the post-training stage. . . . 166
A.12 Randomly selected examples from **bimanual Franka** demonstrations used in the post-training stage. . . . 167
A.13 Randomly selected examples from **Rainbow** demonstrations used in the post-training stage. 168

B.1 **Real-world robot setup.** A Franka Panda robot with a Robotiq gripper. A RealSense D455 depth sensor captures the scene. . . . 179

C.1 **Failure-mode reference coordinate systems.** (Left) translation coordinate system; (right) rotation coordinate system. . . . 183
C.2 **Example `FailGen` configuration files.** . . . 184

C.3 **Data distribution of the AHA dataset for both train and test.** 185

C.4 **Examples of different failure modes.** Row 1: `No_Grasp` and `Rotation_x`. Row 2: `Rotation_y` and `Rotation_z`. Row 3: `Slip` and `Wrong_sequence`. Row 4: `Translation_x` and `Translation_y`. Row 5: `Translation_z` and `Wrong_object`. 186

C.5 **Example improved dense reward functions.** (Left) GPT-4o reflection. (Right) AHA reflection. 188

C.6 **RL policy rollouts improved with AHA.** Row 1: `pickup_YCB`. Row 2: `push_T`. Row 3: `place_sphere`. Row 4: `stack_cube`. Row 5: `open_drawer`. 191

C.7 **Examples of modified reward functions via AHA.** 192

C.8 **TAMP policy rollouts improved with AHA.** Row 1: `put_banana_centre`. Row 2: `stack_banana`. Row 3: `stack_cubes`. 193

C.9 **Examples of modified task plans via AHA.** 194

C.10 **Examples of zero-shot data-generator trajectories with AHA as sub-task verifier.** Row 1: `pickup_cube`, `pickup_cup`. Row 2: `put_block`, `sort_mustard`. 195

List of Tables

2.1 **SimplerEnv evaluation across different policies on Google Robot tasks.** The zero-shot and fine-tuning results denote performance of OXE-dataset-pretrained models [O'Neill et al., 2024] and RT-1-dataset-fine-tuned models [Brohan et al., 2022], respectively. 52

2.2 **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 autoregressive baselines, with strong performance across all categories, particularly in long-horizon tasks. 54

3.1 **Multi-task performance on RLBench.** We evaluate 18 RLBench tasks [James et al., 2020], reporting success rates across all tasks among 3D keyframe-based behavior cloning (BC) policies. We report stats over 4 evaluations for SAM2Act. Our method, SAM2Act, outperforms all baselines, achieving a significant performance margin of 5.8% over RVT-2 [Goyal et al., 2024], the prior state-of-the-art 3D keyframe-based BC policy. 76

3.2 **`The Colosseum` results.** Task-average success rate percentage change for SAM2Act and other baselines across 13 perturbation factors from `The Colosseum`, relative to evaluations without perturbations. Results of 3 evaluations are reported for all models. Our approach, SAM2Act, demonstrates the lowest average percentage change across all perturbations, with a minimal drop of $-4.3 \pm 3.6\%$, highlighting its robustness in handling various environmental and object-level perturbations. 76

3.3 **Performance on `MemoryBench`.** We report the success rates for the three spatial memory tasks in `MemoryBench`. Our method, SAM2Act+, significantly outperforms all baseline methods that lack an explicit memory mechanism, achieving an average improvement of 37.6% across all three tasks. 77

3.4 **Real-world results.** We compare RVT-2 against SAM2Act for the first three tasks and SAM2Act+ on the last real-world task (indicated with $*$), evaluating performance both in-distribution and out-of-distribution at test time. 78

4.1 **AHA datasets for instruction-tuning.** We combine the AHA dataset, our large-scale robotic-manipulation failure dataset, with VQA and object-detection data. By incorporating this diverse data mix into the fine-tuning process, AHA is able to reason about failures in robotic manipulation across different domains, embodiments, and tasks. 85

4.2 **Quantitative evaluation on failure detection and reasoning.** AHA is evaluated and benchmarked against three open and three proprietary VLMs and one visual-prompting baseline across three evaluation datasets. AHA outperforms all other VLMs on every evaluation dataset and nearly every evaluation metric, with the exception of the AHA (Test) dataset, where GPT-4o exceeds it by less than 3%. 90

4.3 **Quantitative evaluation on standard VQA benchmarks.** AHA performs on par with LLaVA-13B [Liu et al., 2023b], the VLM from which AHA adapts its fine-tuning strategy. . 92

A.1 **MOLMOACT's Architecture Hyperparameters.** We specify all hyperparameter information for the different model architectures for MOLMOACT-7B-D and MOLMOACT-7B-O. 131

A.2 **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. 131

A.3 **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. 133

A.4 Open-ended prompts provided by users grouped by scene for language-instruction following. 143

A.5 **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.>`). 144

A.6 **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. 145

A.7 **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. 146

A.8 **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. 147

A.9 **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. 148

A.10 **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. 149

A.11 Detailed per-trial performance for *Fold Towel* for bimanual tasks. Each row shows the task-progress score for a specific trial. 150

A.12 Detailed per-trial performance for *Lift Tray* for bimanual tasks. Each row shows the task-progress score for a specific trial. 151

A.13 Detailed per-trial performance for *Set up Table* for bimanual tasks. Each row shows the task-progress score for a specific trial. 152

A.14 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. 153

A.15 Detailed per-trial performance for *Wipe Table* for single-arm tasks. Each row shows the task-progress score for a specific trial. 154

A.16 Detailed per-trial performance for *Clean the table* for single-arm tasks. Each row shows the task-progress score for a specific trial. 155

A.17 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 scores used to evaluate each model are detailed at §A.1.4. 155

A.18 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. 156

A.19 Per-episode evaluation results for bowl-picking tasks with different steering conditions. Scores indicate task progression for each model configuration. 157

A.20 Task details of MOLMOACT DATASET Home Environment, including scene, task name, language instruction, and all objects used for data collection. 158

A.21 Task details of MOLMOACT DATASET Tabletop Environment, including scene, task name, language instruction, and all objects used for data collection. 159

B.1 **Training hyperparameters of SAM2Act on RLBench and `The Colosseum`.** The batch size is the total batch size across all GPUs. For the learning rate, we follow the scaling strategy used in RVT-2 [Goyal et al., 2024], where the learning rate is scaled by the batch size as $1.25\text{e}{-5} \times bs$. 172

B.2 **Training hyperparameters of SAM2Act and SAM2Act+ on `MemoryBench`.** The batch size is the total batch size across all GPUs. For SAM2Act+ training on the `reopen_drawer` task, we use a maximum window size of 8, resulting in a per-GPU batch size of 8 and a total batch size of 256. Similarly, for the other two tasks, where the maximum window size is 10, the total batch size is $10 \times 32 = 320$. The learning rate follows the same scaling rule as in Table B.1. 173

B.3 **Full comparisons of multi-task performance on RLBench.** We evaluate 18 RLBench tasks [James et al., 2020], reporting success rates across all tasks among all existing approaches, not limited to 3D keyframe-based behavior-cloning (BC) policies. We report stats over 4 evaluations for SAM2Act. Our method, SAM2Act, outperforms all baselines, achieving a performance margin of 1.9% over ARP$^+$ [Zhang et al., 2024b], the prior state-of-the-art approach. 174

B.4 **Full results of SAM2Act for various perturbations on `The Colosseum`.** Mean and standard deviation of 3 evaluations are reported. "–" indicates that the perturbation is not applicable to the given task. 175

B.5 **SAM2Act ablation performance on RLBench.** We report the success rates for 18 RLBench tasks [James et al., 2020], along with the average success rate and ranking across all tasks. The table shows that SAM2Act outperforms all of its variations. 175

B.6 **The 18 RLBench tasks for the multi-task experiment.** We report the language template, the average number of extracted keyframes, the number of task variations, and the variation type. 176

B.7 **Properties of the real-world tasks.** We report the language template, the average number of extracted keyframes, the number of items the robot can interact with, the task variations, and the variation type. 178

Chapter 1

Introduction

Robotics has long stood as one of the grand challenges of artificial intelligence [Duan et al., 2022; Bommasani et al., 2021]. For decades, researchers have sought to build machines that can perceive, reason, and act in the physical world: self-driving cars capable of reaching full autonomy [Hussein et al., 2017], intelligent manufacturing systems that can automate complex factory workflows, mobile robots that can navigate reliably through cities and indoor environments [Zeng et al., 2024; Eftekhar et al., 2023; Sun et al., 2019], and aerial drones that can support last-mile delivery [Gupta et al., 2025; Kaufmann et al., 2023]. Across these domains, advances in perception, planning, control, and machine learning have gradually transformed robotics from a largely laboratory-bound discipline into a field with growing real-world impact.

Yet one of the most important frontiers of robotics remains largely unsolved: building robots that can operate in the unstructured, diverse, and constantly changing environments of everyday human life. Homes, hospitals, offices, and public spaces are far less predictable than factories or roads. They contain long-tail objects, ambiguous instructions, cluttered scenes, changing layouts, and tasks that require physical interaction with the world. A robot operating in these environments must do more than recognize objects or execute pre-programmed motions. It must understand human intent, reason about spatial and temporal context, adapt to unfamiliar situations, recover from mistakes, and manipulate objects safely and effectively.

Recent progress in generative AI, particularly large language models and vision-language models [Reid et al., 2024; OpenAI, 2024; Deitke et al., 2024; Li et al., 2024a], has opened a new path toward this goal. These models demonstrate that scaling data, model capacity, and training objectives can lead to broad generalization,

emergent reasoning abilities, and flexible interaction through natural language. Inspired by these advances, the robotics community has begun to explore Robotics Foundation Models: general-purpose models that can take in multimodal observations and human-understandable language instructions, and produce actions that allow robots to complete a wide range of tasks across environments, embodiments, and domains [Bjorck et al., 2025; NVIDIA et al., 2025; Kim et al., 2024; Black et al.; Lee et al., 2025].

At the same time, robotics differs fundamentally from language and vision. Large language models benefit from internet-scale text data, while vision models can be trained on massive collections of images and videos. Robot learning, by contrast, requires embodied interaction data: observations paired with actions, physical consequences, and task outcomes. Such data is expensive, slow, and difficult to collect at scale. Although recent efforts have produced larger robot datasets and more scalable tools for collecting demonstrations across diverse environments [O'Neill et al., 2024; Khazatsky et al., 2024], the amount of available robot data remains orders of magnitude smaller than the data used to train modern foundation models in language and vision. This gap raises a central question: how can we build generalist robot models that learn efficiently from limited embodied data while still generalizing to the complexity of the real world?

This thesis argues that reasoning in the loop is a key ingredient for building such models. Reasoning has always been central to robotics, especially in the form of task planning [Fikes and Nilsson, 1971; Ghallab et al., 2004], motion planning [Latombe, 1991; LaValle, 2006; Choset et al., 2005], and model-based decision-making [Bertsekas, 2017; Thrun et al., 2005]. Classical robotics systems often reason through explicit representations of states, goals, constraints, and dynamics [Siciliano et al., 2009; Lynch and Park, 2017; Murray et al., 1994], using planners to compute action sequences before execution [Russell and Norvig, 2020; Garrett et al., 2021]. These approaches have been highly successful in structured settings, but they typically depend on carefully designed abstractions, known environments, and well-specified objectives.

This thesis studies reasoning through the lens of modern foundation models, particularly large language models and vision-language models. Here, reasoning is a mechanism for connecting language, perception, memory, action, and feedback within the learning and deployment process itself. Rather than treating reasoning as an external capability applied after perception or planning, this thesis explores how reasoning can be integrated directly into the supervision, representations, and execution of robotics foundation models. By incorporating structured forms of reasoning over language, space, time, memory, actions, and failures,

robot models can learn more effectively from limited data, better connect high-level semantic understanding to low-level control, and improve their ability to adapt and self-correct in open-world environments. From this perspective, reasoning is not limited to producing a plan before acting; it also helps a robot understand what information matters, represent intermediate goals, interpret feedback, remember relevant past experience, and revise its behavior when execution fails.

The central vision of this thesis is to move beyond robots that merely imitate demonstrations toward robots that can reason about what they are doing, why they are doing it, and how to recover when things go wrong. This requires rethinking the interfaces between perception and action, the forms of supervision used to train robot policies, and the mechanisms through which robots use past experience to inform future behavior. Through this lens, Robotics Foundation Models are not only scaled-up policies trained on more robot data, but systems that combine broad multimodal understanding with structured reasoning mechanisms that make their behavior more controllable, interpretable, and robust.

Ultimately, this thesis aims to contribute toward a future in which robots can assist people in everyday environments, from homes and laboratories to hospitals and workplaces, by carrying out mundane, repetitive, or physically demanding tasks. Achieving this future will require robots that can understand human instructions, ground them in perception, act with precision, learn from feedback, and continuously improve through interaction. Reasoning-in-the-loop provides a promising path toward this goal by bringing the flexible reasoning capabilities of foundation models into embodied learning and control, thereby helping bridge the gap between internet-scale intelligence and embodied intelligence in the physical world.

1.0.1 Bridging the Spatial–Semantic–Action Gap in Robotics Foundation Models

A central challenge in building Robotics Foundation Models is bridging the gap between high-level semantic understanding and low-level physical action. Modern vision-language models have demonstrated impressive abilities in object recognition, visual question answering, language grounding, and commonsense reasoning. However, these capabilities do not directly translate into effective robot control. A robot must not only understand what an instruction means, but also determine where to act, how to move, and how its actions will change the physical state of the environment.

This mismatch creates what this thesis refers to as the spatial–semantic–action gap. On the one hand,

foundation models trained on internet-scale image-text data acquire broad semantic knowledge about objects, scenes, and language. On the other hand, robot policies must output precise and temporally coordinated actions in continuous physical space. Between these two levels lies a difficult missing interface: the ability to convert semantic intent into spatially grounded action representations that are useful for control.

For example, an instruction such as "place the mug next to the plate" requires more than recognizing the mug and the plate. The robot must infer the target spatial relationship, identify feasible grasp and placement regions, reason about object geometry and free space, and execute a sequence of motions that satisfies the task. These requirements expose a limitation of directly mapping observations and language to actions: without structured intermediate representations, the model may struggle to learn the spatial reasoning necessary for robust manipulation, especially under limited robot data.

This thesis argues that effective Robotics Foundation Models should incorporate explicit or implicit reasoning interfaces that connect semantics to action. Such interfaces may include spatial points, object-centric representations, visual traces, affordance maps, action waypoints, or other intermediate structures that expose where and how the robot should act. By supervising and modeling these intermediate reasoning steps, robot policies can better leverage the semantic knowledge of foundation models while grounding that knowledge in the spatial constraints of physical interaction.

Bridging the spatial–semantic–action gap is therefore not only a problem of scaling model size or robot data. It requires designing representations that allow models to reason about space in ways that are actionable for robots. This thesis explores how structured spatial reasoning can serve as a bridge between high-level multimodal understanding and low-level manipulation, enabling robot policies to generalize more effectively across objects, tasks, and environments.

1.0.2 Representing and Storing Memory in a Computationally Tractable Way for Manipulation

Many manipulation tasks are not Markovian from a single observation. A robot often needs to remember what it has seen, what it has done, which objects have moved, which parts of the scene are occluded, and what subgoals have already been completed. This is especially important for long-horizon tasks, where success depends on maintaining temporal context across many steps. For example, when organizing objects, opening

containers, or assembling multi-part structures, the robot must use information from earlier observations to guide later actions.

While memory is essential for robust manipulation, representing and using memory in Robotics Foundation Models remains challenging. A simple approach is to attend over all past observations, but this quickly becomes computationally expensive as task horizons grow. Storing full image histories or dense visual features can also be inefficient and difficult to scale. In contrast, overly compressed recurrent states may discard important spatial details that are needed for precise control. This creates a tension between expressiveness and tractability: memory must preserve task-relevant information while remaining efficient enough for real-time robot execution.

This thesis studies memory as a structured reasoning mechanism for manipulation. Rather than treating memory as an unbounded history buffer, the goal is to represent past experience in compact, actionable forms that can be retrieved when needed. Such memory may encode key scene states, object locations, past actions, visual features, or task-relevant spatial context. By storing only the information most useful for future decision-making, a robot can reason over long horizons without incurring the computational cost of full-history attention.

A computationally tractable memory system should satisfy several properties. First, it should be selective: not every past observation is equally useful, so the model must identify which information should be stored. Second, it should be grounded: memory should preserve spatial and visual information in a form that can support manipulation. Third, it should be retrievable: the robot should be able to access relevant past information based on the current observation, instruction, and task context. Finally, it should be scalable: the cost of memory should not grow prohibitively with task length.

By incorporating such memory mechanisms, Robotics Foundation Models can move beyond reactive policies that depend only on the current observation. They can instead reason over temporal context, recover information that is no longer visible, and maintain consistency across long-horizon manipulation tasks. This thesis explores how compact and retrievable memory representations can enable robots to act more robustly in partially observable, changing, and cluttered environments.

1.0.3 Scaling Failure Reasoning for Open-World Manipulation

Failure is unavoidable in open-world manipulation. A robot may grasp the wrong object, miss the intended grasp pose, drop an object, collide with the environment, place an object imprecisely, or execute a motion that does not produce the desired outcome. In structured benchmarks, these failures are often reduced to binary success labels. However, binary feedback provides very limited information: it tells the robot whether the task succeeded, but not why it failed or how it should improve.

This thesis argues that failure reasoning is a crucial form of feedback for scaling Robotics Foundation Models. Instead of viewing failure only as an endpoint, failure can be treated as a source of supervision. A robot that can reason about failures can identify the cause of an unsuccessful execution, connect that cause to the task objective, and propose corrective actions. This moves robot learning beyond scalar rewards or success labels toward richer, non-scalar feedback that explains what went wrong.

Failure reasoning is especially important in open-world settings because the distribution of possible errors is large and difficult to enumerate manually. A robot operating in a home, laboratory, or workplace will encounter unfamiliar objects, unexpected object states, ambiguous instructions, and physical interactions that differ from its training data. In such settings, robust deployment requires not only executing known skills, but also recognizing when execution deviates from the intended outcome and adapting accordingly.

Modern vision-language models provide a promising foundation for scaling failure reasoning. Because they are trained on broad multimodal data, they can bring commonsense and visual reasoning capabilities to robot execution traces. However, directly applying general-purpose VLMs to robotics is insufficient: robot failures often involve subtle spatial, temporal, and physical details that are not captured by generic image-text reasoning. A robot failure may depend on whether the gripper contacted the object at the right pose, whether the object moved as intended, or whether a placement satisfied the task-specific spatial relation. These forms of reasoning require data, supervision, and evaluation protocols specifically designed for embodied manipulation.

This thesis studies how to scale failure reasoning for open-world manipulation by constructing models and datasets that reason over robot trajectories, task instructions, and execution outcomes. The goal is to enable robots to produce structured feedback about failures, such as identifying whether the error came from perception, grasping, motion, placement, task planning, or physical interaction. Such feedback can then

support policy improvement, data filtering, recovery behavior, and self-correction. By treating failure as a reasoning problem rather than merely a binary outcome, Robotics Foundation Models can become more robust and adaptive. They can learn not only from successful demonstrations, but also from mistakes. This ability is essential for deploying robots in open-world environments, where failures are inevitable and where continued improvement depends on understanding and correcting them.

Chapter 2

Spatial Reasoning: Bridging Spatial Semantics and Action in Robotics Foundation Models

2.1 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 [Duan et al., 2022; Xu et al., 2024b; Firoozi et al., 2025]. Vision-Language-Action (VLA) models [Black et al.; Kim et al., 2024; Team et al., 2025; NVIDIA et al., 2025; Yang et al., 2025a] 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 brittle and opaque—struggling to transfer across tasks, scenes, or embodiments, and offering little insight into why a robot chose one action over another [Liu et al., 2025; Pumacay et al., 2024].

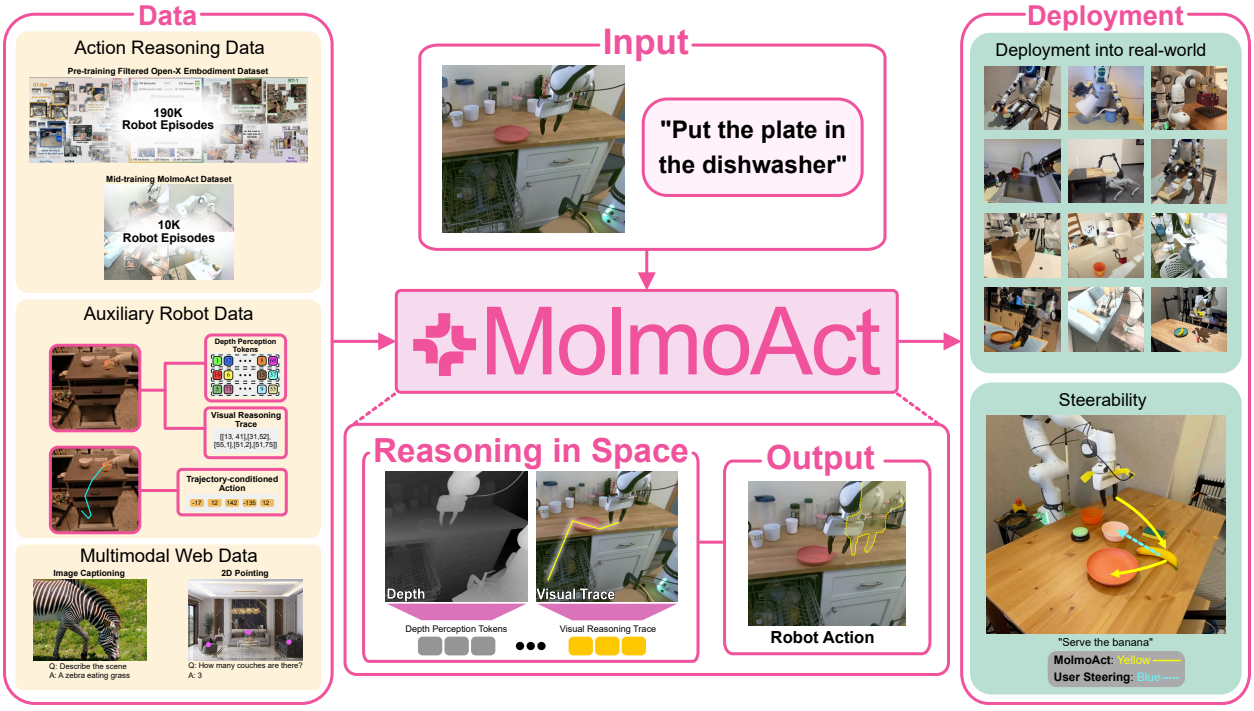


Figure 2.1: Overview. MOLMOACT is an open action reasoning model that, given a user's language instruction, reasons in space and autoregressively predicts three structured reasoning chains: DEPTH PERCEPTION TOKENS for sensing and reconstructing the 3D environment, VISUAL REASONING TRACE TOKENS for representing 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.

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 [Wei et al., 2022; Zelikman et al., 2022; Huang et al., 2022]. We believe robotics can—and must—do the same.

In this chapter, 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 2.1.

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 N1.5 [NVIDIA et al., 2025], π_0 and π_0-FAST [Black et al.], RT-1 [Brohan et al., 2022], and TraceVLA [Zheng et al., 2024a]. 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, serving as a blueprint for building agents that reason, transforming perception into purposeful action through reasoning.

2.2 Related Work

2.2.1 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 [Berscheid et al., 2019; Brohan et al., 2022; Dasari et al., 2019; Ebert et al., 2021; Fang et al., 2023; Jang et al., 2022a; Khazatsky et al., 2024; Mandlekar et al., 2018; Walke et al., 2023; Shafiullah et al., 2023]. 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 [Pumacay et al., 2024; Xie et al., 2024; Lin et al., 2024]. Much

of this progress has been fueled by Large Language Models (LLMs) [O'Neill et al., 2024; Achiam et al., 2023; Groeneveld et al., 2024; Touvron et al., 2023] and Vision-Language Models (VLMs) [Team et al., 2024a; Liu et al., 2023c; Deitke et al., 2024], giving rise to the paradigm of Vision-Language-Action models (VLAs) [Black et al.; Brohan et al., 2022; Zheng et al., 2024a; Zhao et al., 2025; Team et al., 2025; Li et al., 2024b; Qu et al., 2025; Kim et al., 2024]. VLAs pretrain a VLM backbone on large-scale web data to capture rich semantic world knowledge, then fine-tune it for downstream robot control. While most share similar backbones, they differ in their action heads—employing flow matching [Wen et al., 2025b], diffusion [Wen et al., 2025a; Liu et al., 2024e], or advanced action tokenization schemes [Pertsch et al., 2025]. Some adopt hierarchical designs, where a robotics-focused VLM [Bjorck et al., 2025; Li et al., 2025; Shentu et al., 2024] 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.

2.2.2 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 [Ahn et al., 2022; Huang et al., 2023; Bharadhwaj et al., 2024; Fang et al., 2025; Liu et al., 2024d; Shi et al., 2024b; Wang et al., 2024b; Gu et al., 2023]. An alternative line of research seeks to decouple perception and reasoning from low-level control, assigning VLMs the role of performing semantic prediction or generating intermediate representations such as task plans, scene graphs, or spatial layouts [Duan et al., 2024b; Liu et al., 2024a; Li et al., 2025; Huang et al., 2024b; Liang et al., 2022; Singh et al., 2022; Duan et al., 2024a]. Execution is then handled by a separate low-level policy or control module that interprets these high-level outputs and converts them into robot actions.

Action prediction from MOLMOACT can be steered through both natural language and an interactive visual reasoning-trace sketch interface. This dual-modality control improves explainability and enables

more effective diagnosis of model behavior. While prior methods such as RT-Trajectory [Gu et al., 2023], HAMSTER [Li et al., 2025], and inference-time policy steering [Wang et al., 2024b] also offer forms of policy steerability, they differ in important ways. RT-Trajectory and inference-time policy steering are tightly coupled to the architectural constraints and training regimes of robotics transformers or diffusion models, and therefore lack the broader semantic generalization provided by pre-training on a VLM backbone. HAMSTER enables language-conditioned trajectory steering but outputs only 2D trajectories from the high-level VLM, with execution handled by a low-level policy trained on a fixed set of tasks. In contrast, MOLMOACT generalizes its steering to novel spatial configurations, previously unseen objects, and even ambiguous language instructions for a diverse set of tasks, offering a more versatile and semantically grounded control interface with users.

2.2.3 Embodied Reasoning for Robotic Manipulation

Chain-of-thought (CoT) prompting [Wei et al., 2022] has significantly improved the multi-step reasoning capabilities of LLMs across domains such as mathematics, programming, and question answering. This idea has also been extended to the visual domain through multimodal CoT [Bigverdi et al., 2025; Zhang et al., 2023b], where visual information is processed iteratively and reasoned over in conjunction with images. Motivated by these advances, recent works in robotics have explored extending reasoning capabilities to embodied tasks within vision-language-action (VLA) models.

ECoT [Zawalski et al., 2024] synthesizes intermediate subgoals via prompting and uses supervised fine-tuning to teach VLAs to reason before acting. CoT-VLA [Zhao et al., 2025] replaces linguistic CoT with visual subgoal frames generated prior to action prediction. RAD [Clark et al., 2025] leverages action-free human video to curate language-based reasoning to guide low-level actions. ThinkAct [Huang et al., 2025] leverages action-aligned reinforcement learning and visual latent planning to connect embodied reasoning with real-world action prediction in VLAs. Most similar to our reasoning-in-space approach are Emma-X [Sun et al., 2024], which autoregressively fine-tunes OpenVLA with reasoning data formatted as subtasks, predicted future gripper states in 2D, and 3D spatial movement coordinates, and Yang et al. [2025b], which focuses on different forms of mid-level representations, including a trajectory trace with depth awareness, although they only evaluated for diffusion policy on a small data region.

However, unlike ECoT, CoT-VLA, RAD, and ThinkAct—whose reasoning is represented as latent embeddings, generated sub-goals, or textual descriptions that are difficult to ground in the real world and lack the precision required for manipulation—MOLMOACT grounds every step of its reasoning chain directly in the scene. Compared to Emma-X, which reasons primarily over predicted gripper positions without leveraging the full 3D scene context, MOLMOACT performs reasoning in space, where each step can be decoded and visualized both on the image plane and within the 3D environment. This explicit spatial grounding improves explainability and enhances action prediction within a chain-of-thought prompting framework.

2.3 MolmoAct

MOLMOACT is a fully open-source action reasoning model (ARM) for robotic manipulation. It builds on Molmo [Deitke et al., 2024], reusing its vision–language backbone composed of a vision encoder, a vision–language connector, and a large language model (LLM). While Molmo supports chain-of-thought reasoning for language and vision, MOLMOACT extends this capability to generate grounded action sequences. The model is trained on a subset of the Open X-Embodiment (OXE) mixture [O'Neill et al., 2024] consisting of BC-Z [Jang et al., 2022a], BridgeData V2 [Walke et al., 2023], and RT-1 [Brohan et al., 2022], and our in-house collected MOLMOACT DATASET.

In the following sections, we describe the VLM preliminaries (§2.3.1), our method to adapt VLMs for action prediction via action tokenization (§2.3.2), how we transform Molmo into an action reasoning model (ARM; §2.3.3), and our approach to steer action by visual reasoning traces (§2.3.4).

2.3.1 Vision Language Model

To equip an action model with visual and linguistic world knowledge, we build upon vision–language models (VLMs). Most modern VLMs share a three-component structure: (i) a visual encoder that transforms an image into patch-level embeddings, (ii) a projection module that maps these visual features into the input space of a language model, and (iii) a large language model (LLM) backbone. These components are typically trained with a next-token prediction objective on paired or interleaved image–text data.

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 [Radford et al., 2021b] and ViT-SO400M/14 384px SigLIP2 [Tschannen et al., 2025], paired with open LLMs including OLMo2-7B [OLMo et al., 2024] and Qwen2.5-7B [Qwen et al., 2025]. 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.

The degree of openness in the backbone components varies. SigLIP2 and Qwen2.5 do not disclose the full details of their pre-training, and are presumed to use large-scale Internet-sourced multimodal data. In contrast, OLMo2 provides open training datasets (e.g., LAION-2B/5B [Schuhmann et al., 2022], Dolma [Soldaini et al., 2024]), model weights, and complete training code. Although OpenAI CLIP also uses closed training data, it can be reproduced from scratch, as shown by MetaCLIP [Xu et al., 2024a]. We use the model from OpenAI because it was trained for higher resolution images, and also following the previous choice from Molmo [Deitke et al., 2024]. In MOLMOACT, we initialize vision and language components from these open checkpoints whenever available, and use the pre-training procedure from Molmo [Deitke et al., 2024] to train the VLM on dense captioning data. Then, after vision-language alignment, we start to fine-tune MOLMOACT with a subset of the Open X-Embodiment (OXE) mixture and the MOLMOACT DATASET. This enables full reproducibility and supports community-driven re-training and ablation studies on data curation and scaling.

2.3.2 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 [Brohan et al., 2022; Kim et al., 2024; Zitkovich et al., 2023] 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. This yields an N-dimensional action represented as N integers in $[0, 255]$. The model is trained end-to-end with a next-token prediction objective, and the loss is computed only on the action tokens.

Prior work represents the 256 discretized action bins with 256 distinct language tokens taken from the

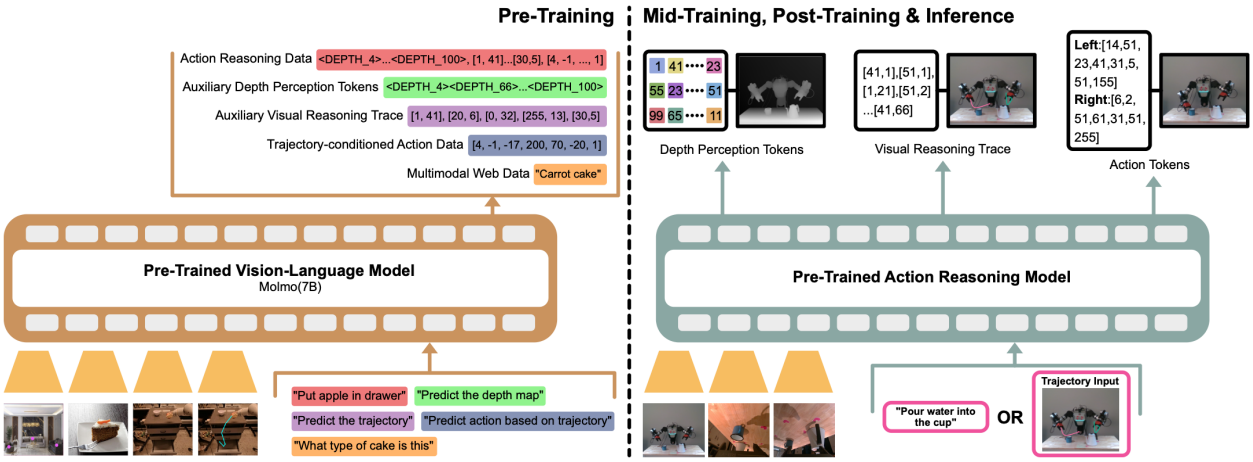


Figure 2.2: Training process of MOLMOACT. The model training process consists of two stages: **Pre-training** (left) and **Post-training, Mid-training & Inference** (right). During pre-training, the vision–language backbone (Molmo) is trained on multimodal and robot reasoning data for diverse objectives, 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.

tail of the vocabulary. Continuous action bins, however, possess ordinal structure and local correlation, whereas arbitrary language tokens are effectively unrelated. This mismatch yields a poor initialization for learning an action codebook. We adopt a simple alternative that better reflects the geometry of the action space. 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, and this becomes our action token vocabulary V_{action}. We notice that these BPE symbols offer a better initialization for action token embeddings by sharing similar characters between adjacent action bins. This similarity-preserving initialization offers a smoother starting point for optimization and, in practice, substantially reduces training time. In contrast to GR00T N1.5's training time of 50,000 GPU hours [NVIDIA et al., 2025], MOLMOACT achieves pre-training with only 9,216 GPU hours: over a $5\times$ reduction.

2.3.3 Action Reasoning Model

Chain-of-Thought (CoT) [Wei et al., 2022] has been shown to significantly improve language models' performance on complex tasks. Likewise, multimodal language models also benefit from multimodal chain of

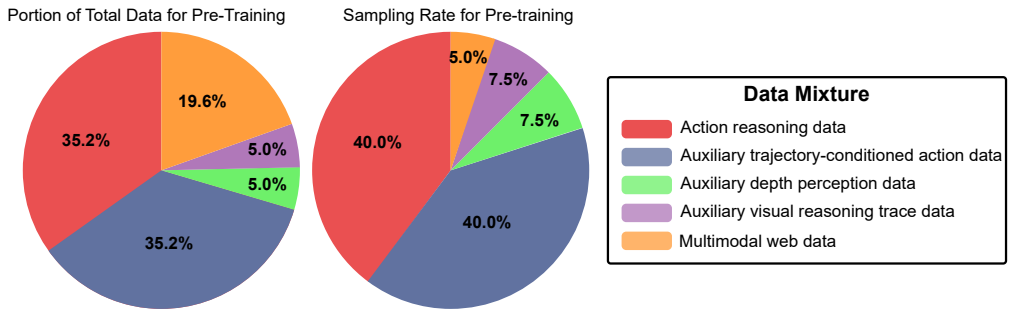


Figure 2.3: Distribution of the data mixture in the overall pre-training mixture (left) and in the sampled subset used for MOLMOACT pre-training (right). The mixture contains primarily action reasoning data (38.7%), trajectory-conditioned data (38.7%), and multimodal web data (21.5%), with small fractions of auxiliary depth and trace data (0.5% each). The sampled subset increases the proportion of auxiliary data (7.5% each for depth and trace) while reducing multimodal web data to 5%.

thought (MCoT) [Lai and Nissim, 2024] 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 into VLAs, they focus on high-level language reasoning [Sun et al., 2024; Zawalski et al., 2024; Intelligence et al., 2025] 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.

In 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 autoregressively generates a sequence of *depth perception tokens*, followed by a *visual reasoning trace* of the intended end-effector motion, before predicting the action tokens.

Depth Perception Tokens. Depth estimation is a key aspect of spatial understanding and robot action prediction. Most conventional VLMs and VLAs condition their outputs solely on the RGB image input and text instruction. Lacking depth estimation, they fail on tasks that require spatial understanding. Prior work [Bigverdi et al., 2024] has shown that depth perception tokens are effective in enhancing chain-of-thought reasoning for visual–spatial tasks. Building on this insight, we leverage depth estimation as a key component to enable fine-grained robotics control in 3D environments. We now examine how each

intermediate step in the reasoning chain is formulated.

We begin by defining the auxiliary vocabulary that contains depth perception tokens. Let

$$V_{\text{depth}} = \{\langle \text{DEPTH_START} \rangle, \langle \text{DEPTH_END} \rangle\} \cup \{\langle \text{DEPTH_}k \rangle\}_{k=1}^{N} \tag{2.1}$$

be the discrete token set used to represent depth, with $N = 128$. For each input image, the target depth string is defined as

$$\mathbf{d} = \big(\langle \text{DEPTH_START} \rangle \langle \text{DEPTH_}z_1^{\text{depth}} \rangle \ \ldots \ \langle \text{DEPTH_}z_M^{\text{depth}} \rangle \langle \text{DEPTH_END} \rangle\big) \in V_{\text{depth}}, \tag{2.2}$$

with $M = 100$, and each $z_i^{\text{depth}} \in \{1, \ldots, N\}$ indexing a code in a VQVAE [Van Den Oord et al., 2017] codebook $\mathcal{C} = \{c_1, \ldots, c_N\}$. The codebook is produced by a pre-trained specialist depth estimator (trained on depth maps from Depth Anything V2), which quantizes the dense depth map into a fixed-length sequence of M indices. There is a deterministic one-to-one correspondence between each codebook index and a depth token in V_{depth} (i.e., index k maps to $\langle \text{DEPTH_}k \rangle$), so $\mathbf{d}$ is a discrete, explainable summary of the depth map of the scene. We employ a specialist-to-generalist distillation strategy to ground MOLMOACT's depth perception token prediction: the specialist produces $\mathbf{d}$ as the ground-truth depth string, and the VLA is trained to predict this string autoregressively from the original RGB observation, thereby internalizing depth in a form that can condition downstream trajectory and action generation.

Visual Reasoning Trace. Planning is a crucial component in robotics. Instead of planning through subtask decomposition in language, we predict an intermediate two-dimensional representation to better align visual inputs and control outputs across diverse robots and tasks. In particular, we generate the 2D trajectory of the end-effector and train the model to predict this trajectory alongside the next action command, a strategy shown to be effective in prior work [Li et al., 2025; Zheng et al., 2024a; Gu et al., 2023; Niu et al., 2024]. These predicted waypoints help align each action to precise end-effector locations, enabling the model to focus on fine-grained localization and thereby improving action-prediction accuracy. To achieve this, we introduce the visual reasoning trace.

Given an image observation, we define the end-effector visual reasoning trace on that image as a polyline

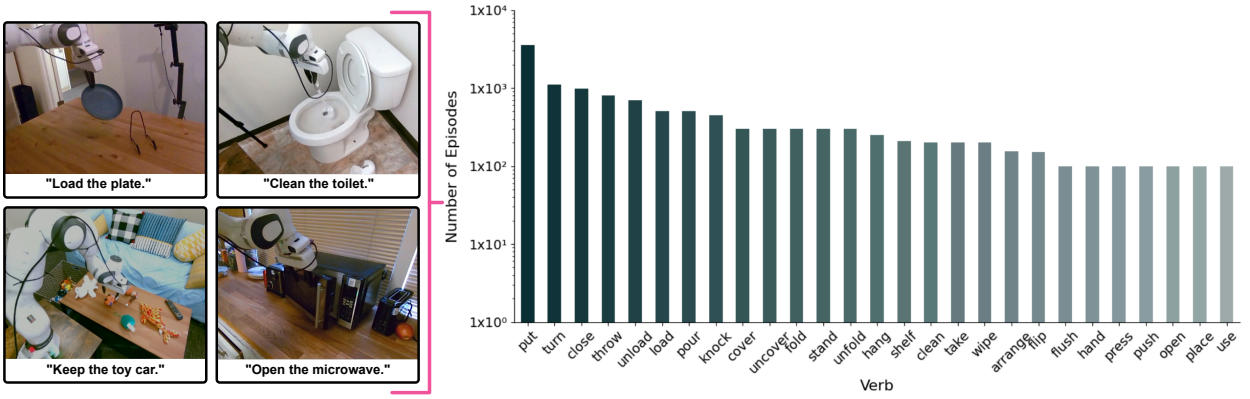


Figure 2.4: Examples and verb distribution in the MOLMOACT DATASET. Left: Sample robot manipulation tasks paired with natural language instructions, spanning diverse household activities such as closing a laptop, loading a plate, cleaning a toilet, and opening a microwave. Right: Log-scale distribution of the top verbs in the dataset, showing a long-tail pattern with "put," "turn," and "close" as the most frequent actions.

with L points, $1 \le L \le 5$ (i.e., 0 to 4 line segments),

$$\boldsymbol{\tau} = (p_1, \ldots, p_L), \qquad p_i = (u_i, v_i), \tag{2.3}$$

where for every $p_i = (u_i, v_i)$, the coordinates of the point are normalized with respect to the given image dimension so that $u_i, v_i \in \{0, \ldots, 255\}$.

Note that p_1 corresponds to the coordinate where the robot end-effector is located in the given image, and all other points are the locations of the end-effector in later frames. For the full polyline, points are subsampled evenly from the future horizon of the episode between the current frame and the terminal frame.

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 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 degrees of freedom. These are produced according to the factorization

$$p(\mathbf{d}, \boldsymbol{\tau}, \mathbf{a} \mid I, T) = \prod_{i=1}^{M+2} p(d_i \mid I, T, \mathbf{d}_{<i}) \times \prod_{j=1}^{L} p(\tau_j \mid I, T, \mathbf{d}, \boldsymbol{\tau}_{<j}) \times \prod_{k=1}^{D} p(a_k \mid I, T, \mathbf{d}, \boldsymbol{\tau}, \mathbf{a}_{<k}). \tag{2.4}$$

In other words, the depth string $\mathbf{d}$ is generated first, then the visual reasoning trace $\boldsymbol{\tau}$, and finally the action tokens $\mathbf{a}$. By conditioning each stage on the preceding depth and trajectory tokens, we ensure that the final actions are spatially grounded in both the inferred depth and the planned motion sketch.

2.3.4 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 $\boldsymbol{\tau} = (p_1, \ldots, p_L)$, $1 \le L \le 5$ (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:

$$p(\mathbf{a} \mid I^+, T) = \prod_{k=1}^{D} p(a_k \mid I^+, T, \mathbf{a}_{<k}). \tag{2.5}$$

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.

2.4 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, and 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.

2.4.1 Action Reasoning Data

As discussed in Section 2.3.3, MOLMOACT frames visuomotor control as an autoregressive sequence modeling problem augmented with an action chain-of-thought (CoT). This formulation allows us to convert any conventional robot action dataset into action reasoning data by appending predicted depth perception tokens and visual reasoning traces to action tokens, conditioned on both language and robot observations. This section details the process for generating ground-truth labels for the depth perception tokens and visual reasoning traces, and explains how these are combined with action labels to train MOLMOACT.

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 traces 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 its latent embedding. We then represent the latent embedding with a learned codebook of 128 dimensions 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 as ground-truth labels for our depth perception tokens.

Visual Reasoning Trace. To generate a *visual reasoning trace* 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 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 $(x_t, y_t) \in \mathbb{R}^2$ with $(x_t, y_t) \in [0, 100]$, 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 $\boldsymbol{\tau}$. In the case of bimanual robots, two separate prompts are issued per frame to obtain $\boldsymbol{\tau}_L$ and $\boldsymbol{\tau}_R$ for the left and right grippers, respectively. At each timestep t, we construct a visual reasoning trace by selecting a subsequence from t to the episode end e. This includes the current point (u_t, v_t), the final point (u_e, v_e), and up to three intermediate points spaced uniformly between them. If fewer than three intermediate points are available (i.e., $e - t < 4$), we include all available points. If $t = e$, the trace contains only one point. This yields a visual reasoning trace between 1 and 5 points representing the future motion of the end-effector.

Auxiliary Robot Data. To strengthen MOLMOACT's ability to reason in space, we extend the same data generation pipeline used for depth perception tokens and visual reasoning traces to curate three auxiliary supervision datasets: (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 sequentially 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.

2.4.2 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 2.4. The average length of each trajectory spans 112 timesteps. Data collection spanned two months and involved five full-time operators following strict protocols. The MOLMOACT DATASET includes manipulation data from two primary settings: home environments and tabletops.

- **Home Environment Data.** To collect diverse home environment data, we mounted a single-arm Franka robot on a lightweight, mobile platform similar to DROID [Khazatsky et al., 2024], 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. For example, the long-horizon task "clean up the dishes" was decomposed into subtasks such as "put the bowl in the dishwasher," "put the fork in the sink," and "cover the pot." This decomposition allows the policy to learn individual skill components in isolation before composing them into more complex behaviors. In total, we collected 7,730 trajectories spanning 73 distinct tasks and 20 verbs across a wide variety of scenes.

- **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. For example, the task "put the bowl in the dishwasher" consists of a sequence of motions such as opening the dishwasher, grasping the bowl, flipping it, and placing it inside. We isolated and separately collected data for each atomic motion—open, pick, flip, put, and close—to build a comprehensive set of motion primitives.

Table 2.1: SimplerEnv evaluation across different policies on Google Robot tasks. The zero-shot and fine-tuning results denote performance of OXE-dataset-pretrained models [O'Neill et al., 2024] and RT-1-dataset-fine-tuned models [Brohan et al., 2022], 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 [Wang et al., 2024a]	56.0%	60.0%	24.0%	46.0%		—		—
TraceVLA [Zheng et al., 2024a]	28.0%	53.7%	57.0%	42.0%	60.0%	56.4%	31.0%	45.0%
RT-1-X [Brohan et al., 2022]	56.7%	31.7%	59.7%	53.4%	49.0%	32.3%	29.4%	39.6%
RT-2-X [Zitkovich et al., 2023]	78.7%	77.9%	25.0%	60.7%	82.3%	79.2%	35.3%	64.3%
Octo-Base [Team et al., 2024b]	17.0%	4.2%	22.7%	16.8%	0.6%	3.1%	1.1%	1.1%
OpenVLA [Kim et al., 2024]	16.3%	46.2%	35.6%	27.7%	54.5%	47.7%	17.7%	39.8%
RoboVLM (zero-shot) [Liu et al., 2025]	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 [Sun et al., 2024]	2.3%	3.3%	18.3%	8.0%	5.3%	7.3%	20.5%	11.0%
Magma [Yang et al., 2025a]	56.0%	65.4%	83.7%	68.4%	53.4%	65.7%	68.8%	62.6%
π_0 (fine-tuned) [Black et al.]	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 N1.5 (fine-tuned) [NVIDIA et al., 2025]	69.3%	68.7%	35.8%	52.4%	46.7%	62.9%	17.5%	43.7%
SpatialVLA [Qu et al., 2025]	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%**

2.4.3 Multimodal Web Data

Prior works have shown that co-training VLAs with the data mixture from 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 [Goyal et al., 2017], Text VQA [Singh et al., 2019], OK-VQA [Marino et al., 2019], ChartQA [Masry et al., 2022], DocVQA [Mathew et al., 2021], Infographic VQA [Mathew et al., 2022], AI2D [Kembhavi et al., 2016], A-OKVQA [Schwenk et al., 2022], AndroidControl [Li et al., 2024c], ScienceQA [Lu et al., 2022], TabMWP [Lu et al., 2023], ST-VQA [Biten et al., 2019], TallyQA [Acharya et al., 2019], DVQA [Kafle et al., 2018], FigureQA [Kahou et al., 2017], and PlotQA [Methani et al., 2020]) for general visual skills and PixMo [Deitke et al., 2024] for fine-grained understanding and pointing. Furthermore, we include LVIS [Gupta et al., 2019], where the model is asked to predict the bounding box center of instances of a certain category to ground language to image regions.

2.5 Training Recipe

MOLMOACT is first pre-trained on action reasoning data curated from a subset of the OXE dataset, along with 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 2.2.

2.5.1 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 2.3. The whole data mixture used for pre-training MOLMOACT totals 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 according to the assigned sampling rates defined above.

2.5.2 Mid-training

Following the initial pre-training stage, we conduct a second stage of mid-training using high-quality action reasoning data closely aligned with our target domain of household manipulation. Specifically, we formulate the MOLMOACT DATASET into 1M action reasoning data samples and an additional 1M trajectory-conditioned action data samples, which we find beneficial for improving overall performance and action steering. To train on the MOLMOACT DATASET, we convert each sample—consisting of two side-mounted camera views and one wrist camera view, all sharing the same instruction and ground-truth action—into two paired-view training examples by pairing each side view with the wrist view. For action reasoning data preparation (§2.4.1), depth perception tokens and visual reasoning traces are generated only from the side views, while the wrist view is used solely for providing additional information. We train the model on this

Table 2.2: 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 autoregressive baselines, with strong performance across all categories, particularly in long-horizon tasks.

Baseline	Spatial	Object	Goal	Long	Avg
TraceVLA [Zheng et al., 2024a]	84.6%	85.2%	75.1%	54.1%	74.8%
Octo-Base [Team et al., 2024b]	78.9%	85.7%	84.6%	51.1%	75.1%
OpenVLA [Kim et al., 2024]	84.7%	88.4%	79.2%	53.7%	76.5%
SpatialVLA [Qu et al., 2025]	88.2%	89.9%	78.6%	55.5%	78.1%
CoT-VLA [Zhao et al., 2025]	87.5%	91.6%	87.6%	69.0%	83.9%
NORA-AC [Hung et al., 2025]	85.6%	89.4%	80.0%	63.0%	79.5%
WorldVLA [Cen et al., 2025]	87.6%	96.2%	83.4%	60.0%	79.1%
π_0-FAST [Black et al.]	96.4%	96.8%	88.6%	60.2%	85.5%
ThinkAct [Huang et al., 2025]	88.3%	91.4%	87.1%	70.9%	84.4%
MOLMOACT-7B-D	87.0%	95.4%	87.6%	77.2%	**86.6%**

modified dataset for 50k gradient steps with a batch size of 128, using 128 H100 GPUs over around 2,304 GPU hours.

2.5.3 Post-training

After mid-training, we conduct the final post-training stage to rapidly adapt the model to new tasks and embodiments. For new tasks, we collect a small set of 30 to 50 tele-operated demonstrations per task, then generate perception tokens and visual reasoning traces for each timestep. These demonstrations are converted into action reasoning data and trajectory-conditioned action data, following the same process as mid-training data curation, with one key difference: we apply action chunking [Zhao et al., 2023] during post-training, formatting action predictions in fixed-length chunks ($N = 8$). For each action chunk in all chunks, we tokenize each of them in the same way as we do for a single action. After grouping them into a list, we train the model autoregressively to predict all action chunks. We adapt MOLMOACT to post-train on single or multi-tasks via parameter-efficient LoRA fine-tuning. In all evaluations, we fix the LoRA rank at 32 and alpha at 16 to preserve the model's pre-trained capabilities. For simulation benchmarks (e.g., LIBERO), we use a batch size of 128, and for real-world tasks we use 64. The number of gradient steps we train varies by task. Our post-training data generally consists of a front- or side-view image paired with a wrist-view image, although some setups provide multiple wrist views (e.g., bimanual scenarios). In all cases, we apply the same LoRA and training configuration described above.

2.6 Experimental Evaluation

Our experimental evaluation comprises a broad suite of studies that rigorously benchmark MOLMOACT against strong baselines. We assess MOLMOACT (with the 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 the real world, we aim to answer the following research questions:

- **In-distribution performance.** How well does MOLMOACT perform, after pre-training, on tasks drawn from the same distribution as its training data? We benchmark MOLMOACT against strong VLA models on the SimplerEnv simulation benchmark [Li et al., 2024e].
- **Post-training adaptation.** How effectively does MOLMOACT adapt to novel tasks, domains, and embodiments through lightweight fine-tuning? We fine-tune MOLMOACT with LoRA [Hu et al., 2022], evaluate it on the LIBERO simulation suite [Liu et al., 2023a], and validate its real-world performance on single-arm and bimanual Franka platforms.
- **Out-of-distribution generalization.** How effectively does MOLMOACT generalize beyond its training distribution? We study this through real-world OOD evaluations and variant-aggregation experiments in SimplerEnv.
- **Effect of mid-training.** How does mid-training on MOLMOACT DATASET improve MOLMOACT's generalist performance? We answer this through real-world ablations comparing MOLMOACT with and without mid-training on MOLMOACT DATASET.
- **Language following.** 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 the resulting actions.
- **Steerability and interaction.** How steerable is MOLMOACT, and how can this steerability enhance user interaction? We perform real-world ablations by guiding MOLMOACT with sketched trajectory cues, analyzing its responses, and examining the resulting human–robot interaction dynamics.

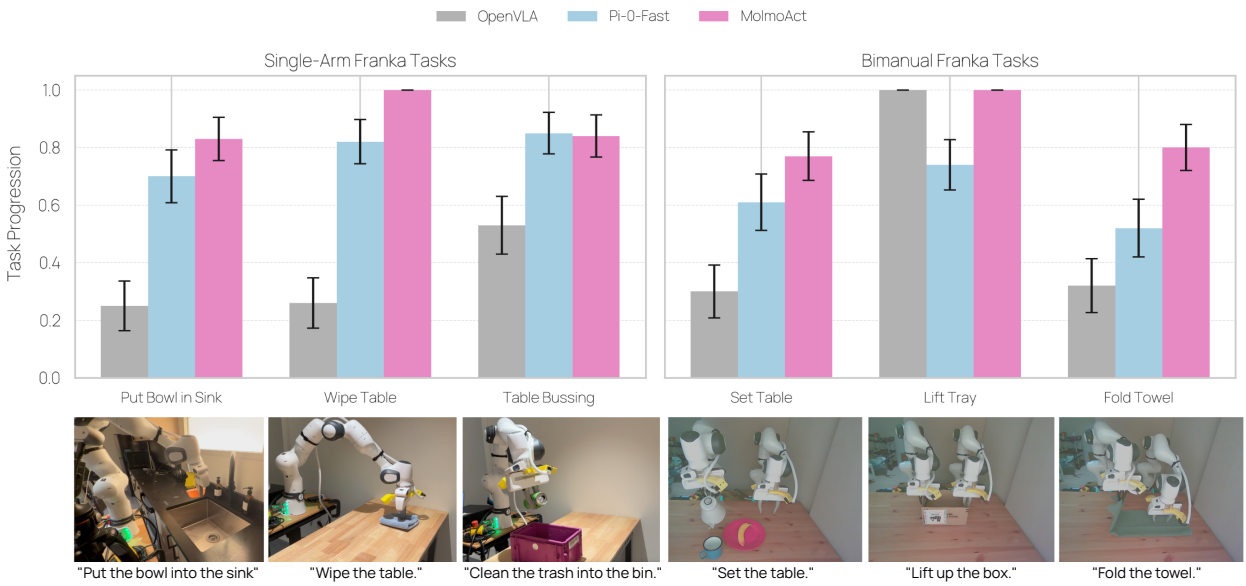


Figure 2.5: Real-world evaluation of OpenVLA, π_0-FAST, and MOLMOACT on single-arm (left) and bimanual (right) Franka tasks. Bar plots report average task progression with standard error across 25 trials per task. MOLMOACT consistently outperforms baselines, particularly on single-arm tasks such as *Wipe Table* and *Table Bussing*, and maintains strong performance on bimanual tasks including *Fold Towel* and *Set Table*. The bottom row shows example task setups with corresponding natural language instructions.

2.6.1 MOLMOACT After Pre-training

Evaluation Setup and Baselines. We first evaluated MOLMOACT's zero-shot capability—its ability immediately after pre-training and before any task-specific fine-tuning. Unlike π_0 [Black et al.], GR00T N1.5 [NVIDIA et al., 2025], and other proprietary VLA models that rely on large-scale private robot datasets and the full OXE dataset for pre-training, MOLMOACT was trained exclusively on curated action reasoning data filtered from a subset of OXE (specifically BC-Z [Jang et al., 2022a], BridgeData V2 [Walke et al., 2023], and RT-1 [Brohan et al., 2022]), combined with multimodal web data and auxiliary robot data, as detailed in §2.5.1. This amounts to approximately 26.3M samples—an order of magnitude smaller than π_0, which uses at least 903M samples for pre-training. To evaluate MOLMOACT's out-of-the-box generalization, we used the SimplerEnv benchmark, which features both visual-matching and variant-aggregation tasks across WidowX and Google Robot platforms. As MOLMOACT's pre-training distribution is most aligned with the Google Robot visual-matching tasks, we focused our evaluation on this suite to best isolate in-distribution performance and capabilities of pre-training.

We compared MOLMOACT-7B-D-PRETRAIN against a set of generalist policies, including TraceVLA [Zheng

et al., 2024a], RT-1X [Brohan et al., 2022], OpenVLA [Kim et al., 2024], RoboVLM [Liu et al., 2025], Emma-X [Sun et al., 2024], π_0 and π_0-FAST [Black et al.], Octo [Team et al., 2024b], Magma [Yang et al., 2025a], HPT [Wang et al., 2024a], SpatialVLA [Qu et al., 2025], and GR00T N1.5 [NVIDIA et al., 2025]. Most baselines were evaluated in the zero-shot setting, with a subset also tested after fine-tuning. We additionally fine-tuned MOLMOACT-7B-D-PRETRAIN on the RT-1 subset of OXE to assess its capacity when given more pre-training data.

Evaluation Results. MOLMOACT-7B-D-PRETRAIN achieved strong zero-shot performance on the SimplerEnv visual-matching suite, reaching 70.5% success rate and outperforming baselines such as GR00T N1.5, π_0, π_0-FAST, and Magma. With fine-tuning on the same RT-1 subset of OXE, MOLMOACT-7B-D improved to 71.6%, exceeding Magma by 3.2%, as shown in Table 2.1. These results indicate that MOLMOACT is both an effective zero-shot generalist and a strong initialization for fine-tuned deployment.

2.6.2 Fast Adaptation of MOLMOACT in Post-training

Evaluation Setups and Baselines. We evaluate MOLMOACT in both simulation and real-world settings to assess its fast adaptation after post-training. In simulation, we evaluate on the LIBERO simulation benchmark [Liu et al., 2023a], which consists of a Franka Emika Panda arm in simulation with demonstrations containing front and wrist view camera images (256×256 px), language instructions, and delta end-effector pose actions. We follow prior work [Kim et al., 2024] and evaluate on four task suites—LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long—each with 500 demonstrations across 10 tasks. Following Kim et al. [2024], we trained on a modified dataset that filtered out no-op actions and unsuccessful demonstrations. Moreover, we set action chunk size to $K = 8$ for evaluation on each task suite and execute full chunks before redoing action reasoning. We fine-tune MOLMOACT-7B-D using Low-Rank Adaptation (LoRA) and compare against state-of-the-art generalist autoregressive policies, such as TraceVLA [Zheng et al., 2024a], OpenVLA [Kim et al., 2024], SpatialVLA [Qu et al., 2025], π_0-FAST [Black et al.], CoT-VLA [Zhao et al., 2025], WorldVLA [Cen et al., 2025], ThinkAct [Huang et al., 2025], and NORA-AC [Hung et al., 2025].

In the real world, we evaluate MOLMOACT on six tasks across single-arm and bimanual Franka setups. The single-arm tasks include `put_bowl_in_sink`, `wipe_table`, and `table_bussing`. The biman-

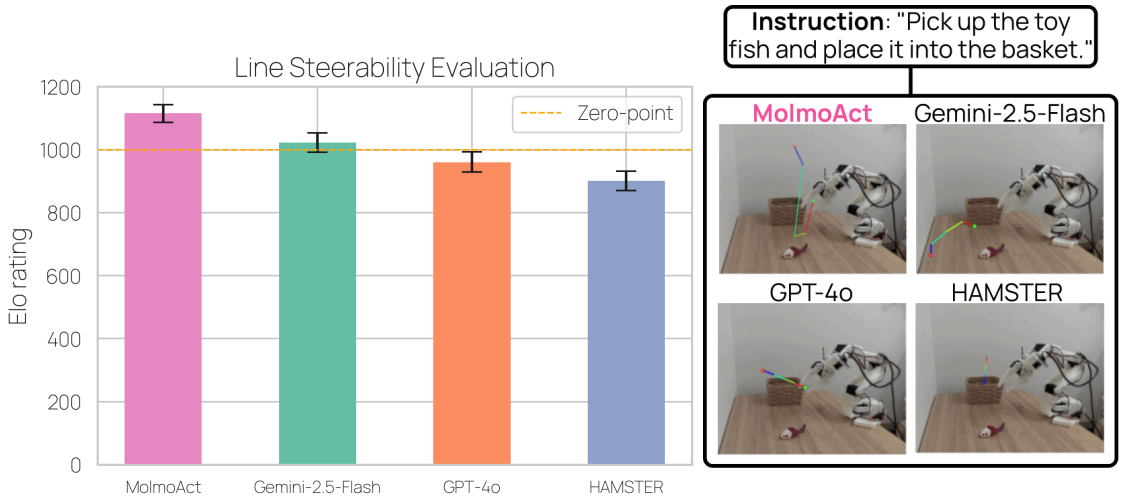


Figure 2.6: Line steerability evaluation across models. Left: Elo ratings show that MOLMOACT achieves the highest performance, surpassing Gemini-2.5-Flash, GPT-4o, and HAMSTER, with error bars indicating 95% confidence interval (CI). **Right:** Example qualitative results showing predicted visual traces overlaid on robot camera views.

ual tasks include `set_table`, `lift_tray`, and `fold_towel`. For each task, we collected 50 human tele-operated demonstrations and post-trained both MOLMOACT-7B-D and baseline models. We evaluate task progress over 25 trials per task. This setup enables a comprehensive comparison of adaptation efficiency across tasks and embodiments.

Evaluation Results. On the LIBERO benchmark, MOLMOACT-7B-D achieves an average success rate of 86.6%, the highest among all compared methods. It performs particularly well on LIBERO-Long, a challenging long-horizon suite, where it exceeds the performance of ThinkAct—the second-best method in this setting—by 6.3%. In the real world, MOLMOACT demonstrates effective fine-tuning and generalization across different embodiments. It outperforms π_0-FAST by an average of 10% in task progression on single-arm tasks and by 22.7% on bimanual tasks, as shown in Figure 2.5.

2.6.3 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,

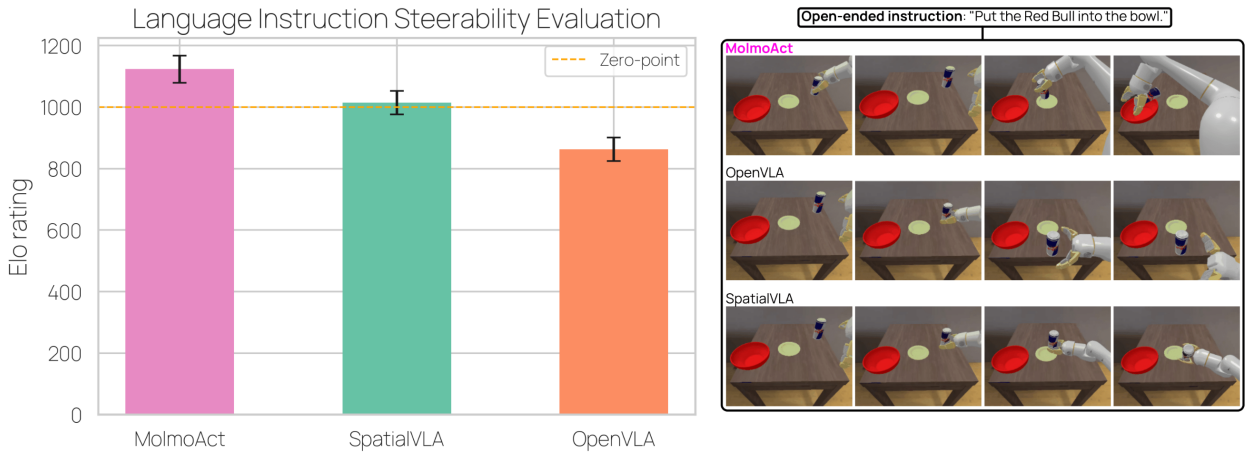


Figure 2.7: Language Instruction Evaluation. Left: Elo ratings for three models based on human votes in a head-to-head instruction-following evaluation. **Right:** Qualitative comparison of execution traces for the open-ended instruction "Put the redbull into the bowl." MOLMOACT aligns more closely with the intended instruction than other models.

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.

Evaluation Results. In simulation, fine-tuned MOLMOACT-7B-D-PRETRAIN achieves 72.1% on the variant-aggregation tasks, as shown in Table 2.1, outperforming all baselines and exceeding the second-best model, RT-2-X, by 7.8%. The performance difference between variant aggregation and visual 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.

2.6.4 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.

Evaluation Results. 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.

2.6.5 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 2.7). 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.

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 MOLMOACT-7B-D-PRETRAIN winning over SpatialVLA in 58%

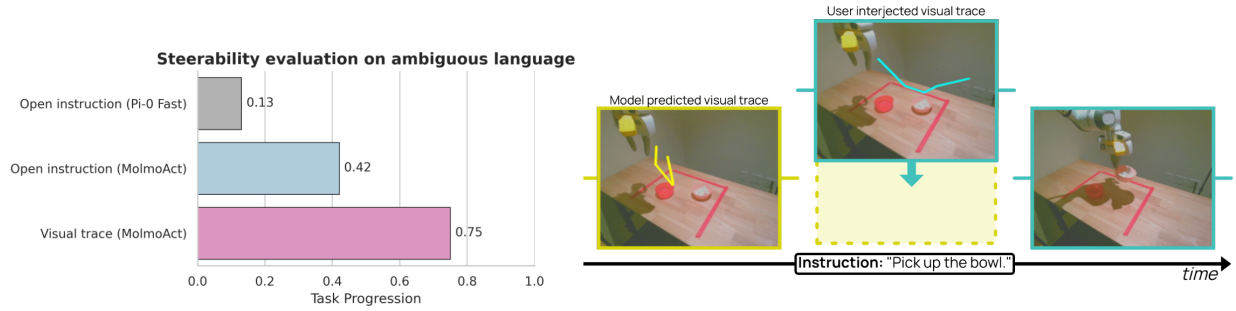


Figure 2.8: Steerability evaluation with open instructions and visual traces. Left: Success rates for different steering modes, showing that MOLMOACT with visual trace steering achieves the highest success rate (0.75), outperforming its open-instruction variant and π_0-FAST. **Right:** Example of the *"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.

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 2.7. 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 2.6.

2.6.6 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 2.8. 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.

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.

2.7 Conclusion

We introduced MOLMOACT, a family of fully open action reasoning models that integrate perception, planning, and control by reasoning in space. By combining depth perception tokens, visual reasoning traces, and action prediction, MOLMOACT produces explainable, spatially coherent behaviors. The behaviors can be executed directly, or steered via trajectory editing. Our evaluations across simulation and real-world settings demonstrate that MOLMOACT consistently outperforms strong vision–language–action baselines, adapts efficiently to novel tasks and embodiments through lightweight fine-tuning, and generalizes robustly to out-of-distribution conditions. We release all model weights, code, and data, including the MOLMOACT DATASET, to enable reproducibility and foster community-driven research toward building foundation models that transform perception into purposeful action through structured reasoning.

Chapter 3

Temporal Episodic Reasoning: Storing and Representing Memory for Robotics Foundation Models

3.1 Introduction

The world in which we live is diverse and constantly changing, encompassing a wide variety of objects, scenes, and environmental conditions. Consider the seemingly simple task of following a recipe when cooking: we can seamlessly perform the action of picking up salt and sprinkling it into the pan, recognize salt even if it comes in different types of containers, and remember whether we have already added salt. Humans excel in such environments because they can interact with their surroundings to achieve specific goals, generalize to unseen scenarios, and retain knowledge from past experiences [Smith and Gasser, 2005]. These abilities—multitask interaction, generalization, and memory—serve as guiding principles for developing robotic systems capable of operating in similarly complex environments.

Significant progress has been made in robotic manipulation through prior work. Early methods, such as the Transporter Network [Zeng et al., 2021] and CLIPort [Shridhar et al., 2022], demonstrated effective 2D action-centric manipulation but were limited in their ability to handle spatially complex tasks. More recent approaches, such as PerAct [Shridhar et al., 2023] and RVT [Goyal et al., 2023], have pushed toward

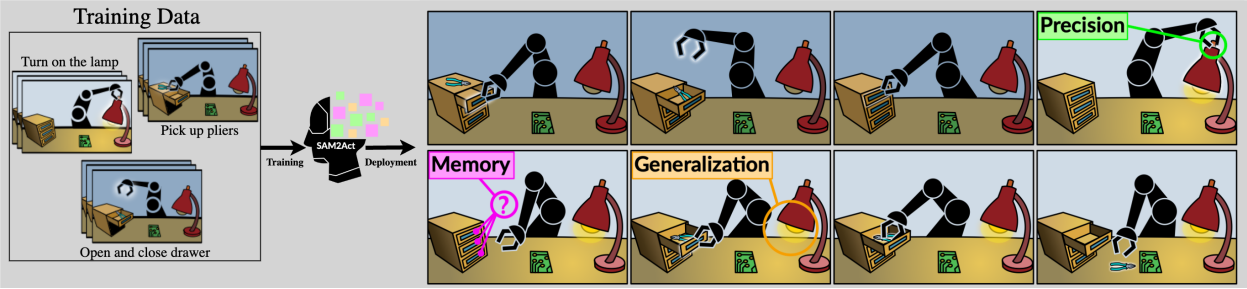


Figure 3.1: SAM2Act is a multi-view, language-conditioned behavior cloning policy trained with fewer demonstrations. Given a language instruction, it can execute high-precision tasks, such as turning the tiny knob on the lamp. It also generalizes to various environmental variations, such as changes in lighting conditions. Through further training with our proposed memory architecture, it now evolves into SAM2Act+, which is capable of solving tasks that require implicit spatial memory—such as remembering where the robot previously stored the pliers, as depicted in the figure.

3D-based manipulation. PerAct employs a multitask transformer that interprets language commands and predicts keyframe poses, achieving strong results across a variety of tasks. RVT builds on this foundation by adopting a 2.5D representation, improving training efficiency and inference speed. Its successor, RVT-2, further enhances performance with a coarse-to-fine strategy, increasing precision for high-accuracy tasks. Despite these advances, important challenges remain, including improving multitask performance, enhancing generalization to novel environment configurations, and integrating memory mechanisms for tasks requiring episodic recall.

In this chapter, we introduce SAM2Act, a multi-view robotics transformer-based policy that enhances feature representation by integrating multi-resolution upsampling with visual embeddings from large-scale foundation models. Built on the RVT-2 multi-view transformer, SAM2Act achieves strong multitask success and generalization. Building on this foundation, we introduce SAM2Act+, which incorporates a memory-based architecture inspired by SAM2's approach. Using a memory bank, an encoder, and an attention mechanism, SAM2Act+ enables episodic recall to solve spatial memory-dependent manipulation tasks. We evaluate SAM2Act and SAM2Act+ using `MemoryBench`, a new benchmark suite that tests policies' spatial memory capabilities and the ability to retain and recall past actions. SAM2Act+ achieves an average success rate of 94.3% across all tasks on `MemoryBench`, outperforming the next highest baseline by a margin of 39.3%. Furthermore, we assess the generalization capabilities of SAM2Act on `The Colosseum` [Pumacay et al., 2024], a benchmark designed to test robotic manipulation under various environmental perturbations.

SAM2Act demonstrates robust performance on `The Colosseum` with an average decrease of only 4.3% across all perturbations, highlighting its ability to generalize effectively in diverse and challenging scenarios. Lastly, our approach outperforms baseline methods in real-world evaluations while exhibiting comparable generalization and spatial memory capabilities.

In summary, this chapter makes three key contributions. First, we introduce a **novel model formulation** that leverages visual foundation models to solve **high-precision, memory-dependent manipulation tasks**. Second, we propose `MemoryBench`, an evaluation benchmark for **assessing spatial memory in behavior cloning models**. Finally, we present **empirical results and insights** on the model's performance across both simulation and real-world tasks.

3.2 Related Work

3.2.1 3D-Based Robotic Transformers for Manipulation

2D-based methods [Zhao et al., 2023; Chi et al., 2023; Zeng et al., 2021; Brohan et al., 2022; Shridhar et al., 2022] are effective for simple pick-and-place tasks due to fast training, low hardware requirements, and minimal computational cost. However, they depend on pretrained image encoders and fail in tasks requiring high precision, robust spatial interaction, or resilience to environmental and camera variations [Pumacay et al., 2024]. Recent work addresses these limitations with 3D perception. Methods like PolarNet [Chen et al., 2023], M2T2 [Yuan et al., 2023], and Manipulate-Anything [Duan et al., 2024b] reconstruct point clouds, while C2F-ARM [James and Abbeel, 2022] and PerAct [Shridhar et al., 2023] use voxel-based 3D representations. Act3D [Gervet et al., 2023] and ChainedDiffuser [Xian et al., 2023] adopt multi-scale 3D features. RVT [Goyal et al., 2023] introduces 2.5D multi-view images for faster training, refined by RVT-2 [Goyal et al., 2024] with a coarse-to-fine architecture for improved precision. Our work, SAM2Act, combines RVT-2's spatial reasoning with enhanced virtual images from the SAM2 visual encoder, achieving high precision and generalization across diverse tasks.

3.2.2 Visual Representations for Robot Learning

Robotics research heavily relies on visual representations from computer vision to process high-dimensional inputs and improve policy learning. Visual representations are integrated into robot learning through pre-training [Majumdar et al., 2023; Ma et al., 2022; Nair et al., 2022], co-training [Laskin et al., 2020b; Yarats et al., 2021; Laskin et al., 2020a; Shang et al., 2024], or frozen encoders [Shah and Kumar, 2021; Wang et al., 2022a; Zhang et al., 2024a], all of which effectively support policy training. These representations also enhance invariance, equivariance, and out-of-distribution generalization [Wang et al., 2022b; Pumacay et al., 2024; Dasari et al., 2023]. Notably, object-centric visual representations (e.g., from SAM) are shown to be even more useful and relevant for robotic manipulation and control [Shi et al., 2024a; Qian et al., 2024]. Specifically, SAM-E [Zhang et al., 2024a] demonstrates the use of a pre-trained SAM encoder for robotic manipulation by leveraging image embeddings for policy learning. Expanding on this, our approach employs the SAM2 visual encoder to generate image embeddings for robotic transformers and utilizes its multi-resolution features to improve convex upsampling for next-action prediction.

3.2.3 Memory in Robotics

Memory is a fundamental component of human cognition, and equipping generalist robotic agents with episodic and semantic memory is crucial for enabling them to perform complex tasks effectively [Jockel et al., 2008]. Early research on memory in robotics primarily addressed navigation tasks, relying on semantic maps that were often constrained in scope [Henry et al., 2012; Bowman et al., 2017; Chaplot et al., 2020]. Other work explicitly models the memory and its representation for a robot cognitive architecture [Peller-Konrad et al., 2023]. Recent advancements leverage representations derived from vision–language models (VLMs) and large vision models (LVMs), utilizing voxel maps or neural feature fields to encode, store, and retrieve information [Huang et al., 2024a, 2023; Duan et al., 2024b; Liu et al., 2024c]. Alternative methods represent semantic memory for manipulation tasks using Gaussian splats to encode spatial information [Kerbl et al., 2023; Shorinwa et al., 2024]. Recent work [Huang et al., 2024c] employs transformer-based relational memory on partial-view point clouds—augmented with object discovery and tracking—to robustly handle occlusions, novel and reappearing objects, and diverse distractors, outperforming implicit-memory baselines in both simulation and real-world experiments. In contrast, our approach draws inspiration from the framework

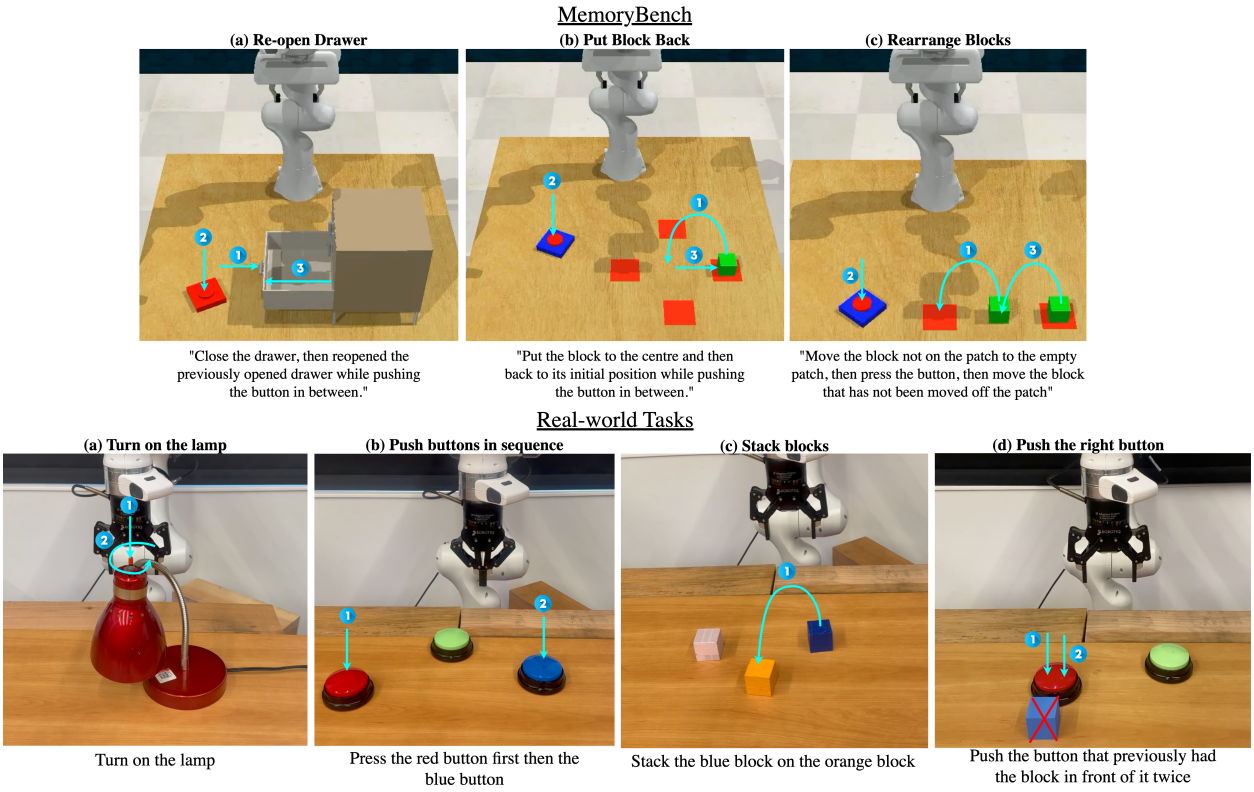


Figure 3.2: Simulation and real tasks. We demonstrate the effectiveness of SAM2Act+ in solving memory-based tasks by evaluating it against baselines on the three benchmark memory tasks (top). Additionally, we validate our approach using a Franka Panda robot on four real-world tasks (bottom), including tests under out-of-distribution perturbations.

of Partially Observable Markov Decision Processes (POMDPs) [Lauri et al., 2022], incorporating memory directly into the training process. By integrating spatial memory from past actions into the agent's belief state, we enhance the robustness and adaptability of learned policies.

3.3 `MemoryBench`: A Memory Benchmark for Robotic Manipulation

We introduce `MemoryBench`, a benchmark designed to systematically evaluate the spatial memory capabilities of robotic manipulation policies. In §3.3.1, we begin by outlining the logic and rules behind task design. We then describe the tasks we have developed in §3.3.2.

3.3.1 Task Design

Unlike standard RLBench tasks [James et al., 2020], many of which involve long-horizon scenarios, our tasks are specifically designed to require spatial memory. Without such memory, the agent would be forced to rely on random actions. To create these tasks, we intentionally violate the Markov assumption, which states that in a Markov Decision Process (MDP), the next observation depends solely on the current observation and action:

$$P\big(o_{t+1} \mid o_1, a_1, \ldots, o_t, a_t\big) \;=\; P\big(o_{t+1} \mid o_t, a_t\big). \tag{3.1}$$

This assumption implies that knowing only o_t and a_t is sufficient to predict o_{t+1}. However, in our tasks, we design scenarios where two distinct action histories lead to the same observation o_t, but require different subsequent actions. This forces the agent to recall which action history led to o_t to perform the correct next action. Furthermore, we standardized the language instructions to prevent unintentional leakage of spatial information that could aid the model in memory-based tasks. These principles guided the development of our spatial memory-based tasks.

3.3.2 Spatial Memory-Based Tasks

`MemoryBench` extends the RLBench simulator to provide scripted demonstrations for three spatial memory tasks: `reopen_drawer`, `put_block_back`, and `rearrange_block`. Each task is designed to evaluate a specific aspect of spatial memory and adheres to the principles outlined in §3.3.1. To introduce complexity, these tasks include two to four variations and additional steps—such as pressing a button mid-sequence—that disrupt the Markov property. This forces the agent to rely on memory rather than solely on immediate observations.

The `reopen_drawer` task evaluates the agent's ability to recall 3D spatial information along the z-axis. Initially, one of three drawers (top, middle, or bottom) is open. The agent must close the open drawer, press a button on the table, and then reopen the same drawer. After the button is pressed, all drawers are closed, and the scene becomes visually indistinguishable, requiring the agent to use memory to identify the correct drawer. This task tests the agent's ability to recall spatial states over a temporal sequence. The `put_block_back` task tests the agent's ability to remember 2D spatial information on the x–y plane. Four red patches are

placed on a table, with a block initially positioned on one of them. The agent should move the block to the center of the patches, press a button, and return the block to its original position. The agent must rely on its memory of the block's initial location to succeed, demonstrating its capability to encode and retrieve 2D spatial information.

The `rearrange_block` task evaluates the agent's ability to perform backward reasoning by recalling and reversing prior actions. Initially, one block is placed on one of two red patches, while the other patch remains empty. A second block is positioned at the center of both patches. The agent must move the second block to the empty patch, press a button, and then relocate the first block off its patch. Successfully completing this task requires the agent to determine which block to move without having interacted with the correct one in previous actions, thereby testing its capacity for backward spatial memory reasoning. These tasks collectively evaluate both forward and backward spatial reasoning across 3D (z-axis) and 2D (x–y plane) spaces. By introducing non-Markovian elements, they emphasize the need for memory representations to solve complex sequential decision-making problems.

3.4 Method

Our method, SAM2Act, enables precise 3D manipulation with strong generalization across environmental and object-level variations. Building upon the RVT-2 framework [Goyal et al., 2024], SAM2Act introduces key architectural innovations that enhance visual feature representation and task-specific reasoning. The architecture reconstructs a point cloud of the scene, renders it from virtual cameras at orthogonal views, and employs a two-stage multi-view transformer (coarse-to-fine) to predict action heatmaps. The coarse branch generates zoom-in heatmaps to localize regions of interest, while the fine branch refines these into precise action heatmaps. SAM2Act leverages the pre-trained SAM2 encoder [Ravi et al., 2024] to extract multi-resolution image embeddings, which are further refined through the multi-resolution upsampling technique to predict accurate translation heatmaps with minimal information loss. To address tasks requiring spatial memory, SAM2Act+ extends the SAM2Act architecture by incorporating memory-based components. These include a Memory Bank, Memory Encoder, and Memory Attention, enabling the model to encode historical actions and condition current observations. This memory-based policy enhances the agent's ability to predict actions based on past contextual information, significantly improving performance in tasks that

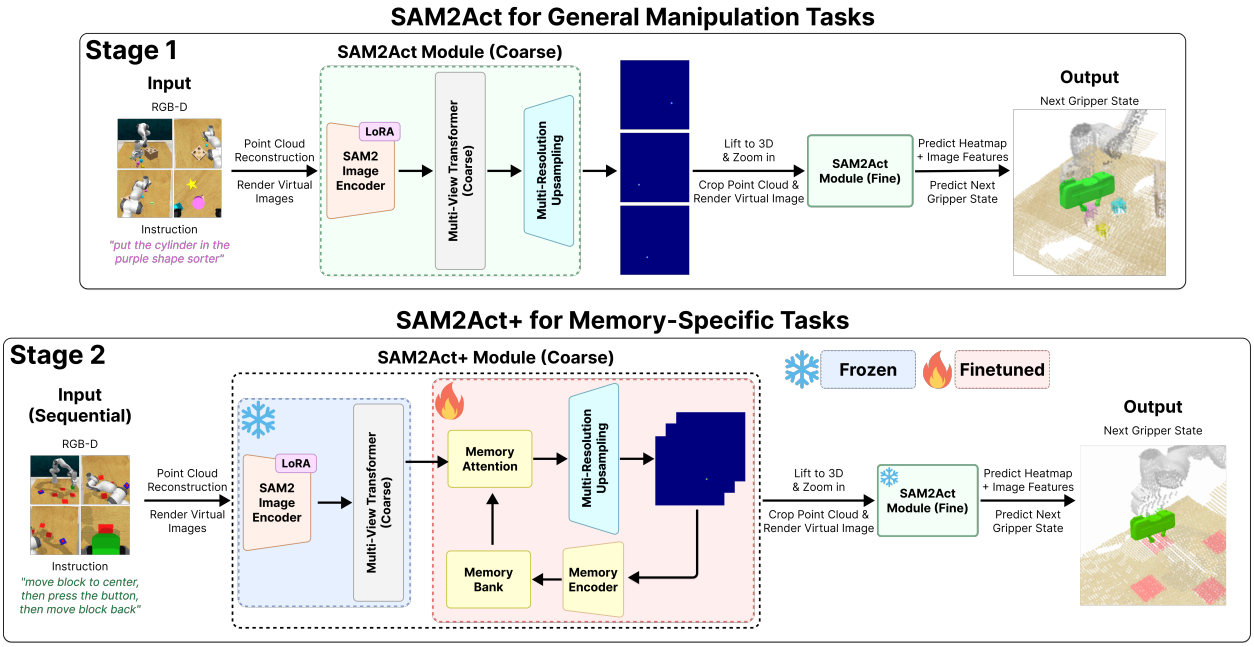


Figure 3.3: Overview of the SAM2Act (top) and SAM2Act+ (bottom) architectures. The SAM2Act architecture leverages the SAM2 image encoder to generate prompt-conditioned, multi-resolution embeddings, fine-tuned with LoRA for efficient adaptation to manipulation tasks. A multi-view transformer aligns spatial coordinates with language instructions, while a cascaded multi-resolution upsampling mechanism refines feature maps and generates accurate translation heatmaps. SAM2Act+ extends this architecture by incorporating memory-based components, including the Memory Encoder, Memory Attention, and Memory Bank, into the coarse branch. These components enable memory-driven reasoning by processing historical heatmaps and integrating prior observations, allowing the agent to predict actions based on stored contextual information. Observations are reconstructed into point clouds, rendered into three virtual images, and lifted into 3D translation points, enabling precise spatial reasoning across both architectures.

require sequential decision-making.

In the following sections, we detail the SAM2Act architecture (§3.4.1), including its multi-resolution upsampling mechanism (Figure 3.4). We also present the SAM2Act+ extension, which integrates memory-based components for solving spatial memory tasks (§3.4.2).

3.4.1 SAM2Act: Multi-Resolution Upsampling for Enhanced Visual Feature Representation

A distinctive feature of SAM2Act is the incorporation of the SAM2Act module into the manipulation backbone for training, as illustrated in Figure 3.4. The coarse and fine SAM2Act modules share the same architecture, with the fine branch generating additional features to predict actions beyond translation, while the coarse branch focuses exclusively on translation. Point-cloud representations are reconstructed from

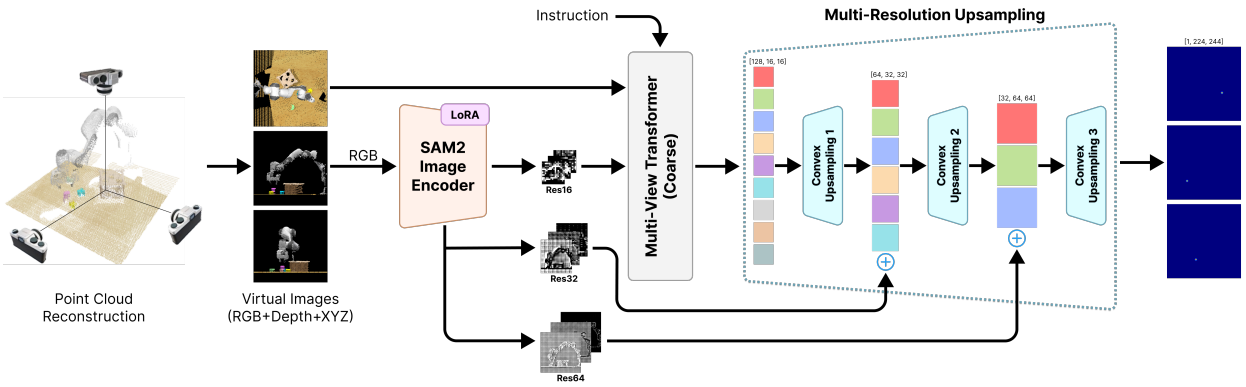


Figure 3.4: SAM2Act module and multi-resolution upsampling mechanism. A cascade of three convex upsamplers processes feature maps at increasing resolutions, integrating multi-resolution embeddings from the SAM2 image encoder through elementwise addition and layer normalization. The upsamplers progressively refine features, doubling spatial dimensions at each stage, to generate accurate translation heatmaps while capturing fine-grained spatial details critical for manipulation tasks.

raw image inputs, and virtual images are generated from three viewpoints using virtual cameras. Instead of directly inputting these images into the multi-view transformer, their RGB channels are duplicated and processed by the SAM2 image encoder [Ravi et al., 2024], which produces object-centric multi-resolution embeddings. These embeddings, generated at three resolution levels, are combined with virtual images containing RGB, depth, 3D translation coordinates, and language instructions before being fed into the multi-view transformer.

To adapt the SAM2 image encoder to our domain, we fine-tune it using Low-Rank Adaptation (LoRA) [Hu et al., 2021] with a default rank of 16, which enables domain adaptation with minimal computational cost while maintaining model efficiency. Additionally, to fully leverage the multi-resolution embeddings produced by the SAM2 image encoder, we introduce a multi-resolution upsampling method. This method uses the embeddings as auxiliary inputs to enhance the generation of translation heatmaps, thereby improving spatial precision and overall system performance. The multi-resolution upsampling mechanism, also detailed in Figure 3.4, leverages cascaded convex upsamplers to progressively refine feature maps across resolutions. Let $X^l \in \mathbb{R}^{B\times C^l\times H^l\times W^l}$ denote the feature maps at stage l and $E^l \in \mathbb{R}^{B\times C^l\times H^l\times W^l}$ the corresponding multi-resolution embedding from SAM2. Also let $U(\cdot)$ denote the upsampling operator that doubles the spatial dimensions. The feature maps are updated at each stage as follows:

$$X^{l+1} = \text{LayerNorm}\big(U(X^l) \oplus E^l\big), \tag{3.2}$$

Algorithm 1 Forward Pass of SAM2Act+ Module

Initialize: Number of steps N, maximum number of memories M, number of views V, empty memory bank Q with V separate FIFO queues, input X
for i = 1 **to** N **do**
  for j = 1 **to** V **do**
    Get embeddings $\mathcal{E}_{raw}$ from MVT $T_{mv}(X_j)$
    Retrieve past memories $\mathcal{M}_{old}$ from $Q[j]$
    Get memory-conditioned embeddings $\mathcal{E}_{mem}$ from Memory Attention $T_{mem}(\mathcal{E}_{raw}, \mathcal{M}_{old})$
    Predict translation heatmap $\mathcal{H}$ with upsampler $U(\mathcal{E}_{mem})$
    Encode new memory $\mathcal{M}_{new}$ using Memory Encoder $E_{mem}(\mathcal{H}, \mathcal{E}_{raw})$
    Store new memory $Q[j] \leftarrow Q[j] \cup \{\mathcal{M}_{new}\}$
    if $|Q[j]|$ = M **then**
      $Q[j] \leftarrow Q[j]_{2:n}$
    end if
  end for
end for

where $\oplus$ represents element-wise addition. The upsampling operator U is defined as:

$$U : \mathbb{R}^{B\times C^l\times H^l\times W^l} \rightarrow \mathbb{R}^{B\times (C^l/2)\times (2H^l)\times (2W^l)}. \tag{3.3}$$

At each stage, the output of the upsampler is combined with the corresponding multi-resolution embedding E^l from the SAM2 encoder, ensuring alignment between the multi-resolution features and the decoder's spatial refinement process. A layer normalization step follows each addition to stabilize training and maintain feature coherence. This results in direct integration of the embeddings into the translation heatmap generation process. The cascading structure refines features across multiple resolutions, capturing fine-grained spatial details critical for manipulation tasks.

3.4.2 SAM2Act+: Action Memory Architecture for Improved Spatial Awareness in Past Observations

To extend the SAM2Act architecture (§3.4.1) with memory-based capabilities inspired by SAM2, we introduce SAM2Act+, a task-specific variant designed for solving memory-based tasks. SAM2Act+ integrates the three core memory components from SAM2—*Memory Attention*, *Memory Encoder*, and *Memory Bank*—into the coarse branch of SAM2Act. Originally developed for object tracking in SAM2, these components are adapted to align with the needs of SAM2Act+, enabling the agent to retain prior actions and observations

for sequential decision-making. In SAM2, the Memory Encoder processes predicted object masks, while the Memory Attention module fuses image embeddings with positional information from previous frames. SAM2Act+ adopts a similar structure: the predicted heatmaps, which serve as binary indicators of spatial positions in the image, function analogously to object masks. This conceptual alignment ensures a seamless integration of memory mechanisms, allowing the agent to leverage stored information to predict subsequent actions based on historical context.

Architecture. The SAM2Act+ architecture is illustrated in Figure 3.3. After pretraining SAM2Act in Stage 1, we freeze the SAM2 image encoder and the multi-view transformer in the coarse branch, as these components effectively generate robust embeddings for multi-view images in manipulation tasks. We also freeze the entire fine branch, given its proven ability to predict fine-grained actions accurately. The reason we only fine-tune the coarse branch is that it focuses on generating heatmaps that provide richer contextual information for recalling past actions. The fine branch, in contrast, primarily emphasizes small objects or localized regions, which typically contain less information relevant to memory-based tasks.

Training. To train SAM2Act+, we fine-tune the coarse branch by integrating the three memory components (and train them from scratch) with the multi-resolution upsampling module. During fine-tuning, consecutive action keyframes are sampled as input, training the multi-resolution upsampler to predict new translations conditioned on memory. The memory components function similarly to their implementation in SAM2 for object tracking, with one key distinction: the input to the Memory Encoder. Instead of using image embeddings from the SAM2 image encoder, we input feature embeddings generated by the multi-view transformer (not conditioned on memory). This adaptation ensures that memory encoding incorporates multi-view information while maintaining independence in handling stored representations. Virtual images are treated independently during memory encoding and attention, with each view's memory encoded separately. Feature embeddings from each view are attended to using their corresponding stored memories, preserving spatial and contextual alignment while leveraging fused multi-view information. This structured approach prevents cross-view interference and enhances the model's ability to reason over sequential tasks. The memory-based forward pass for SAM2Act+ is outlined in Algorithm 1. By incorporating the memory mechanism, SAM2Act+ enhances performance in scenarios requiring long-term reasoning, enabling the

agent to make informed decisions based on historical context.

3.5 Experiments

We study SAM2Act and SAM2Act+ in both simulated and real-world environments. Specifically, we are interested in answering the following questions:

§3.5.2 How does SAM2Act compare with state-of-the-art 3D manipulation policies?

§3.5.3 Can SAM2Act generalize across object and environmental perturbations?

§3.5.4 Can SAM2Act+ solve spatial memory-based tasks that other baselines cannot?

§3.5.5 How well do SAM2Act and SAM2Act+ perform on real-world tasks?

3.5.1 Experimental Setup

We benchmark SAM2Act in both simulated and real-world environments. The simulated environments serve as a controlled platform to ensure reproducible and fair comparisons. The real-world experiments demonstrate the applicability of the method to real-world settings.

Simulation Setup. All simulated experiments were conducted in the CoppeliaSim environment via PyRep, using a 7-DoF Franka Emika Panda robot in a tabletop setting. Observations were captured from five RGB-D cameras—front, left shoulder, right shoulder, overhead, and wrist—each at $128\,\mathrm{px} \times 128\,\mathrm{px}$. The robot receives a keyframe specifying translation and quaternion orientation and utilizes an OMPL-based motion planner to move to the target pose.

Real-Robot Setup. We validate SAM2Act in real-world scenarios using a Franka Emika Panda robot with a Robotiq 2F-85 gripper and an exocentric Intel RealSense D455 depth sensor. We study four manipulation tasks, aligning three with RVT-2 for comparison and introducing a new memory-based task. We use the software stack as in Grotz et al. [2024]. For each task, we collect 10–15 demonstrations via kinesthetic teaching and scripted execution with scene and object variations. As shown in Figure 3.2, we evaluate SAM2Act against RVT-2 for tasks (a)–(c) and SAM2Act+ for the memory task (d). Each task undergoes

10 in-distribution and 10 out-of-distribution trials, including environmental perturbations, measuring total success.

18 RLBench & `MemoryBench` Tasks. To evaluate the general performance of SAM2Act and the memory capabilities of SAM2Act+, we conducted simulation experiments on two benchmarks: a subset of 18 tasks from RLBench and `MemoryBench`. RLBench is a standard multi-task manipulation benchmark, from which we selected 18 tasks well-studied in prior work. `MemoryBench` is a curated set of three tabletop manipulation tasks in CoppeliaSim that require the trained policy to have both semantic and spatial memory of past scenes and actions. In both benchmarks, each task is defined by a language instruction with 2–60 variations (e.g., handling objects, locations, and colors). We collected 100 demonstrations per task for training and held out 25 unseen demonstrations per task for testing. All policies are evaluated four times to obtain standard deviations.

3D Baselines. We benchmark SAM2Act and SAM2Act+ against the current state-of-the-art 3D next-best-pose prediction model, RVT-2. RVT-2 is a multi-view robotics transformer that leverages a coarse-to-fine approach on the constructed point cloud to predict the next best action heatmap. We also compare with RVT [Goyal et al., 2023], PerAct [Shridhar et al., 2023], and SAM-E [Zhang et al., 2024a].

3.5.2 Performance Across 18 RLBench Tasks

Table 3.1 compares SAM2Act with prior keyframe-based 3D BC methods on the RLBench benchmark. Overall, SAM2Act achieves an average success rate of $86.8\% \pm 0.5$, surpassing the previous best (RVT-2) by 5.4%. A closer look at individual tasks reveals that SAM2Act ranks **first in 9 out of 18 tasks** and remains highly competitive in **7 others**, coming within **one successful attempt or** 4% of the best performance. These tasks include Close Jar, Drag Stick, Meat Off Grill, Place Wine, Screw Bulb, Sweep to Dustpan, and Turn Tap. The largest margin of improvement occurs in `Insert Peg`, where SAM2Act **exceeds RVT-2 by** 44% **(approximately** $2.1\times$**)**, and in `Sort Shape`, where it outperforms RVT-2 by 29%. Both tasks require precise manipulation, underscoring the effectiveness of SAM2Act's multi-resolution upsampling strategy. These results establish SAM2Act as a **leading policy for complex 3D tasks**, highlighting its ability to handle high-precision manipulations—an area where prior methods have struggled.

Table 3.1: Multi-task performance on RLBench. We evaluate 18 RLBench tasks [James et al., 2020], reporting success rates across all tasks among 3D keyframe-based behavior cloning (BC) policies. We report stats over 4 evaluations for SAM2Act. Our method, SAM2Act, outperforms all baselines, achieving a significant performance margin of 5.8% over RVT-2 [Goyal et al., 2024], the prior state-of-the-art 3D keyframe-based BC policy.

Method	Avg. Success ↑	Avg. Rank ↓	Close Jar	Drag Stick	Insert Peg	Meat off Grill	Open Drawer	Place Cups	Place Wine	Push Buttons
PerAct [Shridhar et al., 2023]	49.4 ± 4.3	4.6	55.2 ± 4.7	89.6 ± 4.1	5.6 ± 4.1	70.4 ± 2.0	88.0 ± 5.7	2.4 ± 3.2	44.8 ± 7.8	92.8 ± 3.0
RVT [Goyal et al., 2023]	62.9 ± 3.7	3.6	52.0 ± 2.5	99.2 ± 1.6	11.2 ± 3.0	88.0 ± 2.5	71.2 ± 6.9	4.0 ± 2.5	91.0 ± 5.2	**100.0** ± 0.0
RVT-2 [Goyal et al., 2024]	81.4 ± 3.1	1.9	**100.0** ± 0.0	99.0 ± 1.7	40.0 ± 0.0	**99.0** ± 1.7	74.0 ± 11.8	38.0 ± 4.5	**95.0** ± 3.3	**100.0** ± 0.0
SAM-E [Zhang et al., 2024a]	70.6 ± 0.7	2.6	82.4 ± 3.6	**100.0** ± 0.0	18.4 ± 4.6	95.2 ± 3.3	**95.2** ± 5.2	0.0 ± 0.0	94.4 ± 4.6	**100.0** ± 0.0
SAM2Act (Ours)	**86.8** ± 0.5	**1.8**	99.0 ± 2.0	99.0 ± 2.0	**84.0** ± 5.7	98.0 ± 2.3	83.0 ± 6.0	**47.0** ± 6.0	93.0 ± 3.8	**100.0** ± 0.0
Method	**Put in Cupboard**	**Put in Drawer**	**Put in Safe**	**Screw Bulb**	**Slide Block**	**Sort Shape**	**Stack Blocks**	**Stack Cups**	**Sweep to Dustpan**	**Turn Tap**
PerAct [Shridhar et al., 2023]	28.0 ± 4.4	51.2 ± 4.7	84.0 ± 3.6	17.6 ± 2.0	74.0 ± 13.0	16.8 ± 4.7	26.4 ± 3.2	2.4 ± 2.0	52.0 ± 0.0	88.0 ± 4.4
RVT [Goyal et al., 2023]	49.6 ± 3.2	88.0 ± 5.7	91.2 ± 3.0	48.0 ± 5.7	81.6 ± 5.4	36.0 ± 2.5	28.8 ± 3.9	26.4 ± 8.2	72.0 ± 0.0	93.6 ± 4.1
RVT-2 [Goyal et al., 2024]	66.0 ± 4.5	96.0 ± 0.0	96.0 ± 2.8	88.0 ± 4.9	92.0 ± 2.8	35.0 ± 7.1	**80.0** ± 2.8	69.0 ± 5.9	**100.0** ± 0.0	99.0 ± 1.7
SAM-E [Zhang et al., 2024a]	64.0 ± 2.8	92.0 ± 5.7	95.2 ± 3.3	78.4 ± 3.6	**95.2** ± 1.8	34.4 ± 6.1	26.4 ± 4.6	0.0 ± 0.0	**100.0** ± 0.0	**100.0** ± 0.0
SAM2Act (Ours)	**75.0** ± 3.8	**99.0** ± 2.0	**98.0** ± 2.3	**89.0** ± 2.0	86.0 ± 4.0	**64.0** ± 4.6	76.0 ± 8.6	**78.0** ± 4.0	99.0 ± 2.0	96.0 ± 5.7

Table 3.2: `The Colosseum` results. Task-average success rate percentage change for SAM2Act and other baselines across 13 perturbation factors from `The Colosseum`, relative to evaluations without perturbations. Results of 3 evaluations are reported for all models. Our approach, SAM2Act, demonstrates the lowest average percentage change across all perturbations, with a minimal drop of $-4.3 \pm 3.6\%$, highlighting its robustness in handling various environmental and object-level perturbations.

Method	Average ↑	MO-Color ↑	RO-Color ↑	MO-Texture ↑	RO-Texture ↑	MO-Size ↑	RO-Size ↑
RVT-2 [Goyal et al., 2024]	−19.5 ± 2.8	−20.7 ± 1.0	−11.8 ± 0.8	−13.3 ± 4.6	−11.4 ± 3.7	**−13.2** ± 3.1	−17.7 ± 0.1
SAM2Act (SAM2 → SAM)	−20.7 ± 1.2	−26.1 ± 0.7	−15.7 ± 2.9	−15.0 ± 3.3	−16.5 ± 6.2	−18.7 ± 1.9	−19.8 ± 1.3
SAM2Act (w/o Multi-res Input)	−19.1 ± 4.5	−15.5 ± 6.4	−13.5 ± 4.6	−20.4 ± 0.5	−16.6 ± 6.1	−21.3 ± 7.5	−12.6 ± 7.5
SAM2Act (Ours)	**−4.3** ± 3.6	**−1.1** ± 2.5	**−0.7** ± 7.2	**−3.3** ± 2.4	**24.7** ± 6.1	−15.9 ± 5.0	**0.9** ± 6.8
Method	**Light Color ↑**	**Table Color ↑**	**Table Texture ↑**	**Distractor ↑**	**Background Texture ↑**	**Camera Pose ↑**	**All Perturbations ↑**
RVT-2 [Goyal et al., 2024]	−15.6 ± 1.3	−26.5 ± 4.4	−14.6 ± 4.4	−4.9 ± 5.3	−4.4 ± 4.0	−19.5 ± 2.8	−77.9 ± 1.7
SAM2Act (SAM2 → SAM)	−16.3 ± 1.2	−23.5 ± 5.3	−12.3 ± 3.1	0.6 ± 2.9	−5.4 ± 3.2	−20.7 ± 1.2	−79.5 ± 2.5
SAM2Act (w/o Multi-res Input)	−7.2 ± 3.6	−18.3 ± 6.1	−17.5 ± 3.3	−4.6 ± 3.5	−5.7 ± 3.5	−19.1 ± 4.5	−73.8 ± 2.2
SAM2Act (Ours)	**4.5** ± 4.4	**1.1** ± 2.5	**−3.7** ± 5.2	**1.7** ± 1.7	**−1.5** ± 2.7	**−4.3** ± 3.6	**−58.3** ± 4.4

3.5.3 Semantic Generalization Across Tasks

The results evaluated in §3.5.2 were obtained by training and testing models within the same environment. However, to truly assess **generalization performance**, policies must remain robust against both environmental and object-level perturbations. We therefore trained SAM2Act and the baseline methods on 20 tasks from `The Colosseum` benchmark and tested them under 13 different perturbation categories over three runs. **SAM2Act exhibits the smallest performance drop compared to the baselines**, with an average decrease of 4.3% (standard deviation of 3.59%). Notably, it proves particularly robust to environmental perturbations—such as changes in lighting, table color/texture, the addition of distractors, and even camera pose—while also maintaining competitive performance under object-level perturbations.

Table 3.3: Performance on `MemoryBench`. We report the success rates for the three spatial memory tasks in `MemoryBench`. Our method, SAM2Act+, significantly outperforms all baseline methods that lack an explicit memory mechanism, achieving an average improvement of 37.6% across all three tasks.

Methods / Tasks	Avg. Success ↑	(a) Reopen Drawer	(b) Put Block Back	(c) Rearrange Block
RVT-2 [Goyal et al., 2024]	54.0 ± 5.3	60.0 ± 0.0	50.0 ± 2.3	52.0 ± 3.3
SAM2Act (Ours)	55.0 ± 24.3	48.0 ± 0.0	35.0 ± 3.8	82.0 ± 2.3
SAM2Act+ (Ours)	**94.3 ± 9.0**	**84.0** ± 0.0	**100.0** ± 0.0	**99.0** ± 2.0

3.5.4 Performance on `MemoryBench`

In Table 3.3, we evaluate SAM2Act+ against the SoTA 3D BC model, RVT-2, on `MemoryBench`, training all models in a single-task setting to isolate memory-related challenges (e.g., opening the wrong drawer rather than unrelated mid-task failures). This setup ensures that performance differences stem from memory capabilities. For a random agent, the expected success rates are determined by the number of possible choices per task: 33% for `reopen_drawer` (three drawers), 25% for `put_block_back` (four patches), and 50% for `rearrange_block` (two blocks). However, variations in task complexity, fixed training data, and imbalanced task distributions lead to slight deviations from these baselines. Our proposed memory-based model, SAM2Act+, demonstrates **a strong understanding of spatial memory**, achieving an average success rate of 94.3% across all tasks. It **outperforms SAM2Act (without memory) by a margin of** 39.3% **on `MemoryBench`**, highlighting the significant impact of explicit memory modeling.

3.5.5 Real-Robot Evaluations

Table 3.4 presents our real-world experiment results, where our method achieves a 75% task success rate, compared to 43% for RVT-2. SAM2Act significantly outperforms the baseline in high-precision tasks (60% vs. 0%). It excels in memory-based tasks, such as `(d) Push the same button`, which requires recalling the button's previous location. Here, SAM2Act+ achieves 70% success, while RVT-2, relying on random guessing, scores 40%. We also test models' generalization against perturbations like lighting changes, distractors, and position variations.

3.6 Conclusion and Limitations

We introduced SAM2Act, a multi-view, language-conditioned behavior cloning policy for 6-DoF 3D ma-

Table 3.4: Real-world results. We compare RVT-2 against SAM2Act for the first three tasks and SAM2Act+ on the last real-world task (indicated with $*$), evaluating performance both in-distribution and out-of-distribution at test time.

	In-Distribution		Out-Distribution	
Task	**RVT-2**	**SAM2Act**	**RVT-2**	**SAM2Act**
(a) Turn on the lamp	0/10	**6/10**	0/10	**6/10**
(b) Push button sequence	4/10	**9/10**	1/10	**9/10**
(c) Stack cubes	8/10	8/10	3/10	3/10
(d) Push the same button*	4/10	**7/10**	2/10	**6/10**

nipulation, enabling high-precision manipulations while generalizing effectively to unseen perturbations. Building on this foundation, we proposed SAM2Act+, a memory-based multi-view, language-conditioned robotic-transformer-based policy that equips the agent with spatial memory awareness, allowing it to solve spatial memory-based tasks. While both SAM2Act and SAM2Act+ achieve state-of-the-art performance across multiple benchmarks, challenges remain in extending them to dexterous continuous control. Additionally, SAM2Act+ relies on a fixed memory window length, which differs from task to task, limiting its adaptability to tasks of varying length. We also examined whether our memory architecture could retain semantic information (e.g., color), but unfortunately, it appears to be limited to storing spatial information. Despite these challenges, we believe SAM2Act+ is an important step towards memory-based generalist manipulation policies.

Chapter 4

Failure Reasoning: Moving from Binary Success to Non-Scalar Feedback

4.1 Introduction

In recent years, foundation models have made remarkable progress across various domains, demonstrating their ability to handle open-world tasks [Driess et al., 2023; Alayrac et al., 2022; Achiam et al., 2023; Zhang et al., 2023a]. These models, including large language models (LLMs) and vision–language models (VLMs), have shown proficiency in interpreting and executing human language instructions [Ouyang et al., 2022], producing accurate predictions and achieving strong task performance. However, despite these advancements, key challenges remain—particularly with hallucinations, where models generate responses that deviate from truth. Unlike humans, who can intuitively detect and adjust for such errors, these models often lack the mechanisms for recognizing their own mistakes [Lin et al., 2021; Chen et al., 2021; Heyman, 2008].

Learning from failure is a fundamental aspect of human intelligence. Whether it is a child learning to skate or perfecting a swing, the ability to reason over failures is essential for improvement [Young, 2009; Gopnik, 2020; Heyman, 2008]. The concept of improvement through failures is widely applied in training foundation models and is exemplified by techniques such as Reinforcement Learning with Human Feedback (RLHF) [Ouyang et al., 2022; Christiano et al., 2017], where human oversight and feedback steer models toward desired outcomes. This feedback loop plays a critical role in aligning generative models with real-

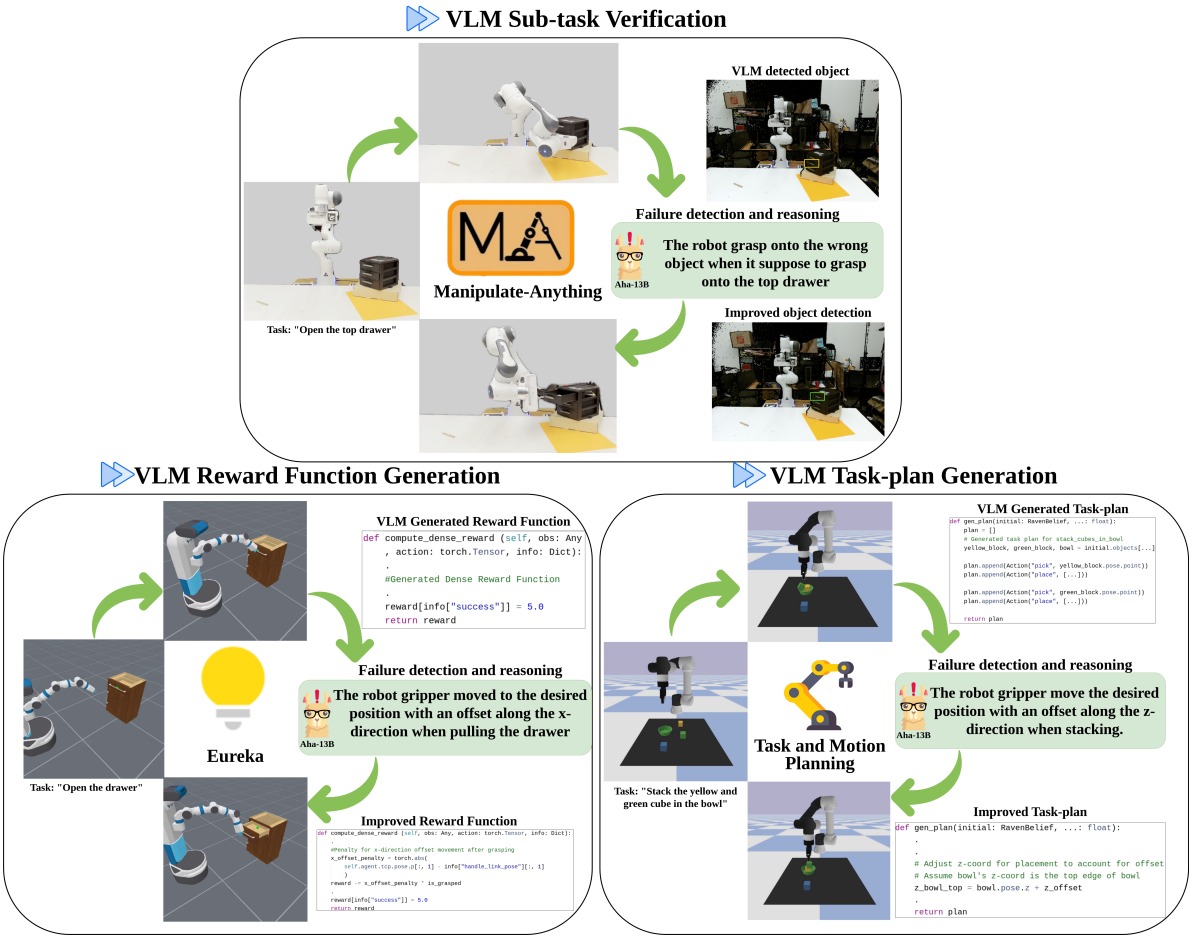


Figure 4.1: AHA is a Vision-Language Model designed to detect and reason about failures in robotic manipulation. As an instruction-tuned VLM, it can enhance task performance in robotic applications that utilize VLMs for reward generation, task planning, or sub-task verification. By incorporating AHA into the reasoning pipeline, these applications can achieve accelerated and improved performance.

world objectives. However, a crucial question persists: how can we enable these models to autonomously detect and reason about their own failures, particularly in robotics, where interactions and environments are stochastic and unpredictable?

This need is particularly pressing in robotics, where foundation models such as VLMs and LLMs are increasingly used to address open-world tasks. Recent advancements have enabled these models to tackle spatial reasoning, object recognition, and multimodal problem-solving—skills vital for robotic manipulation [Reid et al., 2024; OpenAI, 2024; Yuan et al., 2024; Chen et al., 2024; Wang et al., 2023c]. VLMs and LLMs are already being integrated to automate reward generation for reinforcement learning [Ma et al., 2023, 2024], develop task plans for motion planning [Curtis et al., 2024], and even generate zero-shot robot

trajectories [Huang et al., 2023, 2024a; Duan et al., 2024b; Huang et al., 2024b]. While these models excel at task execution, they often face challenges in detecting and reasoning over failures—skills that are crucial for navigating dynamic and complex environments. For example, if a robot drops an object mid-task, a human observer would immediately recognize the error and take corrective action. How can we empower robots with similar capabilities, allowing them not only to perform tasks but also to detect and learn from their mistakes?

To learn from their mistakes, robots must first detect and understand why they failed. In this chapter, we introduce AHA, an open-source VLM that uses natural language to detect and reason about failures in robotic manipulation. Unlike prior work that treats failure reasoning as a binary detection problem, we frame it as a free-form reasoning task, offering deeper insights into failure-mode reasoning. Our model not only identifies failures but also generates detailed explanations. This approach enables AHA to adapt to various robots, camera viewpoints, tasks, and environments in both simulated and real-world scenarios. It can also be integrated into downstream robotic applications leveraging VLMs and LLMs, as shown in Figure 4.1. We make three major contributions.

1. We introduce `FailGen`, a data-generation pipeline for the procedural generation of failure-demonstration data for robotic manipulation tasks across simulators. To instruction-tune AHA, we developed `FailGen`, the first automated data-generation pipeline that procedurally creates the AHA dataset—a large-scale collection of robotic manipulation failures with over 49K image–query pairs across 79 diverse simulated tasks. Despite being fine-tuned only on the AHA dataset, AHA demonstrates strong generalization to real-world failure datasets, different robotic systems, and unseen tasks, as evaluated on three separate datasets not included in the fine-tuning. `FailGen` is a flexible data-generation pipeline that integrates seamlessly with various simulators, enabling scalable procedural generation of failure demonstrations.

2. We demonstrate that AHA excels in failure reasoning, generalizing across different embodiments, unseen environments, and novel tasks, outperforming both open-source and proprietary VLMs. Upon fine-tuning AHA, we benchmark it against six state-of-the-art VLMs, both open-source and proprietary, evaluating performance across four metrics on three diverse evaluation datasets, each featuring different embodiments, tasks, and environments out-of-distribution from the training data. AHA outperforms GPT-4o by more than 20.0% on average across datasets and metrics, and by over 43.0% compared to LLaVA-v1.5-

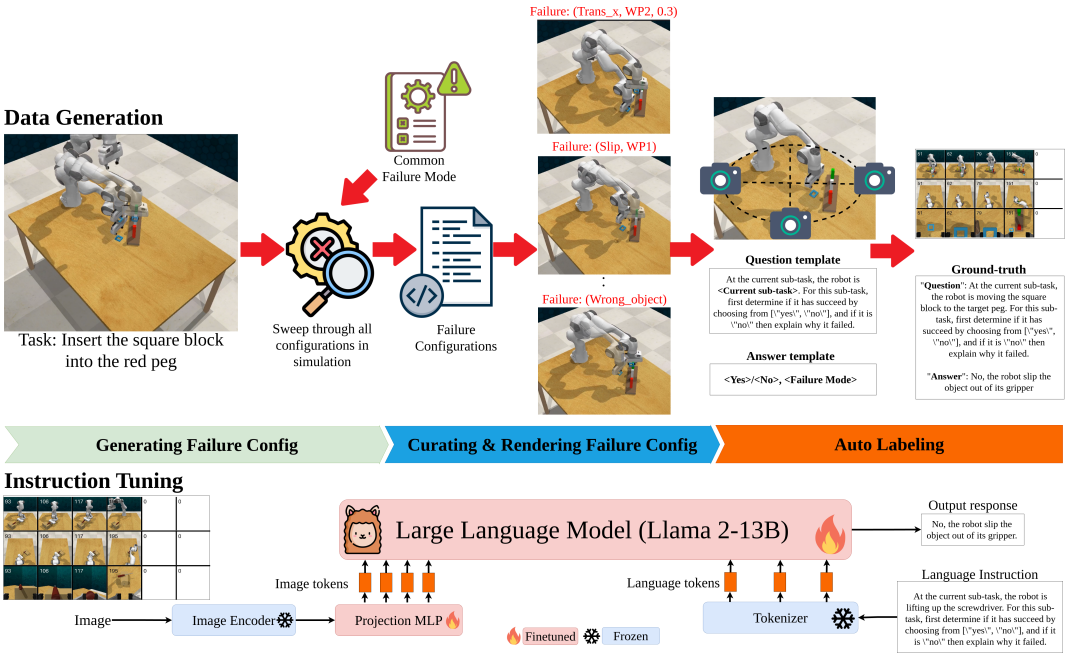


Figure 4.2: Overview of the AHA pipeline. (Top) The data generation for AHA is accomplished by taking a normal task trajectory in simulation and procedurally perturbing all keyframes using our taxonomy of failure modes. Through `FailGen`, we systematically alter keyframes to synthesize failure demonstrations conditioned on the original tasks. Simultaneously, we generate corresponding query and answer prompts for each task and failure mode, which are used for instruction-tuning. (Bottom) The instruction-tuning pipeline follows the same fine-tuning procedure as LLaVA-v1.5 [Liu et al., 2023b], where we fine-tune only the LLM base model—in this case, LLaMA-2-13B—and the projection linear layers, while freezing the image encoder and tokenizer.

13B [Liu et al., 2023b], the base model from which AHA is derived. This demonstrates AHA's exceptional ability to detect and reason about failures in robotic manipulation across embodiment and domains.

3. We show that AHA enhances downstream robotic applications by providing failure-reasoning feedback. We demonstrate that AHA can be seamlessly integrated into robotic applications that utilize VLMs and LLMs. By providing failure feedback, AHA improves reward functions through Eureka reflection, enhances task and motion planning, and verifies sub-task success in zero-shot robotic manipulation. Across three downstream tasks, our approach achieves an average success rate 21.4% higher than GPT-4 models, highlighting AHA's effectiveness in delivering accurate natural-language failure feedback to improve task performance through error correction.

4.2 Related Work

AHA enables language reasoning for failure detection in robotic manipulation, enhancing downstream robotics applications. To provide context, we review progress in (i) failure detection in robotic manipulation, (ii) data generation in robotics, and (iii) foundation models for robotic manipulation.

4.2.1 Failure Detection in Robotic Manipulation

Failure detection and reasoning have long been studied in the human–robot interaction (HRI) community [Ye et al., 2019; Khanna et al., 2023] and in works leveraging task and motion planning (TAMP) [Garrett et al., 2020]. With the recent widespread adoption of LLMs and VLMs in robot manipulation systems—either for generating reward functions or synthesizing robot trajectories [Ma et al., 2023, 2024] in a zero-shot manner—the importance of detecting task failures has regained prominence [Huang et al., 2023; Duan et al., 2024b; Skreta et al., 2024; Ha et al., 2023]. Most modern approaches focus on using off-the-shelf VLMs or LLMs as success detectors [Ma et al., 2022; Ha et al., 2023; Wang et al., 2023a; Duan et al., 2024b; Dai et al., 2024], and some employ instruction-tuning of VLMs to detect failures [Du et al., 2023]. Furthermore, hallucinations often occur in LLMs and VLMs. Methods that leverage these models for failure detection can mitigate this issue by detecting uncertainty in VLMs, as demonstrated in Zheng et al. [2024b]. However, these methods are often limited to binary success detection and do not provide language explanations for why failures occur. Our framework introduces failure reasoning in a new formulation, generating language-based explanations of failures to aid robotic systems that leverage VLMs and LLMs in downstream tasks.

4.2.2 Data Generation in Robotics

There have been many methods in robotic manipulation that automate data generation of task demonstrations at scale [Mandlekar et al., 2023; Hoque et al., 2024], whether for training behavior-cloning policies, instruction-tuning VLMs [Yuan et al., 2024], or curating benchmarks for evaluating robotic policies in simulation [Xie et al., 2024; Pumacay et al., 2024]. A well-known example is MimicGen [Mandlekar et al., 2023], which automates task-demonstration generation via trajectory adaptation by leveraging known object poses. Additionally, works like RoboPoint use simulation to generate general-purpose representations for robotic applications, specifically for fine-tuning VLMs. Similarly, systems like `The Colosseum` [Pumacay

et al., 2024] automate data generation for curating benchmarks in robotic manipulation. Our approach aligns closely with RoboPoint, as we also leverage simulation to generate data for instruction-tuning VLMs. However, unlike RoboPoint, we focus on synthesizing robotic actions in simulation rather than generating representations like bounding boxes or points.

4.2.3 Foundation Models for Robotic Manipulation

In recent years, leveraging foundation models for robotic manipulation has gained significant attention due to the effectiveness of LLMs/VLMs in interpreting open-world semantics and their ability to generalize across tasks [Duan et al., 2022; Hu et al., 2023; Firoozi et al., 2023; Urain et al., 2024]. Two main approaches have emerged: the first uses VLMs and LLMs in a promptable manner, where visual prompts guide low-level action generation based on visual inputs [Liu et al., 2024a; Huang et al., 2024a,b]. The second focuses on instruction-tuning VLMs for domain-specific tasks [Li et al., 2024d]. For example, RoboPoint [Yuan et al., 2024] is tuned for spatial-affordance prediction, and Octopi [Yu et al., 2024] for physical reasoning using tactile images. These models generalize beyond their training data and integrate seamlessly into manipulation pipelines. Our approach follows this second path, developing a scalable method for generating instruction-tuning data in simulation and fine-tuning VLMs specialized in detecting and reasoning about robotic manipulation failures, with applications that extend beyond manipulation tasks to other robotic domains.

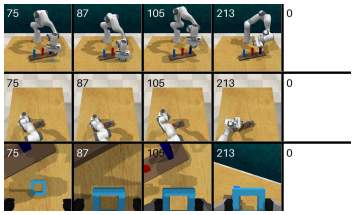
4.3 The AHA Dataset

We leverage `FailGen` to procedurally generate the AHA dataset from RLBench tasks [James et al., 2020] and use it for the instruction-tuning of AHA. In this section, we begin by categorizing common failure modes in robotic manipulation and defining a taxonomy of failures in §4.3.1. Next, we explain how this taxonomy is used with `FailGen` to automate the data generation for the AHA dataset in simulation in §4.3.2.

4.3.1 Failure Modes in Robotic Manipulation

To curate an instruction-tuning dataset of failure trajectories for robotic manipulation tasks, we begin by systematically identifying prevalent failure modes. Our approach involves a review of existing datasets,

Table 4.1: AHA datasets for instruction-tuning. We combine the AHA dataset, our large-scale robotic-manipulation failure dataset, with VQA and object-detection data. By incorporating this diverse data mix into the fine-tuning process, AHA is able to reason about failures in robotic manipulation across different domains, embodiments, and tasks.

Source	The AHA dataset (Train)	VQA [Liu et al., 2023b]	LVIS [Gupta et al., 2019]
			
Quantity	49K	665K	100K
Query	For the given sub-tasks, first determine whether it has succeeded by choosing from ["yes," "no"] and then explain why the current sub-task has failed.	What is the cat doing in the image?	Find all instances of drawer.
Answer	No, the robot gripper rotated with an incorrect roll angle.	The cat is sticking its head into a vase or container, possibly drinking water or investigating the interior of the item.	$[(0.41, (0.42, 0.73, 0.04, 0.08), \ldots]$

including DROID [Khazatsky et al., 2024] and Open-X Embodiment [Padalkar et al., 2023], as well as an analysis of policy rollouts from behavior-cloning models. We examine failures occurring in both teleoperated and autonomous policies. Building upon prior works, such as REFLECT [Liu et al., 2023f], we formalize a taxonomy encompassing seven distinct failure modes commonly observed in robotic manipulation: *incomplete grasp*, *inadequate grip retention*, *misaligned keyframe*, *incorrect rotation*, *missing rotation*, *wrong action sequence*, and *wrong target object*.

Incomplete Grasp (`No_Grasp`) Failure. `No_Grasp` is an object-centric failure that occurs when the gripper reaches the desired grasp pose but fails to close before proceeding to the next keyframe.

Inadequate Grip Retention (`Slip`) Failure. `Slip` is an object-centric failure that happens after the object has been successfully grasped. As the gripper moves the object to the next task-specific keyframe, the grip loosens, causing the object to slip from the gripper.

Misaligned Keyframe (`Translation`) Failure. This action-centric failure occurs when the gripper moves toward a task keyframe, but a translation offset along the X, Y, or Z axis causes the task to fail with respect to a fixed reference coordinate system.

Incorrect Rotation (`Rotation`) Failure. `Rotation` occurs when the gripper reaches the correct position but rotates to an incorrect angle in roll, pitch, or yaw relative to a fixed reference point. Although it attempts the rotation, the misalignment due to rotation results in task failure.

Missing Rotation (`No_Rotation`) Failure. `No_Rotation` occurs when the gripper reaches the correct position but fails to perform the necessary rotation in roll, pitch, or yaw. The absence of any rotation when it is required leads to misalignment and ultimately causes the task to fail.

Wrong Action Sequence (`Wrong_Action`) Failure. `Wrong_Action` is an action-centric failure that occurs when the robot executes actions out of order, performing an action keyframe before the correct one. For example, in the task `put_cube_in_drawer`, the robot moves the cube toward the drawer before opening it, leading to task failure.

Wrong Target Object (`Wrong_Object`) Failure. `Wrong_Object` is an object-centric failure that occurs when the robot acts on the wrong target object, not matching the language instruction. For example, in the task `pick_the_red_cup`, the gripper picks up the green cup, causing failure.

4.3.2 Implementation of the AHA Dataset

To generate the AHA dataset, we systematically sweep through all keyframes in each RLBench task, considering all potential configurations of the seven failure modes that could result in overall task failure. By leveraging the success-condition checker in the simulation, we procedurally generate `YAML`-based configuration files by sweeping through each failure mode across all keyframes. These files provide details on potential failure modes, parameters (such as distance, task sequence, gripper-retention strength, etc.), and corresponding keyframes that `FailGen` should perturb to induce failure. Additionally, we incorporate language templates to describe what the robot is doing between consecutive keyframes. Using these descriptions along with the failure modes, we systematically curate question–answer pairs for each corresponding failure mode.

For specific failure modes, `No_Grasp` is implemented by omitting gripper open/close commands at the relevant keyframes, effectively disabling gripper control. `Slip` introduces a timed release of the gripper shortly after activation. `Translation` and `Rotation` perturb the position and orientation of a keyframe, respectively, while `No_Rotation` constrains the keyframe's rotational axis. `Wrong_Action` reorders keyframe activations to simulate incorrect sequencing, and `Wrong_Object` reassigns the keyframes intended for one object to another, maintaining the relative pose to mimic improper object manipulation. Using this pipeline, we also successfully generate a failure dataset from ManiSkill [Tao et al., 2024] and adapt RoboFail [Liu et al., 2023f] for the evaluation of AHA. This further demonstrates the generalizability and versatility of `FailGen` in generating failure cases across different simulation environments.

4.4 Method

This section outlines the failure-reasoning problem formulation (§4.4.1) used to fine-tune and evaluate AHA. Next, we discuss the curated data mix used for co-finetuning AHA (§4.4.2). Finally, we detail the instruction fine-tuning pipeline and the model-architecture selection for AHA (§4.4.3).

4.4.1 Failure Reasoning Formulation

We extend prior work [Skreta et al., 2024; Duan et al., 2024b] by introducing a two-step framework for robot failure analysis that combines sub-task success detection and failure reasoning. Sub-task success is evaluated as a binary classification problem (*Yes/No*), while failure reasoning is performed using vision–language models (VLMs) to generate natural-language explanations for the causes of failure. This approach allows for both precise failure detection and interpretability in robot manipulation tasks. Manipulation tasks are represented as trajectories consisting of a sequence of sub-tasks $\{S_0, S_1, \ldots, S_T\}$, where each sub-task S_t is defined by two consecutive keyframes (K_t, K_{t+1}). Each sub-task corresponds to an atomic manipulation action, such as "grasping a cube" in a stacking task. For each sub-task, the input to the VLM includes a query prompt and a structured image representation. The query prompt is generated using a template specific to the sub-task and describes the task context and success condition.

The image input is represented as a matrix

$$\mathbf{I} \in \mathbb{R}^{n \times T \times H \times W \times C}, \tag{4.1}$$

where rows correspond to camera viewpoints $\{V_0, V_1, \ldots, V_{n-1}\}$ and columns correspond to temporal keyframes $\{K_0, K_1, \ldots, K_T\}$. To capture the spatiotemporal progression of the task, frames are arranged in temporal order, and missing keyframes are replaced with white patches. We include several camera viewpoints to mitigate occlusions and ensure a comprehensive spatial context. This combined representation enables the VLM to reason over the robot's trajectory and diagnose failure causes effectively, as demonstrated in Table 4.1.

4.4.2 Synthetic Data for Instruction-Tuning

To facilitate the instruction-tuning of AHA, we systematically generate failure-demonstration data. To achieve this, we develop `FailGen`, an environment wrapper that can be easily applied to any robot-manipulation simulator. `FailGen` systematically perturbs successful robot trajectories for manipulation tasks, transforming them into failure trajectories with various modes of failure as depicted in Figure 4.2 (top). Using `FailGen`, we curate the AHA (Train) dataset by alternating across 79 different tasks in the RLBench simulator, resulting in 49K failure image–text pairs. Furthermore, following proper instruction-tuning protocols for VLMs [Liu et al., 2023b] and building on prior work [Brohan et al., 2023; Yuan et al., 2024], co-finetuning is crucial to the success of instruction fine-tuning of VLMs. Therefore, in addition to the AHA dataset, we co-finetune AHA with general visual question-answering (VQA) datasets sourced from internet data, which helps models retain pre-trained knowledge. Specifically, we include the VQA dataset [Liu et al., 2023b], containing 665K conversation pairs, and the LVIS dataset [Gupta et al., 2019], which comprises 100K instances with predicted bounding-box centers and dimensions, as summarized in Table 4.1.

4.4.3 Instruction Fine-Tuning

We follow the instruction-tuning pipeline outlined by Liu et al. [2023d]. As depicted in Figure 4.2, our model architecture includes an image encoder, a linear projector, a language tokenizer, and a transformer-based

language model. The image encoder processes images into tokens, projected by a 2-layer linear projector into the same space as the language tokens. These multimodal tokens are then concatenated and passed through the language transformer. All components are initialized with pre-trained weights. During fine-tuning, only the projector and transformer weights are updated, while the vision encoder and tokenizer remain frozen. The model operates autoregressively, predicting response tokens and a special token marking the boundary between instruction and response.

4.4.4 Impact on Downstream Tasks

AHA integrates failure reasoning to address limitations in downstream robotic methods, improving reward synthesis, decision-making, and feedback efficiency. In reinforcement learning (RL), AHA refines reward synthesis by analyzing rollouts to provide failure explanations, enabling iterative adjustments to dense reward functions and improving sample efficiency, as demonstrated in approaches such as Eureka [Ma et al., 2023]. In task and motion planning (TAMP) systems like PRoC3S [Curtis et al., 2024], AHA enhances feedback loops by interpreting visualizations of failed plans, generating failure explanations, and informing language-model-based plan refinement. This process improves robustness in long-horizon tasks by addressing semantic errors overlooked by finite failure checks. In open-ended frameworks like `Manipulate-Anything` [Duan et al., 2024b], AHA improves subtask verification by analyzing sequential frames for task progression errors, reducing failure propagation in zero-shot data generation. These integrations enable systematic reasoning improvements across RL, TAMP, and data generation, directly enhancing task success and robustness.

4.5 Experimental Results

In this section, we evaluate AHA's detection and reasoning performance against six state-of-the-art VLMs, including both open-source and proprietary models, some utilizing in-context learning. The evaluation spans three diverse datasets, covering out-of-domain tasks, various simulation environments, and cross-embodiment scenarios. We then assess AHA's ability to retain general world knowledge after fine-tuning on domain-specific data. Finally, we explore its potential to improve downstream robotic manipulation applications.

Table 4.2: Quantitative evaluation on failure detection and reasoning. AHA is evaluated and benchmarked against three open and three proprietary VLMs and one visual-prompting baseline across three evaluation datasets. AHA outperforms all other VLMs on every evaluation dataset and nearly every evaluation metric, with the exception of the AHA (Test) dataset, where GPT-4o exceeds it by less than 3%.

	AHA (Test set)				ManiSkill-Fail				REFLECT			
Models	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑
LLaVA-v1.5-13B [Liu et al., 2023b]	0.061	0.208	0.080	0.648	0.000	0.208	0.022	0.270	0.000	0.203	0.000	0.404
LLaVA-NeXT-34B [Liu et al., 2024b]	0.013	0.231	0.017	0.626	0.001	0.195	0.007	0.277	0.018	0.188	0.017	0.351
Qwen-VL [Bai et al., 2023]	0.000	0.161	0.000	0.426	0.037	0.301	0.116	0.034	0.000	0.159	0.000	0.050
Gemini-1.5 Flash [Reid et al., 2024]	0.120	0.231	0.371	0.566	0.003	0.121	0.014	0.032	0.000	0.042	0.000	0.393
GPT-4o	0.251	0.308	0.500	**0.784**	0.142	0.335	0.688	0.453	0.114	0.318	0.554	0.438
GPT-4o-ICL (5-shot)	0.226	0.380	0.611	0.776	0.341	0.429	0.971	0.630	0.236	0.429	0.571	0.418
AHA-7B	0.434	0.574	0.691	0.695	**0.609**	0.680	1.000	0.532	0.204	0.394	0.625	0.439
AHA-13B (Ours)	**0.446**	**0.583**	**0.702**	0.768	0.600	**0.681**	**1.000**	**0.633**	**0.280**	**0.471**	**0.643**	**0.465**

4.5.1 Experimental Setup

To quantitatively evaluate AHA's detection and reasoning capabilities for failures in robotic manipulation, we curate two failure datasets and adapt an existing failure dataset for benchmarking. To ensure a fair comparison of free-form language reasoning, we also employ four different evaluation metrics to measure semantic similarity between sentences.

Benchmarks. We curate three datasets to evaluate AHA's reasoning and failure-detection capabilities, benchmarking against other state-of-the-art VLMs. The first dataset, AHA (Test), includes 11K image–question pairs from 10 RLBench tasks, generated similarly to the fine-tuning data via `FailGen` (§4.3.2) but without overlapping with the fine-tuning dataset. It evaluates AHA's ability to generalize to novel, out-of-domain tasks. The second dataset, ManiSkill-Fail, comprises 130 image–question pairs across four tasks in ManiSkill [Tao et al., 2024], generated using the `FailGen` wrapper on the ManiSkill simulator. This dataset assesses AHA's performance in a different simulator and under changing viewpoints. Lastly, we adapt a failure benchmark from the RoboFail dataset [Liu et al., 2023f], which features real-world robot failures in seven UR5 robot tasks, allowing for evaluation across simulation, real-world trajectories, and different embodiments.

Evaluation Metrics. To fairly evaluate success detection and language reasoning across all datasets and baselines, we employ four metrics. First, the **ROUGE-L score** measures the quality of generated text by focusing on the longest common subsequence between candidate and reference texts. Second, we use

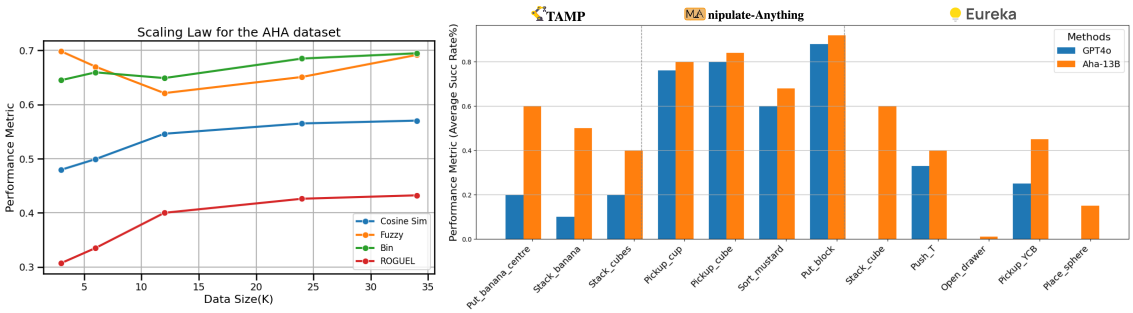


Figure 4.3: (Left) **Scaling law with the AHA dataset.** Effect of model performance with varying domain-specific fine-tuning data. (Right) **Downstream robotic application performance.** AHA outperforms GPT-4o in reasoning about failures within these robotic applications, leading to improved performance of the downstream tasks.

cosine similarity to assess similarity between texts or embeddings, avoiding the "curse of dimensionality." Third, **LLM fuzzy matching** utilizes an external language model—specifically, Anthropic's unseen model, `claude-3-sonnet`—to evaluate semantic similarity in a teacher–student prompting format [Zhou et al., 2023]. Lastly, we calculate a **binary success rate** by comparing the model's predictions directly against the ground truth for success detection.

4.5.2 Quantitative Experimental Results

We contextualize the performance of AHA by conducting a systematic evaluation of failure reasoning and detection across these three datasets, general VQA datasets, and ablation studies.

AHA generalizes across embodiments, unseen environments, and novel tasks. To ensure fairness and eliminate bias in the detection and reasoning capabilities of AHA, we evaluate it on three different datasets that were never seen during fine-tuning, each designed to test a specific form of generalization. First, on the AHA (Test) dataset, AHA demonstrates its ability to **generalize reasoning across tasks and new behaviors within the same domain, outperforming the second-best VLM, GPT-4o ICL,** by an average margin of 12.6% across all evaluation metrics. Second, we assess AHA on a dataset generated by the `FailGen` wrapper in a **different simulation domain,** ManiSkill, showing that our model outperforms GPT-4o-ICL by an average of 13.4% across all metrics, as depicted in Table 4.2. Lastly, to demonstrate **generalization to real-world robots and different embodiments,** we evaluate AHA on RoboFail [Liu et al., 2023f], where it outperforms GPT-4o-ICL by 4.9%.

Table 4.3: Quantitative evaluation on standard VQA benchmarks. AHA performs on par with LLaVA-13B [Liu et al., 2023b], the VLM from which AHA adapts its fine-tuning strategy.

	MMBench [Liu et al., 2023e]	ScienceQA [Lu et al., 2022]	TextVQA [Singh et al., 2019]	POPE [Li et al., 2023]	VizWiz [Gurari et al., 2018]
LLaVA-13B (LLama-2) [Liu et al., 2023b]	**67.70**	**73.21**	**67.40**	**88.00**	53.01
AHA-13B (LLama-2)	65.20	71.94	65.20	85.74	**53.45**

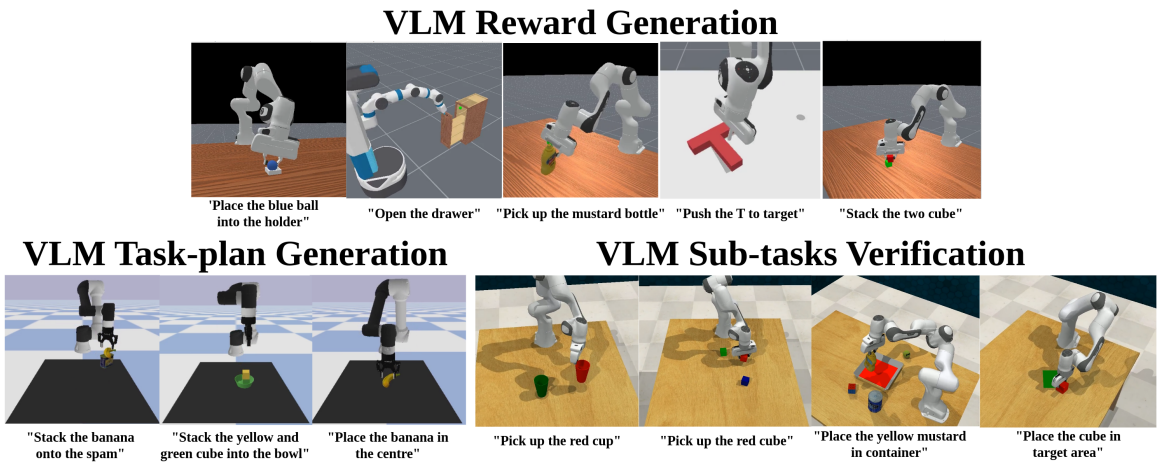


Figure 4.4: Downstream robotic application. We demonstrate that AHA can be integrated into existing LLM/VLM-assisted robotic applications to provide failure reasoning and feedback, helping to accelerate and improve task success rates in these systems.

AHA retains common-sense knowledge. We evaluate AHA's performance on various VQA benchmarks and present the results in Table 4.3. AHA **performs comparably to LLaVA-v1.5-13B (LLama-2)** [Liu et al., 2023b], with only a 1.5% margin difference. Notably, LLaVA-v1.5-13B is a VLM trained on the same pre-trained weights as AHA but fine-tuned on VQA data. This indicates that AHA is capable of functioning as a general-purpose VLM, in addition to excelling at failure reasoning.

AHA's performance scales with data size. We evaluate AHA's performance using a range of AHA data for instruction fine-tuning, spanning $\{3\text{K}, 6\text{K}, 12\text{K}, 34\text{K}, 48\text{K}, 60\text{K}\}$, and co-train individual checkpoints corresponding to these data sizes, as shown in Figure 4.3 (left). The model is then assessed on the ManiSkill-Fail dataset across four evaluation metrics. An average quadratic fit of 0.0022 across all four metrics demonstrates a **scaling effect with fine-tuning on our procedurally generated data pipeline.** This suggests that further scaling can improve model performance.

4.5.3 Downstream Robotic Tasks

We demonstrate that AHA's failure-detection and reasoning capabilities are useful across a wide spectrum of downstream robotic applications. This includes automatic reward generation for reinforcement-learning applications [Ma et al., 2023], automatic task-plan generation for task-and-motion-planning applications [Curtis et al., 2024], and as an improved verification step for automatic data-generation systems [Duan et al., 2024b].

AHA enables efficient reward synthesis for reinforcement learning. To evaluate this downstream task, we adapt Eureka's [Ma et al., 2023] implementation to the ManiSkill simulator, which offers more state-based manipulation tasks. We strictly follow the Eureka reward-function generation and reflection pipeline, modifying it by incorporating perception failure feedback via either AHA or GPT-4o (acting as a baseline) to enhance the original LLM reflection mechanism. Instead of only including a textual summary of reward quality based on policy training statistics for automated reward editing, we further incorporate explanations of policy failures based on evaluation rollouts. We evaluate our approach on five reinforcement-learning tasks from ManiSkill, ranging from tabletop to mobile manipulation. To systematically assess the reasoning capabilities of different VLMs under budget constraints, we sample one reward function initially and allow for iterations over two sessions of GPT API calls. Each policy is trained using PPO over task-specific training steps and evaluated across $1{,}000$ test steps. During policy rollouts, we employ either AHA or GPT-4o for reward reflection to improve the reward function. Comparing the evaluated policy success rates using different failure-feedback VLMs, we observe that AHA provides intuitive, human-level failure reasoning that aids in modifying and improving generated dense reward functions. This results in success across all five tasks within the budget constraints, and our approach **outperforms GPT-4o by a significant margin of** 22.34% **in task success rate** as shown in Figure 4.3 (right).

AHA refines task-plan generation for TAMP. To demonstrate AHA's utility within a planning system, we incorporate our approach into PRoC3S [Curtis et al., 2024]. The PRoC3S system solves tasks specified in natural language by prompting an LLM for a Language-Model Program (LMP) that generates plans, and then testing a large number of these plans within a simulator before executing valid plans on a robot. If no valid plan can be found within a certain number of samples (100 in our experiments), the LLM is re-prompted for a new LMP given failure information provided by the environment. Importantly, as is typical of TAMP

methods, the original approach checks for a finite set of failures (inverse kinematics, collisions, etc.) from the environment, and returns any sampled plan that does not fail in any of these ways. We incorporate a VLM into this pipeline in two ways: (i) we prompt the VLM with visualizations of failed plan executions within the simulator, ask it to return an explanation for the failure, and feed this back to PRoC3S' LLM during the LMP feedback stage; (ii) after PRoC3S returns a valid plan, we provide a visualization of this to the VLM and ask it to return whether this plan truly achieves the natural-language goal, with replanning triggered if not. We compare GPT-4o and AHA as the VLM-based failure-reasoning modules within this implementation of PRoC3S across three tasks (shown in Figure 4.4). Each task is evaluated over 10 trials, with a maximum of 100 sampling steps and three feedback cycles provided by either GPT-4o or AHA. The success rate for each task is recorded. As shown in Figure 4.3 (right), utilizing AHA for **failure reasoning significantly improves the task success rate, outperforming GPT-4o by a substantial margin of** 36.7%**.**

AHA improves task verification for zero-shot robot data generation. To demonstrate AHA's utility in zero-shot robot demonstration generation, we integrate our approach into the `Manipulate-Anything` framework. This open-ended system employs various VLMs to generate diverse robot trajectories and perform a wide range of manipulation tasks without being constrained by predefined actions or scenarios. A critical component of `Manipulate-Anything` is its sub-task verification module, which analyzes past and current frames to decide whether a sub-task has been achieved before proceeding or re-iterating over the previous sub-task. We replace the original VLM (GPT-4V) in the sub-task verification module with AHA and evaluate performance across four RLBench tasks (Figure 4.4), conducting 25 episodes for each task. Our results show that **substituting the sub-task verification module's VLM with AHA improves reasoning accuracy and overall task success by an average of** 5%**.**

4.6 Conclusion

Limitations. AHA currently outputs language reasoning that is closely aligned with the failure scenarios in the fine-tuning data. However, there is an opportunity to output more open-ended failures, to cover those arising from modes outside of the ones included in the failure taxonomy. Additionally, while `FailGen` systematically curates failure data from simulations, distilling large pretrained policies to perform diverse

tasks in simulation and sampling failure modes would allow us to generate more open-ended failure examples, potentially enhancing the instruction-tuned performance of AHA.

Conclusion. We introduced AHA, an open-source vision–language model that significantly enhances robots' ability to detect and reason about manipulation task failures using natural language. By framing failure detection as a free-form reasoning task, AHA not only identifies failures but also provides detailed explanations adaptable to various robots, tasks, and environments. Leveraging `FailGen` and the curated AHA dataset, we trained AHA on a diverse set of robotic failure trajectories. Our evaluations show that AHA outperforms existing models across multiple metrics and datasets. When integrated into manipulation frameworks, its natural-language feedback greatly improves error recovery and policy performance compared to GPT-4 models. These results demonstrate AHA's effectiveness in enhancing task performance through accurate error detection and correction.

Chapter 5

Conclusion

This thesis began with a simple but stubborn question. Modern foundation models in language and vision have advanced largely by scaling data and parameters, and the temptation in robotics has been to follow the same recipe. Yet robot data is not internet data: it is embodied, expensive to collect, and inherently long-tailed in exactly the situations where robustness matters most. A direct transfer of the scaling recipe is therefore not just impractical, it is conceptually mismatched. The question this thesis set out to answer was how robots can learn more from less, generalize beyond the distribution of their training data, and continue to improve through their own experience, when the path of pure scale is closed off. The argument developed across the preceding chapters is that reasoning in the loop offers a principled answer. Rather than treating reasoning as a downstream module bolted onto a perception–action pipeline, this thesis has shown how reasoning can be threaded directly through the way data is represented, how models are trained, and how policies are refined during deployment. Structured reasoning over space, over time, and over failure provides supervision and inductive bias that compensate for the scarcity of embodied data, and that yield robot behaviors which are not only more capable but also more interpretable, more controllable, and more able to recover from their own mistakes.

The grand challenge of building robots that operate reliably in the unstructured environments of everyday human life will not be solved by any single thesis, and certainly not by any single technique. What this thesis has tried to demonstrate is that a particular conceptual move, treating reasoning as something that happens inside the loop of perception, action, memory, and failure rather than something applied from

outside, makes a measurable difference in how much capability and robustness can be extracted from limited embodied data. Spatial reasoning bridges semantics to action; temporal reasoning extends reactive policies into coherent agents; failure reasoning turns mistakes into supervision. Together, these directions reframe robotics foundation models not as scaled-up policies waiting for more data, but as systems that learn through reasoning. The robots that will eventually share our homes, hospitals, and workplaces will need to understand what we ask of them, ground that understanding in the physical world, remember what they have done, recognize when they have erred, and improve themselves through their own experience. The work in this thesis is one step toward such systems. The path ahead is long, but the loop is, at last, beginning to close.

Bibliography

Manoj Acharya, Kushal Kafle, and Christopher Kanan. 2019. TallyQA: Answering complex counting questions. In *AAAI*.

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

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. In *NeurIPS*.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*.

Lars Berscheid, Pascal Meißner, and Torsten Kröger. 2019. 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.

Dimitri P. Bertsekas. 2017. *Dynamic Programming and Optimal Control*, 4th edition. Athena Scientific, Belmont, MA.

Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. 2024. 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.

Mahtab Bigverdi, Zelun Luo, Cheng-Yu Hsieh, Ethan Shen, Dongping Chen, Linda G. Shapiro, and Ranjay Krishna. 2024. Perception tokens enhance visual reasoning in multimodal language models.

Mahtab Bigverdi, Zelun Luo, Cheng-Yu Hsieh, Ethan Shen, Dongping Chen, Linda G Shapiro, and Ranjay Krishna. 2025. Perception tokens enhance visual reasoning in multimodal language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3836–3845.

Ali Furkan Biten, Ruben Tito, Andres Mafla, Lluis Gomez, Marçal Rusinol, Ernest Valveny, CV Jawahar, and Dimosthenis Karatzas. 2019. Scene text visual question answering. In *ICCV*.

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

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π0: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. 2017. Probabilistic data association for semantic slam. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1722–1729. IEEE.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, et al. 2025. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*.

Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. 2020. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33:4247–4258.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Danny Driess, Pete Florence, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. 2024. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. *arXiv preprint arXiv:2401.12168*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Shizhe Chen, Ricardo Garcia, Cordelia Schmid, and Ivan Laptev. 2023. Polarnet: 3d point clouds for language-guided robotic manipulation. *arXiv preprint arXiv:2309.15596*.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668.

Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Jaden Clark, Suvir Mirchandani, Dorsa Sadigh, and Suneel Belkhale. 2025. Action-free reasoning for policy generalization. *arXiv preprint arXiv:2502.03729*.

Aidan Curtis, Nishanth Kumar, Jing Cao, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2024. Trust the proc3s: Solving long-horizon robotics problems with llms and constraint satisfaction.

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. 2024. Racer: Rich language-guided failure recovery policies for imitation learning. *arXiv preprint arXiv:2409.14674*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *ArXiv*, abs/2307.08691.

Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. 2019. Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*.

Sudeep Dasari, Mohan Kumar Srirama, Unnat Jain, and Abhinav Gupta. 2023. An unbiased look at datasets for visuo-motor pre-training. In *Conference on Robot Learning*, pages 1183–1198. PMLR.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv e-prints*, pages arXiv–2409.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. 2023. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. 2024a. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. *arXiv preprint arXiv:2410.00371*.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. 2022. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244.

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. 2024b. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*.

Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. 2021. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*.

Ainaz Eftekhar, Kuo-Hao Zeng, Jiafei Duan, Ali Farhadi, Ani Kembhavi, and Ranjay Krishna. 2023. Selective visual representations improve convergence and generalization for embodied ai. *arXiv preprint arXiv:2311.04193*.

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. 2023. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. 2025. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation.

Richard E. Fikes and Nils J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. 2023. Foundation models in robotics: Applications, challenges, and the future. *arXiv preprint arXiv:2312.07843*.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. 2025. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research*, 44(5):701–739.

Ricardo Garcia, Shizhe Chen, and Cordelia Schmid. 2024. Towards generalizable vision-language robotic manipulation: A benchmark and llm-guided 3d policy. *ArXiv*, abs/2410.01345.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2020. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 440–448.

Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 2023. Act3d: Infinite resolution action detection transformer for robotic manipulation. *arXiv preprint arXiv:2306.17817*.

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Francisco, CA.

Alison Gopnik. 2020. Childhood as a solution to explore–exploit tensions. *Philosophical Transactions of the Royal Society B*, 375(1803):20190502.

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. 2024. Rvt-2: Learning precise manipulation from few demonstrations. *arXiv preprint arXiv:2406.08545*.

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. 2023. Rvt: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning*, pages 694–710. PMLR.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *CVPR*.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. 2022. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. 2024. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*.

Markus Grotz, Mohit Shridhar, Yu-Wei Chao, Tamim Asfour, and Dieter Fox. 2024. Peract2: Benchmarking and learning for robotic bimanual manipulation tasks. In *CoRL 2024 Workshop on Whole-body Control and Bimanual Manipulation: Applications in Humanoids and Beyond*.

Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. 2023. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches.

Michal Guerquin. 2022. Introducing AI2's beaker. *AI2 Blog*. Accessed: 2024-12-31. Original: `https://medium.com/ai2-blog/beaker-ed617d5f4593`.

Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. 2022. Instruction-driven history-aware policies for robotic manipulations. *ArXiv*, abs/2209.04899.

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

Harsh Gupta, Xiaofeng Guo, Huy Ha, Chuer Pan, Muqing Cao, Dongjae Lee, Sebastian Scherer, Shuran Song, and Guanya Shi. 2025. Umi-on-air: Embodiment-aware guidance for embodiment-agnostic visuomotor policies. *arXiv preprint arXiv:2510.02614*.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. 2018. Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3608–3617.

Huy Ha, Pete Florence, and Shuran Song. 2023. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on Robot Learning*, pages 3766–3777. PMLR.

Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. 2012. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The international journal of Robotics Research*, 31(5):647–663.

Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. 2024. Rotary position embedding for vision transformer. In *European Conference on Computer Vision*.

Gail D Heyman. 2008. Children's critical thinking when learning from others. *Current directions in psychological science*, 17(5):344–347.

Ryan Hoque, Ajay Mandlekar, Caelan Garrett, Ken Goldberg, and Dieter Fox. 2024. Intervengen: Interventional data generation for robust and data-efficient robot imitation learning. *arXiv preprint arXiv:2405.01472*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Yafei Hu, Quanting Xie, Vidhi Jain, Jonathan Francis, Jay Patrikar, Nikhil Keetha, Seungchan Kim, Yaqi Xie, Tianyi Zhang, Zhibo Zhao, et al. 2023. Toward general-purpose robots via foundation models: A survey and meta-analysis. *arXiv preprint arXiv:2312.08782*.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. 2025. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *arXiv preprint arXiv:2507.16815*.

Haoxu Huang, Fanqi Lin, Yingdong Hu, Shengjie Wang, and Yang Gao. 2024a. Copa: General robotic manipulation through spatial constraints of parts with foundation models. *arXiv preprint arXiv:2403.08248*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. 2024b. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. 2023. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*.

Yixuan Huang, Jialin Yuan, Chanho Kim, Pupul Pradhan, Bryan Chen, Li Fuxin, and Tucker Hermans. 2024c. Out of sight, still in mind: Reasoning and planning about unobserved objects with video tracking enabled memory models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3108–3115. IEEE.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U Tan, Navonil Majumder, Soujanya Poria, et al. 2025. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. 2025. $\pi_{0.5}$: a vision-language-action model with open-world generalization.

Stephen James and Pieter Abbeel. 2022. Coarse-to-fine q-attention with learned path ranking. *arXiv preprint arXiv:2204.01571*.

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

Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J. Davison. 2021. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13729–13738.

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

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2022b. Bc-z: Zero-shot task generalization with robotic imitation learning. *ArXiv*, abs/2202.02005.

Sascha Jockel, Martin Weser, Daniel Westhoff, and Jianwei Zhang. 2008. Towards an episodic memory for cognitive robots. In *Proc. of 6th Cognitive Robotics workshop at 18th European Conf. on Artificial Intelligence (ECAI)*, pages 68–74. Citeseer.

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. 2018. DVQA: Understanding data visualizations via question answering. In *CVPR*.

Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Ákos Kádár, Adam Trischler, and Yoshua Bengio. 2017. FigureQA: An annotated figure dataset for visual reasoning. *arXiv preprint arXiv:1710.07300*.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. 2023. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 2024. 3d diffuser actor: Policy diffusion with 3d scene representations. *ArXiv*, abs/2402.10885.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. In *ECCV*.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1.

Parag Khanna, Elmira Yadollahi, Mårten Björkman, Iolanda Leite, and Christian Smith. 2023. User study exploring the role of explanation of failures by robots in human robot collaboration tasks. *arXiv preprint arXiv:2303.16010*.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. 2024. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*.

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

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloé Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. 2023. Segment anything. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003.

Huiyuan Lai and Malvina Nissim. 2024. mcot: Multilingual instruction tuning for reasoning consistency in language models. *arXiv preprint arXiv:2406.02301*.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. 2020a. Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning*, pages 5639–5650. PMLR.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. 2020b. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33:19884–19895.

Jean-Claude Latombe. 1991. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA.

Mikko Lauri, David Hsu, and Joni Pajarinen. 2022. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40.

Steven M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press, Cambridge, UK.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. 2025. Molmoact: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. 2024a. LLaVA-OneVision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. 2024b. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024c. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*.

Xiang Li, Cristina Mata, Jongwoo Park, Kumara Kahatapitiya, Yoo Sung Jang, Jinghuan Shang, Kanchana Ranasinghe, Ryan Burgert, Mu Cai, Yong Jae Lee, et al. 2024d. Llara: Supercharging robot learning data for vision-language policy. *arXiv preprint arXiv:2406.20095*.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. 2024e. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Raymond Yu, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, et al. 2025. Hamster: Hierarchical action models for open-world robot manipulation. *arXiv preprint arXiv:2502.05485*.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Evaluating object hallucination in large vision-language models. *arXiv preprint arXiv:2305.10355*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2022. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. 2024. Data scaling laws in imitation learning for robotic manipulation. *arXiv preprint arXiv:2410.18647*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

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

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. 2024a. Moka: Open-vocabulary robotic manipulation through mark-based visual prompting. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023b. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024b. Llava-next: Improved reasoning, ocr, and world knowledge.

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

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

Huaping Liu, Xinghang Li, Peiyan Li, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Tao Kong, and Hanbo Zhang. 2025. Towards generalist robot policies: What matters in building vision-language-action models.

Peiqi Liu, Zhanqiu Guo, Mohit Warke, Soumith Chintala, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. 2024c. Dynamem: Online dynamic spatio-semantic memory for open world mobile manipulation. *arXiv preprint arXiv:2411.04999*.

Peiqi Liu, Yaswanth Orru, Jay Vakil, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. 2024d. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. 2024e. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2023e. Mmbench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281*.

Zeyi Liu, Arpit Bahety, and Shuran Song. 2023f. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *NeurIPS*.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *ICLR*.

Kevin M. Lynch and Frank C. Park. 2017. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, Cambridge, UK.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. 2024. Dreureka: Language model guided sim-to-real transfer. *arXiv preprint arXiv:2406.01967*.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. 2022. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*.

Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Jason Ma, Claire Chen, Sneha Silwal, Aryan Jain, Vincent-Pierre Berges, Tingfan Wu, Jay Vakil, et al. 2023. Where are we in the search for an artificial visual cortex for embodied intelligence? *Advances in Neural Information Processing Systems*, 36:655–677.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. 2023. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*.

Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. 2018. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. 2019. OK-VQA: A visual question answering benchmark requiring external knowledge. In *CVPR*.

Ahmed Masry, Do Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *ACL*.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar. 2022. InfographicVQA. In *WACV*.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. 2021. DocVQA: A dataset for VQA on document images. In *WACV*.

Nitesh Methani, Pritha Ganguly, Mitesh M Khapra, and Pratyush Kumar. 2020. PlotQA: Reasoning over scientific plots. In *WACV*.

Richard M. Murray, Zexiang Li, and S. Shankar Sastry. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. 2022. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*.

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

NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu,

Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. 2025. Gr00t n1: An open foundation model for generalist humanoid robots.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

OpenAI. 2024. Hello gpt-4o.

Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2023. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. 2024. 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.

Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. 2023. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*.

Fabian Peller-Konrad, Rainer Kartmann, Christian RG Dreher, Andre Meixner, Fabian Reister, Markus Grotz, and Tamim Asfour. 2023. A memory system of a robot cognitive architecture and its implementation in armarx. *Robotics and Autonomous Systems*, 164:104415.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. 2025. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*.

Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. 2024. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*.

Jianing Qian, Yunshuang Li, Bernadette Bucher, and Dinesh Jayaraman. 2024. Task-oriented hierarchical object decomposition for visuomotor control. In *Conference on Robot Learning*.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. 2025. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021a. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021b. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. 2024. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Stuart J. Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*, 4th edition. Pearson, Hoboken, NJ.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35:25278–25294.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. 2022. A-OKVQA: A benchmark for visual question answering using world knowledge. In *ECCV*.

Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. 2023. On bringing robots home. *arXiv preprint arXiv:2311.16098*.

Rutav Shah and Vikash Kumar. 2021. Rrl: Resnet as representation for reinforcement learning. *arXiv preprint arXiv:2107.03380*.

Jinghuan Shang, Karl Schmeckpeper, Brandon B May, Maria Vittoria Minniti, Tarik Kelestemur, David Watkins, and Laura Herlant. 2024. Theia: Distilling diverse vision foundation models for robot learning. *arXiv preprint arXiv:2407.20179*.

Yide Shentu, Philipp Wu, Aravind Rajeswaran, and Pieter Abbeel. 2024. 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.

Junyao Shi, Jianing Qian, Yecheng Jason Ma, and Dinesh Jayaraman. 2024a. Composing pre-trained object-centric representations for robotics from "what" and "where" foundation models. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15424–15432.

Lucy Xiaoyang Shi, Zheyuan Hu, Tony Z Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. 2024b. Yell at your robot: Improving on-the-fly from language corrections. *arXiv preprint arXiv:2403.12910*.

Olaolu Shorinwa, Johnathan Tucker, Aliyah Smith, Aiden Swann, Timothy Chen, Roya Firoozi, Monroe David Kennedy, and Mac Schwager. 2024. Splat-mover: Multi-stage, open-vocabulary robotic manipulation via editable gaussian splatting. In *8th Annual Conference on Robot Learning*.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2022. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2023. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR.

Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. 2009. *Robotics: Modelling, Planning and Control*. Springer, London.

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Devi Parikh, and Marcus Rohrbach. 2019. Towards VQA models that can read. In *CVPR*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*.

Marta Skreta, Zihan Zhou, Jia Lin Yuan, Kourosh Darvish, Alán Aspuru-Guzik, and Animesh Garg. 2024. Replan: Robotic replanning with perception and language models. *arXiv preprint arXiv:2401.04157*.

Linda Smith and Michael Gasser. 2005. The development of embodied cognition: Six lessons from babies. *Artificial life*, 11(1-2):13–29.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, et al. 2024. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *ArXiv*, abs/2104.09864.

Qi Sun, Pengfei Hong, Tej Deep Pala, Vernon Toh, U Tan, Deepanway Ghosal, Soujanya Poria, et al. 2024. Emma-x: An embodied multimodal action model with grounded chain of thought and look-ahead spatial reasoning. *arXiv preprint arXiv:2412.11974*.

Yunlong Sun, Lianwu Guan, Zhanyuan Chang, Chuanjiang Li, and Yanbin Gao. 2019. Design of a low-cost indoor navigation system for food delivery robot based on multi-sensor information fusion. *Sensors*, 19(22):4980.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. 2024. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024a. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

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

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. 2024b. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics*. MIT Press, Cambridge, MA.

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

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. 2025. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*.

Julen Urain, Ajay Mandlekar, Yilun Du, Mahi Shafiullah, Danfei Xu, Katerina Fragkiadaki, Georgia Chalvatzaki, and Jan Peters. 2024. Deep generative models in robotics: A survey on learning from multimodal demonstrations. *arXiv preprint arXiv:2408.04380*.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. 2023. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR.

Che Wang, Xufang Luo, Keith Ross, and Dongsheng Li. 2022a. Vrl3: A data-driven framework for visual deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:32974–32988.

Dian Wang, Robin Walters, Xupeng Zhu, and Robert Platt. 2022b. Equivariant q learning in spatial action spaces. In *Conference on Robot Learning*, pages 1713–1723. PMLR.

Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. 2024a. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. *Advances in neural information processing systems*, 37:124420–124450.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. 2023a. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*.

Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2023b. 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.

Yanwei Wang, Lirui Wang, Yilun Du, Balakumar Sundaralingam, Xuning Yang, Yu-Wei Chao, Claudia Perez-D'Arpino, Dieter Fox, and Julie Shah. 2024b. Inference-time policy steering through human interactions. *arXiv preprint arXiv:2411.16627*.

Yi Ru Wang, Jiafei Duan, Dieter Fox, and Siddhartha Srinivasa. 2023c. Newton: Are large language models capable of physical reasoning? *arXiv preprint arXiv:2310.07018*.

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

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. 2025a. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. 2025b. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*.

Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. 2023. Chained-diffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *7th Annual Conference on Robot Learning*.

Annie Xie, Lisa Lee, Ted Xiao, and Chelsea Finn. 2024. 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.

Hu Xu, Saining Xie, Xiaoqing Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. 2024a. Demystifying CLIP data. In *ICLR*.

Zhiyuan Xu, Kun Wu, Junjie Wen, Jinming Li, Ning Liu, Zhengping Che, and Jian Tang. 2024b. A survey on robotics with foundation models: toward embodied ai. *arXiv preprint arXiv:2402.02385*.

Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai,

Seonghyeon Ye, Joel Jang, et al. 2025a. Magma: A foundation model for multimodal ai agents. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 14203–14214.

Jonathan Yang, Chuyuan Kelly Fu, Dhruv Shah, Dorsa Sadigh, Fei Xia, and Tingnan Zhang. 2025b. Bridging perception and action: Spatially-grounded mid-level representations for robot generalization. *arXiv preprint arXiv:2506.06196*.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024. Depth anything v2. *ArXiv*, abs/2406.09414.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. 2021. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International conference on learning representations*.

Sean Ye, Glen Neville, Mariah Schrum, Matthew Gombolay, Sonia Chernova, and Ayanna Howard. 2019. Human trust after robot mistakes: Study of the effects of different forms of robot communication. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–7. IEEE.

H Peyton Young. 2009. Learning by trial and error. *Games and economic behavior*, 65(2):626–643.

Samson Yu, Kelvin Lin, Anxing Xiao, Jiafei Duan, and Harold Soh. 2024. Octopi: Object property reasoning with large tactile-language models. *arXiv preprint arXiv:2405.02794*.

Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. 2024. Robopoint: A vision-language model for spatial affordance prediction for robotics. *arXiv preprint arXiv:2406.10721*.

Wentao Yuan, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. 2023. M2t2: Multi-task masked transformer for object-centric pick and place. *arXiv preprint arXiv:2311.00926*.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. 2024. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. 2021. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR.

Kuo-Hao Zeng, Zichen Zhang, Kiana Ehsani, Rose Hendrix, Jordi Salvador, Alvaro Herrasti, Ross Girshick, Aniruddha Kembhavi, and Luca Weihs. 2024. Poliformer: Scaling on-policy rl with transformers results in masterful navigators. *arXiv preprint arXiv:2406.20083*.

Junjie Zhang, Chenjia Bai, Haoran He, Wenke Xia, Zhigang Wang, Bin Zhao, Xiu Li, and Xuelong Li. 2024a. Sam-e: Leveraging visual foundation model with sequence imitation for embodied manipulation. *arXiv preprint arXiv:2405.19586*.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023a. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847.

Xinyu Zhang, Yuhan Liu, Haonan Chang, Liam Schramm, and Abdeslam Boularias. 2024b. Autoregressive action sequence learning for robotic manipulation. *IEEE Robotics and Automation Letters*, 10:4898–4905.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. 2023b. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. 2025. 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.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*.

Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. 2024a. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*.

Zhi Zheng, Qian Feng, Hang Li, Alois Knoll, and Jianxiang Feng. 2024b. Evaluating uncertainty-based failure detection for closed-loop llm planners.

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

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR.

Chapter A

Appendix: Bridging Spatial Semantics and Action in Robotics Foundation Models

A.1 Appendix

A.1.1 Overview

This appendix expands on the chapter with additional details, experiments, and qualitative results. It covers:

- Model architecture details, including the pre-processor, ViT image encoder, vision–language connector, LLM backbone, and architectural hyperparameters (§A.1.2).
- Training implementation details, GPU cluster description, and full training hyperparameters (§A.1.3).
- Evaluation details, including the protocols for SimplerEnv, LIBERO, real-world post-training, generalization, mid-training ablation, instruction following, and steerability (§A.1.4).
- Data details, including the structure of the MOLMOACT DATASET for both home-environment and tabletop settings (§A.1.5).
- Randomly selected dataset examples covering action-reasoning data, auxiliary visual reasoning trace, depth-perception tokens, trajectory-conditioned action data, multimodal web data, the MOLMOACT DATASET (home and tabletop), and post-training demonstrations (§A.1.6).

- Limitations and potential solutions (§A.1.7).

A.1.2 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 optional image padding, a ViT image encoder, a vision–language connector, and an LLM.

Backbone Overview

MOLMOACT has the following parts: **Pre-processor:** Converts each input image into one low-resolution crop and multiple high-resolution crops.

ViT image encoder: Encodes each crop independently into patch-level visual features.

Vision–language connector: Pools and projects the patch features into the LLM embedding space.

LLM: Autoregressively processes the resulting 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 the fully open OLMo-2-1124-7B and the 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 [Xu et al., 2024a].

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 [Kim et al., 2024] showing that image resolution does not 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 neighbour 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 the OpenAI CLIP vision encoder, we follow the way Molmo [Deitke et al., 2024] resizes and pads 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 the 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.

Vision–Language Connector

After ViT encoding, Molmo aggregates features in two steps: **Layer selection and concatenation:** Features from the third-to-last layer for OpenAI CLIP, or the fourth-to-last layer for SigLIP2, are concatenated with features from the tenth-from-last ViT layer for each patch. This slightly outperforms using features from a single layer, as shown in Molmo [Deitke et al., 2024].

Attention pooling in 2×2 windows: Within each 2×2 patch window, a multi-headed attention layer pools the four patch features into a single vector, using the mean-pooled patch feature as the query. This

reduces sequence length while preserving local spatial structure, and outperforms naive concatenation, as shown in Molmo [Deitke et al., 2024].

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

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.

Multi-Image Inputs

Molmo [Deitke et al., 2024] itself does not provide the capability to take in multi-image inputs. We implement this in a straightforward way: we process all 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 "`Prefix i`", where `i` stands for the i-th image.

Full Hyperparameters

The full hyperparameters of the MOLMOACT architecture are shown in Table A.1. Note that for LoRA implementation, adapters are applied to all linear layers in the model.

A.1.3 Training Details

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 [Bigverdi et al., 2025] 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 fewer 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 §A.1.2. By contrast, the OXE robot data used for pre-training has lower resolution, so no high-resolution 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 A.2. 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.

GPU Cluster

MOLMOACT was trained on Jupiter, an Ai2 GPU cluster in Austin, Texas. MOLMOACT workloads were scheduled using Beaker [Guerquin, 2022], a custom workload-management system. Jupiter comprises 128 GPU nodes and is operated by Cirrascale Cloud Services[1].

Compute. Jupiter provides 1,024 NVIDIA H100 GPUs (80 GB HBM3, 700 W) across 128 servers. Each server has 2× Intel Xeon Platinum 8468 CPUs, 2 TB DDR5 system memory, and 18 TB local NVMe storage.

Storage. The servers are connected over an 800 Gbps local network to a WEKA high-performance storage cluster[2]. The storage system provides 1 PB of NVMe SSD across 11 storage servers and 5 PB of HDD across 12 hosts. Each Jupiter server has two bonded 25 Gbps Mellanox Ethernet NICs (50 Gbps per host). In benchmarks, we achieved 761 Gbps 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 [Wang et al., 2023b]. Each server is equipped with eight 400 Gbps InfiniBand adapters (3.2 Tbps 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.

[1] cirrascale.com
[2] weka.io
[3] cirrascale.com/products-and-services/cabinet-technologies

	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 A.1: MOLMOACT's Architecture Hyperparameters. We specify all hyperparameter information for the different model architectures for MOLMOACT-7B-D and MOLMOACT-7B-O.

	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 A.2: 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.

A.1.4 Evaluation Details

Evaluation on SimplerEnv (Google Robot)

We evaluate on SimplerEnv [Li et al., 2024e] 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 (640×480 px, resized to 320×240 px for our case), task language instructions, and delta end-effector pose actions. SimplerEnv evaluation consists of two components: Visual Matching and Variant Aggregation.

Evaluation on LIBERO

We evaluate on the LIBERO simulation benchmark [Liu et al., 2023a], which consists of a Franka Emika Panda arm in simulation with demonstrations containing front- and wrist-view camera images (256×256 px), task language instructions, and delta end-effector pose actions. We follow prior works [Kim et al., 2024] and evaluate on the four task suites—LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long—each with 500 expert demonstrations across 10 tasks. Following Kim et al. [2024], we trained on a modified dataset which filtered out no-op actions and unsuccessful demonstrations. Moreover, we set the action chunk size to $K = 8$ for evaluation on each task suite and execute full chunks before replanning. We report details of our post-training hyperparameters for LIBERO in Table A.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 Tables A.11–A.16.

	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 A.3: 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.

put_bowl_in_sink. The robot picks up the orange bowl next to the sink and places it fully inside the sink. *Instruction:* Put the bowl into the sink. *Progress labels:* Grasp the bowl (0.25), move it into the sink (0.4), open the gripper (0.7), and release the bowl at the target location (1).

wipe_table. The robot grasps the table cloth and moves it across the surface in one direction. *Instruction:* Wipe the table. *Progress labels:* Grasp the towel (0.25), move it in the desired direction (0.5), and complete the wiping motion (1).

table_bussing. The robot grasps the green tea can and places it into the purple bin. *Instruction:* Clean the trash into the bin. *Progress labels:* Grasp the can (0.25), lift the can (0.5), move it above the bin (0.75), and release the can into the bin (1).

set_table. The right arm grasps the banana and places it onto the plate, while the left arm grasps the teapot and pours. *Instruction:* Set the table. *Progress labels:* Place the banana on the plate (0.25), grasp the teapot (0.75), and pour the tea (1).

lift_tray. The left and right arms approach the tray, grasp it, and lift it together. *Instruction:* Lift up the tray. *Progress labels:* The left arm grasps the tray (0.3), the right arm grasps the tray (0.6), and both arms lift the tray together (1).

fold_towel. The right arm presses down on the center of the towel, while the left arm grasps the towel and folds it. *Instruction:* Fold the towel. *Progress labels:* Grasp the towel (0.25), move it to the correct folding location (0.75), and release the towel so that it is folded (1).

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table A.6 (single-arm) and Table A.7 (bimanual). 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 A.1.

Evaluation on Generalization in Real-World

We collected a multi-task set containing 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*), `put_`

`red_cup_in_blue_plate` (*put the red cup into the blue plate*), `put_banana_in_yellow_plate` (*put the banana into the yellow plate*), and `put_banana_in_blue_plate` (*put the banana into the blue plate*). All models are trained on all tasks in 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 picks up the <object>, then puts it into the yellow/blue plate.

  Language Description: *Put the <object> into the yellow/blue plate.*

  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 A.8. 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 A.17.

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 A.18. **`close_lid`**

The robot goes to the back of the lid, closes its gripper, and pushes the lid to close.

Close the lid.

Move the lid towards the closing direction (0.5), close the lid (1).

`rotate_pot`

The robot goes to a target position near the handle and rotates it by 90 degrees.

Rotate the pot.

Go to the target position of the pot handle (0.3), rotate the pot by 45 degrees (0.6), complete the 90-degree rotation (1).

`pour_tea`

The robot grasps the teapot handle and lifts it above the cup to pour.

Pour tea into the cup.

Grasp the teapot (0.5), move the teapot above the cup (0.8), pour tea into the cup (1).

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table A.9. 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 A.2.

Evaluation of MOLMOACT on Instruction-Following

For the evaluation of language-instruction following, we curated five customized scenes using SimplerEnv [Li et al., 2024e] 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 A.3, and language prompts are shown in Table A.4.

Evaluation of MOLMOACT on Action Steerability

We curated the task `pick_up_bowl`, featuring one dirty and one clean bowl. As shown in Figure 2.8, 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 bowl (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 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 A.5.

We report details of MOLMOACT's post-training hyperparameters for this evaluation in Table A.10. 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 in Table A.19.

A.1.5 Data Details

MOLMOACT DATASET

The 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. We list details of the tasks we collected for the MOLMOACT DATASET in Tables A.20 and A.21.

A.1.6 Data Examples

This section includes **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 traces are annotated with a yellow line.

- **Action Reasoning Data** — Figure A.4
- **Auxiliary Visual Reasoning Trace** — Figure A.5
- **Auxiliary Depth Perception Tokens** — Figure A.6
- **Trajectory-Conditioned Action Data** — Figure A.7
- **Multimodal Web Data** — Figure A.8
- **MOLMOACT DATASET (Home Environment)** — Figure A.9
- **MOLMOACT DATASET (Tabletop)** — Figure A.10
- **Post-Training Single-Arm Franka** — Figure A.11
- **Post-Training Bimanual Franka** — Figure A.12
- **Post-Training Rainbow** — Figure A.13

A.1.7 Limitations and Potential Solutions

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

Camera occlusion of the 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., a 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 centres, 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 [Deitke et al., 2024], RoboPoint [Yuan et al., 2024]) 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 actions 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 tokens. For depth-perception token prediction, we follow Bigverdi et al. [2025] 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.



Figure A.1: Examples of single-arm and bimanual tasks. We list the observation breakdown to show how the robot performs each task.

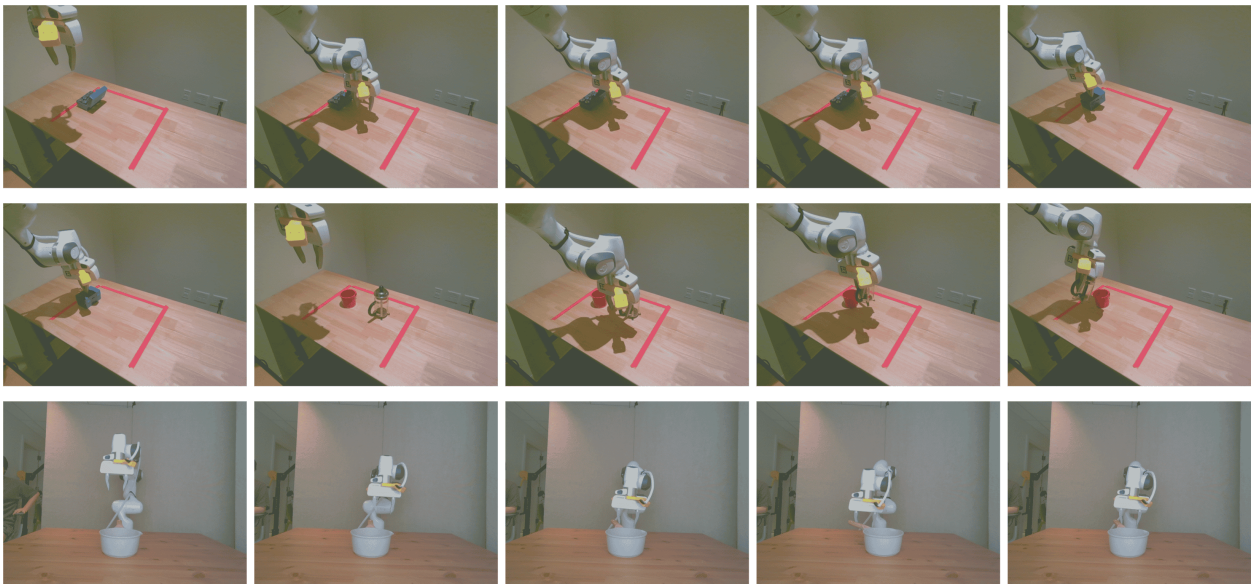


Figure A.2: Examples of MOLMOACT DATASET ablation experiments.



Figure A.3: Language-instruction following. These are the customized scenes curated for open-ended prompting by users.

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

#	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 A.5: 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.>`).

Parameter	Task Name		
	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 A.6: 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 A.7: 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 A.8: 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.

Parameter	Task Name		
	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 A.9: 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 A.10: 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 A.11: 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 A.12: 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 A.13: 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 A.14: 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 A.15: 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 A.16: 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 Variation	put the fruit into the blue plate	0.0	0.0625	0.625
Language Variation	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 A.17: 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 scores used to evaluate each model are detailed at §A.1.4.

Task	Trial	MOLMOACT	MOLMOACT (w/o MOLMOACT DATASET)	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 A.18: 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 A.19: Per-episode evaluation results for bowl-picking tasks with different steering conditions. Scores indicate task progression for each model configuration.

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 A.20: Task details of MOLMOACT DATASET Home Environment, 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 A.21: Task details of MOLMOACT DATASET Tabletop Environment, including scene, task name, language instruction, and all objects used for data collection.

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 [ï³ħ, áļĦ, ïɑĪ, áļĦ, à¬ī, âĶŀ, 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_69><DEPTH_9><DEPTH_101><DEPTH_69><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 [áĪĴ, âķĪ, ðIJɑĶ, ï³ħ, ë±IJ, áį¿IJ, ðŁlij].

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 [ï³ħ, áļĦ, ïɑĪ, áļĦ, à¬ī, âĶŀ, 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 [ðIJŃĵ, ðLįģ, à¬ī, àɑij, ðLĵł, ïɑh, Ÿ¥].

Figure A.4: Randomly selected examples from **action-reasoning data** used in the pre-training stage.

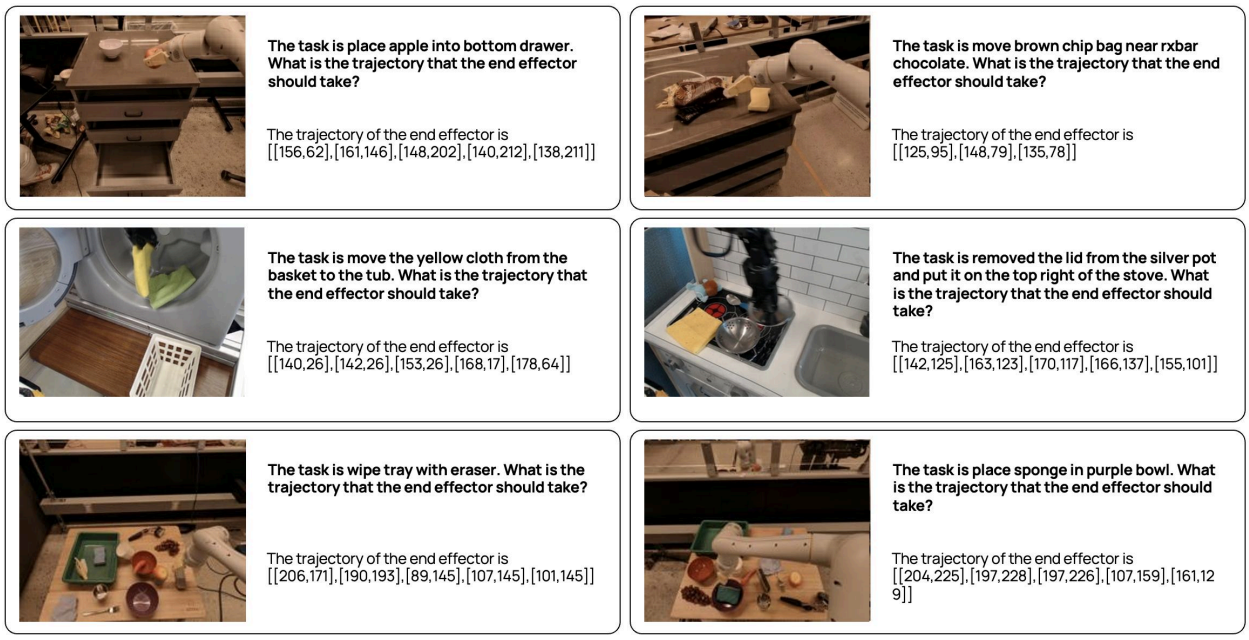
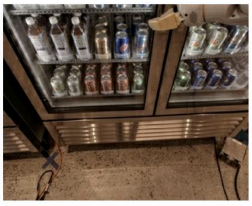


Figure A.5: Randomly selected examples from **auxiliary visual reasoning trace** data used in the pre-training stage.

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_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_32><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_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 A.6: Randomly selected examples from **auxiliary depth-perception tokens** data used in the pre-training stage.

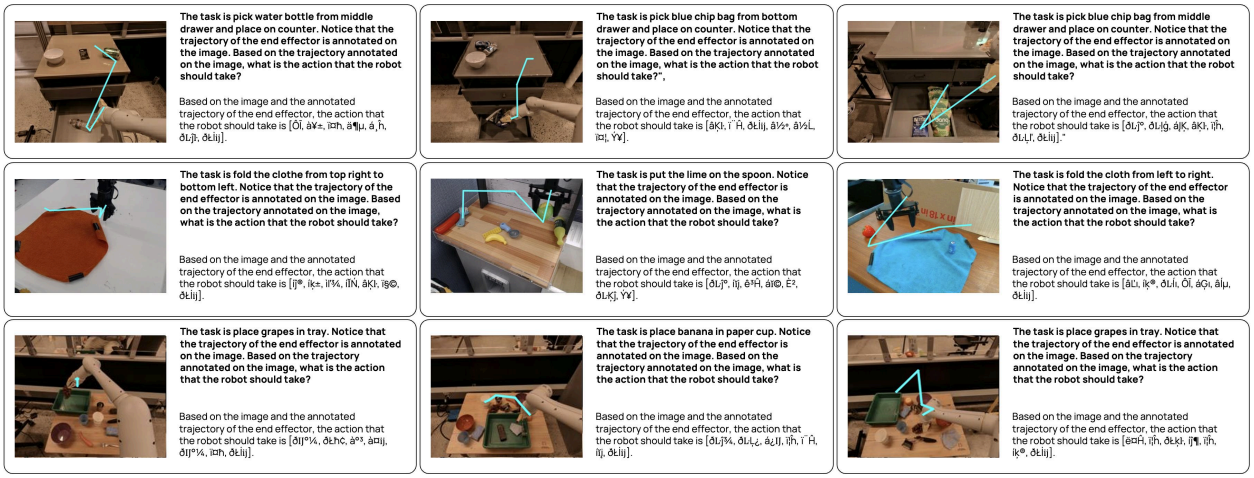
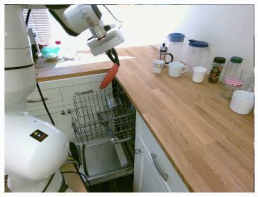


Figure A.7: Randomly selected examples from **trajectory-conditioned action data** used in the pre-training stage.



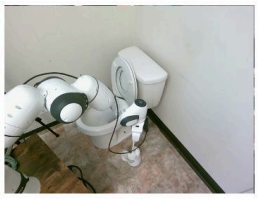
Figure A.8: Randomly selected examples from **multimodal web data** used in the pre-training stage.

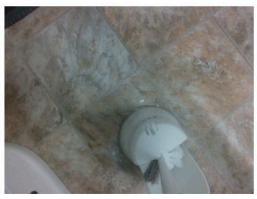




The task is put the plate in the dishwasher. What is the action that the robot should take. To figure out the action that the robot should take to put the plate in the dishwasher, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of 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 [â½Ĺ, âĪĪ, â½Ĺ, ðŁķ±, âĸ¤, ðŁĶĢ, ŸŸ].





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_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, ðLíĮ, ïĩį, íĥĻ, áġ¼, ŸŸ].

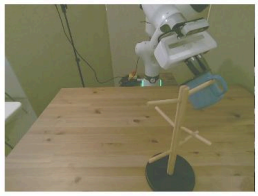


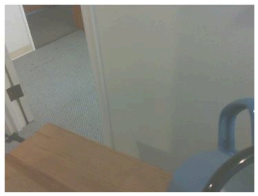


The task is lift the pillow. What is the action that the robot should take. To figure out the action that the robot should take to lift the pillow, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? 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 [Êį, à²Ł, áĶĨ, ðŁıī, ðIJ°¼, áĢı, ðŁIij].

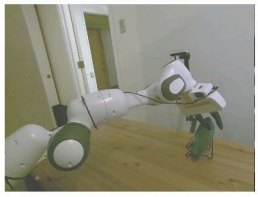
Figure A.9: Randomly selected examples from the **MOLMOACT DATASET (home environment)** used in the mid-training stage.

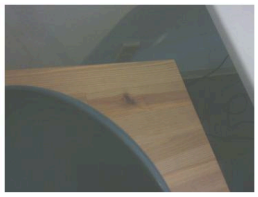




The task is unload the mug. What is the action that the robot should take. To figure out the action that the robot should take to unload the mug, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first image? 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?",

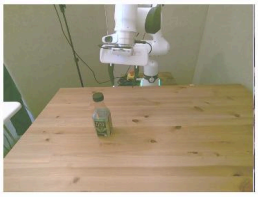
The depth map of the first image is
‹DEPTH_START›‹DEPTH_69›‹DEPTH_9›‹DEPTH_8›‹DEPTH_28›‹DEPTH_8›‹DEPTH_8›‹DEPTH_101›‹DEPTH_32›‹DEPTH_84›‹DEPTH_8›‹DEPTH_69›‹DEPTH_116›‹DEPTH_116›‹DEPTH_25›‹DEPTH_28›‹DEPTH_32›‹DEPTH_101›‹DEPTH_56›‹DEPTH_69›‹DEPTH_32›‹DEPTH_25›‹DEPTH_116›‹DEPTH_116›‹DEPTH_25›‹DEPTH_116›‹DEPTH_30›‹DEPTH_101›‹DEPTH_75›‹DEPTH_103›‹DEPTH_69›‹DEPTH_116›‹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 [áJ̃, Ýj, áJ̃, Õij, ðIJº¼, ðLj̃ì, Ý¥]."

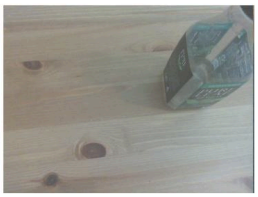




The task is unload the plate. What is the action that the robot should take. To figure out the action that the robot should take to unload the plate, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in the first 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_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 [áJ̃, âĶł, â²¢, ï¥±, ì»ġ, áJĶ, ðŁlij]."

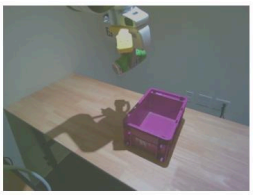




The task is knock down the bottle. What is the action that the robot should take. To figure out the action that the robot should take to knock down the bottle, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end effector in 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_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, áJ̃, áĶĨ, Ýj, ðLĻĽ, àɑij, ðŁlij].

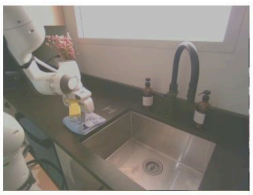
Figure A.10: Randomly selected examples from the **MOLMOACT DATASET (tabletop environment)** used in the mid-training stage.





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?

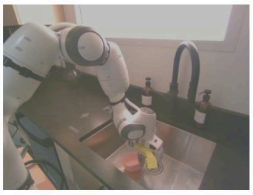
The depth map of the image is
‹DEPTH_START›‹DEPTH_74›‹DEPTH_61›‹DEPTH_104›‹DEPTH_86›‹DEPTH_70›‹DEPTH_41›‹DEPTH_102›‹DEPTH_102›‹DEPTH_90›‹DEPTH_98›‹DEPTH_96›‹DEPTH_32›‹DEPTH_13›‹DEPTH_50›‹DEPTH_9›‹DEPTH_93›‹DEPTH_71›‹DEPTH_91›‹DEPTH_102›‹DEPTH_60›‹DEPTH_8›‹DEPTH_85›‹DEPTH_17›‹DEPTH_13›‹DEPTH_80›‹DEPTH_63›‹DEPTH_70›‹DEPTH_3›‹DEPTH_23›‹DEPTH_34›‹DEPTH_20›‹DEPTH_54›‹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 [[âļĤ, ðĿĮº, ãĪª, Ý¥, Ë², áĮĦ, Ý¥], [âļĺ, ðĿĻĮ', ãĪª, à¥±, ĩĵĸ, áĮĦ, Ý¥], [á¶ı, ðŁĥç, â²ç, ðŁĥç, ðŁĥç, ág¼, Ý¥], [áłĤ, ðĿĮġ, ãĪª, ðĿĵĸ, Ë², Ý¥, ðŁiij], [ájĥ, ðĿĻĮ', âļ£, ðIJ¤Ķ, ðŁĥç, à¥±, ðŁiij], [áÌĪ, ðĿĵº, âļ£, ïʼĥ, íĽ', à°³, ðŁiij], [ðŁķï, ðĿĵł, âĶł, ìĽç, ï¨Ĥ, à°³, ðŁiij], [Ôĥ, ðĿĵł, á¸», ìĽç, ïʼĥ, à²Ł, ðŁiij]].





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 [[à¥±, âĪĪ, áĶĪ, âļ£, ĩ³h, ðĿĵ¹, Ý¥], [à¬ī, âĪ·, áłĦ, âļ£, ĩ³h, ðĿĵ¬, Ý¥], [à¬ī, âĪ·, áłĦ, â²ç, ĩ³h, ðĿiij³, Ý¥], [à¬ī, âĪ·, áłĦ, â²ç, ĩ³h, ðĿĵķ, Ý¥], [à¬ī, âĪ·, áłĦ, âĺµ, ĩ³h, ðκ¥¨, Ý¥], [à¬ī, âĪĪ, áłĦ, â²ç, ĩ³h, ĩĵ», Ý¥], [à¬ī, âĪĪ, áłĦ, ãĪª, ĩ³h, ê³Ĥ, Ý¥], [à¥±, âĪ·, áłĦ, âļ£, íģ», ê³Ĥ, Ý¥]].





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 [[âļĤ, ðĿĮº, ãĪª, Ý¥, Ë², áĮĦ, Ý¥], [âļĺ, ðĿĻĮ', ãĪª, à¥±, ĩĵĸ, áĮĦ, Ý¥], [á¶ı, ðŁĥç, â²ç, ðŁĥç, ðŁĥç, ág¼, Ý¥], [áłĤ, ðĿĮġ, ãĪª, ðĿĵĸ, Ë², Ý¥, ðŁiij], [ájĥ, ðĿĻĮ', âļ£, ðIJ¤Ķ, ðŁĥç, à¥±, ðŁiij], [áÌĪ, ðĿĵº, âļ£, ïʼĥ, íĽ', à°³, ðŁiij], [ðŁķï, ðĿĵł, âĶł, ìĽç, ï¨Ĥ, à°³, ðŁiij], [Ôĥ, ðĿĵł, á¸», ìĽç, ïʼĥ, à²Ł, ðŁiij]].

Figure A.11: Randomly selected examples from **single-arm Franka** demonstrations used in the post-training stage.

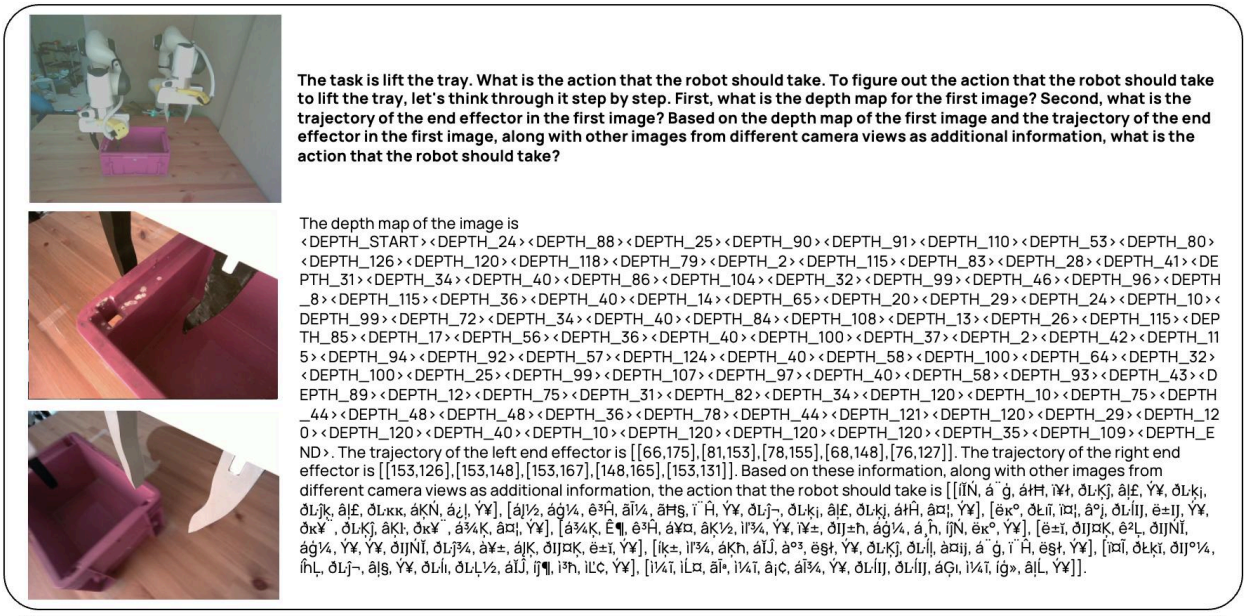
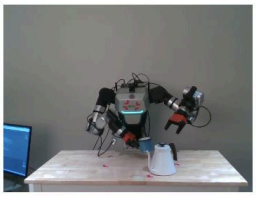
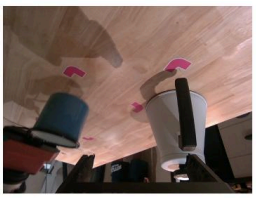
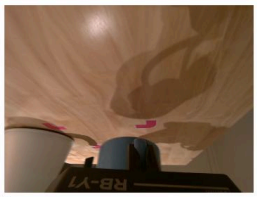


Figure A.12: Randomly selected examples from **bimanual Franka** demonstrations used in the post-training stage.

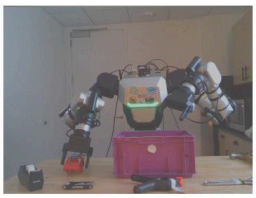


The task is pour the water into the cup. What is the action that the robot should take. To figure out the action that the robot should take to pour the water into the cup, let's think through it step by step. First, what is the depth map for the first image? Second, what is the trajectory of the end 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?



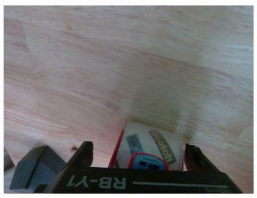


The depth map of the image is
<DEPTH_START><DEPTH_5><DEPTH_53><DEPTH_57><DEPTH_57><DEPTH_57><DEPTH_15><DEPTH_15><DEPTH_41><DEPTH_41><DEPTH_15><DEPTH_14><DEPTH_46><DEPTH_5><DEPTH_46><DEPTH_46><DEPTH_53><DEPTH_53><DEPTH_57><DEPTH_15><DEPTH_57><DEPTH_14><DEPTH_5><DEPTH_19><DEPTH_53><DEPTH_46><DEPTH_53><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¥, áĶh, Ý¥, ðŁhĊ, á¾Ķ, ÕĥÌ, áÎ¨, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, ðLıĺ, âl£, âlş, âl£, âlş, áÎ¨, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, á¸Ī, íĵ®, ðŁhĊ, ê²Ļ, ðŁķĊ, ðŁĨĴ, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, âLı«, íĵ®, ðŁĶĶ, ê²Ļ, ï¨Ĥ, áÎ¨, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, āï«, íĵ®, ðŁĶĶ, ì¼ī, ï¨Ĥ, ìL¿, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, àɬ, ðIJNÎ, ðŁĶĤ, ê²Ļ, ï¨Ĥ, ë»h, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij], [Ý¥, āï«, íĵ®, ðŁĶĶ, íĵ», ï¨Ĥ, ãĽɑ, Ý¥, Ý¥, à²Ł, àɑij, ì³h, ðLĵº, ðLĵ¹, ðLı̣ģ, ðŁiij]]."



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¥, āĩ½, áIJ§, ë±IJ, ÕĪ, íķ®, ÆĪ, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij], [Ý¥, āĩ½, áIJ§, ì»ġ, ÔĪ, Êĵ, âL«, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij], [Ý¥, āĩ½, áIJ§, ì»ġ, ÔĪ, íĽɑ, āï«, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij], [Ý¥, āĩ½, áIJ§, ì»ġ, ÔĪ, íķ®, âL«, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij], [Ý¥, āĩ½, áIJ§, íķ±, ÔĪ, Êĵ, ðLıĽ½, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij], [â½Ł, āĩ½, āĪª, ðLı̣ģ, ëɑĤ, ïş©, ðLĶĵ, Ý¥, Ý¥, á°Ç, ðŁhĊ, ïɑh, ë¡¼, íĨÑ, ðŁh±, ðŁiij]]."

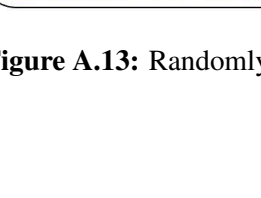


Figure A.13: Randomly selected examples from **Rainbow** demonstrations used in the post-training stage.

Chapter B

Appendix: Storing and Representing Memory for Robotics Foundation Models

B.1 Appendix

B.1.1 Overview

This appendix expands on the chapter with additional details, experiments, and qualitative results. It covers:

- Detailed model-architecture descriptions of the multi-view transformer, memory attention, memory encoder, and memory bank (§B.1.2).
- Training implementation details and hyperparameters for both SAM2Act and SAM2Act+ (§B.1.3).
- Full per-task comparisons on RLBench, including all baselines beyond 3D keyframe-based BC policies (§B.1.4).
- Full per-task results on `The Colosseum` across all 13 perturbation factors (§B.1.5).
- Ablations on the SAM2 image encoder and multi-resolution upsampling (§B.1.6).
- Detailed task summaries for the 18 RLBench tasks (§B.1.7).
- Detailed task descriptions for the three `MemoryBench` tasks and the `reopen_drawer` update (§B.1.8, §B.1.9).

- Real-world experimental setup and per-task descriptions (§B.1.10).

B.1.2 Model Architecture

We explain our model architecture in detail, including the multi-view transformer, memory attention, memory encoder, and memory bank. The multi-resolution upsampling is already explained in §3.4.1.

Multi-View Transformer. The two MVTs used in the coarse and fine branches share the same architecture. Very similar to the MVT proposed by Goyal et al. [2023], the input to the transformer consists of a language description of the task, virtual images of the scene point cloud, and the image embeddings (at the lowest resolution) generated by the SAM2 image encoder. The text is transformed into token embeddings using the pre-trained CLIP [Radford et al., 2021a] model, while the virtual images are converted into token embeddings through patchify and projection operations. Similarly, the image embeddings are converted into token embeddings via a projection layer. For each virtual image, tokens corresponding to the same image are processed through four attention layers. Finally, the processed image tokens, along with the language tokens, are jointly processed using an additional four attention layers. The resulting image tokens are then used to infer the 3D action.

Memory Attention. Akin to the memory attention in SAM2 [Ravi et al., 2024], the purpose of this module is to condition the current observation features on both past observation features and predicted actions, specifically translation. Notably, features from each view are processed independently. We stack four transformer blocks, with the first one taking the image-embedding output of the MVT from the current observation as input. Each block applies self-attention, followed by cross-attention to memories of past observation features and predicted actions stored in a memory bank (described below), and ends with a multi-layer perceptron (MLP). For both self- and cross-attention, we use vanilla attention operations, enabling us to leverage recent advances in efficient attention kernels [Dao, 2023]. In addition to sinusoidal absolute positional embeddings, 2D spatial Rotary Positional Embeddings (RoPE) [Su et al., 2021; Heo et al., 2024] are incorporated in both self-attention and cross-attention layers. We also reduce the dimension size from the original 256 to 128 to align with the image-embedding dimension of the MVT output.

Memory Encoder. The memory encoder constructs memory features by downsampling the output translation heatmap using a convolutional module and summing it element-wise with the unconditioned observation embedding from the multi-view transformer (not shown in Figure 3.4). This is followed by lightweight convolutional layers to integrate the information. Instead of employing an additional image encoder, our memory encoder reuses the image embeddings produced by the MVT (not the SAM2 image encoder) and fuses them with the predicted translation information to generate memory features. This design enables the memory features to leverage rich representations that incorporate language, semantic, and spatial features from multiple views, making them more suitable for encoding action memories. Originally, this module was designed to encode an image embedding with multiple object masks within the same frame. However, we do not utilize this functionality. Instead, we encode one memory per view, where each memory is generated by encoding a single heatmap with a corresponding image embedding from each view.

Memory Bank. The memory bank preserves past translation predictions associated with previous observations in the video by maintaining a FIFO queue of up to N recent memories. Each view has its own independent memory bank, as memories are stored and retrieved separately for different views. These memories are represented as spatial feature maps. Additionally, in our memory bank, the memory features are projected to a dimension of 64.

B.1.3 Training Implementation

All models are trained on 32 NVIDIA H100/A100 GPUs. In some cases, we also train on 16 or 8 NVIDIA H100/A100 GPUs, but we ensure fairness by maintaining the same total batch size across all settings.

SAM2Act

We use the same data and demo augmentation methods and training pipeline as in RVT-2 [Goyal et al., 2024] to train SAM2Act (stage 1). The training hyperparameters are shown in Table B.1. We use this set of hyperparameters to train on RLBench and `The Colosseum`.

Table B.1: Training hyperparameters of SAM2Act on RLBench and `The Colosseum`. The batch size is the total batch size across all GPUs. For the learning rate, we follow the scaling strategy used in RVT-2 [Goyal et al., 2024], where the learning rate is scaled by the batch size as $1.25\text{e}{-5} \times bs$.

Hyperparameter	SAM2Act training
Batch size	256
Learning rate	$3.2\text{e}{-3}$
Optimizer	LAMB
Learning-rate schedule	cosine decay
Weight decay	$1\text{e}{-4}$
Warmup steps	2000
Training steps	56.25K
Training epochs	90
LoRA rank	16

SAM2Act+

We use a different strategy for sampling a batch of data for training. Previous sampling strategies randomly select a batch of independent observations, allowing the model to predict the next action based on each observation independently. However, for SAM2Act+, we aim for the agent to predict the next action based on both the current and past observations. To achieve this, we must sample a batch of data that is spatio-temporally consistent. To implement this, we randomly sample n consecutive observations from a random episode. The forward pass is then performed sequentially from the first to the last observation. The details of the forward pass are provided in Algorithm 1.

When adopting this new sampling method during training, one immediate effect is a significant reduction in data diversity per batch. This can be detrimental, especially when dealing with tasks with numerous variations. We attempted to train the standard SAM2Act model on RLBench tasks using this new sampling method, but the convergence time was excessively long. To address this, we propose a new training pipeline: first, we pre-train the model using the previous sampling method, then fine-tune it with the new sampling approach. This strategy effectively mitigates the issue of slow convergence, significantly reducing training time.

As mentioned in §3.5.4, we train all methods on `MemoryBench` in a single-task setting. However, finding a training configuration that optimizes all tasks is challenging. To address this, we use a universal set of hyperparameters for training but evaluate models across all epochs and select the best-performing one for evaluation. We follow the same approach to determine the optimal pre-trained weights for SAM2Act

Table B.2: Training hyperparameters of SAM2Act and SAM2Act+ on `MemoryBench`. The batch size is the total batch size across all GPUs. For SAM2Act+ training on the `reopen_drawer` task, we use a maximum window size of 8, resulting in a per-GPU batch size of 8 and a total batch size of 256. Similarly, for the other two tasks, where the maximum window size is 10, the total batch size is $10 \times 32 = 320$. The learning rate follows the same scaling rule as in Table B.1.

Hyperparameter	SAM2Act training	SAM2Act+ training
Batch size	256	256 (`reopen_drawer`), 320 (other two)
Learning rate	3.2e−3	3.2e−3 (`reopen_drawer`), 4e−3 (other two)
Optimizer	LAMB	LAMB
Learning-rate schedule	cosine decay	cosine decay
Weight decay	1e−4	1e−4
Warmup steps	2000	2000
Training steps	6.25K	12.5K
Training epochs	10	20
LoRA rank	16	16

before fine-tuning on SAM2Act+. In addition, the window size of the memory mechanism is decided to be different for each task in `MemoryBench`. We keep the batch size the same as the window size during training, and thus the learning rate will be a bit different as it is related to the batch size. The detailed training hyperparameters are listed in Table B.2.

B.1.4 Full Comparisons for RLBench 18 Tasks

The full comparisons of SAM2Act with existing approaches on RLBench 18 tasks are shown in Table B.3.

B.1.5 Full Results for `The Colosseum`

The full results of SAM2Act on `The Colosseum` are shown in Table B.4.

B.1.6 Ablation on SAM2Act

RLBench

We conduct ablation experiments on the proposed SAM2Act, focusing on two key aspects: the SAM2 image encoder and multi-resolution upsampling. We evaluate the model under three different configurations:

Table B.3: Full comparisons of multi-task performance on RLBench. We evaluate 18 RLBench tasks [James et al., 2020], reporting success rates across all tasks among all existing approaches, not limited to 3D keyframe-based behavior-cloning (BC) policies. We report stats over 4 evaluations for SAM2Act. Our method, SAM2Act, outperforms all baselines, achieving a performance margin of 1.9% over ARP$^+$ [Zhang et al., 2024b], the prior state-of-the-art approach.

Method	Avg. Success ↑	Avg. Rank ↓	Close Jar	Drag Stick	Insert Peg	Meat off Grill	Open Drawer	Place Cups	Place Wine	Push Buttons
Image-BC (CNN) [Jang et al., 2022b]	1.3	12.4	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0
Image-BC (ViT) [Jang et al., 2022b]	1.3	12.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C2F-ARM-BC [James et al., 2021]	20.1	11.5	24.0	24.0	4.0	20.0	20.0	0.0	8.0	72.0
HiveFormer [Guhur et al., 2022]	45.3	9.6	52.0	76.0	0.0	**100.0**	52.0	0.0	80.0	84.0
PolarNet [Chen et al., 2023]	46.4	9.1	36.0	92.0	4.0	**100.0**	84.0	0.0	40.0	96.0
PerAct [Shridhar et al., 2023]	49.4 ± 4.3	8.9	55.2 ± 4.7	89.6 ± 4.1	5.6 ± 4.1	70.4 ± 2.0	88.0 ± 5.7	2.4 ± 3.2	44.8 ± 7.8	92.8 ± 3.0
Act3D [Gervet et al., 2023]	65.0	6.8	92.0	92.0	27.0	94.0	93.0	3.0	80.0	99.0
RVT [Goyal et al., 2023]	62.9 ± 3.7	6.9	52.0 ± 2.5	99.2 ± 1.6	11.2 ± 3.0	88.0 ± 2.5	71.2 ± 6.9	4.0 ± 2.5	91.0 ± 5.2	**100.0** ± 0.0
RVT-2 [Goyal et al., 2024]	81.4 ± 3.1	3.7	**100.0** ± 0.0	99.0 ± 1.7	40.0 ± 0.0	99.0 ± 1.7	74.0 ± 11.8	38.0 ± 4.5	95.0 ± 3.3	**100.0** ± 0.0
3D Diffuser Actor [Ke et al., 2024]	81.3	3.9	96.0 ± 2.5	**100.0** ± 0.0	65.6 ± 4.1	96.8 ± 1.6	89.6 ± 4.1	24.0 ± 7.6	93.6 ± 4.8	98.4 ± 2.0
3D-LOTUS [Garcia et al., 2024]	83.1	3.7	96.0 ± 0.0	**100.0** ± 0.0	69.6 ± 3.6	98.4 ± 2.2	85.6 ± 7.3	40.8 ± 12.1	91.2 ± 6.6	**100.0** ± 0.0
ARP$^+$ [Zhang et al., 2024b]	84.9	3.2	95.2	99.2	78.4	97.6	92.8	**48.8**	**96.0**	**100.0**
SAM-E [Zhang et al., 2024a]	70.6 ± 0.7	4.8	82.4 ± 3.6	**100.0** ± 0.0	18.4 ± 4.6	95.2 ± 3.3	**95.2** ± 5.2	0.0 ± 0.0	94.4 ± 4.6	**100.0** ± 0.0
SAM2Act (Ours)	**86.8** ± 0.5	**3.1**	99.0 ± 2.0	99.0 ± 2.0	**84.0** ± 5.7	98.0 ± 2.3	83.0 ± 6.0	47.0 ± 6.0	93.0 ± 3.8	**100.0** ± 0.0

Method	Put in Cupboard	Put in Drawer	Put in Safe	Screw Bulb	Slide Block	Sort Shape	Stack Blocks	Stack Cups	Sweep to Dustpan	Turn Tap
Image-BC (CNN) [Jang et al., 2022b]	0.0	8.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0
Image-BC (ViT) [Jang et al., 2022b]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.0
C2F-ARM-BC [James et al., 2021]	0.0	4.0	12.0	8.0	16.0	8.0	0.0	0.0	0.0	68.0
HiveFormer [Guhur et al., 2022]	32.0	68.0	76.0	8.0	64.0	8.0	8.0	0.0	28.0	80.0
PolarNet [Chen et al., 2023]	12.0	32.0	84.0	44.0	56.0	12.0	4.0	8.0	52.0	80.0
PerAct [Shridhar et al., 2023]	28.0 ± 4.4	51.2 ± 4.7	84.0 ± 3.6	17.6 ± 2.0	74.0 ± 13.0	16.8 ± 4.7	26.4 ± 3.2	2.4 ± 2.0	52.0 ± 0.0	88.0 ± 4.4
Act3D [Gervet et al., 2023]	51.0	90.0	95.0	47.0	93.0	8.0	12.0	9.0	92.0	94.0
RVT [Goyal et al., 2023]	49.6 ± 3.2	88.0 ± 5.7	91.2 ± 3.0	48.0 ± 5.7	81.6 ± 5.4	36.0 ± 2.5	28.8 ± 3.9	26.4 ± 8.2	72.0 ± 0.0	93.6 ± 4.1
RVT-2 [Goyal et al., 2024]	66.0 ± 4.5	96.0 ± 0.0	96.0 ± 2.8	88.0 ± 4.9	92.0 ± 2.8	35.0 ± 7.1	**80.0** ± 2.8	69.0 ± 5.9	**100.0** ± 0.0	99.0 ± 1.7
3D Diffuser Actor [Ke et al., 2024]	**85.6** ± 4.1	96.0 ± 3.6	97.6 ± 2.0	82.4 ± 2.0	97.6 ± 3.2	44.0 ± 4.4	68.3 ± 3.3	47.2 ± 8.5	84.0 ± 4.4	99.2 ± 1.6
3D-LOTUS [Garcia et al., 2024]	78.4 ± 4.6	97.6 ± 3.6	95.2 ± 3.4	88.8 ± 3.4	**99.2** ± 1.8	34.4 ± 4.6	58.4 ± 8.3	75.2 ± 7.7	96.0 ± 2.8	90.4 ± 4.6
ARP$^+$ [Zhang et al., 2024b]	69.6	98.4	86.4	**89.6**	92.8	46.4	63.2	**80.0**	97.6	96.0
SAM-E [Zhang et al., 2024a]	64.0 ± 2.8	92.0 ± 5.7	95.2 ± 3.3	78.4 ± 3.6	95.2 ± 1.8	34.4 ± 6.1	26.4 ± 4.6	0.0 ± 0.0	**100.0** ± 0.0	**100.0** ± 0.0
SAM2Act (Ours)	75.0 ± 3.8	**99.0** ± 2.0	**98.0** ± 2.3	89.0 ± 2.0	86.0 ± 4.0	**64.0** ± 4.6	76.0 ± 8.6	78.0 ± 4.0	99.0 ± 2.0	96.0 ± 5.7

- Replacing the SAM2 image encoder with the SAM image encoder, which requires removing the multi-resolution upsampling since the SAM image encoder does not produce multi-resolution outputs.
- Replacing the multi-resolution upsampling with the original convex upsampling from RVT-2 [Goyal et al., 2024].
- Replacing the multi-resolution image-embedding inputs from SAM2 with single-resolution inputs, while retaining the multi-resolution upsampling module.

Note that SAM-E [Zhang et al., 2024a] proposed a 3D behavior-cloning policy that integrates RVT and the SAM image encoder, along with an action-sequence policy head. We attempted to extend this method to the more powerful RVT-2 backbone for comparison. However, its action-sequence policy proved incompatible with the coarse-to-fine pipeline, resulting in very slow convergence under SAM-E's training setup. To ensure a fair comparison, we also extended SAM-E while keeping its original hyperparameters (notably, a LoRA rank of 4, whereas ours is 16). We trained both versions and found that SAM-E's configuration performed better. Therefore, we adopted their configuration and reported the results accordingly, which also applies to

Table B.4: Full results of SAM2Act for various perturbations on `The Colosseum`. Mean and standard deviation of 3 evaluations are reported. "–" indicates that the perturbation is not applicable to the given task.

Task name	No Var.	All Var.	MO Color	RO Color	MO Texture	RO Texture	MO Size	RO Size	Light Color	Table Color	Table Texture	Distractor	Background Texture	Camera Pose
basketball_in_hoop	100.0 ± 0.0	30.7 ± 2.3	97.3 ± 2.3	100.0 ± 0.0	97.3 ± 2.3	–	100.0 ± 0.0	86.7 ± 2.3	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	94.7 ± 4.6	100.0 ± 0.0	100.0 ± 0.0
close_box	89.3 ± 6.1	61.3 ± 6.1	85.3 ± 6.1	–	–	–	90.7 ± 6.1	–	90.7 ± 2.3	85.3 ± 2.3	81.3 ± 2.3	93.3 ± 4.6	97.3 ± 4.6	92.0 ± 6.9
close_laptop_lid	96.0 ± 0.0	60.0 ± 0.0	100.0 ± 0.0	–	–	–	93.3 ± 11.5	–	94.7 ± 4.6	96.0 ± 0.0	84.0 ± 0.0	93.3 ± 2.3	96.0 ± 0.0	96.0 ± 0.0
empty_dishwasher	0.0 ± 0.0	1.3 ± 2.3	0.0 ± 0.0	0.0 ± 0.0	–	0.0 ± 0.0	1.3 ± 2.3	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
get_ice_from_fridge	93.3 ± 4.6	41.3 ± 2.3	92.0 ± 0.0	93.3 ± 2.3	89.3 ± 2.3	–	84.0 ± 6.9	81.3 ± 2.3	85.3 ± 9.2	98.7 ± 2.3	94.7 ± 2.3	93.3 ± 2.3	89.3 ± 2.3	100.0 ± 0.0
hockey	16.0 ± 4.0	0.0 ± 0.0	30.7 ± 2.3	14.7 ± 2.3	–	9.3 ± 4.6	18.7 ± 4.6	21.3 ± 2.3	29.3 ± 2.3	52.0 ± 6.9	26.7 ± 4.6	6.7 ± 4.6	21.3 ± 2.3	40.0 ± 6.9
meat_on_grill	98.7 ± 2.3	34.7 ± 2.3	100.0 ± 0.0	100.0 ± 0.0	–	–	98.7 ± 2.3	–	62.7 ± 28.9	69.3 ± 9.2	76.0 ± 4.0	100.0 ± 0.0	98.7 ± 2.3	98.7 ± 2.3
move_hanger	1.3 ± 2.3	12.0 ± 0.0	32.0 ± 0.0	0.0 ± 0.0	–	–	–	–	49.3 ± 2.3	64.0 ± 0.0	44.0 ± 6.9	36.0 ± 6.9	0.0 ± 0.0	37.3 ± 18.5
wipe_desk	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	–	0.0 ± 0.0	–	0.0 ± 0.0	–	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 2.3	0.0 ± 0.0
open_drawer	94.7 ± 2.3	70.7 ± 6.1	96.0 ± 0.0	–	–	–	92.0 ± 0.0	–	88.0 ± 0.0	88.0 ± 0.0	100.0 ± 0.0	85.3 ± 2.3	98.7 ± 2.3	90.7 ± 4.6
slide_block_to_target	12.0 ± 0.0	29.3 ± 11.5	42.7 ± 4.6	–	25.3 ± 4.6	–	–	–	25.3 ± 4.6	40.0 ± 0.0	90.7 ± 2.3	49.3 ± 9.2	18.7 ± 2.3	24.0 ± 0.0
reach_and_drag	65.3 ± 14.0	1.3 ± 2.3	54.7 ± 4.6	80.0 ± 10.6	51.7 ± 4.0	69.3 ± 2.3	52.0 ± 27.7	37.3 ± 2.3	76.0 ± 6.9	81.3 ± 12.2	70.7 ± 26.6	65.3 ± 8.3	70.7 ± 16.2	58.7 ± 16.2
put_money_in_safe	74.7 ± 2.3	20.0 ± 4.0	54.7 ± 8.3	52.0 ± 10.6	37.3 ± 2.3	66.7 ± 14.0	69.3 ± 2.3	–	73.3 ± 11.5	69.3 ± 2.3	76.0 ± 20.8	77.3 ± 14.0	50.7 ± 23.4	45.3 ± 18.5
place_wine_at_rack_location	98.7 ± 2.3	38.7 ± 4.6	81.3 ± 2.3	85.3 ± 2.3	–	96.0 ± 6.9	90.7 ± 4.6	97.3 ± 4.6	86.7 ± 4.6	88.0 ± 0.0	97.3 ± 4.6	86.7 ± 4.6	92.0 ± 6.9	69.3 ± 39.3
insert_onto_square_peg	88.0 ± 6.9	46.7 ± 39.3	60.0 ± 4.0	98.7 ± 2.3	–	69.3 ± 4.6	58.7 ± 2.3	61.3 ± 6.1	80.0 ± 0.0	82.7 ± 4.6	64.0 ± 4.0	58.7 ± 2.3	90.7 ± 2.3	82.7 ± 2.3
stack_cups	89.3 ± 4.6	1.3 ± 2.3	88.0 ± 0.0	–	78.7 ± 2.3	–	53.3 ± 11.5	–	88.0 ± 0.0	61.3 ± 2.3	46.7 ± 2.3	73.3 ± 25.4	81.3 ± 2.3	84.0 ± 0.0
turn_oven_on	96.0 ± 0.0	72.0 ± 0.0	92.0 ± 6.9	–	–	–	88.0 ± 0.0	–	89.3 ± 4.6	96.0 ± 0.0	98.7 ± 2.3	96.0 ± 0.0	96.0 ± 0.0	98.7 ± 2.3
straighten_rope	78.7 ± 9.2	6.7 ± 2.3	65.3 ± 4.6	–	70.7 ± 4.6	–	–	–	84.0 ± 0.0	64.0 ± 0.0	61.3 ± 2.3	49.3 ± 2.3	90.7 ± 9.2	76.0 ± 0.0
setup_chess	10.7 ± 2.3	0.0 ± 0.0	12.0 ± 0.0	18.7 ± 2.3	16.0 ± 0.0	–	26.7 ± 2.3	–	22.7 ± 2.3	34.7 ± 11.5	20.0 ± 6.9	22.7 ± 4.6	28.0 ± 6.9	26.7 ± 2.3
scoop_with_spatula	92.0 ± 6.9	10.7 ± 2.3	96.0 ± 6.9	89.3 ± 2.3	88.0 ± 6.9	92.0 ± 6.9	94.7 ± 9.2	78.7 ± 2.3	78.7 ± 4.6	81.3 ± 4.6	76.0 ± 6.9	64.0 ± 6.9	96.0 ± 0.0	94.7 ± 4.6

Table B.5: SAM2Act ablation performance on RLBench. We report the success rates for 18 RLBench tasks [James et al., 2020], along with the average success rate and ranking across all tasks. The table shows that SAM2Act outperforms all of its variations.

Method	Avg. Success ↑	Avg. Rank ↓	Close Jar	Drag Stick	Insert Peg	Meat off Grill	Open Drawer	Place Cups	Place Wine	Push Buttons
SAM2Act (SAM2 → SAM)	80.8 ± 1.4	4.1	96.0 ± 0.0	98.0 ± 2.3	35.0 ± 5.0	**99.0** ± 2.0	80.0 ± 3.3	34.0 ± 3.3	92.0 ± 4.6	**100.0** ± 0.0
SAM2Act (SAM2 → Depth Anything V2)	79.2 ± 1.5	4.4	95.0 ± 2.0	96.0 ± 3.3	65.0 ± 3.8	96.0 ± 3.3	80.0 ± 3.3	28.0 ± 4.6	90.0 ± 3.8	**100.0** ± 0.0
SAM2Act (SAM2 → DINOv2)	80.0 ± 1.0	4.0	95.0 ± 3.3	**100.0** ± 0.0	65.0 ± 3.3	95.0 ± 2.0	86.0 ± 3.8	35.0 ± 5.0	90.0 ± 4.6	**100.0** ± 0.0
SAM2Act (Original Upsampling)	84.2 ± 0.4	3.5	**100.0** ± 0.0	**100.0** ± 0.0	80.0 ± 3.3	96.0 ± 3.3	87.0 ± 3.8	38.0 ± 5.0	91.0 ± 3.8	**100.0** ± 0.0
SAM2Act (w/o Multi-res Input)	85.7 ± 0.3	2.7	99.0 ± 2.0	96.0 ± 0.0	86.0 ± 8.3	98.0 ± 2.3	**99.0** ± 2.0	43.0 ± 10.5	**96.0** ± 0.0	**100.0** ± 0.0
SAM2Act	**86.8** ± 0.5	**2.3**	99.0 ± 2.0	99.0 ± 2.0	84.0 ± 5.7	98.0 ± 2.3	83.0 ± 6.0	**47.0** ± 6.0	93.0 ± 3.8	**100.0** ± 0.0

Method	Put in Cupboard	Put in Drawer	Put in Safe	Screw Bulb	Slide Block	Sort Shape	Stack Blocks	Stack Cups	Sweep to Dustpan	Turn Tap
SAM2Act (SAM2 → SAM)	72.0 ± 8.6	94.0 ± 2.3	**99.0** ± 2.0	**92.0** ± 5.7	97.0 ± 3.8	41.0 ± 3.8	73.0 ± 3.8	71.0 ± 2.0	96.0 ± 3.3	95.0 ± 2.0
SAM2Act (SAM2 → Depth Anything V2)	78.0 ± 2.3	96.0 ± 3.8	95.0 ± 2.0	90.0 ± 2.3	67.0 ± 2.0	45.0 ± 5.0	60.0 ± 3.3	**91.0** ± 2.0	**100.0** ± 0.0	90.0 ± 4.0
SAM2Act (SAM2 → DINOv2)	**80.0** ± 3.3	99.0 ± 2.0	93.0 ± 3.8	89.0 ± 2.0	77.0 ± 2.0	49.0 ± 8.2	69.0 ± 5.0	79.0 ± 6.8	94.0 ± 2.3	92.0 ± 3.3
SAM2Act (Original Upsampling)	69.0 ± 5.0	98.0 ± 2.3	96.0 ± 3.3	84.0 ± 3.3	**99.0** ± 2.0	52.0 ± 3.3	71.0 ± 3.8	80.0 ± 3.3	99.0 ± 2.0	87.0 ± 6.0
SAM2Act (w/o Multi-res Input)	72.0 ± 4.6	**100.0** ± 0.0	96.0 ± 4.6	87.0 ± 2.0	82.0 ± 5.2	54.0 ± 5.2	74.0 ± 2.3	90.0 ± 6.9	97.0 ± 3.8	92.0 ± 4.6
SAM2Act	75.0 ± 3.8	99.0 ± 2.0	98.0 ± 2.3	89.0 ± 2.0	86.0 ± 4.0	**64.0** ± 4.6	**76.0** ± 8.6	78.0 ± 4.0	99.0 ± 2.0	**96.0** ± 5.7

§3.5.3. For all other ablation experiments, the training configuration is kept the same.

Ablation results on RLBench are presented in Table B.5. All variants of SAM2Act exhibit lower performance than the original version. Removing SAM2's multi-resolution image-embedding inputs results in a 1.1% drop in the average success rate. Replacing the entire multi-resolution upsampling with the original convex upsampling leads to a 2.6% decrease. Substituting the SAM2 image encoder with the SAM image encoder [Kirillov et al., 2023] causes a 6.0% drop compared to SAM2Act and a 3.4% drop compared to SAM2Act with the original convex upsampling, where the only differences are the image encoder and some training hyperparameters. In the same setting, we further replace the SAM2 image encoder with more recent image encoders, DINOv2 [Oquab et al., 2023] and Depth Anything V2 [Yang et al., 2024], both of which show a large drop compared to the original SAM2Act. These results indicate that all of our architectural innovations significantly enhance the agent's ability across multiple manipulation tasks.

Table B.6: The 18 RLBench tasks for the multi-task experiment. We report the language template, the average number of extracted keyframes, the number of task variations, and the variation type.

Task name	Language template	Avg. keyframes	# variations	Variation type
put in drawer	"put the item in the __ drawer"	12.0	3	placement
reach and drag	"use the stick to drag the cube onto the __ target"	6.0	20	color
turn tap	"turn __ tap"	2.0	2	placement
slide to target	"slide the block to __ target"	4.7	4	color
open drawer	"open the __ drawer"	3.0	3	placement
put in cupboard	"put the __ in the cupboard"	5.0	9	category
place in shape sorter	"put the __ in the shape sorter"	5.0	5	shape
put money in safe	"put the money away in the safe on the __ shelf"	5.0	3	placement
push buttons	"push the __ button, [then the __ button]"	3.8	50	color
close jar	"close the __ jar"	6.0	20	color
stack block	"stack __ __ blocks"	14.6	60	color, count
place cups	"place __ cups on the cup holder"	11.5	3	count
place wine at rack	"stack the wine bottle to the __ of the rack"	5.0	3	placement
screw bulb	"screw in the __ light bulb"	7.0	20	color
sweep to dustpan	"sweep dirt to the __ dustpan"	4.6	2	size
insert peg	"put the ring on the __ spoke"	5.0	20	color
meat off grill	"take the __ off the grill"	5.0	2	category
stack cups	"stack the other cups on top of the __ cup"	10.0	20	color

`The Colosseum`

We also conducted the same ablation experiments on `The Colosseum` generalization benchmark. The experimental setup remains the same as in Table B.5, except that we did not test the variant of SAM2Act with the original convex upsampling. The results show that removing SAM2's multi-resolution image-embedding inputs leads to a 14.8% drop in performance, representing a relative decrease of 344.2%. This highlights the effectiveness of SAM2's multi-resolution image embeddings in providing robust visual representations, significantly enhancing SAM2Act's generalization ability.

B.1.7 RLBench Tasks

We follow the multi-task, multi-variation simulated experiment setup of PerAct [Shridhar et al., 2023], RVT [Goyal et al., 2023], and RVT-2 [Goyal et al., 2024], using 18 RLBench tasks with 249 unique variations in object placement, color, size, category, count, and shape. A summary of the 18 RLBench tasks is provided in Table B.6. For a more detailed description of each task, please refer to PerAct [Shridhar et al., 2023].

B.1.8 `MemoryBench` Tasks

In the following, we provide details of the `MemoryBench` tasks.

(a) Reopen drawer. The robot is instructed to remember the drawer slot that was initially opened, close it, then press the button on the table, before locating the previously opened drawer to re-open it.

Success metric. The task is considered successful once the initially opened drawer has been re-opened.

Objects. A drawer and a button. *Variation number.* 3. *Keyframes.* 8.

Language instruction. "Close the drawer, then re-open the previously opened drawer while pushing the button in between."

(b) Put block back. The robot is instructed to move the block to the centre, then push the button, and then move the block back to its initial position.

Success metric. The task is considered successful once the initial block has been moved back to its initial pose.

Objects. Four patches, one block, and one button. *Variation number.* 4. *Keyframes.* 11.

Language instruction. "Put the block to the centre and then back to its initial position while pushing the button in between."

(c) Rearrange block. The robot is instructed to move the block in the centre to the empty patch, then press the button, and then move the alternative block to the centre.

Success metric. The task is considered successful once the alternative block has been moved to the centre.

Objects. Two patches, two blocks, and one button. *Variation number.* 2. *Keyframes.* 10.

Language instruction. "Move the block not on the patch to the empty patch, then press the button, then move the block that has not been moved off the patch."

B.1.9 `MemoryBench` Update

We updated the `reopen_drawer` task in `MemoryBench` for the following reasons. During training on the original data, we observed that the gripper often collided with the drawer handle when closing the drawer. To prevent this, we introduced an additional waypoint for the closing motion, mirroring the procedure used

Table B.7: Properties of the real-world tasks. We report the language template, the average number of extracted keyframes, the number of items the robot can interact with, the task variations, and the variation type.

Task name	Language template	# keyframes	# items	# variations	Variation type
(a) turn on the lamp	"turn on the lamp"	4.5	1	1	placement
(b) push buttons in sequence	"push the red button, then the green button"	5	3	1	placement
(c) stack cubes	"stack the __ cube on the __ cube"	4.0	5	3	category, placement
(d) push the right button	"push the button closest to the blue block"	6	3	1	color, placement

for opening the drawer. Consequently, we retrained all policies specifically on this updated task. Furthermore, to standardize the memory window size across all three tasks, we also retrained SAM2Act+ on this task using a window size of 10, which led to improved performance. All results are updated to Table 3.3.

B.1.10 Real-World Experiments

In the following, we provide details of the real-world setup and tasks. Figure B.1 illustrates the real-world setup. Table B.7 summarizes the properties of the real-world tasks.

(a) Turn on the lamp. The robot is instructed to turn on a lamp by rotating its knob.

Success metric. The task is considered successful once the lamp has been turned on by rotating the knob. *Objects.* A single lamp. *Coordination challenges.* High precision is required to properly rotate the knob.

Language instruction. "Turn on the lamp."

(b) Push buttons in sequence. The robot must press the red button first and then the blue button.

Success metric. The task is considered successful if the buttons are pressed in the specified order: red, then blue. A third button is present but should remain unpressed. *Objects.* Three buttons in front of the robot. *Coordination challenges.* Ensuring the robot presses the correct buttons in sequence without pressing the third button.

Language instruction. "Push the red button and then the blue button."

(c) Stack blocks. The robot must place one specified block on top of another specified block.

Success metric. The task is successful if the designated block is stacked on the correct target block. *Objects.* Three single-colored blocks. *Coordination challenges.* Precision in picking and placing, plus



Figure B.1: Real-world robot setup. A Franka Panda robot with a Robotiq gripper. A RealSense D455 depth sensor captures the scene.

correct language understanding to identify which block goes where.

Language instruction. "Stack the <item> block on the <item> block."

(d) Push the same button. The robot must first identify and press the button closest to the blue block, then press the same button again after the block is removed.

Success metric. The task is successful if the robot presses the correct button twice. Pressing the other button at any point results in failure. *Objects.* Two buttons and one blue block (marking proximity). *Coordination challenges.* After the first button press, the blue block is removed; the robot must remember the button location to press it again.

Language instruction. "Push the button that is closest to the blue block. Press the same button again."

Chapter C

Appendix: Moving from Binary Success to Non-Scalar Feedback

C.1 Appendix

C.1.1 Overview

This appendix expands on the chapter with additional details, experiments, and qualitative results. It covers:

- Failure taxonomy with reference coordinate systems (§C.1.2).
- The `FailGen` data-generation pipeline with example configuration files (§C.1.3).
- Additional details on the AHA dataset, including the data distribution and example failure-mode collages (§C.1.4).
- Additional experimental results, including viewpoint, uncertainty, and base-LLM ablations, along with qualitative comparisons (§C.1.5).
- VLM reward-generation rollouts, modified reward functions, and ManiSkill task details (§C.1.6).
- VLM task-plan generation rollouts, modified plans, and PyBullet task details (§C.1.7).
- VLM sub-task verification rollouts and RLBench task details (§C.1.8).

- Limitations and opportunities (§C.1.9).

C.1.2 Failure Taxonomy

We conducted an in-depth study of recent real-world, diverse robot datasets—including Open-X Embodiment [Padalkar et al., 2023], DROID [Khazatsky et al., 2024], and Ego4D [Grauman et al., 2022]—and the policies trained on them. Through this analysis, we identified several common modes of failure, which we categorize into seven types: incomplete grasp, inadequate grip retention, misaligned keyframe, incorrect rotation, missing rotation, wrong action sequence, and wrong target object. Definitions for each mode appear in §4.3.1; here we provide additional implementation details and reference coordinate systems.

Inadequate Grip Retention (`Slip`). For generating the AHA dataset for training and evaluation, we configured a 5-timestep activation for the `Slip` failure mode, triggering the object to drop from the gripper.

Misaligned Keyframe (`Translation`). For the AHA training and evaluation dataset, we introduced a translation offset of $[-0.5, 0.5]$ m. In the ManiSkill-Fail dataset, we applied a translation noise of $[0, 0.1]$ m along either the X, Y, or Z axis from the original waypoint. The translation coordinate system is depicted in Figure C.1 (left).

Incorrect Rotation (`Rotation`). For the AHA dataset, we set a rotation offset of $[-3.14, 3.14]$ rad along roll, yaw, or pitch. The rotation coordinate system is depicted in Figure C.1 (right).

Wrong Target Object (`Wrong_Object`). We perform a sweep through all manipulable objects, swapping them with the target object in the scene.

C.1.3 FailGen Data-Generation Pipeline

We developed `FailGen`, an environment wrapper that can be easily integrated into any simulator. It leverages pre-defined or hand-crafted robot demonstrations for imitation learning, where each trajectory is represented as a waypoint-based system. Two consecutive waypoints form a sub-task, with each sub-task linked to a predefined set of language descriptions. `FailGen` allows for modifications to environment parameters,

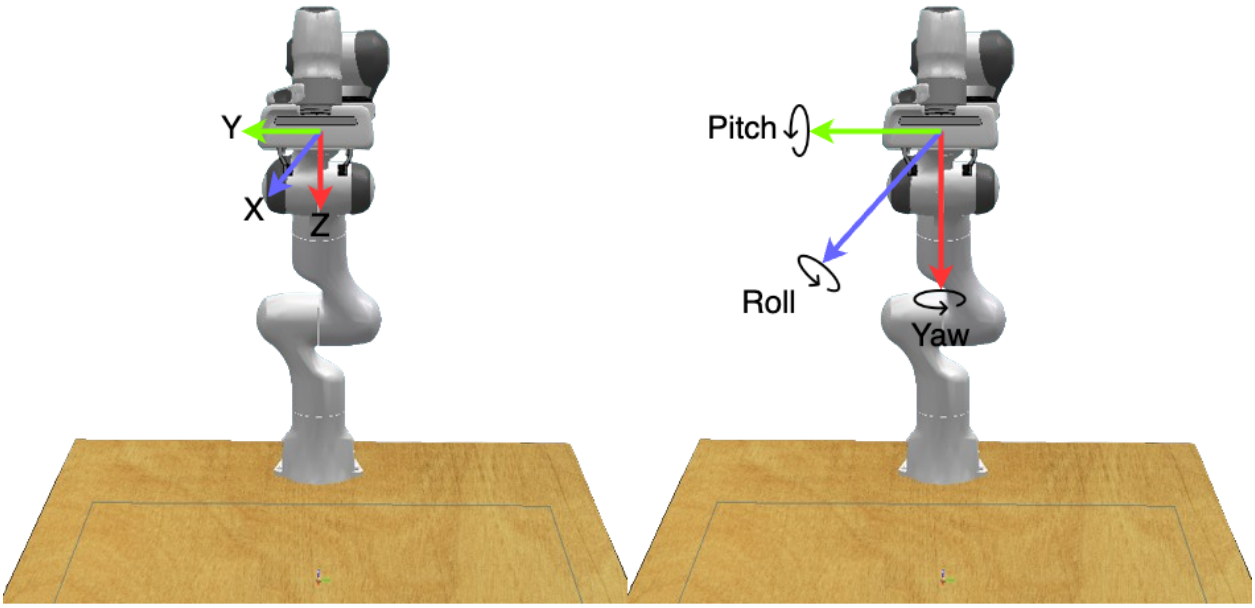


Figure C.1: Failure-mode reference coordinate systems. (Left) translation coordinate system; (right) rotation coordinate system.

such as gripper end-effector translation, rotation, and open/close state. By altering these parameters, we systematically generate failures at every waypoint. However, for the 79 tasks collected from RLBench, we do not initially know which sub-task will fail due to specific failure modes. To address this, we perform a systematic sweep, using RLBench's built-in success conditions to explore all possible combinations. This generates a configuration of failures for each task, which we then use to procedurally generate all failure training data. Additionally, we manually annotate each sub-task with natural-language instructions describing the task, and pair this with failure-mode explanations to serve as language input for instruction-tuning. Examples of the configuration files are depicted in Figure C.2.

C.1.4 AHA Dataset

Using `FailGen`, we curated two datasets from RLBench [James et al., 2020]. The first is the training dataset, AHA (Train), which is used for instruction-tuning AHA alongside the co-train dataset. The second is the testing dataset, AHA (Test), used for evaluation. AHA (Train) contains approximately 49K image–query pairs of failures derived from 79 tasks, while AHA (Test) consists of approximately 11K image–query pairs

```yaml
save_path: /home/${oc.env:USER}/data/failgen_data
obs_mode: rgb
render_mode: sensors
shader: default
sim_backend: auto
image_size: [256, 256]
stages: [0, 1, 2, 3]
failures:
  - type: grasp
    enabled: false
    stages: [2]
  - type: trans_x
    enabled: false
    stages: [0, 1, 3]
    noise: 0.1
  - type: trans_y
    enabled: false
    stages: [0, 1, 3]
    noise: 0.1
  - type: trans_z
    enabled: false
    stages: [0, 1, 3]
    noise: 0.1

```

(a) Example config file of one task for ManiSkill-Fail.

```yaml
data:
  # Where to save the demos
  save_path: /home/data
  # The size of the images to save
  waypoints: [0, 1, 2, 3]
failures:
  - type: grasp
    name: failure_grasp_pose
    enabled: False
    waypoints: [1]

  - type: translation_y
    name: trans_y
    enabled: False
    waypoints: [1,2,3]
    range: [-0.5, 0.5]

  - type: rotation_x
    name: rot_x
    enabled: False
    waypoints: [0]
    range: [-1.57, 1.57]

  - type: wrong_sequence
    name: bad_seq
    enabled: False
    waypoints: [2,3]

sub-tasks:
  - task_no: 0
    enabled: False
    type: dummy
    targets: [ball]
    processes: [waypoint0, waypoint1]
    task_description: [
      "grasp onto the clock knob",
      "pick on the clock knob"
    ]
  - task_no: 1
    enabled: False
    type: dummy
    targets: [ball]
    processes: [waypoint1, waypoint2]
    task_description: [
      "rotate the knob",
      "turn the knob"
    ]
  - task_no: 2
    enabled: False
    type: dummy
    targets: [ball]
    processes: [waypoint2, waypoint3]
    task_description: [
      "let go",
      "release the gripper"
    ]
```

(b) Example config file for an AHA task.

Figure C.2: Example `FailGen` configuration files.

from 10 hold-out tasks.

C.1.5 Additional Experimental Results

We conducted additional experiments to better understand and visualize AHA's predictions. We trained two versions of the AHA model with 13B parameters, using different language models for fine-tuning: LLaMA-2-13B and Vicuna-1.5-13B. The results show less than a 2.5% performance difference between the two models, indicating that our fine-tuning data is effective regardless of the base language model. Additionally, we visualize output predictions from various baselines compared to our model and evaluate performance across all three datasets.

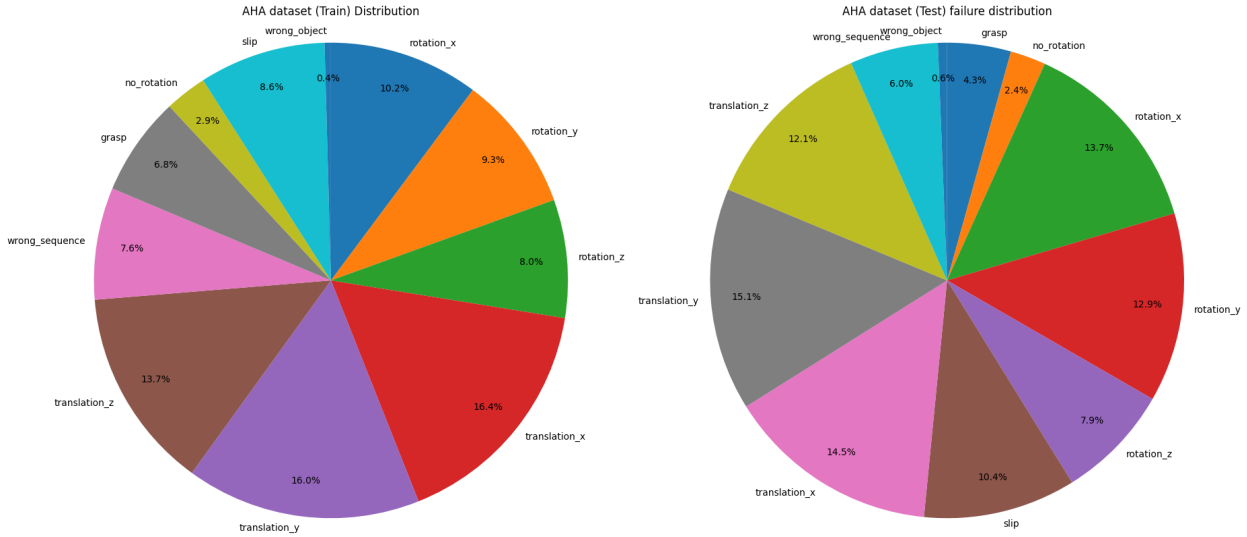


Figure C.3: Data distribution of the AHA dataset for both train and test.

AHA model performance uncertainty estimation. To evaluate the relationship between uncertainty estimation and model performance, we conducted additional experiments across various evaluation datasets. Specifically, we compared the sentence-token prediction probabilities of AHA-13B with those of LLaVA-v1.5-13B. AHA-13B exhibits significantly higher average prediction probabilities, reflecting its superior accuracy. These findings underscore the positive impact of fine-tuning with the AHA failure dataset on model confidence and performance.

Effects of viewpoints on evaluation. We evaluate the reasoning capabilities of AHA on the ManiSkill-Fail dataset across three different viewpoint configurations (one, two, and three viewpoints). Interestingly, we observe a slight performance advantage when using single-viewpoint images. We attribute this to the resolution limitations of the LLaVA-v1.5 visual encoder (256×256 px), where single-viewpoint inputs offer clearer and more focused visual information for failure reasoning.

C.1.6 Downstream Robotic Application: VLM Reward Generation

In this section, we present reward functions generated by GPT-4o and AHA for comparison, as shown in Figure C.5. Additionally, we demonstrate RL policy rollouts improved through AHA's failure feedback across all five tasks along with all the final dense reward functions modified by AHA, shown in Figures C.6 and C.7.

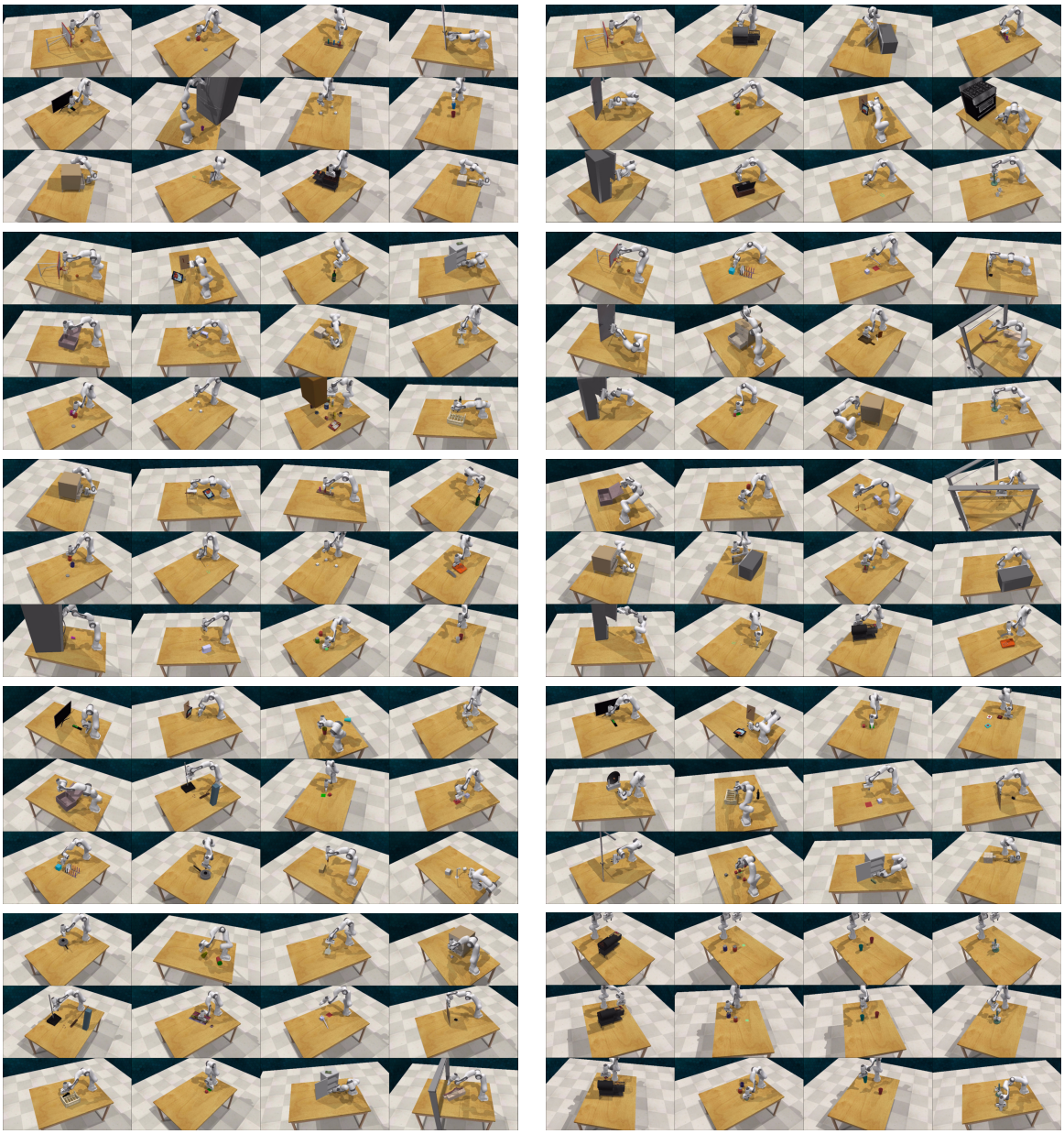


Figure C.4: Examples of different failure modes. Row 1: `No_Grasp` and `Rotation_x`. Row 2: `Rotation_y` and `Rotation_z`. Row 3: `Slip` and `Wrong_sequence`. Row 4: `Translation_x` and `Translation_y`. Row 5: `Translation_z` and `Wrong_object`.

For all tasks, except `put_sphere_on_holder` (trained with PPO for 10M steps), PPO is trained for 25M steps prior to reflection and evaluation. We describe each of the five tasks in detail below, along with their ManiSkill variants and success conditions.

`pickup_YCB`. (`pick_single_ycb.py`) Pick up a single YCB object and lift it to a target height. Success: the object position is within `goal_thresh` (default 0.025 m) Euclidean distance of the goal position.

`push_T`. (`push_T.py`) Push a T-shaped block into a T-shaped target area. Success: the 3D T block covers at least 90% of the 2D T goal zone.

`place_sphere`. (`place_sphere_v1.py`) Pick up a sphere and place it into a bin. Success: the sphere is on top of the bin (i.e., its xy-distance to the bin is near zero and its z-distance is near the sphere radius plus the bottom bin block's side length); the object is static (linear and angular velocities are bounded by a small value); and the gripper is not grasping the object.

`stack_cube`. (`stack_cube_v1.py`) Pick up the red cube and stack it onto the green cube. Success: the red cube is on top of the green cube (within half of the cube size), the red cube is static, and the red cube is not being grasped by the robot.

`open_drawer`. (`open_cabinet_drawer_v1.py`) Pull open a drawer. Success: the drawer is open at least 90% of the way, and the angular/linear velocities of the drawer link are small.

C.1.7 Downstream Robotic Application: VLM Task-Plan Generation

In this section, we present policy rollouts improved by AHA in Figure C.8, along with the modified task plans in Figure C.9. We describe each of the three tasks in detail below, along with their PyBullet variants and success conditions.

`put_banana_centre`. (`ours_raven_ycb_pick.py`) Pick up the banana and place it onto the centre of the table. Success: the final location of the banana with respect to the table area satisfies the success

```diff
 def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):
     # Calculate the distance between the TCP and cubeA
     tcp_to_cubeA_dist = torch.linalg.norm(self.cubeA.pose.p - self.agent.tcp.pose.p, axis=1)
     reaching_reward = 1 - torch.tanh(5 * tcp_to_cubeA_dist)
     reward = reaching_reward
-
+
     # Check if cubeA is grasped
-    is_cubeA_grasped = info["is_cubeA_grasped"]
+    is_cubeA_grasped = info["""is_cubeA_grasped"""]
     reward += is_cubeA_grasped
-
+
+    # Add a lifting reward to encourage the agent to lift CubeA once it's grasped
+    cubeA_height_reward_lifting = torch.heaviside(self.cubeA.pose.p[..., 2] - self.cubeB.pose.p[..., 2] - 0.1, torch.tensor(0.0))
+    reward += cubeA_height_reward_lifting * is_cubeA_grasped
+
     # Calculate the distance between cubeA and cubeB
     cubeA_to_cubeB_dist = torch.linalg.norm(self.cubeB.pose.p - self.cubeA.pose.p, axis=1)
     stacking_reward = 1 - torch.tanh(5 * cubeA_to_cubeB_dist)
     reward += stacking_reward * is_cubeA_grasped

     # Add a positional reward component to encourage lifting CubeA
     cubeA_height_reward = self.cubeA.pose.p[..., 2]
-
+
     reward += cubeA_height_reward * is_cubeA_grasped
-
+
     # Static reward for cubeA being static when placed on cubeB
     static_reward = 1 - torch.tanh(5 * torch.linalg.norm(self.agent.robot.get_qvel()[..., :-2], axis=1))
-    reward += static_reward * info["is_cubeA_on_cubeB"]
-
-    reward += static_reward * info["""is_cubeA_on_cubeB"""]
-
+
     # Bonus reward for successful stacking
-    reward[info["success"]] = 8
-    reward[info["""success"""]] = 8
+    reward[info["success"]] = 8
     return reward
```

(a) Improved dense reward function using GPT-4o for reflection.

```diff
 def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):
+
     # Distance from the TCP (Tool Center Point) to cubeA
     tcp_to_cubeA_dist = torch.linalg.norm(
     self.cubeA.pose.p - self.agent.tcp.pose.p, axis=1
     )
     reaching_reward = 1 - torch.tanh(5 * tcp_to_cubeA_dist)
     reward = reaching_reward
-
+
     # Reward for grasping cubeA
     is_cubeA_grasped = info["is_cubeA_grasped"]
     reward += is_cubeA_grasped
-
-    # Distance from cubeA to cubeB
-    cubeA_to_cubeB_dist = torch.linalg.norm(
-    self.cubeB.pose.p - self.cubeA.pose.p, axis=1
-    )
+
     # Distance from cubeA to cubeB along the x and y directions (ignore z)
     cubeA_to_cubeB_dist_xy = torch.linalg.norm(
     self.cubeB.pose.p[..., :2] - self.cubeA.pose.p[..., :2], axis=1
     )
-    stacking_reward = 1 - torch.tanh(5 * cubeA_to_cubeB_dist)
-    reward += stacking_reward * is_cubeA_grasped
-
+
     stacking_reward_xy = 1 - torch.tanh(5 * cubeA_to_cubeB_dist_xy)
     reward += stacking_reward_xy * is_cubeA_grasped

     # Penalty for misalignment in the z direction
     z_offset_penalty = torch.abs(self.cubeB.pose.p[..., 2] - self.cubeA.pose.p[..., 2])
     reward -= z_offset_penalty * is_cubeA_grasped

     # Reward for keeping cubeA static
     is_cubeA_static = info["is_cubeA_static"]
     static_reward = 1 - torch.tanh(5 * torch.linalg.norm(self.agent.robot.get_qvel()[..., :-2], axis=1))
     reward += static_reward * info["is_cubeA_on_cubeB"]
-
+
     # Final success reward
     reward[info["success"]] = 8
     return reward
```

(b) Improved dense reward function using AHA for reflection.

Figure C.5: Example improved dense reward functions. (Left) GPT-4o reflection. (Right) AHA reflection.

condition.

`stack_banana`. (`ours_ycb_banana_spam_stack.py`) Pick up the banana and place it onto the spam can. Success: the banana rests stably on top of the spam can.

`stack_cubes`. (`ours_raven_bowl_stack.py`) Pick up the green cube and place it into the green bowl, then take the yellow cube and stack it on top of the green cube. Success: the yellow cube is stably stacked on top of the green cube in the green bowl.

C.1.8 Downstream Robotic Application: VLM Sub-Task Verification

In this section, we leverage `Manipulate-Anything` [Duan et al., 2024b] as the main policy framework, integrating it with AHA. AHA functions as a sub-task verifier VLM, playing a crucial role in ensuring task success when using `Manipulate-Anything`. Examples of the rollouts are shown in Figure C.10. We describe each of the four tasks in detail below, along with their RLBench variants and success conditions.

put_block. (put_block.py) Pick up the green block and place it on the red mat. Success: the success condition on the red mat detects the target green block.

pickup_cup. (pickup_cup.py) Pick up the red cup. Success: lift the red cup above the pre-defined location.

sort_mustard. (sort_mustard.py) Pick up the yellow mustard bottle and place it into the red container. Success: the yellow mustard bottle is inside the red container.

pick_and_lift. (pick_and_lift.py) Pick up the red cube. Success: the red cube is lifted up.

C.1.9 Limitations and Opportunities

While AHA is quite capable, it is not without limitations. In this section, we discuss some of these limitations and potential risks for real-world deployment.

Does not cover all failure cases. While AHA is instruction-tuned to address the seven most common failure modes in robotic tabletop manipulation, and provides fine-grained failure identification down to axis-level translation or orientation offsets of the gripper, it is inevitable that certain edge cases remain uncovered. These edge cases fall outside the distribution of failure modes considered in our approach and may occur in real-world scenarios. To address this limitation, future work could focus on distilling policies already trained with action-based "artifacts" to generate naturally occurring failures. These could then be labeled using human annotations or a hybrid approach combining AHA and a GPT-4o-based labeling system. Such efforts could enhance the robustness and applicability of failure detection across a broader range of scenarios.

Improving policies directly with AHA. Currently, we have demonstrated AHA's effectiveness in enhancing existing robotic systems that utilize off-the-shelf VLMs/LLMs to automate various pipeline components. AHA has shown significant impact in improving code generation, task-plan generation, and sub-task success detection. However, AHA is not yet capable of directly influencing low-level trained policies through failure language reasoning. A potential next step would involve training low-level policies with language-conditioned demonstrations covering a diverse range of corrective actions tied to various failure modes. This would

enable low-level policies to interpret failure reasoning in a counterfactual manner and generate corrective actions directly.

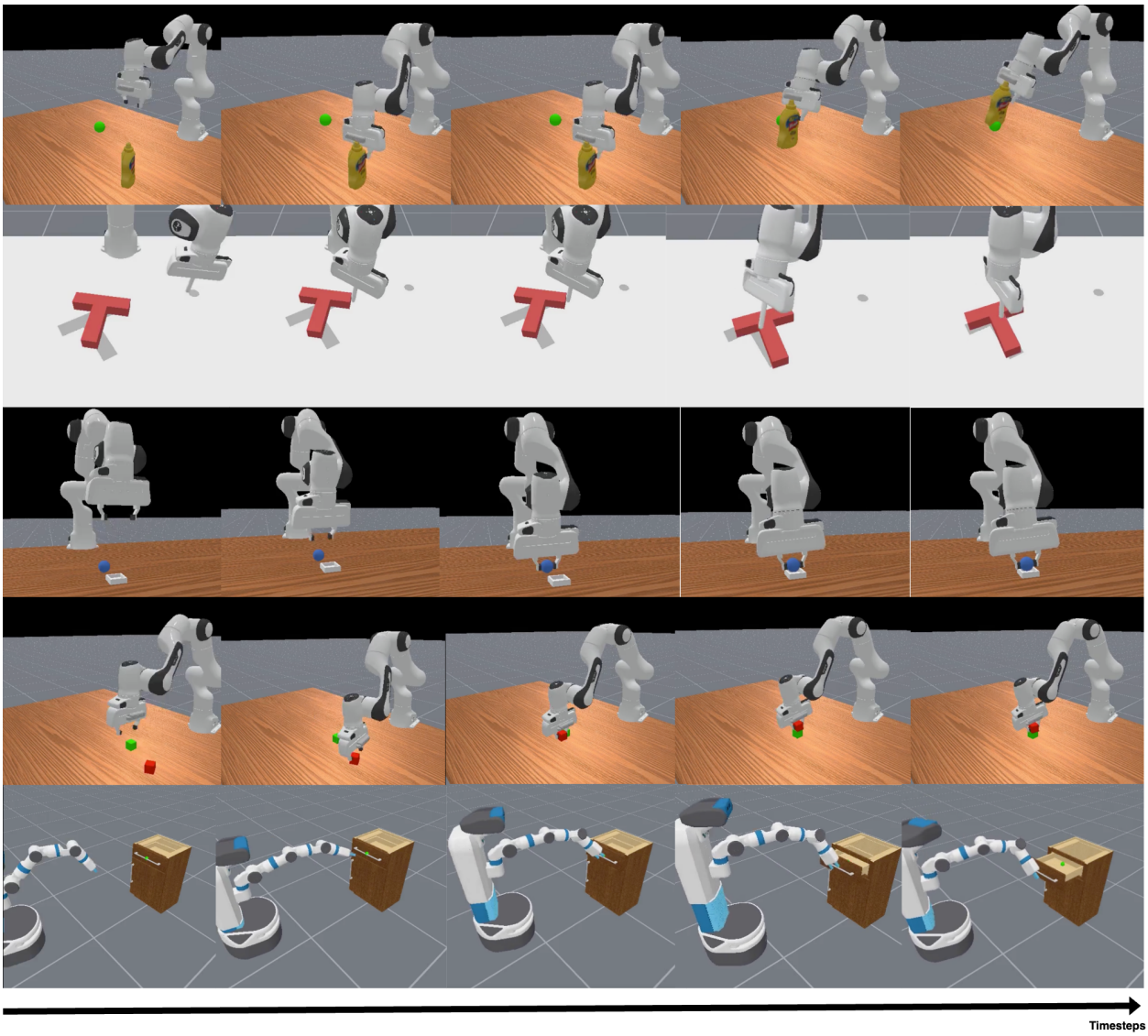


Figure C.6: RL policy rollouts improved with AHA. Row 1: `pickup_YCB`. Row 2: `push_T`. Row 3: `place_sphere`. Row 4: `stack_cube`. Row 5: `open_drawer`.

"Push T to shaped T area"

```
def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):


    # Calculate the distance from the TCP to the T-shaped peg
    tcp_to_obj_dist = torch.linalg.norm(self.tee.pose.p - self.agent.tcp.pose.p,
axis=1)
    reaching_reward = 1 - torch.tanh(5 * tcp_to_obj_dist)
    reward = reaching_reward


    # Calculate the pseudo-rendered intersection area for the T-shaped peg in
the goal region
    intersection_area = self.pseudo_render_intersection()
    intersection_reward = intersection_area / self.intersection_thresh
    reward += intersection_reward


    # Calculate the distance from the TCP to the end-effector goal position
    tcp_to_goal_dist = torch.linalg.norm(self.ee_goal_pos.pose.p -
self.agent.tcp.pose.p, axis=1)
    end_effector_goal_reward = 1 - torch.tanh(5 * tcp_to_goal_dist)
    reward += end_effector_goal_reward * (intersection_area >=
self.intersection_thresh)


    # Adding a penalty term for misalignment along the x direction
    tcp_to_obj_x_dist = torch.abs(self.tee.pose.p[:, 0] -
self.agent.tcp.pose.p[:, 0])
    alignment_penalty = -0.5 * torch.tanh(5 * tcp_to_obj_x_dist)
    reward += alignment_penalty


    # Reward for achieving the task success condition
    reward[info["success"]] = 3.0
    return reward
```

"Stack the red cube onto green cube"

```
def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):
    # Distance from the TCP (Tool Center Point) to cubeA
    tcp_to_cubeA_dist = torch.linalg.norm(
    self.cubeA.pose.p - self.agent.tcp.pose.p, axis=1
    )
    reaching_reward = 1 - torch.tanh(5 * tcp_to_cubeA_dist)
    reward = reaching_reward

    # Reward for grasping cubeA
    is_cubeA_grasped = info["is_cubeA_grasped"]
    reward += is_cubeA_grasped

    # Distance from cubeA to cubeB along the x and y directions (ignore z)
    cubeA_to_cubeB_dist_xy = torch.linalg.norm(
    self.cubeB.pose.p[..., :2] - self.cubeA.pose.p[..., :2], axis=1
    )
    stacking_reward_xy = 1 - torch.tanh(5 * cubeA_to_cubeB_dist_xy)
    reward += stacking_reward_xy * is_cubeA_grasped

    # Penalty for misalignment in the z direction
    z_offset_penalty = torch.abs(self.cubeB.pose.p[..., 2] -
self.cubeA.pose.p[..., 2])
    reward -= z_offset_penalty * is_cubeA_grasped

    # Reward for keeping cubeA static
    is_cubeA_static = info["is_cubeA_static"]
    static_reward = 1 - torch.tanh(5 *
torch.linalg.norm(self.agent.robot.get_qvel()[..., :-2], axis=1))
    reward += static_reward * info["is_cubeA_on_cubeB"]

    # Final success reward
    reward[info["success"]] = 8
    return reward
```

"Put sphere into sphere holder"

```
def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):


    # Compute the distance from the TCP (gripper) to the object (sphere)
    tcp_to_obj_dist = torch.linalg.norm(self.obj.pose.p - self.agent.tcp.pose.p,
axis=1)
    reaching_reward = 1 - torch.tanh(5 * tcp_to_obj_dist)
    reward = reaching_reward

    # Add grasping reward if the object is being grasped
    is_grasped = info["is_obj_grasped"]
    reward += is_grasped

    # Compute the distance from the object (sphere) to the bin
    obj_to_bin_dist = torch.linalg.norm(self.bin.pose.p - self.obj.pose.p,
axis=1)
    placing_reward = 1 - torch.tanh(5 * obj_to_bin_dist)
    reward += placing_reward * is_grasped

    # Add an additional z-axis penalty to ensure precise placement into the bin
    z_offset_penalty = torch.abs(self.bin.pose.p[..., 2] - self.obj.pose.p[...,
2])
    reward -= 5 * torch.tanh(5 * z_offset_penalty) * is_grasped

    # Add reward if the object is on the bin and the robot is static
    reward += info["is_obj_static"] * info["is_obj_on_bin"]

    # Set reward to maximum if the task is successfully completed
    reward[info["success"]] = 13

    return reward
```

"Pick up the mustard bottle"

```
def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):
        # Distance from TCP to object
        tcp_to_obj_dist = torch.linalg.norm(
            self.obj.pose.p - self.agent.tcp.pose.p, axis=1
        )
        reaching_reward = 1 - torch.tanh(5 * tcp_to_obj_dist)
        reward = reaching_reward

        # Reward for grasping the object
        is_grasped = info["is_grasped"]
        reward += is_grasped

        # Introducing a penalty for missing the grasp
        missed_grasp_penalty = torch.where(is_grasped == 0, -0.5, 0.0)
        reward += missed_grasp_penalty

        # Reward for grasping with proper rotation (introducing a reward for
rotation alignment)
        rotation_alignment = 1 - torch.tanh(5 * torch.abs(self.agent.tcp.pose.q -
self.obj.pose.q).sum())
        reward += rotation_alignment * is_grasped

        # Introducing a penalty for misalignment during grasp
        misalignment_penalty = torch.where(rotation_alignment < 0.5, -0.5, 0.0)
        reward += misalignment_penalty * is_grasped

        # Distance from object to goal
        obj_to_goal_dist = torch.linalg.norm(
            self.goal_site.pose.p - self.obj.pose.p, axis=1
        )
        place_reward = 1 - torch.tanh(5 * obj_to_goal_dist)
        reward += place_reward * is_grasped

        # Reward for lifting the object
        lift_reward = torch.where(
            self.obj.pose.p[:, 2] > 0.1, 1.0, 0.0
        )  # Assuming 0.1 as the threshold height for lifting
        reward += lift_reward * is_grasped

        # Final success reward
        reward[info["success"]] = 6
        return reward
```

"Pick up the mustard bottle"

```
def compute_dense_reward(self, obs: Any, action: torch.Tensor, info: Dict):
    # Compute the distance between the TCP (tool center point) and the handle
    tcp_to_handle_dist = torch.linalg.norm(
        info["handle_link_pos"] - self.agent.tcp.pose.p, axis=1
    )
    reaching_reward = 1 - torch.tanh(5 * tcp_to_handle_dist)
    reward = reaching_reward

    # Compute the reward for grasping the handle
    is_grasped = self.agent.is_grasping(self.handle_link)
    reward += is_grasped

    # Compute the distance the drawer has been pulled out
    open_frac = (self.handle_link.joint.qpos - self.handle_link.joint.limits[..., 0]) / (
        self.handle_link.joint.limits[..., 1] - self.handle_link.joint.limits[..., 0]
    )
    open_reward = 1 - torch.tanh(5 * (self.min_open_frac - open_frac))
    reward += open_reward * is_grasped

    # Encourage a progressive motion of opening the drawer: add positive reward only if
drawer's open_frac is increasing
    drawer_change = open_frac - getattr(self, '_prev_open_frac', open_frac)
    progressive_reward = drawer_change.where(drawer_change > 0, torch.tensor(0.0))
    reward += progressive_reward * 10  # Multiplier adjusted to emphasize its importance
    self._prev_open_frac = open_frac


    # Compute the static reward to encourage the agent to keep the drawer open
    link_is_static = (
        torch.linalg.norm(self.handle_link.angular_velocity, axis=1) <= 1
    ) & (torch.linalg.norm(self.handle_link.linear_velocity, axis=1) <= 0.1)
    static_reward = 1 - torch.tanh(5 * (1 - link_is_static.float()))
    reward += static_reward * info["open_enough"]


    # Penalty for y-direction offset movement after grasping
    y_offset_penalty = torch.abs(self.agent.tcp.pose.p[:, 1] - info["handle_link_pos"][:,
1])
    reward -= y_offset_penalty * is_grasped


    # Add a large reward if the task is successfully completed
    reward[info["success"]] = 5.0
    return reward
```

Figure C.7: Examples of modified reward functions via AHA.

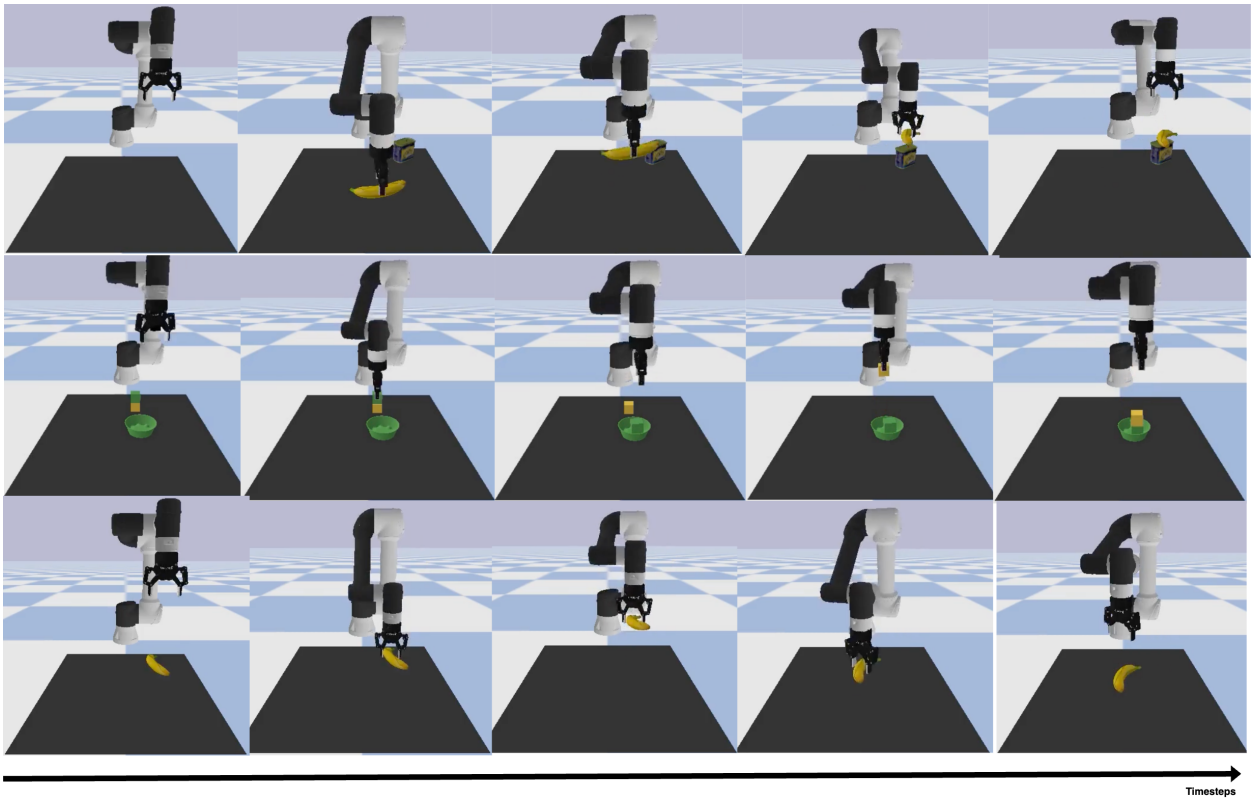


Figure C.8: TAMP policy rollouts improved with AHA. Row 1: `put_banana_centre`. Row 2: `stack_banana`. Row 3: `stack_cubes`.

"Stack banana onto spam can"

````
Thank you for the clarification. Given that the robot gripper moves to the desired position with an offset along the z direction,
we need to account for this offset when placing the banana on top of the potted meat can.
Let's assume the offset is a known constant, `GRIPPER_OFFSET_Z`. We'll use this offset to adjust the placement pose accordingly.
Here's the updated `gen_plan` and `gen_domain` functions:


```python
GRIPPER_OFFSET_Z = 0.02  # Example offset value, adjust as necessary


def gen_plan(initial: RavenBelief, grasp: RavenGrasp):
    plan = []

    # Pick up the banana
    plan.append(Action("pick", ["object_1", grasp]))

    # Get the pose of the potted meat can
    spam_pose = initial.objects["object_0"].pose

    # Calculate the placement pose for the banana on top of the potted meat can
    banana_place_pose = RavenPose(
        x=spam_pose.x,
        y=spam_pose.y,
        z=spam_pose.z + BLOCK_SIZE + GRIPPER_OFFSET_Z,  # Adjust for the gripper offset
        roll=spam_pose.roll,
        pitch=spam_pose.pitch,
        yaw=spam_pose.yaw
    )

    # Place the banana on top of the potted meat can
    plan.append(Action("place", ["object_1", grasp, banana_place_pose]))

    return plan


def gen_domain(initial: RavenBelief):
    return {
        "grasp": GraspSampler()
    }
```


### Explanation


1. **Picking the Banana**: The plan starts by picking up the banana (`object_1`) using a grasp sampled from the `GraspSampler`.
2. **Calculating Placement Pose**: The pose of the potted meat can (`object_0`) is retrieved from the initial state. The placement
pose for the banana is calculated by adding the height of the banana (approximated as `BLOCK_SIZE`) and the gripper offset
(`GRIPPER_OFFSET_Z`) to the z-coordinate of the potted meat can's pose. The roll, pitch, and yaw are kept the same to ensure proper
alignment.
3. **Placing the Banana**: The plan then places the banana at the calculated pose on top of the potted meat can.


### Why This Will Always Satisfy the Goal


- **Grasp Sampling**: The grasp is sampled from a `GraspSampler`, ensuring that a valid grasp is used for picking up the banana.
- **Placement Calculation**: The placement pose is calculated based on the current pose of the potted meat can and adjusted for the
gripper offset, ensuring that the banana is placed directly on top of it.
- **Generalization**: The plan is general enough to handle different initial positions and orientations of the objects, as it
dynamically calculates the placement pose based on the initial state.


This approach ensures that the banana will always be stacked on top of the potted meat can, accounting for the gripper offset along
the z direction.
````

"Stack two cubes into a bowl"

````
Thank you for pointing that out. Let's correct the plan to account for the z-offset when placing the second cube.
```python
def gen_plan(initial:RavenBelief, x_offset:float, y_offset:float, z_offset:float):
    plan = []


    # Pick up the green block
    green_block = initial.objects["object_6"]
    plan.append(Action("pick", green_block.pose.point))


    # Place the green block into the green bowl
    green_bowl = initial.objects["object_5"]
    x, y, z = green_bowl.pose.point
    plan.append(Action("place", [x + x_offset, y + y_offset, z + z_offset]))


    # Pick up the yellow block
    yellow_block = initial.objects["object_4"]
    plan.append(Action("pick", yellow_block.pose.point))


    # Stack the yellow block onto the green block
    green_block_new_pose = [x + x_offset, y + y_offset, z + z_offset + BLOCK_SIZE]
    plan.append(Action("place", [green_block_new_pose[0], green_block_new_pose[1], green_block_new_pose[2] + BLOCK_SIZE]))


    return plan

def gen_domain(initial:RavenBelief):
    return {
        "x_offset": ContinuousSampler(-0.02, 0.02),
        "y_offset": ContinuousSampler(-0.02, 0.02),
        "z_offset": ContinuousSampler(0.01, 0.03),
    }
```
### Explanation

1. **Pick up the green block**: The plan starts by picking up the green block (`object_6`).
2. **Place the green block into the green bowl**: The green block is then placed into the green bowl (`object_5`). The offsets
(`x_offset`, `y_offset`, `z_offset`) allow for some variability in the exact placement within the bowl.
3. **Pick up the yellow block**: The plan continues by picking up the yellow block (`object_4`).
4. **Stack the yellow block onto the green block**: Finally, the yellow block is placed on top of the green block, which is now in
the bowl. The height is adjusted by adding the `BLOCK_SIZE` to the z-coordinate of the green block's new position, ensuring the
yellow block is stacked correctly.


### Why the Function Will Always Satisfy the Goal

- The `x_offset`, `y_offset`, and `z_offset` parameters allow for slight variations in the placement of the green block within the
bowl, ensuring it fits regardless of minor positional differences.
- The yellow block is always placed directly on top of the green block, ensuring a stack is formed.
- The domain for the offsets is chosen to be small enough to ensure the blocks are placed correctly but large enough to allow for
some variability.


### Constraints and Potential Failures

- If the offsets are too large, the green block might not be placed correctly within the bowl, or the yellow block might not stack
properly on the green block.
- The domain for the offsets is chosen to minimize these risks, but if a failure occurs, the offsets can be adjusted to smaller
ranges.

This plan should achieve the goal of placing the green block into the bowl and then stacking the yellow block onto the green block,
given the initial state and the defined domain.
````

"Move banana to centre"

````
To account for the slight offset along the y direction when grasping the object, we can adjust the placement position accordingly.
We will add an additional parameter to the `gen_plan` function to specify the y offset and use it to correct the placement
position.


Here's the updated implementation:


```python
def gen_plan(initial: RavenBelief, grasp: RavenGrasp, y_offset: float):
    plan = []
    object_name = "object_0"  # The banana is identified as "object_0"
    plan.append(Action("pick", [object_name, grasp]))
    # Adjust the y position by the offset when placing the object
    adjusted_y = TABLE_CENTER[1] + y_offset
    plan.append(Action("place", [object_name, grasp, RavenPose(x=TABLE_CENTER[0], y=adjusted_y, z=0.02)]))
    return plan


def gen_domain(initial: RavenBelief):
    return {
        "grasp": GraspSampler(),
        "y_offset": ContinuousSampler(-0.05, 0.05)  # Allowing for a small offset range
    }
```


### Explanation:
1. **gen_plan Function**:
   - The function now takes an additional parameter `y_offset` to account for the slight offset along the y direction.
   - When placing the object, the y position is adjusted by adding the `y_offset` to the table center's y coordinate.


2. **gen_domain Function**:
   - The function returns a dictionary with two entries:
     - `grasp`: Sampled using the `GraspSampler`.
     - `y_offset`: Sampled using a `ContinuousSampler` with a range of -0.05 to 0.05, allowing for a small adjustment in the y
direction.


### Why This Will Always Satisfy the Goal:
- The plan still explicitly picks up the banana and places it at the center of the table, but now with an adjustment for the y
offset.
- The `y_offset` parameter allows for correcting the placement position to account for the slight offset during grasping.
- The table center coordinates are fixed and known, ensuring the banana is always moved to the correct location with the necessary
adjustment.


This approach ensures that the goal is achieved regardless of the specific grasp sampled and the slight offset during grasping, as
long as the offset is within the specified range.
````

Figure C.9: Examples of modified task plans via AHA.

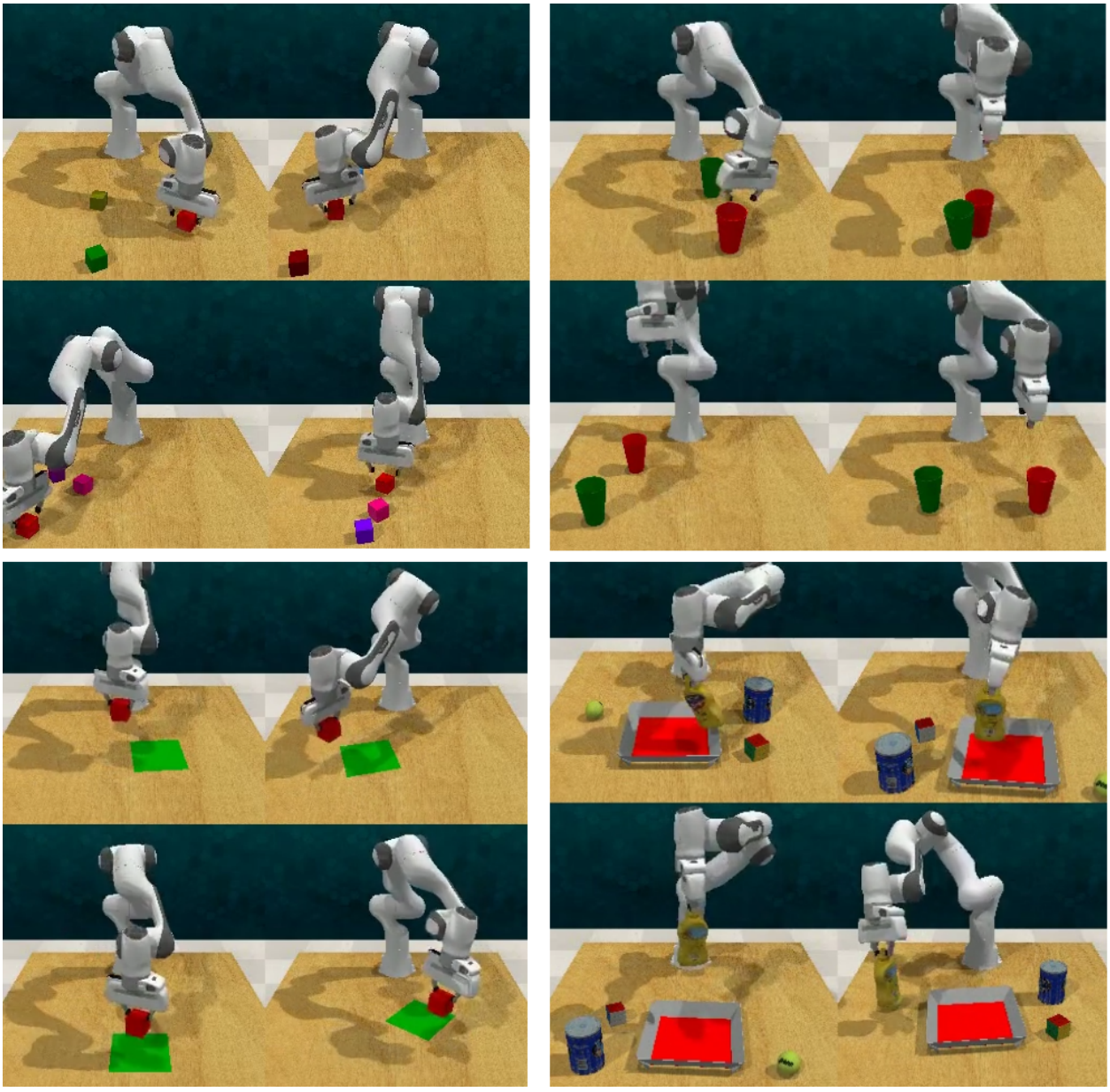


Figure C.10: Examples of zero-shot data-generator trajectories with AHA as sub-task verifier. Row 1: `pickup_cube`, `pickup_cup`. Row 2: `put_block`, `sort_mustard`.