Towards Efficient and Scalable Robot Learning: Trajectory Pre-training, Synthetic Data, and Coding Agents

by

Letian Fu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ken Goldberg, Chair
Professor Pieter Abbeel
Professor Angjoo Kanazawa
Jim Fan

Spring 2026

Towards Efficient and Scalable Robot Learning:
Trajectory Pre-training, Synthetic Data, and Coding Agents

Abstract

Towards Efficient and Scalable Robot Learning:
Trajectory Pre-training, Synthetic Data, and Coding Agents

by

Letian Fu

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Ken Goldberg, Chair

Modern vision and language models are powerful because they are not trained from scratch for every task. They are pre-trained on large, diverse datasets, adapted through post-training, and increasingly deployed as agents that use tools, feedback, and test-time computation. Robots, however, do not yet benefit from the same scaling paradigm. Robot data is expensive to collect, tied to specific hardware and environments, and difficult to obtain at the scale required for general pre-training. As a result, many robot learning systems remain data-hungry, task-specific, and brittle under changes in objects, scenes, embodiments, and task instructions. This dissertation explores how principles from foundation model training and deployment can be adapted to robot learning, with the goal of building robot systems that are more efficient, scalable, and generalizable.

In this dissertation, I study how to make robot learning both efficient and scalable by approaching the problem along three complementary axes: (1) trajectory-level sensorimotor pre-training to improve post-training efficiency and enable few-shot generalization, (2) synthetic robot trajectory data generation techniques that do not rely on human teleoperation of robots, and (3) coding agents for robotic control, strengthened through structured agent harnesses and reinforcement learning with verifiable rewards.

Together, these methods demonstrate how principles from large-scale foundation model training and deployment can be adapted to build general, data-efficient, and scalable robot foundation models.

To my family, mentors, and collaborators.

Contents

List of Figures

List of Tables

Acknowledgments

I began my time at UC Berkeley in 2018 as an undergraduate student. Looking back, the past eight years have been surreal. They have been the most transformative years of my life, and I can say with confidence that choosing Berkeley was the best decision I have ever made. Berkeley has shaped not only my research interests and professional path, but also the way I think, work, and see the world. I have learned to love the intensity and intellectual energy of this place, and the gorgeous physical environment: Berkeley Way West, Campbell Hall, the Marina, Tilden Park, the Mathematical Sciences Research Institute on the hill, and the many beautiful places nearby where I have found peace in times of chaos and challenges.

I am deeply grateful to my advisor, Professor Ken Goldberg, for taking the opportunity to mentor me when I was still an undergraduate, amid the uncertainty of COVID. Since I joined BAIR in late 2020, he has given me tremendous support. Two lessons from him have especially shaped me. The first is to be open to building connections and creating opportunities. He has always been deeply supportive of external collaborations, and many of the most important projects and experiences in my PhD came from following this philosophy. The second is to be critical about ideas while still giving them the benefit of the doubt through execution. This balance between skepticism and optimism has influenced how I approach research, collaborations, and ambitious projects.

Many of the opportunities that shaped my PhD grew out of a fortunate chain of collaborations. My collaborations with Siemens, Autodesk, Google, and later Meta formed a chain of opportunities that eventually brought me to NVIDIA. In particular, presenting the touch-vision-language alignment paper at ICML 2024 led me to meet Professor Yuke Zhu and Jim Fan, whose mentorship and collaboration have deeply shaped my thinking over the past year and a half. I am grateful to Jim and Yuke for encouraging me to think more boldly about research ideas, to ask whether I would do things differently if more resources were available, and to think seriously about whether a method is truly scalable. My experience at NVIDIA has had a major influence on my thinking about robot learning, scaling, and embodied intelligence.

I have also been incredibly fortunate to work closely with Raven Huang and Justin Yu, my core collaborators and co-first authors on more papers than I can easily count at this point. The resilience we developed over the past few years is one of the things I am most proud of. There were many moments, across many papers, when it would have been easy to give up, but we kept pushing and made them happen. I learned so much from both of you. The craziness we went through, both in terms of ideas and execution, is something I will never forget. Some of my favorite memories are the absurd schedules we shared: submitting papers at 4 or 5 a.m., then going on a 5 a.m. ski day trip to Tahoe the day after to decompress. Those combinations somehow became some of the best parts of my winters.

I am also grateful to Jiayi Pan and Tony Lian, with whom I shared many late-night discussions that turned into experiments, projects, and new directions. The past few years have been transformative, and many good things came from those conversations. I also want to thank them for helping me stay at the frontier of language models and vision models.

I owe special thanks to Mike Danielczuk, Ashwin Balakrishna, Jeff Ichnowski, and Daniel S. Brown for mentoring me during my first undergraduate research project. They taught me how to approach research methodically, how to present results clearly, and how to develop the research habits that became the foundation for my later work.

I am grateful to the many friends and collaborators in AUTOLAB who have enlightened me and helped make Berkeley such a rich and supportive research environment: Lawrence Chen, Chung Min Kim, Justin Kerr, Kush Hari, Simeon Adebola, Lars Berscheid, Karim El-Refai, Gaurav Datta, Will Panitch, Adam Rashid, Jaimyn Drake, Andrew Goldberg, Ethan Qiu, Ethan Kou, Zachary Tam, Kavish Kondap, Max Cao, Alejandro Escontrela, Albert Wilcox, Anrui Gu, Ryan Hoque, Satvik Sharma, Kaushik Shivakumar, and Vincent Lim.

I have also been fortunate to collaborate with and learn from many people across Berkeley and beyond, including Fangchen Liu, Xudong Wang, Haoru Xue, Baifeng Shi, Renhao Wang, Zirui Wang, Jiaxin Ge, Junyi Zhang, David Chan, Wenli Xiao, and Ilija Radosavovic. Some of my earliest experiences connecting learning methods to robot control can be traced back to these interactions. I am grateful for the many conversations, ideas, and projects that came from these interactions.

My time at NVIDIA has been an important part of this dissertation journey. I am thankful for the opportunity to work with and learn from Dantong Niu, Ruijie Zheng, Fengyuan Hu, Jing Wang, Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Avnish Narayan, Ryan Julian, K.R. Zentner, Tairan He, Jimmy Wu, Kaiyuan Zheng, Zhengyi Luo, and many others. I am grateful for all the infra battles we went through, the research environment, the technical depth, and the many discussions that helped sharpen my thinking about scalable robot learning.

I am also thankful to my collaborators from Autodesk, Meta, Google, TRI, and other institutions. In particular, I thank Hui Li and Sachin Chitta from Autodesk; Joseph Ortiz, Mike Lambeta, and Mustafa Mukadam from Meta; Brian and Anelia from Google; and Rares Ambrus, Richard Cheng, and Muhammad Zubair Irshad from TRI. I am also grateful to many faculty collaborators and mentors, including Kuan Fang, Sewon Min, Ion Stoica, Joseph E. Gonzalez, Shankar Sastry, Fei-Fei Li, Jiajun Wu, Guanya Shi, Roberto Calandra, Adam Yala, Angjoo Kanazawa, Alyosha Efros, Jitendra Malik, Trevor Darrell, and Pieter Abbeel. Their ideas, feedback, and examples have shaped my research trajectory.

Teaching has also been a meaningful part of the last stretch of my graduate studies. I thank Giuseppe Loianno and Shankar Sastry for teaching EECS 106B and for giving me the opportunity to help teach Berkeley's second course on robotics. I am also grateful to the wonderful teaching team, including Jaeyun Stella Seo, Daniel Municio, and Karim El-Refai. Teaching 106B was a fun and valuable experience, and it gave me the chance to revisit and strengthen my understanding of fundamental robotics.

Beyond research and teaching, I am thankful for the broader BAIR community and for the friendships that made these years joyful. I thank Haven Feng, Qianqian Wang, Brent Yi, Michael Psenka, Paul Zhou, Homer Walke, and many others for the conversations, support, and memories. I will always remember climbing at Pacific Pipe and Ironworks, and skiing at Heavenly, Kirkwood, and Palisades with all these friends.

I also want to thank the joy brought by my girlfriend, Wendy, whom I have known since the beginning of my undergraduate years, and by our cat, Mochi, who has been my companion since 2020. Over the past eight years, Wendy has provided constant support and companionship throughout the time. I am truly grateful for her patience and encouragement.

Finally, I owe my deepest gratitude to my parents. Their love and support have been a constant source of motivation throughout my life. I am grateful for all the debates, for the way they always encouraged me to see both sides of any choice, and for the trust they placed in me as I found my own path. Their encouragement has been the driving force behind all I have achieved.

Chapter 1

Introduction

Over the past five years, vision and language models have improved rapidly, with each successive generation pushing accuracy on benchmarks toward the level of human experts. Many of the architectures, optimization recipes, and pre-training data sources behind that progress have been adopted by recent robot learning systems. However, we have yet to see a robot that can reliably handle the long tail of household manipulation, or be cheaply repurposed to a new factory line and operate for days without intervention.

This gap is not merely a matter of model capacity. The deeper challenges lie in how robot models are designed and trained, what sensorimotor data they are trained on, and how the robot can compose the policies, primitives, and tools we already have. This dissertation argues that closing this gap requires progress along three directions: pre-training models on robot trajectories in the same spirit that language models are pre-trained on text; scaling robot data through pipelines that do not require a human behind every teleoperation interface; and treating capable language models as agents that can orchestrate perception and control for long-horizon, zero-shot manipulation.

This introduction proceeds in three parts. The first two trace the paradigms that produced the contemporary state of vision and language (pre-training and post-training, and the recent shift towards test-time scaling and agentic systems) and ask, for each, how the analogue would look in robotics. The third section introduces the robot data bottleneck that distinguishes robot learning from vision and language, and uses it to motivate the three research threads that constitute this dissertation.

1.1 The Pre-training and Post-training Paradigm

The first paradigm, and the one that has had the largest measured effect, is *pre-training*: training a single model on a large, diverse corpus of data, with the expectation that the resulting representations will transfer to many downstream tasks. Transformer [2], paired with masked or autoregressive prediction objectives [3, 4], gave language a unified backbone that scales predictably in model size, data, and compute [5–12], with each successive genera-

tion pushing accuracy on benchmarks towards human expert level [13–28] and exhibiting capabilities that emerge at scale [29, 30]. Vision followed the same arc, with self-supervised contrastive, generative, and self-distillation objectives [31–34], masked autoencoders [35], and language-aligned visual representations [36–40] all converging on the same recipe: one model, trained once on diverse data, that serves as the starting point for post-training.

Recent robot learning development closely follows the trends in vision and language. The first wave focused on pre-training for better visual representations: encoders trained on egocentric human video [41–43] produced features that gave downstream policies a strong starting point, but the policy networks were still fit from scratch for each task. The second wave broadened to language-conditioned policies that consume vision and a natural-language instruction and predict actions, trained on cross-embodied teleoperation data [44–53], often paired with flexible multi-modal action heads such as diffusion policies and action-chunking transformers [54, 55].

What unifies almost all of these systems is a single-step view of the policy. At each time step, the model maps the current observation, at most a short stack of recent frames, to the next action or short action chunk. The pre-training objective, where there is one, is essentially "predict the next action given the current observation," not "model the joint distribution over the trajectory." This is a faithful translation of behavior cloning, but it imports an implicit Markov assumption that does not hold for many manipulation tasks. A purely reactive policy cannot represent task progression: two distinct moments in a long-horizon manipulation can have nearly identical visual observations and yet require very different actions, and any pre-training objective that conditions only on the current frame will treat the two as the same example. The objective is therefore mismatched to the structure of the problem before any data is collected.

This dissertation pursues a different design on the pre-training side. Rather than predicting the next action from the current observation, we treat a robot trajectory of vision, proprioception, and action as a single multimodal sequence and ask what BERT's masking and GPT's next-token-prediction objectives can do for sensorimotor data. Chapter 3 develops sensorimotor pre-training along these lines and shows that a fine-tuned generalist outperforms a from-scratch specialist [56]; Chapter 4 pushes the same view further to in-context imitation, where a robot policy adapts to a new task at inference from a handful of demonstrations, with no weight update; and Chapter 5 shows that text-aware visual features extracted from a frozen CLIP-style backbone compose cleanly with this trajectory backbone, preserving open-vocabulary alignment that a fine-tuned VLA tends to wash out.

Pre-training produces a useful starting point, not a useful product. Closing the gap is the work of *post-training*, a separate stage that specializes the pre-trained model for a target behavior. Three algorithm families now dominate, each defined by its supervisory signal. *Supervised fine-tuning* (SFT) adapts the model to instruction–response pairs written by humans [57]. *Reinforcement learning from human preferences* (RLHF) layers a learned reward model on top of SFT and optimizes it against pairwise preference labels with policy-gradient methods [57–59]. *Reinforcement learning over verifiable rewards* (RLVR) trades human labels for programmatically checkable outcomes from a sandbox or grader, and has driven the recent

generation of mathematical and coding reasoners [20, 21, 60–63]. The three differ in cost and verifiability: cheap labeled examples, expensive but flexible human preferences, and scarce but auditable verifiable outcomes.

All three carry over to robotics in principle, but the data each requires becomes harder to scale on a robot. SFT in robotics is fine-tuning a pre-trained generalist on teleoperated demonstrations [48, 64]; RLHF on robot trajectories has been studied but rarely deployed at scale [65, 66]; and RLVR, in the form of policy-gradient training against engineered simulator rewards, is the workhorse behind the recent generation of legged-locomotion controllers that transfer zero-shot to physical quadrupeds and humanoids [67–70]. Extending the same recipe to manipulation is constrained instead by the difficulty of simulating contact-rich, deformable, and tool-using interactions at the fidelity required [71, 72], a point we return to in Section 1.3. Part III of this dissertation pursues the RLVR direction directly, by exposing simulator task rewards to a coding agent and fine-tuning the agent on those rewards (Chapter 10).

1.2 Test-time Compute and Agentic Systems

A third paradigm has emerged in vision and language, and it matters specifically for robotics because it offers an alternative to scaling training data. The premise is simple: when one cannot easily acquire more training data, one can spend more compute at *inference time*, and that compute can be a near-substitute for additional training. The paradigm has deep roots: in 1997, Deep Blue [73] beat the reigning world chess champion by deploying a then-extreme amount of search at every move, with no learned policy. It has matured rapidly in the language-model era.

This idea also is exemplified via a combination of reinforcement learning and search. AlphaGo [23] combined deep neural-network policies with Monte-Carlo tree search and was the first program to defeat a top professional Go player. The lesson was that a competent learned policy plus a structured inference-time procedure outperforms the policy alone, and that this remains true even when the supervisory signal is just a verifiable game outcome, with no human demonstrations in sight [74, 75].

Language models now exploit the same lesson in two complementary forms. The first is more compute per single response: chain-of-thought, self-consistency, tree-of-thoughts, and repeated sampling all extend the time a model spends on a problem before committing to an answer [76–79], with process and outcome verifiers [20, 21] selecting good candidates from many without human supervision; these procedures fold back into training as RLVR (Section 1.1). The second is structured tool use: agentic systems interleave reasoning with tool calls and self-correction [80–83], and code-executing agents solve software-engineering problems by writing, running, and iterating on programs [84–88]. From a robotics standpoint, the striking feature is that none of these systems require additional training data: improvement over a single forward pass comes from *spending more compute, more carefully, at inference time.*

The robotics analogue of this third paradigm is now visible. Code as Policies [89] and ProgPrompt [90] cast the language model as a programmer that emits robot programs; SayCan [91] and Inner Monologue [92] closed the loop between LLM planning and learned skills; VoxPoser [93] composes 3D value maps from natural language; and MAESTRO [94] and Gemini Robotics [95] integrate LLM-driven planning with sensorimotor policies in deployable systems. The promise of this thread is precisely the promise that motivated the introduction: when robot data is the bottleneck, a thoughtful inference-time procedure that calls perception, control, and verification primitives can extract more capability from a fixed amount of training than any single end-to-end policy can. Part III of this dissertation pursues this line directly.

1.3 The Robot Data Bottleneck

Pre-training, post-training, and test-time scaling all presume the existence of large, diverse training corpora. Robotics does not yet supply them. The largest open robot datasets [48, 64, 96] are over $10^5\times$ smaller than the text and image corpora behind frontier language and vision–language models [97], and even the largest closed industrial datasets remain far short of the regime that pre-training requires. The gap is not only quantitative. Teleoperated demonstrations cover only the slice of state space operators choose to visit, the long tail of objects, materials, and failures is sparsely sampled, and the data rarely transfers cleanly across embodiments, gripper geometries, or sensor configurations. Every additional hour of robot data requires a physical robot, an operator, an environment, and a real-time interface.

Other areas of vision and language have faced analogous data scarcity and addressed it by replacing direct human annotation with synthetic supervision. In language, Self-Instruct [98], Alpaca [99], and Vicuna [100] bootstrap instruction-following and question-answering data from larger closed-source models. In vision-language reasoning, SpatialVLM [101] synthesizes large-scale spatial question-answer pairs using off-the-shelf detection and segmentation models. In dense 3D prediction, methods such as RAFT [102], DUSt3R [103], MonST3R [104], Zero-1-to-3 [105], and MVGD [106] rely on pseudo-labels derived from multi-view geometry [107]. The corresponding challenge in robotics is more stringent: robot data is not only expensive to label, but expensive to produce. Embodiment-specific demonstrations scale with physical robots, operator time, and real-world execution, whereas embodiment-*agnostic* data, if constructed carefully, can scale with reusable artifacts such as object scans, scripted controllers, simulators, and human videos. Part II of this dissertation studies this opportunity directly. Chapter 6 surveys the available axes for scaling robot data, and Chapters 7 and 8 present two concrete pipelines that generate robot learning data without requiring a human behind a teleoperation interface.

1.4 Dissertation Organization

This dissertation argues that closing the gap between contemporary vision–language models and efficient and scalable robot learning requires research along three different axes, each adapted to the constraints of the robot setting and each motivated by one of the three paradigms reviewed above.

The first axis treats robot trajectories of vision, proprioception, and action as multimodal sequences and pre-trains them with masking and next-token-prediction objectives, the embodied analogue of the pre-training paradigm. The second axis removes the human from the data-collection loop, exchanging real-time teleoperation first for autonomous on-hardware data collection and then for hardware-free synthetic trajectory generation. The third axis treats capable language models as planners that orchestrate perception and control primitives, an inference-time and agentic substitute for additional training data. Together, these threads pursue robot learning that is *efficient* (acquiring a new skill from as few demonstrations as possible) and *scalable* (improving with additional data, compute, or both).

Part I: Trajectory Pre-training for Robot Learning. Chapter 2 motivates trajectory-level pre-training and in-context imitation. Chapter 3 introduces sensorimotor pre-training of a transformer policy on real robot trajectories, with masked prediction as the pre-training objective; the resulting representation transfers across tasks and embodiments. Chapter 4 extends the next-token-prediction view to in-context imitation, identifying the data composition and prompting choices that make a robot transformer adapt to a new task at inference from a single demonstration. Chapter 5 presents OTTER, a vision–language–action model with text-aware visual feature extraction that preserves CLIP's open-vocabulary generalization while still producing action sequences over a multi-step history.

Part II: Scaling Robot Data Without Teleoperation. Chapter 6 surveys the robot data gap and the four available scaling axes (scripted policy, teleoperation, world models, simulation). Chapter 7 develops a safe self-supervised data-collection pipeline that uses tactile and force–torque feedback to identify reversible grasps, enabling autonomous data collection on a real robot for industrial insertion. Chapter 8 describes Real2Render2Real, a pipeline that takes a single human demonstration plus an object scan, renders thousands of randomized synthetic trajectories without any robot hardware, and fine-tunes a vision–language–action model on the synthetic data alone, matching teleop-trained baselines at a fraction of the time and operator effort.

Part III: Code as Policies for Manipulation. Chapter 9 motivates coding agents as a complement to learned policies. Chapter 10 introduces CaP-X, a framework with four components: *CaP-Gym*, 187 simulated manipulation environments with shared low-level and high-level perception/control primitives; *CaP-Bench*, an evaluation suite that varies API abstraction and feedback modality; *CaP-Agent0*, a training-free agent harness that combines visual differencing, a skill library, and parallel query; and *CaP-RL*, an RLVR fine-tuning loop that uses CaP-Gym task rewards to improve the coding model itself, with primitives shared between simulation and real so that the resulting agent transfers directly to physical hardware.

Chapter 11 concludes with reflections from exploring the three axes, and on the open directions they suggest.

Part I

Trajectory Pre-training for Robot Learning

Chapter 2

From Sequences to Sensorimotor Pre-training

2.1 Why Pre-train for Robotics?

Pre-training has driven much of the recent progress in vision and language: a single model is trained on large, weakly supervised data and then specialized to downstream tasks through fine-tuning, prompting, or in-context examples [3, 4, 35]. Robotics, when this work began in 2022, did not yet have a comparable recipe. Most policies were trained one task at a time on small demonstration sets, with little reuse across tasks or embodiments. Earlier attempts sat on either side of the question: visual-only pre-training on internet images and egocentric video [41–43, 108] produced strong perceptual features but did not by itself produce action, and large multi-task transformer policies [45, 109, 110] trained on heterogeneous demonstration data but did not pre-train in the sense that vision and language models do. The key question Part I asks is the one that fell between these two threads: *can sensorimotor data itself be the corpus we pre-train on?* and *what are the ways to pre-train on it that would result in sample efficient learning for downstream tasks?*

Two things stood in the way. The dominant imitation-learning recipe at the time predicted a single action from the current observation or a very short stack of recent frames, as in standard behavior cloning, implicit behavior cloning [111], and early language-conditioned policies [45]. This misaligns with the long-context sequence modeling objectives that drove the success of BERT and GPT [3, 4, 35, 112]. Secondly, the large, language annotated robot datasets that today's robot foundation models are trained on [48, 64] did not yet exist. Part I therefore takes a different path: treat a robot trajectory of vision, proprioception, and action as a single sensorimotor sequence, and ask what the same masked and autoregressive objectives can do for it.

2.2 Modeling Trajectories

Treating a trajectory as a sequence makes the same family of self-supervised objectives that drove progress in language and vision applicable to sensorimotor data: masked prediction [3, 35] for representation learning, and autoregressive next-token prediction [4] for generation and adaptation. The argument for why a trajectory model is a more faithful description of manipulation than a Markov policy is made in Section 1.1; the chapters that follow take that design choice as given and develop it through three concrete questions.

2.3 Research Questions and Roadmap for Part I

The first question is whether trajectory-level pre-training actually pays off. *Does pre-training on robot trajectories produce better downstream control than training a specialist from scratch?* The results in RPT (Chapter 3) suggest that if a transformer is trained via the masked prediction objective over interleaved (image, proprioception, action) tokens, the resulting model can efficiently transfer across tasks and embodiments, and improves with longer context, larger models, and more pre-training data [56].

The second is whether a sequence model can adapt to new robotic manipulation tasks without a weight update. *Can a robot infer a new task purely from a prompt trajectory at inference time?* Instead of the mask prediction objective in RPT, ICRT (Chapter 4) switches to next-token prediction and shows that a long-context sensorimotor sequence model can perform an unseen task in a new scene given only one or a few demonstrations as a prompt, with no gradient update [113]. Earlier few-shot imitation work studied explicit demonstration encoders, cross-attention, or metric aggregation [114–116]; Prompting Decision Transformer first applied prompt conditioning to sequential decision making [117]; and recent in-context robot work studies adaptation from demonstrations or videos [118, 119]. ICRT shows that the autoregressive prediction recipe is enough *if* the provided training data contains scenes where multiple tasks are plausible from the same observation. Otherwise, the model ignores the prompt and relies on shortcut cues in the current image.

The third question concerns the language instruction following capability in robot foundation models. *How should vision–language alignment enter a sensorimotor sequence model without being washed out by fine-tuning on robot data?* Recent VLA systems [46, 51, 52] pass visual and language features into the policy and rely on robot fine-tuning to align them for control. That fine-tuning, on robot data far less semantically diverse than the image–text corpora behind CLIP and SigLIP [36, 37], can degrade the open-vocabulary grounding the backbone provides [120, 121]. OTTER (Chapter 5) takes a different design: extract text-aware visual features from a frozen CLIP-style backbone and feed them into a trajectory model, preserving open-vocabulary grounding while still producing actions over a multi-step history [122].

Together, these chapters argue for the use of longer-context for robot learning. They treat the trajectory as sensorimotor "sentences", not the isolated state–action pair, as the main

object of modeling. Additionally, they show that pre-training, in-context prompting, and language grounding can all be integrated under the trajectory modeling framework.

Chapter 3

Robot Learning with Sensorimotor Pre-training

3.1 Introduction

Over the last couple of years, inspired by vision [33, 35, 123, 124] and language [3, 125, 126], there has been an increased interest in pre-training for robotics. For example, we have seen promising results from self-supervised visual pre-training on large and diverse image collections [108]. However, robotic data contains rich sensory and motor information that is difficult to capture with visual pre-training alone. We ask: *can we learn good sensorimotor representations from robotic trajectories?*

In this chapter, we propose a self-supervised sensorimotor pre-training approach for robotics. We formulate robotic pre-training as a general sensorimotor sequence prediction problem. We instantiate this idea through a masked prediction task, similar to the counterparts in natural language processing and computer vision [3, 33, 35]. We hypothesize that if a robot can predict missing sensorimotor content it will have acquired a good model of the physical world that can enable it to act.

Our model, called RPT, is a Transformer [2] that operates on sequences of sensorimotor tokens. Given an input sequence of camera images, proprioceptive robot states, and actions, we encode the interleaved sequence into tokens, mask out a subset of the sequence, and predict the masked-out content from the rest. We perform masking across all modalities and time using a high masking ratio, which encourages the model to learn cross-modal, spatio-temporal representations.

We encode camera images using a pre-trained vision encoder [108] and use latent visual representations for sensorimotor sequence learning. This enables us to build on strong visual representations, trained on large and diverse image collections from the Internet. Compared to prediction in pixel space, performing prediction in the latent representation space makes the task more tractable. This design decouples the computational vision cost from the sensorimotor context length, making 10 Hz control with over 300M parameter models and

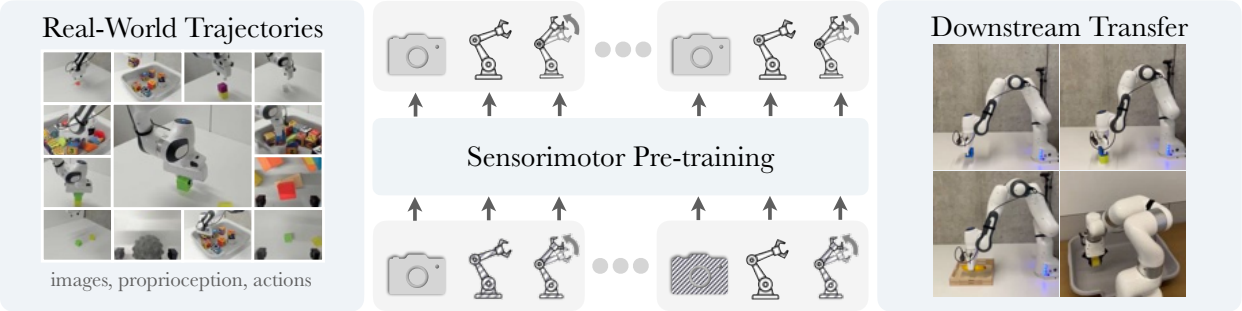


Figure 3.1: Robot learning with sensorimotor pre-training. *Left:* We collect a large dataset of real-world trajectories that contain multiview RGB images, proprioceptive robot states, and actions to use for sensorimotor pre-training. *Middle:* Given a sensorimotor trajectory, we mask out a subset (shown by striped pattern) and train a Transformer model to predict the masked-out content from the rest. *Right:* We transfer the pre-trained representations to different downstream tasks and robots.

large context lengths feasible on a physical robot.

To study our pre-training approach, we collected a dataset of over 20,000 real-world trajectories over 9 months using a combination of motion planning and model-based grasping algorithms. Each trajectory is a sequence of multi-view RGB camera images, proprioceptive robot states, and actions. We collected trajectories for classic robotic tasks, namely, single object picking, bin picking, stacking, and destacking. All of the tasks include variations in object pose, shape, and appearance.

To understand the effect of pre-training, we perform a series of real-world experiments. We find that RPT consistently outperforms training from scratch and that the improvements are larger for harder tasks (up to 2× for the block stacking task). We also find that our sensorimotor pre-training approach enables successful transfer across different tasks, lab environments, and robots. Moreover, our approach has favorable scaling properties and benefits from better vision encoders, longer sensorimotor context lengths, and larger pre-training datasets. Finally, we find that masking across both modalities and time, with a high masking ratio, is important for good performance.

3.2 Robot Learning with Sensorimotor Pre-training

Our approach consists of a pre-training and a fine-tuning stage (Figure 3.1). We pre-train sensorimotor representations with masked prediction on sequences of camera images, proprioceptive states, and actions (Figure 3.2). After pre-training, we transfer representations to downstream tasks (Figure 3.3).

Sensorimotor Pre-training

We begin by describing the pre-training stage of our approach. In the pre-training stage, we are given a dataset $\mathcal{D}$ of sensorimotor trajectories $\mathcal{T}$. Each sensorimotor trajectory is a sequence of camera images, proprioceptive robot states, and actions: $\mathcal{T} = (i_1, s_1, a_1, ..., i_T, s_T, a_T)$. We assume no access to additional semantic information, like language instructions or task labels. We hypothesize that the unlabeled sensorimotor trajectories implicitly encode the structure of the physical world and that we can use them to learn general sensorimotor representations for downstream robotic tasks.

We formulate robotic pre-training as a general sensorimotor sequence prediction problem, across all modalities and time. We instantiate this idea through a masked prediction task, similar to the counterparts in vision and language [3, 33, 35]. We mask out a subset of a trajectory and train a model to predict the missing content. Specifically, given a sensorimotor sequence of L tokens, we sample a mask sub-sequence $M \subset [1, L]$ and train a model to minimize the mean squared error of the masked tokens $\mathcal{T}_M$ conditioned on the observed tokens $\mathcal{T}_{[1,L]\setminus M}$. The intuition is that if a robot can infill missing sensorimotor content it will have acquired a good model of the physical world that can enable it to act. This general formulation enables us to represent many different contextual prediction problems by simply using different masking patterns. We consider a number of variants, including random masking at the modality, timestep, and token level, as well as causal masking.

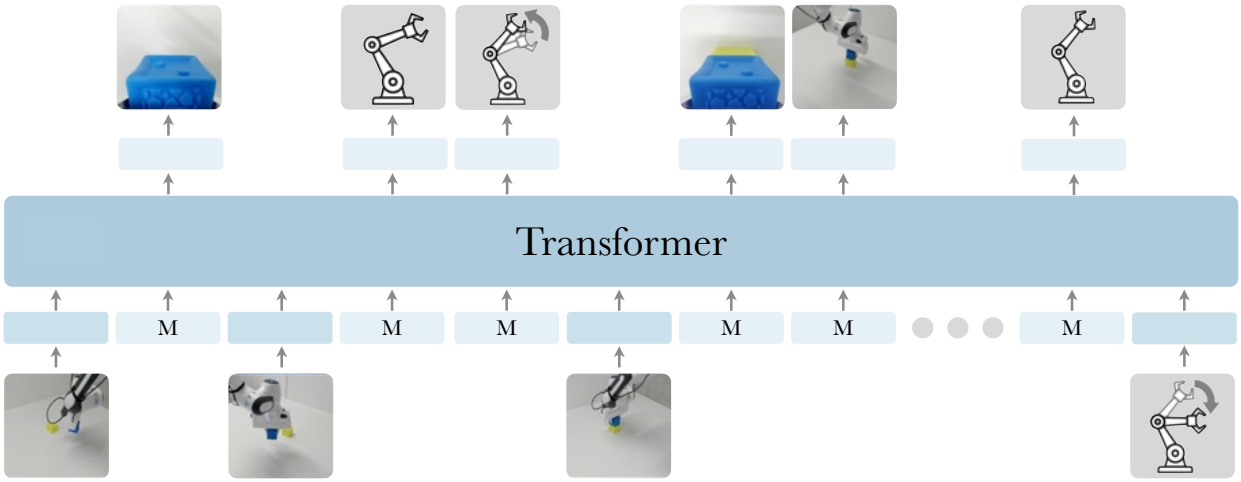


Figure 3.2: Sensorimotor pre-training. Our model is a Transformer that operates on interleaved sequence of camera images, proprioceptive robot states, and past robot actions. We encode sensorimotor inputs into tokens, mask out a subset, and train a model to predict the missing content.

Architecture

Token masking. Given a sequence of sensorimotor tokens we mask out a subset. We represent each masked out input using a mask token that is learnt per modality and shared across time. We mask the input tokens independently without taking the time step or modality into consideration which encourages the model to learn to correlate information across both modalities and time.

Vision latents. Our sensorimotor model must process sequences of high-dimensional images from multiple views, which is challenging from both the learning and the computational perspective. To overcome this, we use a pre-trained vision encoder to compute visual representations [108] and operate in the latent space. This enables us to build on strong visual representations, trained on large and diverse image collections from the Internet. Compared to prediction in pixel space, prediction in the latent space makes the task more tractable. This design also decouples the computational vision cost from the sensorimotor context length, making fast inference with large models feasible.

Token encoders. We use a separate linear encoder per modality. Since our modality encoders do not share weights, we do not use additional modality embeddings. To represent time, we add positional embeddings to each token. All tokens from a single timestep share the positional embedding value.

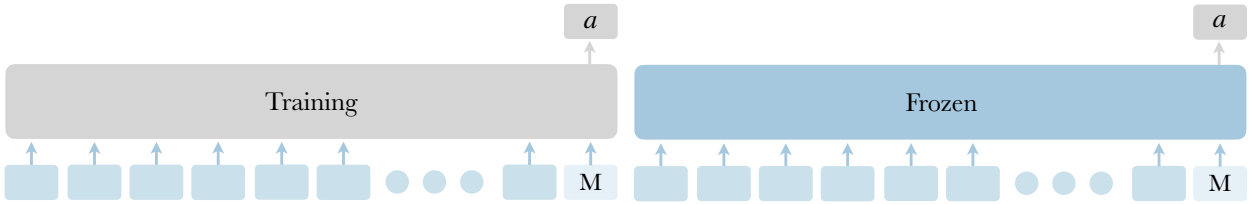
Transformer model. The encoded and masked sequence of tokens is passed into a Transformer model [2]. We follow the standard Transformer design consisting of a series of Transformer blocks with multi-head self-attention operations. Our model predicts latent representations for each input.

Prediction heads. We decode the hidden representations into prediction targets using linear project layers. Each latent is decoded from the hidden size back to the original modality dimension. We make predictions in the original input space for joints and the visual latent space for images.

Masked objective. We compute the mean squared error reconstruction loss between the predictions and the ground truth input values. We apply the loss only to the predictions from the latent representations corresponding to the masked inputs. Predictions for the observed inputs incur no loss. To weight the importances and scale of different modalities we apply a per-modality loss weight.

Downstream Transfer

Our goal is to learn general sensorimotor representations that can be transferred to different downstream tasks and robots. Inspired by vision and language [3, 33, 35], we explore two settings: *(1) Fine-tuning.* We fine-tune a pre-trained model checkpoint on downstream task data. In this setting pre-training can be seen as providing a good initialization for learning. *(2) Linear probe.* We use a frozen pre-trained model to extract sensorimotor features and train a single linear layer to predict actions. This enables us to evaluate the quality of the pre-trained representations alone.



(a) Finetune (b) Linear Probe

Figure 3.3: Downstream transfer. We consider two different settings for evaluating representations on downstream tasks: (a) *Fine-tuning:* We fine-tune the entire pre-trained model on the downstream task data; (b) *Linear Probe:* We freeze the pre-trained model and train a linear action read out layer.

3.3 Experimental Setup

Robot and tasks. We use a 7-DoF Franka robot with the default 1-DoF parallel jaw gripper. We perform joint position control at 10 Hz. We include joint positions and the gripper state in the proprioceptive information. We use three RGB cameras, one attached to the robot hand and two on the sides (Figure A.1). We consider four different tasks: `Pick`, `Bin Pick`, `Destack`, and `Stack`. To study the proposed approach, we collected a dataset of sensorimotor trajectories using a combination of motion planning and model-based grasping algorithms (see Appendix A.1 for more details).

Vision encoder. We use the Vision Transformer (ViT) architecture [127] to encode image inputs. Specifically, we use the pre-trained models from [108] which were trained via MAE [35] on a collection of 4.5M images from Ego4D [128], Epic [129], Something-Something [130], 100 Days of Hands [131], and ImageNet [132] images. We extract features from all three cameras using the same model. We use the mean pooling of output tokens from the vision encoder as the vision feature.

Sensorimotor transformer. Our sensorimotor model is an encoder-only Transformer with ~1M parameters. The transformer has a hidden dimension of 192 and four transformer blocks, each with 4 heads and an MLP ratio of 2. The sensorimotor input contains multiple steps, each step including the visual features from 3 camera views, the proprioceptive states and actions. Since the inputs from different modalities have different dimensions (*e.g.*, 768 for images and 8 for proprioception and actions), we project each modality to the same dimension of 192 using a linear layer per modality.

Pre-training. During sensorimotor pre-training, we first randomly sample a masking probability from a fixed range of masking ratios, and then independently mask each input token (visual, proprioception, or action) with the same probability. The default masking ratio range is $[0.7, 0.9]$, which we find to work the best empirically. We also study other masking strategies such as masking all the tokens from a modality or a timestep. Please see ablations on masking strategy and masking ratio in Section 3.4. The masked reconstruction loss for each modality is weighted together using a uniform weight. All models are pre-trained for

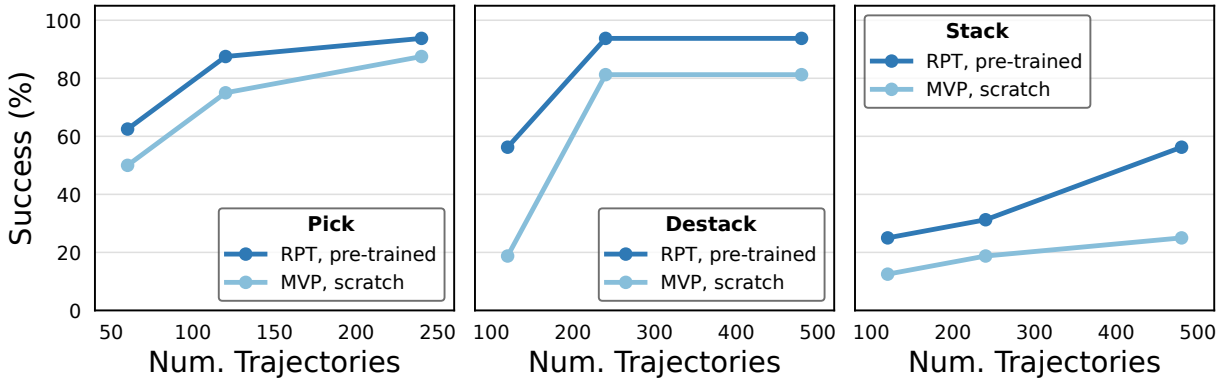


Figure 3.4: Sample complexity, fine-tuning. We study the effect of *sensorimotor pre-training* as the amount of fine-tuning data increases. We find that sensorimotor pre-training (RPT) brings consistent improvements over training the sensorimotor model from scratch (MVP) and that the gains are larger for the harder tasks (Stack). Note that the vision encoders are pre-trained and frozen for both [108].

300 epochs with a batch size of 4096 and 50 warm-up epochs. We use the AdamW optimizer with a learning rate of 4×10^{-4} and a weight decay of 0.01.

Fine-tuning. We initialize the sensorimotor model with the pre-trained weights and fine-tune it with behavior cloning on the downstream task. At fine-tuning time, the model takes in a sequence of the same length as in pre-training with the action at the last time step replaced with a mask token. To predict the action, we use the output token that corresponds to the mask input token and discard the other predictions. We train a linear layer to predict next 16 actions from the last mask token. We fine-tune the model for 900 epochs for the task of `Stack` and 300 epochs for other tasks. Other hyperparameters such as learning rate and batch size stay the same as in pre-training.

Inference. When testing the fine-tuned model, we feed the past sensorimotor trajectory as input and the model predicts the actions for the next 16 steps. The predicted actions are passed to the robot controller which executes the actions at 10 Hz. We experimented with executing one action at the time and re-predicting but did not observe a significant difference in performance. After each action is executed, the visual observations and the achieved state are recorded and, alongside the executed action, are fed back as the sensorimotor inputs for the next prediction in the autoregressive fashion.

3.4 Experimental Results

We perform evaluations in the real-world and study sample complexity on different tasks, transfer across tasks and robots, scaling properties, and different design decisions. For all

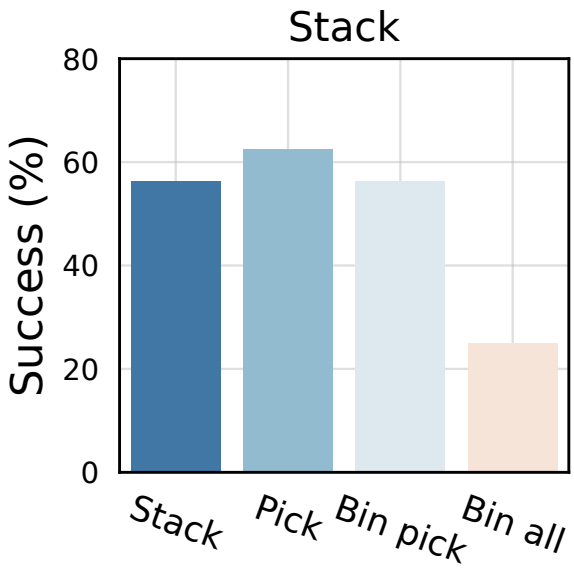


Figure 3.5: Transfer across tasks. We compare pre-training on different tasks and fine-tuning on stacking. We see that pre-training can lead to strong downstream performance even across tasks.

experiments, we report success rates across 16 real-world trials with variations in object positions and orientations.

Sample Complexity

We begin by studying the effect of sensorimotor pre-training by comparing its fine-tuning performance to training from scratch. As our scratch baseline we use an improved version of MVP [108], where the single-step MLP policy is replaced by a multi-step Transformer policy. For fair comparisons, we use the same pre-trained vision encoders, sensorimotor architectures, and optimize the learning rate, batch size, and the number of epochs per model. We consider three downstream tasks of increasing difficulty: picking, destacking, and stacking. We pre-train using a different subset of the data from the same task as in fine-tuning and hold out an unseen set for evaluation. In Figure 3.4 we show the performance as the number of fine-tuning demonstrations increases. We observe that pre-training leads to consistent improvements over training from scratch, across different tasks and data regimes. Moreover, the improvements are the largest for the hardest block stacking task.

Transfer Across Tasks

In the previous section we used the data from the same task for pre-training and fine-tuning. To evaluate if our pre-training approach can learn general sensorimotor representation instead

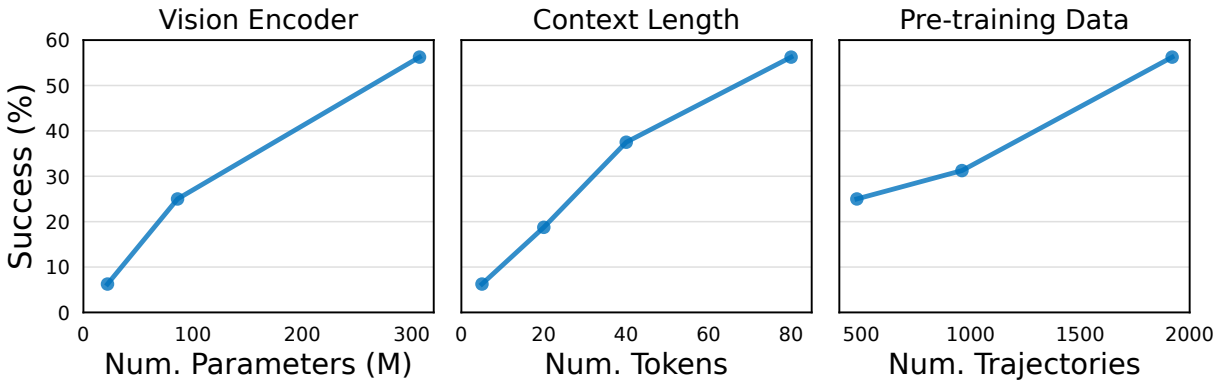


Figure 3.6: Scaling studies. We find that our approach benefits from better vision encoders (left), larger context lengths (middle), and more pre-training data (right). Evaluated on block stacking.

of representation for a specific task, we study pre-training and fine-tuning across different tasks. Specifically, we first pre-train a model on data from different tasks: stacking, picking, successful bin picking trajectories, and bin picking trajectories that include failure cases. We then fine-tune and evaluate the models on stacking. We report the results in Figure 3.5. We observe that pre-training on all of stacking, picking, or bin picking leads to similar downstream performance on stacking, which suggests that our sensorimotor pre-training approach can learn transferable representations across tasks. We also see lower performance when pre-training on all of bin picking data that includes failed trajectories, which highlights the importance of pre-training data quality.

Scaling Studies

Vision encoder. We study the performance of our pre-training approach as the size of the vision encoder increases. In all cases, we use a pre-trained and frozen vision encoder from [108]. Specifically, we consider three ViT variants of increasing size: ViT-S, ViT-B, and ViT-L. We evaluate performance on the block stacking task which requires precise spatial localization. The results are shown in Figure 3.6, left. We observe that the performance improves significantly with better vision models. We note that our model is still capable of 10 Hz inference even when using the ViT-L vision encoder.

Context length. We compare sensorimotor pre-training with varying context lengths. Namely, we consider context lengths of 1, 4, 8, and 16 timesteps. Note that each timestep contains 5 tokens. In Figure 3.6, middle, we observe that pre-training on larger contexts leads to consistent improvements, which may suggest that longer contexts may facilitate richer sensorimotor pre-training problems.

Pre-training data. We study scaling of our approach as the amount of pre-training

data increases. We consider pre-training on 480, 960, and 1920 trajectories and evaluate downstream performance on block stacking. In Figure 3.6, right, we observe that our approach benefits from more pre-training data which is a promising signal for scaling sensorimotor pre-training to larger trajectory collections.

Transfer Across Robots

Cross-lab transfer. We evaluate transfer across different robots. We first consider transfer to a different instance of the same robot type. The downstream robot is in a different lab that has differences in the environmental conditions, varying camera placement, background, and lighting. We compare pre-training to training from scratch. We report the results in Table 3.1 and observe that pre-training outperforms training from scratch considerably.

Cross-robot transfer. Next, we push this setting further and evaluate transfer across different robot types. Specifically, we pre-train on xArm data from [108] which includes 640 trajectories collected via teleoperation across 8 different tasks. Note that the gap here is quite large, with differences in the robot type, camera type and placement, tasks, background, lighting, data frequency, and collection strategy. We compare pre-training on xArm to (a) no pre-training and (b) pre-training on the same amount of original Franka data. In Table 3.1, we see that pre-training on xArm outperforms training from scratch considerably and comes very close to pre-training on the same robot.

Robustness to background change. We further test transfer to a Franka with a substantially altered visual environment by changing the workbench background. Despite the major appearance shift, sensorimotor pre-training continues to improve over training from scratch, raising the success rate from 50.0% to 68.8%. The improvement persists even when the from-scratch baseline is already a strong performer, suggesting that the pre-trained representations capture structure beyond surface visual cues.

Transfer to human demonstrations. We also evaluate transfer from clean, scripted pre-training trajectories to noisy human demonstrations collected via VR teleoperation. We collect 80 demonstrations of picking fruits (8 different fruits with 10 trajectories each) — a longer-horizon and more visually diverse task than the cube picking used in pre-training. Pre-training improves the success rate from 12.5% (scratch) to 50.0%, a larger relative gap than for the simpler cube task. This indicates that sensorimotor pre-training is particularly valuable when the downstream data is small, suboptimal, or distributionally shifted from pre-training.

Ablation Studies

Linear probe. We evaluate pre-trained sensorimotor representations via linear probing. Namely, we extract representations using a frozen sensorimotor model and train a linear layer on top using 120 trajectories for picking. We report the results in Figure 3.7, left. We observe that linear probing reaches a success rate of 43.75% which is non-trivial but considerably lower than 93.8% with fine-tuning.

pre-train	fine-tune	success (%)
	Franka B	25.0
Franka A	Franka B	68.8

(a) Franka → Franka. Effective transfer from a Franka in one lab to another Franka in a different lab, with differences in camera positions, background, and lighting.

pre-train	fine-tune	success (%)
	Franka	25.0
xArm	Franka	50.0
Franka	Franka	56.3

(b) xArm → Franka. Sensorimotor pre-training can be effective even across different robot types and come very close to pre-training on the same robot.

Table 3.1: Transfer across robots. We pre-train on data from one robot and evaluate downstream transfer to a different robot. We experiment with both cross-lab and cross-robot transfer.

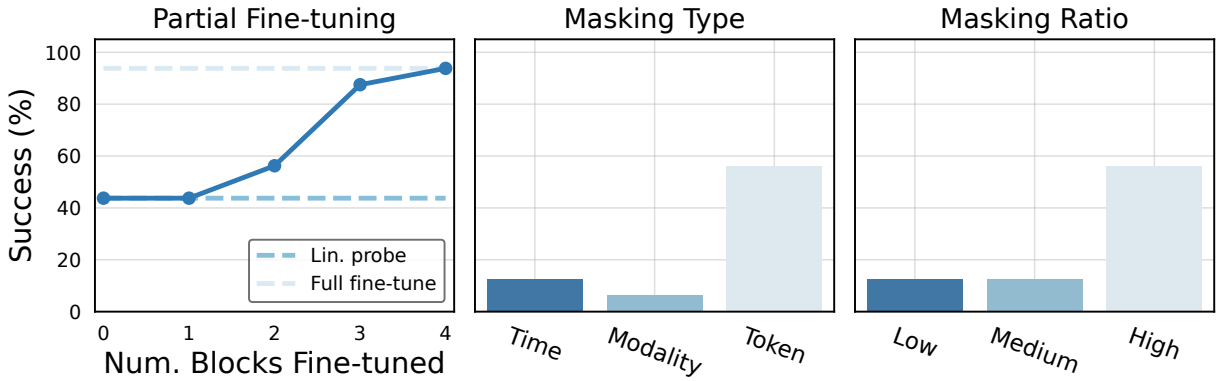


Figure 3.7: Ablation studies. We find that while linear probing achieves non-trivial performance, fine-tuning leads to considerably better results and that fine-tuning a larger portion of the model leads to higher success (left). We observe that masking across all tokens works considerably better than masking all tokens from a timestep or all tokens from a modality at a time (middle). Moreover, using a high masking ratio is essential for learning good sensorimotor representations (right).

Partial fine-tuning. In fine-tuning evaluations we fine-tune the full model by default. Here, we study the impact of fine-tuning an increasing number of Transformer blocks. We report the results in Figure 3.7, left. Note that fine-tuning zero blocks and four blocks is equivalent to linear probing and full fine-tuning, respectively. We observe that the performance increases with the number of blocks fine-tuned and that achieving best performance requires fine-tuning a majority of the blocks.

Masking type. We experiment with different masking strategies (see also Section 3.4). Specifically, we consider time-step masking which masks all tokens from a time step, modality masking which masks all of the tokens from a modality, and token masking which masks across all tokens independently. In Figure 3.7, middle, we see that token masking outperforms the alternatives considerably.

Masking ratio. We ablate different values of the masking ratio. As shown in Figure 3.7, right, there is a significant performance drop when masking ratio is low $[0.1, 0.9]$ or medium $[0.4, 0.9]$ compared to a high masking ratio $[0.7, 0.9]$. This suggests a high redundancy between different time steps and modalities in the input sensorimotor sequences and a high masking ratio is crucial for learning useful representations for downstream tasks. This is consistent with prior work on visual pre-training [35].

Statistical significance. Real-world evaluation is labor-intensive: each set of 16 trials takes approximately 30 minutes of human supervision time, which limits how many independent evaluations we can perform per configuration. To estimate the variance of our reported numbers, we repeat the evaluation across 5 independent sets of 16 trials and measure a standard deviation of 2.79% in success rate. The performance gaps reported throughout this chapter (e.g., the 25%–93% range across pre-training/scratch comparisons and the ∼20-point cross-lab transfer gap) are large relative to this variance, supporting the conclusions drawn from the experiments.

Inference Speed

Our sensorimotor model is designed to operate on latent visual representations that are computed per image, which decouples the vision model from the sensorimotor model and makes the vision computation cost independent of the context length. This enables us to use large vision models (up to ViT-L with 307M parameters) with system level inference speed of 10 Hz on a 2080 Ti GPU.

Emergent Self-Correction

We observe that some of our models that are pre-trained with sensorimotor prediction can exhibit an emergent self-correction behavior at test time. For example, if the robot fails to grasp an object initially, it would move back, and proceed to grasp the object successfully. Since these models were pre-trained and fine-tuned only with successful trajectories, this type of behavior has not been seen during training. Moreover, some of the states were not present in the data at all (e.g., starting a grasp or moving up with a fully closed gripper).

Causal Masking

We experiment with a variant of our sensorimotor pre-training approach that uses causal masking instead of random masking, like in the experiments so far. We make a minimal change and simply sort the sampled mask sequences. Note that the self-attention is still bidirectional rather than causal. The pre-training prediction problem is causal however. We consider two settings: pre-training on the same task (pick to pick) and pre-training on a different task (pick to stack). We find that causal pre-training leads to better performance on the same task and worse transfer across different tasks.

3.5 Related Work

Self-supervised learning in robotics. There is a rich body of work on self-supervised learning in robotics. [133, 134] learn grasping policies from large-scale self-supervision; [135] learns visuomotor policies with self-supervised visual correspondence; [136] calibrates learned grasping policies for particular objects based on grasp success feedback; [137] simultaneously collects robot trajectories for multiple tasks and learns corresponding policies; [138] uses an auxiliary contrastive learning task on human demonstrations; [139] learns goal-conditioned robot policies from teleoperation data; [140] uses multi-view reconstruction for model learning. Overall, prior work has largely focused on using self-supervision for learning a particular robotic task. In contrast, we use self-supervision for pre-training general sensorimotor representations that can be transferred to different downstream tasks.

Multi-task and large models for robotics. A number of works have explored learning multi-task and large models for robotics. BC-Z [116] performs large-scale vision-based imitation learning; [141] trains policies with cross-domain datasets; [142] learns multi-task Transformer policies for manipulation with language; [143] pre-trains with offline reinforcement learning. [109] trains a generalist agent on trajectories from different tasks jointly. In contrast, we use a masked objective, operate on latent visual representations, and focus on real-world trajectories. [45] trains language-conditioned Transformer policies from human or expert demonstrations. Likewise, we leverage Transformer models but focus on self-supervised learning from sensorimotor trajectories. [144] grounds language models with visual inputs via an embodied visual question answering model. Overall, we share the goal of multi-task robot learning and training large models for robotics but propose a general self-supervised pre-training approach that learns from real-world sensorimotor trajectories alone.

3.6 Discussion

Limitations. We note that our work has several important limitations. First, our dataset is collected using a single robot in a single lab, which limits the diversity and realism of the data. Scaling to more diverse environments and robots remains an important area for future research. Second, the tasks we consider, in both pre-training and fine-tuning, are relatively simple variations of pick and place. It would be good to explore more dexterous tasks with complex dynamics and contacts. Next, our model is pre-trained with the MSE loss and may struggle with multimodality. This is partially alleviated with conditional prediction over large context lengths. Nevertheless, it would be good to explore a generative model variant.

Conclusion. We describe an approach for robot learning with sensorimotor pre-training. Our model is a Transformer that operates on sequences of sensorimotor tokens. We pre-train our model by masked prediction. We find that pre-training on this data consistently outperforms training from scratch, leads to 2× improvements in the block stacking task, and

has favorable scaling properties. Finally, we demonstrate successful transfer across different tasks, lab environments, and robots.

Chapter 4

In-Context Imitation Learning via Next-Token Prediction

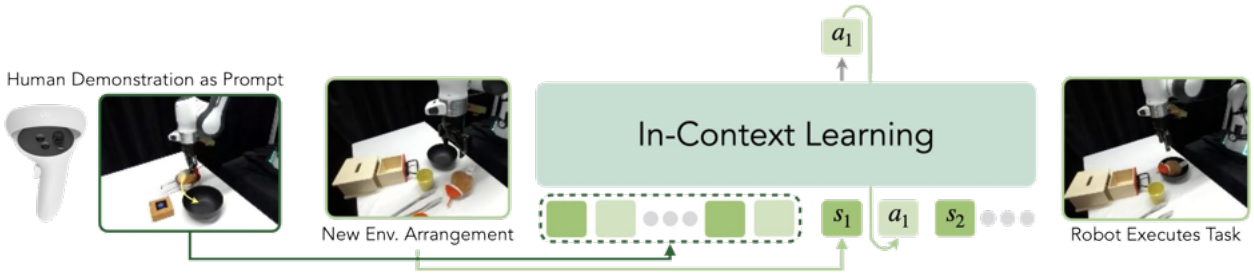


Figure 4.1: In-Context Robot Transformer. When we want a robot to learn a new task, in many cases, either we have to program new primitives to perform the task, or we have to provide many human demonstrations to train an imitation learning model. *Can a model learn the new task with few demonstrations without training?* We train a next-token prediction model to perform in-context imitation learning on a real robot. In particular, the model learns from robot trajectories to perform continuous action predictions. At inference time, we prompt the model with robot sensorimotor trajectories collected by human teleoperation and provide the model with the observation of the new environment, and roll out the policy on a physical robot.

4.1 Introduction

Learning-based single and multi-task robot policies have become increasingly capable [45, 46, 55, 109, 144–149]. This improvement in robot capabilities can largely be attributed to progress in related fields, particularly in vision and language modeling. Inspired by the recent development of large language models (LLMs) and large vision models (LVMs) [8, 11, 150], which formulate natural language processing and vision problems all as next-token-prediction, recent works also have formulated robot learning as next-token-prediction problems and

achieved state-of-the-art performance [45, 46, 50, 151]. Concurrently, there has been a surge in collecting large-scale robot datasets [141, 152–158] and pre-training models on these datasets [50, 159–162].

Despite being pre-trained on large datasets and showing some generalization ability, it is still challenging to teach these models to perform unseen tasks in different environments without additional training. New human demonstrations via teleoperation or new data collected from hand-crafted motion primitives, as well as another round of model-finetuning, are often needed to complete the new tasks. This process adds complexity to the workflow, making it challenging to apply these methods in real-world environments. Ideally, given one or a few demonstrations, the robot should be able to perform the task *immediately*. In their respective domains, LLMs and LVMs [8, 11, 150] have exhibited a similar ability, named *in-context learning*: a capability allowing the model to rapidly adapt to and recognize the task corresponding to the prompt provided at inference time without additional training.

Is the in-context learning capability of next-token prediction models limited to vision and language domains? In this chapter, we introduce In-Context Robot Transformer (ICRT), where we explore how next-token prediction models can be extended to perform real-robot in-context learning. For ICRT, the context is provided as a series of robot trajectories corresponding to a new task. The model learns from this context to perform the task in a different environment configuration without requiring additional training. A robot trajectory is a sequence of image observations, robot proprioceptive states, and actions. This trajectory implicitly encodes task primitives and the objects the robot needs to interact with. The model extracts this information from the prompt and then executes actions following a similar pattern in its current environment.

Compared to existing few-shot imitation learning approaches, ICRT offers a simple framework that avoids complicated loss functions, prior knowledge, and the need to identify key points or key frames, and operates directly on raw robot trajectories for continuous control. Additionally, unlike existing next-token prediction models for robot learning, ICRT features a long context window, allowing it to train on multiple sensorimotor trajectories from the same task and use one or more sensorimotor trajectories as prompts during inference.

Importantly, we observe that certain properties of the dataset are crucial for enabling in-context learning on real robots. Specifically, datasets that allow multiple tasks to be performed from the same initial observation are particularly beneficial. Unlike existing single-task datasets or many multi-task datasets where each environment has a unique object for robot interaction, these scenarios require the model to rely on the prompt to correctly identify the task and determine the appropriate object for interaction.

We make the following contributions:

1. We introduce ICRT, a robot foundation model that performs in-context learning on a real robot, which can effectively learn from context trajectories and perform unseen tasks without training.

2. We provide a new multi-task robot dataset and a training paradigm for fostering multi-task and in-context capability at inference time.

3. Physical experiments on a Franka Emika robot demonstrate that ICRT can learn from the provided context and perform the unseen tasks specified by the prompt at various generalization levels.

4.2 Related Works

Multi-Task Imitation Learning for Robotics

Imitation learning is an effective paradigm for equipping robots with various skills. The simplest algorithm in this domain, behavior cloning, has been successful across a wide range of tasks [163–165]. In recent years, alternative architectures such as energy-based models [111] and diffusion models [145] have also been proposed. Typically, these approaches require training a *separate* model for each task, although multi-task policies can be distilled from these task-specific models after training [166].

Recent advancements have shown that using transformers for next-token prediction in sequence modeling has been particularly effective in both language and vision domains, especially for *multi-task learning* [8, 126, 167]. In pursuit of developing generalist agents and multi-task robot policies, robot action planning is framed as a next-token prediction task using transformer-based architectures trained on large multi-task robot datasets [45, 46, 49, 50, 70, 110, 116, 147, 151, 168, 169]. Octo [50] and OpenVLA [151] represent the state-of-the-art among multi-task robotic policies. Octo [50] conditions on both goal images and language instructions, utilizing a transformer architecture with a diffusion head that fuses these inputs with current image observations to predict robot actions. OpenVLA[151] conditions solely on language instructions, fine-tuning a pre-trained vision-language model to predict robot actions based on visual observations and language inputs.

In-Context Learning

Despite training on large datasets, multi-task policies often struggle when faced with new objects, tasks, or environments, frequently requiring fine-tuning. Meta-learning has been shown to increase fine-tuning efficiency for generalization to new tasks [170–172], which has led to progress in few-shot imitation learning. To simplify the application of learned policies in the real world, recent approaches focus on methods that avoid fine-tuning model parameters for task generalization. Instead, these methods teach the model by providing demonstrations of tasks [118, 119]. Brown et al. [126] refers to this as "in-context learning", distinguishing it from approaches that rely on parameter fine-tuning.

Many in-context learning methods often employ contrastive learning to train context encoders, which identify the most similar training tasks to the test task in the latent space [116, 173]. However, how to effectively integrate these methods within the next-token-prediction framework remains unclear. Valassakis et al. [115] achieved one-shot in-context learning by training a visual servoing network to align the robot's end-effector with the object's relative

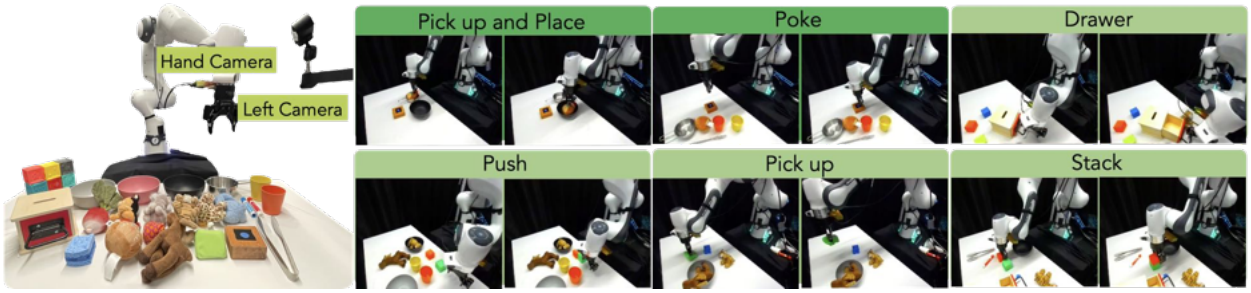


Figure 4.2: Our physical setup with the Franka Emika robot, the wrist and side camera and the objects used in training and evaluation. We consider 6 primitives for training and choose "pick up and place" and "poke" as the primitives for evaluation (dark green).

pose during the demonstration, but this approach requires an additional object segmentation model. Di Palo and Johns [118] introduced Keypoint Action Tokens, demonstrating in-context imitation learning using a large language model by representing demonstration trajectories as 3D coordinates with few-shot prompting. Unlike these approaches, ICRT operates without additional perception modules, processing raw image observations directly. Additionally, Vid2Robot [119] developed an encoder-decoder transformer that uses a demonstration video of a human and the current robot state as the prompt to generate robot actions. However, this method requires many auxiliary losses while ICRT uses a simple next-token prediction loss.

In this chapter, we focus on enhancing next-token-prediction models to perform real-world in-context imitation learning with robots. ICRT bypasses the need for additional context encoders by directly using robot sensorimotor trajectories from new tasks as prompts for the transformer-based model. ICRT is closely related to the seminal work, One-Shot Imitation Learning [114] and Prompting Decision Transformer [117]. [114] predicts the next action by applying cross-attention between a demonstration sequence on a new task and the current environment's state, while [117] employs a short trajectory prompt to encode task-specific information for guiding policy generation in offline reinforcement learning, using full state information and known reward functions. While both of these methods show their effectiveness in simulation, it is hard to have full-state information and known reward functions for all real-world robot manipulation tasks. To address these challenges, ICRT does not model rewards, utilizes a significantly longer context window, and demonstrates in-context learning capabilities in physical experiments using image observations.

4.3 Problem Statement

We consider in-context imitation learning under a real-robot manipulation setting. The goal is to train a model with in-context learning capabilities using a multi-task robotic dataset. At test time, the model can handle unseen tasks in novel environment configurations by

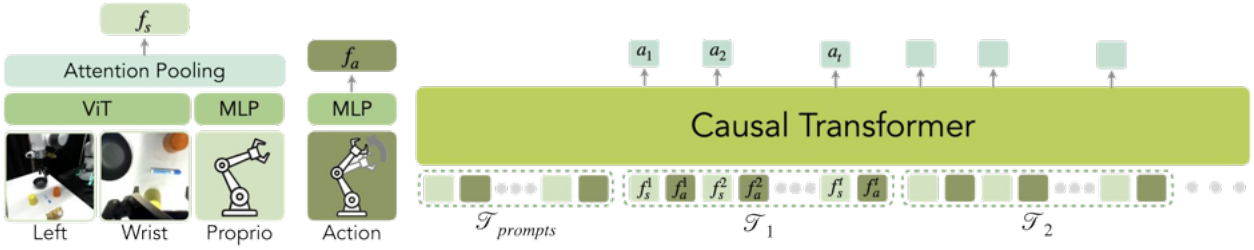


Figure 4.3: Method Overview: (Left) We encode camera observations with a pre-trained vision transformer. Additionally, we encode proprioception with an MLP. We concatenate the visual latent and the proprioception's latent and use attention pooling to extract a feature f_s as the current state representation. We encode the current action with an MLP to get f_a. (Right) We concatenate multiple trajectories of the same task and randomly sample the first k trajectories as the prompt. A causal transformer autoregressively predicts the next series of tokens. We decode the tokens that are at the position of the state features to generate the next $h = 16$ action via an MLP.

using a few new human-teleoperated robot demonstrations as prompts. Here, environment configuration refers to the objects present in the scene and their spatial arrangement. Notably, this process is achieved *without any additional training* on the new demonstrations.

We define *motion primitives* as distinct robot actions utilized to accomplish various *tasks*. Each task is characterized by 1) a specific motion primitive and 2) the set of objects the robot interacts with using that primitive. By altering the environment configuration at test time compared to the one in the prompt, we assess the model's ability to select the appropriate motion primitive and identify the correct object for interaction. In this work, we consider new tasks to be tasks involving unseen objects but using motion primitives from the training data (for example, training on picking up a tiger toy and testing on picking up a cube).

We make the following assumptions for ICRT experiments:

1. The model is trained on a diverse multi-task demonstration dataset. Each trajectory contains RGB observations from a fixed camera and a wrist-mounted camera, proprioception and action.

2. The task tested on the robot is within the reachable workspace of the robot.

4.4 Approach

In this section, we first introduce the data composition to facilitate in-context imitation learning. We then introduce the architecture and training objective for the transformer-based policy to effectively leverage the data.

Data Formulation

For model training, we consider a dataset $\mathcal{D}$ of visuomotor trajectories $\mathcal{T}$. Each trajectory of length t is a sequence of camera images i_t, proprioceptive robot states s_t, and actions a_t: $\mathcal{T} = (i_1, s_1, a_1, ..., i_t, s_t, a_t)$. We use the absolute end-effector pose as the robot's proprioceptive state and the delta robot end-effector pose between time steps as the action, which consists of delta translation, delta rotation and the continuous gripper action. We assume a known grouping of the trajectories so that the dataset can be partitioned into disjoint sets of tasks $\mathcal{D} = \bigcup_{k=1}^{K} S_k$, with $S_k \cap S_\ell = \emptyset,\ k \neq \ell$, where $\mathcal{S}_k = \{\mathcal{T}_{k_1}, ..., \mathcal{T}_{k_n}\}$. In practice, this grouping can be retrieved from the semantic labels of the dataset. In this work, we utilize the existing large robotic dataset DROID [64] and a multi-task dataset manually collected in our robot setup, which we name ICRT-Multi-Task (ICRT-MT).

DROID [64] is a joint effort from different organizations and contains 76k real-world demonstrations. We randomly sample 10k demonstrations from DROID after filtering out demonstrations shorter than 30 steps and longer than 450 steps. DROID dataset labels the task through human-specified language instructions, which may be different for the same task. We organized the DROID data by grouping demonstrations based on their language instructions CLIP text embedding cosine similarity. Specifically, we use a threshold of 0.9 for grouping demonstrations. To further facilitate in-context learning, we make sure that each task group contains at least 4 trajectories so that there are sufficient trajectories to serve as prompts for each other. This results in roughly 2k trajectories that we use for pre-training ICRT.

Many trajectories in the DROID dataset are collected in a single-task setup, where only one task can be completed in a scene (e.g. only one object is presented). In such a setup, the model can learn a shortcut solution to perform the task, by only focusing on the current observation but not the prompt. Therefore, multi-task data is crucial for the model to learn from the prompt. We manually collected a multi-task dataset ICRT-Multi-Task (ICRT-MT) using the DROID setup (Figure. 4.2). This dataset has 1098 trajectories in total, and contains 29 tasks with 6 primitives: picking, pick-and-place, stacking, pushing, poking, opening and closing drawers. Objects used in the data collection and examples of the primitives are shown in Figure. 4.2. In ICRT-MT, each environment is set so that there exist more than 2 possible tasks for the current observation so that the model has to distinguish and learn the motion from the prompt.

During the training, for each trajectory, we independently apply vision augmentation on the image observations by augmenting the brightness and contrast. We additionally apply random crops and scaling to the side camera observation. We also apply proprioception noise sampled from a normal Gaussian distribution $\mathcal{N}(0, 0.005)$. For each epoch, we randomly shuffle the order of trajectories from each task and concatenate them to form the training sequence. For each batch, we sample a subsequence of length $L = 512$ as the input to the model, where L is the sequence length defined as the number of observation, state, and action tuples. In practice, 512 steps usually contain up to 5 trajectories from the same task. We randomly select the first k trajectories and label them as the prompt within the sequence.

At least one complete trajectory is included in the prompt. This data grouping aims to capture inter-trajectory patterns, encouraging the model to generate action conditioned on the prompt trajectories. This approach differs from traditional behavior cloning methods, which typically use short input sequences that focus on modeling intra-trajectory behaviors.

Model Architecture

We construct the ICRT model with three parts: a pre-trained vision encoder, a series of projectors for each input modality, and a causal transformer backbone (Figure 4.3).

Vision Encoder The model processes multi-view image observations through a pre-trained vision transformer. However, most visual pre-trained networks are trained on ImageNet or human videos [43, 159, 162, 174], which exhibit a significant domain gap when compared to typical images from robot datasets, where the images frequently include robots or grippers. To minimize the domain gap, we pre-train a vision transformer [127] (ViT-Base) on an equal mix of ImageNet [132] and Open X-Embodiment [49] data, using CrossMAE as an efficient pre-training method [175]. During the training of the ICRT model, we freeze the vision encoder for efficiency. The vision encoder outputs the entire feature map for each of the cameras and is then fed into the proprioception projector (Figure 4.3 left).

Modality-Specific Projectors To project image observations, the robot's proprioceptive state, and actions into a shared latent space for sequence modeling, we design modality-specific projectors. At each timestep, the model takes as input a token representing either an observation or an action. To produce a single state token that captures fine-grained visual information and the proprioceptive state of the robot, we use attention pooling [176] between all visual tokens from a single camera's observation and a proprioception embedding produced by a multi-layer perceptron (MLP). The resulting embeddings for each camera are concatenated to produce a single state token f_s^t of dimension equal to the transformer latent dimension. Similar to proprioception, the action is embedded with an MLP into an action token f_a^t. This process produces a sequence of state and action tokens that are passed into the transformer.

Transformer Model The encoded sequence of state and actions is passed into a Transformer model [2], following the design of Llama2 [8]. The transformer takes as input the sequence of state and action features $(f_s^1, f_a^1, \cdots, f_s^t, f_a^t)$ that are produced by the modality-specific projectors. We add MLP decoders to produce state and action outputs from the last layer of the transformer at the appropriate positions. We denote the transformer with the decoder heads as g_θ. Therefore, the desired outputs are the shifted sequence of proprioceptive states and actions $(a^1, s^2, a^2, \cdots, a^t, s^{t+1})$. This naturally forms a next token prediction problem, as $g_\theta(f_s^1)$ predicts a^1 and $g_\theta(f_s^1, f_a^1, \cdots, f_s^n)$ predicts a^{n+1}. In practice, we find it beneficial to predict the next h actions at each time step, and use temporal ensembling [55] to execute the final action.

Inspired by Octo [50] and vision transformers [127], we consider a randomly initialized Llama2 model of 12 layers with a latent dimension of 768, which we name *Llama2-Base*. In addition, multiple works have shown that multimodal inputs can be aligned to large-language

models [46, 167, 177–179]. Multi-modal language model, Palm-E [144] has shown success in enhancing generalization when being directly incorporated into robotic control [46]. Therefore, we also investigate the effectiveness of using a large-language model for in-context robot learning by initializing the transformer with a pre-trained Llama2-7B. Due to the large domain gap between natural language and robot trajectories, a frozen language model may not be sufficient. Therefore, similar to prior work in multimodal alignment, we fine-tune the language model with LoRA [180], with a rank of 32. Due to compute resource limitations, we are unable to fully fine-tune the model.

Loss Function To provide more supervision signals so that the model can better respond to the trajectory "prompt" we provide at test time, we reference works in training multi-turn conversation chatbots [167, 181], where they only compute loss on the response generated by the chatbot, instead of the prompt. Recall that in Section 4.4, we randomly sample the subsequence of the concatenated trajectories as the prompt trajectory. Analogously, we only compute action prediction with L1-loss for the actions after the prompt trajectories.

Inference The simplicity of the next-token prediction objective makes inferencing with ICRT straightforward at test time. As shown in Figure. 4.4, we provide one or more human-teleoperated demonstrations in the form of robot sensorimotor trajectories (formatted identically to the training data), along with the current image observations and the robot's proprioceptive state as inputs. The model then predicts the next action, which is executed by the robot. After each action, the policy receives updated image observations and proprioceptive state, allowing it to iteratively predict and execute subsequent actions.

A key advantage of this framework is its use of the transformer's sequential processing capability. Instead of reprocessing the entire sequence history for each model evaluation, as seen in previous works [45, 46, 50, 151], the model employs a key-value (KV) caching mechanism, as discussed in [8]. This mechanism stores previous outputs, allowing the model to compute only the outputs for the new token. This approach significantly reduces computational overhead, lowering the complexity from quadratic to linear relative to the sequence length. Empirically, ICRT can run inference at 39.6 Hz, allowing it to perform real-time close-loop control.

4.5 Experiments

In this section, we design an experimental setup to evaluate the in-context learning capabilities of the proposed models and compare them against several baselines. Instead of focusing on the difficulty of learning a specific task primitive, we design the experiments to assess the policy's ability to accomplish novel tasks based on the provided prompt trajectories.

Experiment Design We consider two action primitives: a *pick-and-place* primitive and a *poking* primitive. For each action primitive, we design *six unseen tasks* (as defined in Section 4.3), with three tasks evaluating *in-domain* object generalization (selected from *yellow cube, red cube, black cube, pink bowl, and blue bear*) and three objects *unseen* during training (selected from *radish, blue sponge, grey dog, and black dog*).

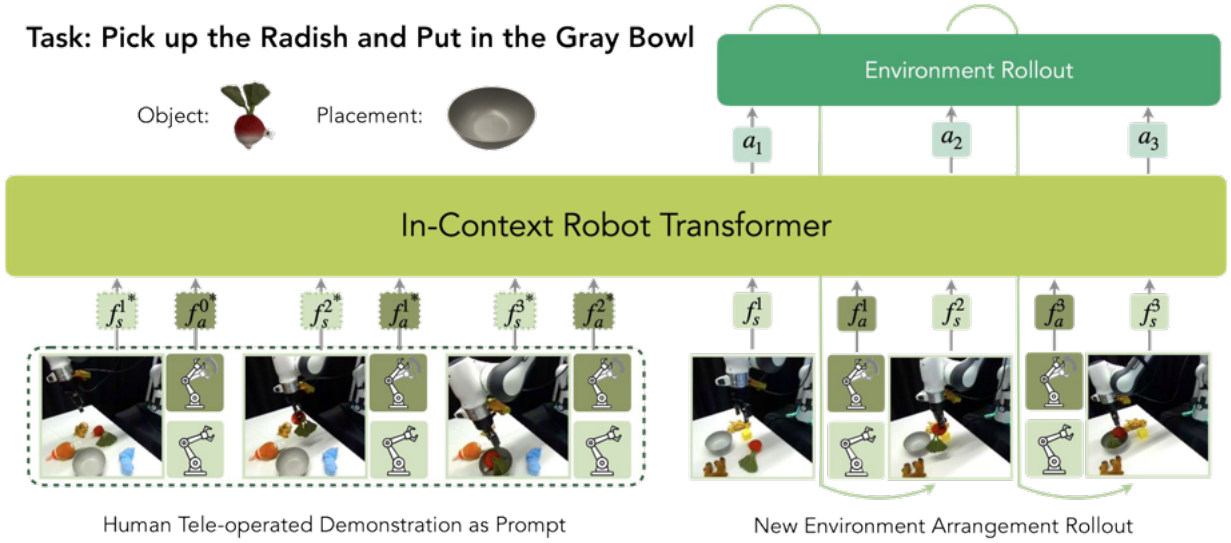


Figure 4.4: Example inference pipeline of ICRT on the task of picking up the radish and putting in the gray bowl. A human teleoperated demonstration trajectory consisting of image observations, proprioception and actions are provided as the prompt. ICRT takes the prompt and the current observation in a different environment to accomplish the task.

Each task has five difficulty tiers. In the pick-and-place task, the model must identify the correct object to grasp and where to place it in a multi-object or multi-placement scenario. The tiers include: 1) no distractors, 2) one distractor object, 3) two distractors, 4) three distractors, and 5) one distractor placement position. In the poking task, the robot closes the gripper, pokes the object, lifts the end-effector, and opens the gripper, with tiers involving 0-4 distractor objects.

The pick-and-place task is scored with 0.5 for a correct pick and 1 for a successful placement. In the poking task, failure is marked if the wrong object is poked. The model has 25 seconds (375 steps) for retries. Each difficulty level is attempted once, and we report the average success rate per task, along with the average success rate and standard deviation across the six tasks for each action primitive.

Models The default **ICRT** is a randomly initialized Llama2-Base model pretrained on DROID and fully fine-tuned on ICRT-MT. We evaluate the impact of model initialization and training datasets by introducing the following three variants: 1) **ICRT-Llama2**, a pre-trained Llama2-7B language model fine-tuned on ICRT-MT with LoRA; 2) **ICRT (DROID)**, a randomly initialized Llama2-Base model trained only on the DROID dataset; and 3) **ICRT (MT)**, a randomly initialized Llama2-Base model trained only on the ICRT-MT dataset.

We consider 3 baseline models. We train a goal-conditioned policy, where the goal observations are always prepended to the sequence, and each sequence is from one trajectory. This resembles the normal goal-conditioned imitation learning setup. Additionally, we finetune Octo [50], the state-of-the-art goal-image and language conditioned policy, and OpenVLA [151],

	Pick and Place	Poke	Average
Goal Condition	33.3 (±6.5)	6.7 (±4.6)	20.0 (±4.3)
Octo [50]	5.0 (±2.7)	13.3 (±6.2)	9.2 (±3.5)
OpenVLA [151]	11.7 (±4.6)	3.3 (±3.3)	7.5 (±2.9)
ICRT	**65.0 (±7.3)**	**93.3 (±4.6)**	**79.2 (±4.6)**

Table 4.1: Main Results. ICRT outperforms two state-of-the-art robot foundation models that are conditioned on goals or language in both pick-and-place and poking tasks. We evaluated each task primitive using six tasks not seen during training, conducting five trials per task for a total of 30 trials per primitive and 60 trials overall to calculate average performance. For each model, we report the mean success rate for each task, the overall success rate, and the corresponding standard error in parentheses.

the state-of-the-art language conditioned multi-task imitation learning policy. Octo is fine-tuned using their official fine-tuning recipe. We incorporate action chunking into OpenVLA by asking it to predict the next 16 actions, which performs better than vanilla OpenVLA which predicts only the next step. Both of these methods are representative of robot policies that use next-token prediction objectives.

Prompt Generation For each task, we collect 3 demonstrations (with zero, one distractor object, a distractor placement for pick-and-place, or two distractor objects for poking) as the prompt in total before running the experiment. Please refer to the website for a visual example. During testing, a random demonstration is drawn as a prompt to assess the model's ability to generalize to different prompts. It's important to note that the environment setup during policy rollout *differs* from the prompts' setup, ensuring that the evaluation measures the model's understanding of task-relevant information from the prompt, rather than simply copying actions from it.

Results We present the results in Table 4.1. For the pick-and-place primitive, we observe that the goal-conditioned policy generally succeeds in identifying the correct object to grasp when no distractor objects are present. However, its performance degrades significantly as the number of distractors increases. When the goal image only specifies the task but not the specific way to achieve it in the current environment, goal-conditioned policies often fail to execute the task effectively.

Octo struggles with determining which object to interact with and where it should be placed, highlighting the challenges posed by our experimental setup for multi-task policies. OpenVLA, while often moving towards the correct object, frequently fails in grasping the object or mistakenly performs the wrong task (e.g., grasping instead of poking, and vice versa). This indicates that OpenVLA may require a greater number of demonstrations (more than 50) per task to achieve better performance, and that relying solely on language conditioning may not be sufficient for generalization to new tasks.

The results suggest that ICRT outperforms the goal-conditioned policy in identifying the correct object to pick up and the appropriate placement location. The poking task presents

	Pick and Place	Poke	Average
ICRT-Llama2	43.3 (±7.9)	73.3 (±8.2)	58.3 (±6.0)
ICRT (DROID)	0.0 (±0.0)	0.0 (±0.0)	0.0 (±0.0)
ICRT (MT)	**76.7 (±7.1)**	70.0 (±8.5)	73.3 (±5.5)
ICRT +Prompt Loss	21.7 (±6.2)	23.3 (±7.9)	22.5 (±5.0)
ICRT	65.0 (±7.3)	**93.3 (±4.6)**	**79.2 (±4.6)**

Table 4.2: Ablation Study. We ablate three key design choices: fine-tuning ICRT using a language model, training solely on the relevant dataset or the fine-tuning dataset, and the impact of including prompt loss in the training process. We report the mean success rates and their corresponding standard errors for the pick-and-place and poke tasks, as well as the overall average performance.

a significant challenge for the goal-conditioned policies, as the goal position often closely resembles the start configuration. However, after conditioning on the prompt trajectory, ICRT is able to correctly identify the task as poking, and the results indicate that it consistently reaches the correct target object while ignoring distractors. Despite this, we do observe some failure modes with ICRT, such as missing the grasp of the target object, grasping the wrong object, or placing objects in incorrect locations. Specifically, when a distractor object shares the same color but has a different shape, the model struggles to accurately determine which object to grasp. This implies that additional fine-tuning of the vision encoder might be required to enhance model performance, a conclusion also reached by OpenVLA [151].

4.6 Ablations

In this section, we provide additional experiments presented Table 4.2 that ablate on a few core design choices. We provide additional ablation studies on the official website.

Model Initialization

We conducted ablation studies to examine the impact of using a pretrained Llama2 on language data and fine-tuning it for robot sensorimotor sequence modeling. The results, presented in Table 4.2, show that although ICRT-Llama2-7B achieves a lower training loss, its performance is worse compared to its smaller counterparts. This discrepancy may be attributed to a lower inference frequency of ICRT-Llama2 (10.7 Hz vs 39.6 Hz). Future work can focus on optimizing the inference speed of ICRT-Llama2 to improve performance.

Training Dataset

We find that training on the DROID subset (see Section 4.4) is insufficient for completing any of the test tasks; the policy (ICRT (DROID)) shows no progress across all tasks. This suggests

that although the DROID subset may offer greater visual diversity, the unique structure of ICRT-MT—where multiple tasks are performed from the same initial observation—is particularly beneficial in developing the in-context learning capabilities of a next-token prediction robot model.

ICRT (MT) shows similar performance to ICRT that is pre-trained on DROID, especially for the pick-up and place primitive, even surpassing ICRT on the *put radish in grey bowl* task. However, ICRT (MT) does not perform as well on the poking primitive. The results suggest that it may be beneficial to pre-train the autoregressive model on a large dataset, as a diverse dataset may help the transformer to perform better alignment between visual features and control.

No Prompt Loss

Following the design of many multi-turn conversation large language models or vision language model fine-tuning works [167, 181–183], we do not calculate the loss for the predicted action in the prompt trajectories but only do so on the predictions after the prompt trajectories. We mark the model that calculates loss on the prompt as **ICRT +Prompt Loss** and the default model as **ICRT**. The results are shown in Table 4.2. We find that only predicting the trajectories after the designated prompt trajectories can significantly improve the model's performance. We hypothesize that in the situation where there is a loss on the prompt trajectories, the model is forced to do unconditional generation based on current observations for those prompts. This may cause the model to stop paying attention to the prompt, especially when there are multiple possible tasks available.

4.7 Limitations and Conclusion

While results suggest that ICRT learns from the prompt trajectories and generalizes to unseen objects, tasks, and certain primitives that resemble the ones in training[1], it is still unclear how to generalize to completely unseen action primitives. Future works should investigate how scaling model capacity and scaling dataset can help with primitive-level generalization. In addition, ICRT assumes a fixed robot morphology with a fixed impedance controller. Future works can also investigate how to facilitate transfer between different robot morphologies by learning a unified policy on different robots. ICRT-Llama2 has a low inference frequency which may contribute to its low performance. We hope to speed up ICRT-Llama2 at inference time in the future.

In summary, we present ICRT, where we study in-context, multi-task imitation learning on a real robot. We achieve this by training a causal transformer model on sequences of robot trajectories, where trajectories from the same task are combined to provide context for task execution. Additionally, we introduce a multi-task dataset to facilitate this in-context learning approach. Our experiments show that by using robot sensorimotor trajectories as

[1]Please refer to the appendix and videos on the website for more detail.

context, the model can generalize learned motion primitives to unseen objects and novel environment configurations, particularly in scenarios where multiple tasks are present.

Chapter 5

OTTER: A Vision-Language-Action Model with Text-Aware Visual Feature Extraction

5.1 Introduction

Recent advancements in Large Language Models (LLMs) and Vision-Language Models (VLMs) have inspired the exploration of scaling datasets and computational resources for vision-language-action (VLA) models [49, 50, 64, 151]. Different input modalities are usually encoded into separate tokens: multi-view images encoded via visual feature extractors, along with tokenized language instructions, optionally with the robot's proprioceptive states, are fed into a transformer-based robot policy for end-to-end action generalization. This approach requires the policy network to connect the vision and language information and conduct precise robot control, which often presents significant challenges, especially in unseen environments.

Existing works such as RT-2 [46] and OpenVLA [151] have demonstrated the benefits of directly fine-tuning pre-trained VLMs on robotic datasets to map vision and language to control. While these approaches leverage rich visual and language features from pre-trained encoders, this fine-tuning may interfere with pre-trained vision and language features, particularly since robot datasets are far less semantically diverse compared to the large vision-language datasets [184] on which these VLMs are trained, often leading to a noticeable drop in performance on unseen objects or environments compared to seen tasks [46].

This finding is consistent with observations in vision-language research [120, 121], where fine-tuning language-aligned vision encoders, such as CLIP [185], has been shown to overfit and degrade generalization and long-tail classification performance. Yet, models like CLIP and SigLIP [37] already exhibit strong vision-language alignment and impressive zero-shot performance on various downstream tasks, including fine-grained tasks like open-vocabulary segmentation [121, 186, 187]. This raises the question of whether a more effective strategy is to extract and utilize the pre-trained alignment rather than risk weakening it through

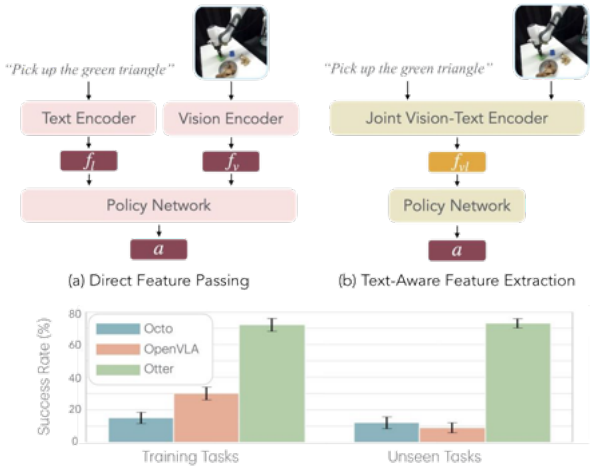


Figure 5.1: *(Top)* Different feature extraction approaches in VLA models. (a) Direct Feature Passing: existing approaches, exemplified by Octo and OpenVLA, extract and pass visual and text tokens independently to the policy network. (b) Text-Aware Feature Extraction: the proposed approach, OTTER, extracts visual tokens that correspond to the text tokens, and then feeds them into the policy. *(Bottom)* Real-world Robot Experiments: OTTER demonstrates higher success rates on both training and unseen real-world robot pick-and-place tasks compared to Octo and OpenVLA. OTTER exhibits better zero-shot generalization to unseen objects, maintaining strong performance across a variety of novel tasks.

fine-tuning.

We seek to preserve the generalizability of VLMs for effective performance in unseen scenarios. To this end, we propose OTTER, a novel VLA architecture that freezes pre-trained vision and language encoders and extracts task-relevant visual features guided by language instructions. Instead of passing all visual features to the policy network, OTTER selectively extracts those semantically aligned with the task description. We use CLIP for its zero-shot capabilities and wide adoption, employing a lightweight selection mechanism to preserve its pre-trained vision-language understanding while adapting it for robotic control.

Figure 5.2 illustrates the architecture of OTTER, which incorporates text-aware feature extraction into the vision-language-action (VLA) pipeline. At each timestep, OTTER first processes visual and textual inputs to pinpoint task-relevant visual tokens. These selected features, along with proprioceptive data, enable the policy network to concentrate specifically on action planning. By keeping the pre-trained vision-language encoders frozen, OTTER effectively decouples task planning (selecting relevant visual features) from robot action planning (predicting appropriate actions). Both physical and simulation experiments demonstrate that OTTER outperforms existing VLA models, showing strong generalization to novel objects and environments with less performance degradation (Figure 5.1).

To summarize, our contributions are:

1. We propose OTTER, a VLA model that leverages the semantic alignment capabilities

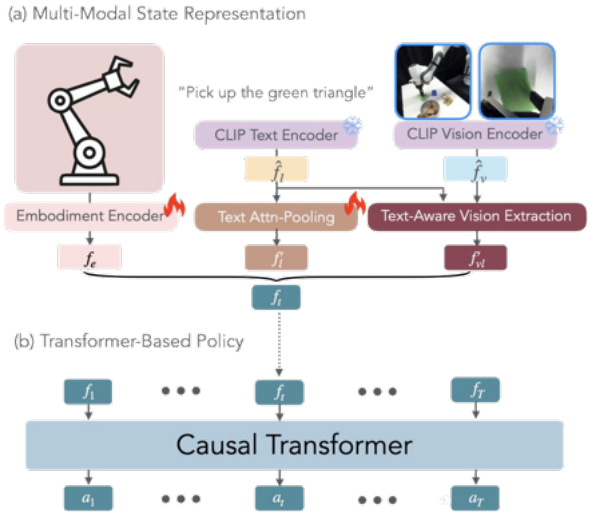


Figure 5.2: OTTER Model architecture. At each timestep t, text-aware visual features f_{vl} are extracted from a pre-trained CLIP model (see Figure 5.3). Then f_{vl} and the text tokens f_l are further compressed through separate attention pooling layers, producing two representations f'_{vl} and f'_l, respectively. The robot's proprioception is encoded by an MLP layer to generate the embodiment representation f_e. The tokens f'_l, f'_{vl}, and f_e are then concatenated to form f_t, which serves as input to a causal transformer. With a context window of T steps, the model autoregressively predicts the future actions (a_t) at each step.

of pre-trained VLMs for better generalization. By extracting text-aware visual features that are semantically aligned with task instructions, OTTER preserves and utilizes the rich visual-language understanding from pre-training for effective robotic task execution.

2. OTTER significantly outperforms state-of-the-art VLA models on unseen robot manipulation tasks through its zero-shot generalization capabilities. By preserving the frozen pre-trained vision-language model rather than fine-tuning, OTTER effectively leverages the semantic understanding from large-scale pre-training to handle novel objects and environments.

3. Empirical results suggest that OTTER's performance on unseen tasks scales along multiple axes: through larger pre-trained vision-language encoders, increased policy network capacity, and pre-training on larger robot datasets.

5.2 Related Work

Vision Language Pre-training

Vision-language pre-training (VLP) seeks to improve the performance of downstream tasks that involve both vision and language by training models on extensive datasets of image-text

pairs. A prominent class of vision-language models leverages contrastive learning [37, 185, 188–192]. Among them, CLIP [185], which was trained on a private WIT-400M dataset of image-text pairs, demonstrates impressive zero-shot capabilities across various downstream tasks, including image-text retrieval and image classification through text prompts. Furthermore, CLIP shows potential for application in broader fields such as decision making and robotics, where robots are required to perform language-specified tasks based on visual inputs.

Recent multimodal foundation models, such Qwen-VL [193] and Llama 3-V [10], all follow a similar pattern: they extract visual features by finetuning language-aligned vision models and train cross-attention layers to inject visual features into the language model. However, many researchers have observed that when data is scarce, fine-tuning or even applying additional layers on top of CLIP (instead of using raw CLIP features) [120, 121] may result in models with weaker reasoning capabilities compared to vanilla CLIP. This motivates our approach in OTTER, where we preserve CLIP's strong vision-language alignment capabilities by keeping the model frozen and extracts visual patch features that correspond to the text query in a parameter-free way.

Vision Language Action Models

In recent years, there has been a surge of interest in developing robot foundation models, largely inspired by the success of large language models (LLMs) and vision-language models (VLMs) [4, 39, 185, 194–198]. A key hypothesis driving this trend is that more capable robot foundation models can emerge by scaling up robot datasets, increasing model capacity, and co-training or pre-training models on vision and language datasets. This has led researchers in the robot learning community to train robot foundation models, investigate pre-training strategies, and iterate on model designs [45, 46, 49, 50, 110, 113, 116, 147, 151, 169].

Many existing VLMs [167, 199, 200] use a similar approach, where visual features and languages are directly passed into the LLM to generate answers. Similarly, the majority of Vision-Language-Action (VLA) models also opt for this approach, where language, vision, and robot proprioception data are separately encoded by modality-specific feature extractors before being fed into a single transformer policy. This method has shown promise in many language-conditioned multi-task learning models [46, 49, 110, 116, 147, 169], including current open-source state-of-the-art models such as Octo [50] and OpenVLA [151].

In contrast to the above approach, OTTER combines vision and language input before feeding them into the robot policies by extracting and passing text-aware visual features. Early works such as FiLM [201] encode text information and fuse these features into each block of a ResNet [202]. RT-1 [45], one of the first language-conditioned robot policies, uses FiLM to encode visual and text information for action generation. However, RT-1 learns the language-vision alignment from robotic data without leveraging pre-trained models such as CLIP [185], where visual features are already aligned with text.

OTTER distinguishes itself by retrieving CLIP's visual patch features that best correspond to the language task description using cosine similarity before the policy transformer. These text-aware visual features, language features and the robot's proprioceptive state are fed into

the robot policy. This approach allows the model to leverage the fine-grained features from the pre-trained vision-language models while incorporating robot-specific information.

5.3 Method

We propose OTTER, a vision-language-action model for learning a robot manipulation policy through extraction of text-aware vision features from a pre-trained VLM. We first describe how OTTER utilizes the vision-language alignment of pre-trained vision and language encoders to extract text-aware vision features, then provide a more detailed explanation of the model architecture.

Text-Aware Visual Feature Extraction

OTTER utilizes a pre-trained CLIP for vision and language features extraction. Consider a ViT-based CLIP vision encoder [185] consisting of a series of residual attention blocks. Each of these blocks takes as input a collection of visual tokens $X = [x_{\text{cls}}, x_1, \ldots, x_{h\times w}]^T$, where x_{cls} represents the learnable global class token, and outputs the feature X_{out} as shown below:

$$q, k, v = \text{Proj}_{q,k,v}(\text{LN}(X)) \tag{1}$$

$$X_{\text{sum}} = X + X_{\text{attn}} = X + \text{Proj}(\text{Attn}(q, k, v)) \tag{2}$$

$$X_{\text{out}} = X_{\text{sum}} + \text{FFN}(\text{LN}(X_{\text{sum}})) \tag{3}$$

Proj, LN, and FFN denote linear projection matrix, layer norm [203], and feed-forward network respectively. A recent work ClearCLIP [121] demonstrates that CLIP's last self-attention block's attention feature X_{attn} contains cleaner semantic information than the CLIP's output feature X_{out}. While ClearCLIP uses the cosine similarity between text and visual features for segmentation, we leverage this similarity to construct task-relevant visual features for robotic control. Specifically, we use the similarity scores to select and combine visual features that best align with the task instruction, creating compact representations for downstream action prediction.

In OTTER, we extract text per-token features from CLIP's language encoder f_l (m tokens). For the visual features, motivated by the improved ability of ClearCLIP to capture text-aligned visual features, we specifically utilize the attention output X_{attn} from the last vision attention layer, rather than the CLIP's output feature X_{out}, denoting it as f_v (n tokens), where $n = h \times w$ is the total number of patch tokens from ViT. Figure C.1 demonstrates how using X_{attn} enhances the alignment between visual features and language semantics, illustrating the effectiveness of this approach. More details are in Section C.3.

Since the language features and the visual features have different dimensions, CLIP uses a matrix per modality to project the network's output feature to the same latent dimension, denoted as w_l and w_v for language and vision respectively. We normalize the text and visual features for vision-language fusion. The text features are normalized using the final layer

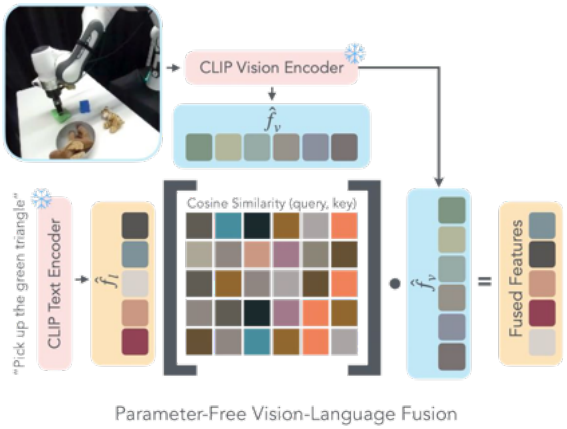


Figure 5.3: Text-aware Visual Features Extraction We calculate the similarity between the visual patch features and per-token language features, then take the softmax over the patch feature dimension. Intuitively, this gives a distribution of semantic similarity over all spatial locations. We then multiply the visual patch features to retrieve the visual semantic features that correspond to each token in the sentence. In Section C.3, we provide more in-depth analysis and visualizations of the proposed method.

normalization: $\hat{f}_l = \text{LN}_{\text{final}}(f_l)w_l$. The visual features are normalized using the post-attention layer normalization: $\hat{f}_v = \text{LN}_{\text{post}}(f_v)w_v$. We apply L2 normalization to both text and visual features: $\hat{f}_l = \hat{f}_l/\|\hat{f}_l\|_2$ and $\hat{f}_v = \hat{f}_v/\|\hat{f}_v\|_2$ as in standard CLIP.

With the normalized features, we perform temperature-weighted attention:

$$f_{vl} = \text{softmax}(\hat{f}_l \hat{f}_v^\top / \tau)(\hat{f}_v + PE) \tag{4}$$

where τ is the temperature parameter. PE is the 2D sin-cos position embedding of the patch [127], which informs the policy network of the spatial location of each patch. Same as in CLIP [185], τ is learnable and is clipped between 0 and 100. The resulting features $f_{vl} \in \mathbb{R}^{m \times d}$ are the text-aware visual tokens, where each row is a linear combination of normalized visual features $\hat{f}_v$. Intuitively, the *softmax* serves as a selection function, where patch features relevant to a particular language token are selected, and a weighted average of these patches is calculated to provide cues about where the robot policy should pay attention. A smaller τ sharpens the *softmax*, concentrating the selection on the patch with the most similar feature, while a larger τ produces a smoother, more evenly distributed selection across patches. Critically, only τ is learnable and the entire CLIP model is *frozen* throughout training.

Model Architecture

Policy Network Input We compress the extracted text-aware visual features f_{vl} into a single token for each camera. To achieve this, we apply a *learnable* cross-attention pooling

operation to each camera's f_{vl} to obtain a single feature f'_{vl}. Specifically, we use $N_q = 4$ learnable queries q, and keys k and values v from f_{vl}, and compute the output using cross attention $X_{attn}(q, k, v)$. We concatenate the N_q output tokens to one single token (f'_{vl}). To facilitate better instruction following capabilities, we additionally employ another *learnable* cross-attention pooling on the text features f_l, resulting in a single text token $f'_l \in \mathbb{R}^{d_l}$. The robot's proprioceptive state is encoded through a Feed-Forward Network (FFN) to extract an embodiment feature f_e. At time step t, we concatenate the embodiment feature f_e with the perception feature f'_l and f'_{vl} along the channel dimension to create a single token f_t. This token serves as input to a policy network for action prediction.

Policy Network and Action Head OTTER uses a transformer as the policy network, consisting of 4 layers and 8 heads, with a hidden dimension of 512. Fed by the combined features from the perception and embodiment, the model generates an action a_t. The model is trained with a context length of $T = 12$ steps. For each output token at a given timestep, we use a FFN to predict the next 12 actions. More details about our model architecture can be found in Section C.2.

Proprioception Parametrization We parameterize the proprioception space using a 10-dimensional representation. This includes the absolute end effector translation (x, y, z), a 6DoF rotation vector, and a continuous end-effector gripper state. The 6DoF rotation vector is derived by flattening the first two rows of the $SO(3)$ rotation matrix.

Action Parametrization We employ delta end effector pose as our action parameterization. At each prediction step, the model predicts t actions. Given a sequence of *absolute* end effector action transforms $T_1, T_2, \cdots, T_t$ in a trajectory and the current end-effector pose T_{ee}, we define the relative transforms that the model needs to predict as $T_{\text{ee}}^{-1}T_1, T_{\text{ee}}^{-1}T_2, \cdots, T_{\text{ee}}^{-1}T_t$. We then append the continuous absolute gripper position to each delta action. Similar to the proprioception representation, we express the delta action using the relative end effector translation and a 6DoF rotation vector, resulting in a 10-dimensional action representation. When executing the predicted actions, we employ temporal ensembling [55] in conjunction with receding horizon control [145]. Through experimentation, we determined that an action horizon of 8 steps yields optimal performance.

5.4 Experiments

We consider two classes of problems: language-conditioned multi-task learning and zero-shot generalization in unseen environments. For language-conditioned multi-task learning, given a multi-task setup (defined as a setting in which there are many tasks that can be performed in the same scene), the policy needs to perform the correct task corresponding to the language instruction. In the zero-shot generalization setup, the policy is provided with a language description of an unseen task, and is asked to perform the specified task in the unseen environments. We introduce our experimental setup to evaluate the instruction-following and text-aware visual features extraction generalization of OTTER in Section 5.4 and the baselines considered in this chapter in Section 5.4.

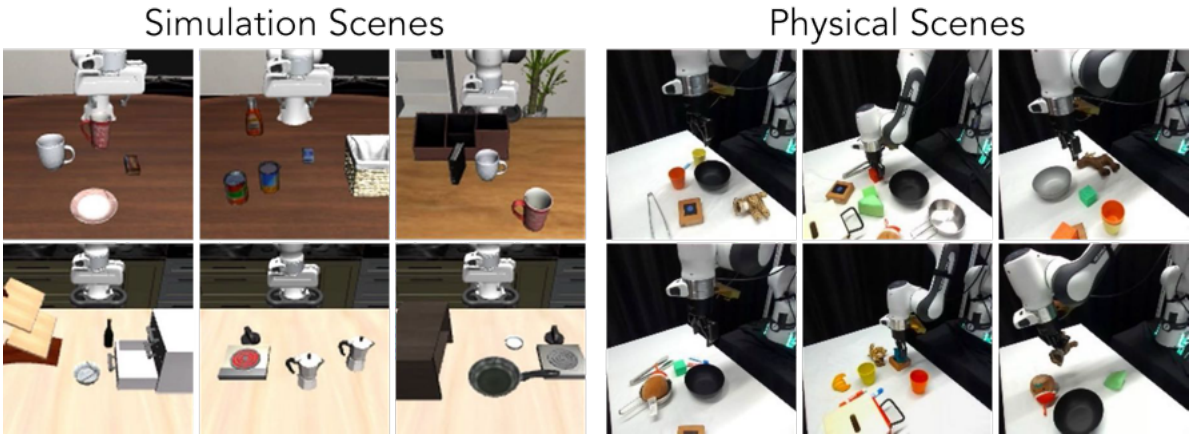


Figure 5.4: Example scenes in the simulation (left) and in the physical environments (right) using a Franka robot.

Environment Setup

Simulation Environment We use the LIBERO benchmark [204] for simulation evaluation. Specifically, we use the tasks and datasets in LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-90, which contains diverse objects, scene layouts, and language instructions. Each simulation task has 50 demonstrations. We evaluate OTTER's capabilities on both in-distribution tasks and unseen tasks. The in-distribution tasks are the 30 tasks in the original LIBERO-Spatial/Object/Goal, which can evaluate the model's multi-task learning capabilities. In addition, we also construct 10 novel tasks, where we modify the language instructions and corresponding objects of 10 original tasks from LIBERO-90. For the 10 unseen tasks, we follow the same convention in LIBERO [204] about object initialization and goal configuration by defining task bddl files. Example scenes in the simulation are shown in the left column of Figure 5.4.

Real Robot Environment For real-robot evaluation, we consider four robotic task primitives: pick up and place, poking, pouring and opening/closing a drawer. We define task as the combination of the primitive and objects involved. We collect robotic datasets on multi-task scenes using a Franka robot, where there are multiple tasks that can be completed in the same scene. We consider 10 pick-and-place tasks each containing 50-80 demonstrations of a human tele-operating the robot, resulting in a total of 724 demonstrations. We denote this dataset as DS-PnP. For each of the other three primitives we collect 100-200 demonstrations, with a total of 1,185 demonstrations. We denote the dataset consisting of all four primitives as DS-ALL.

We consider a task with *unseen* target objects for task completion as an unseen task. The training tasks involve objects encountered during model training, whereas the unseen tasks test the model's ability to generalize to unseen objects or scenes. For example, the model is trained on the task of poking a wooden block and is tested on the new task of poking a

radish. Example scenes in the real world are shown in the right column of Figure 5.4.

We consider 19 in-distribution training tasks and 15 out-of-distribution unseen tasks across the 4 primitives (Table C.2). For each task, we evaluate the model for 10 experiment trials. For each experiment trial, we vary the location of the target object and introduce 2-3 random distractor objects, to evaluate the instruction following capability of the VLA models. In the unseen tasks, we provide the robot with novel target objects that are unseen during training, or novel combinations of target objects and target placement locations. This setup aims to evaluate both object identification and task completion ability under more challenging and previously unseen conditions. The robot must identify and interact with the correct object based on the provided language instruction and complete the assigned task.

The trial is terminated either when the task is completed or when a time limit is reached. The overall performance is measured by calculating the average success rate with standard error across all trials for the training and unseen tasks. The full lists of simulation and real-world environments and more experiment details can be found in Section C.1.

Baselines

To evaluate if the text-aware visual features extracted in OTTER can better leverage the semantic understanding capabilities of the pre-trained VLMs, we consider four baselines that directly take all visual and language features, including three state-of-the-art open-sourced VLA models and one variant of OTTER:

1. Octo [50], an open-sourced transformer-based policy trained from scratch on 800K trajectories from the Open X-Embodiment dataset [49].
2. OpenVLA [151], a fine-tuned Prismatic-7B [200] VLM on the Open X-Embodiment (OXE) dataset.
3. π_0-Fast-Droid [205], a VLA using a pre-trained PaliGemma [206] and a Fast [207] action tokenizer. This model is pre-trained on a subset of OXE and the π dataset, and is fine-tuned on the Droid [64] dataset.
4. Direct Feature Passing OTTER (DFP-OTTER): a variant of OTTER where the text tokens, vision tokens are passed to an attention pooling layer separately to obtain independent tokens, which are then concatenated with the embodiment feature f_e as the input to the transformer. This baseline is constructed to inform the importance of text-aware visual feature extraction.

5.5 Results

We compare OTTER against several baseline in both real-world (Section 5.5) and simulation environments (Section 5.5). We compare OTTER against several ablations in Section 5.5. In Section 5.5, we further investigate OTTER's capabilities by scaling up models.

Method	Training Tasks	Unseen Tasks
π_0-Fast-Droid	-	61% ± 5.3%
Finetuned Octo	15% ± 3.4%	12% ± 3.6%
OTTER w.o. CLIP vision	17% ± 2.9%	11% ± 2.5%
Finetuned OpenVLA	30% ± 3.9%	9% ± 3.1%
DFP-OTTER	29% ± 3.7%	4% ± 1.6%
OTTER (Finetune CLIP)	26% ± 4.0%	15% ± 3.9%
OTTER w.o. f_e	40% ± 4.0%	29% ± 4.3%
OTTER w.o. f_l'	57% ± 4.4%	53% ± 4.6%
OTTER (Ours)	68% ± 4.3%	62% ± 4.2%
OTTER-OXE (Ours)	**72% ± 3.9%**	**73% ± 2.8%**

Table 5.1: Physical Single Primitive Multi-task Experiments. For each model, we conduct physical robot pick and place experiments, with 100 trials on in-distribution training tasks and 70 trials on unseen tasks. OTTER achieves a similar success rate on the in-distribution training tasks and unseen tasks, significantly outperforming the baselines, highlighting the benefits of extracting text-aware visual features and a frozen pre-trained VLM.

Real-world Experiments

Single Primitive We first evaluate all models on both the training and unseen tasks of the pick and place primitive in the real robot environment, as shown in Table 5.1. For fair comparisons, we fine-tune Octo and OpenVLA on DS-PnP using the same amount of learning steps. We compare the performance of OTTER trained from scratch and OTTER-OXE pre-trained on the OXE dataset and fine-tuned on DS-PnP. For π_0-Fast-Droid model, since our experiment setup is the Droid setup, we directly evaluate this model on our setup, denoted as π_0-Fast-Droid. More details about model training and architectures are in Appendix C.2.

For unseen pick up and place tasks, π_0-Fast-Droid is able to achieve a success rate of 61%. This is because this model is fine-tuned on Droid, which contains many pick up and place tasks. In both the training and unseen tasks, Octo struggles to accurately identify the object of interest and determine the correct placement location, leading to a low success rate. We hypothesize this can be attributed to two key factors. First, Octo does not incorporate a pre-trained VLM, such as CLIP, into its network. Instead, it trains its vision encoder from scratch using a large-scale robotic dataset (OXE [49]), which lacks the semantic diversity found in larger vision datasets like LAION [184]. Second, OTTER extracts text-aware visual features from the pre-trained CLIP, which results in a strong vision language association. This enables better visual grounding and generalization capabilities of OTTER to perform better on training and unseen tasks, despite being trained on a small robotic dataset. OpenVLA and DFP-OTTER perform similarly, which is better than Octo on training tasks, but much worse than OTTER. On unseen tasks, they both fail to generalize. We hypothesize this is because it's challenging for the direct feature passing architectures to learn generalizable vision-language connections on a small robotic dataset, while OTTER can utilize the extracted

Method	Pouring	Drawer	Poking	Pick and Place	Mean±Std. Err.
π_0-Fast-Droid	0%	0%	0%	61%	29% ± 3.5%
Finetuned π_0-Fast-Droid	0%	45%	27%	51%	35% ± 3.8%
Finetuned Octo	0%	0%	0%	5%	4% ± 1.2%
Finetuned OpenVLA	0%	0%	0%	1%	0.6% ± 0.5%
OTTER	63%	50%	**93%**	61%	67% ± 3.8%
OTTER-L	65%	55%	90%	69%	71% ± 3.5%
OTTER-OXE	60%	65%	**93%**	66%	70% ± 3.6%
OTTER-OXE-L	**77%**	**75%**	**93%**	**75%**	**77% ± 3.3%**

Table 5.2: Multi-primitive zero-shot generalization: We train models across four manipulation primitives (pouring, drawer manipulation, poking, and pick-and-place) with a total of 1,185 human tele-operated demonstration trajectories and evaluate them on 150 trials of **completely unseen tasks** within these primitives. Despite the inherent difficulty of zero-shot generalization across multiple primitives, OTTER achieves significantly higher success rates compared to baselines across all primitives. While baseline models struggle with generalization, particularly on pouring task (0% success rate), OTTER maintains substantial performance (60-93% success rate) on unseen tasks. The results further suggest that OTTER's generalization capabilities can be enhanced through increased model capacity (OTTER-L) and pre-training on large robotic datasets (OTTER-OXE).

text-aware vision features from the pre-trained VLM. OTTER-OXE performs better than OTTER on both training and unseen tasks, suggesting that OTTER's performance scales with more data, possibly because the policy network can learn better and more precise action planning from more robotics data.

Multiple Primitives We evaluate all models on all four primitives. We compare the performance of Octo and OpenVLA finetuned on DS-ALL and OTTER pretrained on OXE and fine-tuned on DS-ALL, denoted as OTTER-OXE. As there are more primitives, we also consider a deeper and wider OTTER model (details in Table C.3), denoted as OTTER-L. For a fair comparison, we extended the context history length of Octo to 10 (Octo cannot exceed a context length of 10 due to its inherent design constraints) and matched its action prediction horizon to ours. As OpenVLA has many tokens per timestep, its context length cannot be extended and we use its default context length. For π_0-Fast-Droid model, as in the single primitive experiment, we directly evaluate this model on our setup. As most of the Droid tasks are pick and place while our setup contains other primitives, we also consider fine-tuning π_0-Fast-Droid on the DS-ALL, denoted as Finetuned π_0-Fast-Droid.

Results are shown in Table 5.2. All models are evaluated on **unseen** tasks for each primitive, with 10 trials for each task. While π_0-Fast-Droid achieves decent performance on the pick and place primitives, it fails on all the other three primitives as the majority of the Droid dataset is the pick and place primitive. Finetuned π_0-Fast-Droid achieves non-zero success rate on Drawer and Poking primitives, but still fails on the pouring primitive. There

Method	LIBERO-Spatial	LIBERO-Object	LIBERO-Goal	Train (Average)	Unseen
Finetuned Octo	79% ± 1.0%*	86% ± 0.9%*	**85% ± 0.9%***	83% ± 1.0%	26% ± 1.1%
Finetuned OpenVLA	**85% ± 0.9%***	88% ± 0.8%*	79% ± 1.0%*	84% ± 0.9%	48% ± 1.0%
DFP-OTTER	79% ± 0.9%	80% ± 1.0%	78% ± 1.1%	79% ± 1.0%	45% ± 1.0%
OTTER (Ours)	84% ± 1.0%	**89% ± 1.2%**	79% ± 0.9%	84% ± 1.1%	**61% ± 1.1%**

Table 5.3: Simulation results on LIBERO. We evaluate OTTER and other baselines on 30 in-distribution tasks in LIBERO-Spatial/Object/Goal and on 10 unseen tasks we constructed, each task 50 trials. The numbers marked with $*$ of are directly referred from the OpenVLA [151] paper. The detailed training and evaluation on the unseen tasks are described in Section 5.4.

is also a degraded performance on the Pick and Place primitives for Finetuned π_0-Fast-Droid. This highlights the difficulty of our experiment setup. The performance of Octo, OpenVLA, and OTTER-OXE on the pick and place task all drop, showing the difficulty of multi-primitive learning. Notably, both Octo and OpenVLA fail to complete any unseen tasks for the pouring, drawer, and poking tasks, likely due to a relatively small amount of demonstrations provided for each primitive. Both OTTER-OXE and OTTER-OXE-L can achieve high success rate on all four primitives on the same amount of demonstrations, indicating that using text-aware visual features extracted from a pre-trained VLM can increase the data efficiency and enhance the generalization ability. OTTER-L outperforms OTTER on average, with the performance gap increasing as OTTER is pre-trained on the OXE dataset, indicating that OTTER can scale with model size.

Simulation Experiments

We compare OTTER with various baselines in simulation, as shown in Table 5.3. For fair comparison with OpenVLA and Octo, we process the simulation datasets following the procedure described in the OpenVLA paper [151], and use their reported performance on the standard LIBERO tasks. For the unseen tasks, we change 10 tasks in LIBERO-90 with different objects and distractors to test the generalizability of all the models on unseen tasks. From Table 5.3, we found all the models perform similarly on training tasks in LIBERO due to the limited variations of the tasks and sufficient demonstration datasets. For tasks in LIBERO-Object and LIBERO-Spatial, DFP-OTTER is worse than OTTER because OTTER does better in localizing the object to interact with. Note that Octo and OpenVLA are pre-trained on the OXE dataset but DFP-OTTER is trained from-scratch on LIBERO, so DFP-OTTER's performance is worse. In unseen tasks, OTTER can outperform other baselines by a large margin, demonstrating its generalization capabilities to novel scenarios, which is consistent with the conclusion of the real-world experiments.

Ablations

We perform ablation studies with single-primitive, multi-task setting, where all models are trained on DS-PnP. We consider the following ablations on the design choices of OTTER. Additional ablation studies can be found in Section C.4.

1. OTTER w.o. f_e: OTTER without the embodiment representation f_e. The concatenated text token f_l' and fused vision-language token f_{lv}' are passed as the input to the transformer.

2. OTTER w.o. f_l': OTTER without the text token f_l'. Only f_{lv}' and f_e are concatenated as the input to the transformer.

3. OTTER w.o. CLIP vision: OTTER using a smaller ViT to train from scratch instead of a frozen pre-trained CLIP vision encoder.

4. OTTER (Finetune CLIP): OTTER with the CLIP initialized from the pre-trained weight and fine-tuned end to end on the robotic dataset.

Real-world Results Physical results in Table 5.1 suggest that the performance on both the training tasks and the unseen tasks drop significantly for the ablations compared to OTTER. The performance of OTTER without the embodiment features (f_e) drops 28% on the training tasks and 33% on the unseen tasks, indicating that f_e is vital for task completion and generalization, likely because it provides a physical grounding for decision-making. Without f_e, the model's understanding of embodied features, possibly linked to the spatial or physical aspects of the task, is severely impaired. OTTER without the language features (f_l') experiences a performance drop of around 10% on both training and unseen tasks, suggesting that f_l' provides complementary information that may help in more nuanced task understanding.

OTTER w.o. CLIP vision has a significant performance drop of more than 50% on the training and unseen tasks in physical experiments. The results of OTTER w.o. CLIP vision is similar to Octo which also trains a vision encoder from scratch on the robotics dataset. This suggests that pre-trained VLM provides more robust and transferable visual representations. Training a vision encoder from scratch can result in poor performance, as it lacks the generalization capabilities learned from large-scale pre-training.

OpenVLA demonstrates that fine-tuning the vision encoder of the pre-trained VLM on the robotics dataset is crucial for improving the performance. However, we found that fine-tuning the pre-trained vision encoder actually degrades the model's vision-language understanding capabilities. The large performance discrepancy between training and unseen tasks in both OpenVLA and OTTER (Finetune CLIP) suggests that fine-tuning compromises generalization ability, highlighting the benefits of OTTER's approach of extracting text-aware visual features from a frozen pre-trained VLM. We want to emphasize that both the effective feature extraction and the frozen encoder are crucial for learning a generalizable VLA, as shown by the worse performance of DFP-OTTER with a frozen VLM, OTTER (Finetune

Method	LIBERO-Object	Unseen
OTTER w.o. CLIP vision	80% ± 0.7%	29% ± 0.9%
OTTER w.o. f_e	79% ± 0.9%	48% ± 0.8%
OTTER w.o. f_l'	71% ± 1.2%	49% ± 1.0%
OTTER (Ours)	**89% ± 1.2%**	**61% ± 1.1%**

Table 5.4: Ablation results on LIBERO Object tasks and unseen tasks. We evaluate OTTER and other baselines on 100 trials of in-distribution LIBERO-Object tasks, and 100 trials of unseen tasks.

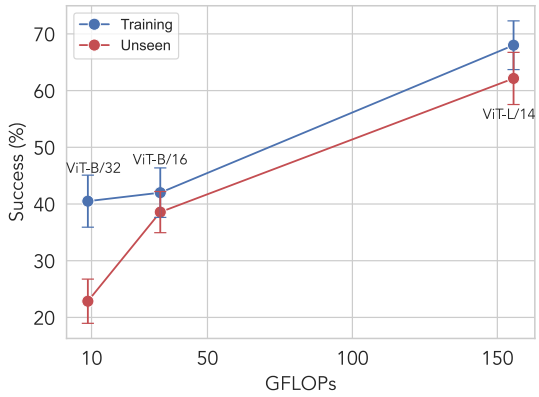


Figure 5.5: We evaluate OTTER's performance with improved vision language features by scaling CLIP. In particular, we train OTTER with three CLIP variants with increasing FLOPs: ViT-B/32, ViT-B/16, and ViT-L/16. We report the task performance vs. the inference FLOPs per image on training and unseen tasks. The results suggest that the OTTER can benefit from scaling up vision-language model.

CLIP) that has a fine-tuned VLM and OpenVLA that is a VLA with direct feature passing and fine-tuned VLM.

Simulation Results Table 5.4 presents the simulation results of OTTER and other ablations. On the in-distribution tasks, OTTER w.o. CLIP vision and OTTER can work similarly well given sufficient demonstrations, but OTTER w.o. CLIP vision is more than 50% worse on unseen tasks, which shows the benefits of using a pre-trained VLM for better generalization capabilities.

The performance of OTTER w.o. f_e drops about 10% on both the in-distribution and unseen tasks, indicating that f_e is beneficial for task completion as it provides explicit spatial information of the robot. OTTER w.o. f_l' is also noticeably worse. We hypothesize this is due to the objects not being very realistic in simulation, so the extracted text-aware visual features in CLIP may highlight multiple objects or wrong objects. f_l' can provide complementary information for the policy to interact with the correct objects.

Scaling up Vision and Language Encoders

Prior work has shown that vision-language encoders exhibit improved semantic understanding capabilities as their computational complexity increases [185]. To investigate whether OTTER can leverage these improvements, we evaluated its performance using three CLIP model variants of increasing computational complexity: ViT-B/32, ViT-B/16, and ViT-L/14. As shown in Figure 5.5, OTTER's task success rate improves substantially as we scale up the CLIP model's FLOPs, with gains observed on both training (+27.5%) and unseen pick-and-place tasks (+39.3%) when moving from ViT-B/32 to ViT-L/14. These results demonstrate that OTTER effectively utilizes larger vision-language encoders for enhanced semantic understanding and serves as a scalable method for integrating large-scale vision-language pre-training with learning from robotic datasets.

Scaling up Pre-training Dataset with Human Videos

Dataset Prior work like Octo and OpenVLA has studied pre-training on large robotic datasets like Open X-Embodiment dataset. To leverage a broader source of data such as human videos, we construct a human dataset consisting of the DexYCB dataset [208] and the HOI4D dataset [209]. Both datasets provide labeled human hand poses in the camera frame. We post-processed the datasets to extract human hand motion trajectories. In total, we obtain 15.6k human hand motion trajectories consisting of RGB image frames, language instructions and human hand actions from the DexYCB dataset and the HOI4D dataset.

Results We first pre-train OTTER using human videos, then finetune it using our robot dataset. We evaluate the benefits of pre-training on this human video dataset. Results are shown in Table 5.5. The human dataset has different camera angles and no wrist view image and the proprioception of the human dataset is also in various coordinate frames. Despite the significant discrepancy between the human and robot dataset, we found naively pre-training OTTER on human dataset shows improved performance on the unseen tasks, indicating that pre-training on human videos can improve the generalization ability of the model. More details and discussions about the human dataset and training regimes can be found in Appendix C.3.

Method	Training Tasks	Unseen Tasks
OTTER w.o. Human Pre-training	**68% ± 4.3%**	62% ± 4.2%
OTTER w. Human Pre-training	67% ± 4.0%	**70% ± 4.5%**

Table 5.5: Physical results of OTTER with and without human pre-training evaluated on 100 trials for in-distribution training tasks and 70 trials for unseen tasks. Pre-training OTTER on human dataset improves the performance on unseen tasks.

5.6 Limitations and Conclusions

While OTTER demonstrates improved task completion rates compared to existing VLAs, it still faces several limitations. One significant challenge is scaling across different morphologies, particularly those that cannot be easily parameterized by SE(3) transforms (i.e. robot multi-finger hand). This limitation restricts the model's adaptability to a wider range of robotic platforms and task types. Furthermore, this study has not extensively explored how this method scales to long-horizon tasks and more complex scenes, which could be an important area for future research.

In summary, we present OTTER, a vision-language-action model that leverages text-aware visual feature extraction from pre-trained vision-language encoders. By utilizing the semantic alignments during large-scale vision-language pre-training, OTTER achieves significantly better generalization than existing VLA models across robot manipulation tasks sampled from multiple motion primitives, maintaining higher success rates on *unseen objects and environments* where previous methods fail. The experiments demonstrate that OTTER's performance scales along multiple axes: through larger pre-trained vision-language encoders, increased policy network capacity, and pre-training on larger robot datasets. These results suggest that better preserving the existing alignment in pre-trained vision-language encoders, rather than learning it directly from robotics data, is beneficial for developing more capable and generalizable robot learning systems.

Part II

Scaling Robot Data Without Teleoperation

Chapter 6

Closing the Robot Data Gap

6.1 The Robot Data Gap

Various experimental results in Part I suggest that the scale and diversity of imitation-learning data is critical for training better robot policies. Indeed, modern vision and language foundation models would not exist without large and diverse datasets [184, 210], which dwarf available robot data by several orders of magnitude. The largest public robot demonstration datasets [48, 64] are roughly $10^5\times$ smaller than the proprietary text and image corpora used to train frontier LLMs and VLMs [97], and even the largest closed industrial datasets remain far short of the regime that pre-training for vision and language requires [211]. If closing the gap is necessary for building physical intelligence, where can the data come from?

6.2 Sources of Robot Data to Close the Gap

Five sources are visible on the horizon, each with a different cost–realism trade-off:

1. **Scripted policies and self-supervised on-hardware collection.** Hand-written controllers and motion-planning algorithms [212–214], optionally paired with reset primitives, can autonomously generate large quantities of robot object interaction. The largest precedents are arm-farm grasping [215, 216] and RL-driven on-policy collection [133, 217], complemented by synthetic-data grasp pipelines such as the Dex-Net family [218–220], and by self-improving generalist agents that close the loop between data collection and policy improvement [221]. The data is real and on-distribution, but the bottleneck is hand-engineering both the controller and the safety envelope around it. Chapter 7 pursues this category for industrial insertion.

2. **Human teleoperation.** Leader–follower teleoperation systems [222–224] produce on-embodiment demonstrations and have driven the largest open robot datasets [48, 64, 96, 156–158] as well as the closed industrial corpora behind recent generalist VLAs [51–53, 225]. However, human teleoperation is limiting: the data collection process is

tied closely to the particular robot that data collection is performed on. However, as new robots are rolling out every year, bringing up a new robot will almost definitely require additional teleoperation data collection on the new robot. The cost is linear in operator hours, robots, and embodiments; coverage of the long tail of objects, scenes, and primitives is at the discretion of the operator and the resources that are available.

3. **Universal Manipulation Interfaces.** Off-embodiment data collection pioneered by UMI [226–229] frees data collection from the fixed robot embodiment, as the devices are directly worn on human hands. This makes transfer between different embodiments possible. Additionally, since UMI usually depends on visual-SLAM to provide the state, it lowers the barrier for collecting data from new environments without having any robot involved. The cost is still linear in operator hours, yet the scalability is much improved compared to human teleoperation.

4. **Human videos and egocentric demonstrations.** Internet-scale and curated egocentric corpora [128, 208, 209, 230] record people performing manipulation tasks from a first-person view. The data is real, abundant, and embodiment-agnostic, but conversion is the central challenge: a robot needs labels in an action space it can execute, not raw RGB, and a human hand and a parallel-jaw gripper share neither morphology nor kinematics. To address this gap, first, frozen visual representations pre-trained on egocentric and instructional video [41–43] transfer well as encoders for downstream policies but do not themselves produce action labels. Subsequently, parametric hand models [231] and recent hand-pose estimators [232] feed retargeting pipelines that map human wrist or fingertip trajectories onto robot end-effectors [233–237]. Then, recent VLA models pre-train directly on action-labeled human video and transfer to robot platforms [119, 122, 225, 238]. While hand pose estimation from RGB images is still much noisier than robot proprioceptive states, human egocentric video data is still much more efficient and scalable to collect.

5. **Generative world models.** Frontier video-generation systems [239–241] and robotic world models [140, 242–248] can synthesize plausible visual rollouts from internet-scale video pre-training. While once the model is trained, the data collection cost is minimal, there are still a few challenges that various world models are currently facing. For example, most video models are trained on causal videos, which do not contain any action labels. Thus, either a large amount of post-training data from the physical robot is required to make it action conditioned, or an inverse dynamics model is required to generate the action labels [249].

6. **Simulation.** Domain-randomized simulation [67, 250–252] is the dominant data source for legged locomotion and whole-body control, where the relevant physics (contact, friction, dynamics) is simulated with sufficient fidelity to transfer zero-shot [68–70, 253, 254]. Manipulation has lagged behind: contact-rich, deformable, and tool-using interactions remain stubbornly hard to simulate. A growing ecosystem of simulators

and benchmarks [67, 252, 255–261] and synthetic-manipulation pipelines [218, 262–270] explores the design space between full physics simulation and rendering-only approaches. Chapter 8 pursues this category by dropping dynamics in favor of high-fidelity rendering of scanned objects.

In this Part of the dissertation, we will focus on works that can scale up robot data without human teleoperation throughout the pipeline. The two chapters in this Part work along this taxonomy, taking up category (1) on real hardware (Chapter 7) and category (5) for manipulation (Chapter 8). Category (3) is not the focus of any chapter here, but recent results suggest it scales predictably; we refer the reader to sections in OTTER (Section 5.5) and EgoScale [238] for empirical evidence and to the citations above for the broader literature.

6.3 Roadmap for Part II

- **Chapter 7 (Visuo-Tactile Insertion).** A safe self-supervised pipeline that uses force–torque and tactile feedback to identify reversible grasp poses, enabling autonomous data collection on a real robot for industrial insertion. The result is a learned visuo-tactile policy that requires only a single human demonstration as a target.
- **Chapter 8 (Real2Render2Real).** A pipeline that takes one human demonstration plus an object scan, renders thousands of randomized synthetic trajectories without any robot hardware, and fine-tunes a vision–language–action model on the synthetic data alone. Performance scales with generated data and matches teleop-trained baselines at a fraction of the time and operator effort.

Chapter 7

Safe Self-Supervised Visuo-Tactile Feedback Policies for Industrial Insertion

7.1 Introduction

Industrial assembly [271] is a precise manipulation task requiring contact between parts. Part feeding, peg insertion and object reorientation (three sub-tasks of industrial assembly) have been extensively studied [271–275]. Early work considers the mechanical design aspect [273, 275] and motion planning aspect [272, 274, 276]. Through Computer Aided Design (CAD), the order of part assembly can be predetermined in simulation with precise pose information [277], allowing robots to plan the necessary actions to assemble the design [278]. Learning-based approaches recently have shown promise on industrial insertion tasks [279] on the NIST taskboard [280], a standard benchmark that represents common industrial insertion tasks with parts that have complex geometries [281]. However, applying learning-based methods for industrial insertion remains challenging due to the requirement for frequent human inputs during learning [282] or high-precision sensors for collecting training data [279]. There is also a need for a safe training and data collection method for learning insertion tasks since parts are prone to breakage.

Another challenge in an industrial insertion task is that the precise grasp pose is often unknown due to variations in kitting and feeding of parts as they arrive for assembly. As the grasped part is often occluded by the gripper visually, grasp pose estimation is better achieved when using tactile sensing [283]. While recent work has shown improvement in simulation accuracy for industrial insertion [284] and successes in Sim2Real transfer for tactile-based insertion tasks [285, 286], the simulation of soft contacts between tactile sensors and objects with complex geometries remains an open problem [287], and often can not transfer to real because the object models are not publicly available. In this work, we present a novel method to safely learn visuo-tactile feedback policies in real for industrial insertion tasks under grasp

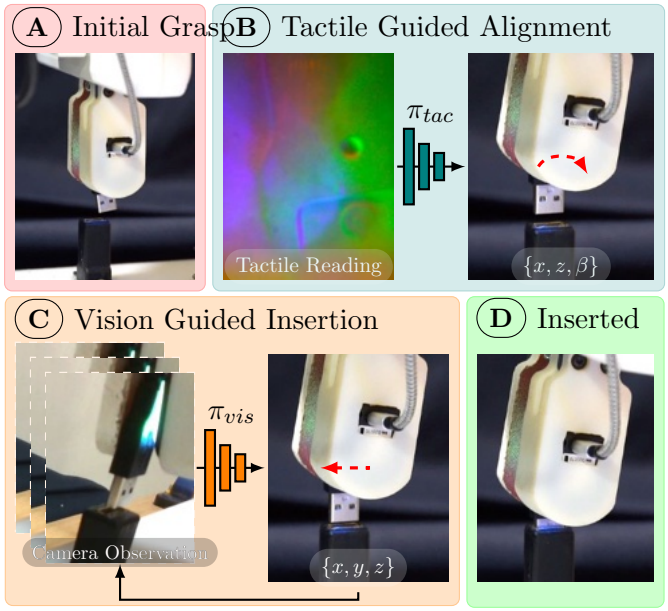


Figure 7.1: Overview of the learned two-phase insertion policy: the red arrows indicating the robot actions given by the policies. (A) The robot grasps the part at an initial pose. (B) The tactile guided policy π_{tac} estimates the grasp pose using the tactile image and aligns the z-axis of the part with the insertion axis. (C) A vision guided policy π_{vis} is used to insert the part. (D) The part is inserted successfully into the receptacle.

pose uncertainties, with inexpensive off-the-shelf sensors. Our approach draws on tactile and visual feedback to deal with the grasp pose uncertainty and force-torque sensing for a self-supervised training procedure that is *safe*, minimizing damage during the training phase. We divide the insertion task into two phases (as shown in Fig. 7.1):

- An initial *Align* phase where a tactile-based policy π_{tac} estimates the grasp pose. The robot reorients and aligns the part with the insertion axis of the receptacle.
- A second *Insert* phase where an RGB image-based policy π_{vis} guides the robot to insert the part.

A significant challenge in learning tactile feedback policies in real for industrial insertion is the frequent slippage of the part that occurs due to collision with the environment and the smooth surface of the tactile sensor gel pad. This makes RL methods difficult to succeed without human intervention or an automatic reset mechanism to detect and correct slippage. In this work, we develop a self-supervised data collection pipeline that avoids collision between the part and its environment, by recognizing that the insertion operation is reversible only from certain target insertion poses – i.e. starting from such poses, the part can be repeatedly unplugged from and inserted into the receptacle. Prior to data collection, a human free-drives the robot to provide one approximate target pose where the part is inserted. The robot

refines this target pose to find such a reversible pose by minimizing the grasping force-torque, which helps minimize collisions during data collection, resulting in a safer training process that is unlikely to damage the insertion part and receptacle.

This chapter presents:

1. A safe self-supervised data collection pipeline with force-torque sensing in real for insertion, designed to minimize contact force for data collection.

2. A two-phase policy learned from the collected data including a tactile-based alignment policy for orienting the part and an RGB image-based insertion policy;

3. Experimental results suggest that the policy achieves 45/45 successes on USB connector insertion, outperforming two baseline methods (1/45 and 0/45).

7.2 Related Work

Industrial insertion has been central in robotics for 50 years. It is challenging due to occlusions brought by the robot gripper, grasp uncertainty from the process of acquiring the part and its collision with the environment, the fragility of the parts, and the precision required in controlling the robot for insertion. Early work approached this problem using CAD information to infer desired assembly sequences [277] and generating designs of part feeders based on object geometry [275]. Other work approached the problem from an algorithmic design perspective, with a focus on developing motion planning strategies for peg insertion [273, 276].

Recently, learning-based methods have shown success on this task. This includes learning insertion policies with a physical robot via Sim2Real transfer [288], online adaptation with meta-learning [289, 290], reinforcement learning [282, 291], self-supervised data collection with impedance control [292], accurate state estimation [279], or decomposing the insertion algorithm into a residual policy that relies on conventional feedback control [288]. These approaches assume that the parts are grasped with a fixed pose. To overcome this assumption, Wen et al. [279] perform accurate pose estimation and motion tracking with a high-precision depth camera and use a behavioral cloning algorithm to insert the part. Spector and Castro [292] and Spector et al. [293] require contact between the part and the environment to occur during data collection, a process that is expensive and often impractical for fragile parts. In comparison, we use inexpensive tactile sensors and a safe self-supervised data collection procedure that does not require such contact.

Grasped parts are often visually occluded by the gripper. Tactile feedback can be an alternative sensing modality for grasp pose estimation. Recent work uses tactile images from vision-based tactile sensors such as GelSight [294] and DIGIT [295] to estimate the pose of grasped objects. Li et al. [296] use Gelsight sensors, BRISK features and RANSAC to estimate grasp pose. Gelsight produces high-quality 3D tactile images and can determine depth imprint, which improves feature detection by isolating the object from the background.

DIGIT, a more affordable tactile sensor, provides a 2D RGB image but not the light incident direction (to generate the depth image). Kelestemur et al. [285] generates tactile image data in simulation for pose estimation of bottle caps but simulating contact and physical interaction between tactile sensors and objects with more intricate geometry is still challenging [287]. In this work, we collect a dataset of tactile images in real for the USB connector with different grasp poses to train a tactile based policy for grasp pose estimation.

Most prior work on tight tolerance insertion tasks [279, 296–298] leverages a single modality, such as vision, tactile, or force-torque, limiting the accuracy of the system due to occlusion, perspective effect, and sensory inaccuracy. Multi-modal systems have been explored to improve the robustness of automated insertion. Spector and Castro [292] and Spector et al. [293] use RGB cameras and a force-torque sensor for learning contact and impedance control. Chaudhury et al. [299] couple vision and tactile data to perform localization and pose estimation, and demonstrate that vision helps with disambiguating tactile signals for objects without distinctive features. Ichiwara et al. [300] leverage tactile and vision for deformable bag manipulation by performing auto-regressive prediction. Hansen et al. [301] use a contact-gated tactile, vision and proprioceptive observation to train reinforcement learning policies. Okumura et al. [283] also tackle the problem of grasp pose uncertainty for insertion by using Newtonian Variational Autoencoders to combine camera observations and tactile images. They demonstrate results for USB insertion accounting for grasp pose uncertainty in one translation direction. In this work, we separate the insertion problem into an alignment phase and an insertion phase, decoupling vision and tactile inputs and also present a novel safe self-supervised approach to data collection. We are able to handle both grasp pose rotation and translation uncertainty for the USB insertion task.

7.3 Problem Statement

Overview: We consider a part insertion task using a 7-DoF robot, equipped with a parallel-jaw gripper with a tactile sensor mounted on one jaw. The end-effector has a wrist-mounted RGB camera, and the robot provides reliable force-torque readings at the end-effector. The objective is to learn a policy that can robustly insert the part into the receptacle with an unknown part's pose within the gripper, while minimizing human inputs and part-receptacle collisions during training. Fig. 7.2 shows the experiment setup and the coordinate frames.

Assumptions: We make the following assumptions:

1. A human provides one top-down grasp pose of the part inserted in the receptacle.
2. The robot can accurately measure force and torque either with an external sensor or internal current sensing;
3. An experiment begins with the part pre-grasped by the robot gripper with the part grasp pose within a range relative to the human-provided pose.

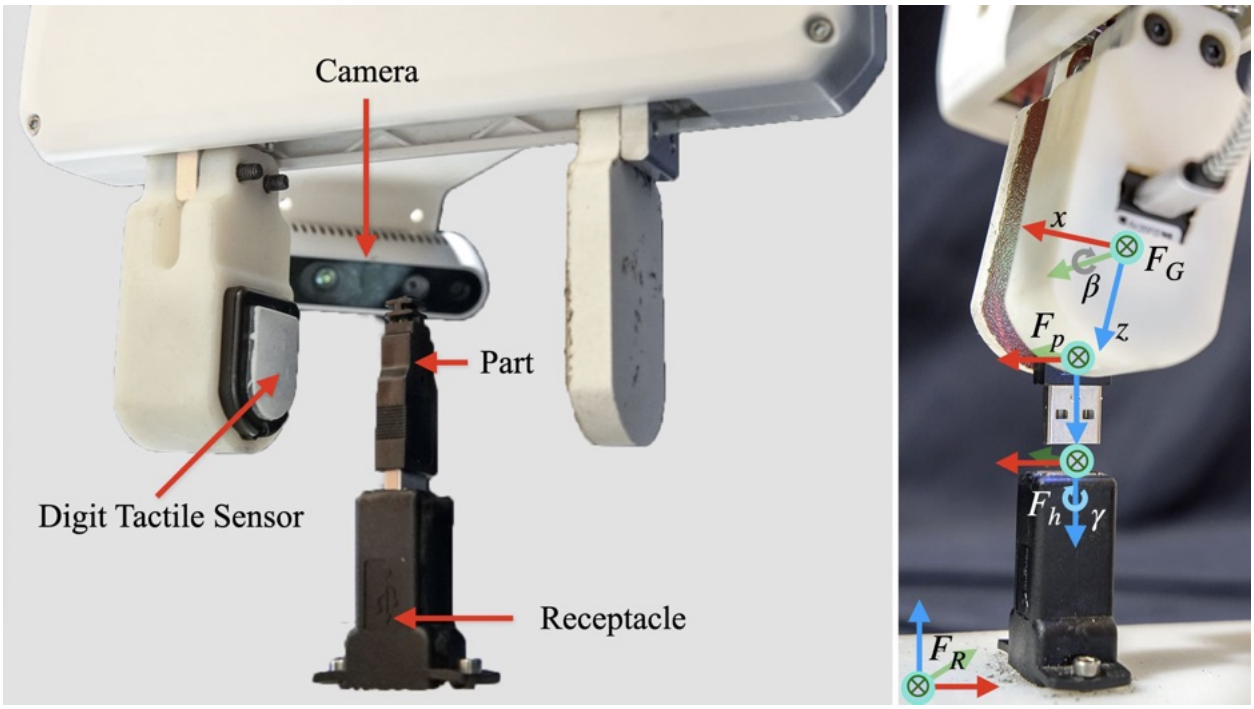


Figure 7.2: Experiment setup and coordinate system. The x, y, z axes are labeled by red, green, blue respectively. We label the gripper frame, part frame, human-provided target pose frame, and robot frame as F_G, F_p, F_h, F_R respectively. The insertion direction is defined as the z-axis of F_h. When the part is inserted, $F_h = F_p$.

4. During data collection, the robot operates in a rectangular collision-free configuration space above the receptacle.

Problem Setup: Given a tight-fitting receptacle for the part to be inserted into (Fig. 7.2), we find the target insertion pose $T_{R,h}$ of the part (grasped at an unknown pose) from one human-provided imperfect demonstration. At any time step t, we have access to the RGB observation $o_\rho(t)$ from the wrist mounted camera, the RGB tactile image $o_\psi(t)$ from the DIGIT tactile sensor, and reliable force $\vec{f}(t)$ and torque $\tau(t)$ readings from the robot. Since we know the insertion axis, we parameterize the action space with 4 degrees of freedom: gripper translation in robot frame and gripper y-axis rotation.

7.4 Method

Hardware

We design a novel parallel jaw gripper mount to accommodate the DIGIT tactile sensor [295] and camera mount (Fig. 7.2, 7.3). The elastomer gel on the DIGIT sensor deforms, causing

torque applied to the part. This torque and the force applied from the receptacle to the part during insertion often produce undesired slippage and rotation. Therefore, we develop an asymmetric mounting setup where we mount the DIGIT sensor on one jaw with reinforcement to prevent outward bending, while keeping the other jaw flat. We apply sandpaper on the surface of the non-tactile gripper, increasing the friction to reduce slippage. We find that we can predict the part's grasp pose of a USB connector using a single DIGIT sensor.

Self-Supervised Data Collection

One Human Provided Imperfect Target Pose

The target pose $T_{R,h}$ is provided by a human free-driving the robot with a pre-grasped part to insert it in the receptacle from top down. Since this target pose may not have a perfect axis alignment with the receptacle, the system performs a z-axis alignment of the target pose. To account for the change in grasp center after axis alignment, we refine $T_{R,h}$ by finding a target pose that minimizes gripper force-torque using a grid search through a set of translations and rotations $\{T_\Delta\}$. Formally, we find

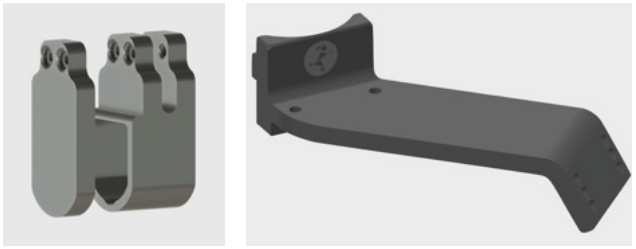
$$\tilde{T}_{R,h} = \underset{\tilde{T}_{R,h} \in \{T_\Delta \cdot T_{R,h}\}}{\arg\min} \left\| \vec{f}(\tilde{T}_{R,h}) \right\| + \left\| \tau(\tilde{T}_{R,h}) \right\|. \tag{7.1}$$

Here $\vec{f}(\tilde{T}_{R,h})$ and $\tau(\tilde{T}_{R,h})$ denote the 3-DoF force and torque vectors respectively when the gripper is at $\tilde{T}_{R,h}$. Intuitively, this objective minimizes the external force applied on the part when being unplugged, increasing the likelihood of the insertion process being reversible. Practically, we perform grid sampling over 5 values of $x \in$ [-1, 1]mm, 5 values of $y \in$ [-1, 1]mm and 4 values of $\gamma \in [-\frac{\pi}{180}, \frac{\pi}{180}]$rad ($x, y, \gamma$ are in F_h, refer to Fig. 7.2). The pose with minimum gripper external force-torque is recorded as the refined demonstration pose $\tilde{T}_{R,h}$, and we have $F_h = F_G$ at $\tilde{T}_{R,h}$.

A cascaded impedance controller, implemented within the robot's real-time control loop, allows fine-grained force control. In case of a force violation, our system calculates a trajectory to a safe state within a single control cycle. After refinement of the target pose, we search for the minimum offset $z_{\min}$ for the part to be unplugged from the receptacle. Finding the minimum height helps to determine the boundary for data collection and allows the pipeline to collect more data closer to the receptacle while reducing collisions. Iteratively, the robot moves the gripper by $-\Delta_z$ in F_G. We then move the gripper by Δ_x (in the F_G frame) and measure $\vec{f_x}$ (x-component of the gripper force in the F_G frame). If $\vec{f_x} \leq \eta$, we register the total upward distance traveled as the minimum height $z_{\min}$ for removal of the part. Empirically, we find setting $\Delta_x = \Delta_z =$1 mm, and $\eta = 3.5$ N works well.

Data Collection for Alignment

The part remains inserted throughout data collection. We explore grasp pose variations in 3-DoF (x, z translation and y-axis rotation β) in F_G (Fig. 7.2). We perform uniform random



(a) Parallel Jaw Gripper (b) Camera Mount

Figure 7.3: CAD models for the parallel jaw grippers and camera mount.

sampling over the range $[-3, 3]$mm, $[-8, -2]$mm, $[-\frac{\pi}{15}, \frac{\pi}{15}]$rad for x, z, β, with 5, 10 and 20 samples respectively. The robot closes the gripper with a force of $70N$ at each of the sampled poses and records the pair of tactile image readings and x, z, β. To account for the noise in the DIGIT tactile sensor, we take a median filter over 5 consecutively captured tactile images. We collect 2000 data points in 120 minutes.

Data Collection for Insertion

Upon completing the tactile image collection phase, we collect robot poses and RGB images for training the insertion policy for different grasps. We perform grid sampling with 5 samples each of x, z, β in F_G within the same range as in the previous stage, resulting in a total of 125 grasps. To account for the difference between the sampled grasp g and $\tilde{T}_{R,h}$, we perform minimum force-torque refinement on the sampled grasp to calculate the grasp-specific target pose $\tilde{T}_{R,h}(g)$. $\tilde{T}_{R,h}(g)$ translated by an offset of $z_{\min}$ gives us the unplugged part pose $T_{\text{unplug}}(g)$.

For each grasp g, we collect image data for the visuoservo policy by moving the gripper to sampled points on a grid above the target pose. In particular, we uniformly sample 5 values each of $x \in [-5, 5]$mm, $y \in [-5, 5]$mm and $z \in [-5, 0]$mm in F_R with respect to $T_{\text{unplug}}(g)$, resulting in 125 different translations for the gripper. For each translation, we collect *one* data point that does not contain additional rotation and sample *two* gripper y-axis rotation conditioned on z to avoid collision. Specifically, given a height z, the two rotations are sampled from the uniform distribution $\frac{z}{5} \cdot \mathcal{U}[-\frac{\pi}{15}, 0]$rad and $\frac{z}{5} \cdot \mathcal{U}[0, \frac{\pi}{15}]$rad. These rotated data points provide the system with additional data for camera pose variation with respect to the target pose, which leads to a balanced dataset with 375 distinct gripper poses. Each data point is composed of the gripper pose (translated and rotated away from the target pose) and the corresponding RGB image observation at that pose. Upon visiting all 375 gripper poses for a given grasp, the robot moves to $T_{\text{unplug}}(g)$, performs a vertical movement to $\tilde{T}_{R,h}(g)$, and opens the gripper jaws, thereby resetting the part in the receptacle. The system repeats this data collection process for all 125 grasps without human supervision.

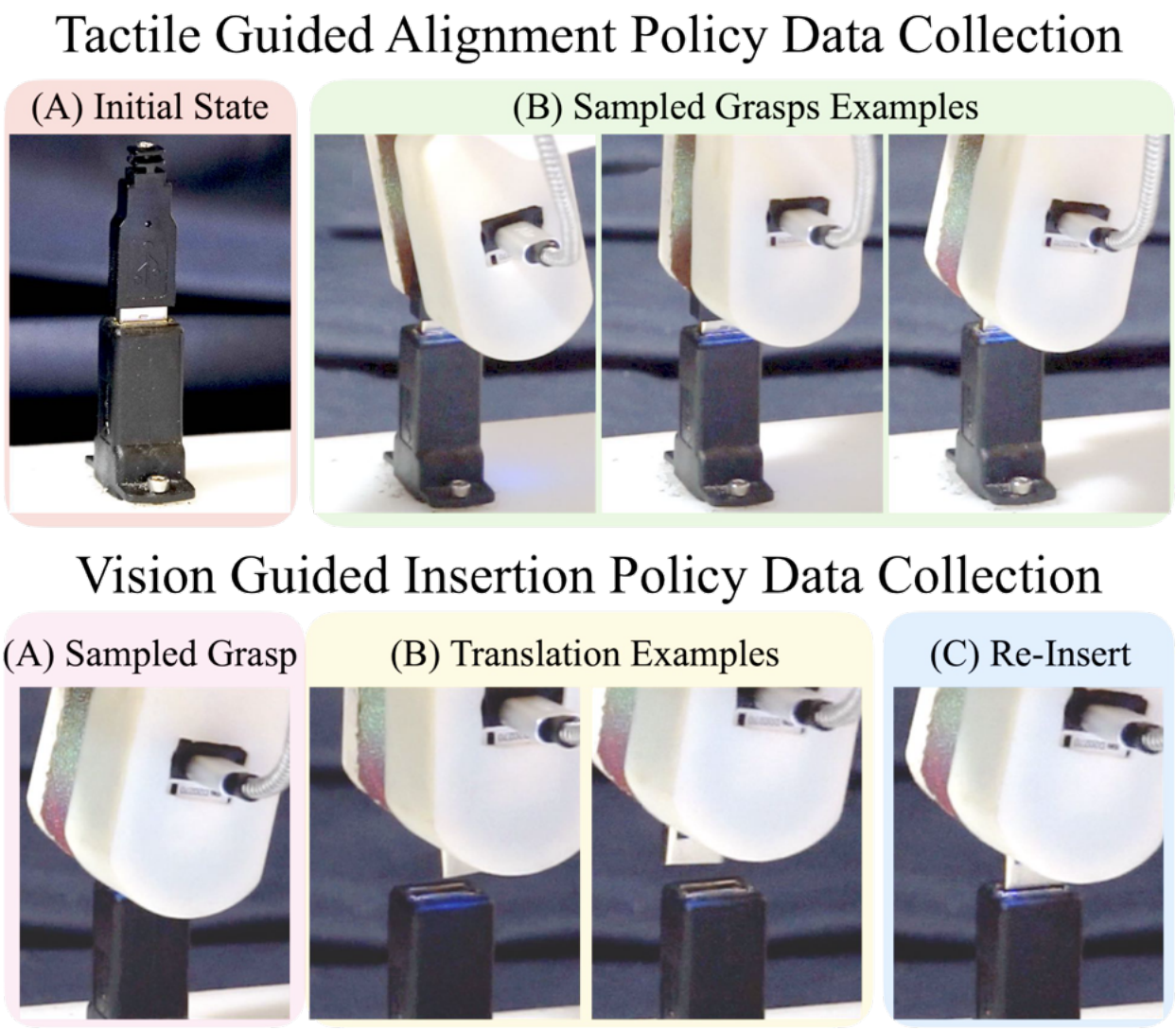


Figure 7.4: Data collection for alignment (Top) and insertion policies (Bottom). Data collection for the **Alignment policy** starts with the part inserted into the receptacle (Top (A)). The robot then samples and records different grasp poses and the corresponding tactile images (Top (B)). Data collection for the **Insertion policy** starts with a sampled refined grasp (Bottom (A)) and unplugs the part to apply sampled transformations (Bottom (B)). Then the robot inserts the part (Bottom (C)) and starts the next round of data collection with a different grasp pose.

Learning to Insert

While human demonstrations usually serve as "expert policies" for industrial insertion tasks, the self-supervised data collection pipeline allows us to collect ground truth actions at scale. This allows us to formulate two supervised-learning problems based on Sec. 7.4 and Sec. 7.4.

Alignment Policy

We use the data collected from Sec. 7.4 to train an alignment policy π_{tac} that, given the tactile image, outputs the desired displacement of the gripper $T_{p,G}$ to align the part with the receptacle (Fig. 7.1.B). We augment the tactile images by randomly jittering the brightness and contrast over the range $\mathcal{U}[0.7, 1.3]$.

Insertion Policy

We use data collected from Sec. 7.4 to train a visuoservo insertion policy π_{vis} taking normalized camera observation, gripper y-axis rotation β and x, y translations in F_h as inputs, and predict the action: desired translation Δ_x, Δ_y and rotation Δ_β in F_R (Fig. 7.1.C). We augment camera observations by randomly jittering the brightness and contrast over the range $\mathcal{U}[0.7, 1.3]$.

We use RegNet 3.2GF [302] as the backbone for both policies. For the alignment policy, we replace the last layer of RegNet with a linear layer with 3 outputs. For the insertion policy, we concatenate the robot's pose with the latent vector of the image and replace the last layer with a linear layer with 3 outputs. For both networks, we use a batch size of 64, a learning rate of 1e-3, and a learning rate decay of 0.99 for every 100 gradient steps. We pick the mean squared error as the loss function and use the Adam optimizer [303].

Execution of Insertion

To avoid catastrophic failure (i.e. collision between the part and the surrounding environment or moving out of the training set distribution), we deliberately formulate the visuoservo policy to only control x, y-axis translation but not z-axis translation since the insertion direction is already aligned with the z-axis by the *Align* policy. The formulation is detailed in Algorithm 1. The execution procedure starts by inferring the grasp pose from the tactile image via π_{tac} (line 2). The system then performs the insertion axis alignment of the part with the receptacle (line 3). We measure the z-direction force based on the gripper force-torque sensor (line 12). We then calculate rotation and translation based on the camera observation via π_{vis} (line 5, 10) and execute the corresponding actions (line 8) until the action has a norm smaller than ϵ (line 7). Note that the rotation prediction from the *Insertion Policy* is not used, because the part is aligned with the insertion axis by the *Alignment Policy*. When the action converges, we lower the gripper in the z direction by a step size of d_z (line 11) and continue to query π_{vis} until the force constraint is satisfied or the number of attempts exceeds the horizon H (line 13-14). We empirically set $d_z = 1.5$mm. Empirically, we find setting the action norm $\epsilon = 0.0005$, $d_z = 1.5$mm and $H = 200$ works well, as the force-based termination condition is usually triggered first for a successful or unsuccessful insertion.

Algorithm 1 Policy Execution Procedure

Require: Tactile image $o_\rho(t)$, camera image $o_\psi(t)$, tactile-based grasp pose estimation network π_{tac}, visuoservo insertion policy π_{vis}, target pose $\tilde{T}_{R,h}$, minimum wrench height $z_{\min}$, action norm threshold ϵ, z-direction step size d_z, attempt horizon H.

$T_{p,G} = \pi_{tac}(o_\rho(t))$
Move gripper to $\tilde{T}_{R,h} T_{p,G} + 2z_{\min}$
attempts $\leftarrow 0$
$(\Delta_x, \Delta_y, \Delta_\beta) = \pi_{vis}(o_\psi(t), T_{R,G}(t))$
while true do
  while $\|[\Delta_x, \Delta_y]\| > \epsilon$ **and** attempts $< H$ **do**
    Move gripper by (Δ_x, Δ_y)
    attempts $\leftarrow$ attempts $+1$
    $(\Delta_x, \Delta_y, \Delta_\beta) = \pi_{vis}(o_\psi(t), T_{R,G}(t))$
  end while
  Translate gripper in insertion direction by d_z
  Measure gripper z-axis force as F_z
  if $F_z > 15\,\mathrm{N}$ **or** attempts $\geq H$ **then**
    terminate
  end if
end while

Algorithms	IL	TD3	Proposed Approach
Success/Total	1/45	0/45	45/45

Table 7.1: Results suggest that (1) the IL trained on 50 human demonstrations is insufficient for training an accurate part pose estimation model, and (2) frequent slippage and rotations of the USB caused by collisions with the receptacle lead to failure in training TD3. Our approach outperforms both baseline policies.

7.5 Experiments

Experiment Setup

We focus on the USB insertion task on the NIST taskboard. We use a 7-DoF Franka Emika robot with a parallel gripper, where one DIGIT tactile sensor is mounted on the inside of one of the fingers. An Intel RealSense camera is mounted offset from the gripper (Fig. 7.3). To control the Cartesian position of the Franka robot, a time-optimal trajectory respecting velocity, acceleration, and jerk constraints is applied to the policy's positional output [304]. We use grid sampling to obtain 5 values of β ranging from $[-\frac{\pi}{20}, \frac{\pi}{20}]$rad, 3 translations in x and z from the range $[-3, 3]$mm and $[-6, -2]$mm in F_h, resulting in a test set of 45 different grasp configurations that lie in the training distribution of the algorithms.

Experimental Procedure

At the beginning of each test experiment, the USB connector (part) is pre-grasped by the robot with a grasp pose selected from the test set and located at a position with a z-axis translation of $2z_{\min}$ relative to F_h. At this starting pose, the gripper is aligned vertically down while the USB connector is misaligned with the receptacle in both translation and rotation as in Fig. 7.1(A). The robot first executes the *Align* policy to estimate the part grasp pose and aligns it with the insertion direction of the receptacle as in Fig. 7.1(B). It then uses the *Insert* policy to visuoservo and inserts the part into the receptacle as in Fig. 7.1(C-D). The robot then resets to the next grasp by releasing the part, re-grasping it, raising it to a start pose as outlined above, and executing the *Align* and *Insert* policies for this new grasp. It steps through all the test grasp poses using the same procedure.

An experiment terminates if the gripper frame (F_G) force in the z direction $\vec{f}_z(t)$ exceeds 15 N. An experiment trial is considered successful if the gripper is within 5 mm of the target pose in $H = 200$ iterations, upon which we also visually inspect whether the insertion is successful. In this set of experiments, the refined human-provided target pose $\tilde{T}_{R,h}$ is provided as an input to the policies. In Sec. 7.5, we perform ablation studies on noisy target poses.

Comparison

The method is designed with two objectives: (1) minimizing human intervention so that ideally no human needs to be involved in data collection or training of the policy, and (2) minimizing collision among the robot, the part, and the environment. We compare our approach with two baseline learning methods described below with the same environment setup (using the same grippers and camera mount).

Twin Delayed Deep Deterministic Policy Gradient(TD3)

An off-policy, online reinforcement learning policy [305] that learns the end-to-end part insertion. This baseline satisfies objective (1) but violates objective (2) — i.e. it is incapable of avoiding collisions among the robot, the part, and the environment. We simplify the problem to a fixed, axis-aligned grasp pose and restrict the action space to translations only, so that the policy only has to learn insertion instead of both alignment and insertion. The learned policy runs with a frequency of 10 Hz. Since the policy outputs Cartesian position changes, we control the Cartesian velocity of the robot, which is equivalent to Cartesian position changes per time step. Due to the part's axis-alignment with the receptacle, the policy's input can be restricted to the (low-dimensional) robot pose, velocity, and the force-torque at frame F_G. We use the default TD3 implementation of Ray RLLib [306]. If the force applied on the gripper exceeds 15 N, the episode terminates and the robot resets to a safe starting pose.

	x (mm)	z (mm)	β (rad)
Mean Error	0.0897	0.1460	0.0056
Standard Deviation	0.0049	0.0662	0.0049

Table 7.2: Mean and standard deviation of the error in predicting part pose (x, z, β) by the tactile-based alignment policy on the test set of 45 grasps.

Imitation learning (IL)

Imitation learning from 50 human demonstrations of insertion trained with behavior cloning. Each human demonstration starts with a randomly sampled grasp pose as in Sec. 7.4. This baseline algorithm violates objective (1) but satisfies objective (2), where the human demonstrator selects actions that minimize collision between the part and the environment. It takes about 30 minutes to provide all these human demonstrations.

Results

The results are summarized in Table. 7.1. The imitation learning agent (IL) is only able to perform a single successful insertion out of the 45 grasp poses. Intuitively, 50 different grasp configurations from human demonstrations are not sufficient for training an accurate part pose estimation model; additionally, for different grasp poses, at the same gripper pose, two different human demonstrations may exist. The multi-modality in the distribution of target insertion poses contributes to the failure of the IL policy. Training the TD3 policy in the physical environment led to divergent results in all 5 training trials we attempted. In all cases, the part collides with the receptacle, leading to a drastic change in the grasp pose. This cannot be corrected directly since there is no reset procedure that can systematically recover the gripper to the original state without human supervision. Our approach succeeds for every single grasp pose tested. Empirically, we find that the part rotation and translation predicted from tactile images are fairly accurate (refer to Table. 7.2).

Ablation Studies

Effects of Leveraging Force-Torque Sensing in Data Collection

We compare the completion rate of the data collection process for insertion with or without grasp pose refinement. We consider the following three different methods for refining the target pose gained from the human demonstration $T_{R,h}$: 1) **ZA**: apply z-axis alignment on $T_{R,h}$, 2) **ZAWF**: perform minimum force-torque refinement only once for $T_{R,h}$ after z-axis alignment (this step is only performed for the first grasp and the results reused for all grasps) and 3) **ZAWFG**: perform z-axis alignment for $T_{R,h}$ and apply minimum force-torque refinement separately for each grasp.

Setup	Trial 1	Trial 2	Trial 3	Mean±Standard Error
ZA	0/125	0/125	0/125	0.0±0.0%
ZAWF	125/125	125/125	34/125	75.7±19.8%
ZAWFG	125/125	125/125	125/125	100.0±0.0%

Table 7.3: Comparing data collection success rate. We measure the number of successful insertions until failure for 125 different grasps configurations. We compare Human Demonstration with axis alignment (**ZA**), Single Minimum Force-Torque Refinement (**ZAWF**), and Minimum Force-Torque Refinement for all grasps (**ZAWFG**). We report the mean success rate and the standard error for three distinct human-provided target poses.

At the beginning of each experiment, a human provides $T_{R,h}$ by free-driving the robot with one pre-grasped part to insert the part. A total of 125 different grasp poses are sampled. For each grasp pose $T_{h,G}$, we calculate the grasp pose in robot frame by $T_{R,G} = \tilde{T}_{R,h} T_{h,G}$ with $\tilde{T}_{R,h}$ determined by one of the three methods **ZA**, **ZAWF** or **ZAWFG**. The robot grasps the part with the pose $T_{R,G}$, lifts the part, and tries to re-insert the part. If insertion is successful, the robot executes the next grasp otherwise the experiment terminates. We report total number of successful insertions before termination (Table. 7.3). We repeat the experiment three times for each method with different human demonstrations.

After applying z-axis alignment for the human-provided target pose (**ZA**), the insertion fails as the center of the grasp is not aligned with the center of the receptacle. **ZAWF** addresses this issue by using minimum pose refinement, and can perform successful insertions. However, since the pose refinement is specific to the human demonstration, the refinement is not sufficiently granular, leading to failures when the new grasp configuration has a large y-axis rotation. **ZAWFG** performs pose refinement for each of the grasps, resulting in consistent insertion performance. **ZAWFG** needs to wait for the force measurements to settle and thus takes longer to execute.

Exploring Utility of Tactile and Vision Information

We study the relative benefits of using tactile and vision for insertion tasks. We test 3 different approaches: (1) A Tactile Only approach (2) A Vision Only approach trained using a limited amount of the camera observation data and (3) a Combined Approach. This ablation study differs from our earlier experiments by injecting a uniformly sampled noise in the range $\pm$1mm into the target pose's x, y translation to imitate imprecise knowledge of the target pose.

The Tactile Only approach attempts the entire insertion task in a single phase using the tactile information to align the USB connector with the receptacle and then move straight down to insert it. For the purpose of this study, we train a new Vision Only model using a third of the collected camera observation data set that has no additional gripper rotation. This mimics a mono-view visuoservo model for receptacle localization and top-down insertion.

Algorithm	Tactile Only	Vision Only (No Rot)	Tactile + Vision (No Rot)
Success/Total	21/45	8/45	40/45

Table 7.4: Ablation study with noisy target poses comparing single-phase Tactile Only, modified Vision Only, and a Combined two-phase approach leveraging tactile and visual information.

We modify the insertion motion from Sec. 7.4 so the model only uses camera observation and the y-axis rotation of the gripper, and it only outputs translation in x and y based on the same regression objective as in Alg. 7.4. The combined approach sequences a Tactile Only approach in an *Align* phase with the modified Vision Only approach in an *Insert* phase. We perform experiments with the three different approaches with the same test set as in Sec. 7.5 and report results in Table 7.4.

With noisy target pose, the Tactile Only model succeeds only 21/45 times since the model does not estimate the target receptacle state. Since the Vision Only model is constrained to translation actions and is trained on a limited set of data that does not include additional gripper rotation, it inserts the part when the grasps do not have any rotation (8/9 successes) but fails otherwise (for a total of 8/45 successes). A separate Vision Only model trained with all the data is similarly unsuccessful (11/45 successes), indicating the importance of the ability to correct for grasp pose variation using tactile data. The combination of the two models outperforms either model by leveraging the part rotation prediction (using tactile) and implicitly estimating the environment state (using visual information), suggesting that tactile and vision observation jointly reduce the uncertainties in the insertion problem.

7.6 Limitations and Conclusion

Despite promising results, this method has not been tested for generalization to other types of assembly tasks, objects with more complex geometries, or objects that are larger than the tactile sensor. Parts made of different materials may require distinct maximal forces; the grid search for finetuning the insertion pose lengthens the data collection process. The robot must also unplug the part, which can pose a challenge as some parts are designed to be difficult to remove (i.e. an Ethernet connector). This work did not measure the time required for data collection. Future research can improve on the time required for collecting data. In summary, we present a safe, self-supervised method for learning a visuo-tactile insertion policy in real industrial settings with unknown grasp poses. We achieve this by using force-torque sensing to refine human-demonstrated target poses and constructing a two-phase approach to insertion that separates the task into alignment and insertion based on tactile and visual feedback.

Chapter 8

Real2Render2Real: Scaling Robot Data Without Dynamics Simulation or Robot Hardware

8.1 Introduction

> The great power of general purpose methods ... [is that they] continue to scale with increased computation.
>
> — Richard Sutton, *The Bitter Lesson* (2019)

Robotics has long benefited from computational scalability—methods like probabilistic planning, trajectory optimization, and reinforcement learning have driven significant progress in agile locomotion [68, 212–214, 307–309]. Dexterous manipulation, however, presents unique challenges: it requires fine-grained visual perception that is tightly coupled with robot control and kinematics to interact with objects and alter the environment. Many systems address this by explicitly separating perception from planning and control, achieving strong performance in structured environments [215, 218, 219, 310], especially when assumptions about scene geometry, object placement, and sensing modalities hold. Yet such pipelines often rely on task-specific perception modules and carefully controlled environments, limiting flexibility in more unstructured, dynamic, or visually diverse settings.

Hoping to address open-world manipulation tasks and inspired by large language models and vision-language models [12, 167, 311, 312], researchers have explored end-to-end robot policies [50–52, 54, 55, 95, 113, 122, 225, 313, 314]—models that learn directly from raw sensory input and suggest capabilities like language instruction following, task transfer, and in-context learning. Yet training such models at scale remains limited by data: the largest human teleoperation datasets are over 100,000× smaller than the corpora used to train frontier LLMs and VLMs [97, 315, 316], and are constrained by the cost, speed, and embodiment-specific nature of human teleoperated data collection.

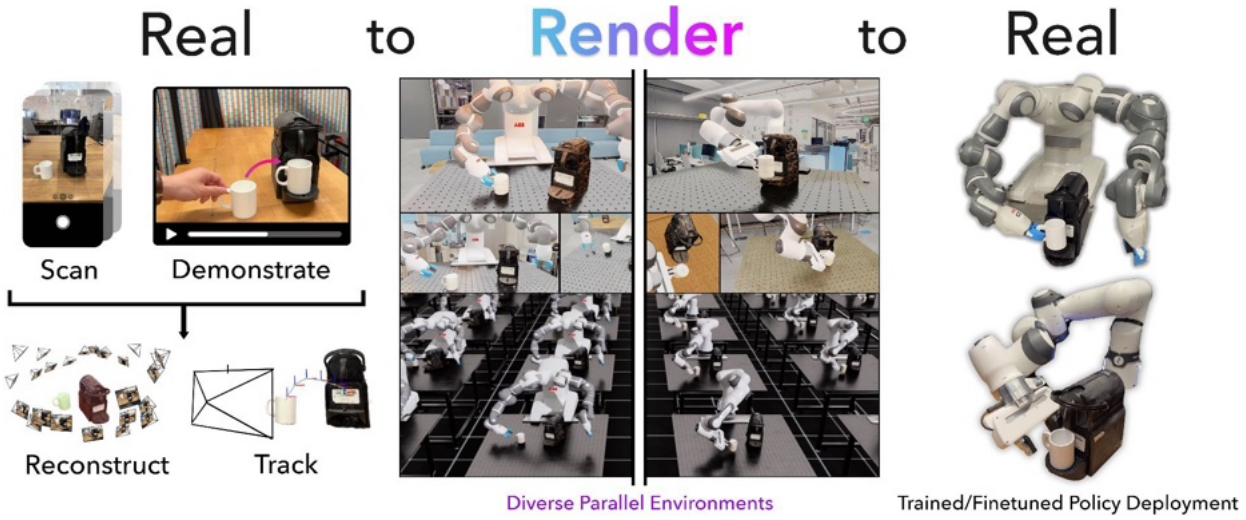


Figure 8.1: Real2Render2Real generating robot training data for the task of "Put the Mug into the Coffee Maker". R2R2R takes as input a multi-view object scan and a monocular human demonstration video. R2R2R then synthesizes diverse, domain-randomized robot executions through parallel rendering and outputs paired image-action data for policy training. This pipeline enables scalable learning across tasks and embodiments without teleoperation or object dynamics simulation.

Other vision-language subfields have faced similar data scarcity—and overcome it through *computational* data generation. Structure-from-motion, detection, and depth pipelines now routinely produce synthetic data to bootstrap large models; for instance, SpatialVLM synthesizes two billion spatial-reasoning QA pairs [101], while RAFT [102], DUSt3R [103], MonST3R [104], Zero-1-to-3 [105], and MVGD [106] all rely on synthetic ground-truth derived from multi-view geometry pipelines (e.g., COLMAP [107]) to supervise dense 3D prediction tasks.

To computationally scale up robot data, prior efforts have turned to physics-based simulation, where trajectories are synthesized via reinforcement learning or motion planning in virtual environments [264–266]. While modern simulators offer high throughput and support large-scale parallelization, they face several fundamental limitations: many commonly used simulators fail to satisfy basic Lagrangian mechanics, such as conservation of energy or momentum [260]; accurately modeling complex object interactions often demands extensive parameter tuning and hand-crafting of contact properties [267]; generating high-quality, compliant, and intersection-free assets for simulation remains labor-intensive, as collision modeling requires careful handling of geometry, friction, and deformation [317, 318].

Can we computationally scale robot-vision-action data – without physics-based simulation or human teleoperation – to train robot learning models?

R2R2R avoids these challenges by discarding dynamics: instead of simulating forces or contacts, we directly set object and robot poses per frame using IsaacLab [255] purely as a photorealistic, parallelized rendering engine by setting all objects as kinematic rather than dynamic bodies. This approach respects robot kinematics while avoiding the complexities

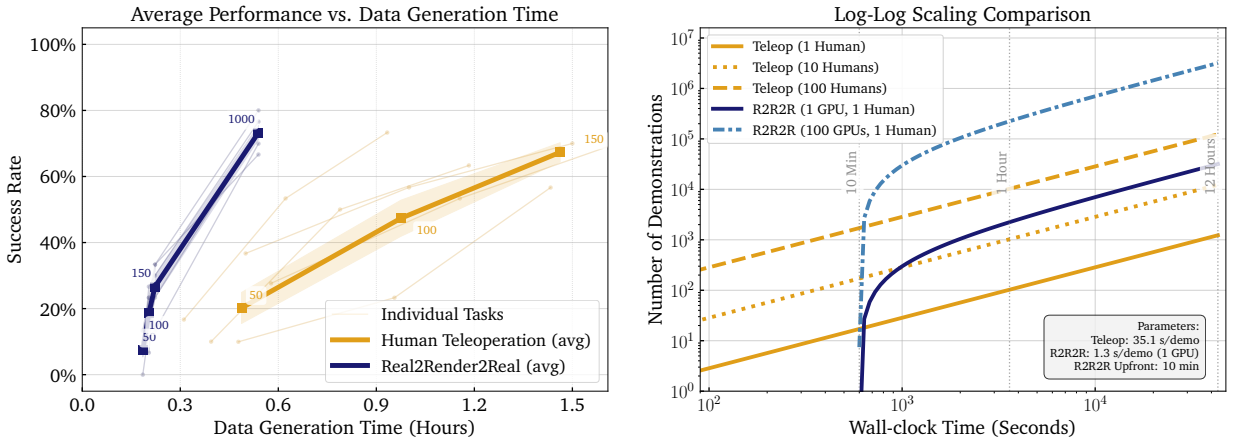


Figure 8.2: Data Generation Efficiency and Average Policy Performance Across Manipulation Tasks. (Left) Performance visualization displaying both task-specific outcomes (faint background lines) and cross-task averages (bold lines with error shading) for policies trained on real (1 human teleoperator) vs. synthetic data (1 human, 1 GPU). The points labeled by demonstration count (50-1000) highlight the scaling in performance and R2R2R's significant throughput advantage, with individual task trajectories illustrating the variance across different manipulation scenarios. **(Right)** Log-log scale comparison showing data generation throughput between R2R2R (1-100 GPUs) and human teleoperation (1-100 operators) over a 12-hour period. R2R2R needs an upfront time of 10 minutes for a human to scan the objects, demonstrate the task, reconstruct the objects and track their trajectory, and subsequentially no human is involved. On a single NVIDIA 4090 GPU, on average, trajectories will be generated at 27x the speed of a single human teleoperator without needing robot hardware.

of contact modeling, naturally aligning with vision-based robot policies trained from RGB images and proprioceptive inputs.

We introduce **Real2Render2Real (R2R2R)**, a pipeline for generating large-scale synthetic robot training data from a smartphone object scan and a human demonstration video. R2R2R scales *trajectory diversity* while preserving visual accuracy: it extracts 6-DoF object part trajectories from one human demonstration video and generates corresponding robot executions via inverse kinematics under randomized object initial poses. Starting from a multi-view scan, it reconstructs 3D object geometry and appearance, supports both rigid and articulated objects via part-level decomposition, and uses 3D Gaussian Splatting to produce mesh assets. The resulting trajectories include robot proprioception, end-effector actions, and paired RGB observations rendered under varied lighting, camera pose, and object placement—making them directly compatible with modern imitation learning policies such as vision-language-action models and diffusion models. By eliminating the need for dynamics simulation or robot hardware, R2R2R enables accessible and scalable robot data collection, allowing anyone to contribute by capturing everyday object interactions with a smartphone.

This chapter makes three contributions. First, we present *Real2Render2Real (R2R2R)*, a novel framework that synthesizes diverse, physically grounded demonstration data using

only smartphone-captured videos: a multi-view object scan and a human demonstration video—without requiring dynamics simulation or robot hardware. Second, we demonstrate that this data is compatible with vision-language-action (VLA) and imitation learning policies, including both transformer-based and diffusion-based architectures that operate from RGB and proprioceptive input. Third, we show that policies trained on R2R2R-generated data based on one human video demonstration can match the performance of those trained on 150 human teleoperation demonstrations, across 1,050 physical robot evaluations, while requiring significantly less time to generate.

8.2 Related Work

Robot Data Collection Paradigms. Scaling robot learning has traditionally relied on two paradigms: data from industrial deployments and data from human teleoperation. Industrial robot logs [319–321] scale with production throughput but are often task- and embodiment-specific. In contrast, teleoperation datasets [48, 141, 157, 158, 322] offer greater visual and task diversity but remain bottlenecked by human effort and real-time collection. At the same time, the rise of generalist robot policies [50–52, 54, 55, 95, 113, 122, 225, 313, 314, 323]—capable of performing diverse manipulation tasks from raw observations—has amplified the need for scalable, diverse, and high-quality training data. Yet the scale of current robot datasets remains orders of magnitude below that of their vision and language counterparts [97, 315].

Procedural Robot Data Generation. To address the challenge of robot data scaling, many works have studied procedural data generation to automate robot data collection for pre-defined tasks. Many works [56, 216, 324–327] use pre-defined motion primitives, optionally with a perception module, to automate data collection using a real robot, with automatic scene reset. While reducing human interventions, they still require robot hardware for data collection, limiting scalability. More recently, simulation data generation has emerged as a scalable alternative to real-world collection, parallelizing data generation without physical robot hardware. Utilizing the privileged information from the simulator, Mahler et al. [218] generates large and diverse data for robot grasping. Wang et al. [264] and Katara et al. [265] generate large-scale robot data in simulation using reinforcement learning, trajectory optimization, and motion planning. MimicGen [262] synthesizes diverse simulations from a single human tele-operation sequence, combining motion planning and trajectory replaying. Despite efforts to bridge the sim-to-real gap through domain randomization [250], improved asset and scene generation [264, 265], the resulting simulation data often exhibit significant visual discrepancies from real-world observations, requiring co-training on real data to enable effective transfer [270].

Real2Synthetic Data Generation. To mitigate this visual domain gap, some works augment and repurpose real RGB data instead of synthesizing it from scratch. For example, Chen et al. [329] employs generative models for inpainting robot embodiment features into real images, enabling data synthesis for robots with different morphologies. However, such

	Tele-Op Free	RL Free	Phys. Engine Free	Robot Agnostic	One-to-Many Trajectories	Articulated Objects
CASHER [267]	✗	✗	✗	✓	✓	✓
RoboVerse [260]	✗	✗	✗	✓	✗	✓
RoboGSim [328]	✗	✓	✓	✗	✗	✗
RoVi-Aug [329]	✗	✓	✓	✓	✗	✓
Video2Policy [266]	✓	✗	✗	✓	✓	✓
MimicGen [262]	✗	✓	✗	✓	✓	✗
DexMimicGen [263]	✗	✓	✗	✓	✓	✗
Phantom [237]	✓	✓	✓	✓	✗	✓
DemoGen [330]	✗	✓	✓	✗	✓	✗
AR2-D2 [331]	✓	✓	✓	✓	✗	✓
Real2Render2Real	✓	✓	✓	✓	✓	✓

Table 8.1: Comparison of Robot Data Generation Methods. *Real2Render2Real* requires no teleoperation, eliminates reliance on reward engineering, reinforcement learning, or accurate asset physics modeling, and provides object-centric demonstrations directly extracted from a video where humans interact with the objects. It also supports various robot embodiments, and generates multiple varied trajectories from a single demonstration.

approaches still require human teleoperation for initial demonstrations. Further, they lack the ability to generate additional diverse trajectories beyond the provided demonstrations. Similarly, Lepert et al. [237] and Duan et al. [331] use hand-pose tracking to guide inpainted robot end-effector trajectories from human demonstrations. While these methods reduce the need for direct teleoperation, they typically generate only a single trajectory per video and lack support for computationally-scaled trajectory diversity. In contrast, R2R2R can generate multiple, diverse robot trajectory renderings and action rollouts from a single human demonstration. Policies trained solely on R2R2R-generated data achieve real-world performance comparable to those trained on human teleoperation data.

Real2Sim2Real Data Generation. To generate diverse trajectories from a single demonstration while bridging the sim-to-real gap, many methods follow a Real2Sim2Real paradigm—using real-world observations to build simulated environments to train policies deployed back in the real-world. Prior work [332] shows that tuning physics parameters can reduce dynamics mismatch, but large visual domain gaps often still necessitate test-time perception modules. Recent methods [260, 263, 266, 269] reduce this visual gap by constructing digital twins or "digital cousins"[333] from real scans. These approaches vary in their reliance on teleoperation, simulation, and trajectory diversity—but many still depend on teleoperated demos, handcrafted rewards, or accurate physics models, limiting scalability. For example, DexMimicGen[263] uses fixed simulation assets; RoboVerse [260] supports only rigid objects; and RialTo [268] and CASHER [267] require manual articulation labeling and reward engineering. While Video2Policy [266] avoids reward tuning via vision-language models, it still requires test-time object detection due to visual mismatches. These pipelines also rely on physics engines, which demand high-fidelity meshes for collision checking and extensive

tuning. RoboGSim [328] avoids simulation but lacks support for trajectory diversity from a single demo. In contrast, R2R2R addresses these limitations by: (1) extracting object trajectories from human videos, (2) segmenting object parts automatically, (3) rendering realistic observations to remove reliance on additional perception models, (4) eliminating the need for collision modeling and detailed meshes, and (5) generating diverse trajectories from a single demonstration.

8.3 Assumptions

We assume objects are rigid or articulated, and manipulated on a table-top setup under quasi-static conditions. Object surfaces are assumed to exhibit low specularity to support robust geometry reconstruction and visual feature extraction. We also assume that during human demonstrations, objects are not placed in configurations that lead to complete mutual occlusion. Approximate camera poses relative to the robot in the physical setup are assumed to be available, enabling the generation of observations from nearby viewpoints during data collection. Learned policies take RGB image observations and robot's proprioceptive states as inputs.

8.4 Method

Real2Render2Real (R2R2R) is a data generation pipeline for synthesizing diverse robot demonstration data consisting of RGB-action pairs from a single human demonstration and multi-view object scan. R2R2R consists of three primary stages: (1) **real-to-sim asset and trajectory extraction**, where rigid or articulated object geometry and part trajectories are extracted from real-world smartphone captures; (2) **augmentation**, where object initialization is randomized and object motion trajectories are interpolated if appropriate; and (3) **parallelized rendering**, where diverse photorealistic robot executions are generated using IsaacLab [255], scalable with the amount of available GPU memory and the numbers of GPUs.

Real-to-Sim Asset Extraction

We extract 3D object assets from smartphone scans using a two-stage process inspired by [334, 335]. First, we reconstruct object geometry and appearance using 3D Gaussian Splatting (3DGS) [336], then apply GARField [337] to segment the scene into semantically meaningful parts by lifting 2D masks into 3D. This enables both object-level and part-level decomposition, including articulated components. To support mesh-based rendering, the resulting Gaussian groups are converted into textured triangle meshes via an extended version of [338].

Figure 8.3: 3D Gaussian Splat Object Reconstructions with part-level segmentations derived from feature-based grouping. Objects are reconstructed and segmented into rigid or articulated components using GARField [337].

Real-to-Sim Trajectory Extraction

Given a smartphone video of a human manipulating the scanned objects, R2R2R extracts the 6-DoF part motion of the object and its parts using 4D Differentiable Part Modeling (4D-DPM) introduced in [334]. Each 3DGS object part is embedded with pre-trained DINO features, enabling part pose optimization through differentiable rendering. We extend [334]'s implementation to track single or multiple rigid objects, as well as articulated ones, from demonstration videos.

While there are many alternative pipelines that convert real images into 3D assets, we adopt 3DGS-to-mesh conversion for two key reasons: (1) it enables background–foreground segmentation and part decomposition via 3D grouping [337], which is critical for extracting object part-specific trajectories from monocular human demonstrations; and (2) it maintains compatibility with both 4D-DPM trajectory reconstruction and instanceable mesh-based rendering engines, allowing seamless integration into our large-scale rendering pipeline. This process requires no fiducials or hardware beyond a smartphone camera, making it well-suited for scalable and accessible real-to-sim data generation.

Interpolation Methods for Object Trajectory Diversity: A key contribution of Real2Render2Real is the ability to synthesize multiple valid 6-DoF object trajectories from a single human demonstration. In the case of multiple rigid objects that interact, (e.g. putting a mug on a coffee-maker) the original demonstration is valid only for a specific initial object configuration, and naively replaying it from a new initial pose would fail. To address this, we introduce a suite of trajectory interpolation and resampling techniques that adapt the original trajectory to new start and end poses while preserving its semantic intent.

Figure 8.4: Trajectory Interpolation R2R2R adapts object motion to varied start/end configurations via spatial normalization and Slerp.

We begin with a reference trajectory $\tau \in \mathbb{R}^{T\times 7}$ consisting of T waypoints provided by the part tracking from the demonstration video, each encoding an object orientation (quaternion) and position. Given a new initial pose $\mathbf{x}_{\text{start}}$ and the desired end pose $\mathbf{x}_{\text{end}}$ from human demonstration, we

apply a spatial normalization that transforms the original trajectory into a canonical space. We compute the affine transform between the original and target endpoint poses, apply it to the translational component of the trajectory, and interpolate keyframe orientations using spherical linear interpolation (Slerp). This results in a new trajectory τ' that begins and ends at the desired poses while respecting the structure of the original motion. While these trajectories preserve high-level semantic intent, they are generated without explicit collision avoidance and may result in infeasible paths when initialized behind occluding objects. To mitigate this, we apply a sampling heuristic that biases the distribution of initial placements away from the goal pose (see Fig 8.4).

Grasp Pose Sampling: R2R2R estimates 3D hand keypoints from the demonstration video using [232], then determines object-hand interactions by computing the Euclidean distance between keypoints (index fingertip and thumb) and the centroids of all segmented object parts. This produces a distance matrix indexed over time and object parts. We identify the grasped part as the one with the minimum aggregate distance across the trajectory, effectively selecting the object most consistently proximal to the hand throughout the demonstration. To generate physically plausible grasps, we sample 3DGS means to construct a coarser triangle mesh (distinct from the high-resolution rendering mesh), apply surface smoothing and decimation to obtain consistent normals, and use an analytic antipodal grasp sampler following [218] to determine candidate grasp axes. For bimanual tasks, this process is applied independently per hand to infer separate object associations and grasps, supporting coordinated actions such as lifting or stabilization.

Differential Inverse Kinematics: For each grasp and object trajectory pair, we solve a differentiable inverse kinematics problem using PyRoki [339]. The solver computes smooth joint-space trajectories that induce the desired object motion across the pre-grasp, grasp, and post-grasp phases. Crucially, our method does *not require modeling object dynamics or simulating physics interactions*. Instead of solving for joint torques that would physically induce object movement (as in dynamic simulation), we assume the object rigidly follows the trajectory during contact. This kinematic assumption avoids challenges like contact modeling, compliance, or friction estimation. The solver simply ensures that robot kinematics can track the desired object motion subject to joint limits, and during pre- and post-grasp phases we additionally add smoothness and velocity costs to the differentiable IK problem, which generates valid grasp approach motions.

Rendering Diverse Environment Contexts: We apply domain randomization across both scene geometry and rendering parameters. This includes randomized lighting conditions (e.g., intensity, color temperature, background images), camera extrinsics (uniformly sampled up to 2cm translation and 5° rotation), and object initial poses (sampled within a workspace-relevant range). By modeling 3D object-centric representations, we can apply these augmentations directly during rendering. Changes in camera pose or lighting do not affect the underlying kinematic rollout, allowing R2R2R to generate diverse visual contexts from a single demonstration. These augmentations expand the data distribution and improve generalization by mitigating the appearance gap and covariate shift between synthetic demonstrations and real-world deployment.

High-Throughput Rendering: IsaacLab [255] supports GPU-parallel execution of multiple environment contexts using tile-based rendering, deep-learning super sampling (DLSS), and mesh asset instancing. On a single NVIDIA RTX 4090, R2R2R uses the IsaacLab framework to render complete robot demonstrations at an average rate of 51 demonstrations per minute—compared to 1.7 demonstrations per minute via human teleoperation—yielding over a **27**× speedup. This throughput scales linearly with the number of rendering GPUs, as depicted in Figure 8.2. Data generation/collection time per task can be found in Table D.6.

Policy Learning

We consider two modern imitation learning architectures: Diffusion Policy [54] and π_0-FAST [340]. We train Diffusion Policy from scratch for 100k steps conditioned on a 4-timestep history of proprioception and 448px RGB observations to iteratively denoise 16 future absolute end-effector poses in SE(3). We finetune π_0-FAST for 30k steps using Low Rank Adaptation (LoRA) [341] (rank=16), which takes 224px square image observations (to match pretraining resolution) and predicts a 10-step relative joint angle action-chunk. Training the diffusion policy takes approximately 3 hours on a single NVIDIA GH200, while π_0-FAST finetuning takes 11 hours. At deployment, both models receive raw RGB images and robot proprioception—SE(3) absolute end-effector pose for diffusion and joint positions for π_0-FAST—and output the corresponding action targets. To improve temporal consistency between actions predicted at different timesteps, we apply temporal ensembling [55] to predicted action-chunks during execution for both models. More details can be found in Section D.1.

8.5 Experiments

We conduct physical robot evaluations on an ABB YuMi IRB14000 Bimanual Robot (a robot embodiment unseen during π_0-FAST pre-training) across five manipulation tasks using the trained policies. Policies are trained on either human teleoperation data or synthetic demonstrations generated by R2R2R. To assess how policy performance scales with training data, we train models with 50, 100, 150, and 1,000 rendered trajectories and up to 150 teleoperation trajectories per task. To ensure a fair comparison, all models are trained for a fixed number of training steps using only third-person RGB observations. For each task, each model is evaluated 15 times on the same set of pre-determined, in-distribution object pose. This sums up to 1,050 physical policy rollouts.

We selected tasks to highlight R2R2R's ability to scale across diverse manipulation scenarios that involve varying physical and kinematic structures. Specifically, the tasks span: single-object picking (*"pick up the toy tiger"*), multi-object interaction (*"put the mug on the coffee maker"*), articulated object manipulation (*"turn the faucet off"* and *"open the drawer"*), and bimanual coordination (*"pick up the package with both hands"*). These categories correspond directly to R2R2R 's support for part-level segmentation, articulated object

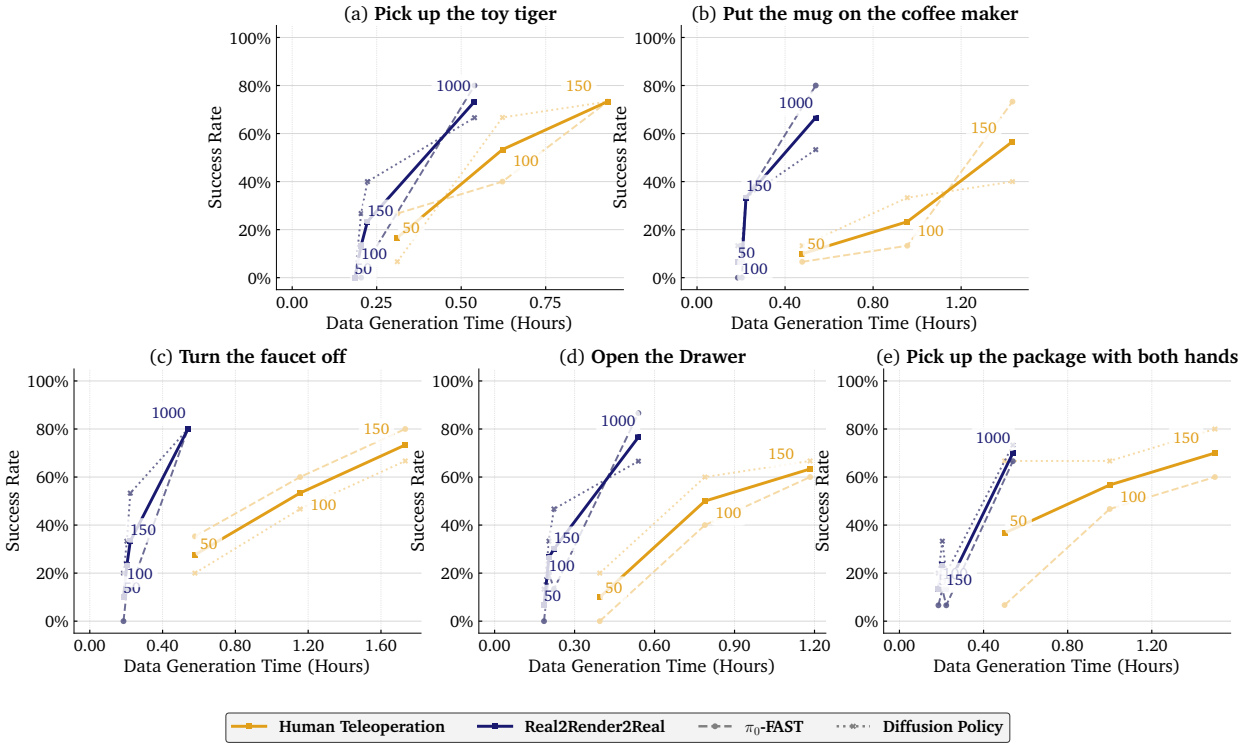


Figure 8.5: Physical Experiments Comparing Real2Render2Real to Human Teleoperation Data Efficiency Task success rate is plotted against data generation time in hours. Solid lines represent performance averaged across π_0-FAST and Diffusion Policy. The Real2Render2Real line (blue square) includes points corresponding to 50, 100, 150, and 1000 trajectories generated by a single Nvidia RTX 4090. The Human Teleoperation line (gold square) includes points corresponding to 50, 100, and 150 trajectories. For each task, each model is evaluated 15 times on the same set of pre-determined, in-distribution object poses. The R2R2R data generation time includes a 10-minute setup cost, while the human teleoperation time is based on the real trajectory collection time of 150 demonstrations. Exact numbers for evaluation results can be found in Section D.1.

reconstruction, and multi-arm grasp planning, and are visualized in appendix Section D.1. We provide additional ablation experiments on trajectory interpolation (Section D.1), increased background randomization (Section D.1), and sim-real co-training (Section D.1) in the appendix.

Performance Scaling and Comparison

To evaluate how well R2R2R-generated data supports policy learning compared to human teleoperated data, we analyze performance trends as a function of dataset size across the five tasks described above. Results are summarized in Figure 8.5. We observe that R2R2R-generated data scales predictably with dataset size: success rates increase roughly linearly for all tasks as the number of demonstrations grows. On the "Put the mug on the coffee maker"

task (see Figure 8.5b), performance of Diffusion Policy trained on R2R2R data improves from 33.3% at 150 demos to 53.3% at 1000 demos, while π_0-FAST jumps from 33.3% to 80.0%. While higher quality, real-world data offers better performance in low-data regimes (e.g., π_0-FAST reaches 73.3% at 150 real demos vs. 33.3% at 150 R2R2R demos as shown in Figure 8.5b), as the scale increases to 1000 demos, R2R2R achieves performance that matches or surpasses teleoperation across multiple tasks. This suggests that while real data is more efficient per demonstration, R2R2R's generation enables scaling *trajectory diversity* far beyond human throughput, achieving competitive final performance with less collection effort.

To assess whether this performance is statistically comparable, we conduct formal significance and equivalence testing across all tasks and models. Appendix D.1 shows that on the evaluated tasks, there are no statistically significant differences between policies trained on R2R2R versus human teleoperation data on the tasks we evaluated. Two One-Sided Tests (TOST) further suggest that the observed differences fall within a $\pm 5\%$ margin, indicating similar overall performance.

8.6 Conclusion

We present R2R2R, a scalable data generation pipeline that creates robot training data from a single smartphone object scan and a human demonstration video. R2R2R mitigates limitations of prior work by removing the need for teleoperation, robot hardware, or dynamics simulation. It leverages 3D Gaussian Splatting to represent both rigid and articulated objects, enabling parallel rendering using Gaussian-converted meshes and scalable rendering engines. These realistic renderings serve as visual observations for policy training. Given the robot's URDF, R2R2R synthesizes diverse robot trajectories with extracted object motion from one human demonstration using differential inverse kinematics. Experiments on five robotic tasks suggest that policies trained on data generated by R2R2R scale with data volume and perform comparably to those trained on teleoperated demonstrations, demonstrating that R2R2R is a practical and scalable pipeline for real-world robot manipulation policy learning.

8.7 Limitations

Real2Render2Real (R2R2R) enables scalable data generation and competitive real-world performance, but several limitations remain.

Reconstruction and Simulation Fidelity. R2R2R relies on vision-based reconstruction methods—such as 3D Gaussian Splatting and mesh conversion—that yield high-fidelity appearance but often lack watertight or physically plausible geometry. These limitations make it difficult to simulate realistic physical interactions, especially in contact-rich settings. As a result, R2R2R forgoes physics simulation entirely. While this design choice boosts scalability, it also restricts modeling of important dynamics such as friction, compliance, and

force feedback. As real-to-sim pipelines mature in their physical realism [342], we anticipate extending R2R2R toward non-prehensile and dynamic tasks.

Two-Stage Input. While a single dynamic video (moving objects, moving camera) as input would offer higher scalability, current image-to-mesh pipelines lack the necessary multi-view consistency and self-supervised articulated part segmentation, both crucial for tasks like opening drawers or turning faucets. Future directions include unified pipelines based on emerging 4D-tracking and reconstruction works.

Scene Diversity and Collision Awareness. Trajectory generation in R2R2R is performed via geometric interpolation, without considering environmental context such as distractor objects or obstacles. As a result, synthesized trajectories may intersect with the scene geometry, leading to physically infeasible plans. Incorporating fast motion planning techniques during trajectory synthesis could improve collision avoidance and robustness, particularly in cluttered or multi-object scenes.

Scope of Manipulation Tasks. The current framework focuses exclusively on rigid and articulated objects using prehensile manipulation. It does not support deformable object handling or non-prehensile strategies such as pushing, toppling, or sliding. These interactions often demand accurate metric depth estimates and fine-grained physical modeling—both of which are challenging with monocular video and approximate geometry. Extending R2R2R to these broader manipulation regimes remains an open direction.

Grasping Generality. R2R2R's grasp generation module currently uses antipodal grasp sampling, which limits compatibility to parallel-jaw grippers. This restricts the generality of trained policies and excludes multi-fingered or anthropomorphic hands, which require richer grasp representations and contact models. Supporting these more complex end-effectors would require advances in grasp synthesis and simulation.

Tracking Robustness. Like other object-centric pipelines, R2R2R is vulnerable to tracking failures under fast motion, heavy occlusion, poor texture, or reflective surfaces. In such cases, object reconstructions and pose tracks may be inaccurate, resulting in degraded data quality. These failures can lead to invalid grasps or trajectories that do not transfer well to the real world. Robustifying tracking and adding confidence-aware filtering or correction is an important area for future work.

Policy Failure. Precise perception is still a major issue for visual imitation learning models (i.e. they will still miss the handle of the mug; sometimes an off-center grasp will lead to rotation of the object, resulting in missed grasps). Since we are only using third-person cameras, we hypothesize the models may achieve better performance by adding wrist cameras to increase grasp approach precision.

Addressing these limitations—through richer physical modeling, context-aware planning, expanded manipulation capabilities, and improved reconstruction robustness—offers a plausible path toward more general and reliable robot learning at scale.

Part III

Agentic Robot Control

Chapter 9

Agentic Robot Control

9.1 Beyond Action Prediction

Parts I and II address two central requirements for scalable robot learning. Part I studies how robot policies can be pre-trained to generalize across tasks, while Part II studies how the data needed for such pre-training can be scaled beyond manual teleoperation. Together, these two parts treat robotics as a problem of learning better sensorimotor predictors from larger and more diverse trajectory distributions. This view has become increasingly powerful: recent vision-language-action (VLA) systems have demonstrated that, with enough data and post-training, learned policies can solve long-horizon manipulation tasks such as T-shirt folding, box folding [53], and industrial assembly [211].

However, scaling imitation learning alone does not fully resolve the problem of embodied intelligence. Many real robot tasks are not merely longer versions of the same closed-loop control problem. They require deciding which subtasks should be performed, when to call different tools or controllers, how to recover from failures, and how to use task-relevant information that may not be directly represented in a learned action policy. A robot operating for minutes or hours in an open-ended environment may need to reason about object relations, constraints, affordances, tool use, and intermediate goals before any low-level action becomes well-defined. In these settings, the central question is not only how to predict the next action, but how to decide what level of action abstraction we should be using.

This distinction has deep roots in robotics [343]. In fact, most robots are not controlled through teleoperation. While humans can operate them directly through teleoperation, we also operate them indirectly through software, perception systems, motion planners, and task planners. With the right interface, a human engineer can make a robot perform a wide range of tasks, from precise industrial assembly to whole-body loco-manipulation. Teleoperation is one such interface, but it is not the only one, and it may not always be the most scalable one. For many tasks, especially those requiring compositional reasoning or explicit use of symbolic structure, the relevant human contribution is not a trajectory, but a program.

These observations motivate the organizing question of Part III: *how much of embodied*

control should be understood as sensorimotor prediction, and how much should be understood as reasoning over tools, programs, and abstractions? The position of this dissertation is that a general-purpose robot system likely requires both. Sensorimotor prediction is essential for fast, continuous, contact-rich behavior. Reasoning and program synthesis are essential for composing skills, using tools, debugging failures, and solving tasks whose structure is not captured by direct imitation. A capable embodied agent should therefore be able to act not only by executing learned policies, but also by writing code that orchestrates perception, planning, control, and learned skills.

Part III tries to push towards the limit for the latter axis. In particular, we ask how far robot control can be driven by agentic code generation alone. That said, the goal is not meant to replace the learned policies developed in Parts I and II. Instead, it isolates the complementary role of coding agents: the ability to reason over explicit abstractions, compose reusable primitives, and turn natural-language goals into executable robot programs.

9.2 Code-as-Policy for Robot Control

The idea of using language models to write robot programs is not new. ProgPrompt [90], Code as Policies [89], MAESTRO [94], and more recent systems such as Gemini Robotics [95] have shown that language models can generate robot control code from natural-language instructions when provided with perception and control APIs. In these systems, code provides a useful intermediate representation. It can express loops, conditionals, geometric computation, tool calls, error handling, and calls to external libraries in ways that are difficult to represent as a single continuous action sequence.

The limitation is that much of the intelligence in these systems is often supplied by the human-designed interface. Prior code-as-policy systems commonly include carefully engineered prompts, worked examples, task-specific helper functions, and high-level APIs that hide much of the difficulty of perception, planning, and control. A model may appear to solve a manipulation task by writing code, but the generated program may only need to call a function such as `stack_objects_in_order()` or `pick_and_place()`, where the hard robotics logic has already been handled by human developers. In this sense, the language model contributes orchestration, but the human engineer contributes the abstractions that make orchestration tractable.

This creates an important ambiguity. When a code-as-policy system succeeds, it is unclear whether success comes from the model's ability to reason about robot control, or from the human-engineered API that simplified the task. As frontier language and vision-language models become stronger at software engineering, tool use, and multimodal reasoning, understanding this ambiguity becomes important. If modern models can write longer, more structured, and more robust programs, then we need to know how much human scaffolding is still necessary, what kinds of scaffolding matter most, and whether agentic test-time computation can compensate when high-level abstractions are removed.

9.3 Open Questions and Roadmap for Part III

The work in this part is organized around three questions that the prior code-as-policy literature has not systematically answered.

1. **How do modern coding agents compare with human robot programmers?** Existing leaderboards measure mathematical reasoning, software engineering, and multimodal understanding, but they do not measure whether a model can write executable robot control code with real perception and control primitives. A useful benchmark should compare models against human developers while varying the level of API abstraction, the available feedback, and the amount of interaction allowed at test time.

2. **How much do coding agents depend on human-engineered abstractions?** High-level robotics APIs can make zero-shot code generation appear highly capable, but they also encode strong task priors and limit what the agent can express. Low-level primitives provide a more flexible and realistic interface, but they require the agent to reason more explicitly about perception, geometry, motion, and failure recovery. Understanding this trade-off is essential for distinguishing genuine agentic control capability from success induced by designer scaffolding.

3. **How can coding agents be improved specifically for robot control?** Prior code-as-policy systems largely treat language models as fixed program generators: given a prompt, an API, and a task instruction, the model produces code that is then executed on the robot. This leaves open how such agents should improve from experience. Robot control is an especially natural setting for this question because failures are grounded and observable: a grasp can miss, a placement can be misaligned, a motion plan can fail, or the final task state can remain unsatisfied. In this work, we ask how coding agents can use these execution outcomes to become better robot programmers, both by adapting their test-time behavior from feedback and by learning from task-level success signals in executable environments.

Chapter 10 addresses these questions through **CaP-X**, a framework for benchmarking and improving coding agents for robot manipulation. Importantly, the goal of Part III is not to argue that code-as-policy should supersede learned robot policies. Rather, it argues that code-based reasoning and learned sensorimotor control solve different parts of the same problem. Pre-trained policies provide the low-level competence needed for robust contact-rich behavior. Synthetic data and trajectory pre-training provide a path toward scaling that competence. Coding agents provide a complementary mechanism for long-horizon composition, tool use, failure recovery, and explicit reasoning over task structure. The conclusion of this dissertation returns to how these threads can be integrated into a single robot learning system.

Chapter 10

CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation

10.1 Introduction

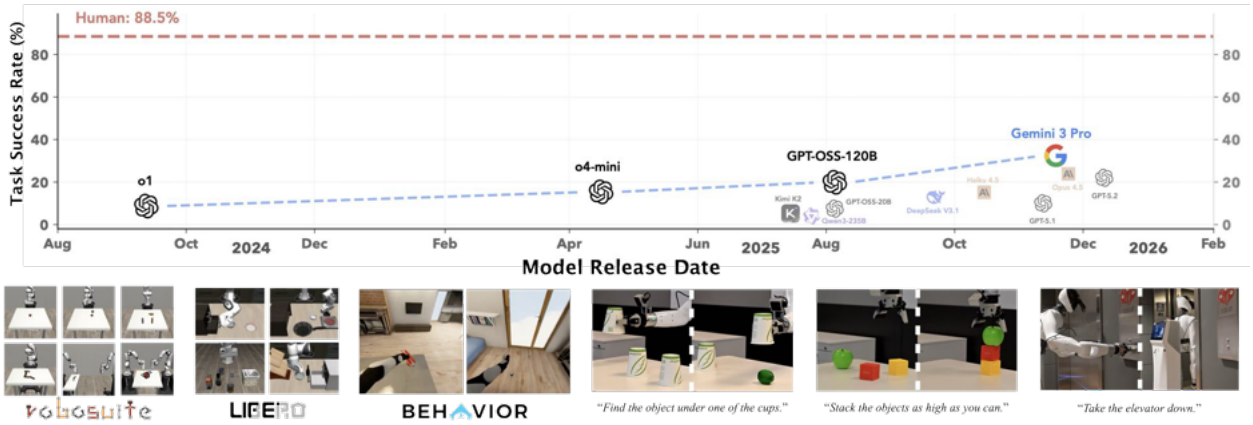


Figure 10.1: (Top) CaP-Bench Task Success Rate over Model Release Date: We primarily compare 7 tasks over 12 models' task success rates resulting from model-generated code against those from human experts. We find that while (vision-) language models have achieved capabilities comparable to humans in other domains [13, 15, 87], they still trail behind human performance in generating programmatic robot control code for diverse manipulation tasks in simulation and in the real world. **(Bottom)**: CaP-Gym integrates RoboSuite [256], LIBERO-PRO [344], and BEHAVIOR [259]. We further present CaP-Agent0 (Section 10.4), a training-free agentic framework that recovers near human-level performance on several manipulation tasks and achieves success rates comparable to—and in some cases exceeding—those of post-trained VLAs, without any task-specific training data.

Robots have long been controlled through explicit programs that combine perception, geometry, planning, and feedback control [345–348]. As robots advanced to continuous, higher-dimensional spaces, these representations were integrated with geometric motion planning [349], evolving into Task-and-Motion Planning (TAMP) [343]. These *classical* control paradigms achieve robustness through explicit structure: engineers manually write software that decomposes high-level goals into subtasks, composes perception and control modules, and handles failure and edge cases through explicit logic. While this offers strong interpretability and geometric precision guarantees, it has traditionally relied heavily on human ingenuity. The *agent* in this loop is a human expert who must manually design abstractions and anticipate corner cases, creating a scalability bottleneck that results in brittle, task-specific solutions that are difficult to generalize to open-ended environments.

Driven by successes in data scaling for foundation models [4, 39, 185, 194–198], another robot control paradigm has emerged in the form of Vision-Language-Action (VLA) models [46, 49, 50, 53, 110, 113, 116, 122, 147, 151, 169, 225, 314]. These approaches learn directly from large-scale visual-motor data, interpolating across massive imitation datasets to achieve impressive performance on contact-rich tasks such as shirt folding, tool use, and whole-body locomotion manipulation. However, VLAs inherit the limitations of their training data and design: they often lack interpretability and struggle to generalize to new robot embodiments and novel long-horizon tasks without additional data collection and retraining.

Recent advances in Large Language Models (LLMs) suggest a way to bridge these paradigms: using general-purpose coding agents to replace the human engineer in the programmatic control loop. Modern coding agents have demonstrated the ability to synthesize executable code, define functions, and debug failures on software benchmarks such as SWE-Bench [87]. Unlike earlier language-conditioned planners limited to function calling [350], today's agents can construct mid- to low-level logic that closely resembles expert code. However, while Code-as-Policy (CaP) pioneers [89, 90] validated this approach in applications to robotics, they mostly rely on high-level primitives (e.g., `stack_objs_in_order()`) that offer significant task-specific simplifications. As a result, it remains unclear how much of the observed performance in robot control stems from the agent itself versus the structure imposed by these primitives. In particular, prior work does not systematically characterize how agent performance changes as such scaffolding is reduced, nor to what extent increased test-time computation—through iterative debugging, skill synthesis, ensembled reasoning, or multimodal grounding—can compensate for operating over lower-level interfaces.

To address this gap, we introduce **CaP-X**, a unified framework for systematically evaluating and improving code-based robot control agents. At the core of CaP-X is **CaP-Gym**, an interactive environment in which agents directly control robots by generating and executing programs that compose perception and control primitives. CaP-Gym integrates 187 tasks from standard robot manipulation simulators—including RoboSuite [256], LIBERO-PRO [258, 344], and BEHAVIOR [259]—under a shared primitive design that is intentionally compatible with both simulation and physical robot systems.

Building on CaP-Gym, we construct **CaP-Bench**, a benchmark designed to systematically study agentic capability along three axes: **Abstraction Level:** Varying the action space

Table 10.1: Comparison of CaP-Bench evaluation tiers.

		Single-Turn				Multi-Turn			
Category	**Characteristic**	**S1**	**S2**	**S3**	**S4**	**M1**	**M2**	**M3**	**M4**
Perception	Noiseless (State-Based)	✓							
	Noisy		✓	✓	✓	✓	✓	✓	✓
Primitive Abstraction	High-level	✓	✓			✓	✓	✓	
	Low-level			✓	✓				✓
In-Context Learning	Primitive Usage Examples			✓					✓
Visual-Grounding Modality	Multimodal Feedback						✓		
	Visual Diff. Module (VDM)							✓	✓

from human-crafted macros (High-Level) to atomic, fundamental primitives (Low-Level); **Temporal Interaction:** Comparing zero-shot single-turn program generation against multi-turn interaction to quantify capabilities in failure recovery and iterative reasoning; and **Perceptual Grounding:** Evaluating how different modalities of visual feedback impact the agent's ability to ground task-relevant visual features into code generation. We instantiate CaP-Bench by focusing on a subset of 7 environments from CaP-Gym and evaluate 12 state-of-the-art open- and closed-source language models and vision-language models.

Guided by insights from CaP-Bench results, we derive **CaP-Agent0**, a training-free agentic framework that augments coding models with multi-turn interaction, visual grounding into text, an automatically synthesized task-agnostic skill library, and parallelized multi-model code generation. CaP-Agent0 achieves performance comparable to, and in some cases exceeding, human expert baselines on CaP-Bench tasks. Finally, we show that CaP-Gym supports **CaP-RL**–reinforcement learning on the coding agent itself. On-policy post-training with environment rewards improves task success, while synthesized programs transfer directly to real robots. This chapter makes the following contributions:

1. CaP-Gym, a unified suite of interactive robot coding environments spanning tabletop, bimanual, and mobile manipulation tasks, designed for evaluating and training code-generating multimodal embodied agents.

2. CaP-Bench, a systematic benchmark that measures robot control performance across tasks and levels of primitives and modalities.

3. Two approaches to improving performance: CaP-Agent0, a training-free, agentic scaffolding approach comprising a combination of multi-turn visual differencing, ensembled reasoning, and automatic skill library synthesis; CaP-RL, reinforcement learning on the coding agent via environment reward.

10.2 CaP-Gym

CaP-Gym is a hierarchical control framework built on the standard **Gymnasium** interface [351]. It binds a Low-Level Environment loop (a physics simulator or the real world) with a stateful Code Executor loop. Architecturally, CaP-Gym preserves the native dynamics of underlying simulators, e.g., Robosuite [256], LIBERO-PRO [258, 344], and BEHAVIOR [259], while exposing them through a Read-Eval-Print Loop (REPL) paradigm tailored for coding agents. In this design, a single code environment "turn" corresponds to the full execution of a generated code program, which may trigger multiple underlying simulation steps depending on the logic invoked.

Low-Level Perception and Control Primitives

All computationally intensive perception and control primitives are implemented as stateless services [352], enabling high-throughput parallel evaluation.

Perception Primitives Agents access perceptual data from the environment through modular perception primitives that abstract raw sensor data into structured semantic objects, e.g., SAM3 [353] for language-conditioned segmentation and Molmo 2 [354] for open-vocabulary pointing, alongside standard vision libraries like OpenCV [355] and Open3D [356].

Control Primitives Instead of directly emitting joint-space action commands, agents call motion planners or inverse kinematics solvers such as PyRoki [357]. They can handle collision checking, reachability constraints, and action-space transformations, allowing agents to reason in a task-oriented Cartesian space while delegating execution feasibility to the controller.

10.3 CaP-Bench: Evaluating Frontier Models

Models. We evaluate 12 state-of-the-art vision-language and language models, including closed-source frontier models (Gemini-3-Pro [358], OpenAI GPT o1 [61], o4-mini [359], 5.1 [360] and 5.2 [361], and Claude Haiku 4.5 [362] and Opus 4.5 [363], and open source models (OpenAI GPT-OSS-20B and 120B [364], Qwen3 235B [365], Qwen-2.5-Coder-7B-Instruct, Kimi K2 Instruct [366], and DeepSeek-V3.1-Terminus [63]).

Task Suite. Primary analysis is performed across 7 core tasks ranging from single-arm manipulation to bimanual coordination: *Cube Lift, Cube Stack, Spill Wipe, Peg Insertion, Cube Re-stack, Two-Arm Lift,* and *Two-Arm Handover*. Each task is evaluated with 100 trials per tier. In Section 10.4, we further extend this analysis to diverse long-horizon settings using tasks from LIBERO-PRO [344] and BEHAVIOR [259] (see Section 10.4).

Protocol. We evaluate models using a **Zero-Shot Pass@1** protocol. Within a trial, agents may interact with the environment over one or multiple turns, observing execution feedback and generating subsequent code to recover from errors or extract additional information; the environment is never reset during the trial. Unlike evaluation settings that permit

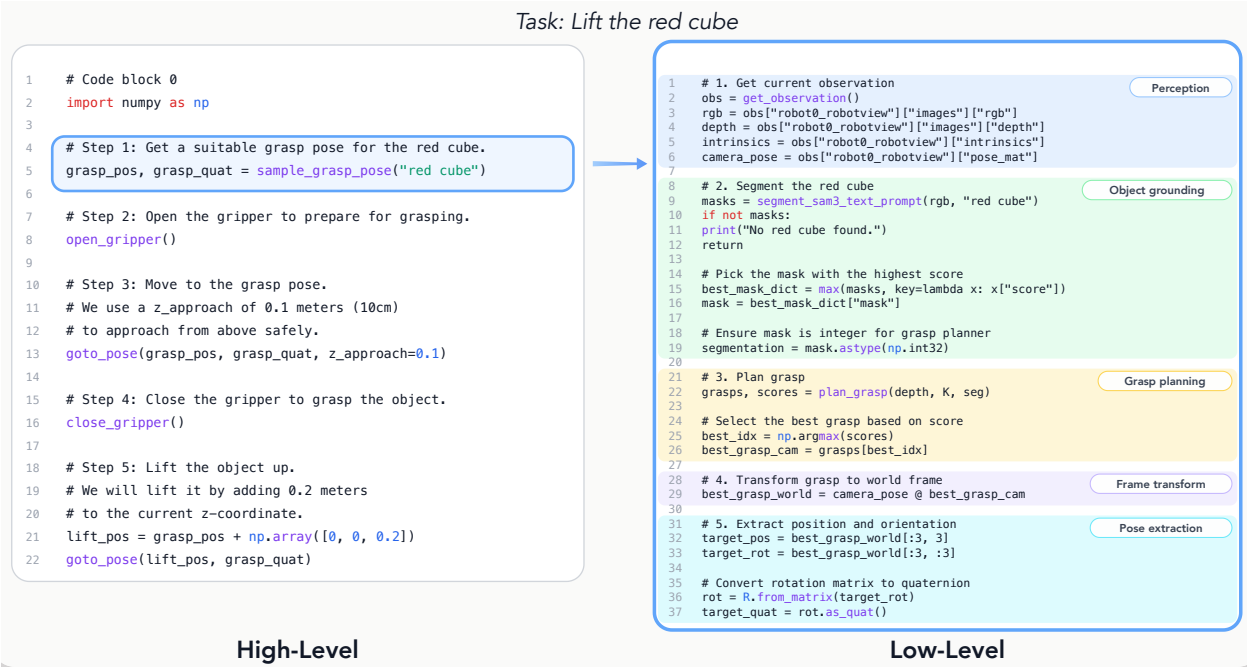


Figure 10.2: (Left) An example of code generated by Gemini-3-Pro for completing the task *"lift the red cube"* using the high-level primitives. **(Right)** Code generated by Gemini-3-Pro using just low-level primitives to achieve the same functionality as a single high-level primitive. For more details on the differences between high and low-level primitives, see Section 10.3 and Appendix E.6.

retries through environment resets, each task instance allows only one continuous interaction episode. Coding agent task performance is compared against that of human expert-written reference solutions under identical environments and primitives. We introduce 4 single-turn tiers (S1-S4) and 4 multi-turn tiers (M1-M4) to CaP-Bench. Please refer to Table 10.1 for a tabular comparison.

Single-Turn Benchmarks (S1-S4)

High-Level (S1 & S2): These tiers evaluate the agent's ability to reason with human-designed primitives. We evaluate this in two modes: **Privileged (S1)**, which uses ground-truth simulation state (masks and object poses), and **Non-Privileged (S2)**, which relies on real perception modules processing raw RGB-D inputs–the default setting for most prior work. We introduce S1 to disentangle high-level planning from perception noise, establishing a reasoning upper bound that allows us to distinguish between algorithmic failures and visual estimation errors.

Low-Level (S3 & S4): In these tiers, human-designed abstractions are replaced by their constituent low-level primitives (e.g., `solve_ik()`, `sam3_text_prompt()`), drawn directly from the APIs of each underlying package—reflecting the interfaces that human developers

use to control robots. We evaluate two settings: **(S3)**, where documentation includes usage examples to scaffold low-level composition, and **(S4)**, where examples are removed and the agent must reason about program structure solely from interface definitions (function signatures and docstrings). See Figure 10.2 for an illustration of high-level (S1 & S2) versus low-level (S3 & S4) primitives; full API details at each abstraction level are provided in Appendix E.6.

Multi-turn Benchmarks (M1-M4)

Text-Only Multi-turn (M1): In this setting, the agent receives the standard output (`stdout`) and error traces (`stderr`) from the Python sandbox after each execution turn. This enables a state introspection loop: agents can proactively inject diagnostic print statements to surface hidden symbolic variables (e.g., perception estimates) and utilize these traces to diagnose logical failures and refine code without access to visual ground truth. All other multiturn tiers (M2-M4) retain access to these code execution traces.

Multimodal (M2): The environment pipes the current RGB observation back into the agent's context window. The M2 tier is only available to multimodal foundation models that accept raw RGB images as input.

Visual Differencing Module (M3): We introduce the Visual Differencing Module (VDM), which uses a vision–language model to convert visual observations into structured natural language. VDM is provided with the task instruction alongside visual observations. In the first turn, it generates a scene description and extracts task-relevant visual attributes. In subsequent turns, it explicitly describes differences between the previous and current image observations and whether the coding agent has completed the task. The resulting text from the VDM is provided as part of the coding agent's observation context for code generation.

Low-Level with VDM (M4): This tier has the same VDM as tier M3 and has access to the same low-level primitives and in-context usage examples as tier S3.

Discussion

Takeaway 1: A Significant Gap Persists Between Frontier Models and Human Experts in Single-Turn Evaluation. We benchmark open- and closed-source agents against human expert solutions under an identical set of perception and control primitives in a single-turn, zero-shot setting (S4). As shown in Figure 10.1, while closed-source models consistently outperform open-source alternatives and newer architectures exhibit stronger capabilities, none yet match the success rate of human-crafted programs in a zero-shot Pass@1 setting.

Takeaway 2: High-Level Abstractions Boost Performance but Limit Expressivity. Figure 10.3 shows a monotonic increase in task success as primitive abstraction increases, mirroring how prior Code-as-Policies [89] approaches relying on high-level primitives report strong zero-shot performance. By collapsing low-level perception, geometric reasoning, and

control into human-designed primitives, these abstractions reduce the effective search space and allow models to focus on task sequencing.

However, this gain comes at the cost of expressivity. As abstraction increases, the agent's action space is increasingly constrained by human priors, imposing a generality ceiling that masks failures in low-level reasoning. In contrast, performance degradation at lower abstraction levels (S3/S4) reflects the difficulty of code synthesis, while enabling expressive behaviors—such as hierarchical perception fallback strategies (Appendix Section E.5)—that cannot be represented by fixed high-level primitives.

This observation motivates a scalable middle ground: rather than relying on human-designed abstractions, agents should be able to recover structure from low-level primitives themselves. In Section 10.4, we demonstrate this capability by enabling agents to distill successful execution traces into a reusable skill library. Consequently, we propose that generalist embodied coding agents be evaluated primarily on primitive-level performance, ensuring that success stems from robust reasoning rather than from the inductive biases of an over-engineered, often task-specific, API set.

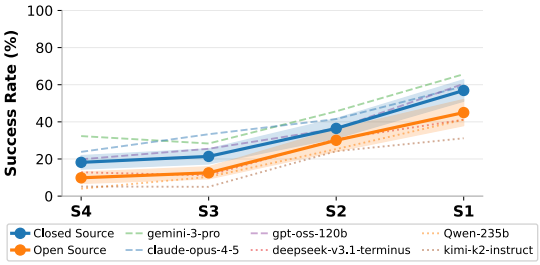


Figure 10.3: Average task success rate across open-source and closed-source models as primitive abstraction increases. As primitive abstraction increases S4 (per model performance illustrated in Figure 10.1) to S1 (high level primitives with full state observation), the success rate increases. This can only in part be attributed to reduced code correctness at S3-S4, see Figure 10.4.

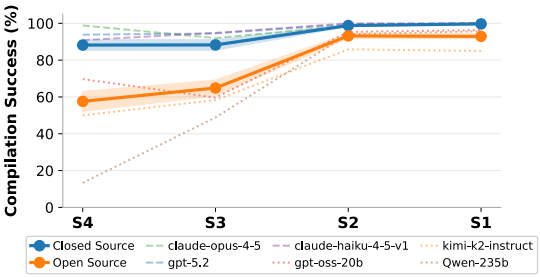


Figure 10.4: Code Execution Success Rate vs primitive abstraction.

Takeaway 3: Closing the Loop with Multi-turn and Visual Grounding Improves Performance. We study how *multi-turn interaction* and different forms of *visual grounding* mitigate the performance gaps identified in Takeaways 1 and 2.

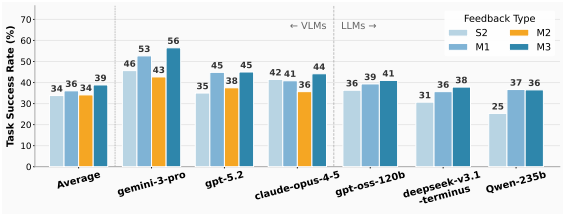


Figure 10.5: Comparison of single-turn (S2) and multi-turn tiers (M1-M3) across models. Enabling textual feedback through multi-turn (M1) improves task success rate in most models. Direct multimodal (M2) visual grounding reduced task success rate. We find that visual differencing into text (M3) instead of direct multimodal visual input consistently improves task success rate across both close and open source models.

Allowing agents to iterate and inspect their own execution traces (`stdout/stderr`, M1) consistently improves performance across all models (Figure 10.5), highlighting the importance of explicit execution feedback for debugging and recovery.

Counter-intuitively, directly interleaving raw RGB observations at each turn (M2) degrades performance relative to the text-only M1 baseline. We hypothesize this degradation is due to a cross-modal alignment gap: foundation models are rarely trained to jointly reason over software coding and images of physical task execution, making raw visual inputs difficult to integrate effectively during code synthesis.

Prior work [367, 368] similarly observes that textually grounded feedback outperforms raw images, though primarily in structured environments with native text states. In contrast, robotic manipulation operates in unstructured, continuous settings without ground-truth textual descriptions. Here, the Visual Differencing Module (M3) bridges this gap by converting visual observations into structured natural language, substantially outperforming both naive image interleaving (M2) and execution-only feedback (M1) across all tasks (Figure 10.5).

Low-Level Primitives with Multi-turn Feedback (M4). Figure 10.6 shows that agents operating over low-level primitives augmented with multi-turn feedback not only surpass high-level single-turn (S2) but can reach parity with high-level multi-turn performance (M3). This supports a *test-time compute scaling* hypothesis: robustness can be synthesized at runtime by increasing an agent's capacity for reasoning, verification, and self-correction over atomic primitives.

Across all multi-turn settings (M1-M4), both coding errors (e.g., exceptions) and physical execution failures (e.g., unstable grasps) are frequently recoverable through iterative interaction. In the absence of explicit visual grounding mechanisms (M1), successful agents compensate by proactively instrumenting perception primitives to expose symbolic state–such as object poses–and performing explicit checks for task completion (e.g., verifying

relative object heights to confirm stacking). Visual grounding (M2-M4) reduces the burden of such self-instrumentation but does not eliminate explicit state verification behavior through perception primitives. Instead, the strongest performance emerges when agents combine structured feedback with iterative reasoning, framing multi-turn interaction as a mechanism for hypothesis testing and error recovery in embodied control.

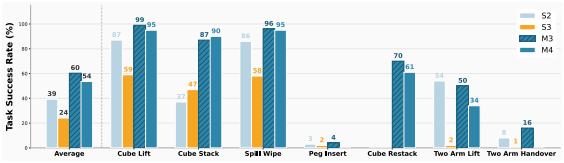


Figure 10.6: Evaluation of Gemini-3-Pro across four tiers of the benchmark. Multi-turn with visual differencing and low-level API (M4) significantly outperforms both single-turn low-level API (S3), and more notably, single-turn high-level API (S2).

10.4 CaP-Agent0: An Agentic Framework for Robot Control

Based on the failure modes and insights identified in CaP-Bench, we introduce **CaP-Agent0**, a training-free, agentic infrastructure for robot control that augments base foundation models with a specialized multi-turn reasoning loop and a dynamically synthesized skill library. The architecture of CaP-Agent0 is designed directly to address three key gaps identified in the benchmark. We demonstrate the efficacy of each design choice in CaP-Agent0 by running ablation studies using the most capable model identified from CaP-Bench (Gemini-3-Pro) bootstrapped with each design choice, shown in Figure 10.8. A visual walkthrough of the agentic framework is presented in Figure 10.7.

1. Multi-turn Visual Differencing (VDM): Directly adopting the insights from Takeaway 3, CaP-Agent0 integrates the Visual Differencing Module as part of the per-turn observation. By grounding observations in structured text rather than raw pixels, the agent mitigates the specific cross-modal alignment failures identified in the M2 tier.

2. Auto-Synthesized and Persistent Skill Libraries: In analyzing low-level S3 and S4 code generations from CaP-Bench, we found that capable models routinely synthesize auxiliary utility functions to perform robotics data manipulations.

Motivated by agentic systems that accumulate reusable tools over time [83], **CaP-Agent0** introduces an *automatically synthesized, task-agnostic skill library* that persists across trials. Rather than requiring the agent to repeatedly re-derive low-level utilities, the library onboards commonly recurring implementation patterns and offloads fragile low-level logic, allowing the coding agent to focus on high-level semantic planning. Importantly, unlike fixed human-designed high-level APIs, these skills are *discovered*: they emerge from successful executions and retain the expressivity of low-level interfaces while improving robustness through reuse.

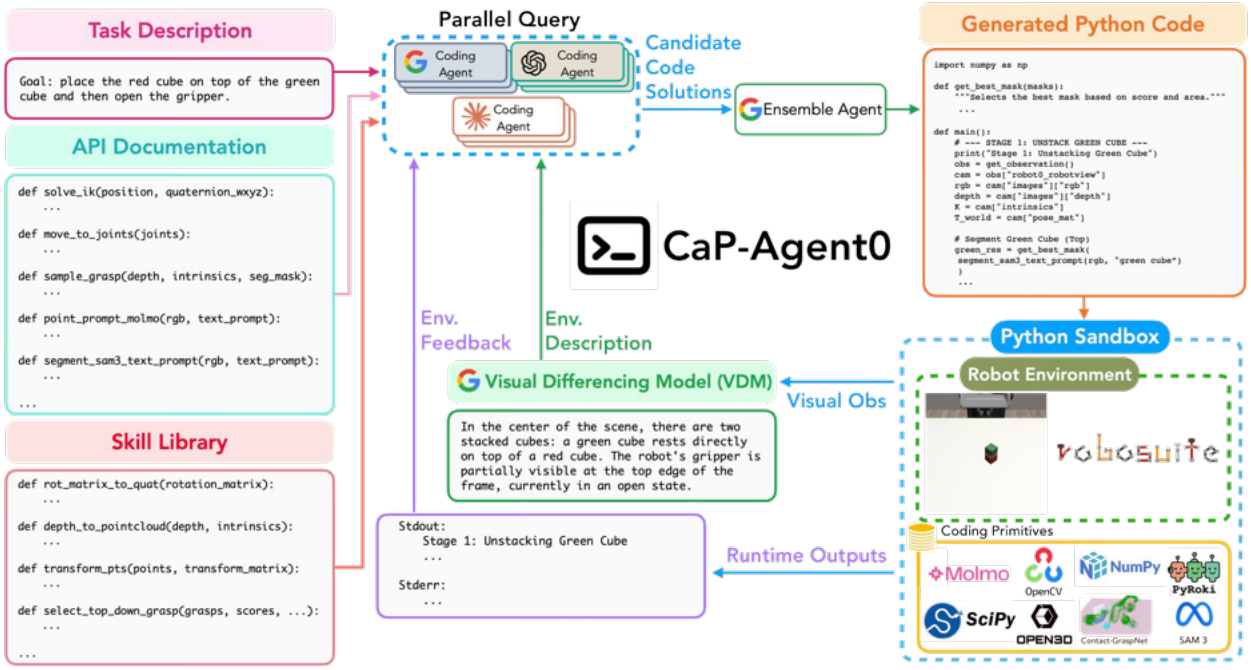


Figure 10.7: CaP-Agent0. CaP-Agent0 incorporates an auto-synthesized skill library of auxiliary utility generated by coding agents during CaP-Bench, a visual differencing model (VDM) which provides a textual description of the initial scene and for each subsequent turn what changes have occurred in the scene since, and a parallel reasoning system where multiple coding agents are provided the same prompt. These coding agents then generate candidate code solutions that may solve the task and the ensemble agent must then synthesize these code generations into a final code snippet which is then executed in the Python sandbox containing the robot environment, whether that environment may be a simulator like Robosuite or on a real robot. For more details see Section 10.4.

The library is constructed via an automated synthesis pipeline that can be executed by the agent itself. Code outputs from *successful rollouts* in tier S3 are collected, and function definitions are extracted using regular-expression matching. An LLM is then queried to perform an analysis over these extracted functions to identify frequently recurring, task-agnostic logic, which is isolated and promoted into reusable skills. While the current implementation performs a single synthesis pass, the process is inherently *iterative*. As additional successful executions are accumulated, the agent can continue to bootstrap, refine, or prune its skill library over time. Furthermore, depending on the desired generality, researchers can modulate the filtering leniency to allow for more task-specific behaviors or maintain a strictly task-agnostic core. A complete list of the nine synthesized functions currently utilized by CaP-Agent0 is provided in Appendix E.7.

3. Parallel Reasoning: Results from tiers S2 and S3 indicate that failures often stem from insufficient test-time exploration rather than a lack of capability. To address this, CaP-

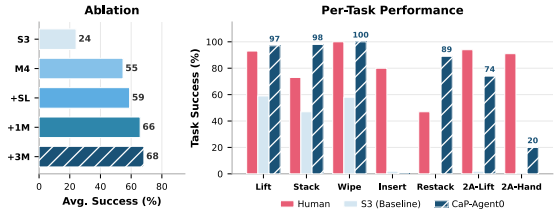


Figure 10.8: Ablation for CaP-Agent0. (Left): Combining VDM (M4), skill library (+SL), and parallel queries (+1M: only Gemini-3-Pro, +3M: Gemini-3-Pro, GPT-5.2, and Claude Opus) significantly improves over single-turn setup with low-level API. (Right): On 4 out of 7 CaP-Bench tasks, CaP-Agent0 achieves comparable or better success rates than human expert code in a single-turn setting.

Agent0 employs a parallel reasoning strategy inspired by recent ensemble methods [369–371]. At each turn, the system concurrently samples candidate solutions via two configurations: Single-model: 9 queries to one model. Multi-model: 3 queries each to GPT-5.2, Claude Opus 4.5, and Gemini-3-Pro. To maximize output diversity, we vary sampling temperatures (see Appendix E.7). A central coding agent then synthesizes these candidates into a final code snippet. This parallel approach is applied to both code generation and the decision-making process for multi-turn continuations.

CaP-Agent0 Performance on CaP-Bench

Evaluated over 100 trials per task, CaP-Agent0 significantly outperforms single-turn baselines by integrating visual differencing, a self-synthesized skill library, and parallel reasoning (Figure 10.8). Despite operating solely on low-level primitives, the system achieves success rates comparable to or exceeding human-written programs on 4 out of 7 tasks, narrowing the gap toward expert-level performance.

CaP-Bench++

We further evaluate the performance of CaP-Agent0 on subsets of tasks from LIBERO-PRO and BEHAVIOR. We summarize the results of CaP-Agent0 on 30 manipulation tasks from LIBERO-PRO in Table 10.2 and compare them against three state-of-the-art VLA methods: OpenVLA [151], π_0 [205], and $\pi_{0.5}$ [53]. Although these VLA methods are training-based, CaP-Agent0 is a training-free method that has comparable or exceeds the performance of VLA post-training on these tasks. LIBERO-PRO [344] extends the LIBERO [258] benchmark by increasing the perturbations of tasks by reasonable amounts along object attribute perturbations, initial position perturbations, instruction perturbations, and environmental perturbations. CaP-Agent0 and these VLA methods are evaluated along initial position perturbations (Pos) and instruction perturbations (Task). Under Pos perturbations, the initial positions of objects in the scene are swapped with one another (ex: the position of

Table 10.2: LIBERO-PRO [344] performance of OpenVLA [151], π_0 [205], $\pi_{0.5}$ [53] and CaP-Agent0 on the **libero-object**, **libero-goal**, and **libero-spatial** benchmarks under initial position perturbations (Pos) and instruction perturbations (Task) averaged across tasks. Detailed individual task-wise performance can be found in Appendix Section E.9.

Method	libero-object		libero-goal		libero-spatial	
	Pos (Avg.)	Task (Avg.)	Pos (Avg.)	Task (Avg.)	Pos (Avg.)	Task (Avg.)
OpenVLA	0.00	0.00	0.00	0.00	0.00	0.00
π_0	0.00	0.00	0.00	0.00	0.00	0.00
$\pi_{0.5}$	0.17	0.01	**0.38**	0.00	**0.20**	0.01
CaP-Agent0	**0.22**	**0.18**	0.26	**0.17**	0.12	**0.14**

Table 10.3: Results on BEHAVIOR [259] Tasks

Task	Nav. Success Rate			Task. Success Rate		
	Human	S3	CaP-Agent0	Human	S3	CaP-Agent0
Pick up Radio	**88%**	72%	80%	36%	24%	**56%**
Pick up Soda Can	80%	52%	**84%**	**72%**	32%	**72%**

the frypan and moka pot) and under Task perturbations, the instruction is changed so that another object in the scene is manipulated (ex: "Put the moka pot on the stove" $\rightarrow$ "Put the frypan on the stove"). Since VLAs are trained on a different instruction distribution, they perform poorly under task perturbations, whereas CaP-Agent0 remains robust to instruction variations. Furthermore, CaP-Agent0 achieves performance comparable to $\pi_{0.5}$ under initial position perturbations.

We evaluate our method on two long-horizon mobile manipulation tasks in BEHAVIOR, where an R1Pro wheel-based humanoid is required to pick up a radio from a table and a soda can from the floor. The results over 25 trials for each task are summarized in Table 10.3, reporting both navigation success rates and task completion success rates. In both tasks, the robot may start with the target object outside its field of view and must actively search for and navigate toward it. Navigation is considered successful if the robot reaches a location within 1 m of the target.

The robot can adjust its camera in both horizontal and vertical directions and move both its base and arm to reach the object, resulting in a substantially larger action space than in tabletop manipulation settings. Moreover, potential collisions with surrounding furniture may prevent the robot from reaching the desired pose, leading to partial execution of planned trajectories and increased task complexity.

In the radio pickup task, although the robot often successfully locates and approaches the object, it may lose sight of it or encounter severe occlusions when navigating too closely,

due to its limited field of view. This frequently results in missing or poor grasp poses and constitutes a major failure mode for both S3 and the human policy. In contrast, CaP-Agent0 mitigates this issue by repositioning the robot to obtain a better view, achieving substantially higher success rates. For the soda can pickup task, the small object size makes accurate vertical camera alignment critical for localization. S3 often fails by adjusting the camera vertically too early, leading to unsuccessful searches within the time limit. CaP-Agent0 adapts its search strategy based on feedback, improving navigation success. Furthermore, grasping may fail when the can is knocked over, further reducing S3's completion rate. CaP-Agent0 can resample grasp poses after such disturbances and successfully recover, resulting in significantly higher task success and similar performance as human expert.

Let the Agent Experience the Real World

The design of the CaP-Gym environment loop deliberately allows it to directly interface with real-world robot perception and control interfaces, and we additionally demonstrate zero-shot performance of CaP-Agent0 in unseen real-world tasks on real robot embodiments including the Franka Panda and AgiBot G1 without requiring any major cross-embodiment modifications (with the exception of single arm to bimanual control primitive modifications). With no post-training, off-the-shelf VLMs such as Gemini-3-Pro and Claude Opus 4.5 can perform complex long-horizon robotics reasoning and manipulation tasks following natural language instructions. For example, when asked to "find the object under one of the cups" (Figure E.4), it mechanically searches through all cups with closed-loop feedback from vision. When the robot is asked to solve a math problem presented in the physical world (Figure E.5), it perceives the visual cue, thinks, and selects the correct blocks on the first attempt. We also demonstrate embodied reasoning ability in Figure E.7, where CaP-Agent0 exhibits common sense physics reasoning and understands the sensible stacking order for objects of various shapes. Optionally, a human-in-the-loop can also interactively correct the robot's behavior by providing additional feedback in between turns. More details are documented in Appendix E.1.

10.5 CaP-RL

CaP-Gym enables on-policy reinforcement learning with verifiable rewards (RLVR) directly on the coding agent. To demonstrate this, we apply Group Relative Policy Optimization (GRPO) [60, 62] to post-train a Qwen2.5-Coder-7B-Instruct base model [372].

Methodology. We RL post-train on three tasks: *Cube Lift, Cube Stack,* and *Spill Wipe*. To ensure stable convergence, we train using the privileged state-based APIs of tier S1. This avoids the noisy reward signals present in tier S2, where compounding perception and control errors can cause otherwise correct programs to fail during execution, introducing credit assignment ambiguity similar to that observed in G1 [373].

Table 10.4: Impact of RL Post-Training in Sim and Real. Comparison of success rates between the base model, our RL post-trained agent, and human experts. Simulation results are averaged over 100 trials per task, while real-world deployment on a Franka Emika robot is evaluated over 25 trials.

	Simulation (N=100)			Real World (N=25)	
Method	**Cube Lift**	**Cube Stack**	**Spill Wipe**	**Cube Lift**	**Cube Stack**
Human Expert	93%	73%	100%	92%	84%
Qwen 2.5 Coder 7B	25%	4%	30%	24%	12%
Qwen w/ CaP-RL	**80%**	**44%**	**93%**	**84%**	**76%**

Simulation Results. Post-training for 50 iterations per task significantly improves code compilation rates and strategic robustness. When evaluated on S2 (noisy perception), the RL post-trained model achieves substantial gains over the base model, as detailed in Table 10.4.

Sim-to-Real Transfer. A key advantage of CaP-RL is the minimal Sim-to-Real gap. Because the agent is optimized to reason over abstract perception APIs rather than raw visual features, the learned policy transfers zero-shot to real-world embodiments without having to contend with a visual gap. We validate this on a Franka Emika robot, where the agent retains high success rates for cube lifting (84%) and stacking (76%), demonstrating that improved strategies learned in simulation remain robust under real-world perception noise. Please refer to Appendix Section E.4 for comparing code generation before and after RL post-training.

10.6 Related Work

Code as Policies. A growing line of work explores programmatic robot control, where LLMs generate executable code that orchestrates perception and control modules [80, 82, 84, 374–377]. In robotics, this paradigm spans grounding LLM plans in physical affordances [91], composing APIs for closed-loop robot behaviors [89, 90, 378], generating Python programs over perception APIs [379–382], and building embodied reasoning agents [47, 383, 384]. Structured intermediate representations such as Planning Domain Definition Language (PDDL) and Signal Temporal Logic (STL) [385, 386] and persistent state tracking [387] further improve plan reliability, while executable code has been shown empirically to be a superior agent action representation [86]. Recent work scales to multi-robot coordination [388], grounded planning over 3D scene graphs [389], and modular vision-language agentic pipelines [92, 94]. Relatedly, Gemini Robotics [390] highlights hybrid systems integrating tool calling with sensorimotor policies. Across all these settings, agents benefit from iterative self-refinement [81, 391–393] and from sampling multiple solutions at inference time [76, 77, 79, 394]; platforms such as OpenHands [85] realize this as software engineering agents that

write, execute, and debug code in sandboxed environments. Despite this progress, much existing work on coding agents for robot control typically relies on high-level, human-crafted APIs that encode significant task structure, leaving open the question of how well coding agents perform when operating over low-level primitives closer to how robots are actually programmed by humans. CaP-Gym targets this gap by providing an interactive environment that exposes the full primitive stack—down to joint-level perception and control—and CaP-Bench systematically evaluates frontier models across abstraction tiers, enabling principled study of agentic inference-time strategies (multi-turn interaction, execution feedback, visual differencing, ensembled reasoning) as a path toward robotic coding agents.

Skill Synthesis and RL with LLM-Generated Code. A large body of work uses LLMs as *static* code generators for reward functions [395–398], environment curricula [399, 400], domain randomization [401], and intrinsic exploration goals [402], or to synthesize environments and skills autonomously [403, 404]—in all cases the LLM is frozen while a separate downstream policy is trained. Complementarily, RL with verifiable rewards (RLVR) has emerged as a powerful paradigm for improving the language model itself, with strong results across reasoning [60–62], code generation [405, 406], and interactive agentic settings spanning software engineering [88, 407], web interaction [408], and tool use [409–413]. CaP-RL extends RLVR to robot manipulation: rather than using the LLM to produce reward code for a separate policy, we directly fine-tune the language model via GRPO using execution outcomes from physics simulation.

Benchmarks for robotics, code generation, and embodied agents. Robotic manipulation benchmarks [256–259, 414–417] provide diverse tasks and standardized simulation assets, but typically evaluate fixed policy interfaces rather than executable program synthesis with tiered APIs and multi-turn debugging. Software engineering and code generation benchmarks [19, 85, 87, 88, 394] measure program synthesis in purely digital environments, without embodied perception or physical constraints. Embodied-agent benchmarks [368, 373, 418–420] broaden evaluation to multimodal interactive environments, but do not require agents to generate executable robot-control code under varying abstraction tiers. CaP-Gym targets this intersection: evaluating coding agents on embodied manipulation with verifiable success signals, and explicitly probing how abstraction, multi-turn recovery, and multimodal grounding affect performance.

10.7 Future Works and Conclusion

Programmatic control performs well on long-horizon, reasoning-heavy tasks, but remains brittle for contact-rich behaviors that require tight visual servoing and continuous feedback (e.g., insertion or pouring). One promising direction is hybrid CaP–VLA policies, in which a coding agent manages high-level task logic and recovery while deferring low-level execution to VLA policies. Results from CaP-Bench further highlight several avenues for improving language-model-based agents, including stronger embodiment-aware planning and reasoning, more effective grounding of task-relevant visual information into code generation, improved

test-time search, and agent or prompt optimization methods [421]. From a robotics perspective, robustness may further improve by incorporating optimization-based control primitives that allow agents to specify task-level constraints and account for collision avoidance during motion planning, rather than relying solely on inverse kinematics solutions that may be suboptimal when directly interpolated in joint space. Expanding CaP-Gym to additional environments and classical robotics problems (e.g., active and interactive perception) would further stress-test agentic reasoning.

We introduce CaP-X, a unified framework for benchmarking and improving coding agents for robot control. CaP-X consists of CaP-Gym, CaP-Bench, CaP-Agent0, and CaP-RL, enabling controlled evaluation across abstraction levels, interaction modes, and learning paradigms. CaP-X frames robot control as a problem of machine intelligence—where agent design, inference-time computation, perception, and control are co-studied—providing a testbed for evaluating and advancing general-purpose embodied intelligence.

Part IV

Conclusion

Chapter 11

Conclusion

11.1 Summary of the Dissertation

This dissertation has studied a central question: how can robot learning become efficient and scalable enough to support broadly useful physical intelligence? The question is motivated by a persistent gap between robotics and the rest of modern AI. In vision and language, capability has improved rapidly as models, data, and self-improving training procedures have scaled. In robotics, however, various obstacles prevent the transfer from happening smoothly. Robot data is expensive, physical interaction is slow, safety and reset are nontrivial, and the behaviors we ultimately care about require contact, long horizons, deployment on various robot embodiments, and interaction with the unstructured real world. This dissertation therefore approached scalability not as a single technical problem, but as a set of coupled questions: what should robot models be trained to predict, where can robot data come from, and how should learned policies be composed with the broader reasoning and tool-use capabilities of foundation models.

Part I addressed the model side of this question. It asked whether robot trajectories themselves can become the pre-training corpus for embodied intelligence. Rather than treating imitation learning as a supervised mapping from the current observation to the next action, the chapters in this part treated robot behavior as a sequence modeling problem over vision, proprioception, language, and action. RPT (Chapter 3) studied masked prediction over interleaved sensorimotor trajectories, showing that a model can learn transferable representations by reconstructing missing states and actions from context. ICRT (Chapter 4) then pushed this idea toward in-context robot learning: by formatting demonstrations and queries as long sensorimotor sequences, an autoregressive model could infer a new task from one or a few examples without gradient updates. OTTER (Chapter 5) addressed the complementary problem of language grounding. It showed that a VLA instantiated with a sequence model can preserve the open-vocabulary semantics of a vision-language backbone when visual features are extracted in a text-aware manner from a frozen CLIP-style encoder. Together, these chapters argue that the trajectory, rather than the isolated state-action pair,

is the right modeling unit for scalable robot learning. They also suggest that pre-training, history, language grounding, and in-context adaptation should not be treated as separate mechanisms, but as different views of the same sequence modeling problem.

Part II turned from model scaling to data scaling. Model capability scales with data. However, the largest public robot datasets remain tiny compared to the corpora used to train frontier language and vision models, and collecting all required demonstrations through teleoperation alone is unlikely to be physically feasible. This part therefore studied alternatives to direct human teleoperation. Chapter 7 explored self-supervised data collection on real hardware, using tactile and force feedback to identify reusable grasp and insertion behaviors. Chapter 8 explored a different route: Real2Render2Real, where a small amount of real-world input is converted into a large synthetic dataset through rendering and randomization, enabling policies trained on generated demonstrations to approach the performance of policies trained on human teleoperation. These two chapters represent different directions in the design space. One collects real interaction data with less human supervision; the other converts limited human input into much larger synthetic corpora. The shared message is that scalable robot learning requires treating data generation itself as a direction for algorithmic exploration to, in turn, help the advancement of trained policies.

Part III studied a third axis of scalability: composition, reasoning, and self-improvement through code. Many robot learning systems currently try to solve all control problems by training a single end-to-end policy. Yet humans rarely program robots this way. We combine perception modules, control primitives, geometry, simulation, debugging, and task-level reasoning. Chapter 10 introduced CaP-X to study whether foundation-model-based coding agents can use this kind of modular structure for robot control. CaP-Gym provides a unified environment where agents act by writing executable Python code over perception and control primitives. CaP-Bench measures how models perform under different levels of API abstraction, feedback, and visual grounding. CaP-Agent0 shows that a training-free agentic harness can substantially improve performance by using execution feedback, visual differencing, skill libraries, and parallel multi-model reasoning. CaP-RL then shows that this substrate can also support reinforcement learning with verifiable robot rewards, improving a coding model through interaction and transferring the resulting behavior to real hardware. The conclusion of this part is not that coding agents replace learned policies. Rather, it is that code provides a powerful interface for combining learned perception, classical control, simulation, environment feedback, and language-model reasoning. It gives robot systems a way to inherit capabilities from many different strategies at once.

The aforementioned three parts are not intended to be competing answers. They are complementary ingredients for efficient and scalable robot learning. Scalable robot learning will likely require trajectory pre-training, scalable non-teleoperation data generation, and agentic systems that can compose learned policies with perception, control, simulation, and code.

11.2 Future Research Directions

History, subgoals, and world models. The first open question is how robot models should use history. Part I argues that trajectories are the right object of modeling, but it does not fully settle what should be stored in context, what should be compressed, and what should be predicted. Recent VLA systems have made different design choices. $\pi_{0.5}$ predicts explicit subtasks, while $\pi_{0.7}$ predicts subgoal images [53, 422]. These approaches suggest that long-horizon robot control may require intermediate representations between low-level actions and high-level language. A robot may not need to remember every frame and action from the past, but it may need to infer what phase of the task it is in, what object state it is trying to reach, and what subgoal should come next.

Additionally, there is the approach of using world models. In robotics, the phrase "world model" often suggests post-training of pixel prediction or video prediction models for action prediction, but it is not obvious that predicting pixels is always the right pre-training objective for control. Pixel prediction may help when it forces the model to learn object permanence, physical dynamics, contact outcomes, and the consequences of its own actions. It may be less useful when the model spends capacity reconstructing irrelevant texture, lighting, or background detail. Similar to MaskDP [423], RPT provides one way to think about this question. Because its masked-prediction objective can inpaint either actions from states or states from actions, it can be viewed as an implicit world model over sensorimotor trajectories. The important question is therefore not simply whether a robot policy should have a world model, but what kind of prediction target makes the model more useful for downstream control. Future work should systematically vary pre-training objective targets, subgoal representations, and context lengths to understand when VLA-style or world-model-style pre-training improves policy learning.

Scaling data beyond teleoperation. The second open question is how to scale robot data beyond direct human demonstration. Part II studied self-supervised hardware collection and Real2Render2Real, but the frontier of robot learning is moving toward tasks that stress both approaches. Dexterous manipulation often involves articulated objects, deformable materials, fine contact, occlusion, and shiny or reflective surfaces. These are exactly the settings where pose tracking, reconstruction, and simulation are hardest. Progress in 3D foundation models, including systems such as SAM 3D [424], may reduce the cost of reconstructing objects and scenes, but real-to-sim-to-real pipelines will still need better treatment of contact, articulation, material properties, and deformable dynamics.

At the same time, the field should more carefully compare different non-teleoperation data sources. Real2Render2Real scales from a small amount of robot input through rendering. UMI-style data collection [226] scales through cheaper, more ergonomic human demonstrations. EgoScale-style approaches scale through large egocentric human video [122, 238]. Self-supervised robot systems scale through autonomous interaction. These sources differ in cost, embodiment gap, task coverage, supervision, and compatibility with downstream policy training. A central future direction is to build a more quantitative understanding of their

scaling laws. Which data source improves fastest with human time? Which improves fastest with compute? Which transfers best to dexterous manipulation? Which is best for pre-training, and which is best for post-training? Answering these questions is likely to matter as much as proposing any single new policy architecture.

Agentic robot systems and self-improving loops. The third open question is how to integrate LLMs, coding agents, classical robot control, and VLA policies into a single system. Part III argues that code is a promising interface because it can compose perception, control, geometry, simulation, memory, and feedback. However, the right division of labor remains unclear. A VLA policy may be best for short-horizon visuomotor skills. A coding agent may be best for long-horizon decomposition, tool use, parameter search, and repair. Classical controllers may be best for precise motion, force regulation, stability, and safety. The most capable robot systems will likely not choose one of these paradigms over the others. They will need interfaces that let each component do what it is best at.

This raises the larger question of self-improvement. In language and software engineering, execution feedback and verifiable rewards have become powerful training signals. Robotics has a natural analogue: a robot program can be executed, observed, scored, and debugged. CaP-Agent0 and CaP-RL are early steps in this direction, but much remains open. Can an agent automatically discover reusable skills from past failures? Can it decide when to call a VLA, when to write code, and when to invoke a classical controller? Can it generate its own curriculum? Can it improve both the harness around the model and the model itself? Additionally, during RL post-training, it is challenging to perform credit assignment: the failure may not be because the code is wrong, but some motion primitives or perception primitives have failed. Understanding how to perform proper credit assignment would be crucial for the success of future large scale RLVR for agentic robot control.

11.3 A Broader Perspective

My own path into this field began long before the specific projects in this dissertation. I started this decade-long journey at Berkeley inspired by AlphaGo [23], and by the rapid progress in deep learning and computer vision that made neural networks feel suddenly capable of perceiving the open world. In 2017, some of the first papers that pulled me into AI were Faster R-CNN [425] and Feature Pyramid Networks [426]. At the time, the ability to detect and segment general objects in natural images already felt remarkable. Since then, the field has changed far beyond recognition, detection and segmentation. Vision models became foundation models. Language models became general-purpose interfaces for reasoning, coding, and tool use. Multimodal models began to connect language, images, video, and action. Capabilities that once seemed distant became ordinary, and benchmarks that once felt challenging to humans were repeatedly surpassed.

Looking back, the progress of deep learning has depended mostly on two forces. The first is data: large, diverse, and increasingly well-structured corpora that allow models to learn

broad priors before they are specialized. The second is self-improvement: systems that can generate feedback, evaluate their own behavior, and improve more efficiently than through human annotation alone. AlphaZero [75] made this point through self-play. More recent reasoning models, including DeepSeek-R1 [427], make a related point through reinforcement learning and verifiable outcomes. The history of modern AI is not only the history of larger models. It is the history of finding the right data and the right feedback loops.

Robot learning has followed the broader trajectory of AI, but it cannot simply copy it. Robotics has its own constraints. Data comes from meaningful physical interactions. Actions have consequences. Reset is costly. Contact is hard. Evaluation is slow. A robot cannot passively scrape the internet for manipulation experience. Nor is robotics exactly like autonomous driving. A car has value even when autonomy fails, because a person can still drive it and generate value (get the person from point A to point B). A general-purpose robot that cannot yet perform useful work has much less inherent value. This changes the scaling problem. It forces us to ask: what kinds of robot data are actually worth scaling? Where will the first real tipping points of usefulness occur? Which tasks provide enough value to justify deployment before generality is solved? What form should physical AGI take if it must be both broadly general and locally specialized?

The research in this dissertation is my attempt to make progress on these questions through three complementary philosophies. The first is to make robot models scale with compute and data, by treating trajectories as pre-training sequences rather than isolated examples. The second is to create scalable ways of generating useful robot data without requiring human teleoperation effort. The third is to build agentic systems that can combine the strengths of many approaches: imitation learning, classical control, perception, simulation, coding, tool use, and reinforcement learning. Scaling imitation learning is not incompatible with classical methods. Learned policies are not incompatible with coding agents. Simulation is not a replacement for real data, but it can be a verifier, a generator, and a training ground. The future is unlikely to belong to any single paradigm in isolation. It will depend on finding the right interfaces between them.

This dissertation therefore ends less with a final answer than with a position. Physical intelligence will require models that learn from broad experience, data engines that make such experience scalable, and systems that can improve through interaction with the world. The path is harder than in language or vision because we are bounded by the physical world, but that is also what makes the problem worth solving. I look forward to continuing to pursue this question: how to build robot algorithms that are efficient and scalable, general enough to adapt, specialized enough to be useful, and capable enough to improve through their own experience.

Part V

Appendix

Chapter A

Appendix A: Robot Learning with Sensorimotor Pre-training

A.1 Data Collection

Hardware. We used a Franka robot with a 7-DOF arm and a parallel jaw gripper (Figure A.1, top left). We recorded proprioceptive information in the form of joint positions and velocities. The workspace is equipped with three high-quality Logitech Brio color cameras. One egocentric camera is attached to the robot hand, and two exocentric cameras are attached to the left and the right side of the robot. We synchronized the data streams from the cameras and the robot at 60 Hz and saved data at 30 Hz.

(1) Pick: The task is to grasp and lift one out of K cubes from the table. We script a data generation scheme where the robot takes in K cubes at randomly generated 3 DoF poses in robot frame, that are not in collision, and generates sequences of 3 DoF picks-and-places poses for each of the cubes. Then a scripted policy grasps the cube. After grasping, the robot places the grasped cube in the next randomly generated pose, moves back to the start joint configuration, and starts the next grasp.

(2) Bin Pick: The robot picks an object from a bin filled with randomly-placed, both soft and rigid, objects (see Figure A.1). We use [220] to generate grasps from depth images, captured by an overhead depth camera. Before each grasp, the robot moves to a joint configuration that does not occlude the camera view of the bin. After we obtain a grasp pose, we use a scripted policy to pick the object. If the gripper is not fully closed at the end of the trajectory, we consider the trajectory successful.

(3) Stack and (4) Destack: Starting with two cubes of different colors resting on the table, the robot is tasked to first move one of the cubes onto the other cube, and then move the stacked cube back onto the table. The cube poses are generated in a similar fashion as (1). For stacking, the height of the place pose is offset by the height of the cube; similarly, for destacking, the pick pose height is offset by the height of the cube. A scripted policy then executes the pick-and-place trajectory.

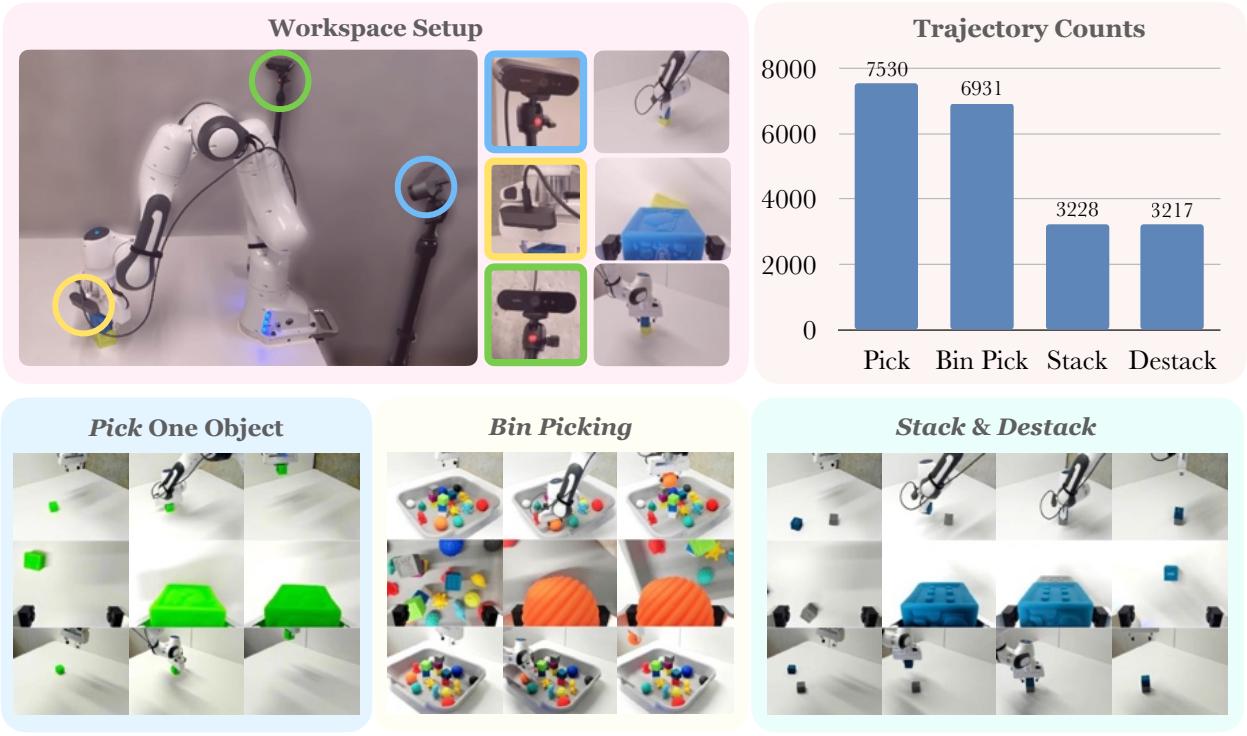


Figure A.1: Dataset. We collected a dataset of over 20,000 real-world trajectories using a Franka robotic arm. Each trajectory is a sequence of high-quality images from three cameras (one attached to the hand and two on the sides), proprioceptive robot states, and actions. We consider picking, bin picking, destacking, and stacking tasks, with variations in object position, shape, and appearance.

Statistics. We collected ~20,000 real-world trajectories over the course of 9 months (Figure A.1). The dataset includes ~7,000 trajectories for both single object and bin picking, as well as ~3,000 trajectories for stacking and destacking each. The average length of picking and destacking trajectories is ~300 while the stacking trajectories are longer at ~600 steps. The dataset contains variations in object poses, shape, and appearance. We show frames from example trajectories in Figure A.1.

Chapter B

Appendix B: In-Context Imitation Learning via Next-Token Prediction

B.1 Supplementary Material

Scene Illustrations

We provide illustrations on the prompt trajectories and test scenes for the pick up the black dog and place in the pink bowl task in Figure B.1. As mentioned in Section 4.5, we collected 3 types of prompt trajectories and test ICRT on 5 tiers of scenes that are different from the scenes in the prompt trajectories.

Ablation Studies

In this section, we provide additional ablation experiments on a few core design choices and different prompting strategies.

Repeatability Experiments

We conduct experiments to evaluate the repeatability of the performance of ICRT. We conduct a pick up the black dog and place in the pink bowl task and a poke blue sponge task for 5 rollouts, where each rollout contains 5 trials as in Section 4.5, resulting in a total of 25 trials. We calculate the average and the standard deviation of the success rate. Results are shown in Table B.1. The low std from Table B.1 suggests that the ICRT can reliably achieve the task.

Prompt Trajectories

We conduct experiments on different prompt types to evaluate the effect of different prompt trajectories on task performance. We consider the task of picking up a black dog and placing

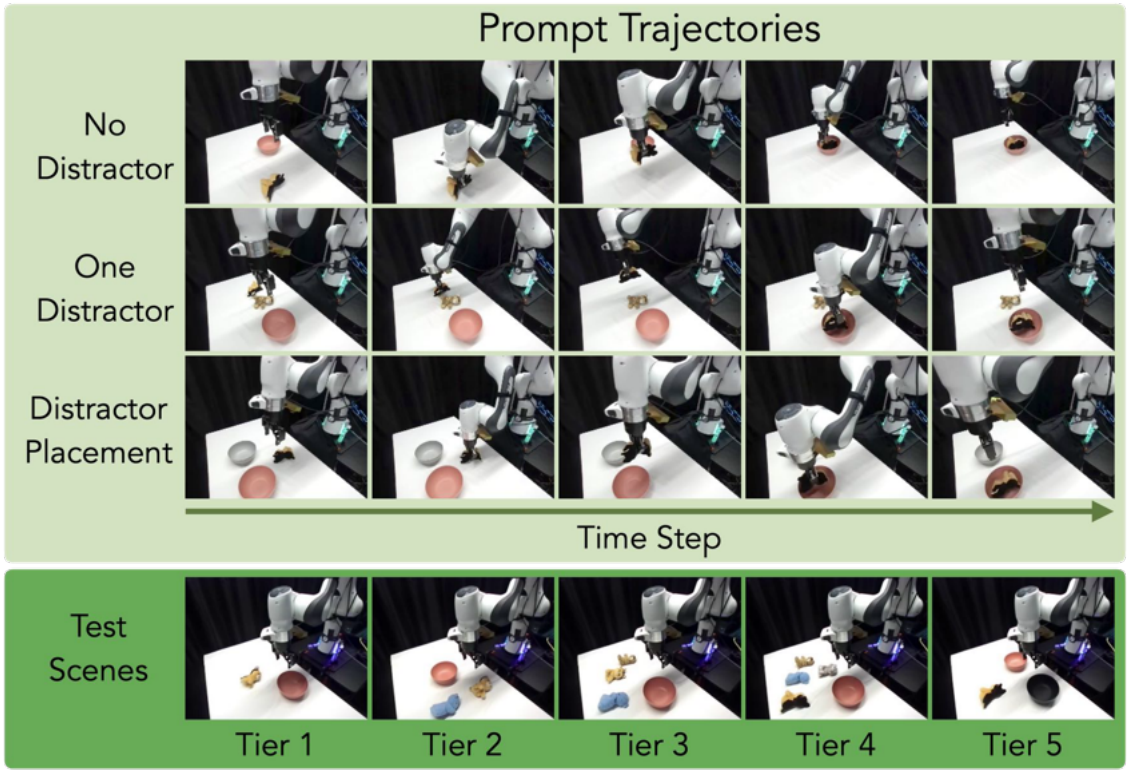


Figure B.1: Illustrations of the prompt trajectories (top) and test scenes (bottom) for the pick up the black dog and place in the pink bowl task. Three prompt trajectories of different types are collected. The test scenes are different from all prompt trajectories and 5 tiers of scenes with different number of distractors are considered.

Task	Pick and Place Block Dog in Pink Bowl	Poke Blue Sponge
Success Rate Ave. ± Std.	60% ± 0.5%	88% ± 3.2%

Table B.1: Repeatability experiments for a pick and place task and a poking task. Each task is conducted by 5 rollouts and each rollout contains 5 trials, resulting in a total of 25 trials.

Prompt Type	No Distractor	One Distractor	Distractor Placement	Two Prompts	Three Prompts
Success Rate	60%	80%	70%	80%	80%

Table B.2: Experiments on different prompt types on a pick up black dog and place in the pink bowl task. The first three columns are results for a single prompt trajectory of different types, while the last two columns are that for using two and three prompts. Success rates are calculated over 5 trials for each experiment.

in a pink bowl. We have three prompt trajectories of different types: one with no distractors, one with one distractor and one with one distractor placement, as shown in Appendix

Figure B.1 top. All three prompt trajectories are collected by a human teleoperating the robot. The object locations and the placement locations at test time are different from that in all three prompts. As in Section 4.5, for each prompt type, we conduct the task with 5 trials as shown in Appendix Figure B.1 bottom. The average success rates are reported in Table B.2. We conduct experiments with one prompt trajectory of different types (the first three columns in Table B.2), two prompt trajectories and three prompt trajectories. All prompt types result in similar performance, indicating ICRT is not sensitive to the prompt trajectory types. We hypothesize this is because during the training, ICRT has seen different types and numbers of prompts.

Unseen Primitives

Task	Grasp and Drop the Toy Tiger	Grasp and Drop the Blue Sponge	Put Blue Sponge to Right of Toy Tiger
Success Rate	40%	80%	80%

Table B.3: Experiments on three tasks using two unseen primitives. Success rates are calculated over 5 trials for each experiment.

We evaluate the generalization capability of ICRT on primitives that are unseen during the training but resemble the training primitives. We consider two such unseen primitives: grasp and drop an object and put object A to the right of object B. We consider three tasks: grasp and drop a toy tiger, grasp and drop a blue sponge (unseen objects during training) and put the blue sponge to the right of the toy tiger. As in Section 4.5, we conduct 5 tiers for each task. Experiment results are summarized in Table B.3, where ICRT shows decent success rate on all three tasks, suggesting that ICRT can generalize to some unseen primitives that resemble the training primitives.

Co-training

For training ICRT, we opt to separate the training into two stages: a pre-training phase where the model is pre-trained on the DROID dataset [64], and a fine-tuning phase where the model is trained on the ICIL-MT dataset. In this ablation, we experiment with whether these two can be combined into a single stage, where the policy is end-to-end trained with DROID and ICIL-MT. To balance the two datasets, we first calculate the median number of trajectories per task across the two datasets, then for each epoch, sample each task with the median number of trajectories. This allows each task to be equally represented in each epoch. We train the model for the same number of epochs as for ICRT fine-tuning and report the results in Table B.4. The results indicate that the model does not converge as quickly in the combined stage and fails to respond to prompts and complete tasks effectively. We hypothesize two reasons for this: firstly, the dataset is heavily biased towards DROID, which contains 200 tasks compared to only 29 tasks in ICIL-MT, making it difficult for the model

to learn the tasks as effectively as in the separate stage training. Future works can analyze the data mixture and how to train with large-scale datasets more effectively.

	Pick and Place	Poke	Average
ICRT (Co-train)	13.3 (±5.8)	0.0 (±0.0)	6.7 (±3.0)
ICRT	**65.0 (±7.3)**	**93.3 (±4.6)**	**79.2 (±4.6)**

Table B.4: Ablation on co-training with DROID [64]. Training both DROID and ICRT-MT datasets in a single stage leads to worse task performance in both pick-and-place and poking tasks. Same as Table 4.1, we evaluated each task primitive using six tasks not seen during training, conducting five trials per task for a total of 30 trials per primitive and 60 trials overall to calculate average performance. For each model, we report the mean success rate for each task, the overall success rate, and the corresponding standard error in parentheses.

Detailed results

In this section, we present the per-task performance for the pick-and-place primitive (Table B.5) and the poking primitive (Table B.6). For each action primitive, we design *six unseen tasks* (as defined in Section 4.3), with three tasks evaluating *in-domain* object generalization (selected from *yellow cube, red cube, black cube, pink bowl, and blue bear*) and three objects *unseen* during training (selected from *radish, blue sponge, grey dog, and black dog*).

Hyperparameters

We provide the hyperparameters for both the pre-training and fine-tuning phase in Table B.7 and Table B.8.

Pick Object Place Location	Yellow Cube Black Bowl	Yellow Cube Grey Bowl	Blue Bear Pink Bowl	Radish Grey Bowl	Black Dog Pink Bowl	Blue Sponge Silver Pot	Average Success (± Std Err.)
Goal Conditioned	40%	30%	20%	40%	40%	30%	33.3% (±6.5%)
Octo	10%	0%	10%	10%	0%	0%	5.0% (±2.7%)
OpenVLA	0%	0%	0%	50%	20%	0%	11.7% (±4.6%)
ICRT-Llama2	40%	40%	40%	60%	40%	40%	43.3% (±7.9%)
ICRT (DROID)	0%	0%	0%	0%	0%	0%	0.0% (±0.0%)
ICRT (MT)	**90%**	**50%**	**80%**	**90%**	**60%**	**90%**	**76.7% (±7.1%)**
ICRT +Prompt Loss	20%	10%	20%	40%	30%	10%	21.7% (±6.2%)
ICRT (Co-train)	10%	0%	10%	0%	40%	20%	13.3% (±5.8%)
ICRT	60%	**50%**	**80%**	50%	**60%**	**90%**	65.0% (±7.3%)

Table B.5: Experimental results for the *pick-and-place* primitive. Here we list the 6 tasks that were evaluated and their corresponding success rate. For each model, we also report the mean success rate and the standard error.

Poke Object	Radish	Red Cube	Grey Dog	Black Cube	Pink Bowl	Blue Sponge	Average Success (± Std Err.)
Goal Conditioned	0%	0%	0%	0%	40%	0%	6.7% (±4.6%)
Octo	20%	0%	60%	0%	0%	0%	13.3% (±6.2%)
OpenVLA	20%	0%	0%	0%	0%	0%	3.3% (±3.3%)
ICRT-Llama2	60%	**100%**	80%	60%	60%	80%	73.3% (±8.2%)
ICRT (DROID)	0%	0%	0%	0%	0%	0%	0.0% (±0.0%)
ICRT (MT)	**100%**	**100%**	40%	60%	60%	60%	70.0% (±8.5%)
ICRT +Prompt Loss	0%	20%	20%	**80%**	0%	20%	23.3% (±7.9%)
ICRT (Co-train)	0%	0%	0%	0%	0%	0%	0.0% (±0.0%)
ICRT	**100%**	**100%**	**80%**	**80%**	**100%**	**100%**	**93.3% (±4.6%)**

Table B.6: Experimental results for the *poking* primitive. Here we list the 6 tasks that were evaluated and their corresponding success rate. For each model, we also report the mean success rate and the standard error.

Config	Value
optimizer	AdamW
base learning rate	1e-3
learning rate schedule	cosine decay
batch size	64
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
warm up epoch	0.5
total epochs	4
proprioception noise	0.005
action noise	0
sequence length	512
brightness augmentation	0.1
contrast augmentation	0.2
num action prediction	16

Table B.7: Pre-training Hyperparameters

Parameterization

Proprioception The proprioception space is parameterized by the absolute end effector translation (x, y, z), a 6DoF rotation vector, and a continuous end-effector gripper state. This results in a 10-dimensional proprioception representation. The 6DoF rotation vector is flattened from the $SO(3)$ rotation's matrix's first two rows.

Action We use delta end effector pose as the action parameterization. At each prediction step, the model predicts t actions. Given *absolute* end effector action transforms in $T_1, T_2, \cdots, T_t$ in a trajectory and the current end-effector pose T_{ee}, we define the relative transforms that the model needs to predict as $T_{\text{ee}}^{-1}T_1, T_{\text{ee}}^{-1}T_2, \cdots T_{\text{ee}}^{-1}T_t$. We then append the continuous absolute gripper position to each delta action. Similar to proprioception, we present the delta action by the relative end effector translation and a 6DoF rotation. This results in a

Config	Value
optimizer	AdamW
base learning rate	5e-4
learning rate schedule	cosine decay
batch size	64
weight decay	0.01
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
warm up epoch	1.25
total epochs	125
proprioception noise	0.005
action noise	0
sequence length	512
brightness augmentation	0.1
contrast augmentation	0.2
num action prediction	16

Table B.8: Finetuning Hyperparameters

10-dimensional action representation. When rolling out the predicted actions, in addition to temporal ensembling [55], we also use receding horizon control [145], and select an action horizon of 10 steps.

System Information

All models are trained on 4 NVIDIA A100 80GB GPUs. ICRT pre-training on DROID takes 56 minutes and fine-tuning on ICRT-MT takes 18 hours. ICRT-Llama7B takes roughly 28 hours to finetune. We report the inference speed of ICRT and ICRT-Llama2 in Table B.9 averaged over 100 steps. All tests are performed on a workstation with NVIDIA RTX 3090Ti and Intel i5-12400F with 64GB memory. We find that using the proposed formulation, which can leverage the KV cache, we can run ICRT-Llama2 at 10Hz naively.

	Inference Frequency
ICRT	39.6 Hz
ICRT-Llama2	10.7 Hz

Table B.9: Inference frequency of ICRT, averaged over 100 steps.

Chapter C

Appendix C: OTTER – A Vision-Language-Action Model with Text-Aware Visual Feature Extraction

C.1 Environment Setup

Simulation Tasks

For the training tasks, we use the original tasks in LIBERO-Goal, LIBERO-Spatial, and LIBERO-Object. We also build unseen evaluation tasks based on 10 original LIBERO-90 tasks, by changing language instructions and target object color and type in the task bddl files. The 10 unseen tasks are listed in Table C.1.

Changes	Unseen
object type	Put the moka pot in the bottom drawer of the cabinet
object type	Put the moka pot on the wine rack
object type	Pick up the ketchup and put it in the basket
object type	Pick up the ketchup on the plate
object type	Pick up the bottle and put it in the tray
object color	Put the black bowl on top of the cabinet
object color	Put the black bowl on the plate
object color	Put the red mug to the right of the plate
object color	Put the yellow and white mug in the front of the red mug
object color	Put the red mug to the front of the moka

Table C.1: The 10 in-distribution tasks and 7 unseen tasks we used in our real-world setting.

Real-world Tasks

The full list of tasks for our real-world evaluation is provided in Table C.2.

In-Distribution	Unseen
Put potato in pot to black bowl	Put yellow cube in black bowl
Pickup potato	Pick up radish and place it in grey bowl
Pick up and place deer in grey bowl	Put blue bear in pink bowl
Pick up green triangle	Put yellow cube in grey bowl
Put tiger to black bowl	Put apple with a green leaf in black bowl
Put red cube into black bowl	Pick up blue sponge and place it in steel pot
Put blue cube into grey bowl	Pick up black dog and place it in the pink bowl
Put the red ball in black bowl	
Put green triangle into pink bowl	
Put blue cube in pink bowl	
Poke a wooden block	Poke the radish
Poke a tiger	Poke the gray dog
Poke a green triangle	Poke the pink bowl
Poke a gray bowl	
Pour from the brown cup to the gray bowl	Pour from the orange cup to the black bowl
Pour from the blue cup to the pink bowl	Pour from the blue cup to the black bowl
Pour from the yellow cup to the black bowl	Pour from the brown cup to the pink bowl
Open the drawer	Open the drawer with a tiger on top
Close the drawer	Close the drawer with a red cube inside

Table C.2: The 10 in-distribution tasks and 7 unseen tasks we used in our real-world setting.

For each experiment trial of poking and pouring, we vary the location of the target object to manipulate and introduce 2 or 3 random distractor objects. For drawer, we vary the location of the drawer on each trial. Similar to the pick and place primitive, for each task, we generate 10 randomized scenes.

Each trial is scored based on the robot's performance in completing the task. For the pick and place primitive, a score of **0.5** is awarded if the robot successfully picks up the correct target object, and a score of **1** is given if the robot not only picks up the correct object but also places it in the correct location as specified by the instruction. If the robot fails to pick up the target object or picks up a distractor object, a score of **0** is recorded. For other primitives, a score of **1** is recorded if the task is completed, otherwise a score of **0** is given.

For all models other than the OpenVLA, each trial is allowed a maximum of 30 seconds to complete. As OpenVLA is a large 7B model wit a lower inference speed, we give it a time limit of 60 seconds to complete a task.

C.2 Model and Training Details

Model Architecture for OTTER and Baselines

The details of our model parameters can be found in Table C.3. All the baselines share the same hyper-parameters with OTTER. For OTTER w.o. CLIP Vision, we use a ViT Encoder based on the implementation of `https://github.com/google-research/vision_transformer` with a ViT-Ti/16 configuration with half of the number of attention layers. For OTTER w.o. f_e and OTTER w.o. f_l', we use the same model configuration but only remove the corresponding attention pooling layers. We incorporate action chunking into OpenVLA by asking it to predict the next 16 actions, which performs better than vanilla OpenVLA which predicts only the next step. For Octo, we use the official Hugging Face Checkpoint at `hf://rail-berkeley/octo-small-1.5` which is in a comparable size with our model. During inference, we cache the CLIP feature outputs. This enables the ViT-L/14 OTTER model to perform inference at $50Hz$ on a single NVIDIA 3090Ti, allowing real-time control.

Hyperparameter	Value
CLIP Model (Real)	ViT-L/14
CLIP Model (Sim)	ViT-L/32
# Pooling Readouts	4
# Pooling Attention Heads	8
# Pooling Attention Blocks	2
# Text-Pooling Output Dimension	128
# Image-Pooling Output Dimension	512
# Proprio-Pooling Output Dimension	64
Causal Transformer Parameters:	
# Attention Blocks	4 (8)
# Attention Heads	8
# Latent Dimension	512 (768)
# Context Length (Real)	12
# Context Length (Sim)	4
# Action Prediction Horizon (Real)	12
# Action Prediction Horizon (Sim)	1
# Parameter	11,790,522 (25,516,986)

Table C.3: Hyperparameters for OTTER model architecture. Values in the parenthesis shows the hyperparameters for a larger and wider OTTER for the real world experiments.

Training Hyper-parameters

We use the AdamW optimizer with a cosine learning rate decay schedule and linear learning rate warm-up. We list training hyperparameters in Table C.4. All these hyper-parameters are shared between real-world and simulation. All the models are trained on 4 NVIDIA A100 80GB GPUs.

Hyperparameter	Value
Learning Rate	3e-4
Warmup Steps	2000
Weight Decay	0.01
Learning Rate Scheduler	cosine
Gradient Clip Threshold	1
Batch Size	64
Total Gradient Steps	40000 (60000)
Image Resolution	224×224
Random Resized Ratio	[0.9, 1.1]
Random Brightness	0.2
Random Contrast	[0.8, 1.2]
Random Saturation	[0.8, 1.2]
Random Hue	0.1

Table C.4: Hyperparameters used for training (pre-training on OXE).

C.3 Vision-Language Attention Visualization

To provide further motivations for why using X_{out} (per [121]) instead of the output feature map of CLIP, we compare the cosine similarity between the output of the CLIP ViT-L/16 encoder and the per-token text features in three different settings: (1) fine-tuning the encoder, (2) a frozen CLIP's output features (X_{out}), and (3) a frozen CLIP's last attention block's feature (X_{attn}) as described in Section 5.3. The similarity visualization is shown in Figure C.1.

It may initially seem unexpected that this type of visualization is reasonable. However, this can be explained by the fact that LayerNorm operates independently of the patch dimension, as it normalizes along the channel dimension. When combined with the vision-alignment weight matrix w_v, the operation $\hat{f}_v = \mathrm{LN}_{\mathrm{post}}(f_v)w_v$ remains linear. Therefore we can linearize the final attention block:

$$\hat{f}_v = \mathrm{LN}_{\mathrm{post}}(X_{out})w_v \tag{C.1}$$

$$= \mathrm{LN}_{\mathrm{post}}(X_{res} + X_{attn} + \mathrm{FFN}(\mathrm{LN}(X_{sum})))w_v \tag{C.2}$$

$$= \mathrm{LN}_{\mathrm{post}}(X_{res})w_v + \mathrm{LN}_{\mathrm{post}}(X_{attn})w_v + \mathrm{LN}_{\mathrm{post}}(\mathrm{FFN}(\mathrm{LN}(X_{sum})))w_v \tag{C.3}$$

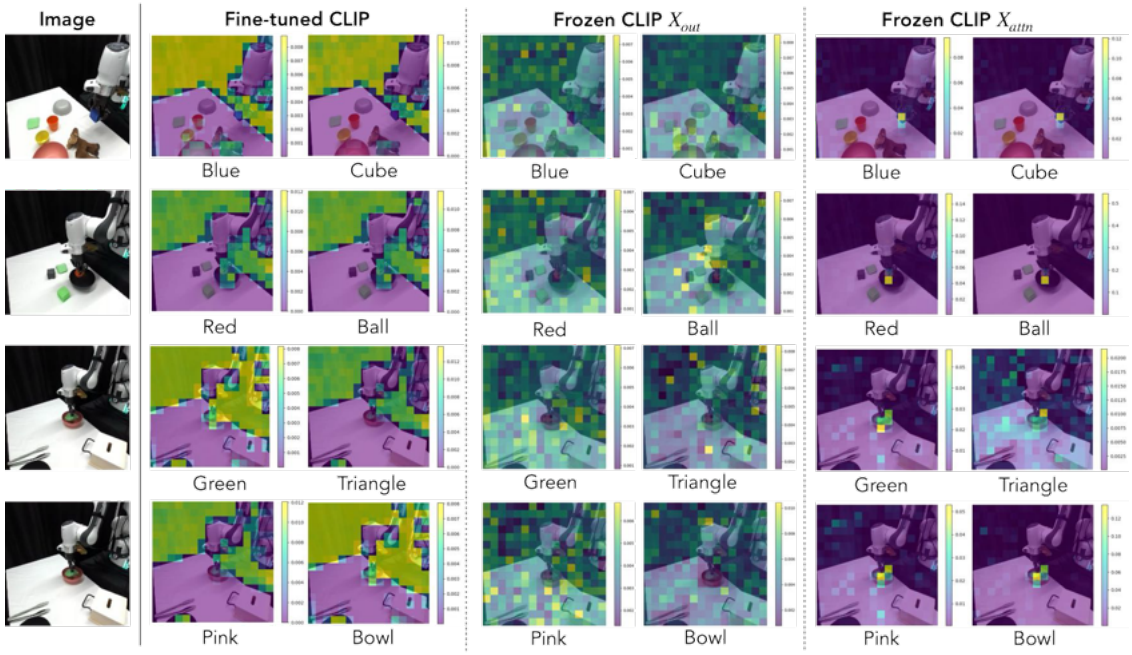


Figure C.1: Examples of attention maps for CLIP fine-tuned with VLA (left) and frozen CLIP's output (X_{out}) (middle) and frozen CLIP's attention features (X_{attn}) (right). The first column shows the side view observation and the text query is below each attention map. Fine-tune CLIP pays attention to the background and the frozen CLIP's output (X_{out}) is noisy. In contrast, the frozen CLIP (X_{attn}) pays attention to the correct object associated with the text query. These examples indicate that fine-tuning CLIP on robotic datasets can degrade the performance of the pre-trained CLIP, especially when the robotics dataset is small. It also highlights the benefits of using X_{attn} for fused vision-language features.

For ClearCLIP, or Frozen CLIP X_{attn}, we are visualizing the $\text{LN}_{\text{post}}(X_{attn})w_v$ term.

Similar to what ClearCLIP has noted, after adding residual connection and the final FFN, the features become noisy and worsen the alignment between language and visual features. The noisy attention map makes it challenging for the model to identify the correct features directly from the feature map, which makes it necessary for existing VLA (i.e. OpenVLA [428]) to fine-tune the CLIP vision encoder. In comparison, by using X_{attn}, object localization becomes an easier task in OTTER: we can extract the location of the object by getting the *softmax* across the attention map without using any parameters (see Figure 5.3). More attention map examples on Open-X dataset are in Figure C.2.

In the finetuning v.s. frozen CLIP (X_{attn}) comparison, fine-tuning OTTER's CLIP results in overfitting to foreground-background separation, causing it to lose zero-shot object detection ability. This limits the model's ability to highlight the correct object, leading to a significant drop in task success rates (26% vs 68% for training tasks and 15 vs 62% for unseen tasks). Conversely, a frozen CLIP (X_{attn}) preserves object detection capabilities, providing

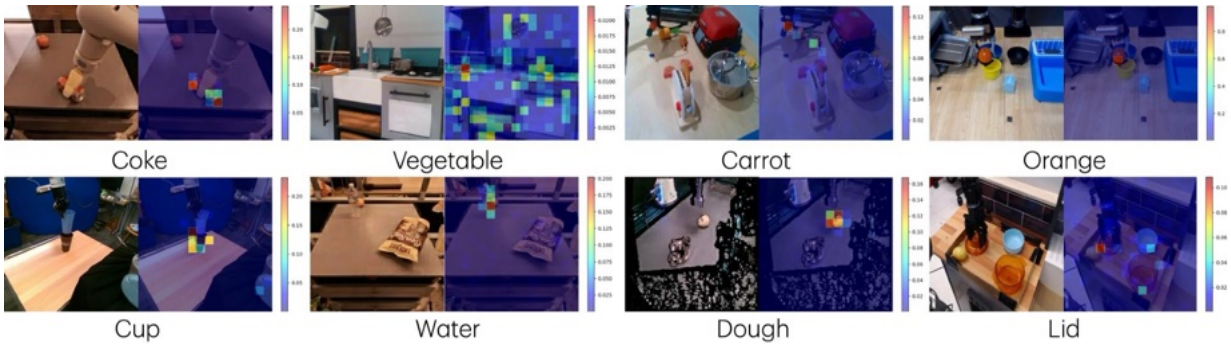


Figure C.2: Examples of attention maps of frozen CLIP's attention features (Xattn) on Open-X dataset. The bottom texts are the corresponding text tokens.

better downstream performance.

Pre-training on the Human Dataste

Both datasets provide labeled human hand poses in the camera frame, represented by the 3d positions of the 21 MANO [429] hand joints. We post-processed the labeled hand pose to extract the human hand wrist transformation as the human state, where the translation is the human wrist joint position and the rotation is the palm rotation, estimated by fitting the palm to a plane from the metacarpophalangeal joint locations of the five fingers and the wrist joint. DexYCB dataset is a dataset consisting third-person view videos of a human grasps different objects from a tabletop environments. We label each image frame with the language instruction of "pick up the {object name}", where {object name} is the object being grasped by the human. HOI4D dataset is a dataset consisting of egocentric view videos of a human manipulating different objects. Different manipulation tasks are considered such as "pick up objects", "push objects" and "open and close the door". We use the provided language instructions to label each image frame. In total, we obtain 8k trajectories of RGB image frames, language instructions and human states from the DexYCB dataset and 7.6k trajectories from the HOI4D dataset, resulting 15.6k human trajectories.

C.4 More Ablations

We consider another 2 ablations of OTTER. Both are trained on the DS-PnP.

1. DFP-OTTER (CLS): another variant of OTTER that utilizes CLIP's ¡*cls*¿ token rather than text-aware visual feature extraction.

2. OTTER (xattn): OTTER using standard cross attention pooling between the text tokens f_l and the vision tokens f_v to obtain the fused vision language features f'_{lv} instead of text-aware visual feature extraction as in Eq.(4).

From Table C.5, both DFP-OTTER (cls) and OTTER (xattn) fail to generalize to unseen tasks, highlighting the benefits of using text-aware visual feature extraction to obtain task-related vision features as the fused vision language features.

Method	DFP-OTTER (CLS)	OTTER (xattn)	OTTER
Success Rate	6% ± 0.8%	2% ± 0.5%	62% ± 4.2%

Table C.5: Physical results on 70 trials on unseen pick up and place tasks for other variants of OTTER.

Chapter D

Appendix D: Real2Render2Real – Scaling Robot Data Without Dynamics Simulation or Robot Hardware

D.1 Appendix

Raw Evaluation Results

We report raw task success rates for each policy in Table D.1.

Task / Policy	Teleop Trajectories			R2R2R Trajectories			
	50	100	150	50	100	150	1000
Pick up the tiger							
Diffusion Policy	6.7%	66.7%	73.3%	0.0%	26.7%	40.0%	66.6%
π_0-FAST (Finetuned)	26.7%	40.0%	73.3%	0.0%	0.0%	6.7%	80.0%
Put the mug on the coffee maker							
Diffusion Policy	13.3%	33.3%	40.0%	13.3%	13.3%	33.3%	53.3%
π_0-FAST (Finetuned)	6.6%	13.3%	73.3%	0.0%	0.0%	33.3%	80.0%
Pick up the package with both hands							
Diffusion Policy	66.7%	66.7%	80.0%	20.0%	33.3%	20.0%	73.3%
π_0-FAST (Finetuned)	6.7%	46.7%	60.0%	6.6%	13.3%	6.6%	66.7%
Open the drawer							
Diffusion Policy	20.0%	60.0%	66.7%	13.3%	33.3%	46.7%	66.7%
π_0-FAST (Finetuned)	0.0%	40.0%	60.0%	0.0%	20.0%	13.3%	86.6%
Turn the faucet off							
Diffusion Policy	20.0%	46.7%	66.7%	20.0%	33.3%	53.3%	80.0%
π_0-FAST (Finetuned)	35.3%	60.0%	80.0%	0.0%	13.3%	13.3%	80.0%

Table D.1: Comparison of Physical Policy Success Rates Across Training Sources. Task success rates for Diffusion Policy and π_0-FAST trained exclusively on either human teleoperation data (left) or R2R2R-generated data (right). Each policy was evaluated on 15 trials per task using a binary success metric: a score of 1 is assigned for successful task completion, and 0 otherwise.

Additional Ablation Experiments

Trajectory Interpolation

R2R2R generates diverse trajectories by adapting a single human demonstration to new object poses through interpolation and spatial transformation (see Section 8.4 and Figure 8.4 for visualization). To evaluate the impact of this trajectory interpolation step, we ablate it by replaying only the original object motion track without adapting to varied initial and goal poses. Table D.2 shows a substantial drop in performance when interpolation is disabled: on the "Put the mug on the coffee maker" task, success rates fall from 80.0% to 0.0% for π_0-FAST and from 53.3% to 6.7% for Diffusion Policy. This highlights that simply replaying object motion from a single demonstration is insufficient for generating transferable robot behaviors—trajectory adaptation is crucial to scaling data diversity in object-centric manipulation.

Background and Tabletop Texture Augmentation

Our default data generation pipeline includes moderate visual augmentation, such as randomized lighting, camera pose, and object placement, as well as sampling from a limited set of

Policy	w/o Trajectory Interpolation (1k)	w/ Trajectory Interpolation (1k)
π_0-FAST (Finetuned)	0.0%	**80.0%**
Diffusion Policy	6.7%	**53.3%**

Table D.2: Success rates on "Put the mug on the coffee maker" using R2R2R-generated data with and without trajectory interpolation (1,000 demos). Interpolation enables adapting object motion to varied contexts, which is critical for policy generalization.

lightbox-style background environments. To study the effect of stronger visual perturbations, we apply more aggressive augmentation that includes a wider variety of lightbox backgrounds and diverse tabletop textures (see Figure D.1).

Table D.3 reports the success rates on the task *Put the mug on the coffee maker* under this more varied visual setting. We observe a consistent drop in policy performance across both π_0-FAST and Diffusion Policy when trained on data with aggressive background and surface augmentation. This result suggests that while visual diversity is generally beneficial, overly strong appearance perturbations may harm policy learning when not properly balanced. Future work may investigate more principled augmentation schedules or adaptive augmentation strategies to preserve generalization while maintaining performance.

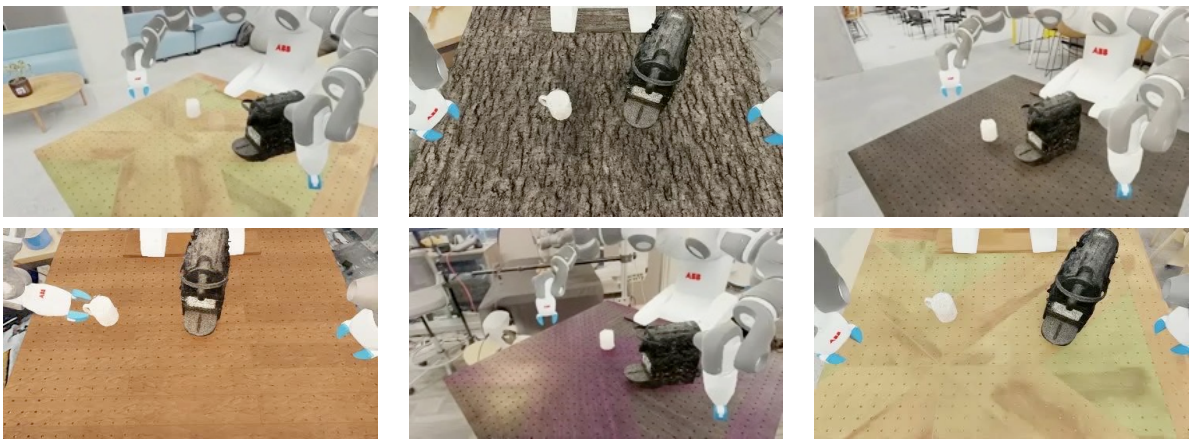


Figure D.1: Background and Tabletop Texture Augmentation – Each image corresponds to a different environment.

Policy	Less Background Aug. (1k)	More Background Aug. (1k)
π_0-FAST (Fine-tuned)	**73.3**%	35.3%
Diffusion Policy	**53.3%**	33.3%

Table D.3: Success rate comparison on *Put the mug on the coffee maker* with and without background and tabletop texture augmentation. We generated 1000 trajectories for each setting and evaluated across 15 trials.

Sim-and-Real Co-training

While sim-and-real co-training is not the main focus of Real2Render2Real, we included additional results comparing policies exclusively on either R2R2R-generated data, human teleoperation data, and co-training setup that combines data from both sources. Specifically, for the task *Put the mug on the coffee maker*, we trained a policy using 1,000 R2R2R-generated demonstrations together with 150 human teleoperation demonstrations. We do not perform additional importance sampling or re-weighting of human teleoperation data. For the π_0-FAST policy, co-training achieved a success rate of 73.3%, which is on par with training using only R2R2R data or only real demonstrations individually. Co-training for diffusion policy yields a significant improvement over either real data only or R2R2R-generated data only, where the performance improved from 40.0% to 86.7%. We hypothesize that since LoRA [341] serves as a significant regularizer for π_0-FAST, end-to-end fine-tuning with completely unfrozen model with additional hyperparameters tuning could lead to better performance. For more in-depth analysis on how co-training can improve policy performance, please refer to [270, 430].

Policy	Real Data Only (150)	R2R2R Data Only (1k)	Co-Training (150+1k)
π_0-FAST	73.3%	**80.0%**	73.3%
Diffusion Policy	40.0%	53.3%	**86.7%**

Table D.4: Success rate comparison on *Put the mug on the coffee maker* under different training datasets mixtures.

Task Visualizations

Physical policy rollout figures show model input RGB frames from real policy evaluation successes using either Diffusion Policy [54] or π_0-FAST [340]. The depicted policies were trained *exclusively* on R2R2R synthetic data.

Put the Mug on the Coffee Maker

Figure D.2: Put the Mug on the Coffee Maker – Demonstration Video Frames.

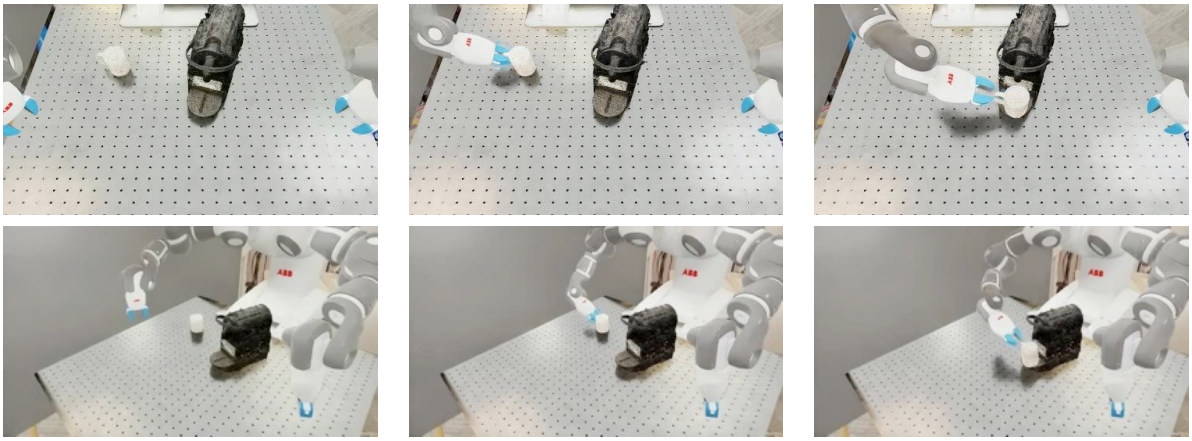


Figure D.3: Put the Mug on the Coffee Maker – Example R2R2R Frames.

Figure D.4: Put the Mug on the Coffee Maker – Physical Policy Rollout.

Turn off the Faucet

Figure D.5: Turn off the Faucet - Demonstration Video Frames.

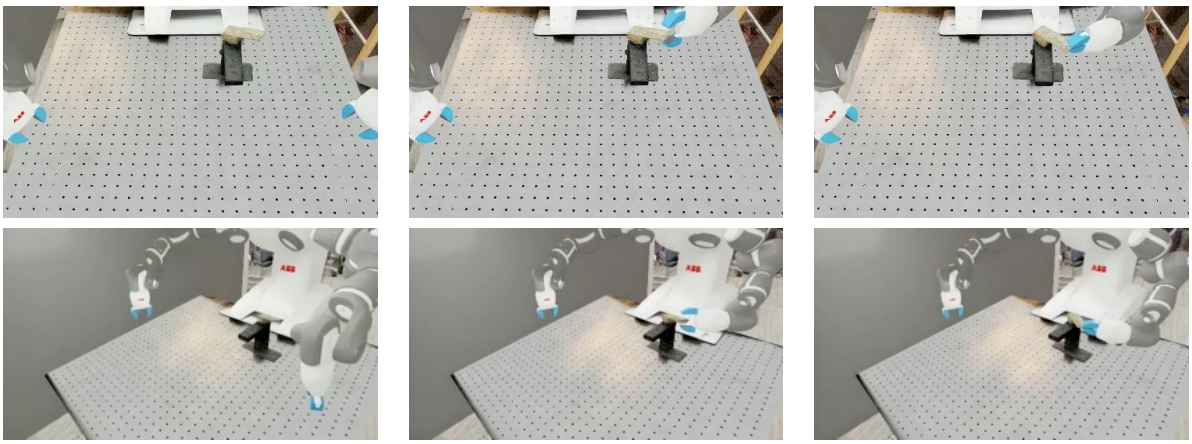


Figure D.6: Turn off the Faucet - Example R2R2R Frames.

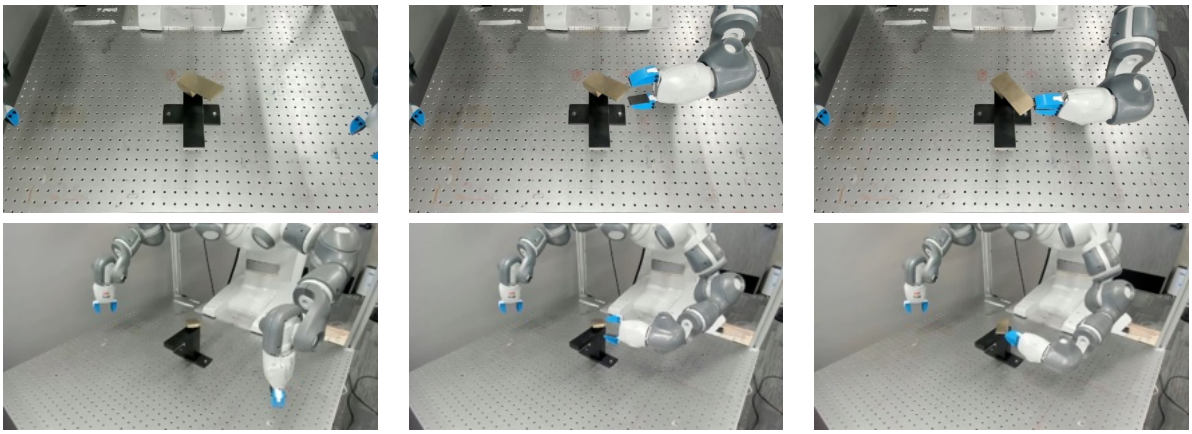


Figure D.7: Turn off the Faucet - Physical Policy Rollout.

Note: For human teleoperated demonstrations, the teleoperator would push down on the faucet handle in a non-prehensile motion to turn it off instead of grasping the handle and twisting it closed as is done with R2R2R–where only prehensile grasping is currently supported.

Open the Drawer

Figure D.8: Open the Drawer - Demonstration Video Frames. **Note:** The true video order–and thus the tracked trajectory–was in reverse, as a full multi-view scan for the inner drawer requires it to first be in an open configuration.

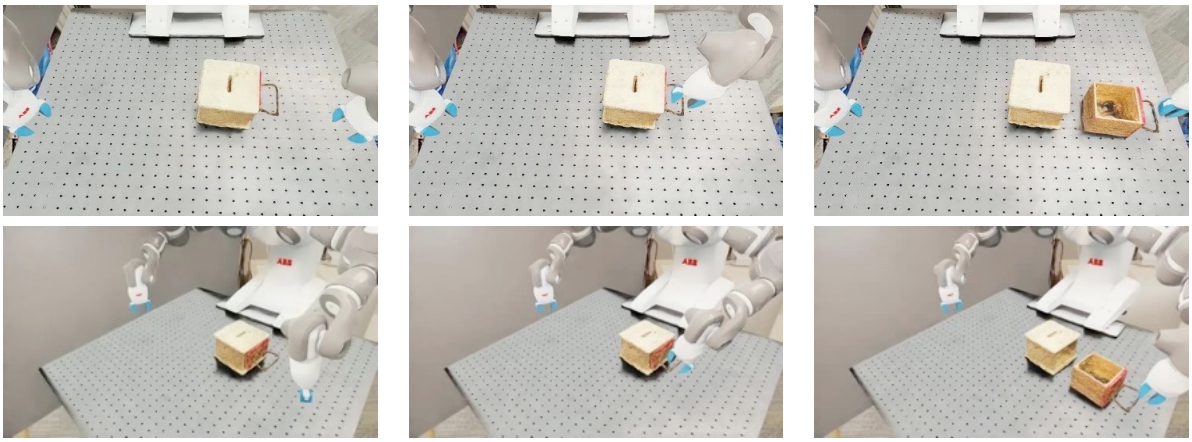


Figure D.9: Open the Drawer - Example R2R2R Frames.

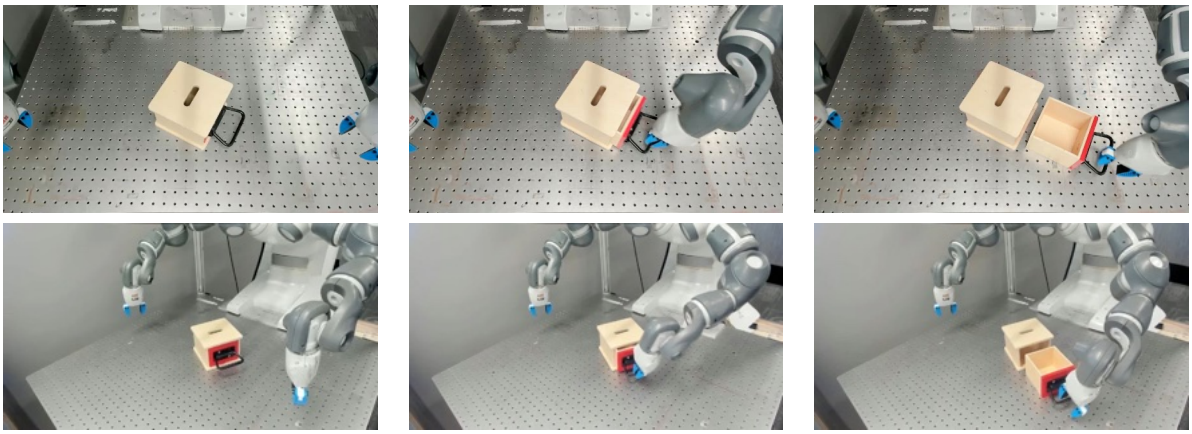


Figure D.10: Open the Drawer - Physical Policy Rollout.

Lift up the Package with Both Hands

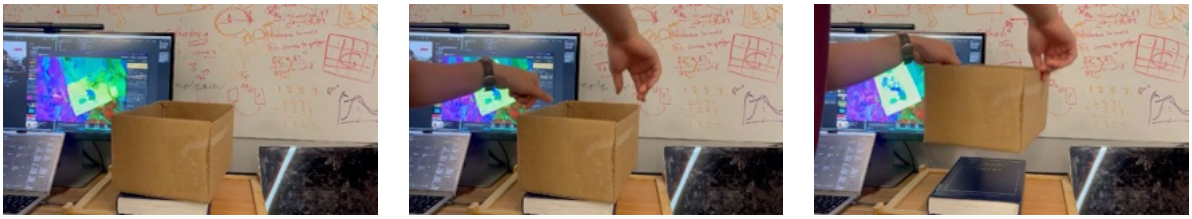


Figure D.11: Lift up the Package with Both Hands - Demonstration Video Frames.

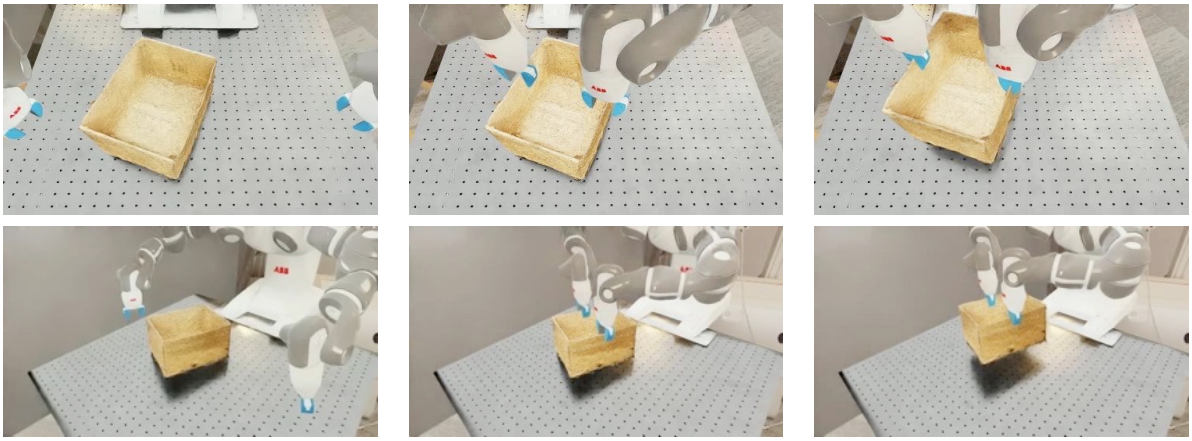


Figure D.12: Lift up the Package with Both Hands - Example R2R2R Frames.

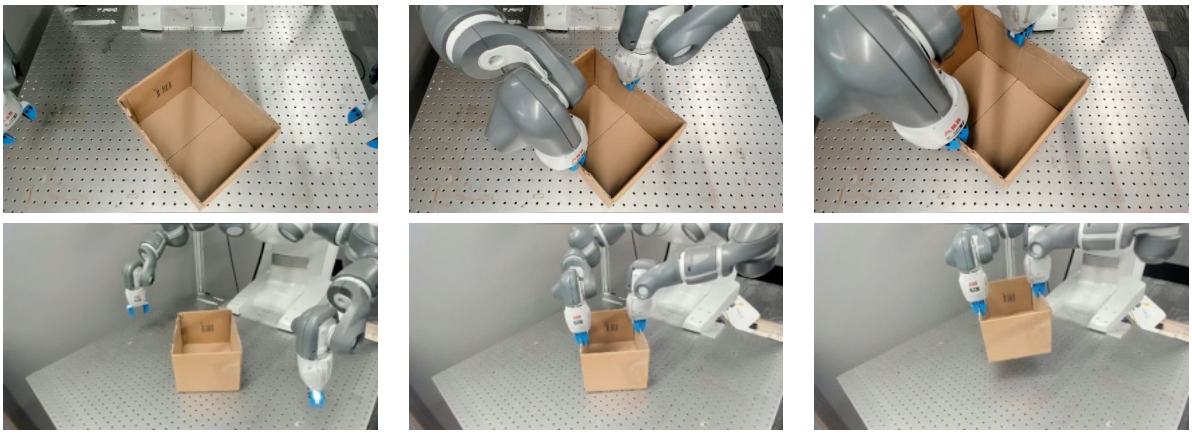


Figure D.13: Lift up the Package with Both Hands - Physical Policy Rollout.

Pick up the Tiger

Figure D.14: Pick up the Tiger - Demonstration Video Frames.

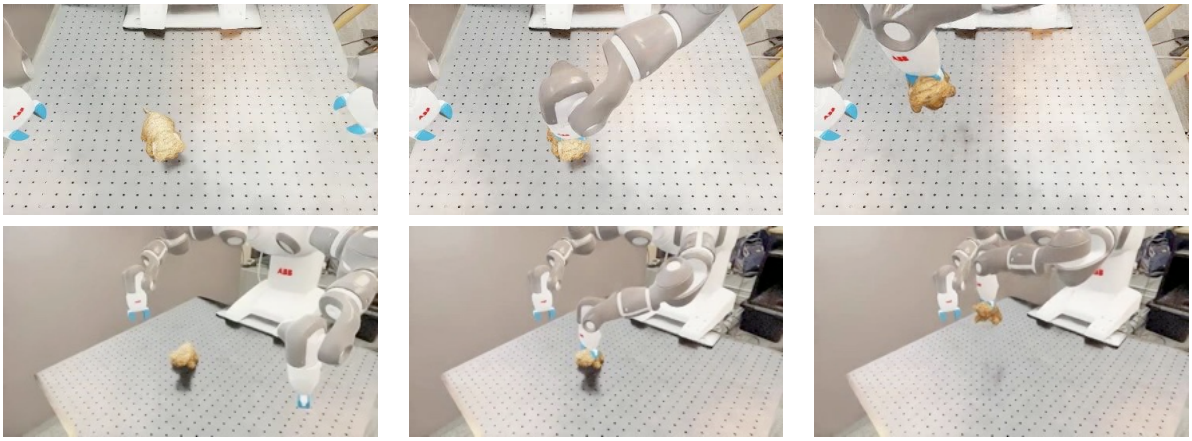


Figure D.15: Pick up the Tiger - Example R2R2R Frames.

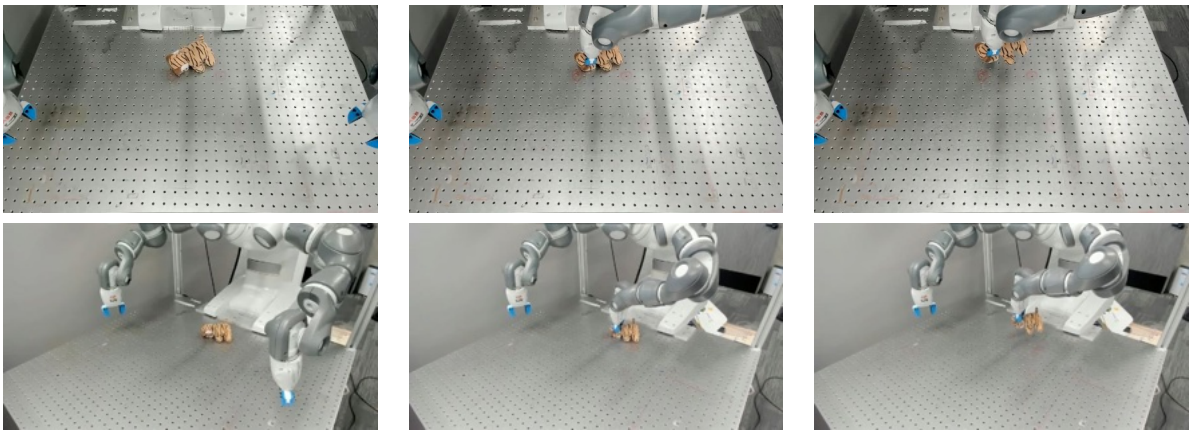


Figure D.16: Pick up the Tiger - Physical Policy Rollout.

Put the Mug on the Coffee Maker (Franka Robot Embodiment)

Unlike vanilla π_0-FAST (DROID) [340], which operates under joint velocity control, R2R2R records only joint positions. To accommodate this difference, we fine-tuned π_0-FAST (DROID) to predict delta joint positions (and absolute gripper position, consistent with [340]). Since Franka's impedance control mode can be imprecise, we apply blocking control with temporal ensembling to improve execution accuracy.

Although the learned policy occasionally completes the task successfully, we observe several consistent failure modes, largely stemming from limitations of the default Franka gripper:

1. Collision during grasping: The grasp approach that is near parallel to the table often causes the gripper to collide with the table surface during mug pickup.

2. Off-center grasp: The gripper's wide jaws tend to produce asymmetric contacts, with one pad closer to the mug than the other. This imbalance induces rotation, leading to slippage.

3. Difficulty in precise placement: The wide gripper also makes it challenging to release the mug accurately onto the coffee machine.

To mitigate these issues, we recommend using the Robotiq 2F-85 gripper in future experiments. Its smaller form factor and improved grasping precision may reduce failure rates and improve placement consistency.

Figure D.17: Put the Mug on the Coffee Maker (Franka Robot) - Example R2R2R Frames.

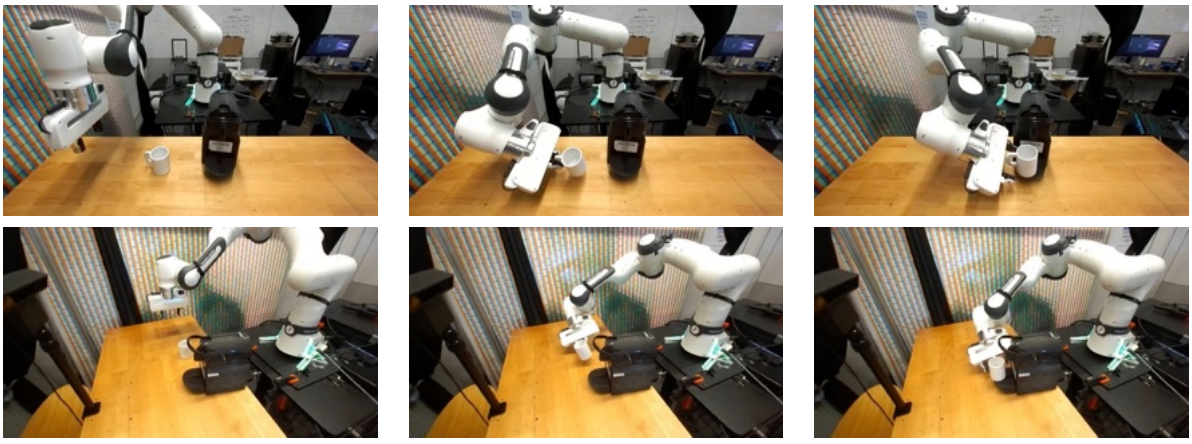


Figure D.18: Put the Mug on the Coffee Maker (Franka Robot) - Physical Policy Rollout.

Qualitative Ablations

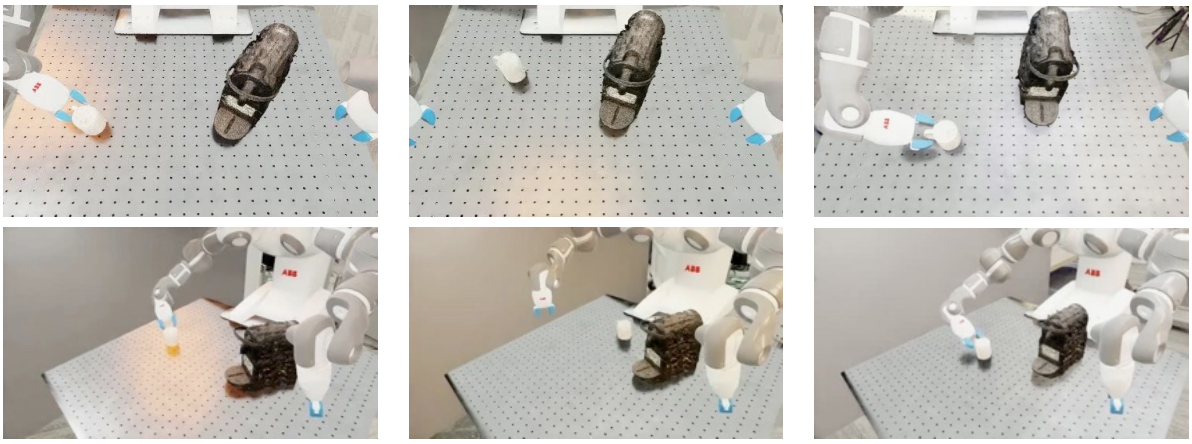


Figure D.19: Real2Render2Real (Views From Both Cameras Shown). Base augmentations used in the main R2R2R experiments include: random sphere lighting, camera pose perturbation, robot initial joint perturbation, and randomized object initialization uniformly distributed via manual parameters.

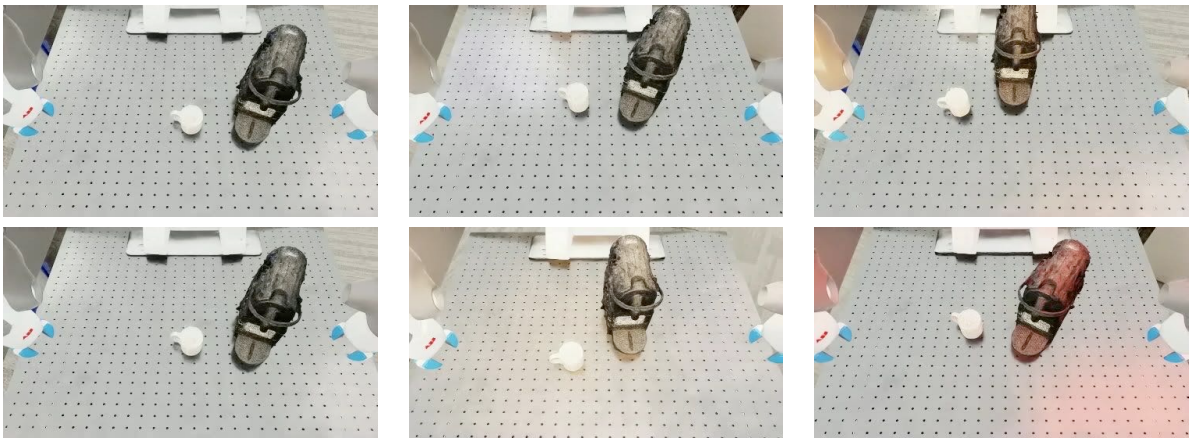


Figure D.20: Trajectory Interpolation Turned Off (Top Camera Views Shown). *Note the fixed configuration of the mug with respect to the coffee maker.* With trajectory interpolation off for multiple rigid bodies, we may only densely follow the tracked trajectories shown in the video demonstration. Without it, the only method for increasing trajectory diversity would be to augment with part trajectories from adding/tracking additional demonstration videos.

Upfront Processing Time Until Generation

Data Collection and Generation Time

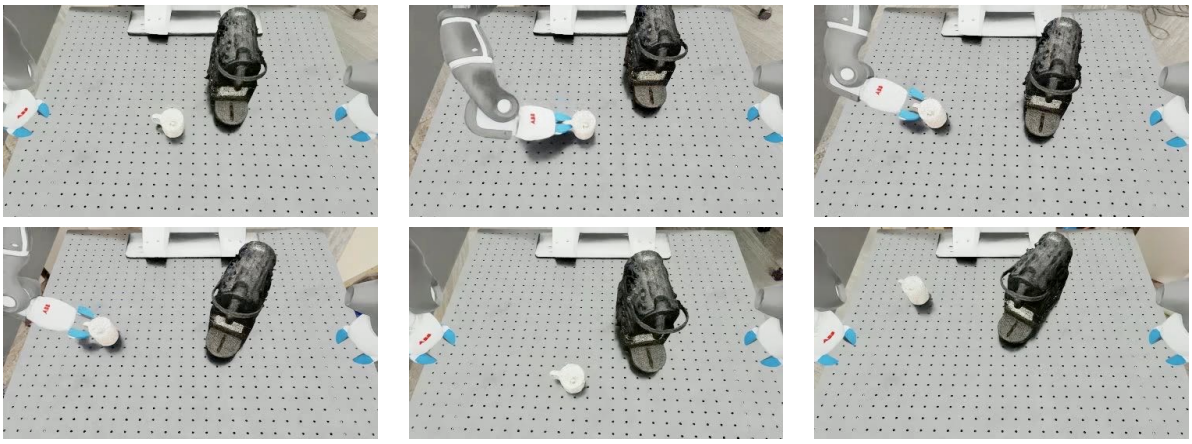


Figure D.21: Random Lighting Augmentation Turned Off (Top Camera Views Shown). We turn off the randomized sphere light sources with varying colors and intensities. Uniform lighting is available from the only light source in the render scene – the skybox/dome-light asset.

Task	Time to Complete
Scanning	1 min
Demonstration	<10 sec
GARField Segmentation [337]	2 min
3DGS Optimization	1 min
4D-DPM Tracking [334]	3 mins
SuGaR [338] Meshification	2 mins
Asset into IsaacLab	1 min

Table D.5: Upfront Processing Time per Task Prior to R2R2R Data Generation. Breakdown of one-time preprocessing steps required to convert a demonstration video and scanned asset into a renderer-ready format for R2R2R. These steps include segmentation, tracking, meshification, and asset import.

Extended Training Details

We provide hyperparameters for training diffusion policy [54] from scratch and fine-tuning π_0-FAST [340] with LoRA in Table D.7 and Table D.8.

Extended Inference Details

In experiments, we use a Pi0-FAST model to predict robot actions as frequency-space tokens. While effective as an out-of-the-box VLA model, its autoregressive decoding procedure is inherently slow. Each token must be generated sequentially before the trajectory can be reconstructed. During evaluation, this latency posed a practical bottleneck: the raw observation-to-action inference time was high and resulted in slow experimental evaluations.

Task	Teleop, 150 demos, 1 operator	R2R2R, 1k demos, 1 GPU
Pick up the tiger	60 mins	26.15 mins
Put the mug on the coffee maker	86 mins	38.22 mins
Pick up the package with both hands	90 mins	13.97 mins
Open the drawer	71 mins	16.95 mins
Turn the faucet off	104 mins	16.67 mins

Table D.6: Time taken per task to either collect 150 demos through teleoperation with one human operator or to generate 1000 synthetic demos with R2R2R. Note: R2R2R generation times do not include the upfront processing time until generation.

Config	Value
optimizer	AdamW [431]
base learning rate	2e-4
learning rate schedule	cosine decay [432]
batch size	64
weight decay	0.09
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$ [33]
warm up steps [433]	500
total steps	100,000
observation history	4
action dimension	20 (YuMi)
proprio format	absolute eef xyz, 6d rotation, absolute gripper position
action format	delta eef xyz, 6d rotation, absolute gripper position
action horizon	16
observation resolution	448

Table D.7: Diffusion Policy Hyperparameters

We investigated whether it was possible to reduce inference latency without retraining the core model.

Our solution is a VLM early-stopping trick that exploits the frequency coefficient token ordering in the FAST implementation (0th and lower harmonics first) and energy compaction property of the Discrete Cosine Transform (DCT). Robot action trajectories tend to be smooth and temporally correlated, meaning that most of their information is concentrated in the low-frequency coefficients. Instead of waiting for the model to decode the full set of coefficients, we stop decoding early, reconstruct the trajectory using only the first few frequency coefficients, and immediately issue the action.

This trades a small increase in trajectory reconstruction mean squared error (since high-

Config	Value
optimizer	AdamW [431]
base learning rate	2.5e-5
learning rate schedule	cosine decay [432]
batch size	32
weight decay	0.09
LoRA Rank	16
LoRA Alpha	16
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$ [33]
warm up steps [433]	1000
total steps	30,000
action/proprio dimension	16 (YuMi) 8 (Franka)
proprio format	absolute joints positions, absolute gripper position
action format	delta joints, absolute gripper position
action horizon	10
observation resolution	224

Table D.8: π_0-FAST hyperparameters

frequency details are dropped) for a substantial reduction in latency—often cutting inference time nearly in half. In practice, the essential structure of the action are preserved with just the first few DCT harmonics, making the trade-off favorable in latency-critical evaluation settings. Optional refinements, such as action-chunk ensembling or other smoothing techniques, can be added to compensate for missing fine-grained detail. We leave as future work more extensive evaluations of this test-time compute to reconstruction error trade-off.

Statistical Comparison Between Teleoperation and R2R2R Data Efficacy

To evaluate whether R2R2R-generated data yields performance comparable to human teleoperation, we apply the Two One-Sided Tests (TOST) procedure across all tasks and policies. Unlike traditional significance tests that ask whether two conditions differ, TOST tests whether the difference between them is small enough to be considered practically negligible. Specifically, we test whether the absolute difference in success rates falls within a $\pm 5\%$ margin—chosen to reflect a practically insignificant difference for robot policy success rates.

As shown in Table D.9, no individual task satisfies both conditions required for statistical equivalence (i.e., both p-values below 0.05). However, the results consistently show no strong evidence that either R2R2R or teleoperation outperforms the other. In particular, the global test across all tasks yields one p-value below 0.05 and one above, suggesting performance is similar but not provably equivalent under the chosen threshold. These results support the interpretation that R2R2R can match the effectiveness of teleoperation across the evaluated tasks, while offering a significantly more scalable method for data generation.

Task	**Policy**	**TOST lower p**	**TOST upper p**
Pick up the toy tiger	Diffusion Policy	0.2656	0.5359
	π_0-FAST (Finetuned)	0.6891	0.1429
Put the mug on the coffee maker	Diffusion Policy	0.5349	0.2712
	π_0-FAST (Finetuned)	0.6891	0.1429
Turn the faucet off	Diffusion Policy	0.6891	0.1429
	π_0-FAST (Finetuned)	0.3729	0.3729
Open the Drawer	Diffusion Policy	0.2656	0.5359
	π_0-FAST (Finetuned)	0.8051	0.0806
Pick up the package with both hands	Diffusion Policy	0.1429	0.6891
	π_0-FAST (Finetuned)	0.3955	0.3955
Overall (All Tasks)	–	0.4271	0.0497

Table D.9: Equivalence testing (TOST) between human teleoperation (150 trajectories) and R2R2R-generated data (1,000 trajectories). We report the p-values from Two One-Sided Tests (TOST) applied to each task and policy, using a $\pm 5\%$ success rate margin as the equivalence threshold. The "lower p" tests whether R2R2R performs *no worse* than teleoperation by more than 5%, while the "upper p" tests whether teleoperation performs *no worse* than R2R2R. Statistical equivalence is only confirmed when *both* p-values fall below 0.05.

Chapter E

Appendix E: CaP-X – A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation

E.1 Interactive Real-World Setup

In this section, we document the interactive real-world experience with CaP-Agent0 centered around a chat-based web UI, where the user can propose new tasks to the robot, view the step-by-step tool-use results, and offer feedbacks for multi-turn improvement. We will show how this system enables a real-world robot (AgiBot G1) to zero-shot complete novel tasks.

User Interface

On the chat UI shown in Figure E.1, the user start with choosing task configurations and models. Then they can view the scene description and code generated by the model. As the code is executed on the robot, they also see the step-by-step progress of each tool call, such as the segmentation masks returned by SAM3, Points annotated by Molmo 2, and wrist camera views when the robot arm moves. In between turns, the user can provide additional feedback to the model by typing in the chatbox on the bottom.

The UI also features a 3D visualization powered by Viser [1], which shows the live robot URDF, trajectories, and depth pointcloud.

Tools

CaP-Agent0, specifically on the AgiBot G1, has access to these APIs for perceiving, reasoning, and interacting.

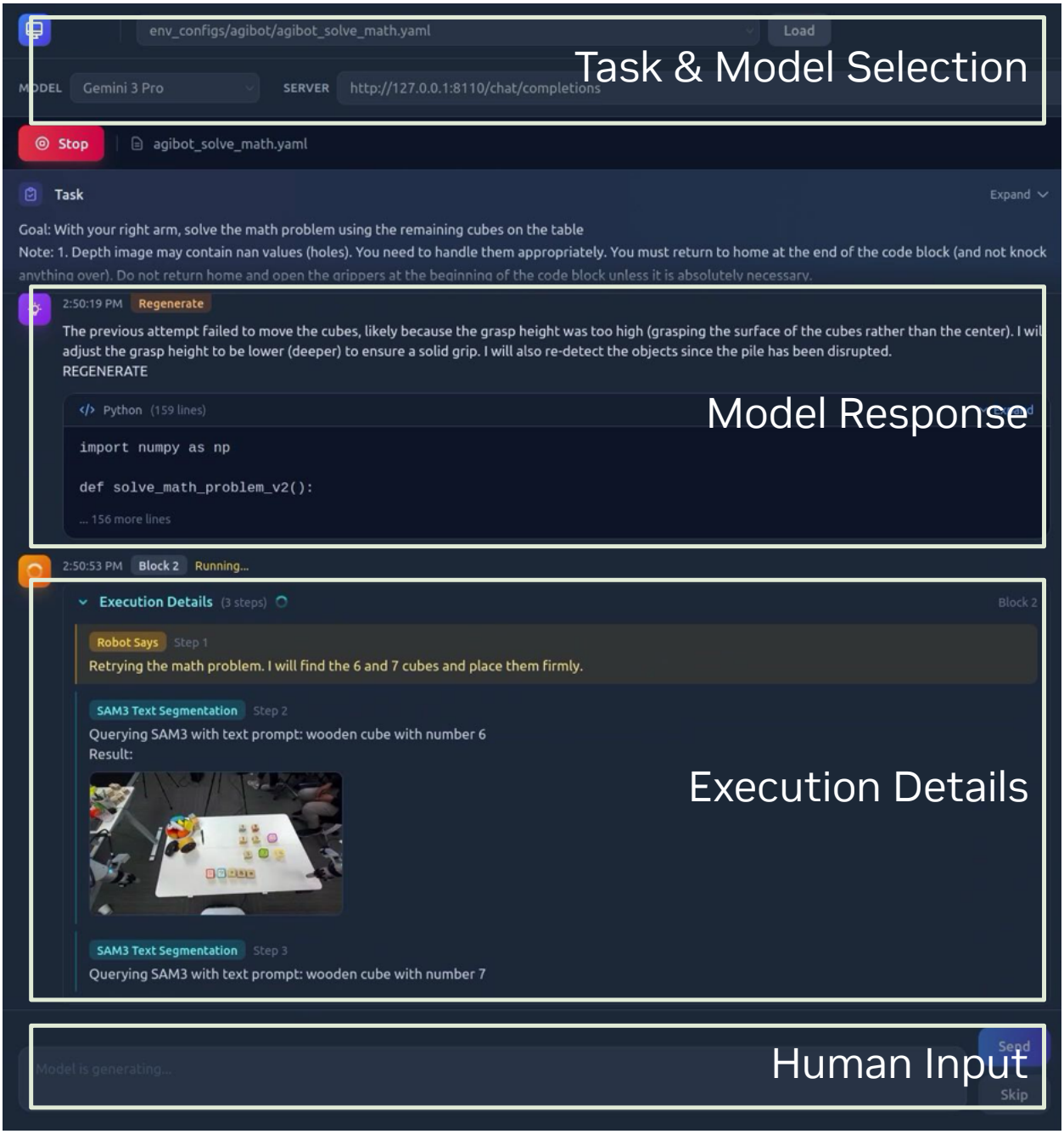


Figure E.1: An example of the web UI, where users can select the task and model at the top, view the model's responses and execution details in the middle, and provide instructions at the bottom.

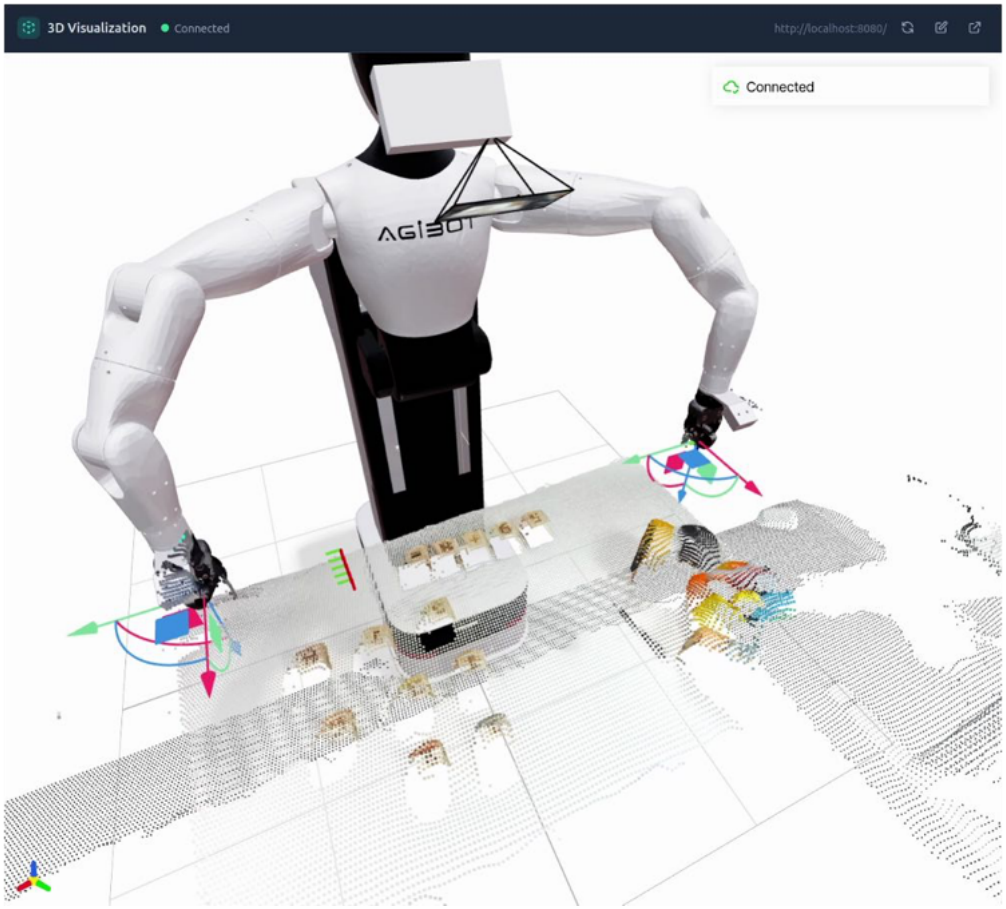


Figure E.2: Viser [1] visualization section of the web UI.

- `get_observation`: returns a dictionary of the current head camera image (RGB, depth, and intrinsics), left and right wrist camera images, end-effector 6-DoF force-wrenches, and joint configuration.
- `segment_sam3_text_prompt`: given an image and a text prompt, returns a list of segmentation masks and scores.
- `segment_sam3_point_prompt`: given an image and a point coordinate on the image, returns a list of segmentation masks and scores.
- `point_prompt_molmo`: given an image and a text prompt, returns a list of points on the image pointed out by Molmo 2.
- `open_gripper`: opens gripper.
- `close_gripper`: closes gripper.

- `goto_pose`: Plans and executes IK to a pose specified by a position and orientation, optionally with a pre-grasp offset and max force-wrench.
- `matrix_to_pose_wxyz_xyz`: Converts a 4×4 transformation matrix to a pose represented in a quaternion and a 3D position.
- `pose_wxyz_xyz_to_matrix`: Converts a pose to a transformation matrix.
- `euler_to_quaternion_wxyz`: Converts extrinsic XYZ Euler angles to a quaternion.
- `quaternion_wxyz_to_euler`: Converts quaternion of extrinsic XYZ Euler angles.
- `query_vlm`: Ask a question with text and images to an expert VLM model (Gemini-3-Pro)
- `go_forward`: Move forward 1 meter.
- `turn_left_45_degrees`: Turn left 45 degrees.
- `turn_right_45_degrees`: Turn right 45 degrees.
- `goto_planar_position`: Plan and execute IK to a position with planar motion constraint, but no orientation (yaw) requirement.
- `convert_depth_to_pointcloud`: Given a depth image, intrinsics, and extrinsics, convert the depth image into a pointcloud.
- `forward_kinematics`: Given a robot configuration, compute the end-effector pose.
- `overlay_eef_axis_on_image`: Visualize end-effector pose on a camera image by rendering the 3D axis on top.
- `overlay_segmentation_masks`: Visualize segmentation masks on an image.
- `say_something`: Convey some intent to the user.

Among the tools, the 3D rigid body transformation helpers are provided although technically not strictly necessary. We find that when not provided, these functions are almost always written by the models who make mistakes from time to time.

Visualization tools such as `overlay_eef_axis_on_image` and `overlay_segmentation_masks` enables advanced multi-modal reasoning when the output images are passed to an expert VLM via `query_vlm`. This allows, for example, code execution conditioning based on if the expert VLM answered "which of these segmentation masks should I pick?".

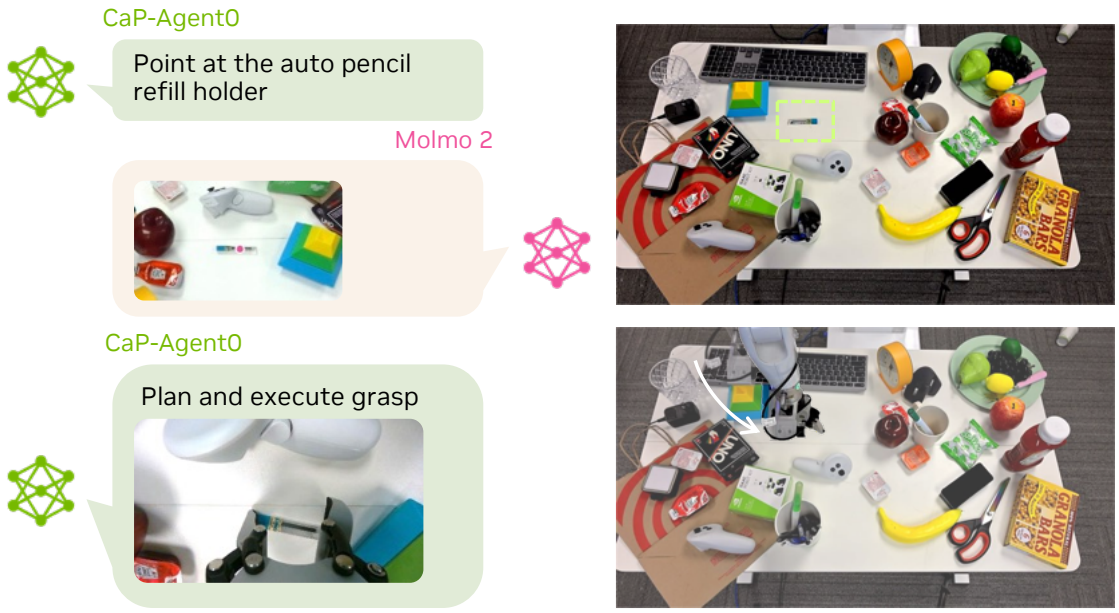


Figure E.3: An example of CaP-Agent0 locating and grasping an auto pencil refill holder, where it uses Molmo2 to locate the object.

Real World Tasks

In this section, we show visualizations of CaP-Agent0 completing tasks in the real world. All experiments are *zero-shot*. The agent's execution details are condensed and presented as pseudo sub-steps, while CaP-Agent0 writes the underlying code in greater detail using tools available in E.1. Raw code example is available in Appendix E.5.

Needle in a Haystack

In this task, we evaluate the ability of CaP-Agent0 to find "needles in a haystack." Specifically, in a cluttered scene containing diverse objects, the robot is tasked with locating and grasping an auto pencil refill holder, a relatively uncommon item. Such uncommon objects are often challenging for end-to-end learning policies such as VLAs. In contrast, CaP-Agent0 leverages pretrained VLMs to successfully localize and retrieve the target object, as shown in Figure E.3.

Mechanical Search

In this task, three inverted cups are placed on a table, and the robot is instructed to retrieve a green lime hidden beneath one of them. This occluded object retrieval problem is commonly referred to as mechanical search [434]. Because the lime is concealed by the cups, the robot must systematically explore each cup to locate it. A representative planning and execution example is shown in Figure E.4.

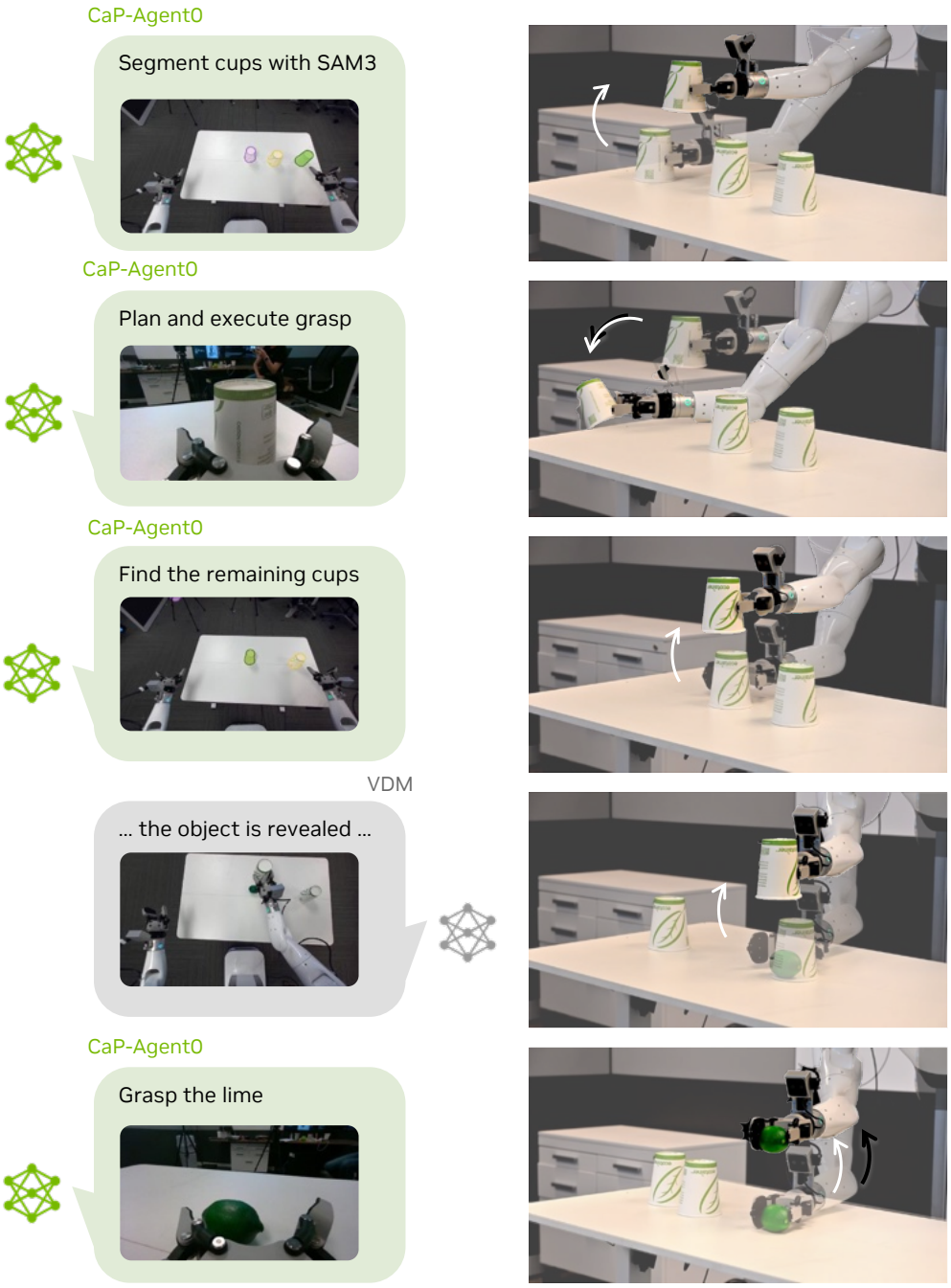


Figure E.4: Mechanical search planning and execution example. The left column shows the agent planning process and the right column shows the physical robot execution.

Multimodal Symbolic Reasoning

In this task, we evaluate the multimodal symbolic reasoning capabilities of CaP-Agent0 by requiring it to solve a mathematical equation formed by numbered wooden blocks. The robot

must first perceive the scene to interpret the equation and then grasp the correct block and place it in correct location to complete the solution, as shown in Figure E.5.

Learning from Human Feedback

CaP-Agent0 is able to incorporate human feedback and adjust its code generation accordingly to complete the task. As shown in Figure E.6, the robot is instructed to pick up an apple but fails in the initial attempt due to an excessively high grasp pose. After receiving the feedback "grasped the apple too high," CaP-Agent0 modifies the generated code and successfully completes the task on the second attempt.

Embodied Reasoning

This task evaluates the embodied reasoning ability of CaP-Agent0, where the robot is asked to "stack objects as high as possible". Because the scene contains both square and round objects, the robot must reason about their physical properties to determine a stable stacking strategy. As shown in Figure E.7, CaP-Agent0 learns to place round objects on top of square ones to maximize stability.

Tool Generalization with Domain Knowledge

This task demonstrates the versatility of the code-as-policy interface, where CaP-Agent0 can use tools from arbituary Python packages to assist with the task. We ask the robot to "take the elevator downstairs", which suggests locating and pressing the button, and moving into the elevator. Since the robot is positioned at an angle with respect to the wall, it is not immediately obvious in which direction the button should be pushed. CaP-Agent0 is able to invoke SciPy RANSAC algorithm on the segmented wall pointcloud, and compute the surface normal direction, shown in Figure E.8.

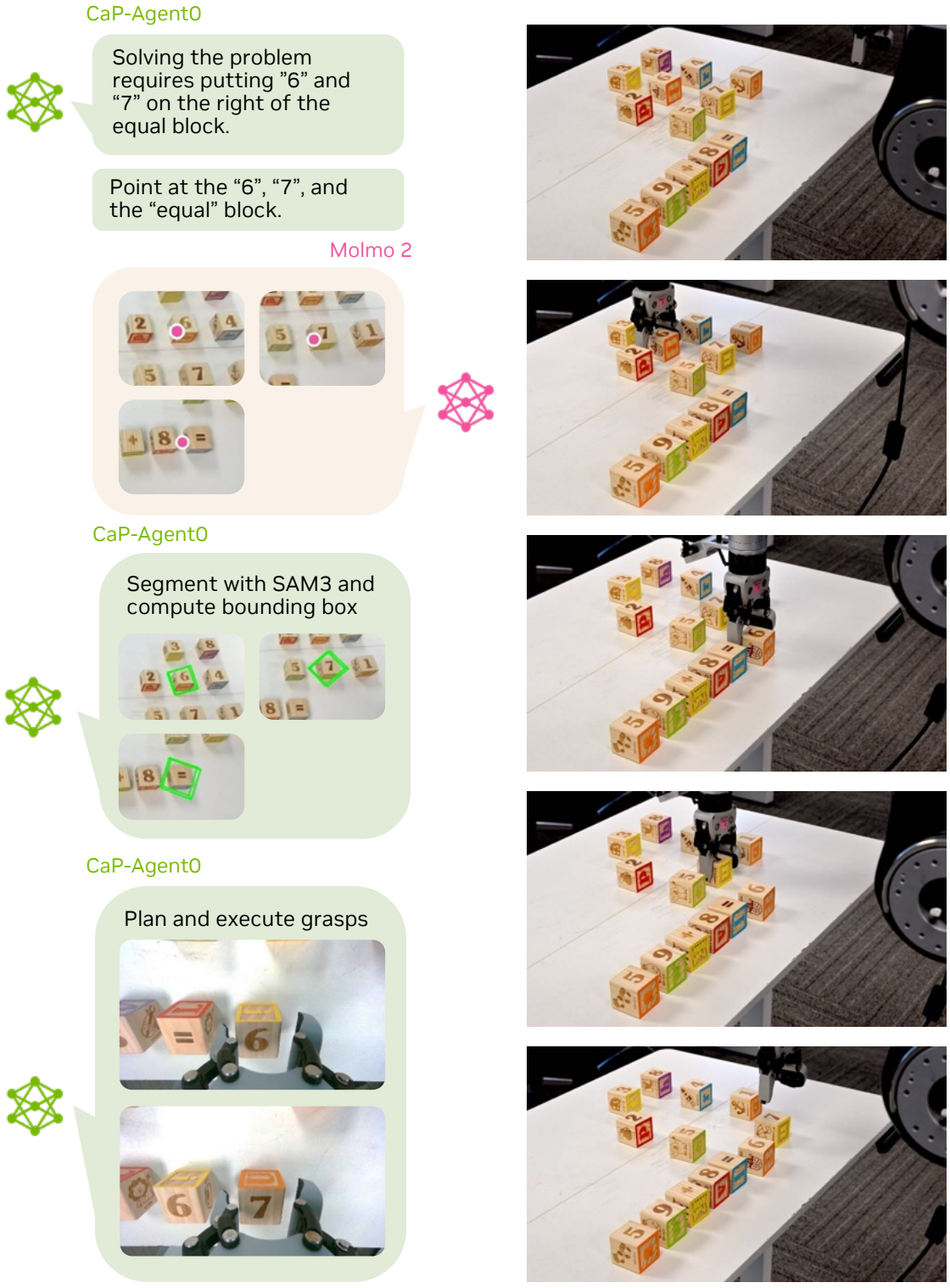


Figure E.5: Example of a Multimodal Symbolic Reasoning task, where the robot is asked to solve the problem of 59 plus 8.

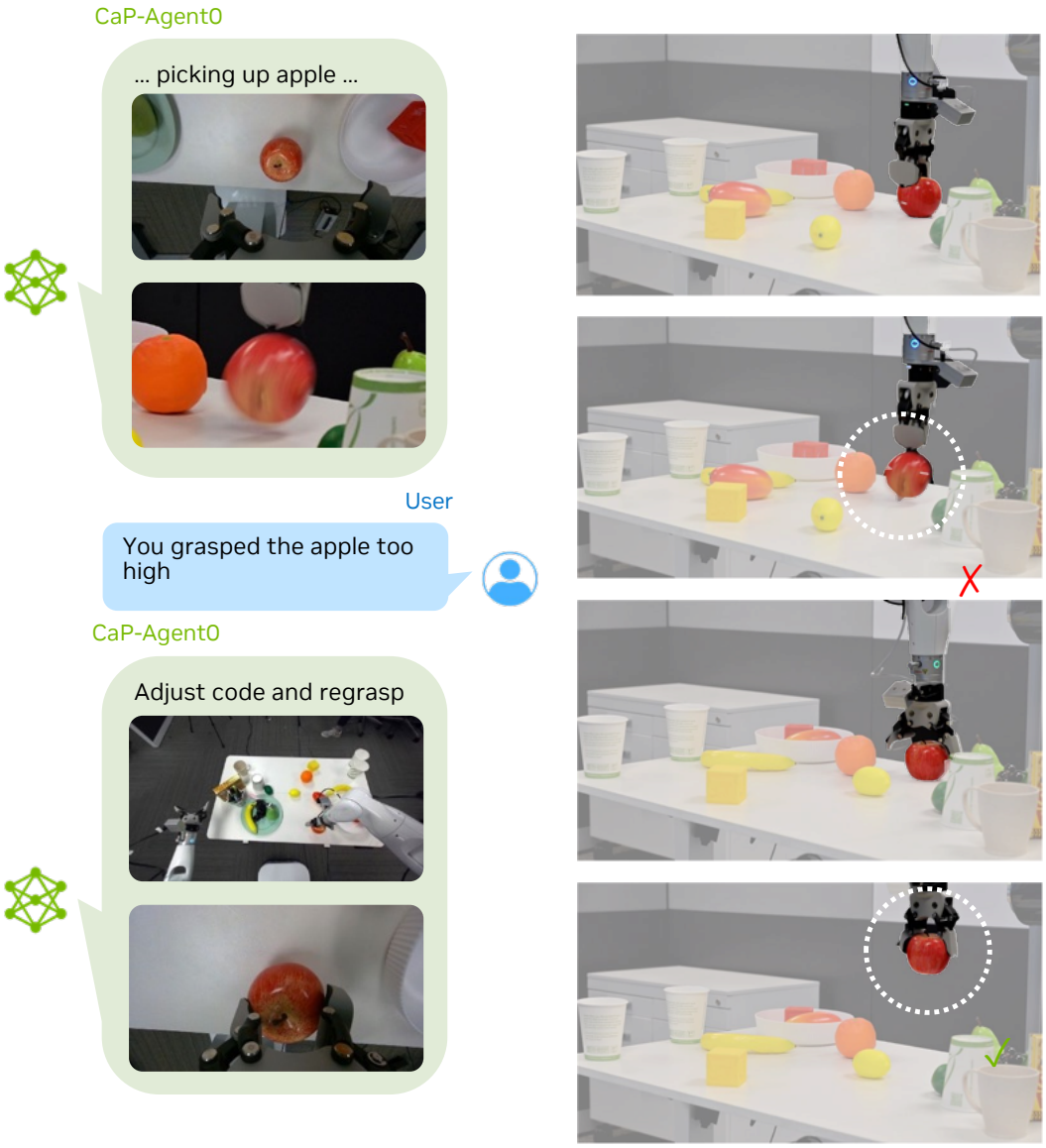


Figure E.6: Example of picking up an apple. While the first attempt fails, CaP-Agent0 takes human feedback as input, adjusts the code and completes the task successfully.

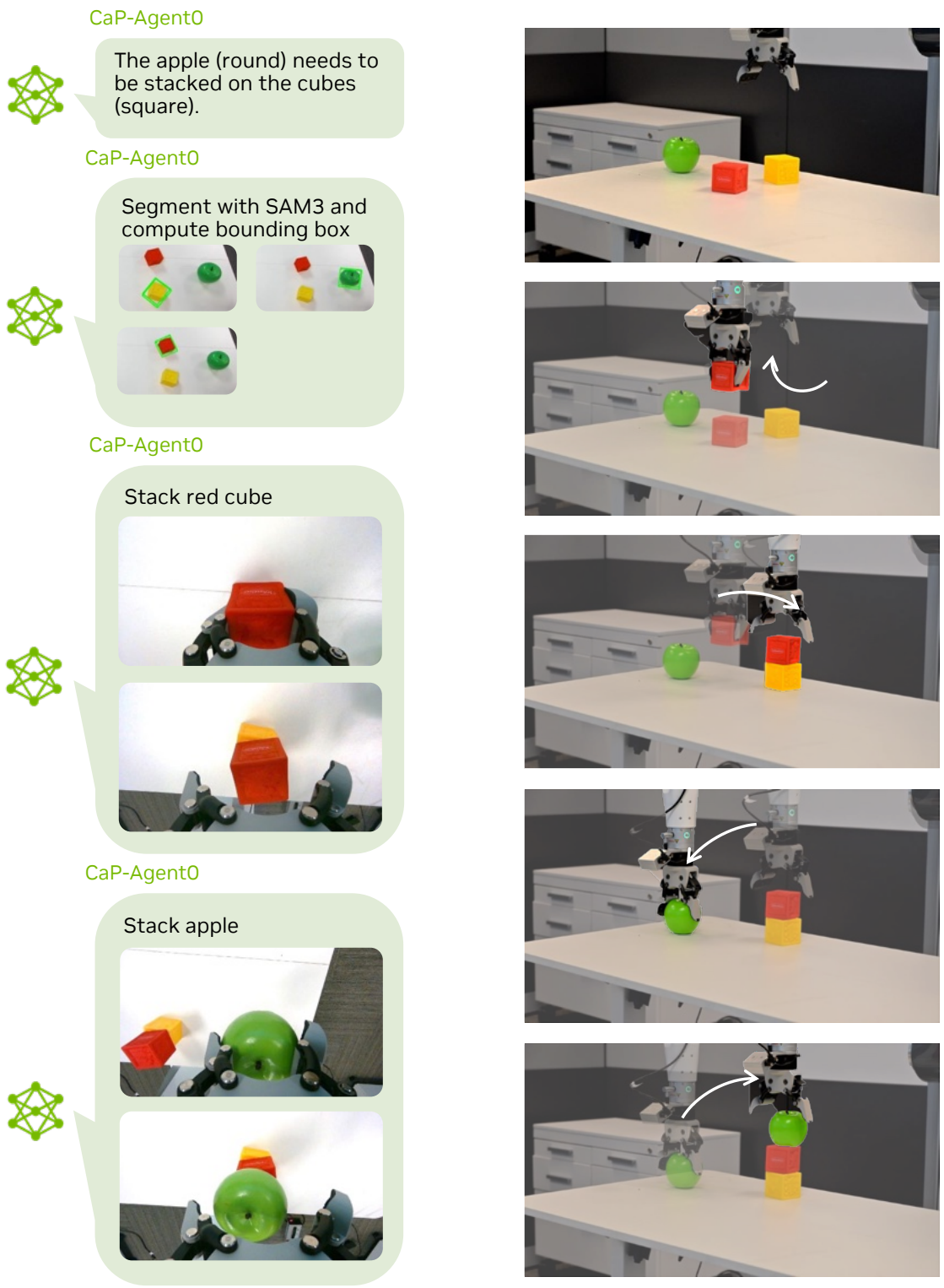


Figure E.7: Example of CaP-Agent0 stacking round objects on top of square objects to form the tallest possible stable stack.

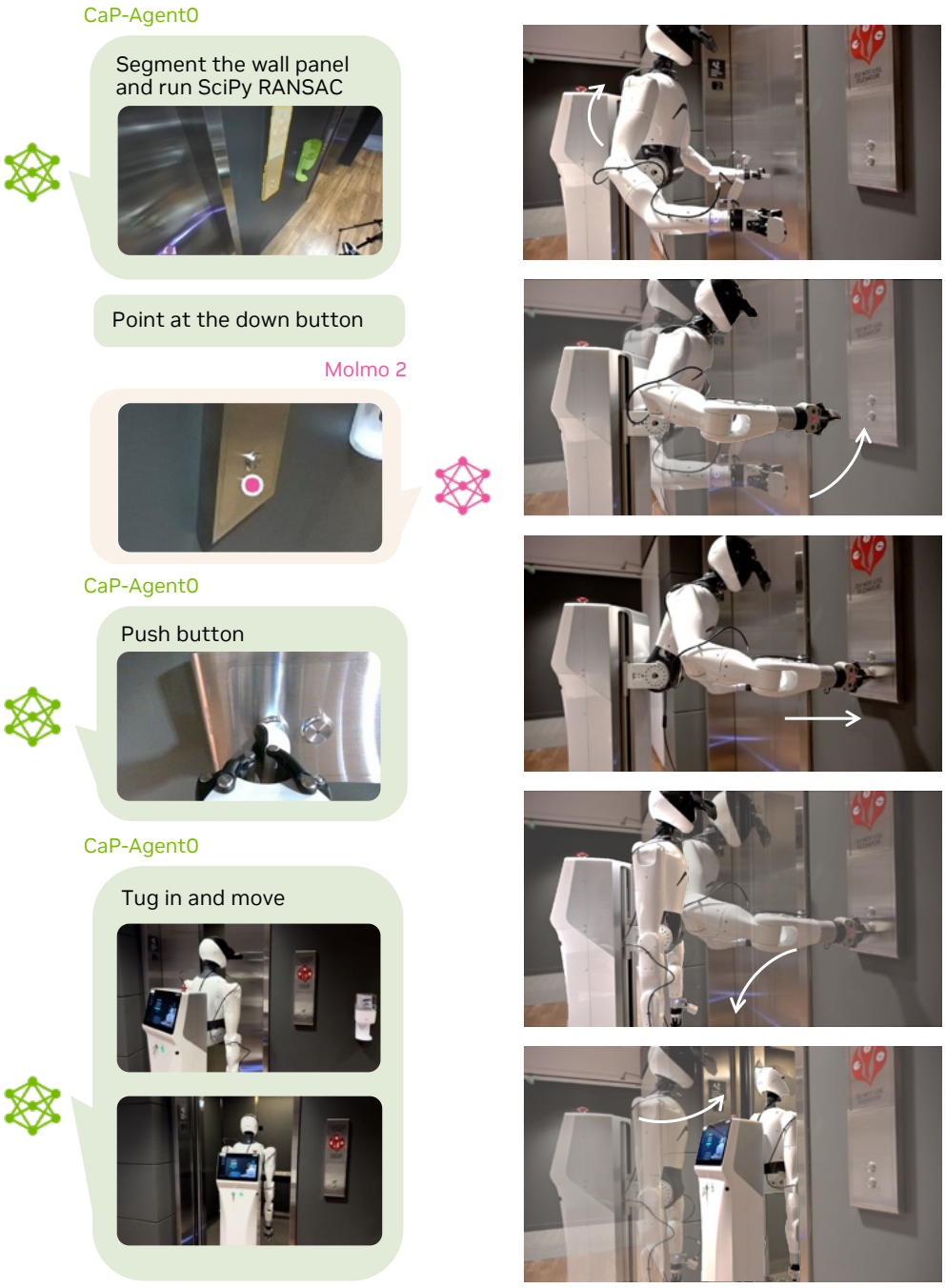


Figure E.8: Example of robot pushing the elevator button.

E.2 Full Benchmark Table

We present the full results from CaP-Bench in Figure E.9. From top to bottom are code compilation success rate, average dense reward, and average task success rate.

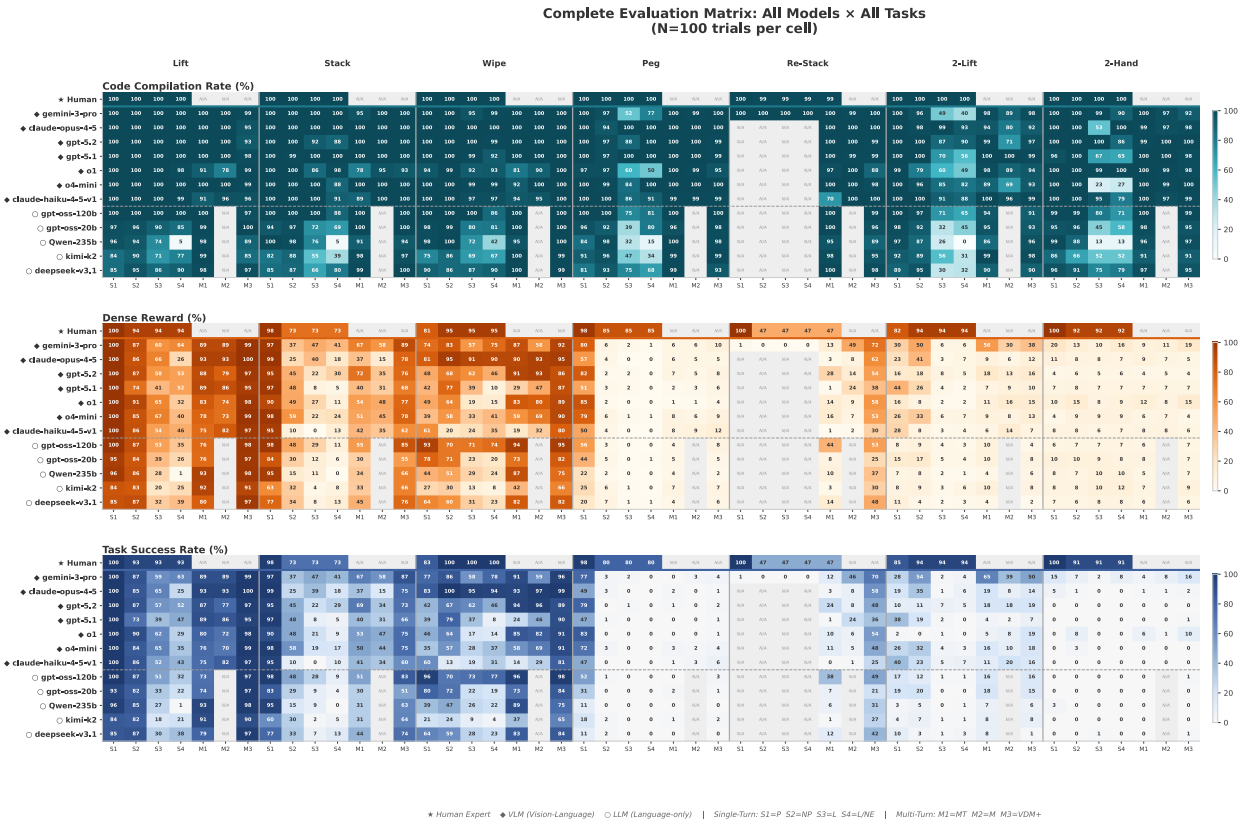


Figure E.9: Full CaP-Bench results. From top to bottom are code compilation success rates, average task reward, and average task success rate. Note that for `Re-Stack`, with a simple task prompt, without visual input, we do not observe model writing code to first check the initial condition and then perform the task, resulting in failures. Therefore, we only evaluatd all models for the multi-turn setup.

E.3 Additional Takeaways

In-Context Examples of API Usage improves performance. In the docstring of each API used in S4, we provided a brief description of the API, the expected datatype, and shape for each input and output. However, due to low code compilation successes from open source models in the initial experiments, we hypothesize that additional prompting may be necessary to increase model performance. Therefore, in addition to the components above, we added API usage examples in the docstring.

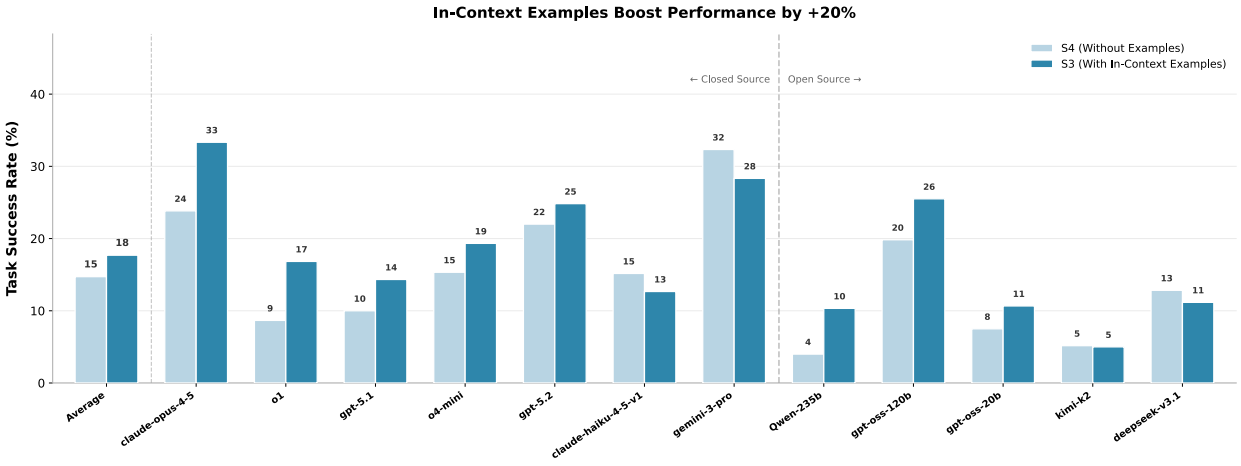


Figure E.10: Per-model breakdown of usefulness of in-context API Usage

We present the results in Figure 10.3 and Figure 10.4, and a per-model breakdown in Figure E.10. We observe that almost all models benefited from adding additional context of examples of how to use the low-level APIs.

E.4 Qualitative Analysis of CaP-RL Post-Training Effects

In this section, we analyze the qualitative behavioral shifts induced by on-policy reinforcement learning (GRPO) on the `Qwen2.5-Coder-7B-Instruct` base model. We focus on the *Stack Red Cube on Green Cube* task to illustrate how RL mitigates fundamental instruction-following failures and induces robust geometric reasoning. Methodologically, CaP-RL leverages the privileged S1 tier to stabilize training by removing perception noise. Below we show the code block examples before and after CaP-RL Post-Training.

Before CaP-RL Post-Training Model Behavior. Qwen2.5-Coder-7B-Instruct base model code generation example on the task *Stack Red Cube on Green Cube* prior to on-policy reinforcement learning with CaP-RL:

```
import numpy as np

# Get the pose of the green cube
green_cube_position, green_cube_quaternion_wxyz, _ = get_object_pose("green cube")

# Get the pose of the red cube
red_cube_position, red_cube_quaternion_wxyz, _ = get_object_pose("red cube")

# Calculate the placement position for the red cube
placement_position = np.array(green_cube_position) + np.array([0, 0, 0.05])  # 5 cm above the green cube

# Move the robot to the placement position
goto_pose(placement_position, (0, 0, 1, 0), z_approach=0.1)  # Use a slight z_approach for precision

# Open the gripper to release the red cube
open_gripper()
```

After CaP-RL Post-Training Model Behavior. Code generation example of Qwen2.5-Coder-7B-Instruct model after on-policy reinforcement learning with CaP-RL on the task *Stack Red Cube on Green Cube* :

```
import numpy as np

# Get the positions and extents of the red and green cubes
red_cube_pos, _, red_cube_extent = get_object_pose("red cube", return_bbox_extent=True)
green_cube_pos, _, green_cube_extent = get_object_pose("green cube", return_bbox_extent=True)

# Calculate the placement position on top of the green cube
placement_pos = green_cube_pos + np.array([0, 0, red_cube_extent[2] / 2 + green_cube_extent[2] / 2])
```

```
9
10 # Sample a grasp pose for the red cube
11 grasp_pos, grasp_quat = sample_grasp_pose("red cube")
12
13 # Move to the grasp pose and approach
14 goto_pose(grasp_pos, grasp_quat, z_approach=0.1)
15
16 # Close the gripper to grasp the red cube
17 close_gripper()
18
19 # Move to the placement position
20 goto_pose(placement_pos, (0, 0, 1, 0), z_approach=0.1)
21
22 # Open the gripper to release the red cube
23 open_gripper()
```

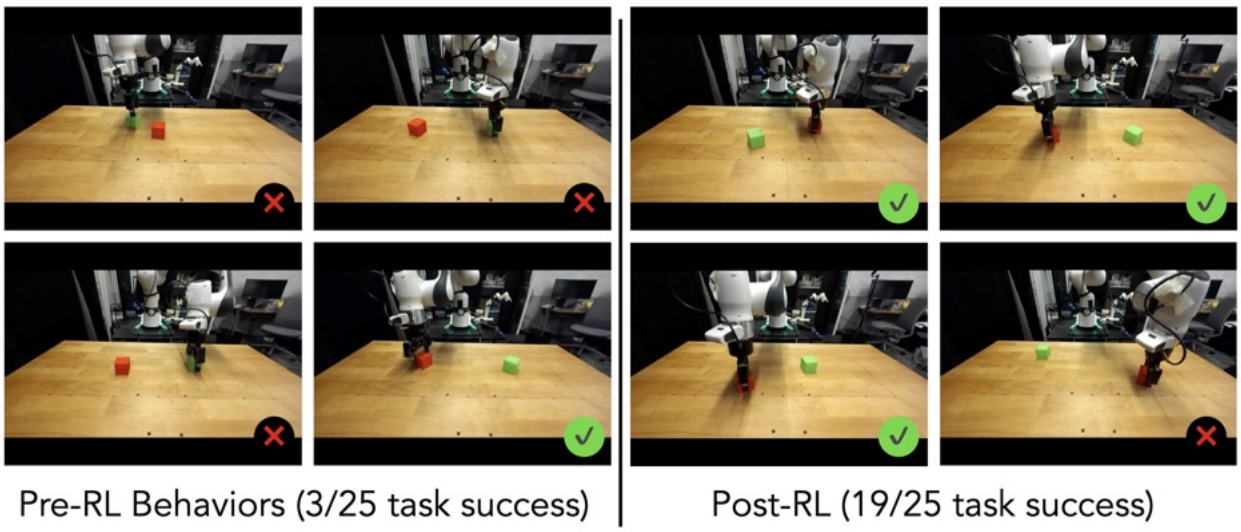


Figure E.11: CaP-RL Finetuned Qwen-2.5-Coder-7B-Instruct deployed on the real task *put the red cube on the green cube*. Human Oracle code gets 21/25 task successes in real. Prior to reinforcement learning, we see behaviors such as grasp approaches on the green cube as the first action, rather than a more sensible approach on the red cube first.

Pre-RL Failure Modes: Step Skipping and Hallucinated State. Prior to RL post-training, the base model frequently exhibits "step skipping", a failure mode where the agent attempts to satisfy the final goal state without executing the necessary prerequisites. As shown in the pre-RL snippet, the model correctly identifies the target location (`placement_position`) but fails to grasp the red cube. It attempts to move the gripper directly to the placement target and open it, seemingly hallucinating that it is already holding the object.

Post-RL Improvement: Causal Sequencing and Geometric Generalization. After GRPO training, two critical improvements emerge:

1. **Causal Sequencing:** The model correctly synthesizes the full manipulation chain: *Identify* → *Grasp* → *Transport* → *Release*. It learns through environment interaction and reward feedback signals that the causal dependency that an object must be grasped (via `close_gripper`) before it can be placed.

2. **Dynamic Geometric Reasoning:** Instead of using hard-coded offsets, the RL-trained model utilizes the `return_bbox_extent=True` parameter to dynamically calculate the stacking height based on object dimensions (`red_extent[2]/2 + green_extent[2]/2`). This indicates a shift from memorization to grounded geometric reasoning.

Zero-shot Generalization to Non-privileged Physical World Setups. Although the model is only post-trained on the privileged S1 tier, the resulting policies demonstrate robust zero-shot transfer to the non-privileged S2 tier. Consequently, the model functions effectively in physical real-world setups, as shown in Figure E.11.

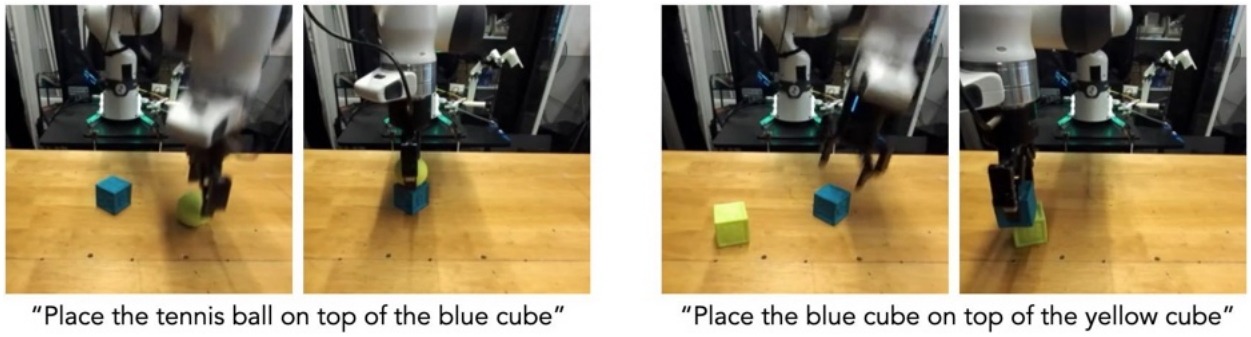


Figure E.12: CaP-RL Finetuned Qwen-2.5-Coder-7B-Instruct successfully deployed on similar task variants. Demonstrates that base model is still capable of instruction following.

Zero-Shot Generalization to Task Variants. We observe instruction-following and zero-shot generalization to similar task variants after RL post-training, as seen in E.12. The same policy successfully executes variants such as *"put the tennis ball on the green cube"* or tasks involving randomized object colors, as the underlying logic relies on abstract properties (bounding boxes) rather than overfitting to specific entity names or training instance scales.

E.5 CaP-Agent0 Case Studies

Real-World Case Studies

Common Sense Physics-Aware Task Decomposition

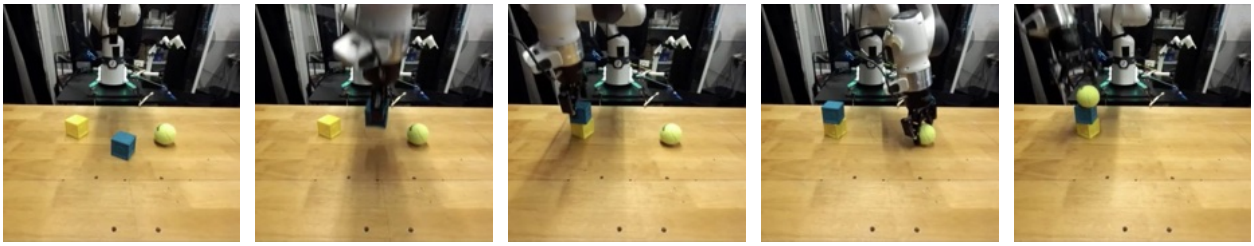


Figure E.13: "Stack these as high as you can."

We validate CaP-Agent0 on a real-world Franka Panda platform using a set of geometrically heterogeneous objects: a yellow cube, a blue cube, and a tennis ball. The agent was given the open-ended instruction: *"stack these as high as you can."* with no additional prompt context outside of API usage documentation and the multimodal agent initial scene description.

Crucially, the task prompt provides neither a specific stacking sequence nor *any* semantic description of the scene or its objects. While a standard visuomotor policy might attempt to stack objects in detection order, potentially attempting to balance a cube on top of the spherical ball, CaP-Agent0 demonstrated common-sense embodied physical reasoning by deriving the only stable construction order.

This behavior is enabled by the CaP-Agent0 auxiliary VDM multimodal agent that extracts task-relevant image information before code generation begins. Prompted to "describe the initial state of the environment with the goal of the task in mind," the VLM emits the following context for the coding agent's initial turn:

> *"...The cubes appear to have flat surfaces suitable for stacking, while the tennis ball is spherical and would likely need to be placed on top or handled carefully."*

Leveraging this textual grounding, the coding agent explicitly decomposed the problem based on stability constraints. The full code generated by the CaP-Agent0 agent for completing this task is found in Appendix E.8.

Implicit Multi-Step Reasoning for Obstructed Goals

We evaluate *CaP-Agent0* on a task where the *Franka Panda* must place a blue cube on top of a yellow cube. In the initial state, the cubes form a three-object tower: blue is at the base, supporting yellow, which in turn supports a green cube (Figure E.14).

Figure E.14: "Place the blue cube on top of the yellow cube."

The goal prompt *"place the blue cube on top of the yellow cube"* is linguistically underspecified; it contains no mention of the green obstruction nor the fact that the blue target currently supports the yellow destination.

CaP-Agent0 resolves this dependency through its auxiliary multimodal perception layer. Before code generation, the VLM provides the coding agent with a high-level strategic prior:

> *"...the blue cube is at the bottom of the stack, underneath the yellow cube, and the yellow cube is underneath a green cube. This implies a complex manipulation task where the robot will likely need to: 1. Remove the top (green) cube. 2. Remove the middle (yellow) cube to access the blue one, or move the stack. 3. Ultimately re-arrange them such that the blue cube ends up on top of the yellow one. The straightforward description is that the target blue cube is currently obstructed and supporting the target destination object (the yellow cube)."*

This behavior demonstrates that programmatic agents, when augmented with a dedicated perception-to-text grounding layer, can derive complex manipulation strategies traditionally requiring extensive task-specific imitation learning data or manual state-machine design. The full executable rollout is provided in Appendix E.8.

Simulation Case Studies

Verification and Retry Behavior Enabled by Multiturn Design

This case study comes from the "pick up the red cube" task from Robosuite ported into CaP-Gym. Unlike one-shot code generation frameworks, the multi-turn architecture of CaP-Agent0 allows for emergent agentic resilience. In several simulation trials, we observed instances where a planned grasp was physically executed but failed to successfully lift the object due to physics-engine stochasticity or sub-optimal contact points. Rather than terminating the task upon failure, the agent utilized the subsequent turn's observation to verify that the goal condition (e.g., $z_{object} > z_{table}$) remained unfulfilled. As shown in the generated logic, the agent autonomously synthesized a recovery routine branch. This capacity to reason about execution failures and reset the environment state demonstrates that CaP-Agent0 can function as a closed-loop reactive agent rather than a static script generator.

```
    ...

masks = segment_sam3_text_prompt(rgb, "red cube")

is_lifted = False
if not masks:
    print("Could not find red cube to verify state. Assuming failed and moving to home pose.")
    is_lifted = False
else:
    best_mask = max(masks, key=lambda x: x["score"])["mask"]

    y_idxs, x_idxs = np.where(best_mask)
    if len(y_idxs) > 0:
        stride = 10
        y_sample = y_idxs[::stride]
        x_sample = x_idxs[::stride]

        z_sample = depth[y_sample, x_sample]
        if z_sample.ndim > 1: z_sample = z_sample.flatten()

        # Filter invalid depth
        valid = z_sample > 0
        if np.any(valid):
            z_sample = z_sample[valid]
            x_sample = x_sample[valid]
            y_sample = y_sample[valid]

            fx = intrinsics[0, 0]
            fy = intrinsics[1, 1]
            cx = intrinsics[0, 2]
            cy = intrinsics[1, 2]

            x_c = (x_sample - cx) * z_sample / fx
            y_c = (y_sample - cy) * z_sample / fy

            points_c = np.stack([x_c, y_c, z_sample], axis=1)
            points_c_homo = np.hstack([points_c, np.ones((len(points_c), 1))])
            points_w = (cam_pose @ points_c_homo.T).T[:, :3]

            mean_z = np.mean(points_w[:, 2])
            print(f"Estimated mean Z of object: {mean_z:.4f}")

            # If Z > 0.15, we assume it's lifted (table usually ~0, cube size ~0.05)
            if mean_z > 0.15:
                is_lifted = True
                print("Object is confirmed to be lifted.")
            else:
```

```
                print("Object appears to be on the table.")

if is_lifted:
    print("Task completed successfully.")
else:
    print("Retrying grasp sequence...")
    open_gripper()

    # Move to home pose to get a better view (avoid occlusion from gripper)
    home_pos = np.array([0.5, 0.0, 0.5])
    home_quat = np.array([0.0, 1.0, 0.0, 0.0]) # w,x,y,z (Approx gripper pointing down)

    move_to_joints(solve_ik(home_pos, home_quat))

    # Re-perceive
    obs = get_observation()
    rgb = obs["robot0_robotview"]["images"]["rgb"]
    depth_im = obs["robot0_robotview"]["images"]["depth"]
    intrinsics = obs["robot0_robotview"]["intrinsics"]
    cam_pose = obs["robot0_robotview"]["pose_mat"]

    masks = segment_sam3_text_prompt(rgb, "red cube")
    if not masks:
        print("No red cube found after reset.")
    else:
        best_mask = max(masks, key=lambda x: x["score"])["mask"]
        if depth_im.ndim == 3: depth_map = depth_im[:, :, 0]
        else: depth_map = depth_im

        grasp_poses, grasp_scores = plan_grasp(depth_map, intrinsics, best_mask.astype(np.int32))

        if len(grasp_poses) == 0:
            print("No grasps found.")
        else:
            best_idx = np.argmax(grasp_scores)
            best_pose_cam = grasp_poses[best_idx]

            pose_world = cam_pose @ best_pose_cam
            target_pos = pose_world[:3, 3]
            target_rot = pose_world[:3, :3]
            target_quat = rot_to_quat(target_rot)

            # Grasp execution
            # 1. Move to Pre-grasp
            pre_pos = target_pos + np.array([0, 0, 0.1])
            move_to_joints(solve_ik(pre_pos, target_quat))

```

```
            # 2. Move to Grasp
            move_to_joints(solve_ik(target_pos, target_quat))

            # 3. Close
            close_gripper()

            # 4. Lift
            lift_pos = target_pos + np.array([0, 0, 0.25])
            move_to_joints(solve_ik(lift_pos, target_quat))
            print("Retry finished.")

    ...
```

Perception Module Cascade Failure Fallbacks

This case study comes from the "wipe up the brown spill" task from Robosuite ported into CaP-Gym. The model frequently constructs robust perception pipelines that utilize hierarchical fallbacks. It understands that a generic text prompt might fail or be imprecise. In several trials, the model employs a VLM (Molmo) to visually ground the target ("brown spill") into a specific 2D point, and then uses that point to prompt a segmentation model (SAM3) for a precise mask. If this high-precision method fails, it falls back to broader text prompts like "stain" or "brown object".

```
    ...

    # 2. Try detection using Molmo (VLM) for better semantic understanding
    print("Attempting to detect spill using Molmo...")
    molmo_res = point_prompt_molmo(rgb, "brown spill")
    # Molmo returns dict[str, tuple], getting the first/only value
    spill_pt = None
    if molmo_res:
        # Check if we got a valid coordinate
        for key, pt in molmo_res.items():
            if pt[0] is not None and pt[1] is not None:
                spill_pt = pt
                print(f"Molmo found spill at: {spill_pt}")
                break
    masks = []
    if spill_pt:
        # Use the point to segment with SAM3
        print(f"Segmenting with SAM3 point prompt at {spill_pt}")
        masks = segment_sam3_point_prompt(rgb, spill_pt)
    else:
        # Fallback to text prompt with different wording if Molmo fails
        print("Molmo failed. Trying SAM3 text prompt 'stain'...")
        masks = segment_sam3_text_prompt(rgb, "stain")
        if not masks:
            print("Fallback to 'brown liquid'...")
            masks = segment_sam3_text_prompt(rgb, "brown liquid")
```

```
    if not masks:
        print("No masks found after all attempts. Cannot proceed.")
        return
    ...
```

Geometric Reasoning for Grasp Selection

This case study illustrates how CaP-Agent0 enhances sub-symbolic planners by injecting context-aware strategy. While modules like *Contact-GraspNet* infer many 6D grasp poses along with confidence scores, the learned scoring lacks awareness of the broader task setting. For this *tabletop* scenario, the agent correctly identified that a top-down approach strategy is inherently more reliable than side-approaches, regardless of their raw grasp scores.

To enforce this strategy, the agent autonomously synthesized a geometric constraint wrapper. By computing the alignment between grasp approach vectors and the world vertical, the generated code filtered out high-scoring but suboptimal side-grasps, explicitly selecting a vertical candidate to maximize stability. The agent also synthesized an autonomous fallback branch: if this strict strategy yielded no candidates, the logic reverted to the global maximum score to prevent execution stagnation. This demonstrates the agent's capacity to wrap generic tools in context-specific logic, bridging the gap between raw geometric perception and high-level task semantics.

Crucially, the automatically synthesized skill library (Appendix E.7) contains an analogous grasp filter, serving as a primary example of the agent's capacity to synthesize task-agnostic geometric logic into its persistent codebase.

```
    ...

    # 2. Grasp Planning
    print("Planning grasp...")
    grasp_poses, grasp_scores = plan_grasp(depth, intrinsics, seg_map)
    if len(grasp_poses) == 0:
        print("Error: No grasp poses found.")
        return

    # 3. Filter and Selection
    # Transform all grasps to world frame to check orientation
    best_score = -1.0
    best_grasp_world = None
    # We iterate through high-scoring grasps
    indices = np.argsort(grasp_scores)[::-1] # Descending order
    print(f"Found {len(indices)} candidates. Evaluating top 10...")
    for idx in indices[:20]:
        grasp_camera = grasp_poses[idx]
        grasp_world = transform_camera_to_world(grasp_camera, camera_pose)
        # Check approach vector (Z-axis of rotation matrix)
        # We want the gripper to approach somewhat downwards (negative Z
    in world)
        rot_mat = grasp_world[:3, :3]
```

```
        approach_vec = rot_mat[:, 2] # The Z-axis
        # Dot product with world down vector (0, 0, -1)
        # If dot product is positive (angle < 90), it's pointing somewhat down.
        # Actually, if approach is (0,0,-1), dot with (0,0,-1) is 1.
        # We want approach_z to be negative.
        if approach_vec[2] < -0.5: # At least 60 degrees downward tilt
            print(f"Candidate {idx}: Score {grasp_scores[idx]:.3f}, Approach Z: {approach_vec[2]:.3f} (Good)")
            best_grasp_world = grasp_world
            break
        else:
            # print(f"Candidate {idx}: Score {grasp_scores[idx]:.3f}, Approach Z: {approach_vec[2]:.3f} (Skipping, too horizontal)")
            pass
    if best_grasp_world is None:
        print("Warning: No steep top-down grasp found. Falling back to highest score.")
        best_grasp_world = transform_camera_to_world(grasp_poses[indices[0]], camera_pose)
    target_pos = best_grasp_world[:3, 3]
    target_rot = best_grasp_world[:3, :3]
    target_quat = rotation_matrix_to_quaternion(target_rot)
    ...
```

Code Generation Ensembling Increases Robustness

In this case study, we analyze the differences in code generation for a VDM agent which uses a single query vs a model ensemble. All generated code is for the cube restack task. The following is a portion of the initial generation from single query.

```
    ...
    import numpy as np

    # Step 1: Unstack the green cube from the red cube.
    # Get grasp pose for the green cube (currently on top)
    green_grasp_pos, green_grasp_quat = sample_grasp_pose('green cube')

    # Approach and pick up the green cube
    open_gripper()
    goto_pose(green_grasp_pos, green_grasp_quat, z_approach=0.1)
    close_gripper()
    ...
```

Notice how there is no fallback for the `sample_grasp_pose()` generation. When the initial code was executed, the outputted grasp was invalid. The model reasoned that the grasp was invalid because it was too close to the origin, so in the subsequent generation, the model retroactively implemented a fallback which only prevents the failure case it just encountered.

```
    ...
```

```
    # Based on the logs, the previous execution seems to have proceeded through the logic,
    # but looking at the sampled grasp pose for the red cube: [ 0.04235514  0.08625826 -0.04337643]
    # compared to the get_object_pose for red cube: [ 0.68666898 0.12055038 -0.09239209]
    # The sampled grasp pose for the red cube seems completely wrong (close to robot base origin 0,0,0).
    # This likely caused the robot to fail picking up the red cube or crash/move incorrectly in Step 2.
    ...
    # Get poses
    red_pos, _, _ = get_object_pose('red cube')
    green_pos, _, green_bbox = get_object_pose('green cube', return_bbox_extent=True)

    # Sample grasp for red cube
    red_grasp_pos, red_grasp_quat = sample_grasp_pose('red cube')

    # Sanity check: if grasp pose is near 0,0,0, fall back to object pose + offset
    if np.linalg.norm(red_grasp_pos) < 0.2:
        # Fallback: use object position directly for grasp center, oriented top-down
        print("Warning: Sampled grasp pose seems invalid. Using object position fallback.")
        red_grasp_pos = red_pos.copy()
        red_grasp_quat = np.array([0, 0, 1, 0]) # Top down for fallback
    ...
```

On the other hand, generations from a model ensemble tend to anticipate failure cases and preemptively implement fallbacks as shown through the following initial generation:

```
    ...
    def pick_object(mask, depth, intrinsics, extrinsics, z_lift=0.2):
        """Plans and executes a grasp for the object defined by the mask."""
        # Plan grasp
        grasps, scores = plan_grasp(depth, intrinsics, mask.astype(np.int32))
        best_grasp_world, _ = select_top_down_grasp(grasps, scores, extrinsics, vertical_threshold=0.7)

        if best_grasp_world is None:
            # Fallback to centroid grasp if grasp planner fails
            print("Grasp planner failed, falling back to centroid grasp.")
            stats = get_object_stats(mask, depth, intrinsics, extrinsics)
            if stats is None:
                raise RuntimeError("Cannot compute object stats for fallback grasp.")

```

```
            # Create a default top-down orientation
            R_down = np.array([[1.0, 0.0, 0.0], [0.0, -1.0, 0.0], [0.0, 0.0, -1.0]])
            quat = rotation_matrix_to_quaternion(R_down)
            pos = stats["center"].copy()
            pos[2] = stats["max_z"] - 0.02  # Grasp slightly below top
        else:
            pos, quat = decompose_transform(best_grasp_world)

        # Execute sequence
        pre_grasp = pos.copy()
        pre_grasp[2] += 0.12  # 12cm above

        open_gripper()
        move_tcp(pre_grasp, quat)
        move_tcp(pos, quat)
    ...
```

E.6 Low-Level Perception and Control Primitives

This section provides the complete specifications for the low-level perception and control primitives used in CaP-Bench, including function signatures, interface definitions, and documentation strings. These primitives constitute the exact specifications provided to agents in Tier S3. Tier S4 utilizes an identical primitive set, but with in-context usage examples (the Example: sections within docstrings) stripped. Tiers S1 and S2 operate on high-level abstractions that are constructed by composing these fundamental primitives.

```
from typing import Any
import numpy as np
import open3d as o3d
import viser.transforms as vtf
from PIL import Image
from scipy.spatial.transform import Rotation as SciRotation
from capx.envs.base_env import BaseEnv
from capx.integrations import pyroki_snippets as pks  # type: ignore
from capx.integrations.base_api import ApiBase
from capx.integrations.grasp_graspnet import init_contact_graspnet
from capx.integrations.molmo import init_molmo
from capx.integrations.pyroki import init_pyroki
from capx.integrations.sam3 import init_sam3, init_sam3_point_prompt

class S3(ApiBase):
    """
    Robot perception and control primitives for the S3 CaP-Bench Tier
    """
    def __init__(
        self,
        env: BaseEnv,
```

```
        tcp_offset: list[float] | None = [0.0, 0.0, -0.107],
        bimanual: bool = False,
    ) -> None:
        super().__init__(env)
        self._TCP_OFFSET = np.array(tcp_offset, dtype=np.float64)
        self.grasp_net_plan_fn = init_contact_graspnet()
        self.sam3_seg_fn = init_sam3()
        self.sam3_point_prompt_fn = init_sam3_point_prompt()
        self.molmo_point_fn = init_molmo()

        self.ik_solve_fn = init_pyroki()
        self.trajopt_plan_fn = init_pyroki_trajopt()
        self.cfg = None
        self.bimanual = bimanual

    def functions(self) -> dict[str, Any]:
        fns = {
            "get_observation": self.get_observation,
            "segment_sam3_text_prompt": self.segment_sam3_text_prompt,
            "segment_sam3_point_prompt": self.segment_sam3_point_prompt,
            "point_prompt_molmo": self.point_prompt_molmo,
            "plan_grasp": self.plan_grasp,
            "get_oriented_bounding_box_from_3d_points": self.
    get_oriented_bounding_box_from_3d_points,
        }
        if self.bimanual:
            fns["solve_ik_arm0"] = self.solve_ik_arm0
            fns["solve_ik_arm1"] = self.solve_ik_arm1
            fns["move_to_joints_both"] = self.move_to_joints_both
            fns["move_to_joints_arm0"] = self.move_to_joints_arm0
            fns["move_to_joints_arm1"] = self.move_to_joints_arm1
            fns["open_gripper_arm0"] = self.open_gripper_arm0
            fns["close_gripper_arm0"] = self.close_gripper_arm0
            fns["open_gripper_arm1"] = self.open_gripper_arm1
            fns["close_gripper_arm1"] = self.close_gripper_arm1
        else:
            fns["solve_ik"] = self.solve_ik
            fns["move_to_joints"] = self.move_to_joints
            fns["open_gripper"] = self.open_gripper
            fns["close_gripper"] = self.close_gripper

        return fns

    def get_observation(self) -> dict[str, Any]:
        """Get the observation of the environment.
        Returns:
            observation:
                A dictionary containing the observation of the environment
    .
                The dictionary contains the following keys:
```

```
                - ["robot0_robotview"]["images"]["rgb"]: Current color camera image as a numpy array of shape (H, W, 3), dtype uint8.
                - ["robot0_robotview"]["images"]["depth"]: Current depth camera image as a numpy array of shape (H, W, 1), dtype float32.
                - ["robot0_robotview"]["intrinsics"]: Camera intrinsic matrix as a numpy array of shape (3, 3), dtype float64.
                - ["robot0_robotview"]["pose_mat"]: Camera extrinsic matrix as a numpy array of shape (4, 4), dtype float64.

        """
        return self._env.get_observation()

    #
    # Vision models: Sam3 segmentation
    #

    def segment_sam3_point_prompt(
        self,
        rgb: np.ndarray,
        point_coords: tuple[float, float],
    ) -> list[dict[str, Any]]:
        """Run SAM3 segmentation on an RGB image, optionally conditioned on an image coordinate point prompt.

        Args:
            rgb:
                RGB image array of shape (H, W, 3), dtype uint8.
            point_coords:
                (x, y) pixel coordinates of the point prompt.

        Returns:
            masks:
                A list of dictionaries. Each dict may contain:

                  - "mask":  np.ndarray of shape (H, W), dtype bool or uint8,
                             where True/1 means the pixel belongs to the instance.
                  - "score": float confidence score.

        Example:
            >>> rgb = obs["robot0_robotview"]["images"]["rgb"]
            >>> masks = segment_sam3_point_prompt(rgb, (100, 100))
        """
        return self.sam3_point_prompt_fn(Image.fromarray(rgb), point_coords)

    def segment_sam3_text_prompt(
```

```
        self,
        rgb: np.ndarray,
        text_prompt: str,
    ) -> list[dict[str, Any]]:
        """Run SAM3 segmentation on an RGB image conditioned on a text prompt.

        Args:
            rgb:
                RGB image array of shape (H, W, 3), dtype uint8.
            text_prompt:
                Text prompt for SAM3 segmentation.

        Returns:
            masks:
                A list of dictionaries. Each dict may contain:

                  - "mask":  np.ndarray of shape (H, W), dtype bool or uint8,
                             where True/1 means the pixel belongs to the instance.
                  - "box": list [x1, y1, x2, y2] in pixel coordinates.
                  - "score": float confidence score.

        Example:
            >>> rgb = obs["robot0_robotview"]["images"]["rgb"]
            >>> masks = segment_sam3(rgb, text_prompt="red mug")
        """
        return self.sam3_seg_fn(rgb, text_prompt=text_prompt)

    #
    ------------------------------------------------------------------------ #
    # Molmo point prompt
    #
    ------------------------------------------------------------------------ #
    def point_prompt_molmo(
        self,
        image: np.ndarray,
        text_prompt: str,
    ) -> dict[str, tuple[int | None, int | None]]:
        """Use Molmo to point to a coordinate in the image based on a text prompt.

        Args:
            image: np.ndarray: The RGB image to process. Shape: (H, W, 3), dtype uint8.
            text_prompt: str: The text prompt to point to.

        Returns:
```

```
            dict[str, tuple[int | None, int | None]]: Pixel coordinates for each
            object query; (None, None) if parsing failed.
        """
        return self.molmo_point_fn(Image.fromarray(image), objects=[text_prompt])

    def get_oriented_bounding_box_from_3d_points(self, points: np.ndarray) -> dict[str, Any]:
        """Get the oriented bounding box from 3D points.

        Args:
            points: np.ndarray: The 3D points to get the oriented bounding box from.
                Shape: (N, 3), dtype float64.

        Returns:
            dict[str, Any]: The oriented bounding box. The dictionary contains the following keys:
                - "center": np.ndarray: The center of the oriented bounding box in point cloud frame.
                - "extent": np.ndarray: The extent of the oriented bounding box.
                - "R": np.ndarray: The rotation matrix of the oriented bounding box in point cloud frame.
        """
        o3d_points = o3d.geometry.PointCloud()
        o3d_points.points = o3d.utility.Vector3dVector(points)
        o3d_points, ind = o3d_points.remove_statistical_outlier(nb_neighbors=20, std_ratio=2.0)
        obb = o3d_points.get_oriented_bounding_box()
        return {
            "center": obb.center,
            "extent": obb.extent,
            "R": obb.R,
        }

    #------------------------------------------------------------------------#
    # Grasp planner (Contact-GraspNet)
    #------------------------------------------------------------------------#
    def plan_grasp(
        self,
        depth: np.ndarray,
        intrinsics: np.ndarray,
        segmentation: np.ndarray,
    ) -> tuple[np.ndarray, np.ndarray]:
        """Plan grasp candidates using Contact-GraspNet for a single instance.
```

```
190
191         This is a thin wrapper around the Contact-GraspNet planner. It
    does not
192         apply any camera/world transforms or TCP offsets: the caller is
193         responsible for transforming the resulting grasp poses into the
    desired
194         frame and applying TCP offsets if necessary.
195
196         Args:
197             depth:
198                 Depth image in meters.
199                 Shape: (H, W) or (H, W, 1), dtype float32/float64.
200             intrinsics:
201                 Camera intrinsic matrix.
202                 Shape: (3, 3), dtype float64.
203             segmentation:
204                 Instance segmentation map where each integer > 0
    corresponds to a
205                 unique object instance ID.
206                 Shape: (H, W) or (H, W, 1), dtype int32/int64.
207
208         Returns:
209             grasp_poses:
210                 np.ndarray of shape (K, 4, 4), dtype float64.
211                 Homogeneous transforms for each candidate grasp IN THE
    CAMERA FRAME.
212             grasp_scores:
213                 np.ndarray of shape (K,), dtype float64.
214                 Confidence score for each candidate grasp.
215
216         Example:
217             >>> cam = obs["robot0_robotview"]
218             >>> rgb = cam["images"]["rgb"]
219             >>> depth = cam["images"]["depth"][:, :, 0]
220             >>> sam3_results = sam3_seg_fn(rgb, text_prompt="red mug")
221             >>> best = max(sam3_results, key=lambda d: d["score"])
222             >>> mask = best["mask"]
223             >>> K = cam["intrinsics"]
224             >>> grasp_sample_tf, grasp_scores = plan_grasp(
225             ...     depth=depth,
226             ...     intrinsics=K,
227             ...     segmentation=mask,
228             ... )
229             >>> best_idx = grasp_scores.argmax()
230             >>> best_T = grasp_poses[best_idx]  # (4, 4)
231             >>> camera_extrinsics = cam["pose_mat"]
232             >>> grasp_sample_world_frame = camera_extrinsics @ best_T
233         """
234         if depth.ndim == 3 and depth.shape[-1] == 1:
235             depth = depth[:, :, 0]
```

```
        if segmentation.ndim == 3 and segmentation.shape[-1] == 1:
            segmentation = segmentation[:, :, 0]

        grasp_sample, grasp_scores, _ = self.grasp_net_plan_fn(
            depth,
            intrinsics,
            segmentation,
            1,
            z_range=[0.2, 3.5] if self.is_handover else [0.2, 2.0],
            forward_passes=1 if self.is_handover else 3,
        )

        grasp_sample_tf = (
            vtf.SE3.from_matrix(grasp_sample) @ vtf.SE3.from_translation(
    np.array([0, 0, 0.12]))
        ).as_matrix()

        return grasp_sample_tf, grasp_scores

    #
    # IK / motion primitives
    #
    def solve_ik(
        self,
        position: np.ndarray,
        quaternion_wxyz: np.ndarray,
    ) -> np.ndarray:
        """Solve inverse kinematics for the panda_hand link.

        Args:
            position:
                Target position in world frame.
                Shape: (3,), dtype float64.
            quaternion_wxyz:
                Target orientation as a unit quaternion in world frame.
                Shape: (4,), [w, x, y, z], dtype float64.

        Returns:
            joints:
                np.ndarray of shape (7,), dtype float64.
                Joint angles for the 7 DoF Franka arm.

        Example:
            >>> target_pos = np.array([0.5, 0.0, 0.3])
            >>> target_quat = np.array([1.0, 0.0, 0.0, 0.0])  # identity,
    wxyz
            >>> joints = solve_ik(target_pos, target_quat)
            >>> move_to_joints(joints)
```

```
        """
        pos = np.asarray(position, dtype=np.float64).reshape(3)
        quat_wxyz = np.asarray(quaternion_wxyz, dtype=np.float64).reshape
    (4)
        quat_xyzw = np.array(
            [quat_wxyz[1], quat_wxyz[2], quat_wxyz[3], quat_wxyz[0]],
    dtype=np.float64
        )
        rot = SciRotation.from_quat(quat_xyzw)
        offset_pos = pos + rot.apply(self._TCP_OFFSET)

        prev_cfg = self.cfg

        for i in range(15):  # run w/ multiple iterations when using
    vel_cost ik solver
            self.cfg = self.ik_solve_fn(
                target_pose_wxyz_xyz=np.concatenate([quat_wxyz, offset_pos
    ]),
                prev_cfg=prev_cfg,
            )
            if prev_cfg is not None:
                if np.allclose(self.cfg, prev_cfg, atol=1e-3):
                    break
                else:
                    prev_cfg = self.cfg
        joints = np.asarray(self.cfg[:-1], dtype=np.float64).reshape(7)
        return joints

    # Single arm control APIs

    def move_to_joints(self, joints: np.ndarray) -> None:
        """Move the robot to a given joint configuration in a blocking
    manner.

        Args:
            joints:
                Target joint angles for the 7-DoF Franka arm.
                Shape: (7,), dtype float64.

        Returns:
            None

        Example:
            >>> joints = np.array([0.0, -0.5, 0.0, -2.0, 0.0, 1.5, 0.8])
            >>> move_to_joints(joints)
        """
        joints = np.asarray(joints, dtype=np.float64).reshape(7)
        self._env.move_to_joints_blocking(joints)

    def open_gripper(self) -> None:
```

```
        """Open gripper fully.

        Args:
            None
        """
        self._env._set_gripper(1.0)
        for _ in range(30):
            self._env._step_once()

    def close_gripper(self) -> None:
        """Close gripper fully.

        Args:
            None
        """
        self._env._set_gripper(0.0)
        for _ in range(30):
            self._env._step_once()


    # Dual arm control APIs
    def move_to_joints_both(self, joints0: np.ndarray, joints1: np.ndarray) -> None:
        """Move the arms 0 and 1 to a given joint configuration in a blocking manner simultaneously.

        Args:
            joints0:
                Target joint angles for the 7-DoF Franka arm 0.
                Shape: (7,), dtype float64.
            joints1:
                Target joint angles for the 7-DoF Franka arm 1.
                Shape: (7,), dtype float64.
        """
        self._env.move_to_joints_blocking_both(joints0, joints1)

    def move_to_joints_arm0(self, joints: np.ndarray) -> None:
        """Move the robot arm 0 to a given joint configuration in a blocking manner.

        Args:
            joints:
                Target joint angles for the 7-DoF Franka arm 0.
                Shape: (7,), dtype float64.
        """
        joints = np.asarray(joints, dtype=np.float64).reshape(7)
        self._env.move_to_joints_blocking(joints)

    def move_to_joints_arm1(self, joints: np.ndarray) -> None:
```

```
        """Move the robot arm 1 to a given joint configuration in a blocking manner.

        Args:
            joints:
                Target joint angles for the 7-DoF Franka arm 1.
                Shape: (7,), dtype float64.
        """
        joints = np.asarray(joints, dtype=np.float64).reshape(7)
        self._env.move_to_joints_blocking_arm1(joints)

    def open_gripper_arm0(self) -> None:
        """Open gripper fully for Arm 0 (robot0).
        Args:
            None
        Returns:
            None
        """
        self._env._set_gripper(1.0)
        for _ in range(30):
            self._env._step_once()

    def close_gripper_arm0(self) -> None:
        """Close gripper fully for Arm 0 (robot0).
        Args:
            None
        Returns:
            None
        """
        self._env._set_gripper(0.0)
        for _ in range(30):
            self._env._step_once()
    def open_gripper_arm1(self) -> None:
        """Open gripper fully for Arm 1 (robot1).
        Args:
            None
        Returns:
            None
        """
        self._env._set_gripper_arm1(1.0)
        for _ in range(30):
            self._env._step_once()

    def close_gripper_arm1(self) -> None:
        """Close gripper fully for Arm 1 (robot1).
        Args:
            None
        Returns:
            None
        """
```

```
        self._env._set_gripper_arm1(0.0)
        for _ in range(30):
            self._env._step_once()

    def solve_ik_arm0(self, position: np.ndarray, quaternion_wxyz: np.ndarray) -> np.ndarray:
        """Solve inverse kinematics for the panda_hand link for Arm 0 (robot0)."""
        pos = np.asarray(position, dtype=np.float64).reshape(3)
        quat_wxyz = np.asarray(quaternion_wxyz, dtype=np.float64).reshape(4)
        quat_xyzw = np.array(
            [quat_wxyz[1], quat_wxyz[2], quat_wxyz[3], quat_wxyz[0]], dtype=np.float64
        )
        rot = SciRotation.from_quat(quat_xyzw)
        offset_pos = pos + rot.apply(self._TCP_OFFSET)

        prev_cfg = self.cfg
        for i in range(15):
            self.cfg = self.ik_solve_fn(
                target_pose_wxyz_xyz=np.concatenate([quat_wxyz, offset_pos]),
                prev_cfg=prev_cfg,
            )
            if prev_cfg is not None:
                if np.allclose(self.cfg, prev_cfg, atol=1e-3):
                    break
                else:
                    prev_cfg = self.cfg

        joints = np.asarray(self.cfg[:-1], dtype=np.float64).reshape(7)
        return joints

    def solve_ik_arm1(self, position: np.ndarray, quaternion_wxyz: np.ndarray) -> np.ndarray:
        """Solve inverse kinematics for the panda_hand link for Arm 1 (robot1)."""
        if not hasattr(self._env, "move_to_joints_blocking_arm1"):
            raise RuntimeError("Environment does not support Arm 1 control")

        if not hasattr(self._env, "base_link_wxyz_xyz_0") or not hasattr(
            self._env, "base_link_wxyz_xyz_1"
        ):
            raise RuntimeError("Environment does not provide base transforms.")

        pose_arm0_base = vtf.SE3.from_rotation_and_translation(
            rotation=vtf.SO3(wxyz=quaternion_wxyz),
```

```
            translation=position,
        )
        base0_transform = vtf.SE3(wxyz_xyz=self._env.base_link_wxyz_xyz_0)
        pose_world = base0_transform @ pose_arm0_base

        base1_transform = vtf.SE3(wxyz_xyz=self._env.base_link_wxyz_xyz_1)
        base1_transform_inv = base1_transform.inverse()
        pose_arm1_base = base1_transform_inv @ pose_world

        pos = np.asarray(pose_arm1_base.translation(), dtype=np.float64).
    reshape(3)
        quat_wxyz = np.asarray(pose_arm1_base.rotation().wxyz, dtype=np.
    float64).reshape(4)
        quat_xyzw = np.array(
            [quat_wxyz[1], quat_wxyz[2], quat_wxyz[3], quat_wxyz[0]],
    dtype=np.float64
        )
        rot = SciRotation.from_quat(quat_xyzw)
        offset_pos = pos + rot.apply(self._TCP_OFFSET)

        prev_cfg = self.cfg
        for i in range(15):
            self.cfg = self.ik_solve_fn(
                target_pose_wxyz_xyz=np.concatenate([quat_wxyz, offset_pos
    ]),
                prev_cfg=prev_cfg,
            )
            if prev_cfg is not None:
                if np.allclose(self.cfg, prev_cfg, atol=1e-3):
                    break
                else:
                    prev_cfg = self.cfg
        joints = np.asarray(self.cfg[:-1], dtype=np.float64).reshape(7)
        return joints
```

E.7 CaP-Agent0 Details

Synthesized Task-Agnostic Skill Library

In this section, we list all nine agent-synthesized function definitions, interfaces, documentation strings, and low-level implementations added to the CaP-Agent0 skill library.

```
    def rotation_matrix_to_quaternion(R: np.ndarray) -> np.ndarray:
        """
        Convert a 3x3 rotation matrix to a unit quaternion [w, x, y, z].

        Implements the robust Sheppard's method (checking trace and
    diagonal elements)
```

```
        to avoid numerical instability when the trace is close to zero.
        Args:
            R: (3, 3) rotation matrix.
        Returns:
            np.array: [w, x, y, z] unit quaternion.
        """
        tr = np.trace(R)
        if tr > 0:
            S = np.sqrt(tr + 1.0) * 2
            w = 0.25 * S
            x = (R[2, 1] - R[1, 2]) / S
            y = (R[0, 2] - R[2, 0]) / S
            z = (R[1, 0] - R[0, 1]) / S
        elif (R[0, 0] > R[1, 1]) and (R[0, 0] > R[2, 2]):
            S = np.sqrt(1.0 + R[0, 0] - R[1, 1] - R[2, 2]) * 2
            w = (R[2, 1] - R[1, 2]) / S
            x = 0.25 * S
            y = (R[0, 1] + R[1, 0]) / S
            z = (R[0, 2] + R[2, 0]) / S
        elif R[1, 1] > R[2, 2]:
            S = np.sqrt(1.0 + R[1, 1] - R[0, 0] - R[2, 2]) * 2
            w = (R[0, 2] - R[2, 0]) / S
            x = (R[0, 1] + R[1, 0]) / S
            y = 0.25 * S
            z = (R[1, 2] + R[2, 1]) / S
        else:
            S = np.sqrt(1.0 + R[2, 2] - R[0, 0] - R[1, 1]) * 2
            w = (R[1, 0] - R[0, 1]) / S
            x = (R[0, 2] + R[2, 0]) / S
            y = (R[1, 2] + R[2, 1]) / S
            z = 0.25 * S
        return np.array([w, x, y, z])

    def decompose_transform(T: np.ndarray) -> tuple[np.ndarray, np.ndarray
    ]:
        """
        Decompose a 4x4 homogeneous transformation matrix into position
    and quaternion.
        Args:
            T: (4, 4) homogeneous transformation matrix.
        Returns:
            tuple:
                - position: (3,) np.array
                - quaternion: (4,) np.array [w, x, y, z]
        """
        position = T[:3, 3]
        R = T[:3, :3]
        quat = rotation_matrix_to_quaternion(R)
        return position, quat

```

```
    def depth_to_point_cloud(depth_img: np.ndarray, intrinsics: np.ndarray) -> np.ndarray:
        """
        Convert a depth image to a 3D point cloud in the Camera Frame.
        Args:
            depth_img: (H, W) depth map in meters.
            intrinsics: (3, 3) camera intrinsic matrix.
        Returns:
            np.array: (H, W, 3) image of 3D coordinates.
        """
        if depth_img.ndim == 3:
            depth_img = depth_img[:, :, 0]

        h, w = depth_img.shape
        fx = intrinsics[0, 0]
        fy = intrinsics[1, 1]
        cx = intrinsics[0, 2]
        cy = intrinsics[1, 2]

        # Vectorized grid generation
        y_grid, x_grid = np.mgrid[0:h, 0:w]

        z = depth_img
        x = (x_grid - cx) * z / fx
        y = (y_grid - cy) * z / fy

        return np.dstack((x, y, z))

    def mask_to_world_points(
        mask: np.ndarray, depth: np.ndarray, intrinsics: np.ndarray, extrinsics: np.ndarray
    ) -> np.ndarray:
        """
        Convert specific pixels defined by a binary mask into 3D points in the World Frame.
        Args:
            mask: (H, W) binary mask (0 or 1).
            depth: (H, W) depth map.
            intrinsics: (3, 3) camera intrinsics.
            extrinsics: (4, 4) camera-to-world pose matrix.
        Returns:
            np.array: (N, 3) array of valid 3D points in world coordinates.
        """
        # Get pixel coordinates
        ys, xs = np.where(mask > 0)
        if len(ys) == 0:
            return np.empty((0, 3))

        z_vals = depth[ys, xs]
```

```
100
101         # Filter invalid depth
102         valid = z_vals > 0
103         ys = ys[valid]
104         xs = xs[valid]
105         z = z_vals[valid]
106
107         fx = intrinsics[0, 0]
108         fy = intrinsics[1, 1]
109         cx = intrinsics[0, 2]
110         cy = intrinsics[1, 2]
111
112         # Deproject to Camera Frame
113         x_cam = (xs - cx) * z / fx
114         y_cam = (ys - cy) * z / fy
115
116         # Stack to (N, 3)
117         points_cam = np.stack([x_cam, y_cam, z], axis=-1)
118
119         # Transform to World Frame
120         # Create homogeneous coordinates (N, 4)
121         points_cam_hom = np.hstack([points_cam, np.ones((len(points_cam),
    1))])
122         points_world_hom = (extrinsics @ points_cam_hom.T).T
123
124         return points_world_hom[:, :3]
125
126     def pixel_to_world_point(
127         u: int, v: int, z: float, intrinsics: np.ndarray, extrinsics: np.
    ndarray
128     ) -> np.ndarray:
129         """
130         Deproject a single pixel to a 3D world point.
131         Args:
132             u, v: Pixel coordinates (col, row).
133             z: Depth at that pixel.
134             intrinsics: (3, 3) matrix.
135             extrinsics: (4, 4) matrix.
136         Returns:
137             np.array: [x, y, z] in world frame.
138         """
139         fx = intrinsics[0, 0]
140         fy = intrinsics[1, 1]
141         cx = intrinsics[0, 2]
142         cy = intrinsics[1, 2]
143
144         x_cam = (u - cx) * z / fx
145         y_cam = (v - cy) * z / fy
146
147         p_cam = np.array([x_cam, y_cam, z, 1.0])
```

```
        p_world = extrinsics @ p_cam
        return p_world[:3]

    def transform_points(points: np.ndarray, transform_matrix: np.ndarray) -> np.ndarray:
        """
        Apply a 4x4 homogeneous transform to a set of 3D points.
        Args:
            points: (N, 3) or (H, W, 3) array of points.
            transform_matrix: (4, 4) homogeneous transformation matrix.
        Returns:
            np.array: Transformed points with same shape as input.
        """
        original_shape = points.shape
        # Flatten to (N, 3)
        points_reshaped = points.reshape(-1, 3)

        # Convert to homogeneous (N, 4)
        ones = np.ones((points_reshaped.shape[0], 1))
        points_hom = np.hstack((points_reshaped, ones))

        # Apply transform: (4,4) @ (4,N) -> (4,N) -> Transpose back to (N,4)
        points_transformed = (transform_matrix @ points_hom.T).T

        # Return to (N, 3) and original shape
        return points_transformed[:, :3].reshape(original_shape)

    def interpolate_segment(
        p1: np.ndarray, p2: np.ndarray, step: float = 0.03
    ) -> list[np.ndarray]:
        """
        Generate waypoints along a line segment between two 3D points.
        Args:
            p1: Start point (3,).
            p2: End point (3,).
            step: Distance between waypoints in meters.
        Returns:
            list[np.ndarray]: List of points including p1 and p2.
        """
        dist = np.linalg.norm(p2 - p1)
        if dist < 1e-6:
            return [p1]
        num_points = int(np.ceil(dist / step))
        # Using linspace to ensure we hit the start and end exactly
        return [p1 + (p2 - p1) * t for t in np.linspace(0, 1, num_points + 1)]

    def normalize_vector(v: np.ndarray) -> np.ndarray:
        """
```

```
        Normalize a vector to unit length.
        Args:
            v: (3,) vector.
        Returns:
            np.array: (3,) unit vector.
        """
        norm = np.linalg.norm(v)
        if norm < 1e-6:
            return v
        return v / norm

    def select_top_down_grasp(
        grasps: np.ndarray,
        scores: np.ndarray,
        cam_to_world: np.ndarray,
        vertical_threshold: float = 0.8,
    ) -> tuple:
        """
        Selects the best grasp that aligns the gripper vertically (Top-Down).
        Args:
            grasps: (N, 4, 4) Grasp poses in camera frame.
            scores: (N,) Grasp scores.
            cam_to_world: (4, 4) Extrinsics matrix.
            vertical_threshold: Dot product threshold (1.0 is perfectly vertical).
        Returns:
            tuple: (best_grasp_world_matrix, best_score) or (None, -inf)
        """
        best_grasp = None
        best_score = -np.float64("inf")

        # World Z axis (vertical)
        world_z = np.array([0, 0, 1])

        for i, g_camera in enumerate(grasps):
            # Transform grasp to world frame
            g_world = cam_to_world @ g_camera

            # Extract rotation
            R = g_world[:3, :3]

            # Assuming Gripper Z or Y is the approach vector depending on gripper definition.
            # For Franka/Robotiq, the approach vector is usually the Z-axis of the end effector.
            gripper_approach = R[:, 2]

            # Check alignment with negative World Z (pointing down)
            # Dot product should be close to -1 for top-down
```

```
            alignment = -np.dot(gripper_approach, world_z)

            if alignment > vertical_threshold:
                if scores[i] > best_score:
                    best_score = scores[i]
                    best_grasp = g_world

        return best_grasp, best_score
```

Model Ensemble Temperature Details

In both single and multi-model settings, 9 candidates responses are generated. For single model, we query Gemini-3-Pro 9 times with temperatures 0.1, 0.2, 0.3,..., 0.9. For multi-model, we query Gemini-3-Pro, Claude-Opus-4.5, and GPT-5.2 3 times each with temperatures 0.1, 0.5, and 0.9.

Model Ensemble Prompt

This section contains the prompts the coding agent uses to generate the final solution from the candidates for both the initial generation and subsequent multi-turn attempts. Text inside curly braces "{}" represent fstring placeholders.

User prompt for both initial and subsequent generations:

```
Synthesize the best solution.

<original_task_description>
{original prompt for candidates}
</original_task_description>

<candidate_solutions>
{candidate generations}
</candidate_solutions>
```

System prompt for initial generation:

```
You are synthesizing {# of candidate generations} candidate Python solutions
into one optimal program.

SYNTHESIS RULES:
1. Analyze critically and assume no candidate is fully correct
2. Prefer explicit checks over assumptions
3. Combine the best ideas from multiple candidates when appropriate
4. If candidates disagree fundamentally, choose the more robust approach

OUTPUT FORMAT (strict):
You may include reasoning before the fenced code block.
```

```
Output ONLY ONE fenced code block (‘‘‘python...‘‘‘) containing the complete final solution.
Do NOT include any other code blocks or code snippets outside this single block.
```

System prompt for subsequent generations:

```
You are synthesizing {# of generations} candidate responses for a multi-turn robot control task.

DECISION ANALYSIS:
- {regenerate_count} candidates voted REGENERATE
- {finish_count} candidates voted FINISH

SYNTHESIS RULES:
1. Analyze critically and assume no candidate is fully correct
2. Prefer explicit checks over assumptions
3. Combine the best ideas from multiple candidates when appropriate
4. If candidates disagree fundamentally, choose the more robust approach
5. Combine best code ideas from REGENERATE candidates

OUTPUT FORMAT (strict):
- You may include brief reasoning first
- Then output "REGENERATE" on its own line followed by exactly ONE fenced code block, OR output "FINISH" on its own line
```

Multi-turn prompt incentivizing debugging

We noticed that one failure case was inaccurate verification of task completion. Therefore, we experimented with a modified multi-turn prompt which incentivizes verification and debugging, however this did not empirically improve success rate.

	Cube Lift	**Cube Stack**	**Spill Wipe**	**Peg Insert**	**Cube Restack**	**Two Arm Lift**	**Two Arm Handover**	**Avg.**
3M	97	98	100	0	89	74	20	68.29
3M + debug	94	100	98	0	88	66	12	65.43

Table E.1: Task completions between 3M and 3M + debug.

Modified multi-turn prompt:

```
You are acting as experienced debugger and task completion verifier. You will make no assumptions without explicit evidence.
```

```
    Your task is to determine whether the program has actually completed the intended task, and to fix all bugs if any exist.
    You can treat the observed differences between the current and previous state of the environment that are provided to you as a source of evidence, but they are not guaranteed to be accurate.

    The following code blocks have been executed so far:
    ```python
    {executed_code}
    ```
    The current console stdout from the most recent code execution is:
    ```
    {console_stdout}
    ```
    The current console stderr from the most recent code execution is:
    ```
    {console_stderr}
    ```

    Do NOT trust printed outputs or logs blindly.

    Proceed as follows to verify task completion:
    1. Line-by-line inspect the executed code
    2. Cross-check the code's behavior against the console stdout and stderr
    3. Identify explicit, concrete evidence that each required step of the task was completed
    4. Treat missing evidence, implicit assumptions, or partial signals as failure

    Decision Rules:
    - If the task is verifiably complete, respond with the single word 'FINISH'.
    - If the task is not verifiably complete:
      1. Identify specific bugs, failures, missing steps, or incorrect assumptions.
      2. Explain why each issue prevents task completion, citing evidence from stdout, stderr, or code behavior.
      3. If the same approach has been attempted multiple times without success, you MUST try a fundamentally different strategy.
      4. Brainstorm multiple strategies to fix or verify these issues in the next code generation.
      5. Synthesize your brainstorming into a single improved Python program that addresses all identified issues and verifiably completes the task.
      6. Respond with the single word 'REGENERATE' followed immediately by new
```

```
   Python code in a fenced code block (‘‘‘python...‘‘‘)
35
36     If you choose to regenerate code, you must include your reasoning process
   only AS COMMENTS at the start of the code explaining:
37     - What strategies/APIs have already been tried and why they were
   insufficient
38     - How your solution effectively addresses the identified issues
39
40     To reiterate, you must only respond with EXACTLY ONE of the following:
41     - The word ’FINISH’ if you decide to stop generating code
42     - The word ’REGENERATE’ followed immediately by new Python code in a
   SINGLE fenced code block (‘‘‘python...‘‘‘).
```

Model Ensemble Decreases Average Turn Count

We observed that applying a model ensemble for code generations decreases the average turn count. We noticed that M4 tends to retroactively implement bug fixes and API fallbacks, while the model ensemble preemptively anticipates failures beforehand, resulting in more robust code and lower turn count. See Appendix E.5 for a case study on retroactive vs preemptive behavior.

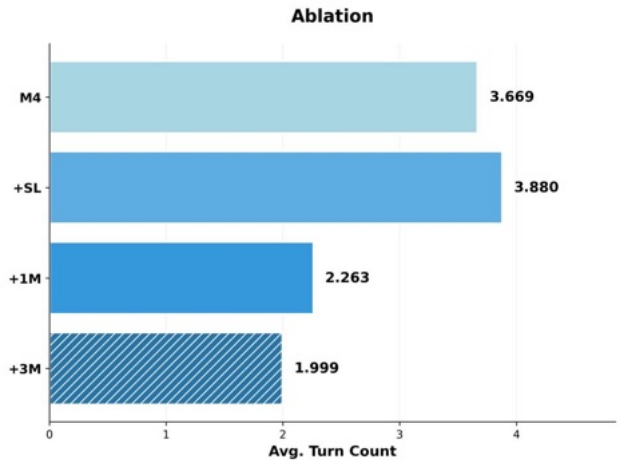


Figure E.15: CaP-Agent0 decreases average turn count compared to M4

E.8 Select Case Studies - Full Generated Code

Stack These as High as You Can

Generated by Gemini-3-Pro as the coding agent and Gemini-3-Pro as the VDM agent.

```
import numpy as np

def get_best_mask(masks):
    if not masks:
        return None
    return max(masks, key=lambda x: x["score"])

def get_object_metrics(mask, depth, intrinsics, extrinsics):
    """Returns center (x,y,z), top_z, bottom_z for a given object mask."""
    points = mask_to_world_points(mask, depth, intrinsics, extrinsics)
    if points.shape[0] == 0:
        return None, None, None

    # Get Oriented Bounding Box
    bbox = get_oriented_bounding_box_from_3d_points(points)
    center = bbox["center"]

    # Calculate robust top and bottom Z from points
    zs = points[:, 2]
    if len(zs) == 0:
        return None, None, None

    top_z = np.percentile(zs, 98) # Use 98th percentile to filter outliers
    bottom_z = np.percentile(zs, 2) # Use 2nd percentile

    return center, top_z, bottom_z

def execute_pose_move(pose_matrix):
    """Moves robot to the specified 4x4 pose matrix."""
    pos, quat = decompose_transform(pose_matrix)
    joints = solve_ik(pos, quat)
    move_to_joints(joints)

def pick_and_place(obj_mask, target_center, target_top_z, depth,
   intrinsics, cam_pose):
    """
    Picks object defined by obj_mask and places it on target defined by
   target_center and target_top_z.
    """
    # 1. Analyze Object to be picked
    obj_center, obj_top, obj_bottom = get_object_metrics(obj_mask, depth,
   intrinsics, cam_pose)
    if obj_center is None:
        print("Could not calculate object metrics.")
        return

    # 2. Plan Grasp
    seg_map = obj_mask.astype(np.int32)
    # plan_grasp returns poses in CAMERA FRAME
    grasp_poses_cam, grasp_scores = plan_grasp(depth, intrinsics, seg_map)
```

```
48
49      if len(grasp_poses_cam) == 0:
50          print("No grasps generated by planner.")
51          return
52
53      # Select best top-down grasp.
54      # select_top_down_grasp takes grasps in CAMERA FRAME and returns the
    best in WORLD FRAME.
55      best_grasp_world, best_score = select_top_down_grasp(grasp_poses_cam,
    grasp_scores, cam_pose, vertical_threshold=0.8)
56
57      if best_grasp_world is None:
58          print("No strictly vertical grasp found. Trying relaxed threshold
    .")
59          best_grasp_world, best_score = select_top_down_grasp(
    grasp_poses_cam, grasp_scores, cam_pose, vertical_threshold=0.5)
60
61      if best_grasp_world is None:
62          print("No valid grasp found even with relaxed threshold.")
63          return
64
65      print(f"Executing grasp with score {best_score}")
66
67      # Calculate offset from TCP to bottom of object
68      # TCP Z at grasp - Object Bottom Z
69      grasp_z = best_grasp_world[2, 3]
70      tcp_to_bottom_offset = grasp_z - obj_bottom
71
72      # 3. Execute Pick
73      # Move to Pre-Grasp (Safety Hover)
74      pre_grasp = best_grasp_world.copy()
75      pre_grasp[2, 3] += 0.10
76      execute_pose_move(pre_grasp)
77
78      # Move to Grasp
79      execute_pose_move(best_grasp_world)
80      close_gripper()
81
82      # Lift Up
83      lift_pose = best_grasp_world.copy()
84      lift_pose[2, 3] += 0.20
85      execute_pose_move(lift_pose)
86
87      # 4. Execute Place
88      # Calculate Place Pose
89      place_pose = best_grasp_world.copy()
90      place_pose[0, 3] = target_center[0]
91      place_pose[1, 3] = target_center[1]
92      # Target Z for TCP = Target Surface Z + Offset + Padding
93      # Adding a small padding (1cm) to ensure we don't collide hard,
```

```
    # but close enough to drop safely.
    place_pose[2, 3] = target_top_z + tcp_to_bottom_offset + 0.015

    # Move to Pre-Place (Safety Hover)
    pre_place = place_pose.copy()
    pre_place[2, 3] += 0.15
    execute_pose_move(pre_place)

    # Move to Place
    execute_pose_move(place_pose)
    open_gripper()

    # Move to Post-Place (Lift Up)
    execute_pose_move(pre_place)


# --- Main Execution ---

# 1. Reset
open_gripper()

# 2. Perception & Strategy: Stack Blue Cube on Yellow Cube
obs = get_observation()
rgb = obs["robot0_robotview"]["images"]["rgb"]
depth = obs["robot0_robotview"]["images"]["depth"]
intrinsics = obs["robot0_robotview"]["intrinsics"]
cam_pose = obs["robot0_robotview"]["pose_mat"]

# Find Yellow Cube (Base)
masks_yellow = segment_sam3_text_prompt(rgb, "yellow cube")
yellow_data = get_best_mask(masks_yellow)

# Find Blue Cube (Object 1)
masks_blue = segment_sam3_text_prompt(rgb, "blue cube")
blue_data = get_best_mask(masks_blue)

if yellow_data is not None and blue_data is not None:
    # Get base metrics
    yellow_center, yellow_top, _ = get_object_metrics(yellow_data["mask"],
    depth, intrinsics, cam_pose)

    if yellow_center is not None:
        print("Stacking Blue Cube on Yellow Cube...")
        pick_and_place(blue_data["mask"], yellow_center, yellow_top, depth
    , intrinsics, cam_pose)
    else:
        print("Could not compute metrics for Yellow Cube")
else:
    print("Could not find Blue or Yellow cube")

```

```

# 3. Perception & Strategy: Stack Ball on Blue Cube
# Re-observe because the blue cube has moved
obs = get_observation()
rgb = obs["robot0_robotview"]["images"]["rgb"]
depth = obs["robot0_robotview"]["images"]["depth"]
intrinsics = obs["robot0_robotview"]["intrinsics"]
cam_pose = obs["robot0_robotview"]["pose_mat"]

# Find Blue Cube again (New Base)
masks_blue_new = segment_sam3_text_prompt(rgb, "blue cube")
blue_data_new = get_best_mask(masks_blue_new)

# Find Tennis Ball (Object 2)
masks_ball = segment_sam3_text_prompt(rgb, "yellow tennis ball")
ball_data = get_best_mask(masks_ball)

if blue_data_new is not None and ball_data is not None:
    # Get base metrics (blue cube is now the base)
    blue_center_new, blue_top_new, _ = get_object_metrics(blue_data_new["mask"], depth, intrinsics, cam_pose)

    if blue_center_new is not None:
        print("Stacking Tennis Ball on Blue Cube...")
        pick_and_place(ball_data["mask"], blue_center_new, blue_top_new, depth, intrinsics, cam_pose)
    else:
        print("Could not compute metrics for Blue Cube (new base)")
else:
    print("Could not find Blue Cube or Tennis Ball for second stack")
```

Place the Blue Cube on Top of the Yellow Cube

Generated by Gemini-3-Pro as the coding agent and Gemini-3-Pro as the VDM agent.

```
import numpy as np

# --- Helper functions ---

def get_object_centroid(obs, text_prompt):
    """
    Finds the 3D centroid of an object specified by a text prompt.
    Returns the centroid (x, y, z) in world frame and the mask.
    """
    rgb = obs["robot0_robotview"]["images"]["rgb"]
    depth = obs["robot0_robotview"]["images"]["depth"]
    intrinsics = obs["robot0_robotview"]["intrinsics"]
    extrinsics = obs["robot0_robotview"]["pose_mat"]

    # Segment object
```

```
    results = segment_sam3_text_prompt(rgb, text_prompt)
    if not results:
        print(f"Object '{text_prompt}' not found.")
        return None, None

    best_result = max(results, key=lambda x: x["score"])
    mask = best_result["mask"]

    # Get 3D points
    points = mask_to_world_points(mask, depth, intrinsics, extrinsics)
    if points.shape[0] == 0:
        print(f"No valid depth points for '{text_prompt}'.")
        return None, None

    centroid = np.mean(points, axis=0)
    return centroid, mask

def pick_object(obs, text_prompt, z_offset=0.0):
    """
    Standard pick routine:
    1. Segment object
    2. Plan grasp
    3. Execute pick trajectory (pre-grasp -> grasp -> close -> lift)
    """
    rgb = obs["robot0_robotview"]["images"]["rgb"]
    depth = obs["robot0_robotview"]["images"]["depth"]
    intrinsics = obs["robot0_robotview"]["intrinsics"]
    extrinsics = obs["robot0_robotview"]["pose_mat"]

    # 1. Segment
    results = segment_sam3_text_prompt(rgb, text_prompt)
    if not results:
        print(f"Could not find {text_prompt} to pick.")
        return False

    # Get mask with highest score
    best_res = max(results, key=lambda x: x["score"])
    mask = best_res["mask"]

    # 2. Plan Grasp
    # Note: depth needs to be (H, W) or (H, W, 1) - API doc says (H, W)
   usually preferred for planner wrappers,
    # but the function signature accepts (H, W, 1).
    if depth.ndim == 3 and depth.shape[2] == 1:
        depth_map = depth[:, :, 0]
    else:
        depth_map = depth

    grasp_poses, grasp_scores = plan_grasp(depth_map, intrinsics, mask)

```

```
    if len(grasp_scores) == 0:
        print(f"No grasps found for {text_prompt}.")
        return False

    # Select best top-down grasp
    best_pose, best_score = select_top_down_grasp(grasp_poses,
    grasp_scores, extrinsics)

    if best_pose is None:
        print("No valid top-down grasp found, falling back to highest
    score.")
        best_idx = np.argmax(grasp_scores)
        best_pose_cam = grasp_poses[best_idx]
        best_pose = extrinsics @ best_pose_cam # Convert to world frame

    # Decompose grasp pose
    grasp_pos, grasp_quat = decompose_transform(best_pose)

    # 3. Execute Pick
    # Pre-grasp (hover 10cm above)
    pre_grasp_pos = grasp_pos + np.array([0, 0, 0.1])

    # Move to pre-grasp
    joints = solve_ik(pre_grasp_pos, grasp_quat)
    move_to_joints(joints)

    # Open gripper
    open_gripper()

    # Move to grasp
    joints = solve_ik(grasp_pos, grasp_quat)
    move_to_joints(joints)

    # Close gripper
    close_gripper()

    # Lift up (20cm)
    lift_pos = grasp_pos + np.array([0, 0, 0.2])
    joints = solve_ik(lift_pos, grasp_quat)
    move_to_joints(joints)

    return True

def place_at_position(position, height_offset=0.05):
    """
    Place currently held object at a specific world position.
    Target orientation is usually top-down (gripper pointing down).
    """
    # Standard top-down orientation (gripper pointing -z)
    # A common quaternion for top-down is pointing down z-axis.
```

```
    # Let's assume the current grasp orientation is maintained or we define a fixed top-down.
    # For simplicity, we often maintain the orientation we picked with, or reset to a known neutral top-down.
    # Here, we will define a fixed top-down orientation [0, 1, 0, 0] (x-axis rotation 180 deg) or similar.
    # However, to be safe, let's just use the current robot configuration's orientation or a hardcoded one.
    # Often [0, 1, 0, 0] is top down for Panda.
    place_quat = np.array([0.0, 1.0, 0.0, 0.0])

    target_pos = position + np.array([0, 0, height_offset])

    # Move over target (high)
    hover_pos = target_pos + np.array([0, 0, 0.15])
    joints = solve_ik(hover_pos, place_quat)
    move_to_joints(joints)

    # Move down to place
    joints = solve_ik(target_pos, place_quat)
    move_to_joints(joints)

    # Open gripper
    open_gripper()

    # Move back up
    joints = solve_ik(hover_pos, place_quat)
    move_to_joints(joints)

def main():
    obs = get_observation()

    # --- Strategy ---
    # The stack is Green (top) -> Yellow (mid) -> Blue (bottom).
    # Goal: Blue on Yellow.
    # 1. Move Green to the side.
    # 2. Move Yellow to another spot (target base).
    # 3. Pick Blue.
    # 4. Place Blue on top of Yellow.

    # Defined drop zones relative to the table center or just hardcoded offsets based on workspace knowledge.
    # Let's dynamically find a spot by looking at the initial stack position.

    # Find the stack location (using the green cube at the top)
    green_center, _ = get_object_centroid(obs, "green cube")
    if green_center is None: return

    # Define temporary drop zones relative to the stack
```

```
    # Drop zone 1 (for green): 20cm to the left
    drop_zone_green = green_center + np.array([-0.2, 0.0, -green_center[2] + 0.02]) # Z is roughly table height
    # Drop zone 2 (for yellow): 20cm to the right (this will be the final destination for yellow)
    drop_zone_yellow = green_center + np.array([0.2, 0.0, -green_center[2] + 0.02])

    # --- Step 1: Remove Green Cube ---
    print("Moving Green Cube...")
    success = pick_object(obs, "green cube")
    if success:
        place_at_position(drop_zone_green, height_offset=0.05) # Place on table
    else:
        print("Failed to move green cube.")
        return

    # Refresh observation
    obs = get_observation()

    # --- Step 2: Move Yellow Cube ---
    # Now Yellow should be exposed.
    print("Moving Yellow Cube...")
    success = pick_object(obs, "yellow cube")
    if success:
        # We place it at drop_zone_yellow. This is where we want to stack the blue one later.
        place_at_position(drop_zone_yellow, height_offset=0.05)
    else:
        print("Failed to move yellow cube.")
        return

    # Refresh observation to find the new position of the yellow cube and the exposed blue cube
    obs = get_observation()

    # Get precise location of yellow cube now that it's moved
    yellow_center, _ = get_object_centroid(obs, "yellow cube")
    if yellow_center is None:
        print("Lost track of yellow cube.")
        return

    # --- Step 3: Pick Blue Cube ---
    print("Picking Blue Cube...")
    success = pick_object(obs, "blue cube")
    if not success:
        print("Failed to pick blue cube.")
        return

```

```
    # --- Step 4: Place Blue on Yellow ---
    print("Placing Blue on Yellow...")
    # Target is yellow center, but offset Z by cube height (approx 5cm usually for these cubes)
    # We add a small buffer.
    cube_height_approx = 0.05
    place_at_position(yellow_center, height_offset=cube_height_approx + 0.02)

    # Done, gripper is already opened in place_at_position
    print("Task completed.")

main()
```

Additional Analysis Plots

We perform additional quantitative analysis on the cube stack task to understand the impact of multi-turn on success rate. We focus our analysis on Gemini 3 Pro [358], GPT 5.2 [361], and Claude Opus 4.5 [363], the three strongest performers on CaP-Bench. In particular, we plot the probability density function (PDF) of success over turns (Figure E.16), PDF of success over tokens (Figure E.17), multi-turn success rate vs turn for the cube stacking task in Figure E.18. We find that for the cube stacking task, these models would perform the task on the first try, and use the subsequent turns for recovery. Successes for GPT 5.2 and Claude Opus 4.5 seem to have shorter code length than that of Gemini 3 Pro.

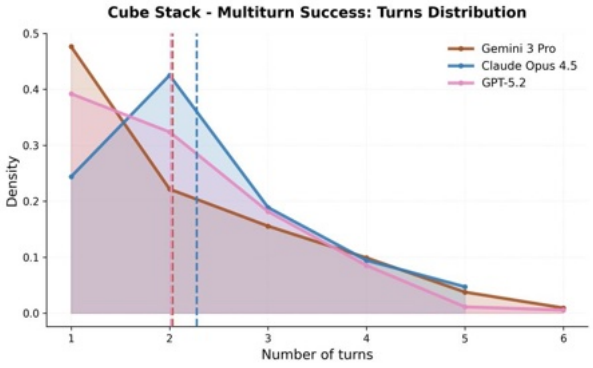


Figure E.16: Distribution of multi-turn cube stack successes over turns. Results are averaged across M1, M2, and M3.

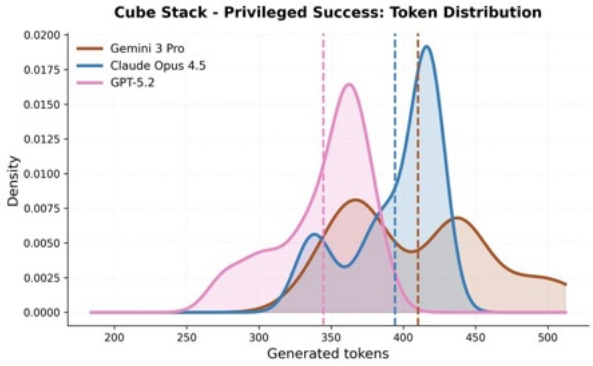


Figure E.17: Distribution of cube stack successes (S1) over tokens

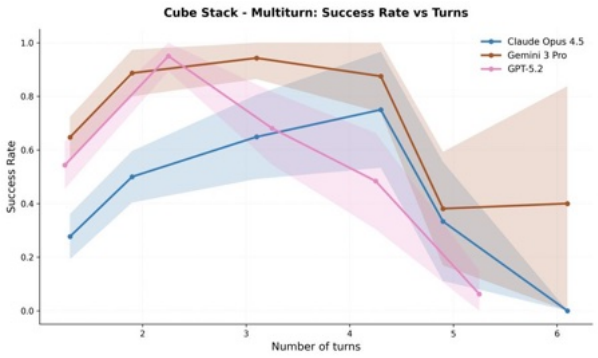


Figure E.18: Cube stack multi-turn success rate vs. turns. Trials with too few or too many turns have lower success rate. Results are averaged across M1, M2, and M3.

E.9 LIBERO-PRO Evaluation Results

We present detailed task-wise performance of OpenVLA, π_0, $\pi_{0.5}$, and CaP-Agent0 on LIBERO-PRO, which evaluates model generalization under initial position perturbations (Pos) and instruction perturbations (Task). Results are summarized in Table E.2, Table E.3 and Table E.4. Each task was executed over 50 trials. In these tasks, CaP-Agent0 struggles with failures in perception, grasp generation, control APIs. For example, queries of "alphabet soup can" to SAM 3 often results in segmentations of the "tomato sauce can" also present in the scene. Also due to the occluded nature of these scenes and the camera not being top down, many grasps generated on desired objects will be at an angle, this, when combined with the lack of collision-aware motion planning often results in other objects in the scene being knocked over during execution, preventing a secure grasp on the desired object.

Table E.2: Task-wise LIBERO-PRO performance of OpenVLA, π_0, $\pi_{0.5}$ and CaP-Agent0 on the **libero-object** benchmark. **Action notation:** $Place(obj, loc)$ = pick up object *obj* and place *obj* into/onto target *loc*.

Task (Symbolic Form)	OpenVLA		π_0		$\pi_{0.5}$		CaP-Agent0	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task
Place(alphabet_soup, basket)	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.04
Place(bbq_sauce, basket)	0.00	0.00	0.00	0.00	1.00	0.02	0.12	0.42
Place(butter, basket)	0.00	0.00	0.00	0.00	0.54	0.00	0.26	0.18
Place(chocolate_pudding, basket)	0.00	0.00	0.00	0.00	0.00	0.02	0.18	0.48
Place(cream_cheese, basket)	0.00	0.00	0.10	0.00	0.00	0.00	0.12	0.06
Place(ketchup, basket)	0.00	0.00	0.00	0.00	0.20	0.02	0.32	0.12
Place(milk, basket)	0.00	0.00	0.00	0.00	0.00	0.00	0.38	0.02
Place(orange_juice, basket)	0.00	0.00	0.00	0.00	0.00	0.02	0.30	0.02
Place(salad_dressing, basket)	0.00	0.00	0.10	0.00	0.00	0.00	0.32	0.00
Place(tomato_sauce, basket)	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.48
Average	0.00	0.00	0.00	0.00	0.17	0.01	**0.218**	**0.182**

Table E.3: Task-wise LIBERO-PRO performance of OpenVLA, π_0, $\pi_{0.5}$ and CaP-Agent0 on the **libero-goal** benchmark. **Action notation:** $Open(x, y)$ = open target y of container x; $Put(obj, loc)$ = place object *obj* onto/into location *loc*; $Push(obj, loc)$ = push object *obj* toward location *loc*; $TurnOn(obj)$ = activate object *obj*.

Task (Symbolic Form)	OpenVLA		π_0		$\pi_{0.5}$		CaP-Agent0	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task
$Open(cabinet, drawer_{mid})$	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00
$Put(bowl, drawer_{top})$	0.00	0.00	0.00	0.00	0.94	0.02	0.04	0.00
$Push(plate, stove_{front})$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10
$Put(bowl, plate)$	0.00	0.00	0.00	0.00	0.00	0.02	0.36	0.38
$Put(bowl, stove)$	0.00	0.00	0.00	0.00	0.00	0.04	0.22	0.12
$Put(bowl, cabinet_{top})$	0.00	0.00	0.00	0.00	0.00	0.02	0.60	0.04
$Put(cream\_cheese, bowl)$	0.00	0.00	0.00	0.00	0.98	0.02	0.04	0.34
$Put(wine\_bottle, rack)$	0.00	0.00	0.00	0.00	0.88	0.02	0.02	0.12
$Put(wine\_bottle, cabinet_{top})$	0.00	0.00	0.00	0.00	0.98	0.02	0.62	0.40
$TurnOn(stove)$	0.00	0.00	0.00	0.00	0.00	0.00	0.66	0.18
Average	0.00	0.00	0.00	0.00	**0.38**	0.00	0.256	**0.168**

Table E.4: Task-wise LIBERO-PRO performance of OpenVLA, π_0, $\pi_{0.5}$ and CaP-Agent0 on the **libero-spatial** benchmark. **Action notation:** $Pick(src, dst)$ = pick up object $bowl_{black}$ from location src and place $bowl_{black}$ onto/into target dst.

Task (Symbolic Form)	OpenVLA		π_0		$\pi_{0.5}$		CaP-Agent0	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task
$Pick(between(plate, ramekin), plate)$	0.00	0.00	0.00	0.00	0.02	0.00	0.22	0.14
$Pick(table\_center, plate)$	0.00	0.00	0.00	0.00	0.00	0.02	0.22	0.14
$Pick(drawer_{top}(cabinet_{wood}), plate)$	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.10
$Pick(next\_to(cookie\_box), plate)$	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.10
$Pick(next\_to(plate), plate)$	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.20
$Pick(next\_to(ramekin), plate)$	0.00	0.00	0.00	0.00	0.12	0.02	0.30	0.14
$Pick(on(cookie\_box), plate)$	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.08
$Pick(on(ramekin), plate)$	0.00	0.00	0.00	0.00	0.98	0.02	0.02	0.20
$Pick(on(stove), plate)$	0.00	0.00	0.00	0.00	0.02	0.00	0.08	0.14
$Pick(on(cabinet_{wood}), plate)$	0.00	0.00	0.00	0.00	0.90	0.00	0.08	0.16
Average	0.00	0.00	0.00	0.00	**0.20**	0.01	0.118	**0.14**

Bibliography

[1] Brent Yi et al. *Viser: Imperative, Web-based 3D Visualization in Python*. 2025. arXiv: 2507.22885 [cs.CV]. URL: https://arxiv.org/abs/2507.22885.

[2] Ashish Vaswani et al. "Attention is all you need". In: *NeurIPS*. 2017.

[3] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *NAACL-HCT*. 2019.

[4] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[5] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv:2001.08361* (2020).

[6] Jordan Hoffmann et al. "Training compute-optimal large language models". In: *arXiv:2203.15556* (2022).

[7] Jack W Rae et al. "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". In: *arXiv:2112.11446* (2021).

[8] Hugo Touvron et al. "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288* (2023).

[9] Hugo Touvron et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models". In: *arXiv preprint arXiv:2307.09288* (2023). URL: https://doi.org/10.48550/arXiv.2307.09288.

[10] Abhimanyu Dubey et al. "The llama 3 herd of models". In: *arXiv preprint arXiv:2407.21783* (2024).

[11] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[12] Gemini Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.

[13] Dan Hendrycks et al. "Measuring massive multitask language understanding". In: *arXiv preprint arXiv:2009.03300* (2020).

[14] Dan Hendrycks et al. "Measuring Mathematical Problem Solving with the MATH Dataset". In: *NeurIPS Datasets and Benchmarks Track*. 2021.

[15] David Rein et al. "Gpqa: A graduate-level google-proof q&a benchmark". In: *First Conference on Language Modeling*. 2024.

[16] Mirac Suzgun et al. "Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them". In: *Findings of the Association for Computational Linguistics: ACL*. 2023.

[17] Wanjun Zhong et al. "AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models". In: *Findings of NAACL*. 2024.

[18] Long Phan et al. *Humanity's Last Exam*. 2025.

[19] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *arXiv preprint arXiv:2107.03374* (2021).

[20] Karl Cobbe et al. "Training verifiers to solve math word problems". In: *arXiv:2110.14168* (2021).

[21] Hunter Lightman et al. "Let's Verify Step by Step". In: *International Conference on Learning Representations*. 2024.

[22] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* (2015).

[23] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[24] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[25] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[26] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision*. 2015.

[27] Karan Singhal et al. "Large language models encode clinical knowledge". In: *Nature* 620.7972 (2023), pp. 172–180.

[28] Daniel Martin Katz et al. "GPT-4 passes the bar exam". In: *Philosophical Transactions of the Royal Society A* 382.2270 (2024).

[29] Jason Wei et al. "Emergent abilities of large language models". In: *Transactions on Machine Learning Research* (2022).

[30] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[31] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *ICML*. 2020.

[32] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *CVPR*. 2020.

[33] Mark Chen et al. "Generative pretraining from pixels". In: *ICML*. 2020.

[34] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV].

[35] Kaiming He et al. "Masked autoencoders are scalable vision learners". In: *arXiv:2111.06377* (2021).

[36] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *ICML*. 2021.

[37] Xiaohua Zhai et al. "Sigmoid loss for language image pre-training". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 11975–11986.

[38] Junnan Li et al. "BLIP: Bootstrapping Language-Image Pre-Training for Unified Vision-Language Understanding and Generation". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 12888–12900.

[39] Junnan Li et al. "BLIP-2: Bootstrapping Language-Image Pre-Training with Frozen Image Encoders and Large Language Models". In: *arXiv preprint arXiv:2301.12597* (2023).

[40] Alexander Kirillov et al. "Segment Anything". In: *arXiv:2304.02643* (2023).

[41] Suraj Nair et al. "R3m: A universal visual representation for robot manipulation". In: *arXiv:2203.12601* (2022).

[42] Tete Xiao et al. "Masked visual pre-training for motor control". In: *arXiv:2203.06173* (2022).

[43] Arjun Majumdar et al. "Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?" In: *arXiv preprint arXiv:2303.18240* (2023). arXiv: 2303.18240 [cs.CV].

[44] Eric Jang et al. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *Conference on Robot Learning* (2021), p. 12. DOI: 164:991-1002.

[45] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv:2212.06817* (2022).

[46] Anthony Brohan et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *arXiv preprint arXiv:2307.15818* (2023).

[47] Danny Driess et al. "PaLM-E: An Embodied Multimodal Language Model". In: *arXiv preprint arXiv:2303.03378*. 2023.

[48] Embodiment Collaboration. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. 2024. arXiv: 2310.08864 [cs.RO]. URL: https://arxiv.org/abs/2310.08864.

[49] Embodiment Collaboration et al. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. 2024. arXiv: 2310.08864 [cs.RO].

[50] Octo Model Team et al. "Octo: An Open-Source Generalist Robot Policy". In: *Proceedings of Robotics: Science and Systems*. Delft, Netherlands, 2024.

[51] Moo Jin Kim et al. "OpenVLA: An Open-Source Vision-Language-Action Model". In: *Proceedings of The 8th Conference on Robot Learning*. Ed. by Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard. Vol. 270. Proceedings of Machine Learning Research. PMLR, June 2025, pp. 2679–2713. URL: https://proceedings.mlr.press/v270/kim25c.html.

[52] Kevin Black et al. "π0: A vision-language-action flow model for general robot control, 2024". In: *URL https://arxiv. org/abs/2410.24164* ().

[53] Physical Intelligence et al. "$\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization". In: *arXiv preprint arXiv:2504.16054* (2025).

[54] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion". In: *The International Journal of Robotics Research* (2023), p. 02783649241273668.

[55] Tony Z Zhao et al. "Learning fine-grained bimanual manipulation with low-cost hardware". In: *arXiv preprint arXiv:2304.13705* (2023).

[56] Ilija Radosavovic et al. "Robot Learning with Sensorimotor Pre-training". In: *arXiv preprint arXiv:2306.10007* (2023).

[57] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[58] Paul F Christiano et al. "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*. 2017.

[59] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv:1707.06347* (2017).

[60] Zhihong Shao et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models". In: *arXiv preprint arXiv:2402.03300* (2024).

[61] OpenAI. *OpenAI o1 System Card*. https://openai.com/index/openai-o1-system-card/. System card describing the OpenAI o1 large language model and its capabilities. Dec. 2024.

[62] Daya Guo et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning". In: *arXiv preprint arXiv:2501.12948* (2025).

[63] DeepSeek-AI. *DeepSeek-V3 Technical Report*. 2024. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

[64] Alexander Khazatsky et al. *DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset*. 2024. arXiv: 2403.12945 [cs.RO].

[65] Zijian Zhang et al. "Grape: Generalizing robot policy via preference alignment". In: *arXiv preprint arXiv:2411.19309* (2024).

[66] Yuxin Chen et al. "Fdpp: Fine-tune diffusion policy with human preference". In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2025, pp. 12010–12016.

[67] Viktor Makoviychuk et al. "Isaac Gym: High performance GPU-based physics simulation for robot learning". In: *NeurIPS*. 2021.

[68] Ashish Kumar et al. "Rma: Rapid motor adaptation for legged robots". In: *RSS* (2021).

[69] Nikita Rudin et al. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *CoRL*. 2021.

[70] Ilija Radosavovic et al. *Humanoid Locomotion as Next Token Prediction*. 2024. arXiv: 2402.19469 [cs.RO].

[71] Tairan He et al. "VIRAL: Visual Sim-to-Real at Scale for Humanoid Loco-Manipulation". In: *arXiv preprint arXiv:2511.15200* (2025).

[72] Ritvik Singh et al. "Dextrah-rgb: Visuomotor policies to grasp anything with dexterous hands". In: *arXiv preprint arXiv:2412.01791* (2024).

[73] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1-2 (2002), pp. 57–83.

[74] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[75] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[76] Xuezhi Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2203.11171.

[77] Shunyu Yao et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023. URL: https://arxiv.org/abs/2305.10601.

[78] Bradley Brown et al. "Large Language Monkeys: Scaling Inference Compute with Repeated Sampling". In: *arXiv:2407.21787* (2024).

[79] Charlie Snell et al. "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Model Parameters". In: *arXiv preprint arXiv:2408.03314* (2024). URL: https://arxiv.org/abs/2408.03314.

[80] Shunyu Yao et al. "React: Synergizing reasoning and acting in language models". In: *The eleventh international conference on learning representations*. 2022.

[81] Noah Shinn et al. "Reflexion: Language Agents with Verbal Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023. URL: https://arxiv.org/abs/2303.11366.

[82] Timo Schick et al. "Toolformer: Language models can teach themselves to use tools". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 68539–68551.

[83] Guanzhi Wang et al. "Voyager: An Open-Ended Embodied Agent with Large Language Models". In: *arXiv preprint arXiv:2305.16291* (2023).

[84] John Yang et al. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://arxiv.org/abs/2405.15793.

[85] Xingyao Wang et al. "OpenHands: An Open Platform for AI Software Developers as Generalist Agents". In: *International Conference on Learning Representations (ICLR)*. 2025. URL: https://arxiv.org/abs/2407.16741.

[86] Xingyao Wang et al. "Executable Code Actions Elicit Better LLM Agents". In: *International Conference on Machine Learning (ICML)*. 2024. URL: https://arxiv.org/abs/2402.01030.

[87] Carlos E Jimenez et al. "SWE-bench: Can Language Models Resolve Real-world Github Issues?" In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=VTF8yNQM66.

[88] Jiayi Pan et al. "Training software engineering agents and verifiers with swe-gym". In: *arXiv preprint arXiv:2412.21139* (2024).

[89] Jacky Liang et al. "Code as policies: Language model programs for embodied control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 9493–9500.

[90] Ishika Singh et al. "ProgPrompt: Generating Situated Robot Task Plans Using Large Language Models". In: *IEEE International Conference on Robotics and Automation*. IEEE. 2023, pp. 11523–11530.

[91] Michael Ahn et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Conference on Robot Learning (CoRL)*. 2022. URL: https://arxiv.org/abs/2204.01691.

[92] Wenlong Huang et al. "Inner monologue: Embodied reasoning through planning with language models". In: *arXiv preprint arXiv:2207.05608* (2022).

[93] Wenlong Huang et al. "Voxposer: Composable 3D Value Maps for Robotic Manipulation with Language Models". In: *arXiv preprint arXiv:2307.05973* (2023).

[94] Junyao Shi et al. "Maestro: Orchestrating Robotics Modules with Vision-Language Models for Zero-Shot Generalist Robots". In: *NeurIPS 2025 Workshop on Space in Vision, Language, and Embodied AI*. 2025.

[95] Gemini Robotics Team et al. "Gemini Robotics: Bringing AI into the Physical World". In: *arXiv preprint arXiv:2503.20020* (2025).

[96] Qingwen Bu et al. "Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems". In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2025.

[97] Ken Goldberg. *Igniting the Real Robot Revolution Requires Closing the "Data Gap"*. Invited Talk at NVIDIA GPU Technology Conference (GTC). Talk ID: S74739. NVIDIA, Mar. 2025. URL: https://www.nvidia.com/gtc/.

[98] Yizhong Wang et al. *Self-Instruct: Aligning Language Model with Self Generated Instructions*. 2022.

[99] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[100] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[101] Boyuan Chen et al. "SpatialVLM: Endowing Vision-Language Models with Spatial Reasoning Capabilities". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 14455–14465.

[102] Zachary Teed and Jia Deng. "Raft: Recurrent all-pairs field transforms for optical flow". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 402–419.

[103] Shuzhe Wang et al. "DUSt3R: Geometric 3D Vision Made Easy". In: *CVPR*. 2024.

[104] Junyi Zhang et al. "Monst3r: A simple approach for estimating geometry in the presence of motion". In: *arXiv preprint arXiv:2410.03825* (2024).

[105] Ruoshi Liu et al. *Zero-1-to-3: Zero-shot One Image to 3D Object*. arXiv:2303.11328 [cs]. Mar. 2023. URL: http://arxiv.org/abs/2303.11328 (visited on 03/23/2023).

[106] Vitor Guizilini et al. *Zero-Shot Novel View and Depth Synthesis with Multi-View Geometric Diffusion*. 2025. arXiv: 2501.18804 [cs.CV]. URL: https://arxiv.org/abs/2501.18804.

[107] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[108] Ilija Radosavovic et al. "Real-world robot learning with masked visual pre-training". In: *arXiv:2210.03109* (2022).

[109] Scott Reed et al. "A generalist agent". In: *arXiv:2205.06175* (2022).

[110] Yunfan Jiang et al. "VIMA: General robot manipulation with multimodal prompts". In: *International Conference on Machine Learning (ICML)* (2023).

[111] Peter R. Florence et al. "Implicit Behavioral Cloning". In: *CoRL*. 2021.

[112] Zhan Tong et al. "VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training". In: *arXiv:2203.12602* (2022).

[113] Letian Fu et al. "In-Context Imitation Learning via Next-Token Prediction". In: *arXiv preprint arXiv:2408.15980* (2024).

[114] Yan Duan et al. "One-shot imitation learning". In: *Advances in neural information processing systems* 30 (2017).

[115] Eugene Valassakis et al. "Demonstrate once, imitate immediately (dome): Learning visual servoing for one-shot imitation learning". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 8614–8621.

[116] Eric Jang et al. "Bc-z: Zero-shot task generalization with robotic imitation learning". In: *Conference on Robot Learning*. 2022.

[117] Mengdi Xu et al. "Prompting decision transformer for few-shot policy generalization". In: *international conference on machine learning*. PMLR. 2022, pp. 24631–24645.

[118] Norman Di Palo and Edward Johns. "Keypoint Action Tokens Enable In-Context Imitation Learning in Robotics". In: *arXiv preprint arXiv:2403.19578* (2024).

[119] Vidhi Jain et al. "Vid2Robot: End-to-end Video-conditioned Policy Learning with Cross-Attention Transformers". In: *arXiv preprint arXiv:2403.12943* (2024).

[120] Justin Kerr et al. "LERF: Language Embedded Radiance Fields". In: *International Conference on Computer Vision (ICCV)*. 2023.

[121] Mengcheng Lan et al. "ClearCLIP: Decomposing CLIP Representations for Dense Vision-Language Inference". In: *ECCV*. 2024.

[122] Huang Huang et al. "Otter: A Vision-Language-Action Model with Text-Aware Feature Extraciton". In: *arXiv preprint arXiv:2503.03734* (2025).

[123] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *NeurIPS* (2012).

[124] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CVPR*. 2014.

[125] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[126] Tom B Brown et al. "Language models are few-shot learners". In: *NeurIPS* (2020).

[127] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *ICLR*. 2020.

[128] Kristen Grauman et al. "Ego4d: Around the world in 3,000 hours of egocentric video". In: *arXiv:2110.07058* (2021).

[129] Dima Damen et al. "Rescaling Egocentric Vision: Collection, Pipeline and Challenges for EPIC-KITCHENS-100". In: *IJCV* (2021).

[130] Raghav Goyal et al. "The" something something" video database for learning and evaluating visual common sense". In: *ICCV*. 2017.

[131] Dandan Shan et al. "Understanding human hands in contact at internet scale". In: *CVPR*. 2020.

[132] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *CVPR*. 2009.

[133] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv:1806.10293* (2018).

[134] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *ICRA*. 2016.

[135] Peter Florence, Lucas Manuelli, and Russ Tedrake. "Self-supervised correspondence in visuomotor policy learning". In: *RA-L* (2019).

[136] Michael Danielczuk et al. "Exploratory grasping: Asymptotically optimal algorithms for grasping challenging polyhedral objects". In: *arXiv:2011.05632* (2020).

[137] Dmitry Kalashnkov et al. "MT-OPT: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *arXiv* (2021).

[138] Albert Zhan et al. "Learning Visual Robotic Control Efficiently with Contrastive Pre-training and Data Augmentation". In: *IROS*. 2022.

[139] Zichen Jeff Cui et al. "From play to policy: Conditional behavior generation from uncurated robot data". In: *arXiv:2210.10047* (2022).

[140] Younggyo Seo et al. "Multi-View Masked World Models for Visual Robotic Manipulation". In: *arXiv:2302.02408* (2023).

[141] Frederik Ebert et al. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *arXiv:2109.13396* (2021).

[142] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation". In: *Proceedings of the 6th Conference on Robot Learning (CoRL)*. 2022.

[143] Aviral Kumar et al. "Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials". In: *arXiv:2210.05178* (2022).

[144] Danny Driess et al. "Palm-e: An embodied multimodal language model". In: *arXiv:2303.03378* (2023).

[145] Cheng Chi et al. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *arXiv preprint arXiv:2303.04137* (2023).

[146] Corey Lynch et al. "Interactive language: Talking to robots in real time". In: *IEEE Robotics and Automation Letters* (2023).

[147] Dhruv Shah et al. "ViNT: A Foundation Model for Visual Navigation". In: *7th Annual Conference on Robot Learning (CoRL)*. 2023.

[148] Homanga Bharadhwaj et al. "RoboAgent: Towards Sample Efficient Robot Manipulation with Semantic Augmentations and Action Chunking". In: *arxiv* (2023).

[149] Xi Chen et al. *PaLI-X: On Scaling up a Multilingual Vision and Language Model*. 2023. arXiv: 2305.18565 [cs.CV].

[150] Yutong Bai et al. "Sequential modeling enables scalable learning for large vision models". In: *arXiv preprint arXiv:2312.00785* (2023).

[151] Moo Jin Kim et al. *OpenVLA: An Open-Source Vision-Language-Action Model*. 2024. arXiv: 2406.09246 [cs.RO]. URL: https://arxiv.org/abs/2406.09246.

[152] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Jacquard: A large scale dataset for robotic grasp detection". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3511–3516.

[153] Dmitry Kalashnikov et al. "Scalable deep reinforcement learning for vision-based robotic manipulation". In: *CoRL*. 2018.

[154] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *IJRR* (2018).

[155] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. "ACRONYM: A Large-Scale Grasp Dataset Based on Simulation". In: *2021 IEEE Int. Conf. on Robotics and Automation, ICRA*. 2020.

[156] Nur Muhammad Mahi Shafiullah et al. *On Bringing Robots Home*. 2023. arXiv: 2311.16098 [cs.RO].

[157] Hao-Shu Fang et al. "RH20T: A Robotic Dataset for Learning Diverse Skills in One-Shot". In: *RSS 2023 Workshop on Learning for Task and Motion Planning*. 2023.

[158] Homer Walke et al. *BridgeData V2: A Dataset for Robot Learning at Scale*. 2023. arXiv: 2308.12952 [cs.RO].

[159] Suraj Nair et al. "R3m: A universal visual representation for robot manipulation". In: *arXiv:2203.12601* (2022).

[160] Tete Xiao et al. "Masked visual pre-training for motor control". In: *arXiv:2203.06173* (2022).

[161] Yecheng Jason Ma et al. "Vip: Towards universal visual reward and representation via value-implicit pre-training". In: *arXiv preprint arXiv:2210.00030* (2022).

[162] Ilija Radosavovic et al. "Real-world robot learning with masked visual pre-training". In: *arXiv:2210.03109* (2022).

[163] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *NeurIPS*. Ed. by D. Touretzky. Vol. 1. Morgan-Kaufmann, 1988.

[164] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[165] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *JMLR* (2016).

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

[167] Haotian Liu et al. "Visual Instruction Tuning". In: *NeurIPS*. 2023.

[168] Ilija Radosavovic et al. "Robot Learning with Sensorimotor Pre-training". In: *arXiv:2306.10007* (2023).

[169] Scott Reed et al. "A generalist agent". In: *arXiv preprint arXiv:2205.06175* (2022).

[170] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.

[171] Chelsea Finn et al. "One-shot visual imitation learning via meta-learning". In: *Conference on robot learning*. PMLR. 2017, pp. 357–368.

[172] Mengdi Xu et al. "Hyper-decision transformer for efficient online policy adaptation". In: *arXiv preprint arXiv:2304.08487* (2023).

[173] Zhao Mandi et al. *Towards More Generalizable One-shot Visual Imitation Learning*. 2022. arXiv: 2110.13423 [cs.RO]. URL: https://arxiv.org/abs/2110.13423.

[174] Shizhe Chen et al. "SUGAR: Pre-training 3D Visual Representations for Robotics". In: *arXiv preprint arXiv:2404.01491* (2024).

[175] Letian Fu et al. "Rethinking Patch Dependence for Masked Autoencoders". In: *arXiv preprint arXiv:2401.14391* (2024).

[176] Juho Lee et al. "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 3744–3753.

[177] Jiaming Han et al. *ImageBind-LLM: Multi-modality Instruction Tuning*. 2023. arXiv: 2309.03905 [cs.MM].

[178] Letian Fu et al. "A Touch, Vision, and Language Dataset for Multimodal Alignment". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=tFEOOH9eH0.

[179] Suvir Mirchandani et al. *Large Language Models as General Pattern Machines*. 2023. arXiv: 2307.04721 [cs.AI].

[180] Edward J Hu et al. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[181] Wei-Lin Chiang et al. "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality". In: *See https://vicuna. lmsys. org (accessed 14 April 2023)* 2.3 (2023), p. 6.

[182] Haotian Liu et al. *Improved Baselines with Visual Instruction Tuning*. 2023.

[183] Wenliang Dai et al. "Instructblip: Towards general-purpose vision-language models with instruction tuning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[184] Christoph Schuhmann et al. "Laion-5b: An open large-scale dataset for training next generation image-text models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 25278–25294.

[185] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[186] Yongming Rao et al. "DenseCLIP: Language-Guided Dense Prediction with Context-Aware Prompting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[187] Xiaoyi Dong et al. "Maskclip: Masked self-distillation advances contrastive language-image pretraining". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 10995–11005.

[188] Jean-Baptiste Alayrac et al. "Self-supervised multimodal versatile networks". In: *Advances in neural information processing systems* 33 (2020), pp. 25–37.

[189] Mehdi Cherti et al. "Reproducible scaling laws for contrastive language-image learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2818–2829.

[190] Chao Jia et al. "Scaling up visual and vision-language representation learning with noisy text supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 4904–4916.

[191] Lewei Yao et al. "Filip: Fine-grained interactive language-image pre-training". In: *arXiv preprint arXiv:2111.07783* (2021).

[192] Lu Yuan et al. "Florence: A new foundation model for computer vision". In: *arXiv preprint arXiv:2111.11432* (2021).

[193] Jinze Bai et al. "Qwen-vl: A frontier large vision-language model with versatile abilities". In: *arXiv preprint arXiv:2308.12966* (2023).

[194] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[195] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: (2018).

[196] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* 1.8 (2019), p. 9.

[197] Aakanksha Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[198] Josh Achiam et al. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023).

[199] Hugo Laurençon et al. "What matters when building vision-language models?" In: *arXiv preprint arXiv:2405.02246* (2024).

[200] Siddharth Karamcheti et al. "Prismatic vlms: Investigating the design space of visually-conditioned language models". In: *arXiv preprint arXiv:2402.07865* (2024).

[201] Ethan Perez et al. "Film: Visual reasoning with a general conditioning layer". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[202] Kaiming He et al. "Deep residual learning for image recognition". In: *CVPR*. 2016.

[203] Jimmy Lei Ba. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[204] Bo Liu et al. "Libero: Benchmarking knowledge transfer for lifelong robot learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[205] Kevin Black et al. *π_0: A Vision-Language-Action Flow Model for General Robot Control*. 2024. arXiv: 2410.24164 [cs.LG]. URL: https://arxiv.org/abs/2410.24164.

[206] Lucas Beyer et al. *PaliGemma: A versatile 3B VLM for transfer*. 2024. arXiv: 2407.07726 [cs.CV]. URL: https://arxiv.org/abs/2407.07726.

[207] Karl Pertsch et al. *FAST: Efficient Action Tokenization for Vision-Language-Action Models*. 2025. arXiv: 2501.09747 [cs.RO]. URL: https://arxiv.org/abs/2501.09747.

[208] Yu-Wei Chao et al. *DexYCB: A Benchmark for Capturing Hand Grasping of Objects*. 2021. arXiv: 2104.04631 [cs.CV]. URL: https://arxiv.org/abs/2104.04631.

[209] Yunze Liu et al. *HOI4D: A 4D Egocentric Dataset for Category-Level Human-Object Interaction*. 2024. arXiv: 2203.01577 [cs.CV]. URL: https://arxiv.org/abs/2203.01577.

[210] Common Crawl. *Common Crawl - Open Repository of Web Crawl Data*. https://commoncrawl.org/. Accessed: 2026-05-05. 2026.

[211] Generalist AI Team. "GEN-1: Scaling Embodied Foundation Models to Mastery". In: *Generalist AI Blog* (2026). https://generalistai.com/blog/apr-02-2026-GEN-1.

[212] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[213] Steven M LaValle and James J Kuffner. "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan". In: *Algorithmic and computational robotics* (2001), pp. 303–307.

[214] Yuval Tassa, Tom Erez, and Emanuel Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 4906–4913.

[215] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3406–3413.

[216] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[217] Yevgen Chebotar et al. "Q-Transformer: Scalable Offline Reinforcement Learning via Autoregressive Q-Functions". In: *Conference on Robot Learning (CoRL)*. arXiv:2309.10150. 2023.

[218] Jeffrey Mahler et al. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *arXiv preprint arXiv:1703.09312* (2017).

[219] Jeffrey Mahler et al. "Learning ambidextrous robot grasping policies". In: *Science Robotics* 4.26 (2019), eaau4984.

[220] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. "On-Policy Dataset Synthesis for Learning Robot Grasping Policies Using Fully Convolutional Deep Networks". In: *RAL* (2019).

[221] Konstantinos Bousmalis et al. "RoboCat: A Self-Improving Generalist Agent for Robotic Manipulation". In: *Transactions on Machine Learning Research* (2024). arXiv:2306.11706.

[222] Tony Z. Zhao et al. "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware". In: *Robotics: Science and Systems (RSS)*. 2023. URL: `https://arxiv.org/abs/2304.13705`.

[223] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". In: *Conference on Robot Learning (CoRL)*. arXiv:2401.02117. 2024.

[224] Philipp Wu et al. "GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. arXiv:2309.13037. 2024.

[225] Johan Bjorck et al. "GR00T N1: An Open Foundation Model for Generalist Humanoid Robots". In: *arXiv preprint arXiv:2503.14734* (2025).

[226] Cheng Chi et al. "Universal Manipulation Interface: In-the-Wild Robot Teaching Without In-the-Wild Robots". In: *Robotics: Science and Systems (RSS)*. arXiv:2402.10329. 2024.

[227] Mengda Xu et al. "DexUMI: Using Human Hand as the Universal Manipulation Interface for Dexterous Manipulation". In: *Conference on Robot Learning*. PMLR. 2025, pp. 437–459.

[228] Harsh Gupta et al. *UMI-on-Air: Embodiment-Aware Guidance for Embodiment-Agnostic Visuomotor Policies*. 2026. arXiv: 2510.02614 [cs.RO]. URL: https://arxiv.org/abs/2510.02614.

[229] Hao Li et al. *UMI-Underwater: Learning Underwater Manipulation without Underwater Teleoperation*. 2026. arXiv: 2603.27012 [cs.RO]. URL: https://arxiv.org/abs/2603.27012.

[230] Kristen Grauman et al. "Ego-exo4d: Understanding skilled human activity from first- and third-person perspectives". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 19383–19400.

[231] Javier Romero, Dimitrios Tzionas, and Michael J. Black. "Embodied Hands: Modeling and Capturing Hands and Bodies Together". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 36.6 (2017).

[232] Georgios Pavlakos et al. "Reconstructing Hands in 3D with Transformers". In: *CVPR*. 2024.

[233] Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. "Videodex: Learning dexterity from internet videos". In: *Conference on Robot Learning*. PMLR. 2023, pp. 654–665.

[234] Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. "Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube". In: *arXiv preprint arXiv:2202.10448* (2022).

[235] Himanshu Gaurav Singh et al. "Hand-Object Interaction Pretraining from Videos". In: *arXiv preprint arXiv:2409.08273* (2024).

[236] Yuzhe Qin et al. "AnyTeleop: A General Vision-Based Dexterous Teleoperation System". In: *Robotics: Science and Systems (RSS)*. arXiv:2307.04577. 2023.

[237] Marion Lepert, Jiaying Fang, and Jeannette Bohg. "Phantom: Training Robots Without Robots Using Only Human Videos". In: *arXiv preprint arXiv:2503.00779* (2025).

[238] Ruijie Zheng et al. *EgoScale: Scaling Dexterous Manipulation with Diverse Egocentric Human Data*. 2026.

[239] Google DeepMind. "Veo: Generative Video Model". In: (2024). URL: https://deepmind.google.

[240] Tim Brooks et al. "Video generation models as world simulators. 2024". In: *URL https://openai. com/research/video-generation-models-as-world-simulators* 3 (2024), p. 1.

[241] Team Wan et al. "Wan: Open and advanced large-scale video generative models". In: *arXiv preprint arXiv:2503.20314* (2025).

[242] Philipp Wu et al. "Daydreamer: World models for physical robot learning". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2226–2240.

[243] Anthony Hu et al. *GAIA-1: A Generative World Model for Autonomous Driving*. 2023. arXiv: 2309.17080 [cs.CV].

[244] Mengjiao Yang et al. *Learning Interactive Real-World Simulators*. 2023. arXiv: 2310.06114 [cs.LG].

[245] Hassan Abu Alhaija et al. "Cosmos-Transfer1: Conditional World Generation with Adaptive Multimodal Control". In: *arXiv preprint arXiv:2503.14492* (2025).

[246] Mahmoud Assran et al. *V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning*. 2025. arXiv: 2506.09985 [cs.CV].

[247] Jack Parker-Holder, Jake Bruce, and Genie Team, Google DeepMind. *Genie 2: A Large-Scale Foundation World Model*. Google DeepMind blog post. 2024. URL: https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/.

[248] Shenyuan Gao et al. "DreamDojo: A Generalist Robot World Model from Large-Scale Human Videos". In: *arXiv preprint arXiv:2602.06949* (2026).

[249] Joel Jang et al. "Dreamgen: Unlocking generalization in robot learning through video world models". In: *arXiv preprint arXiv:2505.12705* (2025).

[250] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[251] Fereshteh Sadeghi et al. "Sim2real viewpoint invariant visual servoing by recurrent control". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4691–4699.

[252] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IROS*. 2012.

[253] Jemin Hwangbo et al. "Learning Agile and Dynamic Motor Skills for Legged Robots". In: *Science Robotics* 4.26 (2019). DOI: 10.1126/scirobotics.aau5872.

[254] OpenAI et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv:1910.07113* (2019).

[255] Mayank Mittal et al. "Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments". In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034.

[256] Yuke Zhu et al. "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv:2009.12293* (2020).

[257] Jiayuan Gu et al. "ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills". In: *International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2302.04659.

[258] Bo Liu et al. "Libero: Benchmarking knowledge transfer for lifelong robot learning". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 44776–44791.

[259] Chengshu Li et al. "BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation". In: *arXiv preprint arXiv:2403.09227* (2024).

[260] Haoran Geng et al. *RoboVerse: Towards a Unified Platform, Dataset and Benchmark for Scalable and Generalizable Robot Learning*. 2025. URL: `https://roboverseorg.github.io/`.

[261] Soroush Nasiriany et al. "RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots". In: *Robotics: Science and Systems (RSS)*. arXiv:2406.02523. 2024.

[262] Ajay Mandlekar et al. "MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations". In: *7th Annual Conference on Robot Learning*. 2023.

[263] Zhenyu Jiang et al. "Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning". In: *arXiv preprint arXiv:2410.24185* (2024).

[264] Yufei Wang et al. "Robogen: Towards unleashing infinite data for automated robot learning via generative simulation". In: *arXiv preprint arXiv:2311.01455* (2023).

[265] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. "Gen2sim: Scaling up robot learning in simulation with generative models". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6672–6679.

[266] Weirui Ye et al. "Video2Policy: Scaling up Manipulation Tasks in Simulation through Internet Videos". In: *arXiv preprint arXiv:2502.09886* (2025).

[267] Marcel Torne et al. "Robot Learning with Super-Linear Scaling". In: *arXiv preprint arXiv:2412.01770* (2024).

[268] Marcel Torne et al. "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation". In: *arXiv preprint arXiv:2403.03949* (2024).

[269] Nicholas Pfaff et al. "Scalable Real2Sim: Physics-Aware Asset Generation Via Robotic Pick-and-Place Setups". In: *arXiv preprint arXiv:2503.00370* (2025).

[270] Abhiram Maddukuri et al. "Sim-and-Real Co-Training: A Simple Recipe for Vision-Based Robotic Manipulation". In: *Proceedings of Robotics: Science and Systems (RSS)*. Los Angeles, CA, USA, 2025.

[271] KE McKee. "Automatic Assembly by G. Boothroyd, C. Poli and LE Murch, Marcel Dekker, New York, 378 pp., 1982". In: *Robotica* 3.3 (1985), pp. 195–196.

[272] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. "Automatic synthesis of fine-motion strategies for robots". In: *The International Journal of Robotics Research* 3.1 (1984), pp. 3–24.

[273] Tomas Lozano-Pérez. "Motion planning and the design of orienting devices for vibratory part feeders". In: *in IEEE Journal Of Robotics And Automation. MIT AI Laboratory* (1986).

[274] Kenneth Y Goldberg. "Orienting polygonal parts without sensors". In: *Algorithmica* 10.2 (1993), pp. 201–225.

[275] Balas K Natarajan. "Some paradigms for the automated design of parts feeders". In: *The International journal of robotics research* 8.6 (1989), pp. 98–109.

[276] Hong Qiao, BS Dalay, and RM Parkin. "Fine motion strategies for robotic peg-hole insertion". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 209.6 (1995), pp. 429–448.

[277] LS Homem De Mello and Arthur C Sanderson. "A correct and complete algorithm for the generation of mechanical assembly sequences". In: *1989 IEEE International Conference on Robotics and Automation*. IEEE Computer Society. 1989, pp. 56–57.

[278] Yotto Koga, Heather Kerrick, and Sachin Chitta. "On CAD Informed Adaptive Robotic Assembly". In: *arXiv preprint arXiv:2208.01773* (2022).

[279] Bowen Wen et al. "You Only Demonstrate Once: Category-Level Manipulation from Single Visual Demonstration". In: *Robotics: Science and Systems (RSS)* (2022).

[280] K. Kimble et al. "Benchmarking protocols for evaluating small parts robotic assembly systems". In: 5(2) (2020), pp. 883–889.

[281] Wenzhao Lian et al. "Benchmarking Off-The-Shelf Solutions to Robotic Assembly Tasks". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (2021).

[282] Jianlan Luo et al. "Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study". In: *Robotics: Science and Systems (RSS)* (2021).

[283] Ryo Okumura, Nobuki Nishio, and Tadahiro Taniguchi. "Tactile-Sensitive Newtonian-VAE for High-Accuracy Industrial Connector-Socket Insertion". In: *arXiv preprint arXiv:2203.05955* (2022).

[284] Yashraj Narang et al. "Factory: Fast Contact for Robotic Assembly". In: *Robotics: Science and Systems (RSS)* (2022).

[285] Tarik Kelestemur, Robert Platt, and Taskin Padir. "Tactile Pose Estimation and Policy Learning for Unknown Object Manipulation". In: *Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (2022).

[286] Shaoxiong Wang et al. "TACTO: A Fast, Flexible, and Open-source Simulator for High-resolution Vision-based Tactile Sensors". In: *IEEE Robotics and Automation Letters (RA-L)* 7.2 (2022), pp. 3930–3937. ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3146945. URL: https://arxiv.org/abs/2012.08456.

[287] Yikai Wang et al. "Elastic tactile simulation towards tactile-visual perception". In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 2690–2698.

[288] Tobias Johannink et al. "Residual reinforcement learning for robot control". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6023–6029.

[289] Gerrit Schoettler et al. "Meta-reinforcement learning for robotic industrial insertion tasks". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 9728–9735.

[290] Tony Z Zhao et al. "Offline meta-reinforcement learning for industrial insertion". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6386–6393.

[291] Gerrit Schoettler et al. "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5548–5555.

[292] Oren Spector and Dotan Di Castro. "InsertionNet - A Scalable Solution for Insertion". In: *IEEE Robotics and Automation Letters* (2021).

[293] Oren Spector, Vladimir Tchuiev, and Dotan Di Castro. "InsertionNet 2.0: Minimal Contact Multi-Step Insertion Using Multimodal Multiview Sensory Input". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)* (2022).

[294] Wenzhen Yuan, Siyuan Dong, and Edward H Adelson. "Gelsight: High-resolution robot tactile sensors for estimating geometry and force". In: *Sensors* 17.12 (2017), p. 2762.

[295] Mike Lambeta et al. "Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation". In: *IEEE Robotics and Automation Letters* (2020).

[296] Rui Li et al. "Localization and manipulation of small parts using GelSight tactile sensing". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (2014).

[297] Pete Florence et al. "Implicit Behavioral Cloning". In: *Conf. on Robot Learning (CoRL)* (2021).

[298] Yongxiang Fan, Jieliang Luo, and Masayoshi Tomizuka. "A Learning Framework for High Precision Industrial Assembly". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 811–817.

[299] Arkadeep Narayan Chaudhury et al. "Using Collocated Vision and Tactile Sensors for Visual Servoing and Localization". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3427–3434.

[300] Hideyuki Ichiwara et al. "Contact-rich manipulation of a flexible object based on deep predictive learning using vision and tactility". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 5375–5381.

[301] Johanna Hansen et al. "Visuotactile-RL: Learning Multimodal Manipulation Policies with Deep Reinforcement Learning". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 8298–8304.

[302] Ilija Radosavovic et al. "Designing network design spaces". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10428–10436.

[303] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: 2015.

[304] Lars Berscheid and Torsten Kröger. "Jerk-limited Real-time Trajectory Generation with Arbitrary Target States". In: *Robotics: Science and Systems XVII* (2021).

[305] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[306] Philipp Moritz et al. "Ray: A distributed framework for emerging AI applications". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 561–577.

[307] Michael Posa, Scott Kuindersma, and Russ Tedrake. "Optimization and stabilization of trajectories for constrained dynamical systems". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1366–1373.

[308] Carlos Mastalli et al. "Crocoddyl: An efficient and versatile framework for multi-contact optimal control". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2536–2542.

[309] Ilija Radosavovic et al. "Real-world humanoid locomotion with reinforcement learning". In: *Science Robotics* 9.89 (2024).

[310] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

[311] OpenAI. "GPT-4 Technical Report". In: *CoRR* abs/2303.08774 (2023). URL: `https://doi.org/10.48550/arXiv.2303.08774`.

[312] Jinze Bai et al. "Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond". In: *arXiv preprint arXiv:2308.12966* (2023). URL: `https://doi.org/10.48550/arXiv.2308.12966`.

[313] Yanjie Ze et al. "3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2024.

[314] TRI LBM Team et al. "A Careful Examination of Large Behavior Models for Multitask Dexterous Manipulation". In: (2025). arXiv: 2507.05331 [cs.RO]. URL: https://arxiv.org/abs/2507.05331.

[315] Suvir Mirchandani et al. "So You Think You Can Scale Up Autonomous Robot Data Collection?" In: *Proceedings of the 8th Conference on Robot Learning (CoRL), November 2024*. 2024.

[316] Ken Goldberg. *Good old-fashioned engineering can close the 100,000-year "data gap" in robotics*. 2025.

[317] Minchen Li et al. "Incremental potential contact: intersection-and inversion-free, large-deformation dynamics." In: *ACM Trans. Graph.* 39.4 (2020), p. 49.

[318] Chung Min Kim et al. "IPC-GraspSim: Reducing the Sim2Real gap for parallel-jaw grasping with the incremental potential contact model". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6180–6187.

[319] Chaitanya Mitash et al. "ARMBench: An object-centric benchmark dataset for robotic manipulation". In: (2023). URL: https://www.amazon.science/publications/armbench-an-object-centric-benchmark-dataset-for-robotic-manipulation.

[320] Andrew Sohn et al. *Introducing RFM-1: Giving robots human-like reasoning capabilities*. Accessed: 2025-04-29. Mar. 2024. URL: https://covariant.ai/insights/introducing-rfm-1-giving-robots-human-like-reasoning-capabilities/.

[321] Vishal Satish, Jeff Mahler, and Ken Goldberg. *PRIME-1: Scaling Large Robot Data for Industrial Reliability*. Accessed: 2025-04-29. Jan. 2025. URL: https://www.ambirobotics.com/blog/prime-1-scaling-large-robot-data-for-industrial-reliability/.

[322] Zachary Teed and Jia Deng. "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras". In: *Advances in neural information processing systems* (2021).

[323] Anthony Brohan et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *arXiv preprint arXiv:2307.15818*. 2023.

[324] Justin Kerr et al. "Self-supervised visuo-tactile pretraining to locate and follow garment features". In: *arXiv preprint arXiv:2209.13042* (2022).

[325] Albert Wilcox et al. "Learning to localize, grasp, and hand over unmodified surgical needles". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 9637–9643.

[326] Lawrence Yunliang Chen et al. "Efficiently learning single-arm fling motions to smooth garments". In: *The International Symposium of Robotics Research*. Springer. 2022, pp. 36–51.

[327] Letian Fu et al. "Safe self-supervised learning in real of visuo-tactile feedback policies for industrial insertion". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10380–10386.

[328] Xinhai Li et al. "Robogsim: A real2sim2real robotic gaussian splatting simulator". In: *arXiv preprint arXiv:2411.11839* (2024).

[329] Lawrence Yunliang Chen et al. "Rovi-aug: Robot and viewpoint augmentation for cross-embodiment robot learning". In: *arXiv preprint arXiv:2409.03403* (2024).

[330] Zhengrong Xue et al. "DemoGen: Synthetic Demonstration Generation for Data-Efficient Visuomotor Policy Learning". In: *arXiv preprint arXiv:2502.16932* (2025).

[331] Jiafei Duan et al. "AR2-D2: Training a Robot Without a Robot". In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=JdpleC92J4.

[332] Vincent Lim et al. *Planar Robot Casting with Real2Sim2Real Self-Supervised Learning*. 2022. arXiv: 2111.04814 [cs.RO]. URL: https://arxiv.org/abs/2111.04814.

[333] Tianyuan Dai et al. "Automated creation of digital cousins for robust policy learning". In: *arXiv preprint arXiv:2410.07408* (2024).

[334] Justin Kerr et al. "Robot See Robot Do: Imitating Articulated Object Manipulation with Monocular 4D Reconstruction". In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=2LLu3gavF1.

[335] Justin Yu et al. "Persistent Object Gaussian Splat (POGS) for Tracking Human and Robot Manipulation of Irregularly Shaped Objects". In: *ICRA* (2025).

[336] Bernhard Kerbl et al. "3d gaussian splatting for real-time radiance field rendering." In: *ACM Trans. Graph.* 42.4 (2023), pp. 139–1.

[337] Chung Min* Kim et al. "GARField: Group Anything with Radiance Fields". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024.

[338] Antoine Guédon and Vincent Lepetit. "SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering". In: *CVPR* (2024).

[339] Chung Min Kim* et al. *PyRoki: A Modular Toolkit for Robot Kinematic Optimization*. 2025. arXiv: 2505.03728 [cs.RO]. URL: https://arxiv.org/abs/2505.03728.

[340] Karl Pertsch et al. "Fast: Efficient action tokenization for vision-language-action models". In: *arXiv preprint arXiv:2501.09747* (2025).

[341] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *CoRR* abs/2106.09685 (2021). arXiv: 2106.09685. URL: https://arxiv.org/abs/2106.09685.

[342] Nicholas Pfaff et al. "Scalable Real2Sim: Physics-Aware Asset Generation Via Robotic Pick-and-Place Setups". In: (2025). arXiv: 2503.00370 [cs.RO]. URL: https://arxiv.org/abs/2503.00370.

[343] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 1470–1477.

[344] Xueyang Zhou et al. *LIBERO-PRO: Towards Robust and Fair Evaluation of Vision-Language-Action Models Beyond Memorization*. 2025. arXiv: 2510.03827 [cs.CV]. URL: https://arxiv.org/abs/2510.03827.

[345] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208.

[346] Richard M Murray et al. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[347] Constructions Aeronautiques et al. "Pddl—the planning domain definition language". In: *Technical Report, Tech. Rep.* (1998).

[348] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*. Vol. 200. Springer, 2008.

[349] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[350] Stefanie Tellex et al. "Understanding natural language commands for robotic navigation and mobile manipulation". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. 1. 2011, pp. 1507–1514.

[351] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[352] Tom Christie, Marcelo Trylesinski, and contributors. *Uvicorn: ASGI Web Server for Python*. https://pypi.org/project/uvicorn/. Version 0.40.0, BSD-3-Clause License. 2025. URL: https://uvicorn.dev.

[353] Nicolas Carion et al. *SAM 3: Segment Anything with Concepts*. 2025. arXiv: 2511.16719 [cs.CV]. URL: https://arxiv.org/abs/2511.16719.

[354] Christopher Clark et al. *Molmo2: Open Weights and Data for Vision-Language Models with Video Understanding and Grounding*. 2026. arXiv: 2601.10611 [cs.CV]. URL: https://arxiv.org/abs/2601.10611.

[355] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[356] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A modern library for 3D data processing". In: *arXiv preprint arXiv:1801.09847* (2018).

[357] Chung Min Kim et al. "PyRoki: A Modular Toolkit for Robot Kinematic Optimization". In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2025. URL: https://arxiv.org/abs/2505.03728.

[358] Google DeepMind. *Gemini 3 Pro Model Card*. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf. Model card for the Gemini 3 Pro multimodal AI model, published November 2025. Nov. 2025.

[359] OpenAI. *OpenAI o3 and o4-mini System Card*. https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf. System card for the o3 and o4-mini reasoning models (dated April 16, 2025). Apr. 2025.

[360] OpenAI. *GPT-5.1*. https://openai.com/index/gpt-5-1/. GPT-5.1: A smarter, more conversational ChatGPT. Nov. 2025.

[361] OpenAI. *Introducing GPT-5.2*. https://openai.com/index/introducing-gpt-5-2/. Official introduction of the GPT-5.2 model series as OpenAI's most advanced large language model. Dec. 2025.

[362] Anthropic. *Introducing Claude Haiku 4.5*. https://www.anthropic.com/news/claude-haiku-4-5. Announcement of Claude Haiku 4.5, Anthropic's newest small, efficient model. Oct. 2025.

[363] Anthropic. *Introducing Claude Opus 4.5*. https://www.anthropic.com/news/claude-opus-4-5. Official announcement of Claude Opus 4.5, Anthropic's flagship model with advanced coding and agentic capabilities. Nov. 2025.

[364] OpenAI. *gpt-oss-120b & gpt-oss-20b Model Card*. https://cdn.openai.com/pdf/419b6906-9da6-406c-a19d-1bb078ac7637/oai_gpt-oss_model_card.pdf. Model card for the GPT-OSS open-weight reasoning models including the 120B model. 2025.

[365] QwenLM / Alibaba Cloud. *Qwen3-235B-A22B*. https://huggingface.co/Qwen/Qwen3-235B-A22B. Official model card for Qwen3-235B-A22B, a 235 billion parameter mixture-of-experts (MoE) large language model in the Qwen3 family. Apr. 2025.

[366] Moonshot AI. *Kimi-K2-Instruct (Revision 2f7e011)*. 2025. DOI: 10.57967/hf/5976. URL: https://huggingface.co/moonshotai/Kimi-K2-Instruct.

[367] Lanxiang Hu et al. "lmgame-Bench: How Good are LLMs at Playing Games?" In: *arXiv preprint arXiv:2505.15146* (2025).

[368] Zirui Wang et al. "VisGym: Diverse, Customizable, Scalable Environments for Multimodal Agents". In: *arXiv preprint arXiv:2601.16973* (2026). URL: https://arxiv.org/abs/2601.16973.

[369] Jiayi Pan et al. "Learning Adaptive Parallel Reasoning with Language Models". In: *arXiv preprint arXiv: 2504.15466* (2025).

[370] Tian Jin et al. "Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding". In: *arXiv preprint arXiv:2502.11517* (2025).

[371] Gleb Rodionov et al. "Hogwild! inference: Parallel llm generation via concurrent attention". In: *arXiv preprint arXiv:2504.06261* (2025).

[372] Binyuan Hui et al. "Qwen2. 5-coder technical report". In: *arXiv preprint arXiv:2409.12186* (2024).

[373] Liang Chen et al. "G1: Bootstrapping Perception and Reasoning Abilities of Vision-Language Model via Reinforcement Learning". In: *arXiv preprint arXiv:2505.13426* (2025).

[374] Tanmay Gupta and Aniruddha Kembhavi. "Visual Programming: Compositional Visual Reasoning Without Training". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 14953–14962. URL: https://arxiv.org/abs/2211.11559.

[375] Dídac Surís, Sachit Menon, and Carl Vondrick. "ViperGPT: Visual Inference via Python Execution for Reasoning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 11888–11898. URL: https://arxiv.org/abs/2303.08128.

[376] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 24824–24837. URL: https://arxiv.org/abs/2201.11903.

[377] Shaofeng Yin et al. *Vision-as-Inverse-Graphics Agent via Interleaved Multimodal Reasoning*. 2026. arXiv: 2601.11109 [cs.CV]. URL: https://arxiv.org/abs/2601.11109.

[378] Sai Vemprala et al. *ChatGPT for Robotics: Design Principles and Model Abilities*. Tech. rep. MSR-TR-2023-8. Microsoft Research, 2023. URL: https://arxiv.org/abs/2306.17582.

[379] Siyuan Huang et al. "Instruct2Act: Mapping Multi-modality Instructions to Robotic Actions with Large Language Model". In: *arXiv preprint arXiv:2305.11176* (2023). URL: https://arxiv.org/abs/2305.11176.

[380] Yao Mu et al. "RoboCodeX: Multimodal Code Generation for Robotic Behavior Synthesis". In: *arXiv preprint arXiv:2402.16117* (2024). URL: https://arxiv.org/abs/2402.16117.

[381] Ruiying Li et al. *RoboClaw: An Agentic Framework for Scalable Long-Horizon Robotic Tasks*. 2026. arXiv: 2603.11558 [cs.RO]. URL: https://arxiv.org/abs/2603.11558.

[382] Andrew Goldberg et al. "Blox-Net: Generative Design-for-Robot-Assembly Using VLM Supervision, Physics Simulation, and a Robot with Reset". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2025. URL: https://arxiv.org/abs/2409.17126.

[383] Yao Mu et al. "EmbodiedGPT: Vision-Language Pre-Training via Embodied Chain of Thought". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023. URL: https://arxiv.org/abs/2305.15021.

[384] Andy Zeng et al. "Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language". In: *International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2204.00598.

[385] Bo Liu et al. "LLM+P: Empowering Large Language Models with Optimal Planning Proficiency". In: *arXiv preprint arXiv:2304.11477* (2023). URL: https://arxiv.org/abs/2304.11477.

[386] Yongchao Chen et al. "AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024. URL: https://arxiv.org/abs/2306.06531.

[387] Takuma Yoneda et al. "Statler: State-Maintaining Language Models for Embodied Reasoning and Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 15083–15091. URL: https://arxiv.org/abs/2306.17840.

[388] Zhao Mandi, Shreeya Jain, and Shuran Song. "RoCo: Dialectic Multi-Robot Collaboration with Large Language Models". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024. URL: https://arxiv.org/abs/2307.04738.

[389] Krishan Rana et al. "SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning". In: *Conference on Robot Learning (CoRL)*. 2023. URL: https://arxiv.org/abs/2307.06135.

[390] Gemini Robotics Team et al. "Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer". In: *arXiv preprint arXiv:2510.03342* (2025).

[391] Aman Madaan et al. "Self-Refine: Iterative Refinement with Self-Feedback". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023. URL: https://arxiv.org/abs/2303.17651.

[392] Xinyun Chen et al. "Teaching Large Language Models to Self-Debug". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: https://arxiv.org/abs/2304.05128.

[393] Zihao Wang et al. "Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023. URL: https://arxiv.org/abs/2302.01560.

[394] Yujia Li et al. "Competition-Level Code Generation with AlphaCode". In: *Science* 378.6624 (2022), pp. 1092–1097. URL: https://arxiv.org/abs/2203.07814.

[395] Yecheng Jason Ma et al. "Eureka: Human-Level Reward Design via Coding Large Language Models". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: https://arxiv.org/abs/2310.12931.

[396] Wenhao Yu et al. "Language to Rewards for Robotic Skill Synthesis". In: *Conference on Robot Learning (CoRL)*. 2023. URL: https://arxiv.org/abs/2306.08647.

[397] Tianbao Xie et al. "Text2Reward: Reward Shaping with Language Models for Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: https://arxiv.org/abs/2309.11489.

[398] Minae Kwon et al. "Reward Design with Language Models". In: *arXiv preprint arXiv:2303.00001* (2023).

[399] William Liang et al. "Eurekaverse: Environment Curriculum Generation via Large Language Models". In: *Conference on Robot Learning (CoRL)*. 2024. URL: https://arxiv.org/abs/2411.01775.

[400] Kanghyun Ryu et al. "CurricuLLM: Automatic Task Curricula Design for Learning Complex Robot Skills using Large Language Models". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2025. URL: https://arxiv.org/abs/2409.18382.

[401] Yecheng Jason Ma et al. "DrEureka: Language Model Guided Sim-to-Real Transfer". In: *Robotics: Science and Systems (RSS)*. 2024. URL: https://arxiv.org/abs/2406.01967.

[402] Yuqing Du et al. "Guiding Pretraining in Reinforcement Learning with Large Language Models". In: *International Conference on Machine Learning (ICML)*. 2023. URL: https://arxiv.org/abs/2302.06692.

[403] Yufei Wang et al. "RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation". In: *International Conference on Machine Learning (ICML)*. 2024. URL: https://arxiv.org/abs/2311.01455.

[404] Michael Ahn et al. "AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents". In: *arXiv preprint arXiv:2401.12963* (2024). URL: https://arxiv.org/abs/2401.12963.

[405] Hung Le et al. "CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning". In: 35 (2022), pp. 21314–21328. URL: https://arxiv.org/abs/2207.01780.

[406] Parshin Shojaee et al. "Execution-based Code Generation using Deep Reinforcement Learning". In: *Transactions on Machine Learning Research* (2023). URL: https://arxiv.org/abs/2301.13816.

[407] Yuxiang Wei et al. "SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution". In: *arXiv preprint arXiv:2502.18449* (2025). URL: https://arxiv.org/abs/2502.18449.

[408] Zehan Qi et al. "WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning". In: *arXiv preprint arXiv:2411.02337* (2024). URL: https://arxiv.org/abs/2411.02337.

[409] Kimi Team et al. *Kimi K2.5: Visual Agentic Intelligence*. 2026. arXiv: 2602.02276 [cs.CL]. URL: https://arxiv.org/abs/2602.02276.

[410] Jiazhan Feng et al. "ReTool: Reinforcement Learning for Strategic Tool Use in LLMs". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=tRk1nofSmz.

[411] Cheng Qian et al. *ToolRL: Reward is All Tool Learning Needs*. 2025. arXiv: 2504.13958 [cs.LG]. URL: https://arxiv.org/abs/2504.13958.

[412] Siyi Chen et al. *SpaceTools: Tool-Augmented Spatial Reasoning via Double Interactive RL*. 2025. arXiv: 2512.04069 [cs.CV]. URL: https://arxiv.org/abs/2512.04069.

[413] Yi Han et al. *TIGeR: Tool-Integrated Geometric Reasoning in Vision-Language Models for Robotics*. 2026. arXiv: 2510.07181 [cs.RO]. URL: https://arxiv.org/abs/2510.07181.

[414] Oier Mees et al. "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7327–7334.

[415] Stephen James et al. "Rlbench: The robot learning benchmark & learning environment. arxiv e-prints, art". In: *arXiv preprint arXiv:1909.12271* (2019).

[416] Xuanlin Li et al. "Evaluating Real-World Robot Manipulation Policies in Simulation". In: *arXiv preprint arXiv:2405.05941* (2024).

[417] Tianhe Yu et al. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2020. URL: https://arxiv.org/abs/1910.10897.

[418] Rui Yang et al. "Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents". In: *arXiv preprint arXiv:2502.09560* (2025).

[419] Xiao Liu et al. "AgentBench: Evaluating LLMs as Agents". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: https://arxiv.org/abs/2308.03688.

[420] Mohit Shridhar et al. "ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. URL: https://arxiv.org/abs/1912.01734.

[421] Omar Khattab et al. "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines". In: 2024.

[422] Physical Intelligence. *$\pi_{0.7}$: A Vision–Language–Action Model with Open-World Generalization*. Technical Report, Physical Intelligence. 2025. URL: https://www.pi.website/download/pi07.pdf.

[423] Fangchen Liu et al. "Masked autoencoding for scalable and generalizable decision making". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 12608–12618.

[424] Meta AI. *SAM 3D: 3Dfy Anything in Images*. 2025. arXiv: 2511.16624 [cs.CV].

[425] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

[426] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[427] DeepSeek-AI. "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning". In: *arXiv preprint arXiv:2501.12948* (2025).

[428] OpenAI. *GPT-4o System Card*. `https://cdn.openai.com/gpt-4o-system-card.pdf`. Accessed: 2024-09-14. 2024.

[429] Javier Romero, Dimitrios Tzionas, and Michael J. Black. "Embodied Hands: Modeling and Capturing Hands and Bodies Together". In: *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*. 245:1–245:17 36.6 (Nov. 2017).

[430] Adam Wei et al. "Empirical Analysis of Sim-and-Real Cotraining Of Diffusion Policies For Planar Pushing from Pixels". In: *arXiv preprint arXiv:2503.22634* (2025).

[431] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[432] Ilya Loshchilov and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts". In: (2017).

[433] Priya Goyal et al. "Accurate, large minibatch sgd: Training imagenet in 1 hour". In: *arXiv:1706.02677* (2017).

[434] Huang Huang et al. "Mechanical search on shelves with efficient stacking and destacking of objects". In: *The International Symposium of Robotics Research*. Springer. 2022, pp. 205–221.